Contribution à l'étude de systèmes hiérarchisés de ressources dans l'architecture des machines informatiques

François Anceau

THESE

présentée à

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

POUR OBTENIR LE GRADE DE

DOCTEUR ES SCIENCES

François ANCEAU

Contribution à l'étude des systèmes hiérarchisés de ressources dans l'architecture des machines informatiques.

Thèse soutenue le 5 décembre 1974 devant la commission d'examen

Président : Monsieur le Professeur J. KUNTZMANN

Jury : Monsieur le Professeur J. ARSAC
Monsieur le Professeur L. BOLLIET
Monsieur S. KRAKOWIAK
Madame G. SAUCIER

UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

M. Michel SOUTIF	Présidents	M. Louis NEEL
M. Gabriel CAU	Vice-Présidents	MM. Lucien BONNETAIN, Jean BENOIT

MEMBRES DU CORPS ENSEIGNANT DE L'U.S.M.G.

PROFESSEURS TITULAIRES

MM.	ANGLES D'AURIAC Paul	Mécanique des Fluides
	ARNAUD Georges	Clinique des maladies infectieuses
	ARNAUD Paul	Chimie
	AUBERT Guy	Physique
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale
	BEAUDOING André	Pédiatrie
	BERNARD Alain	Mathématiques pures
Mme	BERTRANDIAS	Mathématiques pures
MM.	BEZES Henri	Chirurgie générale
	BLAMBERT Maurice	Mathématiques Pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Georges	Electrotechnique
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Pathologie médicale
	BOUCHERLE André	Chimie et Toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BOUSSARD Jean-Claude	Mathématiques appliquées
	BRAVARD Yves	Géographie
	CABANEL Guy	Clinique rhumatologique et hydrologie
	CALAS François	Anatomie
	CARRAZ Gilbert	Biologie animale et pharmacodynamie
	CAU Gabriel	Médecine légale et Toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques pures
	CHARACHON Robert	Oto-Rhino-Laryngologie
	CHATEAU Robert	Thérapeutique
	CHIBON Pierre	Biologie animale
	COEUR André	Pharmacie chimique et chimie analytique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie pathologique
	CRAYA Antoine	Mécanique
Mme	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DEPORTES Charles	Chimie minérale
	DESRE Pierre	Métallurgie

MM.	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée
	DOLIQUE Jean-Michel	Physique des plasmas
	DREYFUS Bernard	Thermodynamique
	DUCROS Pierre	Cristallographie
	DUGOIS Pierre	Clinique de dermatologie et syphiligraphie
	FAU René	Clinique neuro-psychiatrique
	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques pures
	GALVANI Octave	Mathématiques pures
	GASTINEL Noël	Analyse Numérique
	GAVEND Michel	Pharmacologie
	GEINDRE Michel	Electroradiologie
	GERBER Robert	Mathématiques pures
	GERMAIN Jean-Pierre	Mécanique
	GIRAUD Pierre	Géologie
	KAHANE André	Physique générale
	KLEIN Joseph	Mathématiques pures
	KOSZUL Jean-Louis	Mathématiques pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
	LAJZEROWICZ Joseph	Physique
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biologie pharmaceutique
	LAURENT Pierre-Jean	Mathématiques appliquées
	LEDRU Jean	Clinique médicale B
	LLIBOUTRY Louis	Géophysique
	LONGEQUEUE Jean-Pierre	Physique nucléaire
	LOUP Jean	Géographie
Mle	LUTZ Elisabeth	Mathématiques pures
MM.	MALGRANGE Bernard	Mathématiques pures
	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Seméiologie médicale
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et Pétrographie
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	NEEL Louis	Physique du solide
	OZENDA Paul	Botanique
	PAYAN Jean-Jacques	Mathématiques pures
	PEBAY-PEYROULA Jean-Claude	Physique
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	REULOS René	Physique industrielle
	RINALDI Renaud	Physique
	ROGET Jean	Clinique de pédiatrie et de puériculture
	DE ROUGEMONT Jacques	Neuro-chirurgie
	SEIGNEURIN Raymond	Microbiologie et hygiène
	SENGEL Philippe	Zoologie
	SOUTIF Michel	Physique générale
	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLANT François	Zoologie
	VALENTIN Jacques	Physique nucléaire
	VAUQUOIS Bernard	Calcul électronique
Mme	VERAIN Alice	Pharmacie galénique
MM.	VERAIN André	Physique
	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale
	YOCCOZ Jean	Physique nucléaire théorique

PROFESSEURS ASSOCIES

MM.	ASCARELLI Gianni	Physique
	CHEEKE John	Thermodynamique
	GILLESPIE John	I.S.N.
	ROCKAFELLAR Ralph	Mathématiques appliquées
	WOHLFARTH Erich	Physique du solide

PROFESSEURS SANS CHAIRE

Mle	AGNIUS-DELORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	BELORIZKY Elie	Physique
	BENZAKEN Claude	Mathématiques appliquées
	BERTRANDIAS Jean-Paul	Mathématiques appliquées
	BIAREZ Jean-Pierre	Mécanique
Mme	BONNIER Jane	Chimie générale
MM.	BRUGEL Lucien	Energétique
	CARLIER Georges	Biologie végétale
	CONTE René	Physique
	DEPASSEL Roger	Mécanique des Fluides
	GAUTHIER Yves	Sciences biologiques
	GAUTRON René	Chimie
	GIDON Paul	Géologie et Minéralogie
	GLENAT René	Chimie organique
	HACQUES Gérard	Calcul numérique
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Hygiène et Médecine préventive
	IDELMAN Simon	Physiologie animale
	JANIN Bernard	Géographie
	JOLY Jean-René	Mathématiques pures
	JULLIEN Pierre	Mathématiques appliquées
Mme	KAHANE Josette	Physique
MM.	KUHN Gérard	Physique
	LUU-DUC-Cuong	Chimie organique
	MAYNARD Roger	Physique du solide
	MULLER Jean-Michel	Thérapeutique
	PERRIAUX Jean-Jacques	Géologie et minéralogie
	PFISTER Jean-Claude	Physique du solide
Mle	PIERY Yvette	Physiologie animale
MM.	REBECQ Jacques	Biologie (CUS)
	REVOL Michel	Urologie
	REYMOND Jean-Charles	Chirurgie générale
	ROBERT André	Chimie papetière
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIBILLE Robert	Construction mécanique
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale
MM.	VIALON Pierre	Géologie
	VAN CUTSEM Bernard	Mathématiques appliquées

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

MM.	AMBLARD Pierre	Dermatologie
	AMBROISE-THOMAS Pierre	Parasitologie
	ARMAND Yves	Chimie
	BEGUIN Claude	Chimie organique
Mme	BERIEL Hélène	Pharmacodynamique
MM.	BILLET Jean	Géographie
	BOUCHARLAT Jacques	Psychiatrie adultes
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BOUCHET Yves	Anatomie
	BRODEAU François	Mathématiques (IUT B)
	BUISSON Roger	Physique
	BUTEL Jean	Orthopédie
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHERADAME Hervé	Chimie papetière
	CHIAVERINA Jean	Biologie appliquée (EFP)
	COHEN-ADDAD Jean-Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie médicale
	COULOMB Max	Radiologie
	CROUZET Guy	Radiologie
	CYROT Michel	Physique du solide
	DELOBEL Claude	M.I.A.G.
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	FONTAINE Jean-Marc	Mathématiques pures
	GENSAC Pierre	Botanique
	GIDON Maurice	Géologie
	GRIFFITHS Michaël	Mathématiques Appliquées
	GROS Yves	Physique (stag.)
	GROULADE Joseph	Biochimie médicale
	GUITTON Jacques	Chimie
	IVANES Marcel	Electricité
	JALBERT Pierre	Histologie
	KRAKOWIAK Sacha	Mathématiques appliquées
Mme	LAJZEROWICZ Jeannine	Physique
MM.	LEROY Philippe	Mathématiques
	LOISEAUX Jean-Marie	Physique nucléaire
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et médecine préventive
	MARECHAL Jean	Mécanique
	MARTIN-BOUYER Michel	Chimie (CUS)
	MICHOULIER Jean	Physique (I.U.T. "A")
Mme	MINIER Colette	Physique
MM.	MICOUD Max	Maladies infectieuses
	NEGRE Robert	Mécanique
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (I.U.T. "B")
	PEFFEN René	Métallurgie
	PELMONT Jean	Physiologie animale
	PERRET Jean	Neurologie
	PHELIP Xavier	Rhumatologie
	RACHAIL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RAYNAUD Hervé	M.I.A.G.
	RENAUD Maurice	Chimie
Mme	RENAUDET Jacqueline	Bactériologie
M.	RICHARD Lucien	Botanique
Mme	RINAUDO Marguerite	Chimie macromoléculaire
M.	ROMIER Guy	Mathématiques (I.U.T. "B")

MM.	SHOM Jean-Claude	Chimie Générale
	STIEGLITZ Paul	Anesthésiologie
	STOEBNER Pierre	Anatomie pathologique
	VROUSOS Constantin	Radiologie

MAITRES DE CONFERENCES ASSOCIES

MM.	CRABBEE Pierre	C.E.R.M.O.
	CABOT	Mathématiques appliquées
	CURRIE Jan	Mathématiques appliquées

CHARGES DE FONCTIONS DE MAITRES DE CONFERENCES

MM.	BARGE Michel	Neuro-chirurgie
	CONTAMIN Charles	Chirurgie thoracique et cardio-vasculaire
	CORDONNIER Daniel	Néphrologie
	DENIS Bernard	Cardiologie
	KOLODIE Lucien	Hématologie
	RAMBAUD Pierre	Pédiatrie
	ROCHAT Jacques	Hygiène et hydrologie

MEMBRES DU CORPS ENSEIGNANT DE L'I.N.P.G.

PROFESSEURS TITULAIRES

MM.	BENOIT Jean	Radioélectricité
	BESSON Jean	Electrochimie
	BONNETAIN Lucien	Chimie minérale
	BONNIER Etienne	Electrochimie, Electrométallurgie
	BRISSONNEAU Pierre	Physique du solide
	BUYLE-BODIN Maurice	Electronique
	COUMES André	Radioélectricité
	FELICI Noël	Electrostatique
	PAUTHENET René	Physique du solide
	PERRET René	Servomécanismes
	SANTON Lucien	Mécanique
	SILBERT Robert	Mécanique des Fluides

PROFESSEUR ASSOCIE

M.	BOUDOURIS Georges	Radioélectricité

PROFESSEURS SANS CHAIRE

MM.	BLIMAN Samuel	Electronique
	BLOCH Daniel	Physique du solide et cristallographie
	COHEN Joseph	Electrotechnique
	DURAND Francis	Métallurgie
	MOREAU René	Mécanique
	POLOUJADOFF Michel	Electrotechnique
	VEILLON Gérard	Informatique fondamentale et appliquée
	ZADWORNY François	Electronique

MAITRES DE CONFERENCES

MM.	BOUVARD Maurice	Génie mécanique
	CHARTIER Germain	Electronique
	FOULARD Claude	Automatique
	GUYOT Pierre	Chimie minérale
	JOUBERT Jean-Claude	Physique du solide
	LACOUME Jean-Louis	Géophysique
	LANCIA Roland	Physique atomique
	LESPINARD Georges	Mécanique
	MORET Roger	Electrotechnique nucléaire
	ROBERT François	Analyse Numérique
	SABONNADIERE Jean-Claude	Informatique fondamentale et appliquée
Mme	SAUCIER Gabrielle	Informatique fondamentale et appliquée

MAITRE DE CONFERENCES ASSOCIE

M.	LANDAU Ioan Doré	Automatique

CHARGE DE FONCTIONS DE MAITRES DE CONFERENCES

M.	ANCEAU François	Mathématiques appliquées

Fait à St Martin d'Hères JANVIER 1974

ERRATA

page 92

3° ligne: ... des opérations S, lorsque l'être b est toujours candidat.

4° ligne: $proj_A(AS) \neq 0$

pages 92 (12° ligne) et page 93 (1° ligne)

opération $S(\Delta,b)$ exécutée à l'origine des temps"

page 146

commentaire de la figure:

lire: articulations de déroutement et non articulations d'interruption.

page 196

commentaire de la figure:

programme de sélection des périphériques.

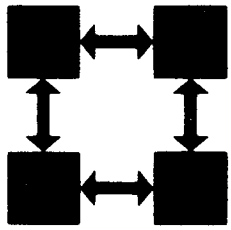
page 239

4° ligne:

.... processeurs virtuels candidats à l'exécution et toujours candidate à cette association.

pages 261, 263, 265, 268.

solution publiée [22].



Je désire, par ces quelques mots, exprimer ma reconnaissance aux personnes qui ont contribué à l'aboutissement de ce travail:

- Monsieur le Professeur KUNTZMANN, Directeur de l'ENSIMAG, pour avoir couvert ces recherches de son autorité bienveillante qui m'a permis de les mener à bier
- Monsieur le Professeur ARSAC pour l'intérêt qu'il a manifesté pour ces idées.
- Monsieur le Professeur BOLLIET pour m'avoir toujours guidé de ses judicieux conseils.
- Monsieur KRAKOWIAK pour l'aide précieuse dont il m'a fait profiter lors de la rédaction de cette thèse.
- Madame SAUCIER pour avoir accepté de participer au jury.
- Messieurs BOCKSENBAUM, GUIBOUD-RIBAUD, OTRAGE et ULLMANN pour leurs fructueuses discussions et suggestions.
- Monsieur ROHMER pour l'étude d'un simulateur de systèmes hiérarchisés.
- Messieurs CHUPIN, DROUET, les Ingénieurs et Techniciens de l'I.F.P. pour avoir créé le GEOPROCESSEUR en accord avec ces idées.
- Les membres de l'équipe de recherche en architecture d'ordinateur de l'ENSIMAG pour avoir adopté ce mode de pensée et l'avoir confronté à de nombreuses applications.
- Madame DIAZ, Monsieur RASOLONJATOVO et le personnel du service de tirage pour la réalisation matérielle de cet ouvrage.

F.ANCEAU

Contribution à l'étude des systèmes hiérarchisés de ressources dans l'architecture des machines informatiques.

TABLE DES MATIERES

pages

pages

pages

I N T R O D U C T I O N

1. MOTIVATION
2. ETAT DE LA DEFINITION DE LA NOTION DE PROCESSUS
3. PRESENTATION INFORMELLE DES CONCEPTS UTILISES

1 - MOTIVATION

Le travail présenté ici n'est qu'une tentative d'organisation d'un sous-ensemble de l'informatique concerné principalement par la structure interne des ordinateurs.

Dans ce domaine, où la vérité n'est pas une, la notion d'optimum évolue au point d'y devenir insaisissable parce qu'elle tient compte de notions subjectives de commodité et de clarté. Les compromis qui en découlent, et sur lesquels reposent des réalisations, entraînent la création de styles et d'écoles qui définissent une Architecture, forme technique de l'Art.

Loin d'être déconcertante, cette constation nous permet d'entrevoir des développements insoupçonnés de cette technique d'où émergera, peut-être, une science à la recherche de ses concepts primaires.

La démarche que nous avons suivie a consisté à réutiliser, en les adaptant, les méthodes et concepts issus de l'étude des systèmes d'exploitation des ordinateurs. En effet, il nous semble que la structure interne des machine informatiques ne doit être qu'un prolongement de celle de leurs systèmes d'exploitation, et que l'étude doit être menée globalement, sans à-priori concernant la répartition des fonctions entre le domaine technologique et celui de la programmation.

Dans ce travail, nous nous sommes plus attachés à l'aspect conceptuel des choses qu'à l'étude de leurs propriétés déductives. Notre but s'est donc limité à définir les objets qui nous semblaient dignes d'intérêt et les relations qui les lient. Dans l'état actuel de ces recherches, il nous sembl en effet plus important de comprendre la nature des choses plutôt que d'en rechercher des propriétés déductives souvent inexploitables.

Ce travail se décomposera en trois parties. La première présentera le formalisme introduit tandis que les suivantes le justifieront par des exemples et des applications.

La première partie consistera en la définition des structures de processus/ processeurs imbriqués. Il est en effet fréquent que, dans la structure interne des machines informatiques, un processus d'un certain niveau, se comporte comme un processeur plus élaboré vis-à-vis des processus du niveau immédiatement supérieur.

Nous étudierons, dans le cadre de telles structures, les problèmes liés à l'allocation des différents processeurs aux processus qu'ils doivent exécuter.

Ces notions seront ensuite étendues au cas des structures de ressources. En effet, une ressource peut être considérée comme un processeur interprétant les requêtes à l'aide desquelles elle est utilisée.

L'étude des problèmes de synchronisation dans de telles tructures nous conduira ensuite à proposer des outils adaptés,car les outils classiques ne présentent pas toute la généralité requise pour cette application.

Le fonctionnement global de ces structures sera ensuite envisagé et conduira à définir des mécanismes d'enchaînement pour l'exécution des opérations de synchronisation entre les différents niveaux.

La seconde partie traitera de l'application de ce formalisme aux principaux mécanismes rencontrés dans l'étude de l'architecture des machines informatiques (interruptions, gestion des processus, parallélisme d'exécution).

Une troisième partie présentera la réalisation d'un hyperviseur microprogrammé pour un calculateur prototype, dans le but de concrétiser les notions précédentes.

2 - ETAT DE LA DEFINITION DE LA NOTION DE PROCESSUS

Le travail présenté repose sur une tentative de définition de la notion de processus. Son but est de proposer une notion applicable aux différents niveaux de la réalisation d'une machine informatique (machine physique, micro-programmation, programmation) et incluant les relations de dépendance entre ces niveaux.

La définition de la notion de processus est l'un des grands sujets de réflexion et de discussion depuis bientôt près d'une décennie entre les théoriciens de l'informatique et les concepteurs de systèmes d'exploitation. Peu de travaux font à ce niveau intervenir la structure interne des machine physiques dans leurs raisonnements.

L'un des aspects de ce travail consiste à proposer certaines généralisatior du concept de processus de manière à combler cette lacune.

De manière à suivre l'évolution chronologique de cette notion, citons quelques-unes de ses définitions les plus marquantes:

- E.W. DIJKSTRA [28]
  "Cooperating sequential processes" EWD 123, 1965

  p.8: *"The technical term from what we have called "rules of behaviour is an algortihm or a program. (It is not customary to call it "a sequential program" although this name would be correct). Equipment abl to follow such rules, "to execute such a program" is called "a general purpose sequential computer" or "computer" for short ; what happens during such a program execution is called "a sequential process".*

  p.10: *"The subject matter of this monograph is the cooperation betwee loosely connected sequential processes...."*.

- Jack B.DENNIS et Earl C.HORN [26]
  Programming semantics for multiprogrammed computation" CACM, mars 1966

  "A process is a locus for control within an instruction sequence. Tha

is a process is that abstract entity which moves trough the instructions of a procedure as the procedure is executed by a processor".

- A.N. HABERMANN [40]
"On the harmonious co-operation of abstract machines"
Thèse THE Eindhoven, octobre 1967

p.18: *"... a "sequential process" can be considered as a sequence of actions that will be performed when an imput tape has been supplied to an abstract machine. A sequence of actions in which a machine processes a task (from homing position to homing position) is called a "run" of the abstract machine".*

- Robert C.DALEY et Jack B.DENNIS [24]
"Virtual memory, processes and sharing in multics" CACM, mai 1968

p.307: *"Several interpretations of the term "process" have come into recent use. The most common usage applies the term to the activity of a processor in carrying out the computation specified by a program. In MULTICS, the concept of process is intimately connected with the concept of address space. Processes stand in one-to-one correspondence with virtual memories. Each processe runs in its own address space, which is established independently of other address spaces. Processes are run on a processor at the discretion of the traffic controller module of the supervisor".*

- Butler W.LAMPSON [50]
"As sheduling philosophy for multiprocessing ssytems" CACM, mai 1968

p.347: *"We consider first, therefore, the basic ideas of process and static vector and the properties of processes which are, so to speak, independent of the hardware on which they may happen to be executing".*

- N. WIRTH [71]
"On multiprogramming, machine coding and computer organization"
CACM, septembre 1969

p.490: *"... a process is a logical piece of a program, which at any instance contains a program point denoting the instruction currently being executed".*

- Per BRINCH HANSEN [16]
"The nucleus of a multiprogramming system"
CACM, avril 1970

p.238: *"More precisely, an internal process is the excution of one or more interruptable programs in a given storage area.*

.....

A sharp distinction is made between the concepts program and internal process. A program is a collection of instructions describing a compu tational process, whereas an internal process in the execution of thes nstructions in a given storage area".

- C.BETOURNE, J.BOULENGER, J.FERRIE, C.KAISER, S.KRAKOWIAK, J.MOSSIERE [12]
"Présentation générale du système ESOPE"
Congrés d'informatique, septembre 1970

p.11/13: *"Un processus est une entité dynamique, elle représente l' exécution d'une procédure, travaillant sur des données, sur une unité centrale ou un canal d'E/S.*

A tout instant un processus est complétement décrit, par définition, par son vecteur d'état, c'est-à-dire la quantité minimale d'informatio nécessaire pour le refaire partir, si on l'interrompait à cet instant"

- Robert H.THOMAS [67]
"A model for process representation and synthesis"
Projet MAC, juin 1971

sec.2.2., p.27: *A process is an activity comprised of a time-ordered sequence of actions. A process is an abstract entity and therefore can not be directly observed. The evidence for the existence of a process is change.*

.....

It is useful to think of there being both a passive member and an activ agent associated with a process. As the process evolves the passive member changes in response to the actions of the active agent.
The condition of the passive member is described by the process state At any given time the state of a process, together with the "rules" used by the active agent to change it, represents all there is to be known about the process. The state describes all that is accessible to the active agent. An important idea has been introduced:

The extent of the changes resulting from the acti-

vity of a process is isolated ; the effect the actions of a process can have is limited to that whi is accessible from its state.

The active agent is called the processor. Although the process representation method to be presented is capable of describing the time multiplexing of processors, this dissertation is not concerned with such issues. Consequently each process is assumed to have its own processor. The periods of process inactivity the model is capable of describing do not result from the non-availability of physical processor but rather occur whenever a process temporarily runs out of things to do awaits an occurrence that will enable it to continue".

- William B.EASTON [32]
"Process synchronisation without long term interlock"
Third Symposium on operating systems principles, Standford University, octobre 1971

p.95: *"A process is an object which consists of a virtual-processor state, a description of an address space, and some historical information".*

- J.J. HORNING et B. RANDELL [42]
"Process structuring"
ACM Computing Survey, mars 73

p.7: *" ... we use the concept of "process" as a mathematical tool to explain, predict, and understand the behavior of a class of physical devices exemplified by digital computer systems. These devices (which are viewed as "processors") are characterized by the fact that their interesting behavior is predictable, and consists of sequences of values of well-defined physical quantities (e.g., voltages on wires, magnetizations of cores, characters printed on paper), which represent the "information of a process to a processor is similar to the relation of a theory of physics (e.g., the "law of gravitation") to the objects of that theory (e.g., the motion of planets) ; the theory is useful to the degree that it provides a sufficiently close approximation to its objects, yet remains understandable. An important difference is that processes are often developed before the processors they model exist. If a process exhibits desirable properties, then it is generally possible to construct a corresponding processor, using the process as a specifica tion.*

Our definitions are based on the well-known concepts of state variables, state variable sets, and states . State variables are elementary quantities which can assume certain well-defined values. A set of named state variables constitutes a state variable set. An assignment of values to all the variables in a state variable set defines a state of the set ; conversely, a set defines a value for each state variable. The set of possible states for a given state variable set . (N-B. The definitions of state and state space are formally equivalent to those used in state machine theory . However, we use the state variables to introduce a structure into states that is absent from the classical theory..."

p.8: *"A computation in a state space is a sequence of states from that space. The first element of the sequence is its initial state, the last (if it is finite), its final state".*

- G. BELPAIRE et J.P. WILMOTTE [11]
"Correctness of realisation of levels of abstraction in operating systems"
Aspects théoriques et pratiques des systèmes d'exploitation - Colloque IRIA, avril 74

"This paper presents a method of proving properties of the implementation of levels of abstraction in hierarchically structured Operating Systems .
Within such systems a given level of abstraction is a representation for data types and information structures and for operations on these types and structures. Processes can be seen as computations of these operations.
Processes of level 1 are realized by the processors of level 0 (the hardware processors). For any two adjacent levels L (low) and H (high) the processes of level L are seen as (virtual) processors realizing the processes of level H. Processors at level L can be specialized processors (e.g. the I/O handler) able to realize only specific operations. Processes at level H can consist of operations of different kinds, needing to be realized by specific processors. Usually, the number of level L processors is smaller than the number of level H processes. A scheduling of the realization of Level H by level L is

therefore necessary: for every pair of adjacent levels, there must be a scheduling program whose function is to assign processors to processes and to establish the necessary queues. This scheduling program must not neccesarily be a program in the usual sense (cf. for instance the multiprogramming of sixteen concurrent processes in the Venus machine).

Processes are considered to be the basic objects which form a computing system. Usually, a process is defined by three kinds of objects:

1. *states S,*
2. *transformations (also called transitions or actions) which are mappings from S to S,*
3. *initial states".*

Nous constatons que ces définitions se regroupent autour des deux notions suivantes:

- un processus est une entité formelle correspondant à l'exécution d' un code par un processeur. Cette notion se ramène soit à considérer l'évolution d'un point de contrôle, soit une séquence d'états successi
- un processus est un objet ayant une sorte d'indépendance d'exécution vis-à-vis de ses semblables.

La mise en œuvre pratique de la première de ces notions est rarement réalisée. Par contre, il est fréquent que les réalisations pratiques relèvent de la seconde notion (même si quelquefois leurs auteurs auteurs s'en défendent).

D'un point de vue étymologique PROCESSUS est un mot latin signifiant progrés. Son sens moderne est: méthode, développement, marche.
(ex.: le processus de transformation de l'acier).

Ce sens courant correspond assez bien à la première notion présentée, tandi que, à la seconde, correspond plutôt un être chargé d'exécuter cette méthod (ex.: le processus de gestion d'un organe périphérique).

Avec cet usage, un processus devient donc un être capable d'exécuter un certain travail et est utilisé comme outil par d'autres êtres (éventuellement humains). Certains auteurs [50, 71, 42, 11] ont établi des formalisa du concept de processus qui aboutissent à le rapprocher de la notion de ma ne séquentielle.

Dès que l'on désire faire intervenir la gestion des travaux exécutés par un processus, soit séquentiellement, soit simultanément, il est nécessaire d'étendre cette notion à celle d'un groupement de machines séquentielles.

Du point de vue de la sémantique de la langue française, appliquée à l'informatique, il est nécessaire de se poser la question suivante:

- doit-on conserver le mot processus pour nommer les êtres effectivement utilisés et, de ce fait, l'écarter de sa définition officielle?

- ou doit-on admettre que ces objets ne doivent plus être appelés des processus mais, par exemple des groupements de machines séquentielle

Le choix, dans ce travail, de la première attitude, ne doit pas être considéré comme une prise de position sur ce problème mais, simplement comme le désir de se situer dans une certaine ligne de travaux vis-à-vis d'un concept universellement pressenti.

Il faut noter que certains auteurs [19] , conscients de ces problèmes, ont introduit le nom de "tâche" pour parler de ces êtres.

3 - PRESENTATION INFORMELLE DES CONCEPTS UTILISES

3/1. CONCEPT D'INTERPRETATION

3/1.1. Sémantique des langages algorithmiques

Le concept d'interpréteur est étroitement lié à la définition sémantique des langages algorithmiques [53].

La sémantique d'un langage algorithmique est toujours définie par la description dans un autre langage, supposé connu, du fonctionnement d'une machine apte à exécuter tous les programmes écrits dans le langage étudié.

Exemples:

- La sémantique du langage PL/1 est décrite à l'aide du langage VDL [70].
- La sémantique du langage d'instruction utilisé par les machines des séries IBM 360 et SIEMENS 4400 a été décrite à l'aide du langage APL [33].

Certains langages ont été conçus de manière à faciliter la description de certaines classes de machines. En effet, ils simplifient l'expression des mécanismes intervenant dans l'exécution des primitives des langages à étudier.

Exemple:

- Le langage VDL, dit "de Vienne" [70] est un tel langage.

La machine décrite sera appelée un interpréteur du langage considéré. Elle n'est pas unique pour un langage donné.

L'interpréteur utilisé pour définir la sémantique d'un langage algorithmique est souvent très différent de ceux utilisés pour exécuter, de manière efficace, les programmes écrits dans ce langage. En effet, ces deux types de machines répondent à des buts différents. Le premier doit être le plus compréhensible possible et le second le plus performant possible. Cette adaptation au but recherché se fera d'une part,en choisissant le langage de description tel que ses primitives présentent des propriétés de concision ou d'efficacité convenable et, d'autre part, en choisissant un algorithme d'interprétation le plus adapté possible au but recherché.

3/1.2. Unités de traitement

Certains interpréteurs seront réalisés technologiquement, celà implique que leur algorithme d'interprétation ait été choisi en fonction de cette réalisation particulière. Ils constituent alors des machines appelées des unités de traitement dont le rôle est d'exécuter des programmes écrits dans le langage considéré.

Le passage de l'interpréteur utilisé pour définir la sémantique d'un langag à une machine qui l'interprète résume l'art de l'architecte en machines informatiques. Cette démarche ne s'applique pas seulement à l'interpréteur du langage dans lequel sont écrits les programmes soumis au traitement d'u ordinateur, mais aussi à un grand nombre d'organes dont certains doivent traiter, à titre de chaînes d'entrée, des séquences de signaux électriques issus du milieu extérieur.

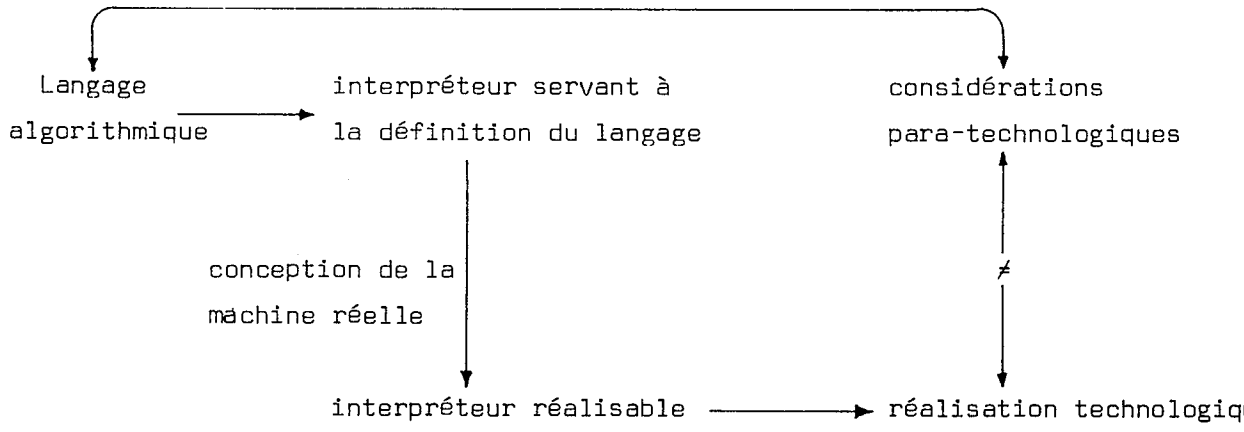
La quasi totalité des ordinateurs exécutent des programmes écrits dans des langages d'instruction plus ou moins adaptés à un usage particulier. (ex.: le langage d'instruction de l'ordinateur BURROUGH B 6500 est adapté à l'exécution de programmes primitivement écrits dans des langages voisins d'ALGOL [23]). La syntaxe (ensemble des règles de construction des chaînes de ces langages est très simple puisqu'elle permet l'écriture de pratiquement n'importe quelle séquence d'instructions. Cette remarque s'applique au cas de presque toutes les machines orientées vers le traitement direct des langages évolués. En effet, le travail de ces machines se décompose pr que toujours en une étape de traduction des chaînes du langage évolué en de chaînes d'un langage d'instructions adapté à cet usage, suivis d'une étape d'exécution des programmes de ces chaînes d'instructions.

La sémantique des langages d'instructions, conçu pour être directement utilisables, s'accompagne presque toujours de considérations para-technologiques (utilisation des mots comme: registres généraux, accumulateur, regis tres multiplieur/quotient, etc...) qui semblent être des vestiges d'un pass durant lequel les instructions de ces langages découlaient plus des possibi lités de la machine que l'on avait réussi à construire que de l'analyse des besoins de ses futurs utilisateurs.

Ces langages d'instruction para-technologiques ont la propriété de ne manipuler qu'un nombre fini de variables pré-définies (à l'exclusion de la mémoire souvent manipulée comme si elle était potentiellement infinie).

Il semble vraisemblable que de tels langages vont voir croître leur complexité soit dans un sens qui les rapproche de la structure de ceux utilisés par les programmeurs (ex.: la machine WANG exécute directement le langage BASIC), soit pour devenir très adaptés à l'interprétation, après une traduction simple, des langages de haut niveau qui seront de toute façon le seuls utilisés pour programmer ces machines (ex.: machines BURROUGHS' B 650 HEVELETT-PACKARD HP 3000, etc...) et ne plus être directement accessibles aux programmeurs.

Actuellement, les êtres para-technologiques manipulés par les langages d'instruction n'ont en général plus rien à voir avec les éléments technologiques de la machine qui les exécute. Par exemple, dans les ordinateurs petits et moyens, les registres dits "programmables" ne sont en fait que de positions, ou groupes de positions, dans des mémoires locales servant également pour d'autres fonctions de ces machines (par exemple pour y ranger l'état de la machine ou ceux des canaux simulés).
Le processus de la définition de la structure réelle d'une unité de traitement peut être idéalisé comme la transformation de l'interpréteur servant à la définition du langage que cette machine a charge d'exécuter en un interpréteur équivalent réalisable technologiquement. Cette transformation, devant préserver l'équivalence des algorithmes, pourra être, par exemple, effectuée à l'aide d'outils assurant le respect de cette condition.



3/1.3. Interpréteurs imbriqués

Les structures d'interpréteurs imbriqués sont largement utilisées dans la réalisation pratique des ordinateurs.

Très souvent une machine élémentaire (appelée micro-machine) réalisée techr logiquement, est conçue pour interpréter un langage d'instructions très sin ple (appelées micro-instructions). Un ensemble d'interpréteurs (appelés micro-programmes) est écrit à l'aide de ce langage pour réaliser l'exécutic de commandes formulées dans des langages plus complexes (souvent plus dynamiques), par exemple, un langage d'instructions évolué et un langage de commande et de gestion d'organes périphériques.

Ces assemblages ne se limitent pas toujours à deux niveaux. Par exemple, nous pouvons voir, sous la condition d'adopter un point de vue suffisamment général, la micro-machine de l'ordinateur IBM 360 modèle 25 (unité 2025)[45 comme interprétant, dans une première étape, le langage de micro-instructio du modèle 30 de la même gamme(ou celui de la machine ENDICOTT). Tandis que cette dernière machine interprète à son tour le code 360 et les langages de gestion et de commande des organes périphériques.

3/1.4. Extension du concept d'interprétation

D'un point de vue intuitif, une interprétation peut être vue comme l'exécution et l'enchaînement de primitives (instructions) soumises par un élément (le processus) à une machine décrite par un algorithme (le processeur) en vue de leur exécution. Cet algorithme peut, soit s'exécuter localement, soit soumettre de nouvelles primitives à son propre processeur et ainsi de suite jusqu'à ce qu'il n'y ait plus de sous-traitance.

3/2. RESSOURCE

Nous conviendrons ici d'appeler ressource, un organe réel ou formel susceptible d'être utilisé par d'autres organes.

ex.: - unité de traitement,
- canal d'entrée-sortie,

- mémoire,
- compilateur,
- module de gestion mémoire.

Une ressource sera utilisée par l'intermédiaire de requêtes appartenant à un langage L_r dit d'accès à cette ressource.

ex.: dans le cas d'une mémoire $L_{mémoire}$ peut être {lecture, écriture

Ce point de vue sous-entend que toute ressource inclut son propre mécanisr d'accès et qu'elle peut être considérée comme étant l'interpréteur de son langage d'accès L_r . La différence entre la notion de ressource et celle d'interpréteur provient du fait que, dans un cas, on s'intéresse plus à la nature de l'organe ressource (état interne, fonctions réalisées), et dans l'autre cas, aux propriétés du langage interprété.

3/3. DECOMPOSITION HIERARCHIQUE DES SYSTEMES D'EXPLOITATION

De nombreux systèmes d'exploitation sont structurés de manière hiérarchiqu soit en couches, soit de manière arborescente.

Nous devons toutefois noter qu'ils sont souvent structurés par des relatio différentes.

3/3.1. Relations utilisées pour structurer les systèmes d'exploitation

Nous pouvons mettre en évidence les relations suivantes dans la définition des systèmes d'exploitation:

- relation d'interprétation et d'allocation de ressource,
- relation de création des êtres,
- relations de privilèges,
- relation d'adressabilité.

La relation de création permet de construire une structure de dépendance entre les êtres basée sur le fait que si un être en crée un autre, il lui cède une partie de ses propres ressources.

La relation de privilèges permet de classer les êtres d'un système suivant la nature des privilèges, c'est-à-dire des droits qui leur sont accordés

(droit d'accéder à des données, droit d'exécuter des modules, etc...).

ex.: Le système MULTICS [24] classe les êtres dans une structure d'"anneau suivant les privilèges qui leur sont accordés.

La relation d'adressabilité permet aux objets du système de se nommer mutuellement.

ex.: Dans de nombreux systèmes existe une structure d'adressage arborescen dont certaines articulations représentent les objets du système consi déré. Cette structure permet de définir le nom d'un objet, relativement à un autre, comme le chemin qu'il faut parcourir dans cette arbo rescence pour atteindre la première de ces articulations en partant d la seconde.

Les informations données par la relation d'adressabilité recoupent quelque fois celles données par la relation de privilèges, en effet les privilèges d'un objet sont en partie liés à l'ensemble d'objets qu'il peut nommer.

Chacune de ces relations peut devenir prépondérante dans la spécification d'un système. Ce choix dépend du but recherché et de l'école des concepteurs de ce système. S'il privilégie une relation, ce choix ne peut éliminer les autres qui deviendront alors souvent des appendices de la structure choisie.

ex.: - Les relations de privilèges et d'adressabilité jouent un grand rôle dans le système MULTICS [24] sous la forme des anneaux de protection et de la structure arborescente des noms.

- La relation de création de processus joue un rôle central dans la structure du système du RC 4000 [16].

- La relation d'allocation de ressource joue un rôle important dans la définition des niveaux du système THE [29].

- L'hyperviseur microprogrammé du GEOPREOCESSEUR [§ 3° partie] est organisé autour de la relation d'allocation des processeurs virtuels

3/4. NIVEAU TECHNOLOGIQUE

Le domaine de la réalisation technologique des ordinateurs est caractéri par un dynamisme très faible des structures d'adressage proposées. En effet, l'introduction ou la suppression d'organes ne peut se faire que p une intervention humaine pour connecter, ou déconnecter, les éléments ph siques correspondants. Les actions internes à de tels systèmes ne se ra nent qu'à la mise en service, ou hors service, d'équipements existants e connus, jugés utiles, inutiles ou nuisibles. De telles actions ne peuve être qualifiées de dynamiques.

Cette propriété entraîne que l'adressage des organes ne consiste plus qu en un moyen de les reprérer au sein d'ensembles finis, relativement peti La complexité des structures d'adressage est donc faible et la relation adressage ne peut plus être prépondérante dans ces domaines. Ceux-ci co tiennent, par contre, les ressources élémentaires indispensables au fonc tionnement de toute réalisation.

Il semble donc normal de considérer comme prépondérante dans cette étude une relation liée à l'allocation des ressources. Nous verrons [§ 1° par 2/3.2] que nous avons été amenés à choisir la relation d'utilisabilité ressources. Une ressource r sera dite utilisable par un être a si a peut en solliciter l'allocation. Une telle relation met en évidence d'une part, l'ensemble des ressources qu'un organe peut être amené à sol citer et d'autre part, l'ensemble des organes pouvant être amené à solli ter une ressource. Ces informations peuvent être utilisées pour définir la structure d'une machine à réaliser, et les mécanismes d'allocation et de communication devant y être inclus.

Il faut ici remarquer que la rigidité de cette relation diminue son inté rêt en ce qui concerne l'étude des systèmes programmés.

PREMIERE PARTIE

PRESENTATION DU FORMALISME

1. Notion de processus
2. Systèmes hiérarchisés
3. Outils de synchronisation
4. Mécanismes d'induction

1 . NOTION DE PROCESSUS

1/1. MACHINE SEQUENTIELLE

Une machine séquentielle est définie par un 6-uplet

$$< Q, q_o, E, S, \delta, \gamma, >$$

dans lequel

$Q = \{q_o, q_1 .., q_i, ..\}$ - est l'ensemble, fini ou infini dénombrable, de ses états internes,

q_o - est l'état initial à l'origine des temps,

$E = \{e_1, ..., e_i, ...\}$ - est l'ensemble, fini ou infini dénombrable, des vecteurs d'entrée que peut recevoir cette machine,

$S = \{s_1, ..., s_i, ...\}$ - est l'ensemble, fini ou infini dénombrable, des vecteurs de sortie que peut émettre cette machine,

δ - est la fonction de transition entre les états internes. Elle définit l'état q^{t+1} à l'instant t+1, à partir de l'état q^t et du vecteur d'entrée e^t à l'instant t ,

$$q^{t+1} = \delta(q^t, e^t)$$

γ - est la fonction de sortie, elle définit le vecteu de sortie s^t à partir de l'état interne q^t et du vecteur d'entrée e^t à l'instant t ,

$$s^t = \gamma(q^t, e^t)$$

De cette définition, il résulte que les instants auxquels une machine séquentielle fait évoluer son état, constituent une suite dénombrable d'instants

$$t_o, t_1, t_2, ...$$

Nous dirons que le temps d'une telle machine évolue de manière discrète.

1/2. NOTION DE PROCESSUS

Nous confondrons les notions de processus et de machine séquentielle. Nou devons dés lors établir un lien entre les vocabulaires utilisés pour ces de notions.

Un processus sera défini par un 5-uplet:

$$p = \langle CX, cx_o, E, S, A \rangle$$

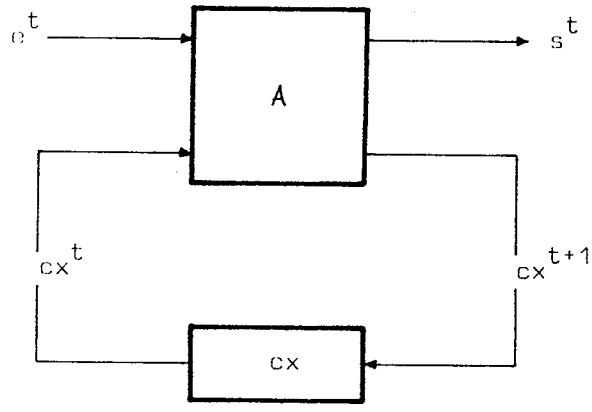
tel que:

- son état interne $cx_i \epsilon CX$ sera appelé son contexte.
- ses fonctions de transition et de sortie seront regroupées en ce que nous appellerons son algorithme A

$$\langle cx^{t+1}, s^t \rangle = A(cx^t, e^t)$$

dont les projections sont:

$$cx^{t+1} = A\delta(cx^t, e^t)$$
$$s^t = A\gamma(cx^t, e^t)$$



L'intérêt que nous porterons à un processus sera lié à la succession des transformations qu'il fera subir à son contexte.

Exemples de processus:

- un automate,
- une machine de TURING,
- un ordinateur.

1/2.1. Composantes significatives du contexte

Le contexte d'un processus se décompose habituellement en un grand nombre (finis) de composantes.

$$cx = \langle cx_1, cx_2, \ldots\ldots, cx_n \rangle \in CX_1 \times CX_2 \times \ldots\ldots \times CX_n = CX$$

ex.: Les différents mots de la zône mémoire allouée à un processus programmé.

Un tel n-uplet de composantes peut être vu comme une application de l'ensemble d'indices $IND = \{1, 2, \ldots, n\}$ dans l'ensemble des valeurs des composantes $VCX = \bigcup CX_i$ pour $i \in IND$.

$$cx : IND \longrightarrow VCX \quad \text{tel que} \quad cx_i \in CX_i$$

L'algorithme A est généralement tel que chaque transition d'état ne dépend que de seulement quelques unes de ces composantes.
L'ensemble d'indices IND peut donc être partitionné à chaque pas en:

$$IND = SIGN(A, t) \cup \overline{SIGN}(A, t)$$

nous appellerons:

- composantes significatives: $cx_s^t : SIGN(A, t) \longrightarrow VCX$
- composantes non significatives: $cx_{\bar{s}}^t : \overline{SIGN}(A, t) \longrightarrow VCX$

nous noterons:

$$cx^t = cx_s^t \mid cx_{\bar{s}}^t$$

le fait que le n-uplet cx^t est constitué des composantes significatives et non significatives.
Cette partition entraîne que:

$$cx^{t+1} = A_\delta(cx^t, e^t) = A_\delta(cx_s^t \mid cx_{\bar{s}}^t, e^t) = A_{\delta s}(cx_s^t, e^t) \mid A_{\delta\bar{s}}(cx_{\bar{s}}^t, e^t)$$

tel que:

$$A_{\delta\bar{s}}(cx_{\bar{s}}^t, e^t) = cx_{\bar{s}}^t$$

d'ou

$$cx^{t+1} = A_{\delta s}(cx_s^t, e^t) \mid cx_{\bar{s}}^t$$

et ceci quelque soit les valeurs des composantes non significatives $cx_{\bar{s}}^t$.

1/2.2. Eléments de contexte communs entre deux processus

Plusieurs processus peuvent avoir des composantes de leur contexte en commun. La cohérence du contexte de chaque processus devant être garanti des précautions doivent être prises pour éviter que cette zône ne devienn simultanément significative pour chacun des processus (celà revient à évi les conflits d'accès à cette zône). Plaçons nous dans le cas où deux pro cessus P_1 et P_2 partagent une partie de leur contexte. Soit COM l' semble des couples de indices, relativement à chacun de deux processus, d composantes communes.

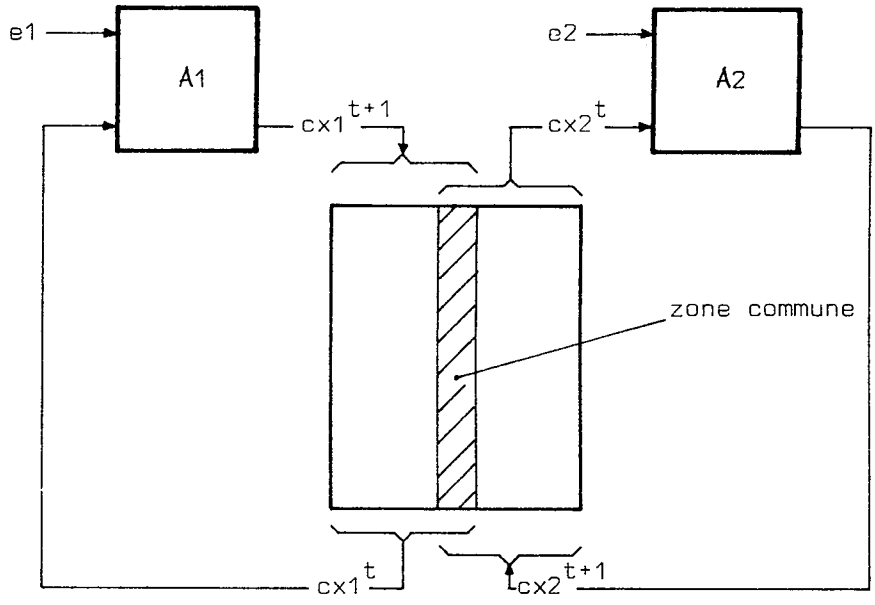
Soit a une composante commune à cx1 et cx2, si:

$$a = cx1_i \quad \text{pour} \quad P_1$$
$$a = cx2_j \quad \text{pour} \quad P_2$$

celà entraîne que le couple $(i, j) \in COM$

La condition énoncée stipule que:

$$\forall t \quad SIGN(A_1, t) \times SIGN(A_2, t) \cap COM = \emptyset$$



Une modification de composantes communes par l'un des processus peut infl encer le comportement ultérieur de l'autre au moment où il considérera ce composantes comme significatives.

D'une manière générale, l'état suivant d'un processus sera déterminé par

$$cx1^{t+1} = A1_{\delta_{s1}}(cx^t_{s1}, e1^t) \mid A2_{\delta_{\bar{s}2 \cap proj_2(COM)}}(cx^t, e2^t) \mid cx^t_{\bar{s}1 - proj_1(COM)}$$

où

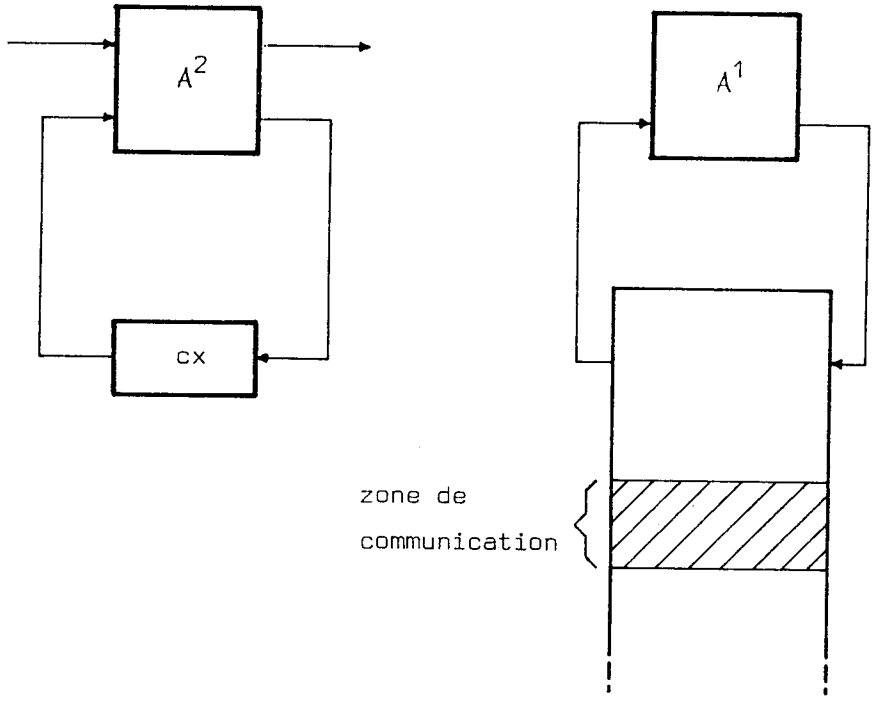
$$A2_{\delta_{\bar{s}2 \cap proj_2(COM)}}(cx^t, e2^t)$$

est la restriction de $A2_\delta$ à la modification des composantes non significativ communes du processus p_1.

1/2.3. Communications avec un processus

Les processus peuvent s'échanger de l'information de deux manières différente

- par des connexions d'entrée-sortie,
- par des zônes partagées de leur contexte.



Nous pouvons montrer que ces deux types d'échanges sont équivalents en effet:
Soit un processus P1

$$P1 = \langle CX, cx_0, E, S, A^1 \rangle$$

tel que l'une de ses transitions possibles soit influencée par la valeur d'une

composante partagée cx_i de son état, significative à cet instant.

$$cx^{t+1} = A^1_\delta(cx^t) \qquad cx_i \in SIGN(A^1, t)$$

Cette composante fut modifiée antérieurement par un autre processus à un instant $t' < t$ où elle n'était pas significative.

$$cx_i \in \overline{SIGN}(A^1, t')$$

Nous pouvons définir un processus P2

$$P2 = <CX, cx_o, E, S, A^2>$$

tel que la transition précédemment étudiée ne dépende maintenant que de la valeur d'une ligne d'entrée portant la même information.

L'état cx'^t de P2 au temps t ne dépend maintenant que du seul comportement du processus et la composante cx_i n'a plus à être significative pour cette transition.

$$cx^{t+1} = A^2(cx'^t, e^t) = A^1(cx^t) \qquad cx_i \in \overline{SIGN}(A^2, t)$$

réciproquement montrons qu'une transition

$$cx^{t"+1} = A^2_\delta(cx^{t"}, e^{t"})$$

de P2 qui dépend de la valeur des lignes d'entrée peut être remplacée, dans un processus P3 par un couple de transition influencées par la valeur d'une composante partagée de son état.

$$P3 = <CX3, cx3_o, E, S, A^3>$$

tel que si

$$cx = <cx_1, \ldots\ldots\ldots, cx_n>$$

alors

$$cx3 = <cx_1, \ldots\ldots\ldots, cx_n, cx_{n+1}>$$

la composante cx_{n+1} ne sera pas significative en t" et sera à cet instant positionnée par un autre processus qui la partagera.

$$cx_{n+1} \in \overline{SIGN}(A^3, t")$$

La transition de P3 en t" sera telle que:

$$cx3^{t"+1} = cx3^{t"}$$

la composante cx_{n+1} deviendra significative en t"+1 où nous ajouterons la transition

$$cx3^{t"+2} = A^3_\delta(cx3^{t"+1}) \qquad cx_{n+1} \in SIGN(A^3, t"+1)$$

telle que les composantes de 1 à n de $cx3^{t"+2}$ soient identiques à celle de $cx^{t"+1}$.

Remarque:

Le processus P3 possède un état de plus que le processus P2. Celà sous-entend que la notion d'équivalence que nous utilisons est assez lâche et ne s'établit que par la comparaison des composantes significatives d'une sous-suite des états. Cette attitude se justifie lorsque nous remarquons que les processus que nous manipulons comportent en général un grand nombre d'états (quelques milliers) et que souvent une information est présente sur une ligne d'entrée bien avant son test ce qui permet d'anticiper la modification de la composante qui représente cette information dans le cas d'une communication par une zône de contexte partagée entre plusieurs processus.

Nous devons noter, pour terminer la comparaison, que nous pouvons recopier sur des lignes de sortie la valeur de toutes les composantes de la zône de communication susceptibles d'être positionnées par le processus et que, réciproquement, nous pouvons remplacer ces lignes par des composantes supplémentaires constituant une zône de communication avec d'autres processus.

1/2.4. Représentation d'un processus relativement à un ensemble d'instructions

Soit un processus

$$p = \langle CX, cx_0, E, S, A \rangle$$

et un ensemble d'opérations

$$OP = \{op_1, op_2, \ldots\ldots\ldots\ldots, op_n\}$$

appelées instructions et décomposables en classes, par exemple:

- des instructions de transformation du contexte:

ex.: $a \leftarrow a+b$

où a et b sont des variables représentant des composantes du contexte cx.

- des instructions de lecture de vecteurs d'entrée:

ex.: $a \leftarrow e_i$

où a est une variable et e_i un vecteur d'entrée de E

- des instructions d'émission de vecteurs de sortie:

ex.: $s_i \leftarrow a$

où s_i est un vecteur de sortie de S et a une variable.

- des instructions de contrôle agissant sur l'enchaînement de l'exécution des instructions,

ex.: si $\Phi(a)$ alors aller à @

où Φ est un prédicat sur la variable a et @ est le nom d'une instruction.

Remarque:

Il est évidemment possible de définir des instructions relatives à plusieurs de ces classes,

ex.: test de lignes d'entrées: si $\Phi(e_i)$ alors aller à @

Nous pouvons définir une représentation du processus p , relativement à l'ensemble d'instructions OP , par le processus:

$$OP(p) =< CX^{op}, cx_o^{op}, E, S, \mathcal{P} >$$

dans lequel

$\mathcal{P}$ représente l'expression de A sous la forme d'une suite d'instructions de OP appelée programme. L'exécution d'une instruction constituera une transition élémentaire de OP(p) (un pas du programme).

CX^{op} représente l'ensemble des contextes possibles de OP(p). Ses composantes constituent les données de A . Il est obtenu par un certain codage (par exemple sous la forme de variables entières) de CX x CX'

$$CX^{op} \subseteq \text{codage } (CX \times CX')$$

dans lequel

CX représente l'ensemble des contextes possibles du processus initial

CX' représente un ensemble de contextes auxiliaires liés au fait qu'une transition élémentaire du contexte du processus p peut se trouver décomposée en plusieurs opérations op_i dans $OP_{(p)}$.

Remarque: La représentation d'un processus, relativement à un ensemble donné d'instructions, n'est pas unique.

ex.: soit un processus

$$p = <CX, cx_o, E, S, \mathcal{P} >$$

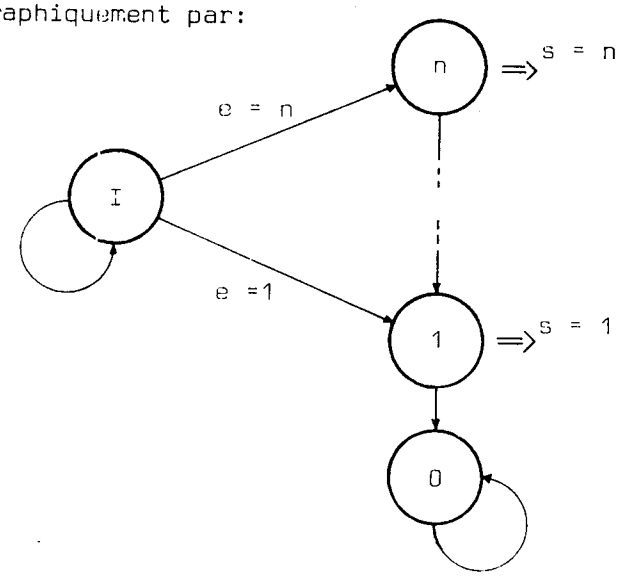
avec

$$CX = \{0, 1, 2, \ldots\ldots, n, I\}$$

$$cx_o = I$$

$$E = \{1, 2, \ldots\ldots\ldots, n\} = S$$

Λ représenté graphiquement par:

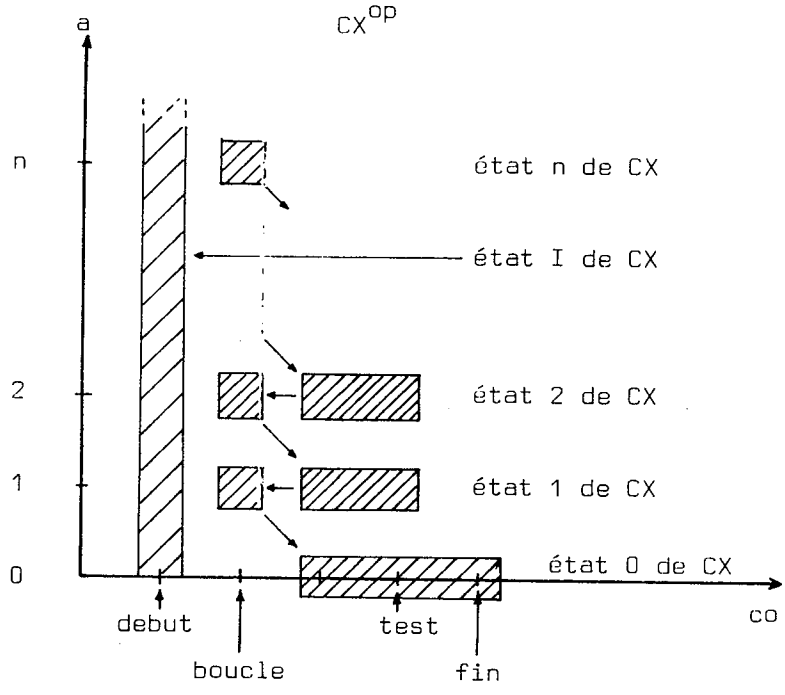


Une représentation de ce processus est

$$OP(p) = \langle CX^{op}, cx^{op}_{o}, E, S, P \rangle$$

avec

$cx^{op} = \langle a, co \rangle \in CX^{op} \sim \mathbb{N}^2$ a, co variables entières les valeurs 0, 1,....., n de a représentent les états 0, 1,....., n de CX , co représente l'état du programme P (contexte auxiliaire) et l'état I de CX .



cx_o^{op} = < a indéfini, co = début >

P = début : a ← e
boucle : s ← a
a ← a-1
si a>0 alors aller à boucle
fin

Remarque:
Le cas des programmes automodifiables (calcul d'instruction) peut être traité de la manière suivante:

- introduction dans CX^{op} de variables prenant leurs valeurs dans OP ou dans l'un de ses sous-ensembles. Un symbolisme particulier doit donc permettre de représenter des instructions en tant que constantes,
ex.: c ← "a ← a + 1"
La variable c prend pour valeur la représentation codée de l'instruction a ← a + 1 .

- représentation, à l'aide d'un symbolisme particulier, du fait que la nature d'une instruction dépend de la valeur d'une variable de type ci-dess
ex.: c (a ← a + 1 | a ← a - 1)
Cette instruction sera soit a ← a+1 soit a ← a-1 suivant la valeur de l variable c . Cette valeur se trouvant être la représentation codée de l' instruction à exécuter.

Il faut remarquer que ces symbolismes ne font pas partie de l'ensemble d' instructions OP mais doivent être inclus dans le formalisme de représentation des éléments des processus.

1/2.5. Réalisation interprétative d'un processus

Soit une représentation d'un processus p , relativement à un ensemble d' instructions OP,

$$OP(o) = <CX^{op}, cx_p^{op}, E, S, P>$$

Nous pouvons définir une réalisation une réalisation interprétative du même processus p , relativement au même ensemble d'instructions, par le processu

$$IOP(p) = <CX^I, cx_o^I, E, S, A^I>$$

tel que:

$$CX^I = CD \times CX^{op}$$

dans lequel:

- CD est un ensemble de valeurs constitué d'un seul j-uplet de constantes représentant le codage du programme P sous la forme d'une suite de constant

$$CD = \{<op_1, op_2, \ldots\ldots, op_j>\} = \{ cd \}$$

d'où $\forall t \quad cx^I = <op_1, op_2, \ldots\ldots, op_j, cx^{op}>$

- A^I est un algorithme, dit d'interprétation, dont le rôle est:

  . de sélectionner une instruction op_i dans cd d'après la valeur de certains éléments de cx^{op} (le compteur ordinal par exemple).

  . d'exécuter la transition dans CX^I indiquée par cette instructic sélectée.

Remarque: La réalisation interprétative d'un processus p , relativement à un ensemble d'instructions donné, n'est pas unique.

Exemple:

Considérons une réalisation interprétative du processus p dont une représer tation programmée est donnée dans l'exemple précédent.

$$IOP(p) = <CX^I, cx_o^I, E, S, A^I>$$

avec

$$CD = \{<I_1, I_2, I_3, I_4, I_5>\} = \{cd\}$$

$$CX^I = CD \times \mathbb{N}^2$$

A^I est tel que $\forall t \quad SIGN(A^I, t) = <a, co>$

d'où $<a, co>^{t+1} \leftarrow A_\delta^I (cd, a^t, co^t, e^t)$

Remarque:

Dans le cas de programmes automodifiables, les instructions représentées par des expressions de la forme

$$cx_i^{op} \{op_1 | \ldots. | op_m\}$$

sont à la fois des éléments de <cd> et de cx^{op}

ex.: $cd = <op_1, op_2, \ldots\ldots, cx_i^{op}, \ldots\ldots, op_k>$

1/2.5.1. Codage de la réalisation interprétative d'un processus

Nous appellerons un codage de la réalisation interprétative d'un processus:

$$IOP(p) = \langle CX^I, cx^I_o, E, S, A^I \rangle$$

le processus obtenu en codant les éléments de CX^I par exemple à l'aide de chaînes de bits, exemple: assemblage d'un programme.

exemple: soit un codage de la réalisation interprétative présentée dans l'exemple précédent.

```
cd = < 111oo11o  oooooo1o  111oo111oo11o11o ,
       lire      e         @a
@boucle → 111oo111  oooooooo1  111oo111oo11o11o ,
          sortir    s          @a
          11oooooo  oooooooo1  111oo111oo11o11o ,
                incr           @a
          1111oooo  ooooo11o  111oo111oo11o11o  oooooo111o1o111o ,
          test      >o        @a                @boucle
          oooooooo  oooooooo >
                fin
```

$$cx^{op} = \langle \underbrace{\quad a \quad}_{16 \text{ bits}} , \underbrace{\quad co \quad}_{16 \text{ bits}} \rangle$$

↑ @a

Remarque: Il est fréquent qu'un tel codage ne permette pas de représenter la totalité de CX^I. Il suffit qu'il puisse représenter le sous-ensemble effectivement utilisé lors des différentes transitions de la représentation du processus p.

1/2.6. Définitions

Relativement aux définitions précédentes, nous appellerons I-processus l'êtr suivant:

$$Ip = \langle CX^{op}, cx^{op}_o , E, S, CD \rangle$$

son état instantané sera défini par: $<cd, cx^{op}>$

Nous appellerons <u>processeur</u>, compatible avec le I-processus Ip :

$$P = < E, S, A^I, C >$$

Nous appellerons <u>machine</u>, compatible avec le I-processus Ip :

$$MA = < M, P >$$

dans lesquels:

- A^I représente la définition fonctionnelle de l'unité de traitement et d'entrée/sortie constituant le processeur P ,
- M est l'ensemble des valeurs d'un vecteur m dont les composantes sont de différents types (vecteurs de bits, entiers, réels, instructions, adresses, etc...). m est tel qu'il peut contenir le vecteur d'état $<cd, cx^{op}>$ de la réalisation interprétative du processus p considéré. Le I-processus Ip sera dit être <u>rangé</u> dans M .
  M représente les organes de mémorisation de la machine MA.
  Ex.: mémoire centrale et registres connus du programmeur.
- C est un ensemble de conventions liant un I-processus, rangé dans M , au processeur P utilisé.
  - . sens de certaines composantes privilégiées de m
    ex.: compteur ordinal.
  - . éventuellement localisation de cd et cx^{op} dans m
    ex.: mémoire de code séparée.
  - . l'ensemble des instructions reconnues par le processeur P , ainsi que le codage qui leur est attribué (propriétés de A^I)
  - . les codages à utiliser pour la représentation des éléments de CX^{op}.

Tous les processeurs admettant les mêmes conventions seront dits <u>fonctionnellement équivalents</u>.

Nous dirons qu'un I-processus Ip et qu'un processeur P sont <u>associés</u> si:

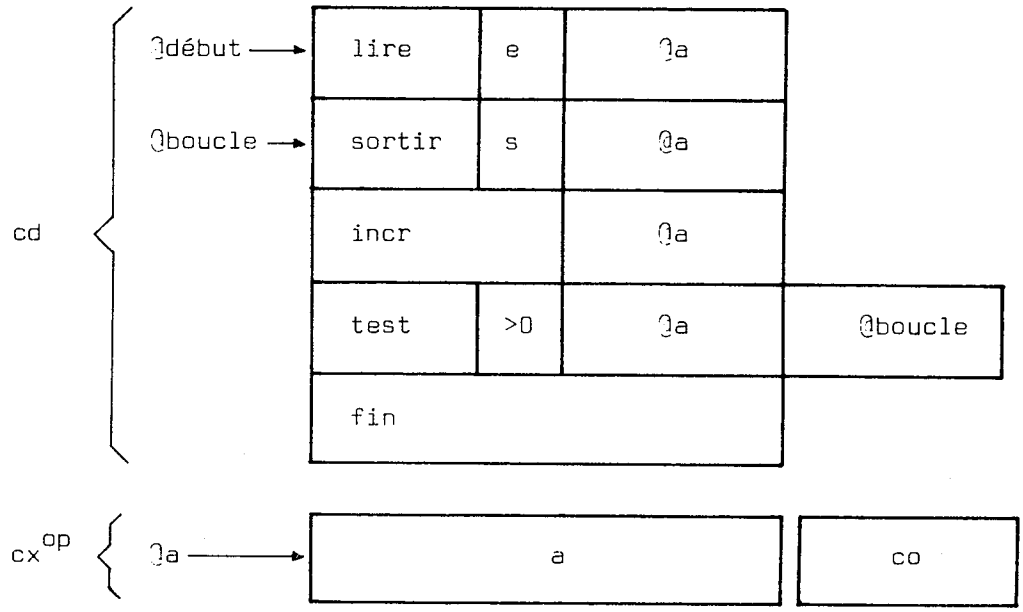
- le I-processus est rangé dans les organes de mémorisation M de la machine MA utilisant ce processeur P .

- les conventions C sont respectées de manière que si la zône de m utilisée est initialisée avec l'état $<cd, cx_o^{op}>$ la machine MA constitue alors la réalisation interprétative IOP(p) du processus p .

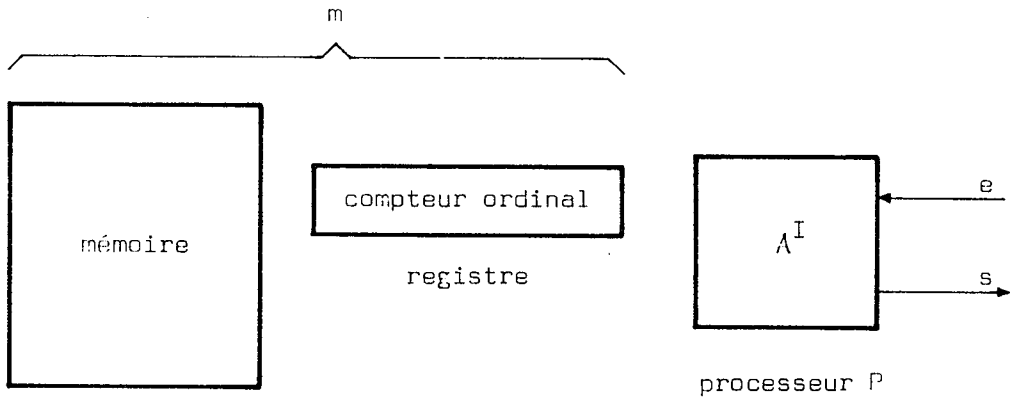
Le processeur P sera alors dit <u>exécuter</u> le I-processus Ip.

<u>exemple</u>:

Le I-processus découlant des exemples précédents est:



une machine MA peut être:



Pour associer ce I-processus et ce processeur nous rangerons cd et a d
la mémoire de la machine et co dans son compteur ordinal.

1/2.7. Remarques

- L'état instantané $<cd, cx^{op}>$ d'un I-processus Ip résume la quantité d'informations qu'il faut assurer à un processeur pour que la machine utilisée se comporte comme la réalisation interprétative du processus p .

- Nous voyons qu'un I-processus Ip peut être étudié comme une forme de processus p , en effet il contient:

. un contexte $cx^{op} \varepsilon CX^{op}$

. le code CD d'un programme P décrivant l'algorithme A d'un processu

. des actions d'entrée/sortie décrites par le code CD ,

Pour progresser, un I-processus doit être associé à un processeur.

- Les processeurs et les I-processus constituent des entités indépendantes. Le même processeur peut souvent servir à exécuter une très large classe de I-processus.

- Le codage de CD et de CX^{op} peut dépendre de l'emplacement où ils sont rangés dans m . Le codage de ces éléments devra donc être terminé après que leur implantation ait été décidée (variables translatables au chargement).

- Plusieurs processus peuvent utiliser des sections communes de code dans la mesure où ils ne les modifient pas (sections réentrantes de code).

- Les états instantanés de plusieurs I-processus peuvent co-habiter dans m (si la taille de ce dernier le permet). Nous verrons qu'au processeur, tel que nous l'avons défini, nous devons ajouter un mécanisme, dit d'allocation, choisissant le I-processus à exécuter parmi ceux rangés dans M

- Un processeur peut se décomposer en:

. un processeur d'entrées/sorties $< E, S, A^{I}_{\gamma}, C' >$

. un processeur de traitement $<A^{I}_{\delta}, C">$ aussi appelé unité de traitement.

- Nous verrons qu'un processeur, au sens où nous l'entendons, peut être constitué de la duplication d'une même unité physique, pour des besoins c performances (machines multi-processeurs banalisés).

1/2.8. Imbrication des réalisations d'un processus

Soit une réalisation interprétative du processus p

$$IOP(p) = < CX^I, cx_o^I, E, S, A^I >$$

Considérons maintenant une représentation du processus IOP(p) relativement à un ensemble d'instructions OP'

$$OP'(IOP(p)) = < CX^{op'}, cx_o^{op'}, E, S, P^I >$$

dans laquelle

$cx^{op'} = codage(cx^I, cx")$ où cx" est un nouvel ensemble de composantes auxiliaires.

En général ce second codage est trivial, c'est-a-dire que:

$$cx^{op'} = <cx^I, cx"> = <cd, cx^{op}, cx">$$

d'où $cx^{op'} \supset cx^{op}$

P^I est la décomposition de l'algorithme d'interprétation A^I en une séquenc d'instruction de OP' (généralement plus simple que OP).

Considérons maintenant une réalisation interprétative, relativement à OP', de IOP(p)

$$IOP'(IOP(p)) =< CX^{I'}, cx_o^{I'}, E, S, A^{I'} >$$

dans laquelle

$$cx^{I'} = <cd', cx^{op'}>$$

avec cd' un vecteur de constantes représentant P^I

compte tenu de la remarque précédente:

$$cx^{I'} = <cd', \underbrace{cd, cx^{op}}_{cx^I}, cx">$$

d'où

$$cx^{I'} \supset cx^I$$

$A^{I'}$ est l'algorithme d'interprétation des instructions de OP'. Nous pouvor

définir un I-processus:

$$Ip' = \langle CX^{op'}, cx_o^{op'}, E, S, CD' \rangle$$

d'état $\langle cd', cx^{op'} \rangle = \langle cd', cd, cx^{op}, cx'' \rangle$

Cet état contient l'état $\langle cd, cx^{op} \rangle$ du I-processus $Ip = \langle CX^{op}, cx_o^{op}, E, S, CD$

Nous pouvons également définir un processeur:

$$P' = \langle E, S, A^{I'}, C' \rangle$$

qui se trouve être une partie de la réalisation du processeur

$$P = \langle E, S, A^{I}, C \rangle$$

La machine correspondante est :

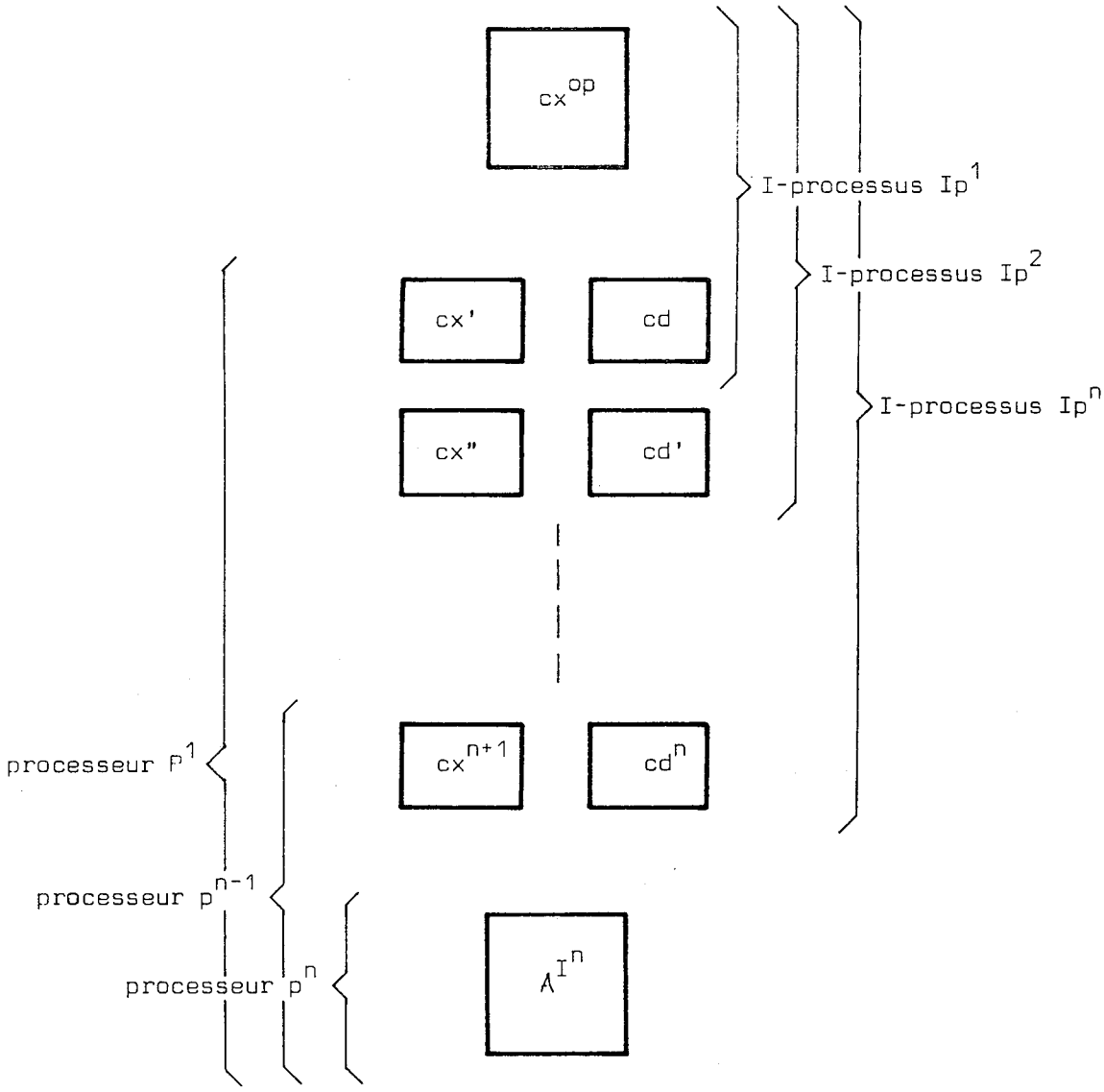
$$MA' = \langle M', P \rangle$$

est telle que $m' \supset m$ dans laquelle $m \in M$ de la machine $MA = \langle M, P \rangle$

Remarque:

Cette décomposition peut se poursuivre aussi loin que l'on désire. La chaîne ainsi formée se terminera toujours par un processeur dont la description n'est que fonctionnelle, c'est-à-dire ni représentée par un programme ni codée. L'hypothèse simplificatrice concernant le codage des éléments de cx^{op^n} se trouve pratiquement vérifiée à tous les niveaux correspondant à la réalisation interne des machines considérées.

La situation précédente peut se représenter par le diagramme suivant :



D'une manière symétrique à l'imbrication des I-processus, nous constatons que les différents processeurs introduits sont également imbriqués, dans le sens que le processeur p^{i+1} entre dans la réalisation du processeur p^{i}.

Nous appellerons étape d'interprétation l'élément de contexte

$$<cx^{i+1}, cd^{i}>$$

constitué d'un code cd^{i} et d'un contexte auxiliaire cx^{i+1} qui lui est associé. Cette étape d'interprétation pourra, suivant le point de vue que l'on adoptera, entrer dans la constitution d'un I-processus ou d'un processeur.

1/2.8.1. Graphe d'interprétation

Une telle chaîne peut être représentée par le graphe linéaire suivant:

$$GI = < AI, DI >$$

dans lequel

AI est l'ensemble d'articulations constitué de:

- un I-processus unique a_{ps} considéré comme élémentaire, c'est-à-dire supposé ne contenant pas d'autres I-processus.
- un processeur a_{pr} unique considéré comme élémentaire, c'est-à-dire supposé ne contenant pas d'autres processeurs plus élémentaires.
- des étapes d'interprétation.

DI est la relation de définition du graphe linéaire. Elle est définie récursivement.

Il résulte des définitions précédentes que:

$\nexists\, a_k$ tel que $DI(a_k, a_{ps}) \Rightarrow [a_{ps}]$ est un I-processus

$\nexists\, a_h$ tel que $DI(a_{pr},, ah) \Rightarrow [a_{pr}]$ est un processeur

Si $[a_{ps}, \ldots\ldots, a_i]$ est un I-processus

$DI(a_i, a_{pr})$ signifie que $[a_{ps}, \ldots\ldots, a_i, a_{pr}]$ est un processus.

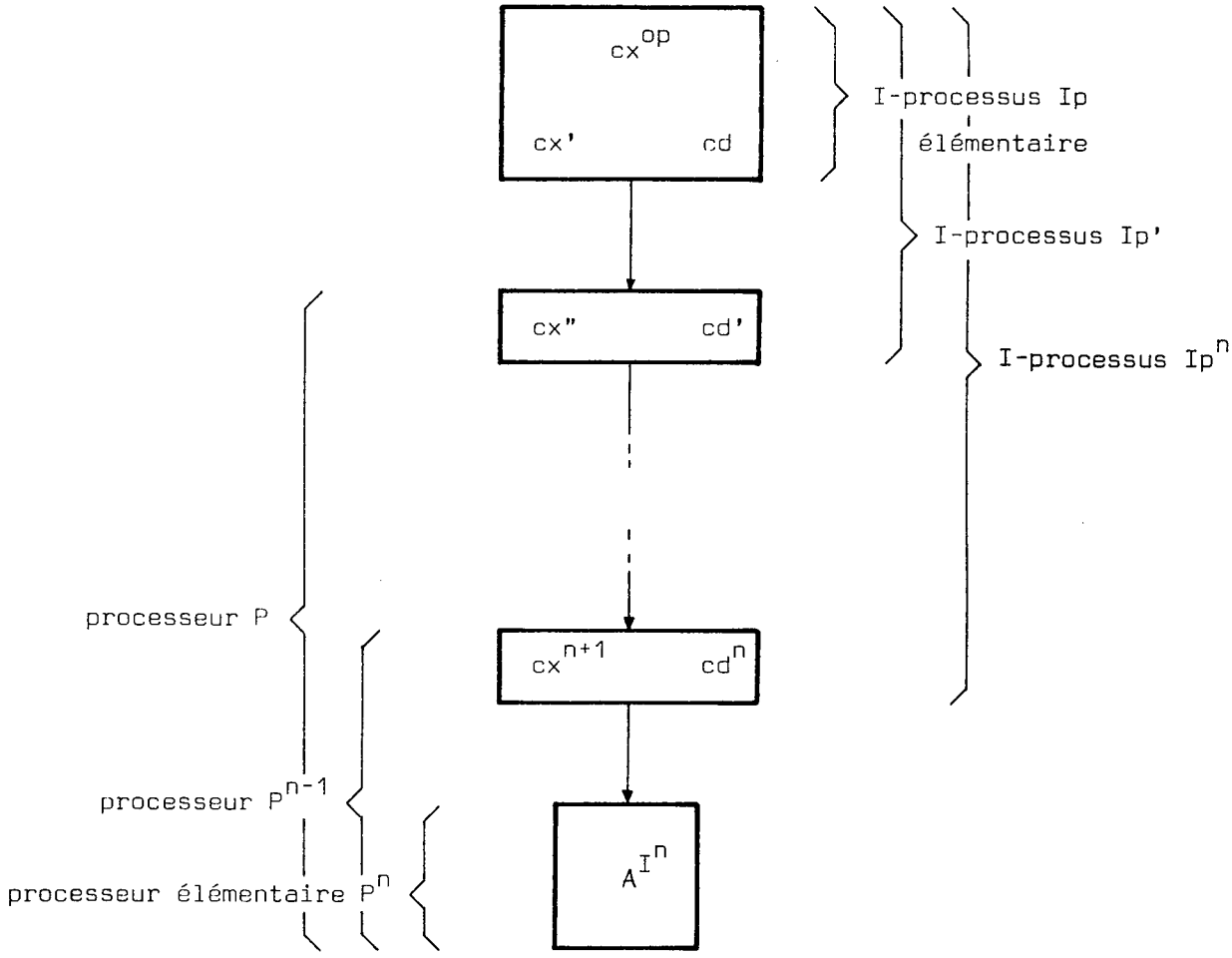
Si a_j est une étape d'interprétation $DI(a_i, a_j)$ signifie que $[a_{ps}, \ldots\ldots\ldots, a_i, a_j]$ est un I-processus.

De même si $[a_t, \ldots\ldots, a_{pr}]$ est un processeur

$DI(a_{ps}, a_t)$ signifie que $[a_{ps}, a_t, \ldots\ldots, a_{pr}]$ est un processus

$DI(a_j, a_t)$ signifie que $[a_j, a_t, \ldots\ldots\ldots, a_{pr}]$ est un processeur.

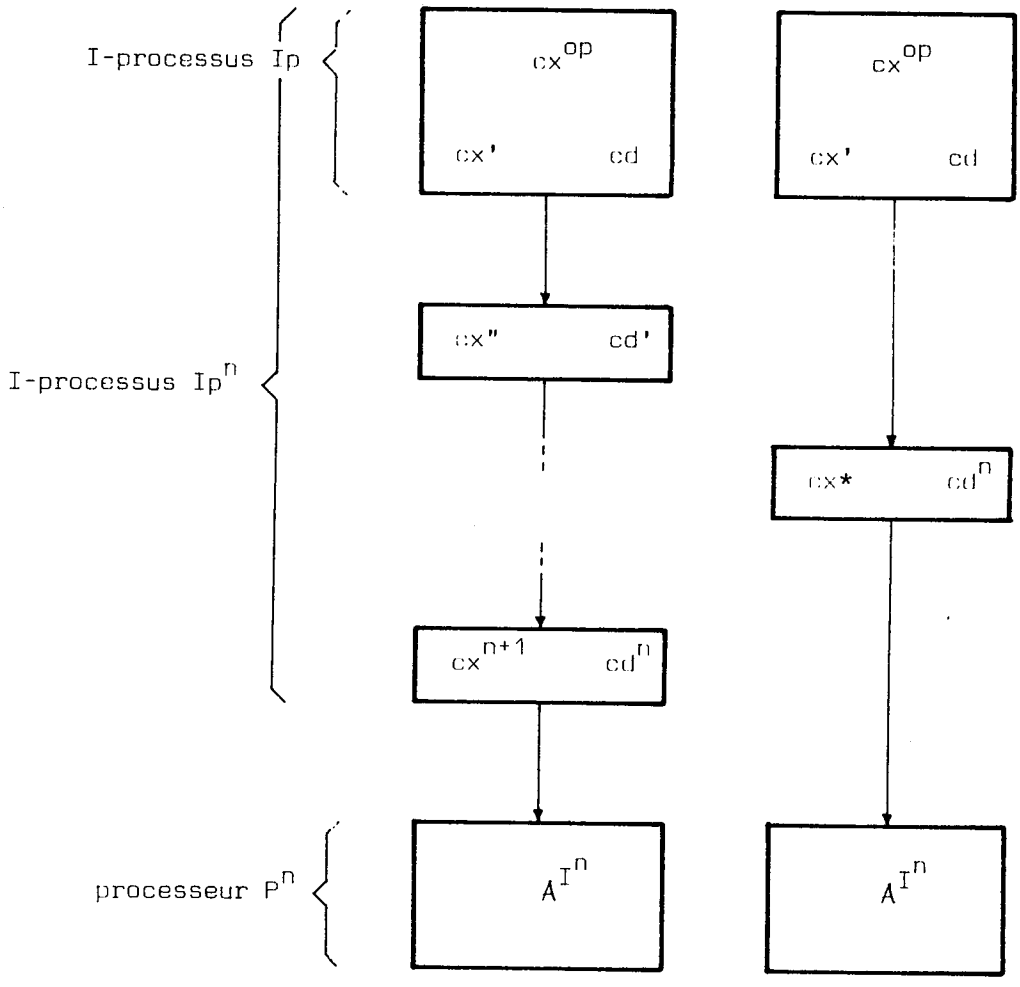
Exemple:



Une articulation de ce graphe, à l'exclusion des deux extrêmes, peut apparaître, soit dans la constitution d'un I-processus, soit dans celle d'un processeur.

Remarque:

Une chaîne d'articulations intermédiaires peut être condensée en une seule articulation.



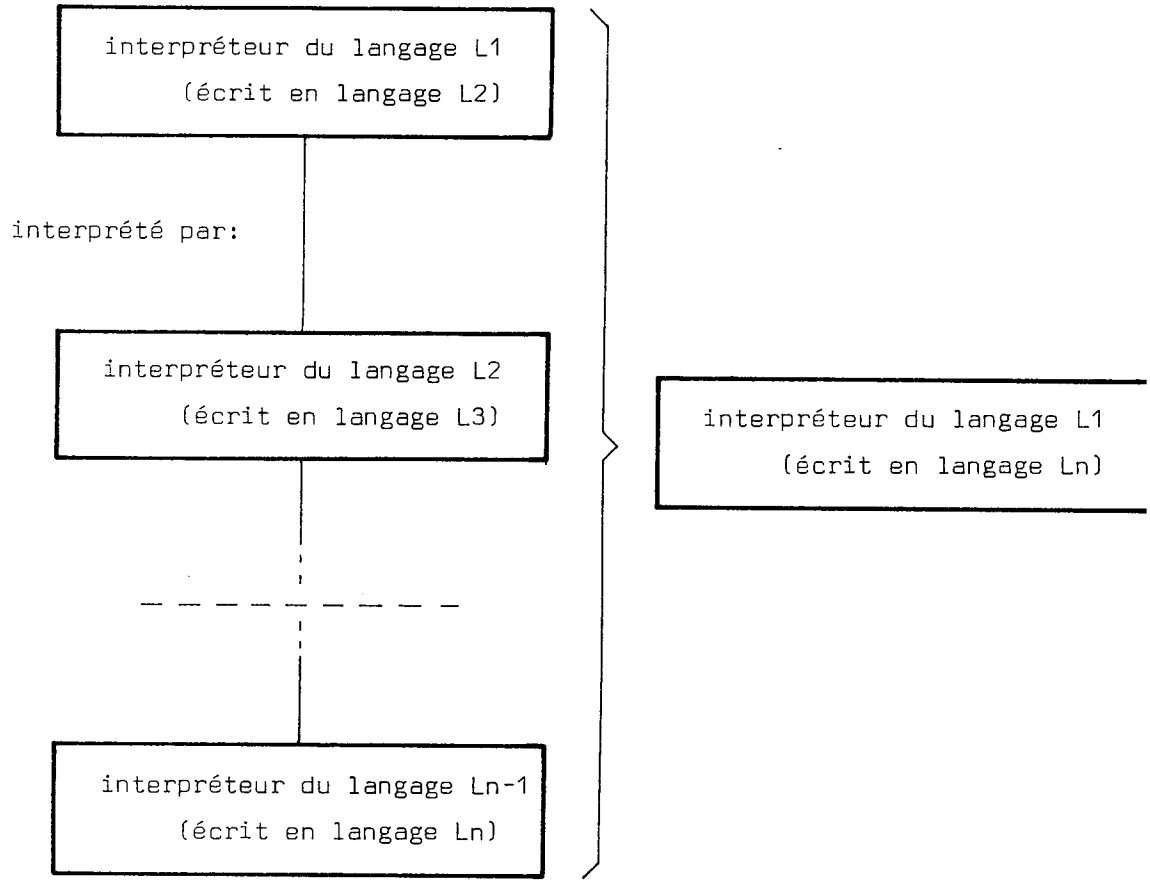
En effet le I-processus $Ip^n = < CX^{op^n}, cx_o^{op^n}, E, S, CD^n >$

a pour état $< cd^n, cd^{n-1}, \ldots\ldots, cd', cd, cx^{op}, cx', cx'', \ldots\ldots, cx^{n+1} >$

Cet état pourra s'écrire: $< cd^n, cd, cx^{op}, cx', cx^* >$ (en réordonnant les composantes) avec $cx^* =< cd^{n-1}, \ldots\ldots, cd', cx'', \ldots\ldots, cx^{n+1} >$

Exemple:

Considérons la chaîne suivante d'interpréteurs:



Cette chaîne d'interpréteurs peut être vue comme un seul interpréteur.

Exemple:

Si nous poursuivons la chaîne d'exemples précédents, nous pouvons étudier une représentation de la réalisation interprétative IOP(p) relativement à un ensemble de micro-instructions OP' .

$$OP'(IOP(p)) = < CX^{op'}, cx_o^{op'}, E, S, P^I >$$

avec

$$cx^{op'} = < cd, cx^{op}, cx" >$$

dans lequel

cx" représente des variables de travail du micro-programme et le compteur de micro-instructions.

P^I représente le micro-programme d'interprétation des instructions de OP écrit dans le langage de micro-instructions OP'.

Une réalisation interprétative, relativement à ce même ensemble de micro-instructions OP' , de IOP(p) serait:

$$IOP'(IOP(p)) = < CX^{I'}, cx^{I'}, E, S, A^{I'} >$$

avec

$$cx^{I'} = < cd', cd, cx^{op}, cx" >$$

dans lequel

cd' représente le micro-programme P^I traduit en une suite de constantes représentant le codage des micro-instructions,

$A^{I'}$ représente les fonctions des organes technologiques réalisant l'exécution des micro-instructions.

Le micro-I-processus correspondant est:

$$Ip' = < CX^{op'}, cx_o^{op'}, E, S, CD' >$$

son état $< cd', cd, cx^{op}, cx" >$ contient l'état du I-processus Ip

Le processeur correspondant:

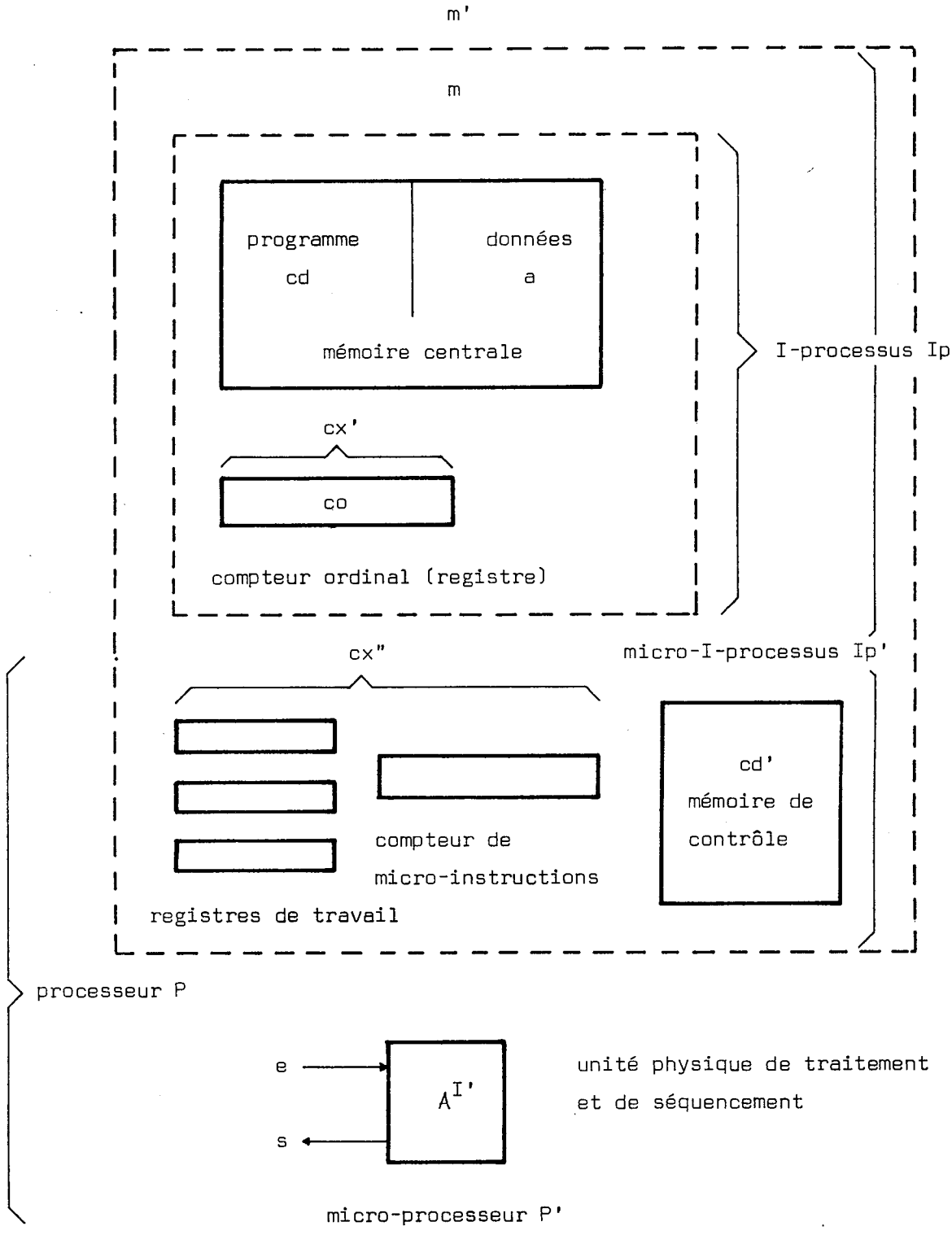
$$P' =< E, S, A^{I'}, C' >$$

appelé quelquefois un micro-processeur, définit une micro-machine

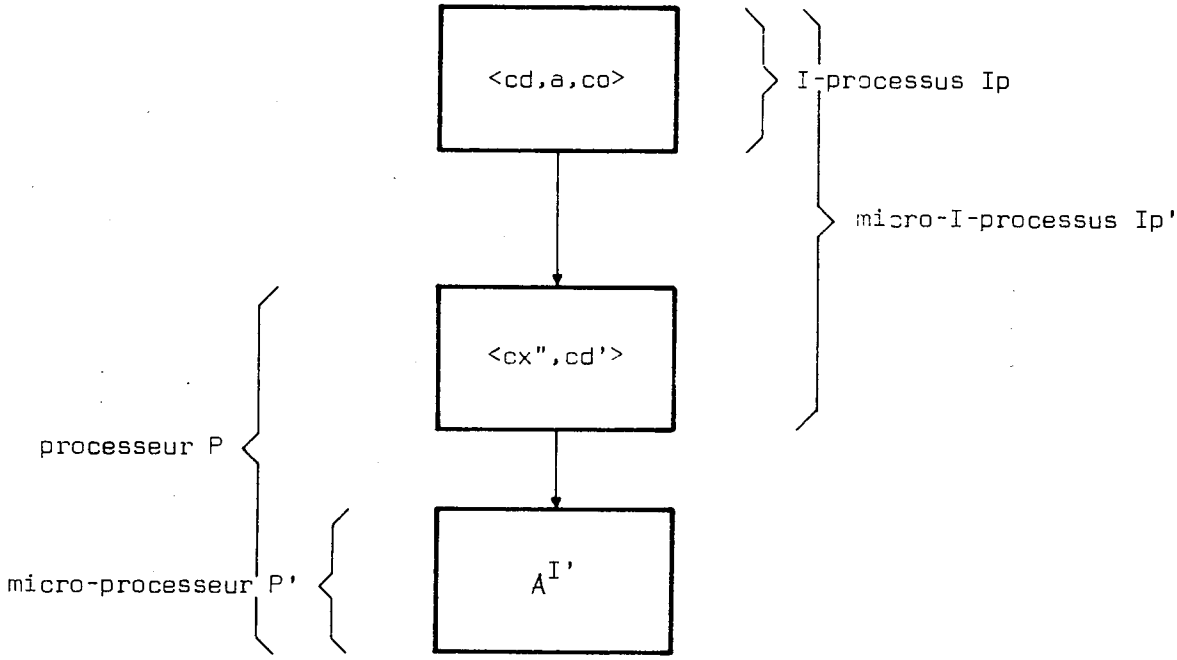
$$MA' = < M', P' >$$

qui représente la machine physique apte à exécuter les micro-instructions. Ses organes de mémorisation sont ceux de la machine programmable augmentés des registres de travail du micro-programme, du compteur de micro-instructions et de la mémoire de contrôle destinée à recevoir le micro-code cd' .

Les entrées/sorties sont réalisées par le micro-processeur.


m'
m
programme
cd
données
a
mémoire centrale
I-processus Ip
cx'
co
compteur ordinal (registre)
cx"
micro-I-processus Ip'
cd'
mémoire de
contrôle
compteur de
micro-instructions
registres de travail
processeur P
e
s
$A^{I'}$
unité physique de traitement
et de séquencement
micro-processeur P'

Le graphe correspondant est:



1/2.9. Temps virtuel d'un I-processus

Le temps virtuel d'un I-processus sera défini de manière discrète. Son écoulement sera directement lié à la vitesse avec laquelle le processeur exécute les instructions. Nous pouvons définir le temps virtuel relatif d'un I-processus comme une variable réelle qui sera incrémentée, à chaque exécution d'une instruction de ce I-processus, par son processeur, d'une quantité déterminée uniquement par la nature de cette instruction et par la valeur des opérandes manipulées

$$TR \leftarrow TR + \Delta t(op_i, a_1, \ldots, a_n)$$

Ce temps virtuel relatif permet d'évaluer l'avance du I-processus indépendamment de la nature exacte et du comportement (ex.: fonctionnement en partage de temps) du processeur avec qui il est associé.

La variable réelle, mesurant l'écoulement du temps virtuel relatif, devra être considérée comme appartenant à cx^{op} du I-processus considéré.

1/2.10 Instants de définition des états d'un I-processus

La définition du temps virtuel pour un I-processus se fait indépendamment de la définition du temps virtuel des I-processus qui le contiennent et qui résultent de la décomposition de son processeur.

En particulier une transition d'état au niveau d'un I-processus peut se décomposer en plusieurs transitions au niveau de ces I-processus. Les instants de définition des états d'un I-processus correspondront donc aux instants auxquels son contexte a atteint les valeurs qui résultent de ses transitions. Entre ces instants, l'état de ce I-processus est supposé non défini.

Ex.: Soit l'opération $C \leftarrow A \times B$

Si cette opération est considérée comme élémentaire vis-à-vis d'un certain I-processus et si la réalisation du processeur qui l'exécute utilise la variable C pour ses calculs intermédiaires, l'état de ce I-processus ne sera considéré comme défini que lorsque cette variable aura atteint la valeur finale correspondant au résultat demandé. Les états intermédiaires de la varaible C ne seront pas considérés comme définissant des états du I-processus

1/3. OPERATEURS

Nous appellerons opérateur une machine séquentielle

$$opr = \langle Q, q_o, E, S, \delta, \gamma, \rangle$$

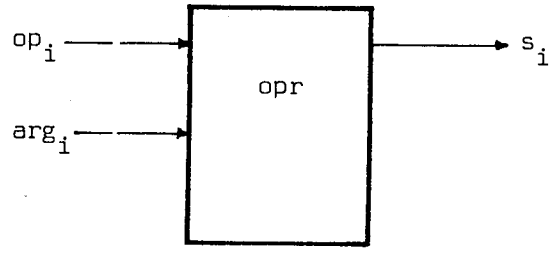
dont le rôle est d'exécuter des opérations

$$op_i \in OP = \{op_1, op_2, \ldots, op_n\}$$

qui lui sont soumises, avec les arguments arg_i nécessaires, par ses voies d'entrées

$$e_i = \langle op_i, arg_i \rangle$$

$$s_j = op_i\ (arg_i)$$



Remarque:
La sortie du résultat s_j de l'opération op_i peut s'effectuer après plusieurs transitions élémentaires de la machine opr .

Exemple d'opérateurs:
Les opérateurs sont en général des organes périphériques d'ordinateurs chargés d'exécuter, de manière autonome, des opérations telles que:

- multiplications,
- transformées de Fourrier,
- convolutions,
- opérations vectorielles,

.....

Ces organes peuvent soit être cablés, soit être réalisés comme des imbrications de I-processus et de processeurs.

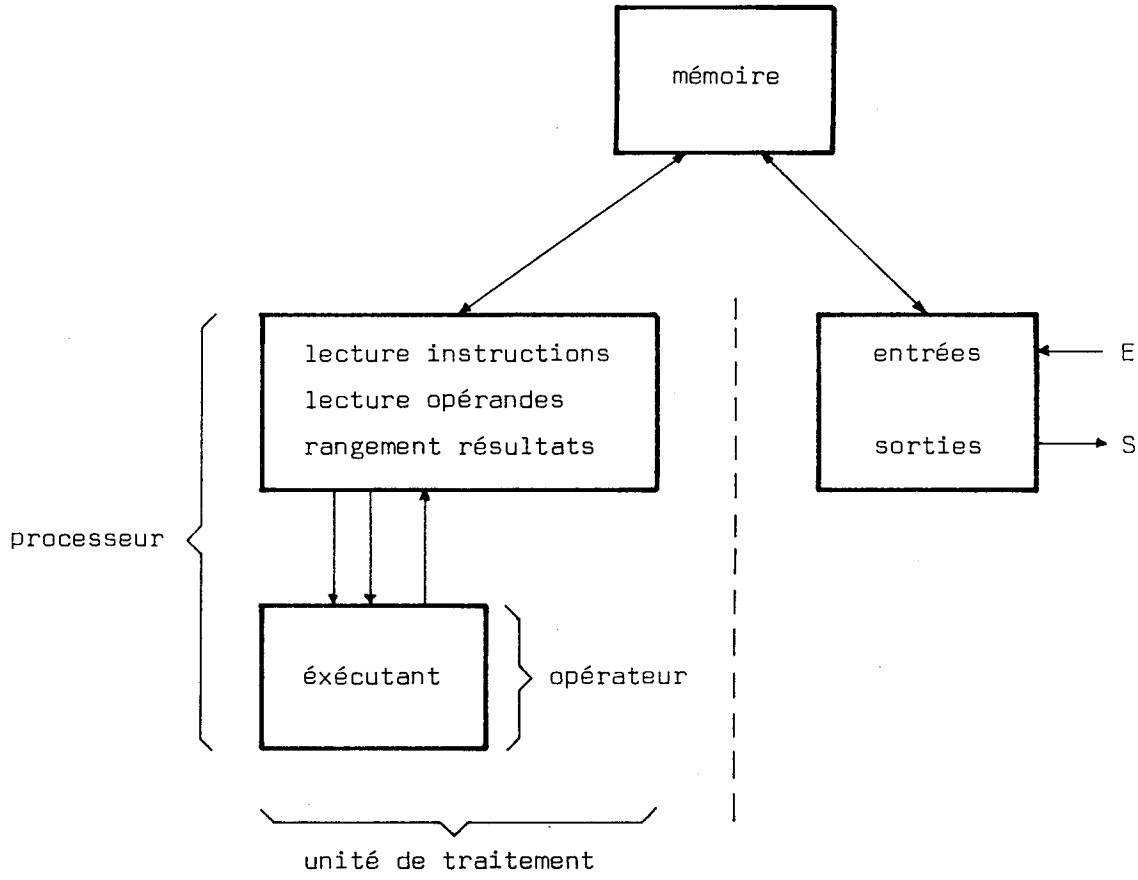
Un opérateur est un processus puisque nous avons identifié cette notion avec celle de machine séquentielle. Il reçoit ses opérations et échange ses données avec d'autres processus qui seront appelés ses utilisateurs.

1/3.1. Relations entre un processeur et un opérateur

Un processeur élémentaire peut être considéré comme réalisé à l'aide d'un opérateur et d'organes chargés:

- de la lecture des instructions en mémoire,

- de la lecture des opérandes en mémoire,
- du rangement des résultats en mémoire,
- des entrées/sorties.



Sous ce point de vue, l'unité de traitement d'un processeur peut être considérée comme un opérateur auquel on communique les opérations à réaliser (instructions op_i de cd), les opérandes et les résultats de ces opératior (cx^{op} et cx') par l'intermédiaire de la mémoire qui lui est associée.

Ce point de vue rejoint celui qui nous a fait comparer les mécanismes de

communication avec un processus, soit par ses lignes d'entrées/sorties, soit par une zône momentanément non significative de son contexte [§ 1/2.3.] .

Nous pouvons définir des êtres hybrides entre les processeurs et les opérateurs suivant qu'ils vont:

- chercher leurs instructions dans une mémoire, ou les recevoir de l'extérieur,
- lire leur opérandes et écrire leurs résultats dans une mémoire, ou les échanger avec l'extérieur.

1/4. LIENS AVEC LA NOTION DE RESSOURCE

Nous allons utiliser les notions de ressource et d'utilisateur, introduites en [§ Introduction 3/2] pour y confondre les notions de processeur et de I-processus d'une part, d'opérateur et de processus utilisateur d'autre part.

- Un opérateur sera une ressource sollicitée par un processus utilisateur.
- Un processus sera considéré comme la ressource du I-processus (utilisateur) qu'il exécute. Ce I-processus sera dit "solliciter" à son processeur l'exécution des instructions qui le composent par le simple fait qu'elles sont rangées dans un certain ordre, sous un certain codage et à une certaine place dans la mémoire associée au processeur.

Nous remarquons qu'un processus peut solliciter le service de plusieurs opérateurs au même instant mais qu'un I-processus ne peut solliciter l'usage que d'un seul processeur à la fois.

1/4.1. Graphe de ressources

Nous allons chercher à représenter sous la forme d'un graphe les structur de I-processus et de processeurs contenant des opérateurs.

$$GR = < AR, DR >$$

dans lequel

$$AR = AI^0 \cup AI^1 \cup \ldots\ldots \cup AI^n \cup AO$$

où

AI^0 est l'ensemble d'articulations associé au graphe GI^0 du processus principal considéré, à l'exclusion des opérateurs.

AI^i est l'ensemble d'articulations associé au graphe GI^i associé à l'opérateur i supposé décomposé en une chaîne de I-processus/processeurs.

AO est l'ensemble d'opérateurs considérés comme élémentaires, c'est-à-dire supposés non décomposés.

$$DR = DI^0 \cup DI^1 \cup \ldots\ldots \cup DI^n \cup DO$$

où

DI^0 est la relation de définition de GI^0

DI^i est la relation de définition de GI^i

DO est telle que

$$DO\ (a_i, a_j)$$

signifie que

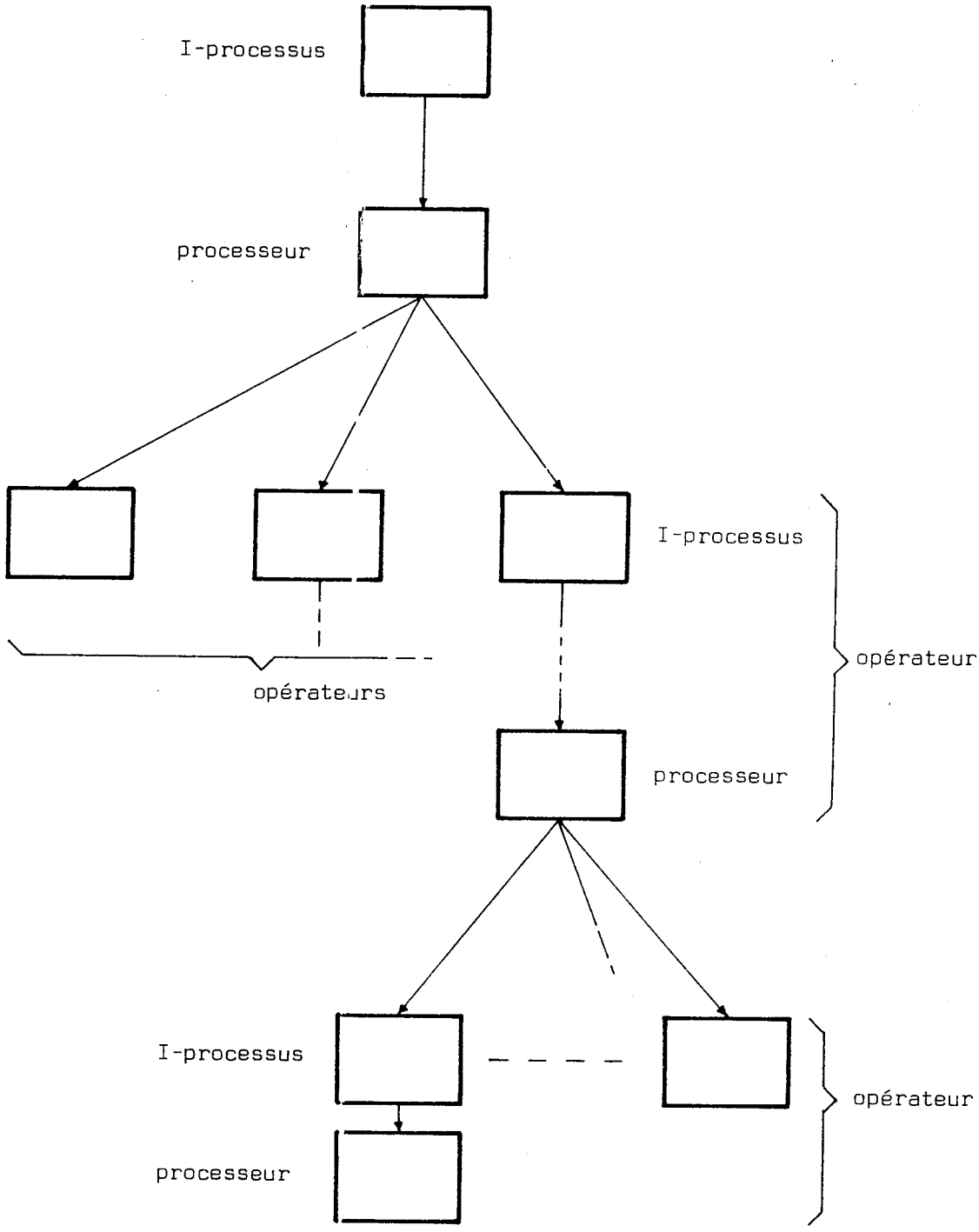
a_i est un processeur élémentaire

a_j est soit:

- un opérateur élémentaire de a_i
- un I-processus élémentaire d'un opérateur décomposé de

Le graphe est arborescent car, pour une articulation a_i donnée, il peut y avoir plusieurs articulations a_j telles que $DO(a_i, a_j)$.

exemple:


I-processus
processeur
I-processus
opérateurs
opérateur
processeur
I-processus
opérateur
processeur

1/4.2. Evolution du temps dans une chaîne de ressources

La vitesse d'évolution des contextes de chaque niveau, d'un tel assemblage, est très variable. En particulier, dans le cas d'une chaîne de I-processus, la fréquence des variations des contextes croît lorsqu'on descend vers le processeur (classiquement par un facteur 10 à chaque niveau). Ceci est dû au fait que chaque transition élémentaire d'un I-processus se décompose en plusieurs transitions du I-processus immédiatement inférieur (interprétation). Ce phénomène est moins sensible, voire inversé, lorsqu'on examine le cas des opérateurs. En effet, souvent, ceux-ci ne sont utilisés que pour l'exécution de certaines opérations du processus utilisateur et leur fréquence de sollicitation est très inférieure à la vitesse avec laquelle ce processus utilisateur évolue.

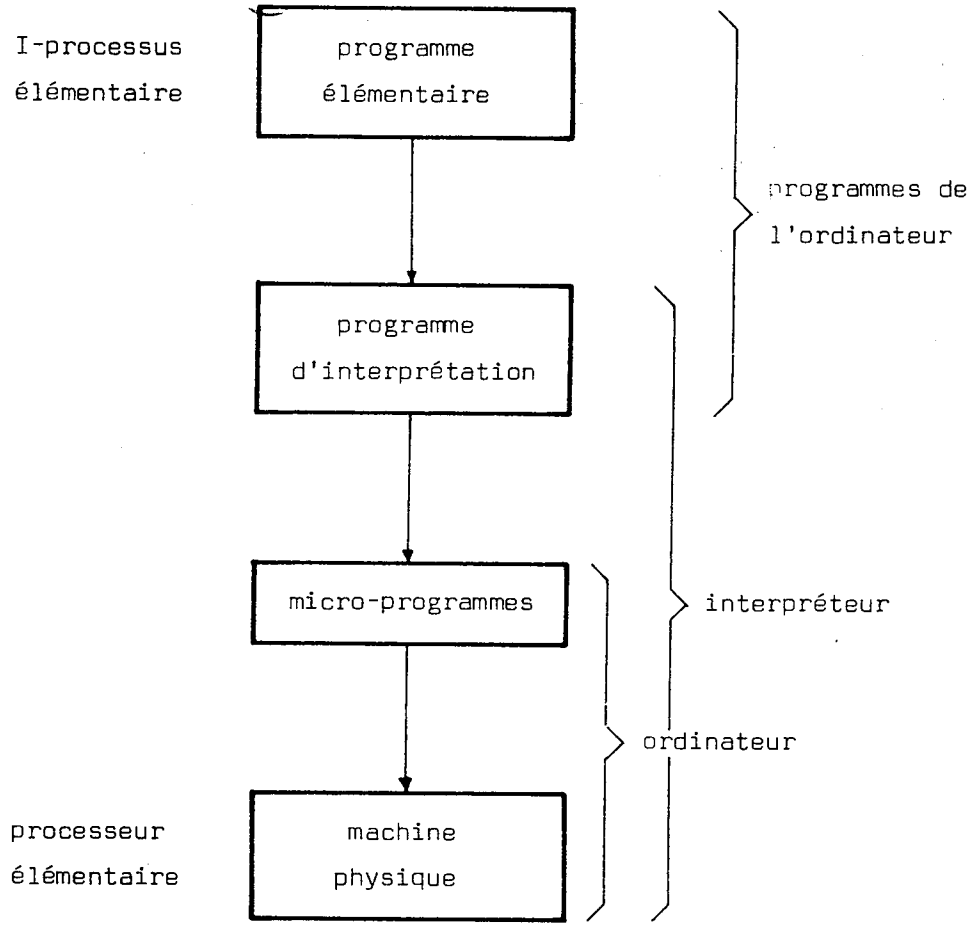
1/4.3. Représentation d'une ressource, ou d'un utilisateur, par une articulation

- Nous dirons qu'une articulation représente une ressource si celle-ci est l'articulation "supérieure" (au sens du graphisme adopté pour la représentation des graphes GI et GR) de ce processeur ou de cet opérateur. Nous confondrons souvent, pour plus de commodité, la ressource étudiée et l'articulation qui la représente. Cette façon de nommer une ressource met en évidence l'indépendance qui existe entre sa structure fonctionnelle, décrite par l'articulation qui la représente, et sa réalisation décrite par les autres articulations qui la constituent.

- Nous dirons de même qu'une articulation représente un utilisateur (de ressources) si celle-ci est l'articulation "inférieure" (au sens du graphisme adopté pour la représentation du graphe GI) de ce I-processus ou de ce processus. Nous confondrons souvent, pour plus de commodité, l'utilisateur et l'articulation qui le représente. Cette façon de nommer un utilisateur met en évidence l'indépendance qui existe entre sa structure fonctionnelle, telle qu'elle est vue par ses ressources, et les données qu'il manipule décrites par les autres articulations qui le composent.

Exemple:

Considérons le graphe GI suivant:



Nous confondrons la ressource interpréteur avec le programme d'interprétation qui la représente et le I-processus utilisateur de l'ordinateur avec ce même programme d'interprétation.

Remarque:

Pour éviter toute confusion nous parlerons du programme d'interprétation soit comme une ressource (un processeur), soit comme un programme utilisant l'ordinateur.

1/5. RELATIONS AVEC LES NOTIONS HABITUELLES

La notion habituelle de processus, entité formelle correspondant à l'exécution d'un code par un processeur, est telle qu'elle correspond également à l'exécution des séquences plus élémentaires (micro-instructions) associées à l'exécution de chaque instruction du code considéré. Le raisonnement peut se poursuivre aux niveaux plus élémentaires, par exemple aux phases constituant chaque micro-instruction.

Ceci montre une intrusion évidente du processus dans ce qu'il est convenu d'appeler le processeur. Cette intrusion a pour conséquence:

- de rendre imprécise la séparation entre le processus et le processeur,
- de rendre la définition du processus dépendante de la nature des mécanismes internes utilisés pour l'exécution. En effet, si l'on admet qu'un processus est purement séquentiel, certaines techniques d'exécution avec recouvrement des instructions risquent d'amener des contradictions insurmontables dans ces définitions.

Nous opposons à cette notion, un assemblage d'objets bien définis et bien séparés (I-processus, processeur), l'un soumettant au second l'exécution des instructions qui le constituent. La nature du processeur n'influe donc plus sur celle du I-processus. Le mécanisme d'exécution est clair dans la mesure où l'on admet que le I-processus "soumet" ses instructions à l'exécution du processeur par le simple fait qu'elles sont rangées d'une certaine manière et sous un certain codage dans la mémoire de ce dernier.

2 - SYSTEMES HIERARCHISES

2/1. Formalisme des systèmes hiérarchisés

Nous allons maintenant introduire le formalisme qui nous servira à exprimer la structure des systèmes et machines informatiques.

Notre but sera de présenter de telles structures sous la forme de réseaux orientés d'articulations.

2/2. Distribution de l'activité d'un processeur

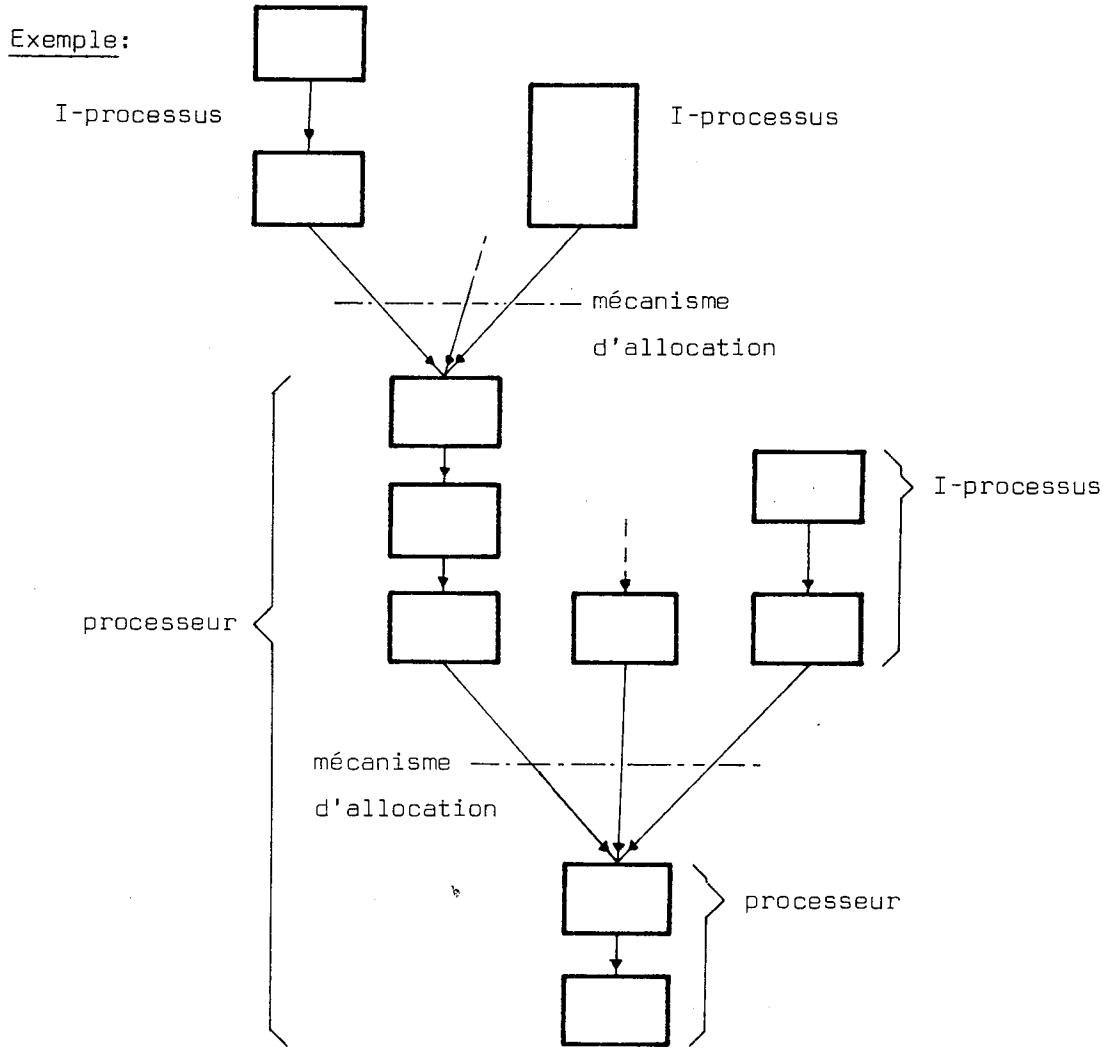
La répartition de l'activité d'un processeur entre plusieurs I-processus candidats représente une facilité fréquemment utilisée pour rentabiliser son coût(*).

Une telle facilité est obtenue au prix de l'adjonction d'un mécanisme d'allocation à ce processeur. Ce mécanisme a pour but de gérer l'attribution des services offerts par ce processeur aux différents I-processus qui peuvent progresser grâce à lui.

Cette facilité peut être accordée à chaque niveau de processeurs imbriqués.

Ce raisonnement peut être évidemment étendu au cas de la répartition d'un opérateur entre plusieurs processus utilisateurs.

(*) Le compromis économique évoqué peut s'exprimer de la manière suivante, en première approximation:
Le coût de n processeurs de vitesse v (et de taux d'utilisation k_i, $0 \leq k_i \leq 1$) doit être comparé au coût de 1 processeur de vitesse $\sum_i k_i.v$ augmenté du coût du mécanisme d'allocation.



Ces raisonnements nous amènent à associer un mécanisme d'allocation à chaque articulation du graphe GR entrant dans la constitution d'un processeur ou d'un opérateur. Ce mécanisme sera distinct ou confondu avec l'algorithme d'interprétation de ce processeur ou de cet opérateur. Il sera réalisé, soit à l'aide d'opérateurs, soit sera lui-même décomposé en un assemblage de I-processus et de processeurs imbriqués.

Cette possibilité d'allocation des processeurs et des opérateurs à différents utilisateurs nous amène à généraliser la notion de graphe de ressources.

2/3. Systèmes hiérarchisés

Nous appellerons systèmes hiérarchisés cette troisième famille de graphes

GSH =< ASH, DSH >

Nous allons, dans un premier temps, définir ce que nous entendrons par articulation d'un tel graphe.

2/3.1 Articulations

Nous appellerons articulations $a_i \in$ ASH d'un système hiérarchisé GSH un être défini comme suit:

articulation =< requêtes, mécanismes, ressources >

dans lequel:

requêtes =< ensemble des requêtes d'allocation, ensemble des requêtes d'utilisation >

- les requêtes d'allocation sont utilisées pour solliciter l'usage de la ressource (processeur ou opérateur) représentée par cette articulation.

- les requêtes d'utilisation correspondent aux opérations dont on peut soumettre l'exécution à cette ressource une fois qu'elle a été allouée.

mécanismes = < mécanismes d'allocation, mécanisme d'interprétation

- le mécanisme d'allocation, unique par articulation, est chargé d'allouer cette articulation en fonction des requêtes d'allocation qu'elle reçoit. Ce mécanisme peut être:
  - soit confondu avec le mécanisme d'inteprétation,
  - soit distinct et être:
    - soit un processus élémentaire,
    - soit un I-processus.
  - soit inexistant si l'articulation n'a pas d'utilisateurs,
  - soit trivial si elle n'en a qu'un.

- Le mécanisme d'interprétation est chargé d'exécuter les requêtes d'utilisation soumises à la ressource représentée par cette articul tion. Il peut être simple ou multiplexeur.

. Un mécanisme d'interprétation simple signifie qu'il est unique pour cette articulation qui ne peut donc qu'être allouée à un seul utilisateur à un instant donné. Ce mécanisme permet c exécuter une seule suite de requêtes émanant de cet utilisateu Dans ce cas ce mécanisme peut être:

- soit un processeur élémentaire,
- soit une étape d'interprétation,
- soit un opérateur élémentaire.

Dans le cas où l'articulation étudiée n'a pas d'utilisateur, c mécanisme peut être:

- soit un processus élémentaire,
- soit un I-processus élémentaire.

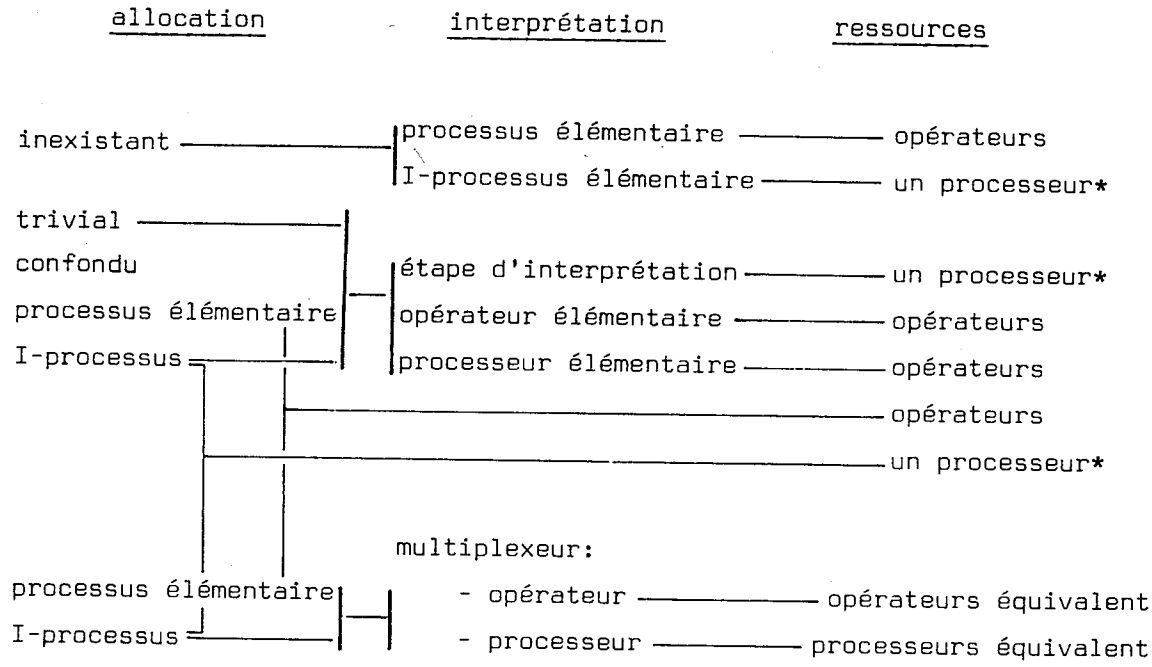
. Un mécanisme d'interprétation multiplexeur signifie que l' l'articulation considérée peut être allouée simultanément à plusieurs utilisateurs. Elle représente un ensemble de ressources. Son mécanisme d'interprétation a pour simple rôle d' aiguiller les diverses requêtes émanant des utilisateurs vers des articulations représentant ces ressources. Elle sera alor appelée une articulation multiple.

ressources =< ensemble d'articulations représentant des ressources

- Les mécanismes d'allocation et d'interprétation peuvent utiliser des ressources représentées par des articulations. Ces ressources peuvent être:

. soit des processeurs,
. soit des opérateurs.

Les régles d'assemblage des diverses réalisations des mécanismes d'alloca tion et d'interprétation sont données par le tableau suivant donnant également la nature des ressources qu'ils peuvent utiliser.



Exemple:

La notion d'articulation peut se représenter sous la forme d'une structure particulière dans un langage algorithmique.

Articulation UNITE DE TRAITEMENT

requêtes

d'allocation SOLLICITATION, LIBERATION

d'utilisation I_1, I_2, ..., I_n "instructions"

mécanismes

d'allocation

"procédure d'allocation de l'unité de traitement"

d'utilisation

"procédure d'interprétation des instructions"

ressources MEMOIRE, CANAL,

* Les mécanismes d'interprétation et d'allocation ne peuvent chacun utiliser qu'un processeur au plus.

Remarques:

- Dans la suite le mot: articulation, sous-entendra toujours: articulation d'un système hiérarchisé.

- La démarche qui nous a amenés à définir le concept d'articulation d'un système hiérarchisé a de nombreuses analogies avec celle qui a conduit à la définition du concept de MONITEUR [41, 17]. Toutefois, la principale différence entre une articulation et un moniteur réside dans le fait que les mécanismes d'un moniteur ne sont que des procédures qui s'exécutent dans le processus utilisateur alors qu'une articulation peut avoir un fonctionnement autonome, parallèle à celui de ses utilisateurs, avec qui elle dialogue par un échange de messages. Une articulation peut également décider, de manière autonome, de son allocation à ses divers candidats, puis la modifier de manière impromptue (lorsqu'elle représente une ressource requérable).

2/3.2. Relation de définition

Pour constituer un système hiérarchisé nous assemblerons les articulations en un graphe orienté GSH par la relation de définition suivante:
soit a_i, a_j ε ASH DSH(a_i, a_j) signifie que l'articulation a_i peut utiliser la ressource représentée par l'articulation a_j.

Remarque:
D'autres choix auraient été possibles, par exemple considérer la relation correspondant à l'allocation des ressources à leurs utilisateurs.

La relation choisie présente l'intérêt de représenter la structure potentielle des dialogues entre les articulations qui préfigurent quelquefois la structure de la machine étudiée.

2/3.3. Structure hiérarchique

La relation de définition DSH donne au graphe GSH la structure d'une hiérarchie [48] .

En effet:

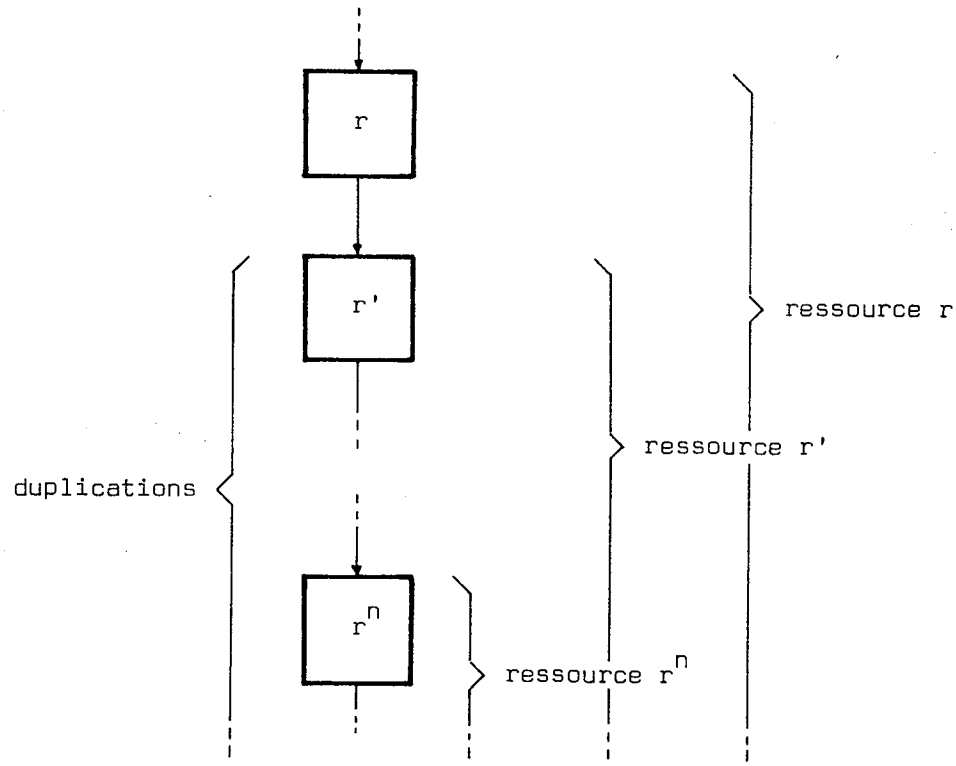
- Il est orienté par définition,
- Il est sans circuit. En effet, nous imposerons la restriction qu'une articulation ne puisse jamais s'utiliser elle-même en tant que ressource, directement ou indirectement.

Cette propriété nous permet de considérer le graphe GSH comme un ensemble ordonné. En effet, sous ces conditions, nous pouvons définir une relation d'ordre sur ASH:

$$DSH\ (a_i, a_j) \Rightarrow a_i > a_j$$

Remarque:

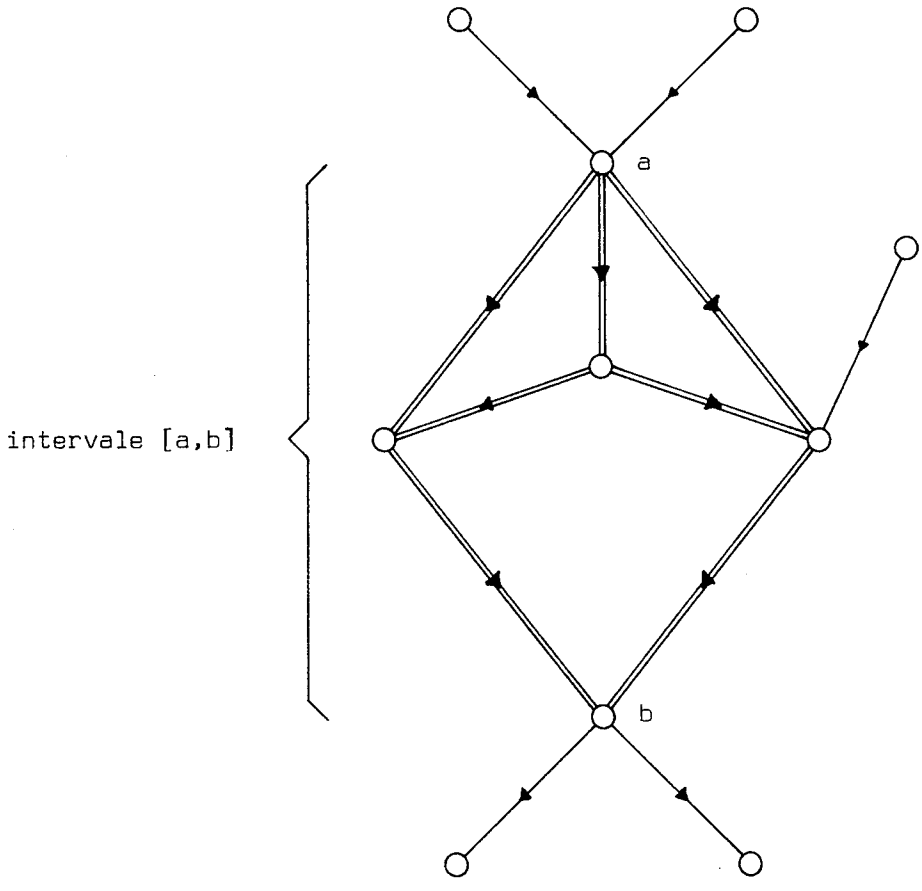
L'utilisation de ce formalisme à la représentation des structures programmées contenant des utilisations récursives de ressources ne peut se faire qu'en supposant ces ressources dupliquées en autant d'exemplaires qu'il est nécessaire pour remplacer chaque récursion par une utilisation d'une ressource pré-existante.



2/4. INTERVALLES DANS UNE HIERARCHIE

Soit deux éléments a et b d'une hiérarchie tels que $a \geq b$
Nous appellerons intervalle [a, b] le sous-graphe suivant:

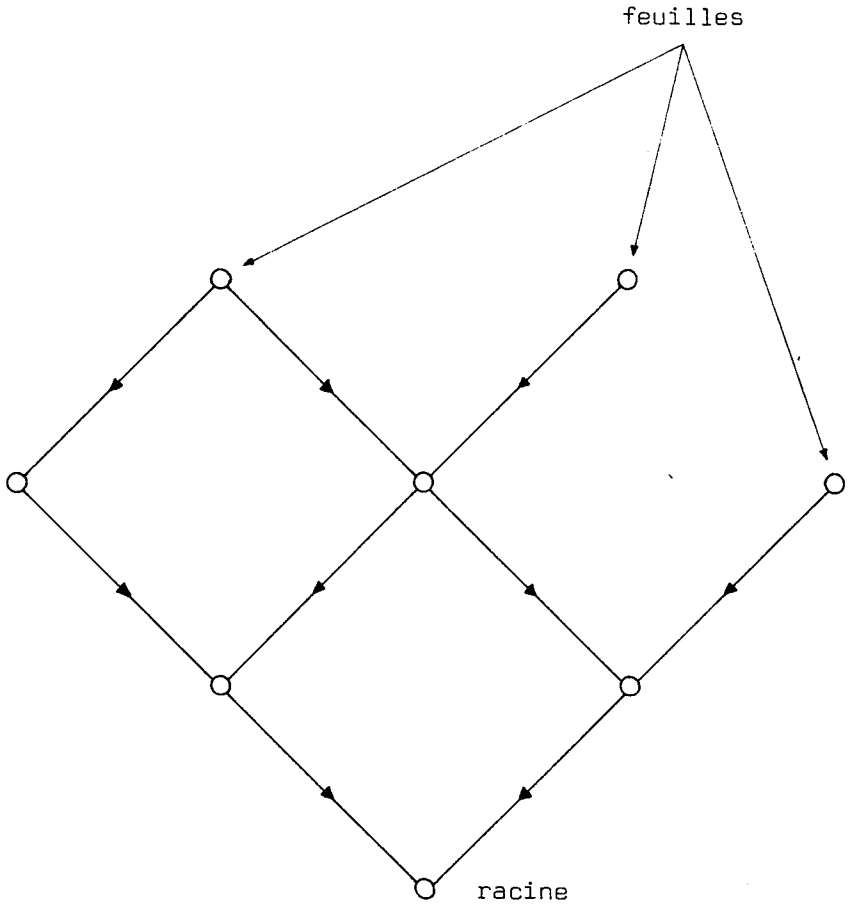
$$[a, b] = \{\sigma \in H \mid \sigma \leq a \text{ et } \sigma \geq b \}$$



2/5. HIERARCHIES A ELEMENT INFIMUM

Pour simplifier les raisonnements ultérieurs, nous supposerons que les hiérarchies que nous étudierons possèdent un élément infimum.

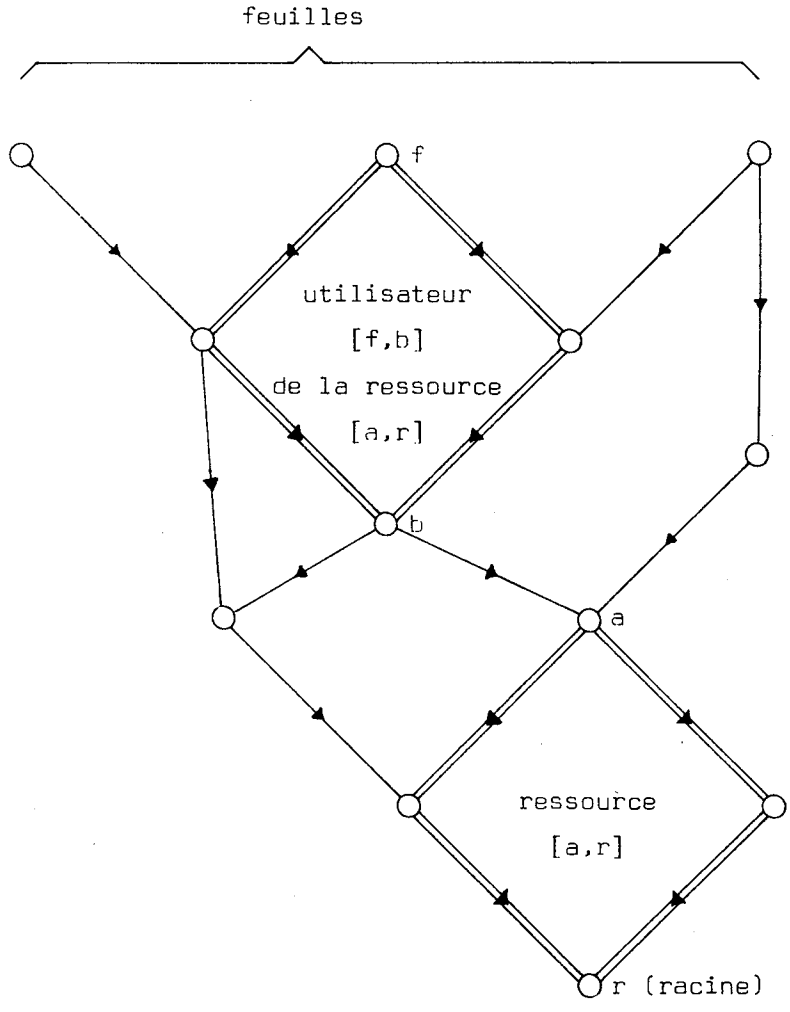
Nous appellerons racine cet élément infimum. Les éléments maximaux de ces hiérarchies seront appelés des feuilles.



2/6. NOTION DE RESSOURCE DANS UN SYSTEME HIERARCHISE

Nous appellerons ressource dans un système hiérarchisé un intervalle [a, r] dans lequel r est la racine de la hiérarchie.

Un utilisateur de cette ressource sera [f, b] dans lequel f est une feuille de la hiérarchie et b une articulation qui couvre a .



2/7. NIVEAUX

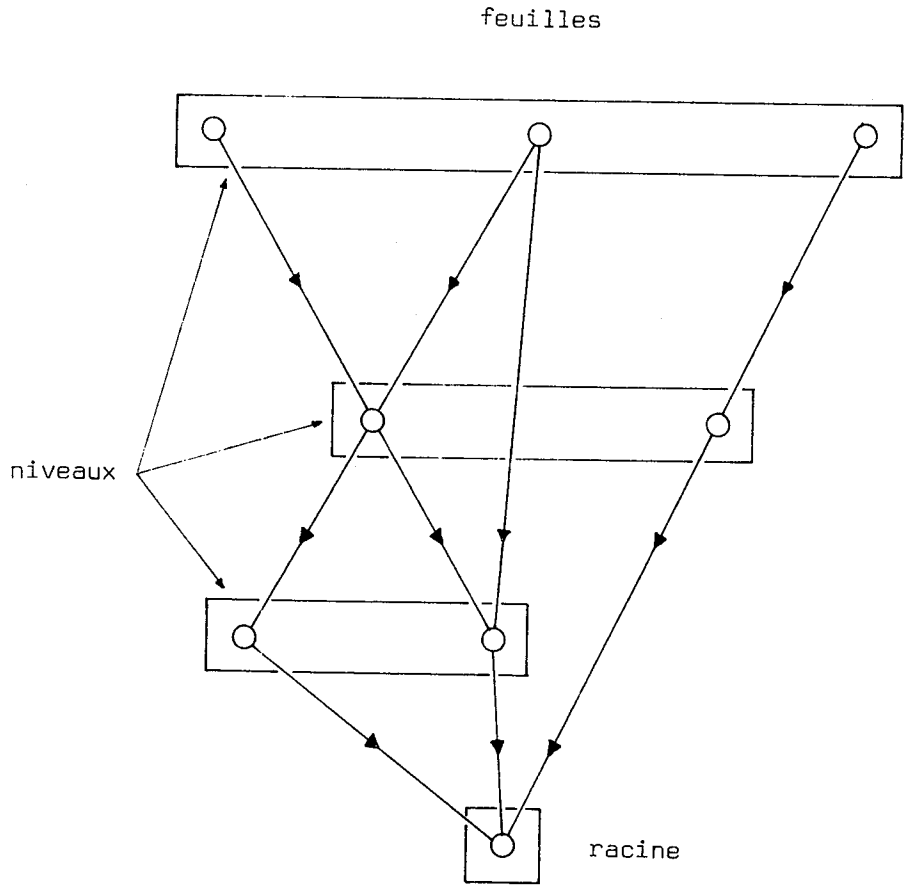
Nous pouvons introduire une notion de niveaux, dans un système hiérarchisé, à l'aide d'une relation d'équivalence NSH sur l'ensemble ASH des articulations. Une telle relation doit posséder les propriétés suivantes:

- NSH (a_i, a_j) entraîne que les articulations a_i et a_j ne sont pas comparables au sens de la relation d'ordre issue de DSH.
- le graphe quotient GSH/NSH est une hiérarchie.

Sous ces consitions, les classes d'équivalence de la relation NSH constituent les niveaux.

ex.: Les relations suivantes peuvent servir à définir des niveaux:

- la racine constitue un niveau, deux articulations non comparables sont au même niveau si leurs distances maximales à la racine sont égales.
- les feuilles constituent un niveau, deux articulations non comparables sont au même niveau si leurs distances maximales à cette classe sont égales



Le but de cette relation est de représenter les analogies dans la technique de réalisation des articulations d'un système hiérarchisé.

ex.:
Nous pouvons distinguer le niveau des articulations microprogrammées dans la représentation d'un ordinateur en tant que système hiérarchisé.

2/8. ETAT D'UNE ARTICULATION

L'état d'une articulation se décompose en:

- l'état de son mécanisme d'allocation,
- l'état de son mécanisme d'interprétation. Nous admettrons qu'un mécanisme d'interprétation multiplexeur n'a pas d'état propre.

Suivant la réalisation de ces mécanismes, la progression de ces états est sous le contrôle:

- de l'articulation elle-même, cas des processeurs ou processus élémentaires,
- de ses ressources, cas des I-processus et des étapes d'interprétatio

2/9. MODIFICATION DE L'ALLOCATION D'UNE ARTICULATION

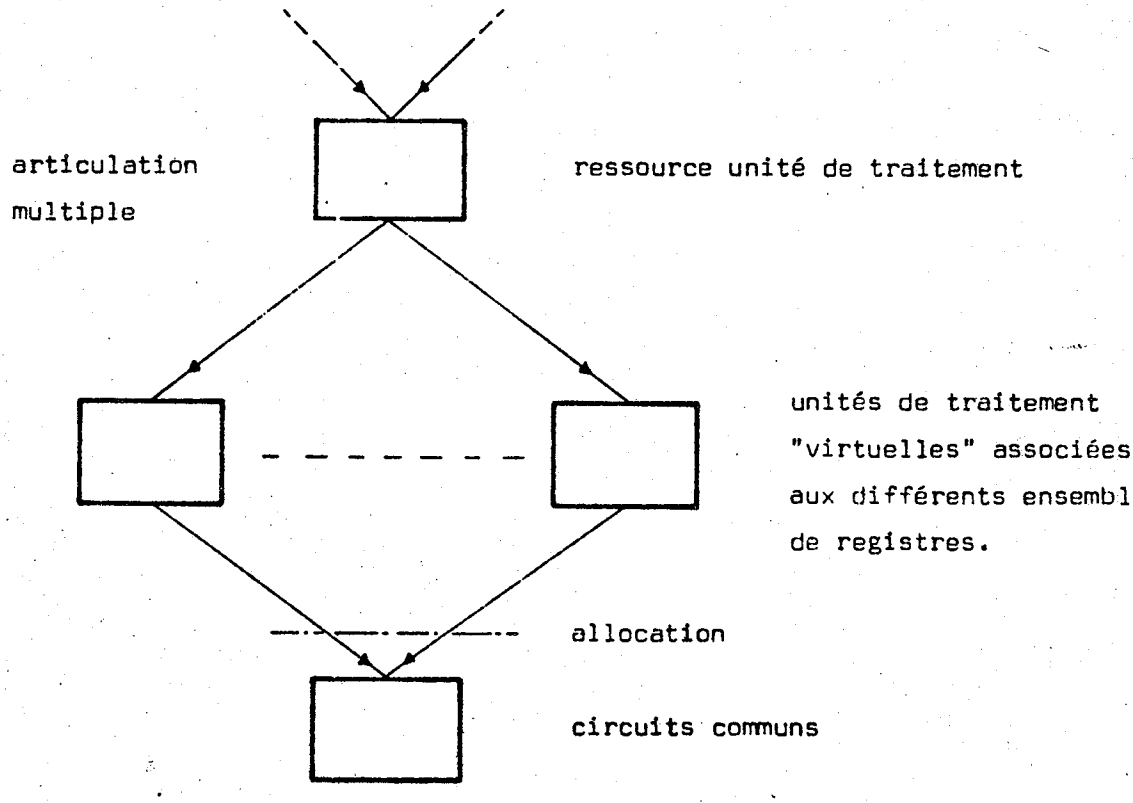
Une articulation possédant un mécanisme d'interprétation simple peut s'allouer successivement à divers utilisateurs. Pour ce faire, l'état de son mécanisme d'interprétation devra être sans signification au moment de la commutation. Pour ce faire nous pourrons:

- ne permettre la commutation qu'en certains instants pour lesquels cette condition est remplie (points interruptibles).
  ex.: une interruption n'est en général prise en compte qu'entre l'exécution de deux instructions dans une machine classique.

- rendre non significatif l'état du mécanisme d'interprétation en

abandonnant l'exécution de l'opération en cours. Cet abandon doit avoir lieu avant que l'état de l'utilisateur de cette articulation ait été modifié. La reprise du traitement pour cet utilisateur se ra en réexécutant l'opération interrompue.
ex.: Dans une machine "pipe-line" une interruption peut annuler le début d'exécution des instructions situées dans les premiers étages la chaîne de traitement.

- décomposer l'articulation en un ensemble d'articulations équivalent particularisées au traitement d'un utilisateur par le contexte résic qu'elles contiennent. Ces articulations sont regroupées à l'aide d une articulation multiple représentant la ressource processeur partageable.
ex.: L'unité de traitement de certains ordinateurs (CII 10070) pos plusieurs ensembles de registres associés aux différents utilisateu qui peuvent la solliciter à un instant donné.



2/10. CAS DES MEMOIRES

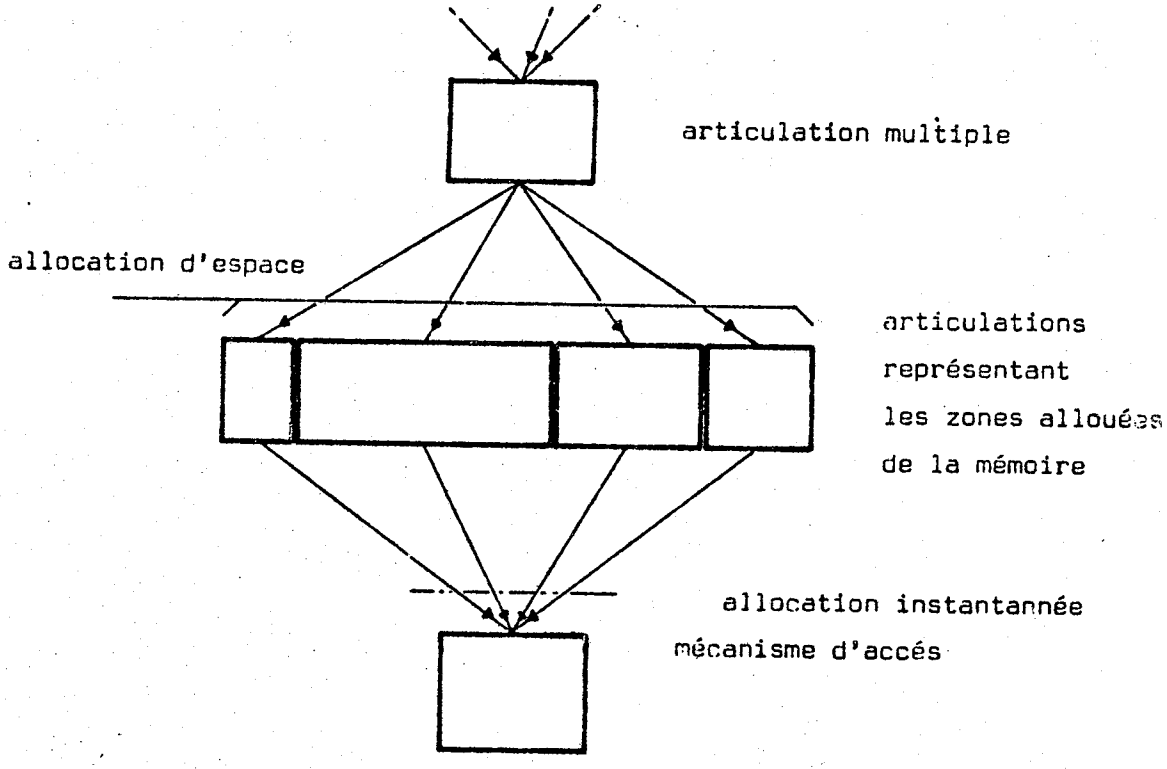
Nous aimerions pouvoir considérer la mémoire d'un ordinateur comme un opérateur du processeur élémentaire de la machine qui l'utilise.

Ce point de vue soulève la difficulté suivante: Le contenu de la mémoire constitue d'une part, le codage du ou des I-processus exécutés par le processeur de cette machine et, d'autre part, l'état de cet opérateur mémoire.

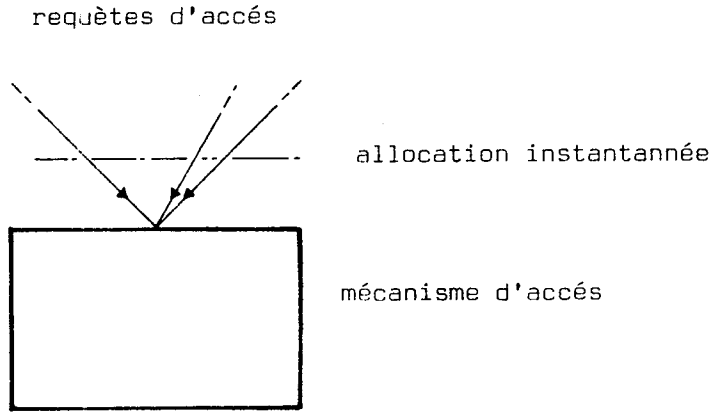
Ce dilemne peut être levé,

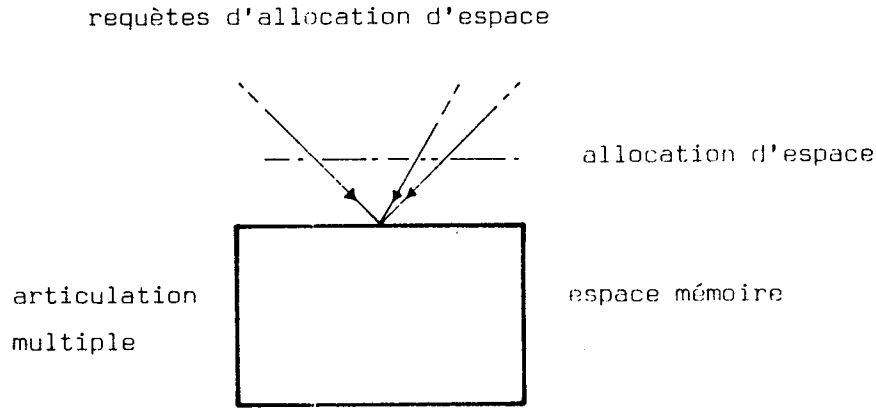
- soit en admettant:
  - qu'une mémoire doit être décomposée en plusieurs ressources correspondant aux zônes mémoires allouées. Ces ressources ne doivent pas être obligées d'utiliser elles-mêmes une ressource unique correspondant à l'ensemble de la mémoire physique.
  - les ressources correspondant aux zônes de mémoire allouées ne sont pas concernées par le sens de leur contenu.

Sous ces hypothèses, l'opérateur mémoire se présentera donc de la manière suivante:



- soit en ne considérant la mémoire que du point de vue de ses allocations en la décomposant en deux articulations distinctes.





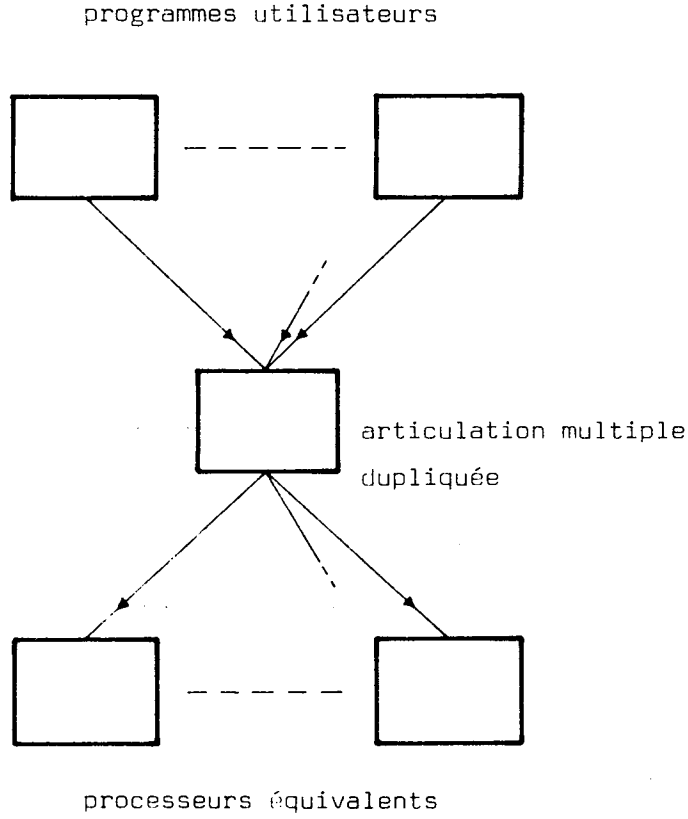
La première de ces articulations gère les conflits d'accés au niveau des opérations de lecture et d'écriture dans l'organe physique, la seconde le partage de l'espace disponible. Ces deux représentations ne se situent do pas au même niveau de formalisme et souvent nous ne considérons que l'une d'entre elles.

2/11. ARTICULATIONS ET RESSOURCES PARTICULIERES

2/11.1. Articulations multiples dupliquées

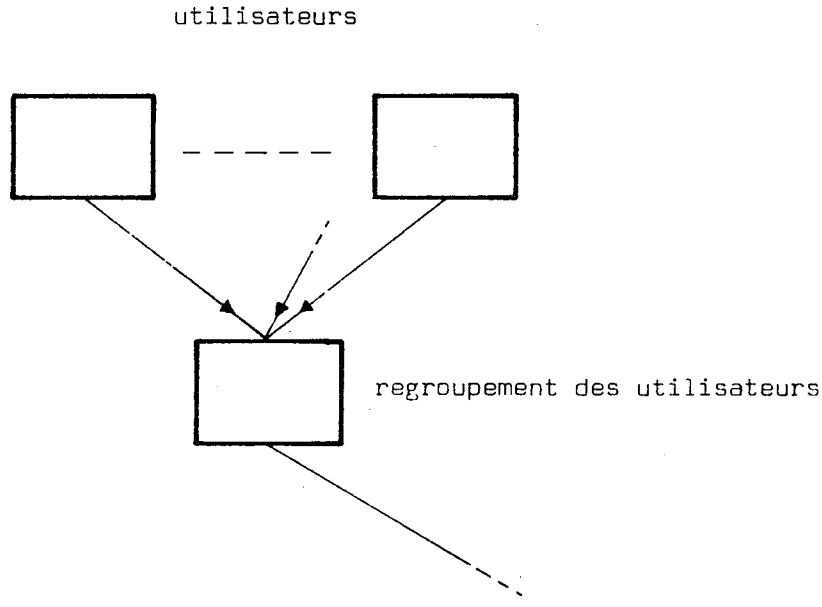
Une articulation multiple sera dite dupliquée si elle regroupe un ensemble de ressources équivalentes. Le rôle de son mécanisme d'allocation sera de choisir l'une de ces ressources équivalentes disponible pour chercher à satisfaire les demandes d'allocation des utilisateurs.

ex.:
Dans une configuration multiprocesseurs, les programmes solliciteront la ressource globale processeur qui sera représentée par une articulation multiple dupliquée regroupant les processeurs équivalents.



2/11.2. Articulations multiples dégénérées

Une articulation multiple sera dite dégénérée si elle représente un ensemble de ressources vides, c'est-à-dire incapables d'offrir un service quelconque. Elle servira à rassembler des utilisateurs devant être gérés de manière semblable [§ annexe 1].



Une telle articulation ne recevra que des requêtes d'allocation. La racine d'un système hiérarchisé sera souvent représentée par une telle articulation.

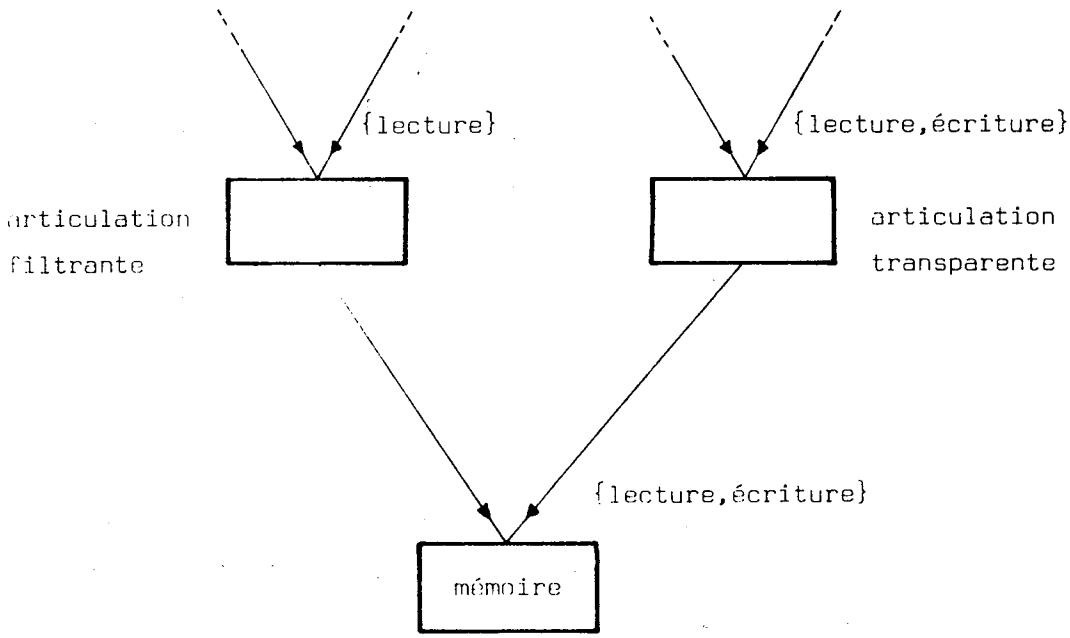
Remarque:

Il est possible d'admettre qu'une telle articulation puisse s'allouer à un nombre non borné à priori d'utilisateurs.

2/11.3. Articulations transparentes et filtrantes

Une articulation sera dite transparente lorsque son mécanisme d'interprétation se résumera à la simple transmission, à une ressource plus élémentaire, de toutes les requêtes d'utilisation reçues par cette articulation. Une telle articulation sera dite filtrante lorsque l'ensemble des requêtes d'utilisation qu'elle admet est inclu, au sens strict, dans celui admis par la ressource plus élémentaire utilisée.

ex.:



2/11.4. Ressource privée

Nous appellerons ressource privée (d'un utilisateur donné) une ressource ne pouvant être sollicitée que par cet utilisateur.

2/11.5. Ressource virtuelle

Nous appellerons ressource virtuelle une ressource représentée par une articulation transparente ou filtrante.

2/11.6. Ressource principale

Nous appellerons ressource principale (d'un utilisateur donné) une ressou ce indispensable à tout fonctionnement de cet utilisateur. Il en découle immédiatement qu'une telle ressource ne peut être sollicitée explicitemer par cet utilisateur.

ex.: Un processeur est une ressource principale pour un I-processus. En effet, il doit interpréter toutes les opérations qui constituent ce I-processus. Ceci implique que ce I-processus ne contient pas d'instructions pour solliciter l'usage de ce processeur.

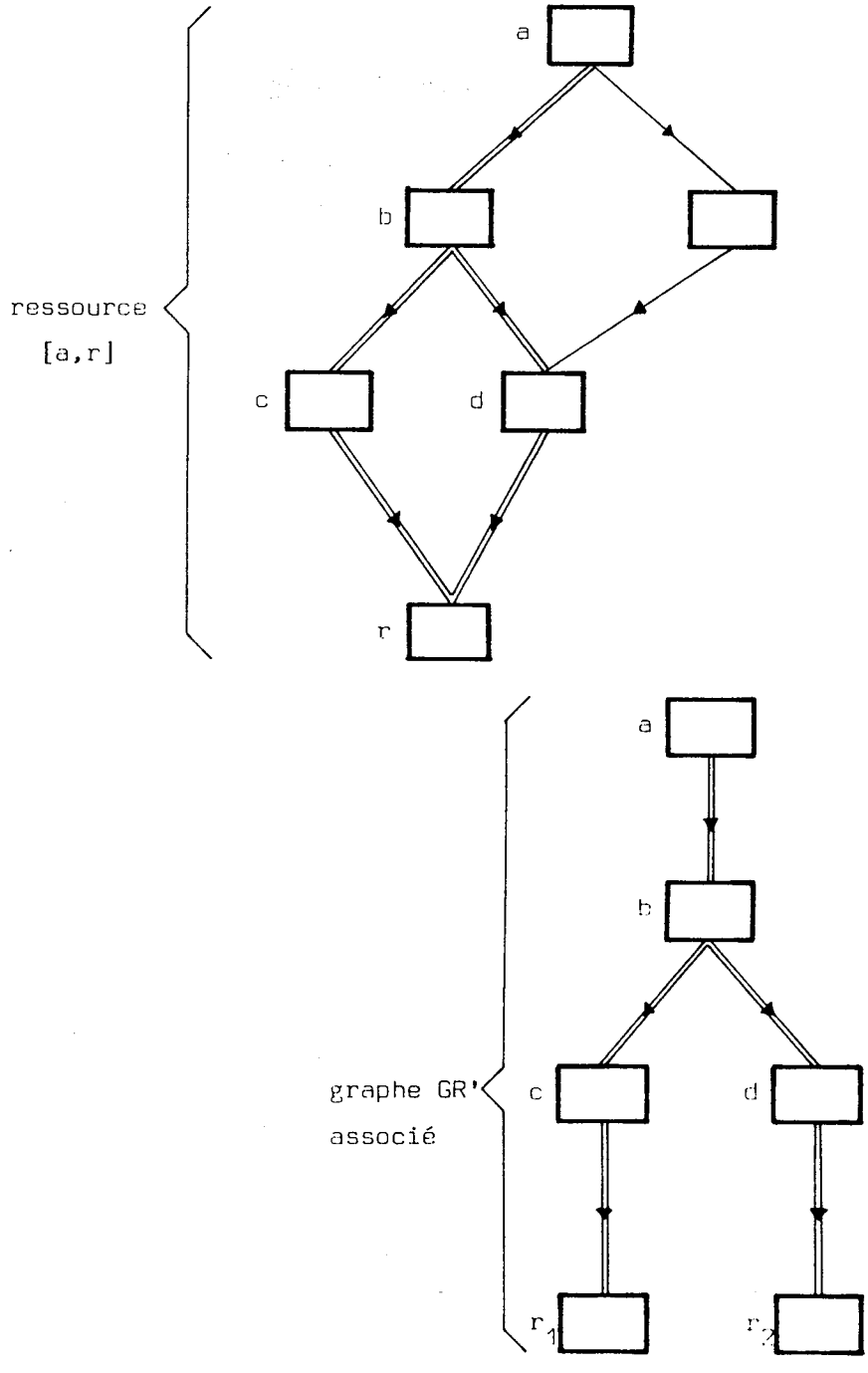
2/12. BI-VALENCE DE LA NOTION DE RESSOURCE

D'après les définitions précédentes, une ressource, dans un système hiéra chisé, est un être bi-valent caractérisé par 2 mécanismes indépendants, étudiés séparément:

- un mécanisme d'allocation,
- un mécanisme d'interprétation.

- La représentation des ressources sous l'unique point de vue des mécanis mes d'allocation peut se faire à l'aide du graphe GSH lui-même, dans la mesure où l'on ne tient compte que des mécanismes d'allocation dans la définition des articulations.

- La représentation des ressources sous l'unique point de vue des mécanis mes d'interprétation peut se faire, à chaque instant, donc pour un cas précis d'allocation, par l'étude du graphe de ressources GR' associé à cet instant à une ressource [a, r].



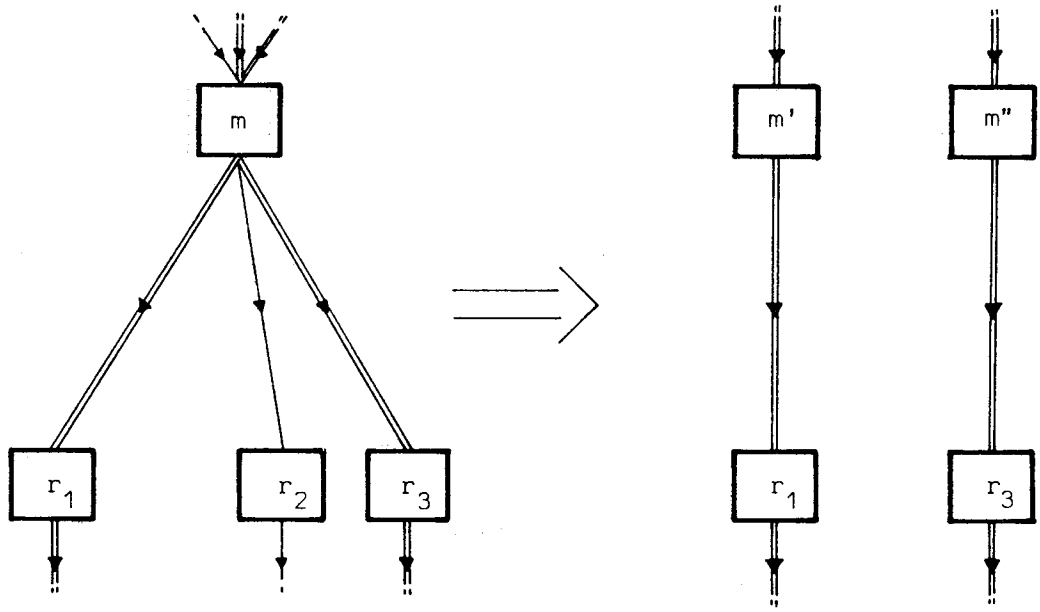
Remarque: Dans un fonctionnement correct d'un système hiérarchisé, seules les articulations multiples peuvent être amenées, à un instant donné, à servir simultanément plusieurs utilisateurs.

La construction du graphe de ressources GR' associé à une ressource [a, r] d'un système hiérarchisé GSH , à un instant i , peut se faire de la manière suivante:

$$GR'([a,r], i) = < AR', DR' >$$

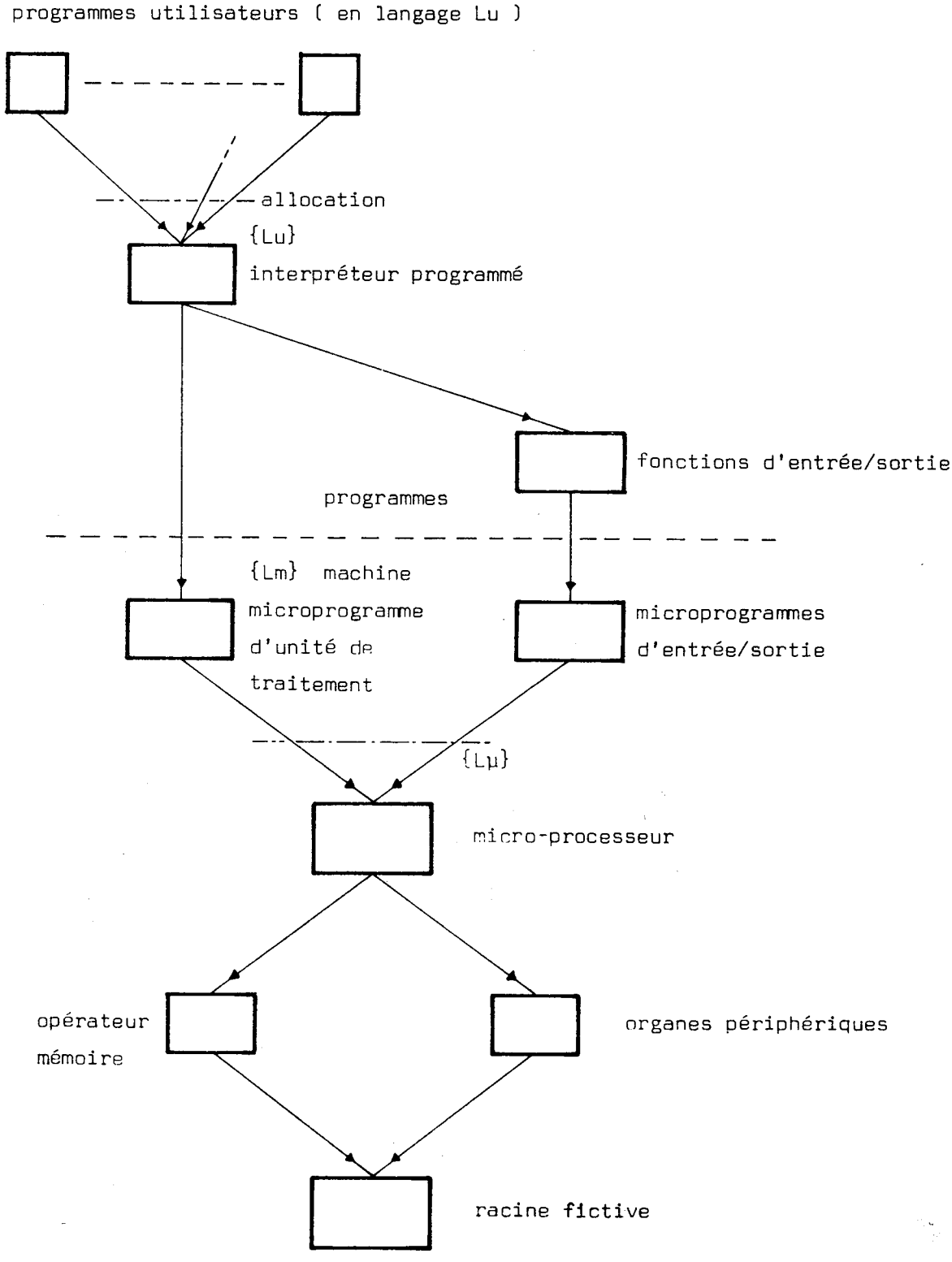
dans lequel

AR' est l'ensemble des articulations [a, r] en service à l' instant i à l'exclusion des articulations multiples. Celles en service à cet instant seront éclatées en autant d'articulations transparentes qu'il y a d'utilisateurs simultanés.



DR' est induit sur AR' par la relation DSH .

Exemple d'un système hiérarchisé:



3 - OUTILS DE SYNCHRONISATION

3/1. MECANISMES DE SYNCHRONISATION

Nous appellerons mécanismes de synchronisation un ensemble d'outils (variables et opérations sur ces variables) destinés à coordonner les progressions individuelles des éléments d'un ensemble d'êtres synchronisable qui peuvent être:

- des processus,
- des utilisateurs de ressources,
- des ressources,
- etc...

Les raisons de cette coordination concernent:

- Les problèmes d'allocation des ressources à leurs utilisateu
- Les problèmes de la régulation de la transmission des flots messages entre les processus.
- Les problèmes de liaison entre les êtres externes et les prc cessus considérés (actions d'un manipulateur humain par exen ple).

Remarque:
Théoriquement, n'importe quelle variable commune à deux êtres peut servir à synchroniser leur progression.
ex.:

attendre tant que A + B > 0

Toutefois, pour des raisons d'implémentation dues, d'une part, aux mécanismes de surveillance qu'il faut déclencher à l'affectation de chaque va riable susceptible d'entrer dans une telle condition, et, d'autre part, aux mécanismes de mutuelle exclusion qu'il faut déclencher pour garantir la cohérence du test des conditions, les mécanismes usuels de synchronisa tion se restreignent à l'usage de variables spéciales, d'un seul type, nc susceptibles d'entrer dans des expressions et liées directement ou indirectement à la comptabilisation des ressources manipulées.

3/2. OUTILS SPECIFIQUES

Nous avons été amenés à étendre et à modifier les mécanismes habituels de synchronisation [27] (variables sémaphores et opérations p et v) car leur application à la synchronisation interne des systèmes hiérarchisés présente quelques difficultés:

- l'expression à l'aide de ces outils,de synchronisation entre une ressource et ses utilisateurs, demande une séquence complexe due au fait que l'articulation qui représente la ressource a un fonctionnem propre et peut elle-même utiliser d'autres ressources.

- les opérations de synchronisation usuelles font, malgré leur généralité, des hypothèses sur les mécanismes d'allocation des ressource utilisées:
  - . mise en file du nouvel arrivant si la ressource sollicitée est occupée,
  - . redémarrage d'un processus en attente lors de la libération de la ressource dont il sollicite l'emploi.

Nous avons également remarqué que les principales actions de synchronisation peuvent être exprimées à l'aide d'une opération unique en raison de la symétrie qu'elles présentent lorsqu'il leur est demandé de gérer elles mêmes des échanges de messages entre les processus (cas des sémaphores communicants [64]).

3/3. SEPARATION ENTRE SYNCHRONISE ET EXECUTANT

L'opération de synchronisation que nous allons introduire présente l'intérêt de séparer nettement l'être qui l'exécute (l'exécutant) de celui dont la progression peut être modifiée par cette opération (le synchronis

Cette propriété permet de rechercher quelles sont les opérations de synchronisation à réaliser indépendamment de l'identité de ceux amenés à les exécuter. Elle permet en particulier d'enrichir l'ensemble des êtres

synchronisables par des êtres inanimés (messages, blocs de données) dans la mesure où d'autres êtres, appelés délégués, seront chargés d'exécuter les opérations de synchronisation concernant les êtres inanimés.

Ex.:

- un processeur exécute les opérations de synchronisation pour le compte du I-processus qu'il exécute.
- le processeur émetteur d'un message exécute l'opération par laquelle ce message attend un destinataire.

3/4. ETAT DE FONCTIONNEMENT

Le fonctionnement des êtres synchronisables, c'est-à-dire la possibilité qu'ils ont de faire évoluer leur état, est caractérisé par la valeur d'une composante particulière de cet état appelé état de fonctionnement qui peut prendre les valeurs:

- MARCHE qui signifie que l'état de l'être peut évoluer de manière interne (cas des processeurs, des opérateurs,...) ou de manière externe (cas des I-processus...).

- ARRET qui signifie que cet état ne doit plus progresser.

. L'opération ARRETER (a) met l'état de fonctionnement de l'être a à la valeur ARRET. Cette opération n'a de sens que si cet état était précédemment à la valeur MARCHE. Nous verrons que cette opération doit déclencher toute une série d'actions pour stopper la progression de l'état de l'être a et prévenir les êtres dialoguant avec lui de ce changement d'état. Cette opération peut:

- arrêter la soumission des requêtes aux ressources utilisées par l'être concerné,
- arrêter la marche autonome de cet être,

- arrêter ou désallouer les ressources qu'il utilise,
- être ineffective dans le cas où l'arrêt de l'être provient de l'arrêt de l'unique ressource (principale) qu'il utilise.
  ex.: arrêt d'un I-processus dû à l'arrêt de son processeur.

. L'opération DEMARRER (a) met l'état de fonctionnement de l'être a à la valeur MARCHE. Cette opération n'a de sens que si cet état était précédemment à la valeur ARRET. Elle doit également déclencher les actions nécessaires pour permettre à l'être a de faire à nouveau progresser son état.

3/5. PARTITION DES ETRES SYNCHRONISABLES

Nous supposerons que les êtres synchronisables, par un mécanisme particulier, se répartissent en deux classes et, qu'une action de synchronisation consiqtera à établir, ou à rompre, des liens entre les êtres de ces deux classes.
ex.:

- des ressources et leurs utilisateurs,
- des processus producteurs et consommateurs d'information,
- des processus et des messages qui leur sont destinés.

3/6. ASSOCIATION

Nous appellerons association l'opération qui consiste à lier un être a de la classe A avec un être b de la classe B .
L'association entre deux êtres cesse par une opération de dissociation.

Considérons l'ensemble AS des couples d'êtres associés vis-à-vis d'un mécanisme de synchronisation.

Nous utiliserons les opérations suivantes:

- l'opération ASSOCIER (a, b) qui a pour effet de ranger le couple (a, b) dans l'ensemble AS. Cette opération n'a de sens que si

précédemment:

$$a \notin proj_A(AS) \quad et \quad b \notin proj_B(AS)$$

- l'opération DISSOCIER (a, b) qui a pour effet de retirer le couple (a, b) de l'ensemble AS. Cette opération n'a de sens que si précédemment:

$$(a, b) \in AS$$

Les opérations de synchronisation sont destinées à permettre aux êtres de chaque classe de se porter candidats à l'association avec un être de l'autre classe.

Remarque:

Les opérations ASSOCIER et DISSOCIER sont supposées commutatives, c'est-dire que:

ASSOCIER (a, b) a le même effet que ASSOCIER (b, a)

DISSOCIER (a, b) a le même effet que DISSOCIER (b, a)

Dans le cas général, l'opération d'association donne à chaque être associé le nom de son partenaire, de manière à ce qu'ils puissent travailler ensemble . Cette information est évidemment inutile pour les êtres inanimés.

ex.:

Il est inutile de donner à un paramètre le nom de son processus utilisateur, mais il est indispensable de donner à ce processus le nom du paramètre qui lui est fourni.

3/7. STRUCTURE DE DONNEES DE SYNCHRONISATION

Les associations entre les éléments de deux classes d'êtres synchronisables A et B se feront par l'intermédiaire d'une structure de données appelées SDS qui réunit:

- une file FA des noms des êtres de la classe A candidats à l'association avec des êtres de la classe B ,

- une file FB des noms des êtres de la classe B candidats à l'association avec des êtres de la classe A ,

- un ensemble AS de couples d'êtres associés par ce mécanisme

structure SDS = < FA, FB, AS >

Remarque:
Les êtres sont extraits des files au moment de leur association. Dans la plupart des cas les files FA et FB sont supposées ne pas pouvoir simultanément contenir des êtres candidats. Cette remarque permet de simplifier les réalisations de ce mécanisme.

Nous imposerons également la restriction qu'un être ne peut simultanément être associé et candidat, c'est-à-dire se trouver au même instant dans une file et dans un couple de AS . La manipulation des files FA et FB est assurée par les deux opérations:

- METTRE (a, F) qui signifie que l'on range l'être a dans la file
- PRENDRE (b, F) qui signifie que l'on prend un être, que l'on appellera b , dans la file F . Cette opération n'aura de sens que lorsque la file F contiendra au moins un élément. Le sens précis de cette opération est:

opération PRENDRE (b, f)
pointeur p
p ← CHOIX(F) "choisir un élément de F et le désigner par p"
b ← EXTRAIRE ($p^{\uparrow}$, F) "extraire l'élément désigné par p et l'affecter à b "

A ce niveau de l'étude, la file F est supposée potentiellement infinie Les algorithmes associés à ces deux opérations définissent la politique de gestion propre à chacune des files (queue, pile, barillet, aléatoire.

3/8. ATTENTE ET CONCURRENCE

Nous dirons qu'un être a de la classe A est attendu par des êtres de la classe B s'il se porte candidat à l'association via une SDS dont la file FB n'est pas vide, c'est-à-dire qu'elle contient des noms d'êtres associables à a .

Nous dirons qu'un être a de la classe A est en concurrence avec les êtres de sa classe en attente d'être associés (c'est-à-dire dans FA) associés (c'est-à-dire dans AS) lorsqu'il se porte candidat à l'association via une SDS d dont la file FB est vide.

Remarque:
Cette défintion de la concurrence inclut le cas où FA et AS sont vides. Dans la majorité des applications il ne sera pas utile de le discier des autres cas de concurrence. Si celà s'avérait utile, il pourrait en être séparé par des tests appropriés.

3/9. OPERATION DE SYNCHRONISATION

Nous utiliserons une opération S unique pour porter un être a de la classe A candidat à l'association avec un être de la classe B via une SDS d = < FA, FB, AS >

```
opération  S (d, a)
           si   FB = Ø  alors
                        CONCURRENT (d, a)
           sinon
                        ATTENDU (d, a)
```

- L'opération CONCURRENT décrit ce qu'il convient de faire lorsque a est en concurrence avec d'autres êtres de sa classe. L'action la plus fréquente consiste à ranger a dans FA et à l'arrêter.

```
opération  CONCURRENT' (d, a)
           METTRE (a, FA)
           ARRETER (a)
```

Nous verrons que d'autres actions sont possibles, en particulier nous pourrons remettre en cause l'association des êtres appartenant à des couples de AS en fonction de régles de priorité, par exemple dans le cas des ressources requérables.

- L'opération ATTENDU décrit ce qu'il convient de faire lorsque a est attendu par des êtres de la classe B . L'action la plus fréquente consiste à associer a avec un être de FB qui devra être démarré.

opération ATTENDU' (d, a)
PRENDRE (b, FB)
ASSOCIER (a, b)
DEMARRER (b)

Nous verrons que d'autres actions sont possibles, en particulier différer l'association de a dans le cas où l'on recherche un regroupement des ressources, par exemple dans certains mécanismes de gestion mémoire.

Nous utiliserons implicitement les opérations CONCURRENT' et ATTENDU' pour illustrer les mécanismes de synchronisation que nous donnerons en exemple par la suite.

Remarque:
Nous noterons S(d,-) l'opération S(d, "exécutant") par laquelle un être se soumet lui-même à l'association.

Lorsque l'indication de la classe de l'être soumis présentera de l' importance, nous utiliserons la notation S(d, a ε A) pour indiquer la soumission d'un être de la classe A à l'association via la SDS d . Nous noterons S(d, - ε A) l'auto-soumission d'un être de cette même classe.

L'association entre deux êtres est supposée prendre fin, soit:

- implicitement par une nouvelle soumission à l'association de l'un de ces deux êtres par une opération S . Cette dissociation se trou

donc être à la charge des opérations CONCURRENT et ATTENDU.
Ex.: un processus se trouve implicitement dissocié du paramètre qu'
il exploitait par le fait de se porter candidat à l'association avec
un nouveau paramètre qui se substituera au précédent.

- explicitement par une opération DISSOCIER.

3/9.1. MUTUELLE EXCLUSION D'EXECUTION D'UNE OPERATION S

Nous supposerons mutuellement exclusives les exécutions des opérations S qui portent sur la même SDS d . L'algorithme décrivant cette opération constitue donc une région critique qui ne peut avoir qu'un exécutant à un instant donné [18].

```
opération  S (d, a)
           région (d)
           |  |
           |  |------ algorithme décrivant l'opération S
           fin
```

Nous nous heurtons alors au problème de la sortie de cette région en cas d'arrêt de l'exécutant. La solution adoptée doit tenir compte que l'exécutant n'est pas toujours celui sur lequel porte l'arrêt. Les solutions envisageables sont:

- sortie artificielle de la région en cas d'arrêt de l'exécutant. Cette solution impose un retour dans la région à son redémarrage, lorsque l'opération d'arrêt n'est pas la dernière à exécuter dans la région.
- augmenter de 1 le degré de multiprogrammation de la région en cas d'arrêt de l'exécutant dans la région. Ce degré doit être décrémenté de 1 au moment de son redémarrage.

Ces mécanismes imposent de prévoir un ensemble d'indicateurs associés à chaque être pour y ranger leur état vis-à-vis des régions au moment de l'arrêt.

3/9.2. RELATIONS AVEC LES OUTILS HABITUELS DE SYNCHRONISATION

Pour établir cette comparaison, considérons un ensemble d'êtres synchronisables constitué des deux classes suivantes:

- la classe N des processus
- la classe P des paramètres échangés entre ces processus.

Les synchronisations entre ces deux classes seront établies par l'intermédiaire d'une SDS

$$d =< FN, FP, AS >$$

qui regroupe les files FN et FP des êtres des classes N et P .
Considérons le sémaphore communicant

$$s =< n, FN, FP >$$

dans lequel n est une variable entière.

Nous pouvons comparer les diverses formes de l'opération S utilisant les opérations CONCURRENT' et ATTENDU' données en exemple, aux opérations p et v de manipulation du sémaphore s .

a) S (d, p ε P) est comparable à une opération v(s,p) donnant le paramètre p

b) S (d, - ε N) est comparable à une opération p(s)

c) S (d, a ε N) est comparable à une opération p sur le sémaphore s exécutée par un être c pour le compte d'un être a

d) S (d, - ε P) est comparable à une opération v sur le sémaphore s donnant le nom de son exécutant comme paramètre

Les formes a et b correspondent à des opérations classiques tandis que les formes c et d sont moins usuelles. La forme c est comparable à une opération p en ce qui concerne la manipulation des files FN et FP mais elle ne peut pas mettre l'exécutant en attente. La forme d est comparable à une opération v en ce qui concerne la manipulation des files mais elle peut mettre l'exécutant en attente.

La possibilité d'utiliser d'autres formes d'opérations CONCURRENT et ATTENDU élargit encore les possibilités de l'opération S .

3/9.3. Relations liées à l'utilisation des opérations S

Les relations suivantes sont valables lorsqu'il est fait usage des opérations CONCURRENT' et ATTENDU' données à titre d'exemple.

Soit:

ℓFA la longueur de la file FA d'une SDS d ($\ell FA \geq 0$)

ℓFB la longueur de sa file FB ($FB \geq 0$)

la relation

$$\ell FA \,.\ell FB = 0 \qquad R1$$

est conservée par l'exécution d'une opération S , en effet:

- Action d'une opération S(d, a ε A)

si $\ell FB = 0$ => $\ell FA \leftarrow \ell FA + 1$ et ℓFB inchangée

si $\ell FB > 0$ => $\ell FB \leftarrow \ell FB - 1$ et ℓFA inchangée (elle peut en particuli
rester = 0

- Action d'une opération S(d, b ε B)

si $\ell FA = 0$ => $\ell FB \leftarrow \ell FB + 1$ et ℓFA inchangée

si $\ell FA > 0$ => $\ell FA \leftarrow \ell FA - 1$ et ℓFB inchangée (elle peut en particuli
rester = 0

Dans tous les cas la relation R1 est conservée.

La relation R1 signifie que l'usage des files FA et FB est exclusif.

Il semble cohérent de l'imposer à l'origine des temps par

$$\ell FA^{o} . \ell FB^{o} = 0$$

Soit:

nSA le nombre d'opérations de type S(d, a ε A) exécutées depuis l'origine des temps,

nSB le nombre d'opérations de type S(d, b ε B) exécutées depuis la même origine,

nA le nombre de couples d'êtres associés depuis cette origine.

les relations

$$nA + \ell FA = nSA + \ell FA^{O} \qquad R21$$
$$nA + \ell FB = nSB + \ell FA^{O} \qquad R22$$

sont conservées par l'exécution d'une opération S , en effet:

- Action d'une opération S(d, a ε A) sur ces relations

$$nSA \leftarrow nSA + 1$$

si $\ell FB = 0 \Rightarrow \ell FA \leftarrow \ell FA + 1$

si $\ell FB > 0 \Rightarrow \ell FB \leftarrow \ell FB - 1$ et $nA \leftarrow nA + 1$

- Action d'une opération S(d, b ε B)

$$nSB \leftarrow nSB + 1$$

si $\ell FA = 0 \Rightarrow \ell FB \leftarrow \ell FB + 1$

si $\ell FA > 0 \Rightarrow \ell FA \leftarrow \ell FA - 1$ et $nA \leftarrow nA + 1$

Dans tous les cas les relations R21 et R22 sont conservées.
A l'origine des temps on a $nA = 0$, $\ell FA = \ell FA^{O}$, $\ell FB = \ell FB^{O}$, $nSA = nSB = 0$ ce qui satisfait les relations R21 et R22 .
La relation

$$nA = \min (nSA + \ell FA^{O}, nSB + \ell FB^{O}) \qquad R3$$

découle des relations R1, R21 et R22 en effet:

si $\ell FB = 0 \Rightarrow \ell FA \geq 0 \Rightarrow nA = nSB + \ell FB = (nSA + \ell FA^{O}) - \ell FA$

si $\ell FA = 0 \Rightarrow \ell FB \geq 0 \Rightarrow nA = nSA + \ell FA = (nSB + \ell FB^{O}) - \ell FB$

Dans les deux cas

$$nA \leq nSA + \ell FA^{O}$$
$$nA \leq nSB + \ell FB^{O}$$

3/10. RELATIONS ENTRE L'ALLOCATION ET LA SYNCHRONISATION

L'allocation des ressources à leur utilisateur est une forme d'association entre les êtres de ces deux classes. Les mécanismes d'allocation se décomposent donc en plusieurs niveaux:

- procédures d'allocation utilisant les opérations S ,
- opérations ATTENDU et CONCURRENT utilisées par ces opérations S,
- opérations METTRE et PRENDRE utilisées pour les gestion des files

L'établissement et la suppression de l'association entre les êtres de deux classes peut être liée à l'occurrence de certains événements:

- exécution d'opérations de synchronisation,
- évènements extérieurs, ex.: horloge.

Cette association, une fois établie, peut être remise en cause pour optimiser le fonctionnement global du système. Cette action peut également être liée à l'occurrence d'évènements tels que:

- exécution de toutes ou certaines opérations de synchronisation portant sur n'importe quels êtres de ces classes,
- évènements exétrieurs, ex.: horloge, variations de certains paramètres tels que la priorité relative des êtres synchronisés.

Pratiquement, les combinaisons suivantes sont utilisées:

- L'établissement et la suppression de l'association sont liés à l'exécution d'opérations de synchronisation. La remise en cause de cette association est liée à l'exécution de certaines opérations de synchronisation et à des évènements exétrieurs.
  Ex.: cas des systèmes d'exploitation multiprogrammés ou en partage de temps

- L'établissement et la remise en cause de l'assocation sont liés à l'occurrence d'évènements extérieurs. La suppression de l'associati restant liée à l'exécution de certaines opérations de synchronisatio
  Ex.: mécanismes de barillet.

Dans ces mécanismes les opérations de synchronisation qui soumettent les êtres de l'une des classes se résument à de simples rangements de ces êtres dans les files d'attente des SDS utilisées pour ces synchronisations .

Ces remarques signifient que dans certains cas les seules opérations de synchronisation ne peuvent assurer l'ensemble des mécanismes d'allocation, en particulier lorsque des évènements extérieurs sont susceptibles d'établir ou de remettre en cause l'association.

3/10.1. Synchronisation dissymétrique

Il est fréquent que la nature des opérations CONCURRENT et ATTENDU, utilisées par une opération S dépende de la classe de l'être qui se porte candidat à l'association par cette opération. Une telle dissymétrie apparaît généralement lorsque deux classes d'êtres à synchroniser sont très différentes.

Ex.: Considérons le problème de la synchronisation de I-processus vis-à-vis d'un ensemble de processeurs. Les I-processus sont supposés ordonnés par une règle de priorité qui nous incitera à favoriser l'association des plus prioritaires, tandis que les processeurs sont banalisés.

Ex.: Considérons un mécanisme de barillet entre un ensemble d'utilisateurs U et une ressource r :

- la soumission d'un utilisateur $u \in U$ se résumera à une simple mise en liste de manière à ce que le mécanisme d'association, lié à une horloge externe, puisse réaliser les associations sollicitées.

En ce qui concerne la manipulation des utilisateurs les opérations CONCURRENT et ATTENDU seront:

opérations $CONCURRENT^B$, $ATTENDU^B$ (d, $u \in U$)
METTRE (u, FU)
ARRETER (u)

- la soumission de la ressource r aura pour effet de dissocier le couple dont elle faisait partie, puis de mettre cette ressource en liste pour une association ultérieure.

En ce qui concerne la manipulation de la ressource les opérations CONCURRENT et ATTENDU seront:

opérations $CONCURRENT^B$, $ATTENDU^B$ (d,r)
DISSOCIER (u, r)
METTRE (r, FR)
ARRETER (r)

Le mécanisme d'association indépendant explore, au rythme de l'horloge les différents êtres de la classe U et les associe avec la ressource s'ils sont candidats à l'association.

```
boucle:
        attendre occurrence horloge
        si   u_i ε proj_U(AS)  alors
                    début
                          DISSOCIER (u_i, r)
                    fin
        sinon
             DEMARRER (r)
             i  ←  i + 1 / mod n
             si  u_i ε FU  alors
                    début
                          EXTRAIRE (u_i, FU)
                          ASSOCIER (u_i, r)
                    fin
  fin
```

3/10.2. Requisition des ressources

Considérons un mécanisme de synchronisation entre une classe d'utilisateu U ordonnés par priorité et une classe de ressources R considérées comme équivalentes.

Les ressources sont dites requérables lorsqu'on permet la remise en cause de leur association avec les utlisateurs de manière à favoriser les plus prioritaires des utilisateurs candidats.

Ces mécanismes de synchronisation sont dissymétriques, la nature des opérations CONCURRENT et ATTENDU dépend des libertés que l'on se permet dans l'évolution de la priorité dans le temps:

- priorité fixe,
- priorité modifiée au moment de l'exécution des opérations de synchr nisation,
- priorité variable à n'importe quel instant.

Ce dernier cas impose l'usage d'un mécanisme d'allocation autonome déclenché par un évènement associé à toute évolution de la priorité.
Ex.: Le cas d'utilisateurs de priorité fixe peut être résolu à l'aide des opérations CONCURRENT et ATTENDU suivantes.

Cas des utilisateurs:

```
opération CONCURRENT^R (d, u ε U)
        si AS = Ø alors
            début
            METTRE (u, FU)
            ARRETER (u)
            fin

    sinon
        (u* , r*) ← min              { (u', r') ε AS }
                    priorité (u')
        si priorité (u) < priorité (u*) alors
            début
            METTRE (u, FU)
            ARRETER (u)
            fin
        sinon
            début
            ARRETER (u*)
            DISSOCIER (u* , r*)
            METTRE (u*, FU)
            ASSOCIER (u, r*)
            fin
```

Cas des ressources:

```
opération  CONCURRENT^R  (d, r ε R)
           CONCURRENT'   (d, r)
```

Cette synchronisation utilise l'opération ATTENDU' déjà présentée.

3/11. SDS LIEES

Nous rencontrerons souvent des mécanismes de synchronisation entre deux classes d'êtres A et B dont l'une se réduit à un seul élément (B ={b Nous appellerons liée (à la classe B) une SDS chargée de gérer l'asso ciation des êtres de ces deux classes.

La file FB de cette SDS n'est susceptible que de contenir, au plus, l'être b . L'ensemble AS de couples d'êtres associés n'est susceptib de contenir que le seul couple (b, x) dans lequel x est un élément que conque de la classe A. Nous pouvons profiter de ces propriétés pour sir plifier la réalisation des opérations S qui manipulent ces SDS.

Une première simplification consiste à représenter l'état de la file FB par un simple indicateur binaire I dont la signification est:

I = 0 → FB = ∅

I = 1 → b ∈ FB

Dans le cas où l'on utilise les adaptations des opérations CONCURRENT' et ATTENDU' données en exemple, les opérations S deviennent:

```
opération S (Δ, a)
    si I = 0 alors
        CONCURRENT' (Δ, a)
    sinon
        ATTENDU'_S (Δ, a)
```

```
opération ATTENDU'_S (Δ, a)
    I ← 0 "PRENDRE (b, FB)"
    ASSOCIER (b, a)
    DEMARRER (b)
```

```
opération S (Δ, b) " b peut être implicite"
    si FA = ∅ alors
        CONCURRENT'_S (Δ, b)
    sinon
        ATTENDU' (Δ, b)
```

```
opération CONCURRENT'_S (Δ, b)
    I ← 1 "METTRE (b, FB)"
    ARRETER (b)
```

- Une seconde simplification consiste à remarquer que l'examen de l'état de $proj_A(AS)$ suffit à prendre les décisions concernant la réalisation des opérations S , en effet:

$proj_A(AS) \iff b \in proj_B(AS) \iff FB = \emptyset$

car il n'y a qu'un seul élément b.

Les opérations deviennent:

```
opération S (Δ, a)
  si proj_A(As) ≠ ∅ alors
      CONCURRENT'(Δ, a)
  sinon
      ATTENDU'_sm (Δ, a)
```

```
opération ATTENDU'_sm (Δ, a)
          ASSOCIER (b, a)
          DEMARRER (b)
```

```
opération S (Δ, b)
  si FA = ∅ alors
      CONCURRENT'_sm (Δ, b)
  sinon
      ATTENDU' (Δ, b)
```

```
opération CONCURRENT'_sm (Δ, b)
          ARRETER (b)
```

Remarque:

Dans cette réalisation la file FB est inutile.

Souvent les files FA et $proj_A(AS)$ seront regroupées en une seule file FΔ qui pourra être testée de la même manière. En effet:

$proj_A(AS) \neq \emptyset \iff FB = \emptyset$

$FA \neq \emptyset \iff FB = \emptyset$

d'où

$(\ F\Delta = FA \,||\, proj_A(AS)\ ) \neq \emptyset \iff FB = \emptyset$

Les opérations S deviennent alors

```
opération S (Δ, a)
  si FΔ = ∅ alors
      CONCURRENT'_sn (Δ, a)
  sinon
      ATTENDU'_sm (Δ, a)
```

```
opération CONCURRENT'_sn (Δ, a)
          METTRE (a, FΔ)
          ARRETER (a)
```

```
opération S (Δ, b)
    si FA ≠ Ø alors
        CONCURRENT'sm (Δ, b)
    sinon
        ATTENDU'sn (Δ, b)
```

```
opération ATTENDU'sn (Δ, b)
        PRENDRE (a', FΔ)
        ASSOCIER (a', b)
        DEMARRER (a')
```

3/12. SYNCHRONISATION DES SYSTEMES HIERARCHISES

Les mécanismes d'allocation des articulations d'un système hiérarchis font appel aux mécanismes de synchronisation précédents.

Les êtres à synchroniser se répartissent en deux classes vis-à-vis de chacun de ces mécanismes:

- les ressources qui constituent une classe appelée R ,
- les utilisateurs de ces ressources qui constituent une classe appelée U.

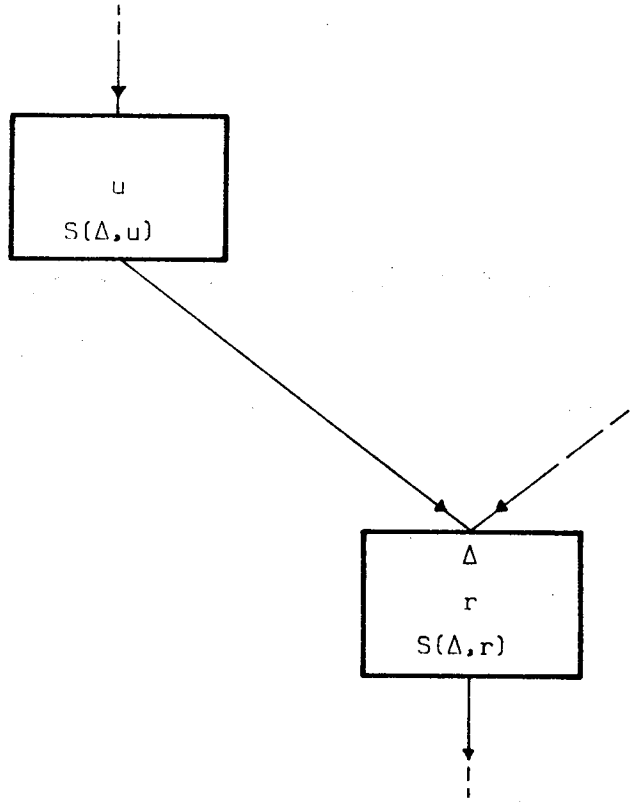
Chacun des êtres à synchroniser est donc représenté par une articulation. Nous attacherons les mécanismes d'allocation aux articulations représentant les ressources.

Ce point de vue entraîne une légère dissymétrie dans les raisonnements Une SDS Δ est attachée à chacune de ces articulations

$$\Delta = < FR, FU, AS >$$

- Un utilisateur u exécute une opération S(Δ, u) pour se porter candidat à l'association avec une ressource utilisant la SDS Δ pou gérer son allocation. Nous dirons que cet utilisateur demande cette ressource ou qu'il se soumet à son traitement.

- Une ressource r , utilisant la SDS Δ pour gérer son allocation exécute une opération S(Δ, r) pour se porter candidate à l'association avec un nouvel utilisateur. Nous dirons que cette ressource de mande un nouvel utilisateur ou qu'elle se met au service d'un nouvel utilisateur.

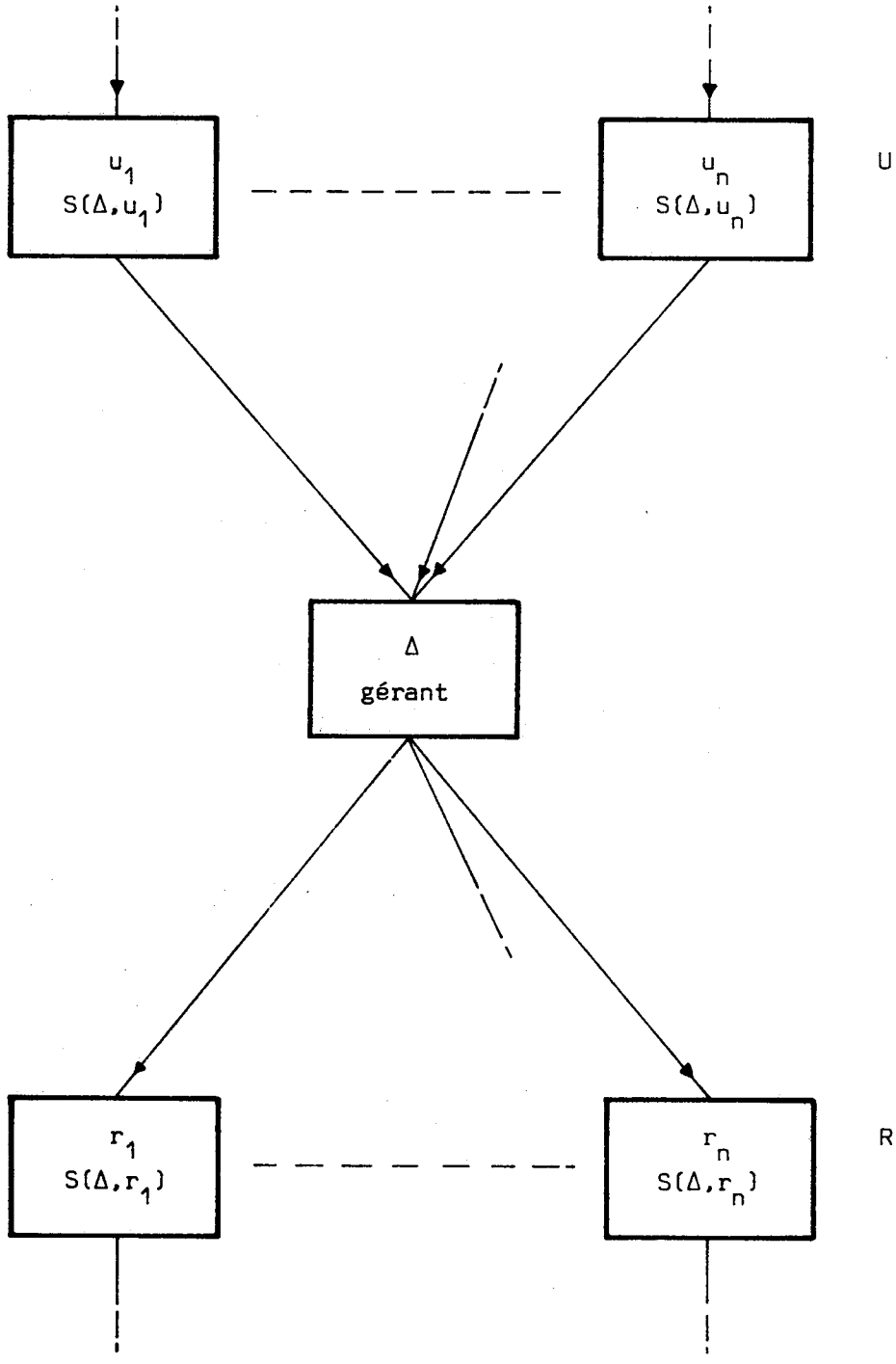


Nous verrons [§ 4] que ces mécanismes peuvent propager des modifications d'allocation dans tout le système hiérarchisé considéré.

3/12.1. Cas des articulations multiples

Les mécanismes de synchronisation, liés aux articulations, sont tels que dans les cas courants, la classe R des ressources considérées par l'un de ces mécanismes ne comporte qu'un seul élément (la ressource représentée par cette articulation).

L'utilisation d'articulations multiples dupliquées permet d'envisager le cas où la classe R contient les noms r_i des ressources dupliquées Un tel mécanisme est représenté de la manière suivante.


u_1
$S(\Delta, u_1)$
u_n
$S(\Delta, u_n)$
U
Δ
gérant
r_1
$S(\Delta, r_1)$
r_n
$S(\Delta, r_n)$
R

- Le gérant dispose d'une SDS Δ liée à cette ressource multiple.
- Un utilisateur exécute une opération $S(\Delta, u_i)$ pour solliciter l'usage d'une ressource de R .
- Chaque ressource r_j exécute une opération $S(\Delta, r_j)$ pour demander un nouvel utilisateur.

Les relations entre le gérant et les ressources dupliquées peuvent être régies par le même mécanisme de synchronisation sous l'hypothèse que les ressources dupliquées sont toujours allouées au gérant.

Remarque: L'opération PRENDRE appliquée à la file FR de ce mécanisme doit tenir compte de la disponibilité relative de chacune des ressources dupliquées. Nous verrons [§ 4/6] de tels mécanismes.

Le cas des articulations multiples regroupant des ressources de nature différente se traite de la même manière que celui des articulations simples puisqu'il ne s'agit que d'un moyen pour rassembler ces différentes ressources en une seule.

3/12.2. Cas des articulations multiples dégénérées

Une articulation multiple dégénérée joue le rôle d'un ensemble de ressources vides. Son fonctionnement nécessite l'usage de fonctions CONCURRENT et ATTENDU particulières pour gérer son allocation, dans le cas où l'on admet qu'elle peut être allouée simultanément à un nombre non borné d'utilisateurs.

- manipulation des utilisateurs:

opération CONCURRENT (Δ, u ε U)
ASSOCIER (u, r)

opération ATTENDU (Δ, u ε U)
PRENDRE (r, FR)
ASSOCIER (u, r)
DEMARRER (r)

} premier utilisateur

- manipulation de la ressource multiple

  opération CONCURRENT (Δ, r ε R)
  PRENDRE (u, $proj_u$ (AS))
  DISSOCIER (u, r)
  si AS = Ø alors
  ARRETER (r)

  opération ATTENDU (Δ, r ε R)
  l'exécution de ce cas de l'opération ATTENDU
  est logiquement impossible puisque les utilisateurs
  d'une telle ressource sont servis dès leur requête.

3/12.3. Ressources particulières

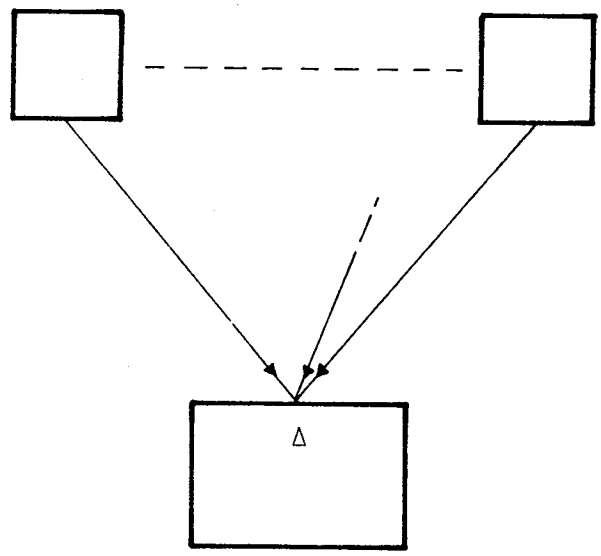
Très fréquemment un sytème hiérarchisé contient de nombreuses articulations représentant des ressources requérables (par exemple les processeurs). Ce signifie que les opérations de synchronisation manipulant la SDS associée à cette articulation utiliseront des opérations CONCURRENT et ATTENDU particulières.

Certaines articulations sont munies d'un mécanisme d'allocation autonome qui décide spontanément de modifier les associations en cours.

3/12.4. Mutuelle exclusion entre les articulations

Un mécanisme de mutuelle exclusion entre des articulations situées dans un même niveau peut être aisément introduit à l'aide d'une ressource fictive commune à toutes ces articulations.

articulations à synchroniser



ressource fictive utilisée
pour établir la mutuelle
exclusion

Le mécanisme d'allocation de cette ressource doit être simple, c'est-à-dire qu'il ne doit l'allouer qu'à un utilisateur à la fois, sans réquisition possible.
Ce mécanisme d'allocation réalise donc l'arbitrage des conflits relatifs à la mutuelle exclusion souhaitée.
Les opérations utilisées sont:

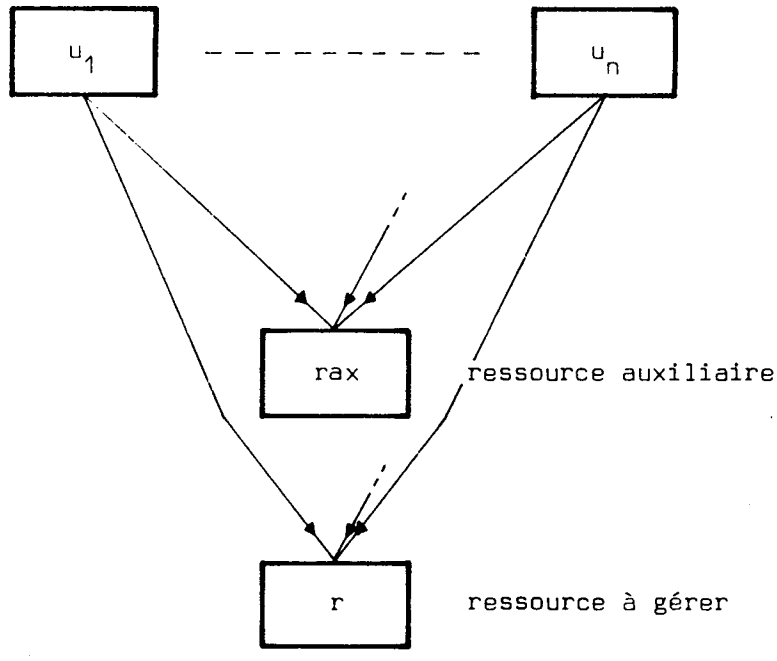
- La mutuelle exclusion est sollicitée en demandant l'usage de la ressource par une opération S(Δ, -)
- Elle est abandonnée en libérant la ressource par une opération S(Δ, ress).

3/12.5. Allocation par l'intermédiaire d'une ressource auxiliaire

Lorsque, pour des raisons de réalisation, une ressource r ne peut être munie d'un mécanisme d'allocation en propre, il peut être fait usage d'une ressource auxiliaire rax réduite à son simple mécanisme d'allocation, du type désiré, pour gérer indirectement les utilisateurs de r .

Un utilisateur u doit d'abord solliciter l'usage de la ressource rax avant de solliciter celui de r . La ressource auxiliaire peut être utilisée pour gérer l'allocation de plusieurs ressources utilisées successivement.

Ex.: bascules de mutuelle exclusion utilisées pour gérer l'accès à des ressources non munies de mécanismes d'allocation.



3/12.6. Synchronisation complémentaire

3/12.6.1. Synchronisation par échange d'information

Il est assez fréquent que les articulations d'un même niveau aient à échanger des messages entre eux. Le point de vue que nous développons nou incite à assimiler chacun de ces messages à une ressource crée par l'articulation qui l'émet et sollicitée par celles qui l'attendent.

3/12.6.2. Ressources de synchronisation complémentaire

Nous pouvons toutefois adopter un point de vue différent dans le but d'éviter la prolifération de telles ressources à caractère temporaire.

Les échanges d'information peuvent être vus comme des associations entre des éléments de deux classes d'êtres:

- les articulations qui attendent les messages (classe A),
- les messages eux-mêmes (classe M).

Nous supposons également que les échanges de messages peuvent se faire vi des ressources permanentes (boîtes aux lettres) que nous appellerons RSC (ressources de synchronisation complémentaire) dont le but est de contenir une SDS objet ds relative à l'association des messages avec les articulations qui les attendent. Cette SDS ne doit pas être confondue avec celle Δ qui gère la RSC en tant que ressource.

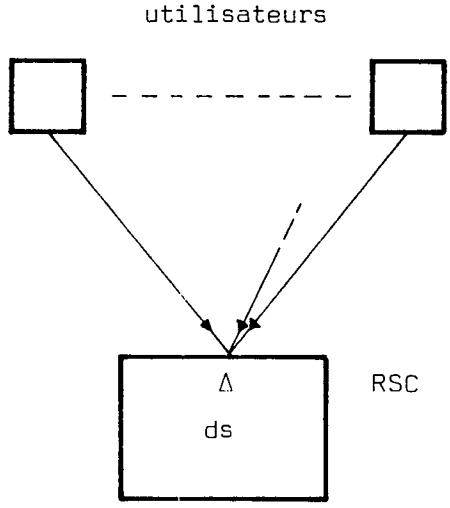
ds = < FA, FM, AS >

Les articulations qui émettent des messages doivent donc exécuter la séquence:

- solliciter l'usage de la RSC par S(Δ, -)
- émettre le message par S(ds, message)
- libérer la RSC par S(Δ, rsc).

Pour solliciter un message une articulation exécute la séquence:

- solliciter l'usage de la RSC par S(Δ, -)
- solliciter un message par S(ds, -)
- libérer la RSC par S(Δ, rsc)



Le mécanisme d'allocation de la RSC assure la mutuelle exclusion entre les utilisateurs, nécessaire à l'exécution des opérations S vis-à-vis de la SDS objet ds . Ce mécanisme doit tenir compte d'une éventuelle mise en attente de l'exécutant dans la région critique [§ 3/9.1] .
Définition: Nous appellerons RSC liée à une articulation a une RSC dont la SDS objet est liée à cette articulation.

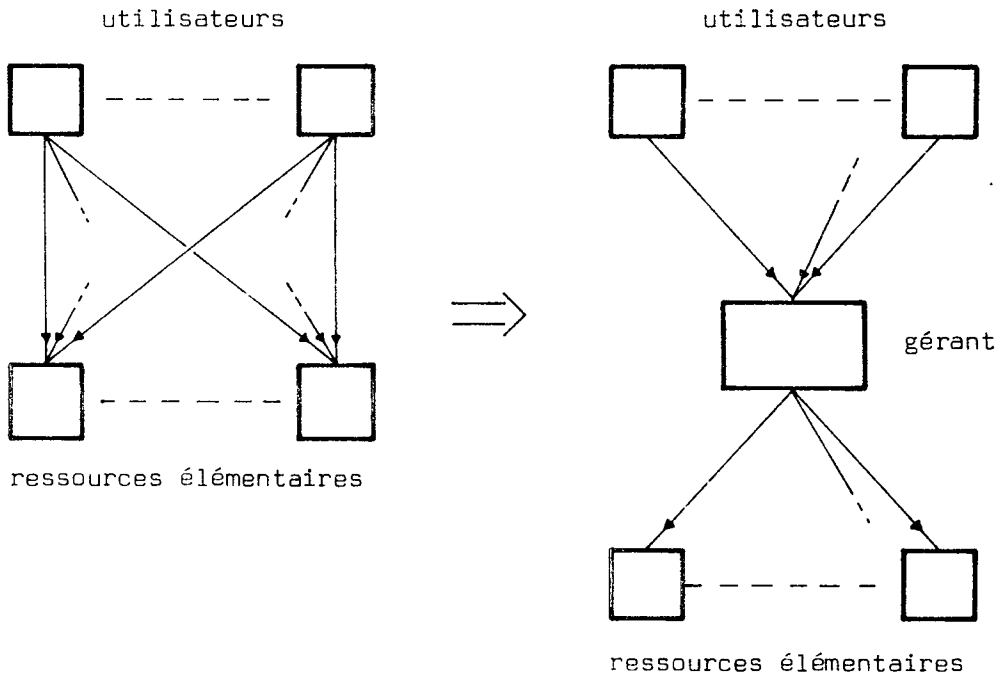
3/12.7. Regroupement des ressources

Une articulation utilisatrice d'une RSC peut se trouver en attente pour l'une des deux raisons suivantes:

- la RSC est indisponible,
- la file FM de la SDS objet ds est vide.

L'apparition de deux causes d'attente est un phénomène très général qui apparaît chaque fois que l'on regroupe, sous le contrôle d'un gérant qui ne peut être alloué qu'à un utilisateur, un ensemble de ressources.

Cette opération revient simplement à introduire un niveau supplémentaire dans la hiérarchie.



Les deux causes d'attente sont donc:

- attente de la disponibilité du gérant,
- attente de la disponibilité d'une ressource (quelquefois attente de son existence).

Nous verrons dans le chapitre consacré aux mécanismes d'induction [§ 4] les moyens pour transmettre l'attente due à l'indisponibilité des ressources élémentaires,au gérant puis aux utilisateurs.

Il n'est donc pas nécessaire de prévoir au niveau de cet utilisateur un état spécial pour enregistrer de telles attentes.

Remarques:

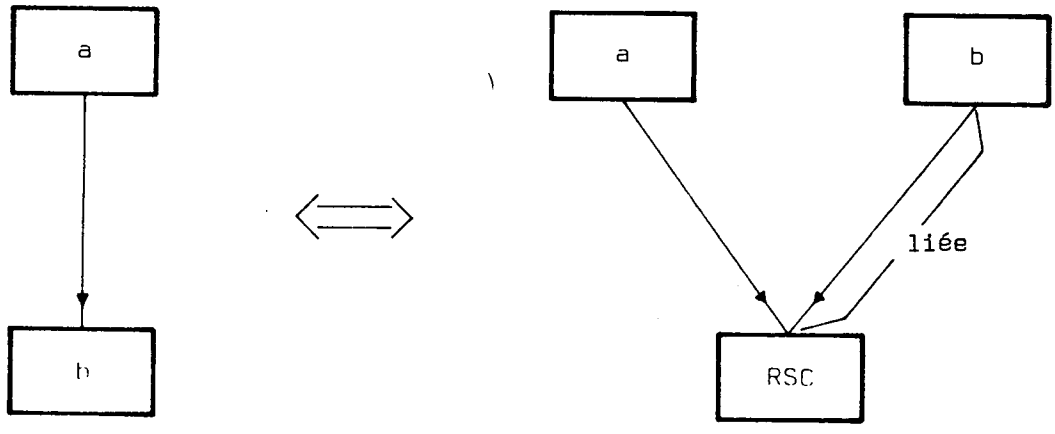
L'attente de l'existence d'une ressource élémentaire n'est pas un mécanisme prévu par le formalisme de base mais il peut simplement être ramené au cas de l'indisponibilité de cette ressource à l'aide d'un mécanisme approprié au niveau du gérant (par exemple la SDS objet d'une RSC remplit cette fonction).

Exemples de ressources groupées:

- groupement de ressources messages par une RSC,
- un processeur peut être vu comme le groupement de ressources plus élémentaires correspondant aux opérations qu'il peut exécuter.

3/12.8. Transformation de la hiérarchie

Nous pouvons montrer qu'il est équivalent d'admettre une relation hiérarchique entre deux êtres a et b ou de les relier par une RSC liée à l'un d'entre eux.

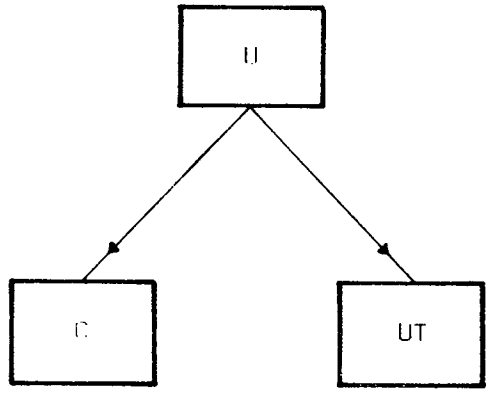


En effet, dans les deux cas l'articulation a soumet des messages à l'articulation b qui les voit comme des opérations à exécuter. Dans les deux cas cette dernière ne peut progresser qu'en accord avec l'arrivée de ces messages.

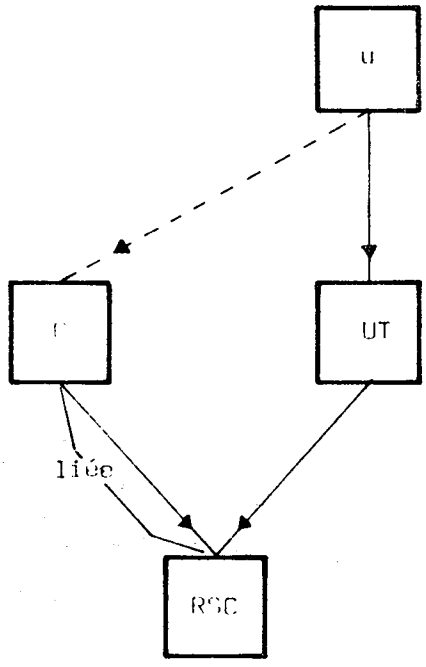
Cette transformation sera très utilisée par la suite:

- pour réaliser une relation hiérarchique lorsque celà n'est pas physiquement possible.

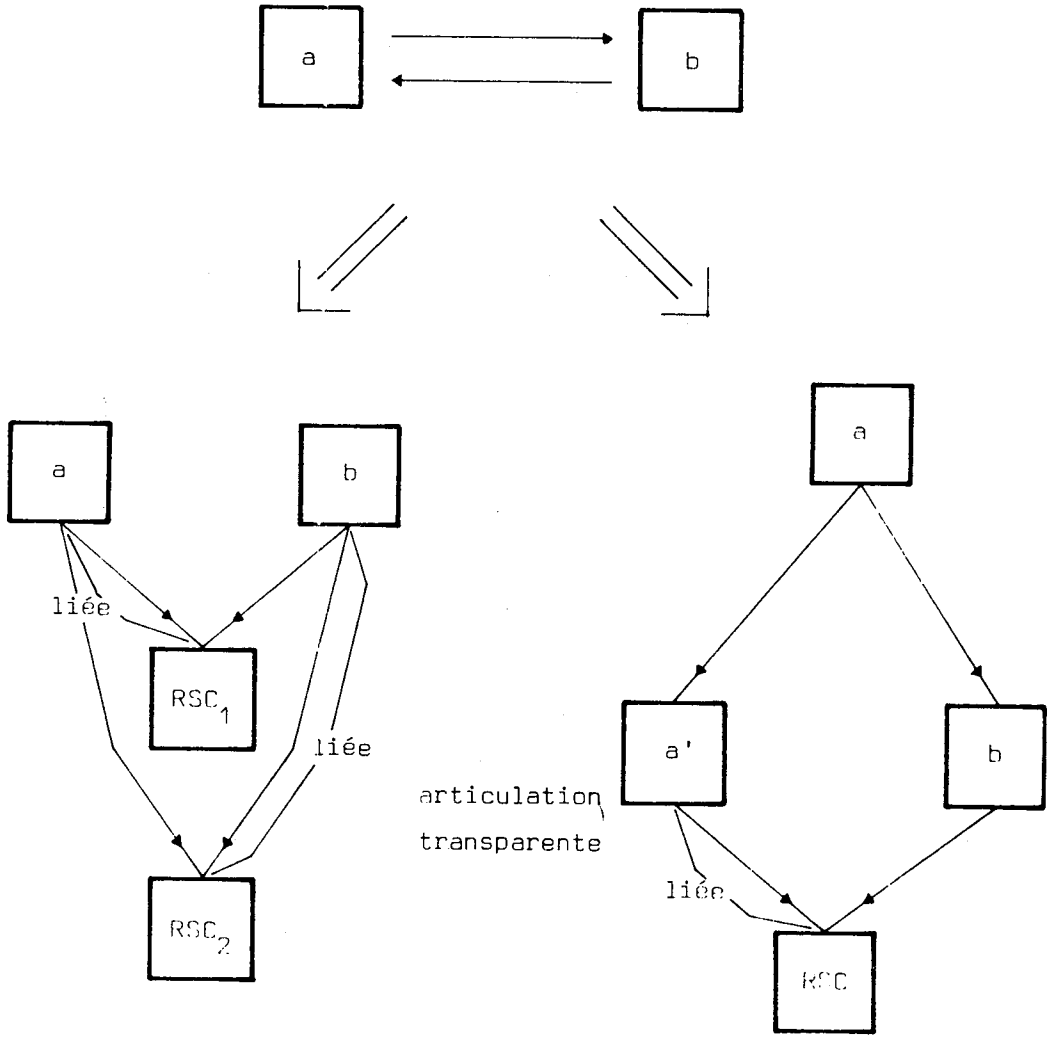
  Ex.: Considérons le cas d'un être programmé utilisant l'unité centrale et un canal d'un ordinateur.



Si cet utilisateur est un I-processus, il s'exécute uniquement sur l'unité de traitement qui est sa seule ressource et il doit passer par elle pour solliciter l'usage du canal.

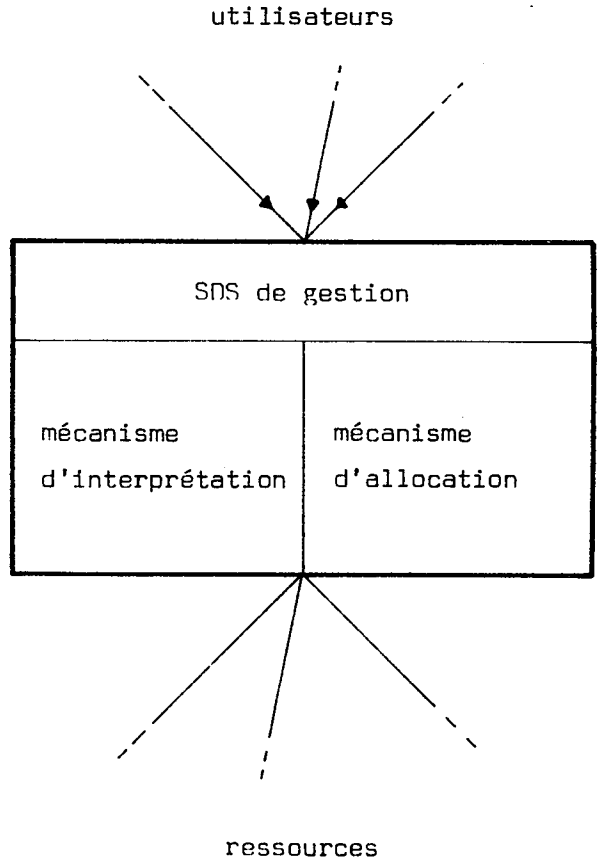


- Lorsque deux êtres s'envoient alternativement des messages qui peuvent être vus comme des travaux à exécuter. Une relation hiérarchique peut être associée au sens d'échange qui semble le plus important.



3/12.9. Représentation graphique d'une articulation

Nous représenterons, par la suite, les articulations à l'aide de la convention graphique ci-dessous lorsque l'on désirera mettre en évidence les différents mécanismes qui les composent.



Remarque: Nous dissocierons quelquefois, dans cette représentation, les ressources propres au mécanisme d'interprétation et celles propr au mécanisme d'allocation.

4 - MECANISMES D'INDUCTION

4/1. PROPAGATION DE L'ACTIVITE

La finalité des systèmes représentés à l'aide de ce formalisme est de fournir un certain travail, considéré comme utile, au niveau des feuilles de la hiérarchie.

Pour qu'une articulation puisse progresser, il faut que ses ressources exécutent les opérations qu'elle leur soumet. Cette condition implique que cette articulation, pour progresser, doit être reliée à la racine de la hiérarchie par des chaînes d'articulation actives. La racine de la hiérarchie doit être supposée toujours active pour que le système puisse continuer à évoluer.

4/2. INDUCTIONS

Les mécanismes que nous allons étudier sous le nom d'inductions ont pour rôle de propager l'activité dans la structure:

- soit vers la racine pour établir les chaînes d'articulation actives reliant une articulation à la racine,
- soit vers les feuilles pour prévenir les articulations de niveau supérieur de la disponibilité d'une ressource.

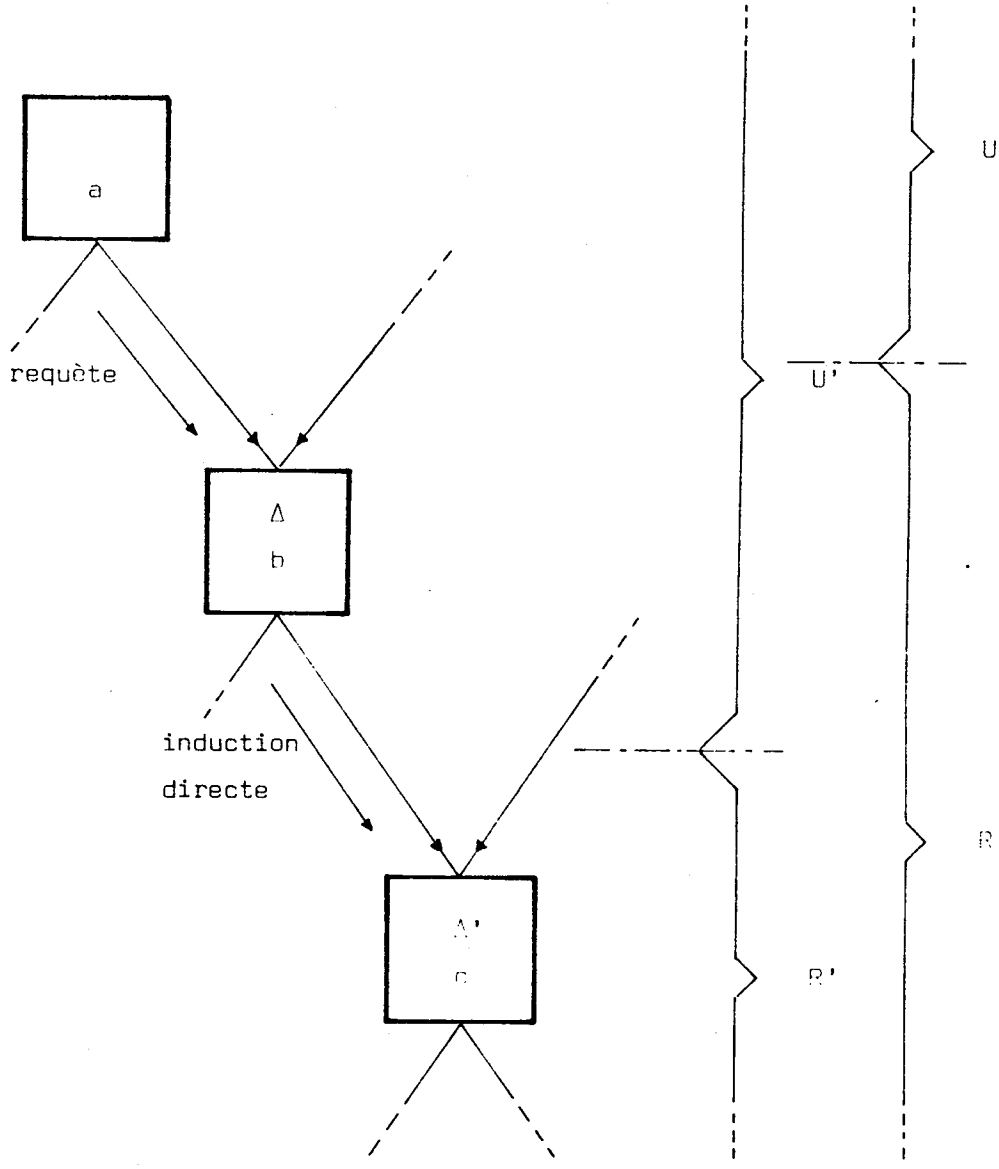
4/2.1. Inductions directes

L'induction directe est un mécanisme qui consiste à propager le désir d'activité d'une articulation vers ses ressources. Cette propagation se poursuit vers la racine tant que l'on rencontre des ressources en attente d'utilisateurs.

Ce mécanisme est généralement déclenché par les opérations DEMARRER et ARRETER indirectement contenues dans les opérations S .

Donnons à titre d'illustration la forme de ce mécanisme dans le cas de l' utilisation des opérations CONCURRENT' et ATTENDU' donnés à titre d'exempl

Considérons une chaîne de trois articulations a, b, c telles que c rep sente une ressource de b qui représente elle-même une reccource de a .



Si l'articulation a sollicite l'association avec b par une opération S(Δ, a ε U). L'exécution de cette opération sera, si nous nous plaçons dar le cas où l'articulation b est arrêtée:

```
opération S(Δ, a ε U)
          si FR = Ø alors
                  :
          sinon
               ATTENDU (Δ, a)

opération DEMARRER (b)
                  :
        S (Δ', b ε U')
```

```
opération ATTENDU (Δ, a)
          PRENDRE (b, FR)
          ASSOCIER (b, a)
          DEMARRER (b)
```

Le démarrage de l'articulation b entraîne qu'elle se porte candidate à l'association avec l'articulation c par l'opération S(Δ', b ε U').

Supposons maintenant que la ressource représentée par l'articulation b désire s'associer avec un nouveau candidat en exécutant l'opération S(Δ, b ε R). L'exécution de cette opération sera, si nous nous plaçons dans le cas où il n'y a aucun candidat à cette association.

```
opération S(Δ, b ε R)
          si FU = Ø alors
              CONCURRENT (Δ, b)
          sinon
                  :

opération ARRETER (b)
                  :
                  :
                  :
          S (Δ', c ε R')
```

```
opération CONCURRENT (Δ,
          RANGER (b, FR)
          ARRETER (b)
```

L'arrêt de l'articulation b entraîne que la ressource représentée par l'articulation c se porte candidate à l'association par l'opération S (Δ', c ε R').

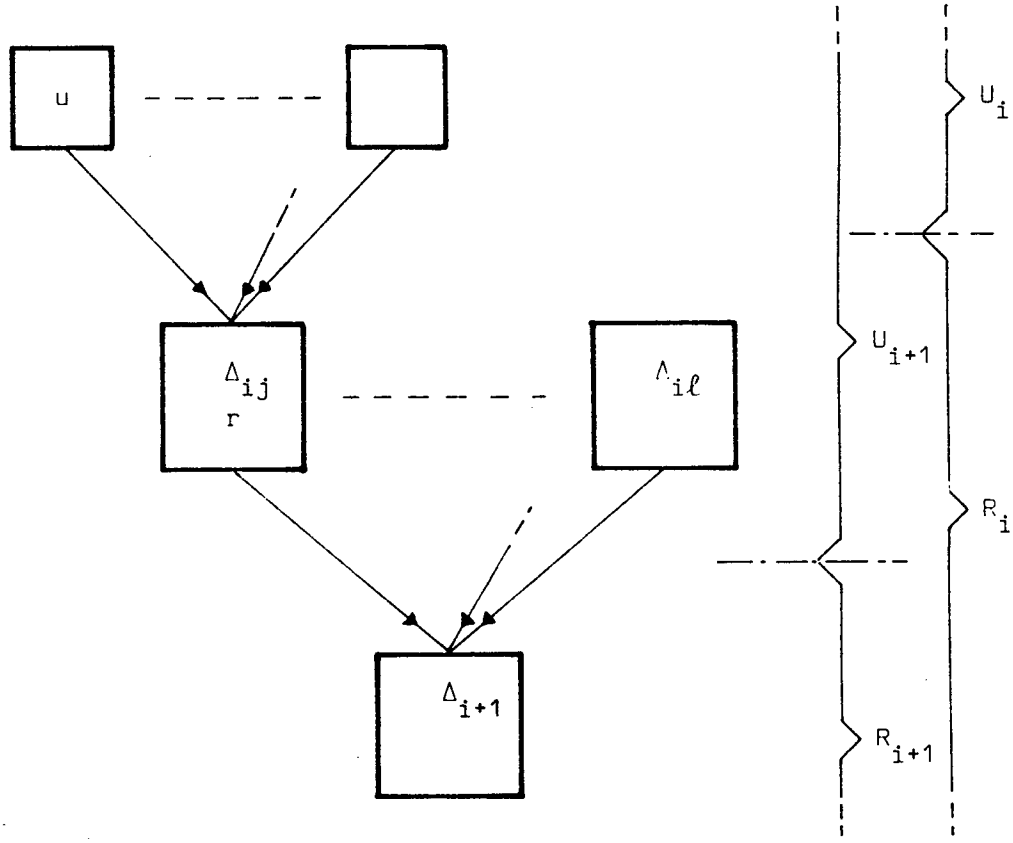
Nous remarquons que ces mécanismes d'induction directe provoquent l'exécution récursive d'opérations de synchronisation pour le compte des articulations de niveau inférieur. Cette récursion s'arrête pour le premier cas, sur des articulations déjà allouées ($FR^n = \emptyset$), pour le second cas sur des articulations ayant d'autres candidats ($FR^n = \emptyset$).

4/2.1.1. Relation associée aux mécanismes d'induction directe.

Sous la condition d'utiliser les opérations CONCURRENT' et ATTENDU' donnée en exemple, les mécanismes d'induction directe satisfont la relation suivante:

$$RID \qquad \sum_{j \in Ui} \ell FR_{ij} + (nSU_{i+1} - nSR_{i+1}) = \sum'_{j \in Ui} \ell FR^{o}_{ij}$$

Considérons deux niveaux de ressources dans un système hiérarchisé.



L'exécution d'une opération $S(\Delta_{ij}, u \varepsilon U_i)$ a pour effet:

si $\ell FR_{ij} = 0$ l'opération n'a aucune action sur les paramètres de la relation

si $\ell FR_{ij} > 0$ l'opération a les effets suivants:

$$\ell FR_{ij} \leftarrow \ell FR_{ij} - 1$$

$$nSU_{i+1} \leftarrow nSU_{i+1} + 1 \qquad \text{pour l'induction.}$$

L'exécution d'une opération $S(\Delta_{ij}, r \varepsilon R_i)$ a pour effet:

si $\ell FU_{ij} = 0$ l'opération a les effets suivants:

$$\ell FR_{ij} \leftarrow \ell FR_{ij} + 1$$

$$nSR_{i+1} \leftarrow nSR_{i+1} + 1 \qquad \text{pour l'induction.}$$

si $\ell FU_{ij} > 0$ l'opération n'a aucune action sur les paramètres de la relation.

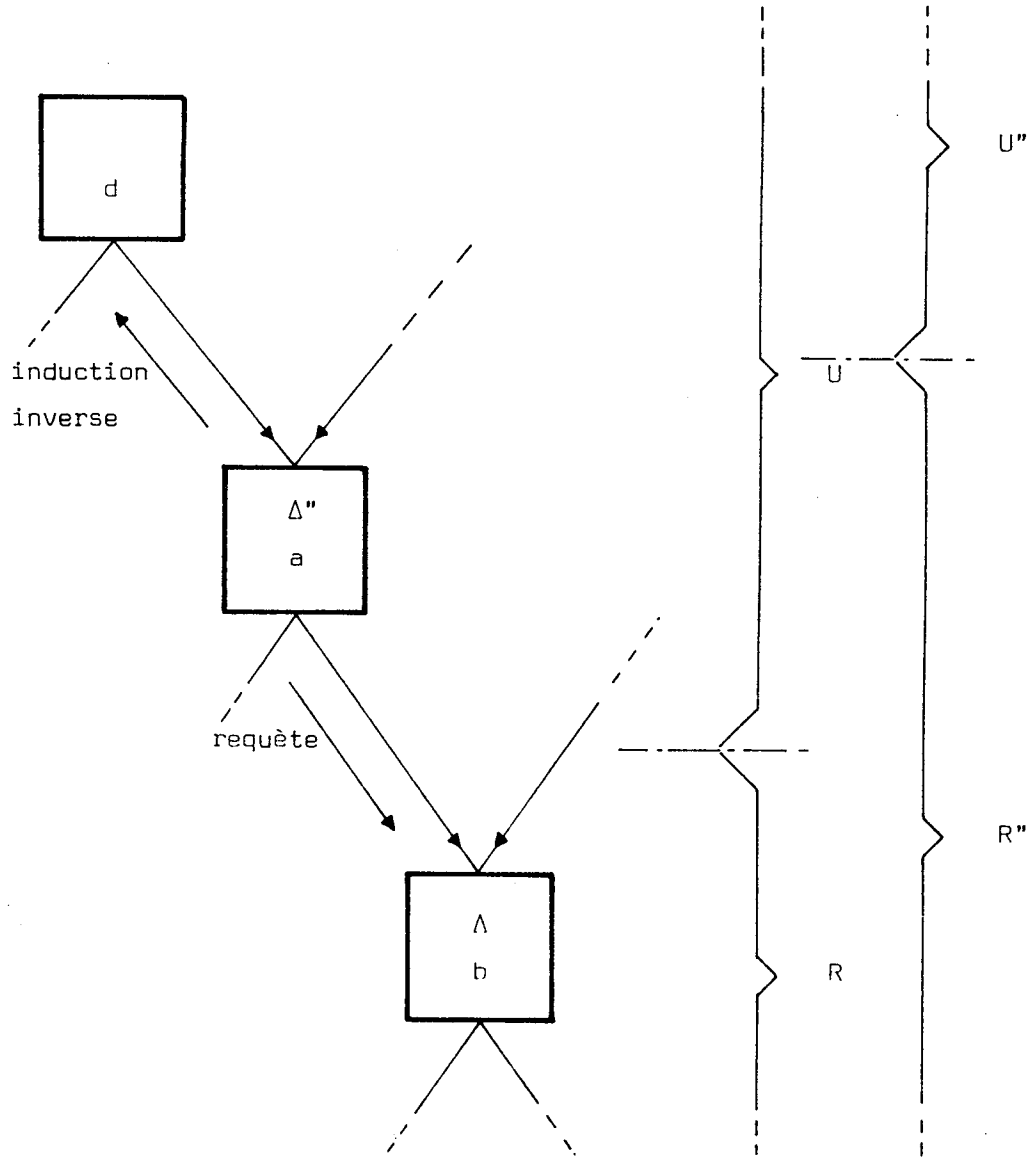
Dans tous les cas RID est conservée par l'exécution d'une opération S . A l'origine des temps la relation est vérifiée car $\forall j \varepsilon U_i \quad \ell FR_{ij} = \ell FR^o_{ij}$

4/2.2. Inductions inverses

L'induction inverse est un mécanisme qui consiste à propager vers les utilisateurs d'une ressource, l'information résultant des variations d'activité de cette ressource. Ce mécanisme est également généralement déclenché par les opérations DEMARRER et ARRETER indirectement contenues dans les opérations S .

Donnons à titre d'illustration la forme de ce mécanisme dans le cas de l' utilisation des opérations CONCURRENT' et ATTENDU' données à titre d'exemples.

Considérons une chaîne de trois articulations d, a, b telles que b repr
sente une ressource pour a qui représente elle-même une ressource de d



Si l'articulation a demande l'usage de la ressource b par une opération S(Δ, a ε U), l'exécution de cette opération sera, si nous nous plaçons dans le cas où cette ressource b n'est pas disponible:

```
opération S(Δ, a ε U)
    si FR = Ø alors
        CONCURRENT (Δ, a)
    sinon

opération ARRETER (a)
    S(Δ", d ε U")
```

```
opération CONCURRENT (Δ, a)
        RANGER (a, FU)
        ARRETER (a)
```

L'arrêt de l'articulation a entraîne que l'articulation d se porte candidate à l'association avec une ressource équivalente à a , si elle existe, par une opération S(Δ", d ε U").

Supposons maintenant que l'articulation b désire s'associer avec un nouveau candidat en exécutant l'opération S(Δ, b ε R).
L'exécution de cette opération sera, dans le cas où a est le seul candidat à cette association:

```
opération S(Δ, b ε R)
    si FU = Ø alors
    sinon
        ATTENDU (Δ, b)

opération DEMARRER (a)
    S(Δ", a ε R")
```

```
opération ATTENDU (Δ, b)
        PRENDRE (a, FU)
        ASSOCIER (a, b)
        DEMARRER (a)
```

Le démarrage de l'articulation a entraîne qu'elle se porte candidate à l'association avec un candidat de U" .
Nous remarquons que ces mécanismes d'induction inverse provoquent l'exécution récursive d'opérations de synchronisation pour le compte des articulations de niveau supérieur. Cette récursion s'arrête pour le premier cas

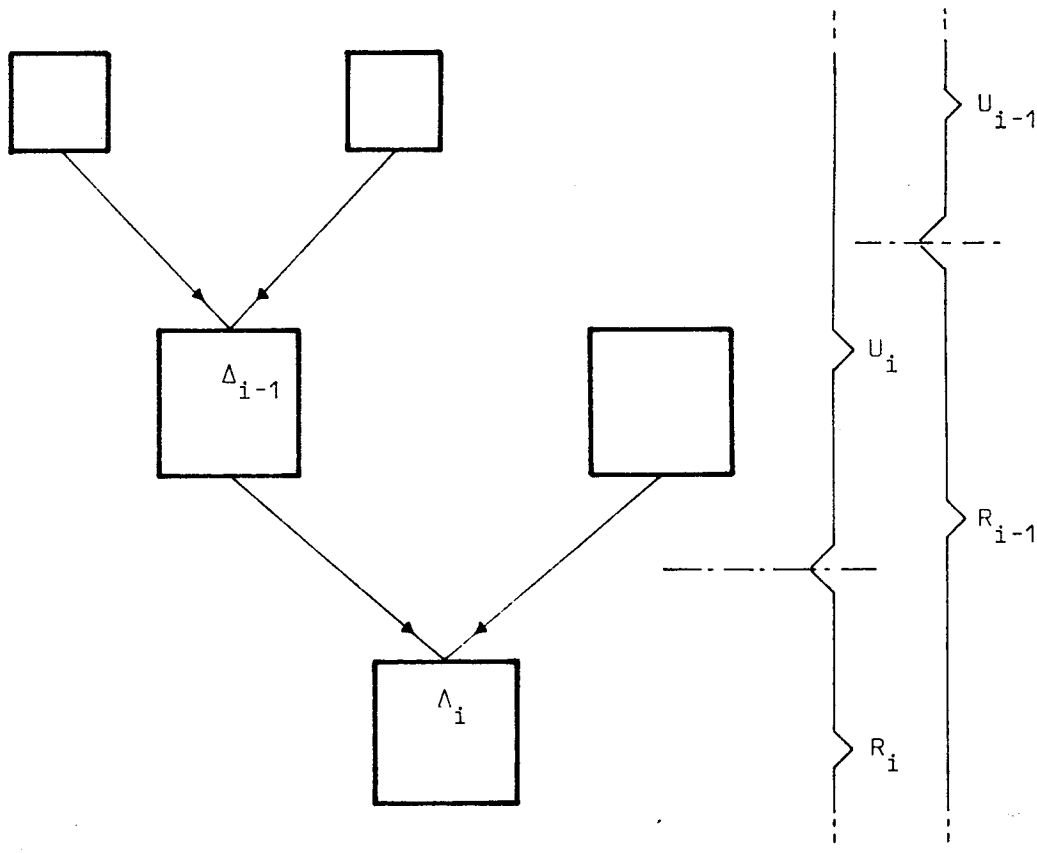
sur des articulations disposant d'autres ressources équivalentes ($FR^m \neq \emptyset$) et, pour le second cas sur des articulations n'ayant pas de candidat ($FU^m = 0$).

4/2.2.1. Relation associée aux mécanismes d'induction inverse

Sous la condition d'utiliser les opérations CONCURRENT' et ATTENDU' données en exemple, les mécanismes d'induction inverse satisfont la relation suivante:

$$RII \qquad \sum_{j \in Ui} (nSR_{i-1,j} - nSU_{i-1,j}) + \ell FU_i = \ell FU_i^0$$

Considérons deux niveaux de ressources dans un système hiérarchisé:



L'exécution d'une opération $S(\Delta_i, u \in U_i)$ a pour effet:

si $\ell FR_i = 0$ l'opération a les effets suivants:

$$\ell FU_i \leftarrow \ell FU_i - 1$$

$$nSU_{i-1,j} \leftarrow nS_{i-1,j} + 1 \qquad \text{pour l'induction}$$

si $\ell FR_i > 0$ l'opération n'a aucune action sur les paramètres de la relation.

L'exécution d'une opération $S(\Delta_i, r \in R_i)$ a pour effet:

si $\ell FU_i = 0$ l'opération n'a aucune action sur les paramètres de la relation,

si $\ell FU_i > 0$ l'opération a les effets suivants:

$$\ell FU_i \leftarrow \ell FU_i - 1$$

$$nSR_{i-1,j} \leftarrow nSR_{i-1,j} + 1 \qquad \text{pour l'induction.}$$

Dans tous les cas RII est conservée par l'exécution d'une opération S . A l'origine des temps la relation est vérifiée car $\ell FU_i = \ell FU_i^0$.

4/2.3. Conservation des formes d'opération de synchronisation

Nous remarquons dans les illustrations des deux types d'inductions précédentes que les formes de l'opération de synchronisation se conservent dans ces mécanismes d'induction. Une opération de la forme $S(\Delta, u)$ induit des opérations de la forme $S(\Delta', u')$ et une opération de la forme $S(\Delta, r)$ induit des opérations de la forme $S(\Delta', r')$. Ce fait a déjà été signalé dans [69] pour les opérations usuelles de synchronisation (p et v).

La justification de ce phénomène réside dans le fait que le démarrage d'une articulation ne peut entraîner que des démarrages d'articulations situées à des niveaux supérieurs et inférieurs. De même, l'arrêt d'une articulation ne peut entraîner que des arrêts de ces mêmes articulations. Cette correspondance des actions n'est plus vraie pour les articulations situées au même niveau que celle considérée. Nous verrons par exemple que l'arrêt d'une articulation peut entraîner le démarrage d'une autre articulation située au même niveau.

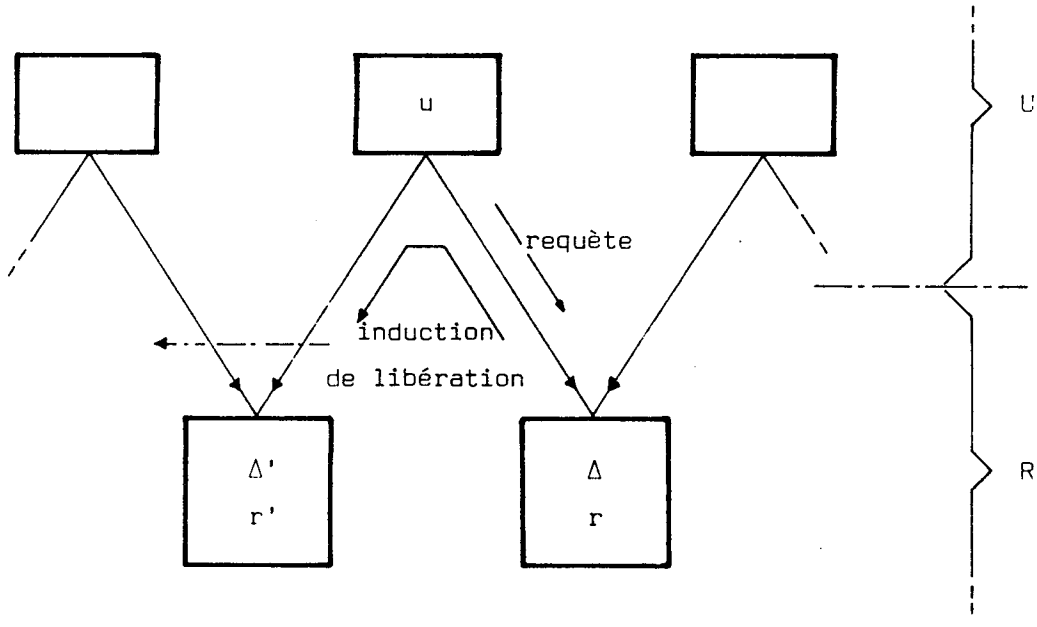
4/2.4. Inductions de libération

L'ensemble des ressources avec lesquelles un utilisateur dialogue à un instant donné peut comprendre des ressources requérables.

La mise en attente de cet utilisateur,occasionnée par l'indisponibilité de certaines des ressources qu'il sollicite,doit être transmise (induite) aux ressources qui lui sont déjà allouées de manière à ce qu'elles puissent chercher à s'associer à d'autres utilisateurs en attente.

Ce type d'induction sera appelé une induction de libération.

Donnons à titre d'illustration la forme de ce mécanisme dans le cas de l'utilisation des opérations CONCURRENT' et ATTENDU' données à titre d'exemple. Considérons deux articulations r et r' représentant des ressources d'un utilisateur u .



Si l'articulation u sollicite l'association avec r par une opération S(Δ, u),
L'exécution de cette opération sera, si nous nous plaçons dans le cas où l'articulation r est indisponible

```
opération  S(Δ, u)
        si  FR = Ø  alors
            CONCURRENT (Δ, u)
        sinon
            :

opération  ARRETER (u)
            :
        S(Δ', r' ε R)
            :
```

```
opération  CONCURRENT (Δ, u)
           RANGER (u, FU)
           ARRETER (u)
```

L'arrêt de l'articulation u entraîne que sa ressource requérable r' se porte candidate à l'association avec un autre de ses candidats.

4/2.5. Induction de réquisition

Lorsque dans schéma précédent la ressource r devient disponible il convie de solliciter à nouveau l'usage des ressources qui ont modifié leur allocation. Ce mécanisme est appelé une induction de réquisition.

Plaçons nous dans les hypothèses de l'illustration précédente.

Si l'articulation r sollicite l'association avec un nouveau candidat par une opération S(Δ, r ε R).
L'exécution de cette opération sera, dans le cas où u est le seul candida

```
opération  S(Δ, r ε R)
        si FU = Ø  alors
        sinon
            ATTENDU (Δ, r)
                :

opération DEMARRER (u)
            :
        S(Δ', u ε U)
```

```
opération  ATTENDU(Δ, r)
           PRENDRE(u, FU)
           ASSOCIER(u, r)
           DEMARRER(u)
```

Le démarrage de l'articulation u entraîne qu'elle se porte candidate à l'association avec sa ressource r' .

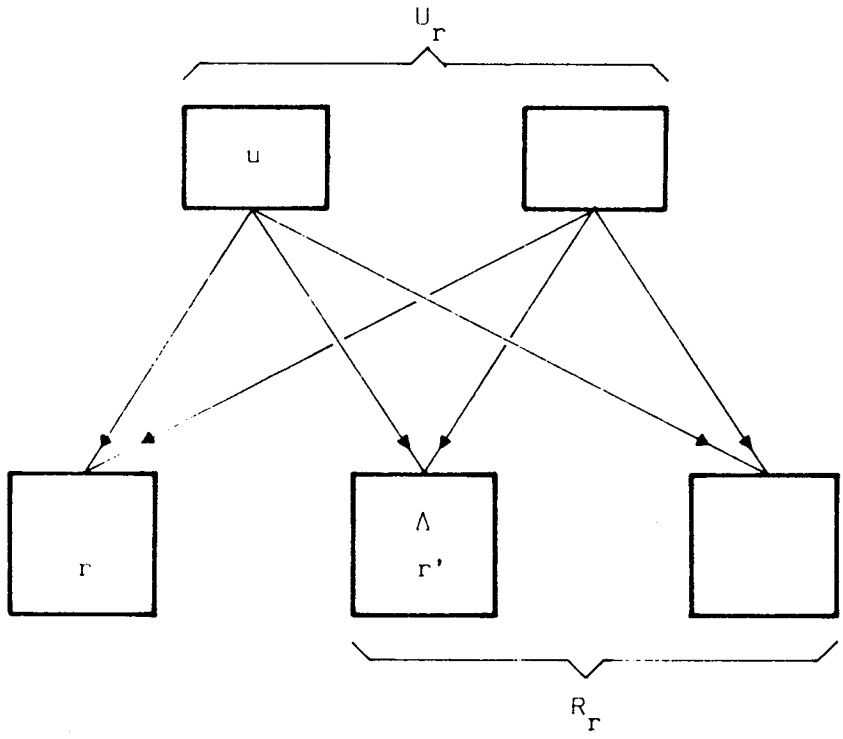
Nous remarquons que les inductions de libération et de réquisition ne conse vent pas la forme des opérations de synchronisation.

4/2.6. Relation associée aux mécanismes d'induction de libération et de réquisition

Sous la condition d'utiliser les opérations CONCURRENT' et ATTENDU' données en exemple, les mécanismes d'induction de libération et de réquisition satisfont la relation suivante

$$\text{RIR} \quad \forall r \in \text{Req} \qquad (nSR_r - nSU_r) + \sum_{r' \in R_r} \ell FU_{r'} = \sum_{r' \in R_r} \ell FU^{o}_{r'}$$

Considérons une ressource requérable r , soit U_r l'ensemble de ses utilisateurs possibles et soit R_r l'ensemble de leurs ressources, à l'exclusion de r elle-même.



L'exécution d'une opération $S(\Delta_{r'}, U \varepsilon U_r)$ a pour effet:

si $FU_{r'} = 0$ l'opération a les effets suivants:

$$\ell FU_{r'} \leftarrow \ell FU_{r'} + 1$$

$$nSR_r \leftarrow nSR_r + 1 \qquad \text{pour l'induction de libération}$$

si $\ell FU_{r'} > 0$ l'opération n'a aucune action sur les paramètres de la relation.

L'exécution d'une opération $S(\Delta_{r'}, r' \varepsilon R_r)$ a pour effet:

si $\ell FR_{r'} = 0$ l'opération n'a aucune action sur les paramètres de la relation,

si $\ell FR_{r'} > 0$ l'opération a les effets suivants:

$$\ell FU_{r'} \leftarrow \ell FU_{r'} - 1$$

$$nSU_r \leftarrow nSU_r + 1 \qquad \text{pour l'induction de réquisition.}$$

Dans tous les cas RIR est conservée par l'exécution d'une opération S . A l'origine des temps la relation est vérifiée car

$$\forall\ r' \in R_r \qquad \ell FU_{r'} = \ell FU^0_{r'}$$

4/2.7. Dissymétrie des mécanismes d'induction de libération et de réquisition

Les mécanismes d'induction de libération et de réquisition sont dissymétriques en ce sens qu'ils présupposent un comportement différent des ressource et des utilisateurs vis-à-vis de l'association.

Cette dissymétrie provient du fait que les ressources ont en général un désir constant de s'associer avec les utilisateurs alors que ces derniers ne sollicitent les ressources qu'en fonction de l'exécution du travail dont ils sont chargés.

4/3. ETAT COHERENT DU SYSTEME

Un état du système sera dit cohérent s'il répond aux conditions suivantes:

- Les relations liées à l'utilisation des mécanismes de synchronisation sont respectées. En particulier on en déduit, sous la conditic d'utiliser les opérations CONCURRENT' et ATTENDU', que toute articulation ayant sa file d'utilisateurs non vide à l'origine des temps ($\ell FU^o = \emptyset$) sera associée avec au moins un utilisateur à cet instant ($nA > 1$).

- Les relations liées aux mécanismes d'induction sont respectées. E particulier sous les mêmes conditions toute articulation associée avec un utilisateur sera soit candidate à ses propres ressources, soit associée avec elles.

En effet d'après la relation RID

$$\sum_U \ell FR_u = 0 \quad \Rightarrow \quad (nSU - nSR)_r = \sum_U \ell FR^o_u$$

des relations R21 et R22 on déduit

$$nSU - nSR = \ell FU - \ell FU^o - (\ell FR - \ell FR^o)$$

d'où

$$\sum_U \ell FR^o_u = \ell FU_r - \ell FU^o_r - (\ell FR - \ell FR^o_r)$$

Si nous admettons que $\ell FU^o_r = 0$, la relation peut s'écrire:

$$\sum_U \ell FR^o_u = \ell FU_r + (\ell FR^o_r - \ell FR_r)$$

qui signifie que la somme des longueurs initiales des files de ressources du niveau supérieur $\sum_U \ell FR^o_u$, c'est-à-dire le nombre total de ces utilisateurs est égal à la somme du nombre de candidat à l'usage de la ressource r (ℓFU_r) et du nombre d'exemplaires de cette ressource associée avec ces utilisateurs ($\ell FR^o_r - \ell FR_r$).

4/4. REALISATION PRATIQUE DES MECANISMES D'INDUCTION

L'ensemble des mécanismes d'induction n'est pas en général à considérer au niveau de chaque articulation. La nature des inductions à réaliser à parti d'une articulation donnée dépend de sa nature et de celle des articulations qui représentent ses ressources et ses utilisateurs. En particulier:

- Une articulation possédant un mécanisme d'allocation déclenché par les opérations de synchronisation doit recevoir les inductions directes et inverses qui la concernent. Elle doit recevoir égalemen celles de libération et de réquisition si elle est requérable.

- Lorsque la progression d'une articulation est rythmée par l'exécution d'opérations exécutées par une ressource principale (cas des I-processus), il est inutile de lui transmettre les inductions inver ses venant de cette ressource principale puisque son arrêt signifie celui de l'articulation considérée.

- Une articulation possédant un mécanisme d'allocation autonome ne propage pas les inductions directes ni n'émet d'inductions inverses puisque une telle articulation ne s'arrête jamais.

Ces mécanismes peuvent être réalisés de différentes manières:

- Soit par un mécanisme unique consultant des informations associées à chaque articulation et décrivant les inductions à réaliser (cas de: réalisations programmées).

- Soit par des extensions spécifiques des opérations de synchronisation dans le cas où celles-ci ont une réalisation particulière pour la gestion de l'allocation de chaque ressource (cas des réalisations technologiques).

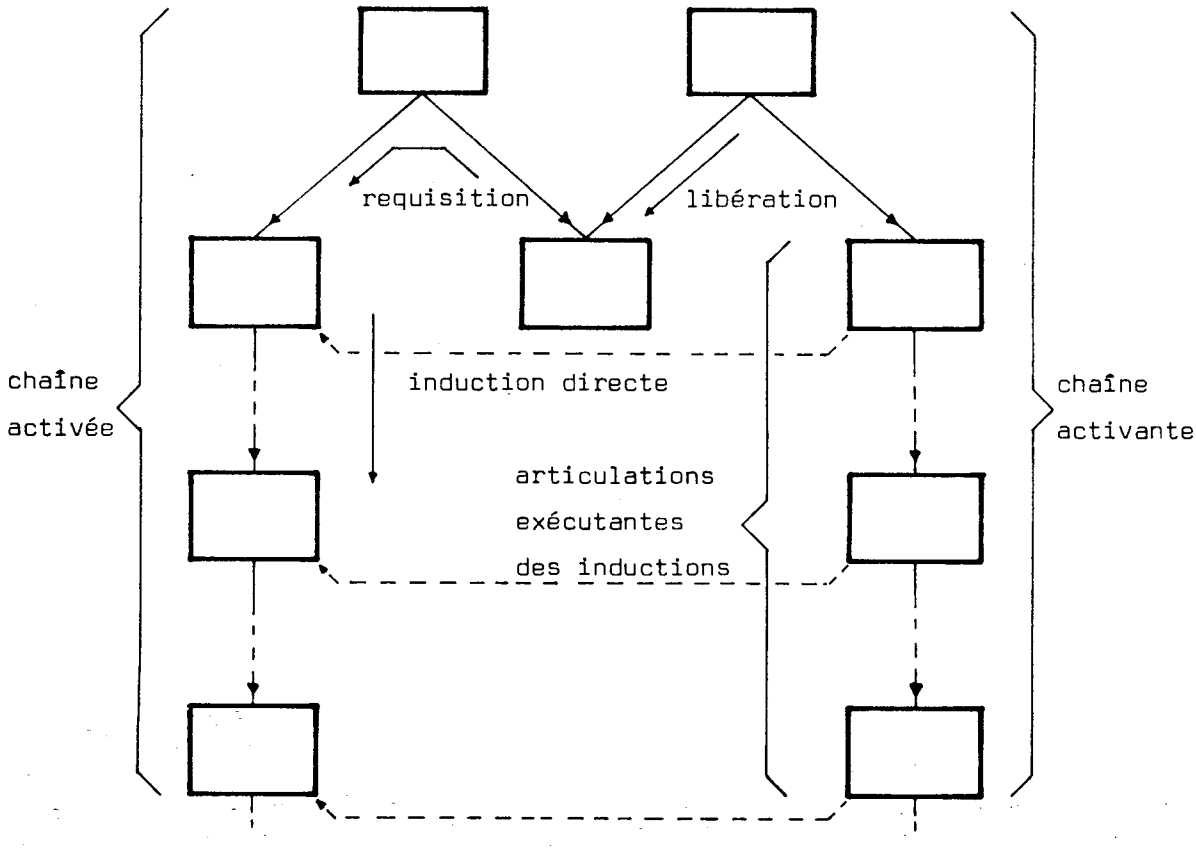
La réalisation pratique des mécanismes d'induction suppose que l'on a déterminé les exécutants des opérations de synchronisation induites.

Plusieurs solutions sont possible:

- L'exécution des inductions est réalisée par l'articulation qui déclenche ces mécanismes. Cette solution suppose que cette articulatic ait la connaissance de la structure du sous-ensemble de la hiérarchie dans laquelle les inductions qu'elle déclenche peuvent se propager.

- L'exécution des inductions est réalisée par les articulations utilisatrices,ou ressources,de l'instruction qui déclenche ces mécanismes et situées au même niveau que celles sur lesquelles portent les inductions. Lorsque le but d'un tel mécanisme est le démarrage d'une chaîne d'articulations, que nous appellerons une chaîne activée, les articulations exécutantes forment ce que nous appellerons la chaîne activante.

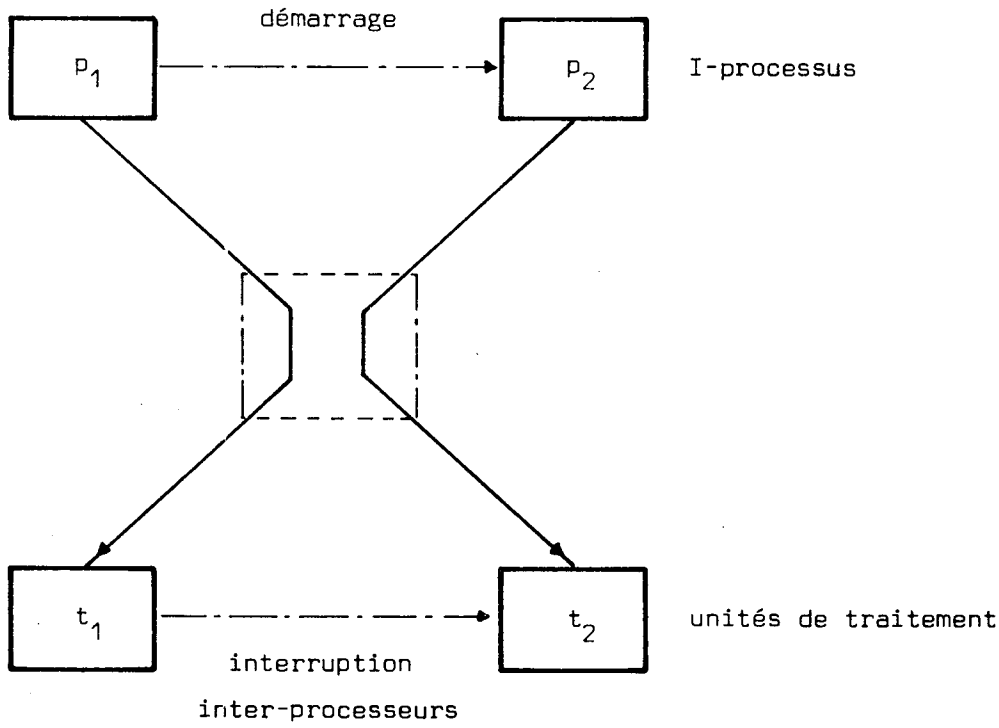
Exemple:



Exemple:

Considérons un multiprocesseur constitué par la connexion à une mémoire commune de deux unités de traitement t1 et t2 .

Supposons d'autre part que l'unité de traitement t2 soit arrêtée et que le I-processus p1 interprété par t1 demande le départ en parallèle d'un travail représenté par le I-processus p2 . Le I-processus p1 , qui peut être le système d'exploitation, détermine que p2 doit s'exécuter sur t2 . Le démarrage effectif de ce I-processus p2 demande, d'une part, qu'il soit initialisé, ce qui peut être fait par p1 s'exécutant sur t1 , et d'autre part que l'unité de traitement t2 soit démarrée par un mécanisme d'induction directe. Cette induction déclenchée par l'opération de synchronisation exécutée par p1 est généralement réalisée sous la forme d'une interruption inter-unité de traitement émise par t1 à l'attention de t2 . Cette interruption démarre l'unité t2 lorsqu'elle est reçue par ses circuits de contrôle.



- L'exécution des inductions est réalisée par les mécanismes d'alloca tion des articulations concernées. Ce mode de fonctionnement suppose donc que ces mécanismes soient toujours actifs, c'est-à-dire qu'ils soient réalisés de manière différente des mécanismes d'interprétation de ces articulations.
Ex.: Nous verrons que dans certaines organisations de multiprocesseur la gestion de ceux-ci est réalisée par programme.

4/5. REQUERABILITE OBLIGATOIRE D'UNE RESSOURCE

Lorsque le rythme avec lequel une ressource satisfait les requêtes qui lui sont présentées ne peut varier, il est indispensable que son allocation à un utilisateur ne soit réalisée que lorsque l'articulation qui représente cet utilisateur peut progresser, c'est-à-dire après que toutes les autres ressources qu'il a sollicitées lui aient été allouées.

Il faut également qu'une induction de libération soit déclenchée vers cette ressource dès que l'une de ces autres ressources vient à faire défaut.

Les ressources de ce type sont en général principales et une articulation en utilise au plus une.

Ex.: Une unité de traitement ne peut souvent pas spontanément attendre pour exécuter une instruction que l'un de ses I-processus ait obtenu une autre ressource.

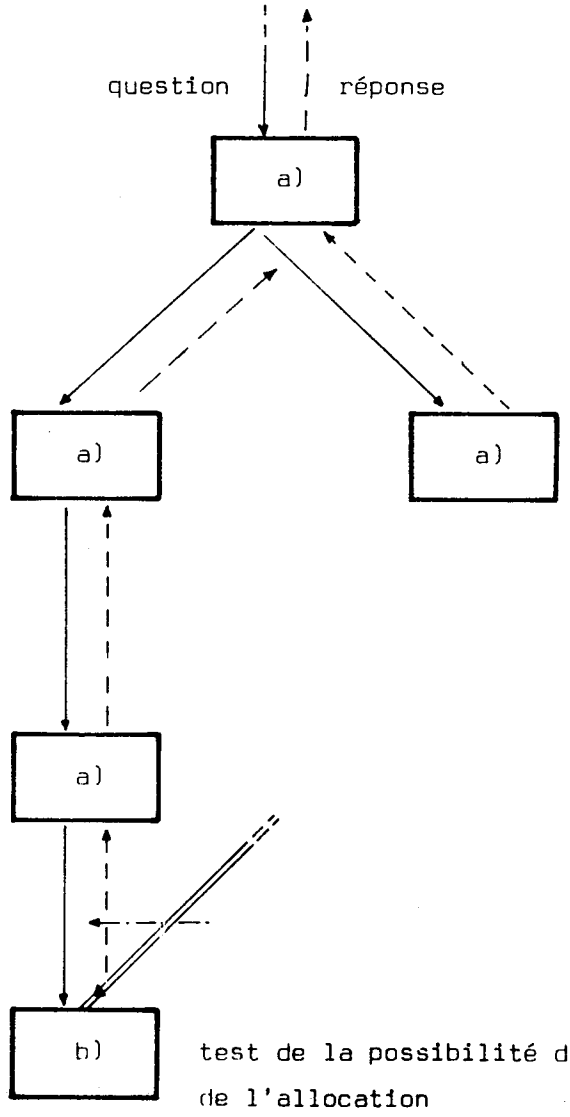
4/6. MESURE DE LA DISPONIBILITE DES RESSOURCES

Une articulation appartenant à une ressource est disponible si:

a) elle n'est pas allouée et n'a aucun candidat,
b) elle est requérable et allouée à un utilisateur de priorité inférieure à l'éventuel candidat.

Le cas b) implique que la demande de disponibilité d'une ressource soit accompagnée de la priorité d'un éventuel candidat.

La détermination de cette disponibilité peut se faire récursivement en parcourant vers la racine les articulations correspondant au cas a) jusqu'à rencontrer une articulation allouée que l'on doit alors tester comme correspondant au cas b) . A chaque étape de cette interrogation, les articulatior correspondant au cas a) transmettent la question à leurs propres ressources en l'accompagnant de leur propre priorité.



test de la possibilité d'une commutation de l'allocation

La détermination de la disponibilité de la ressource r relativement à un éventuel candidat de priorité p sera déterminée par:

```
booléenne fonction DISPONIBILITE (r, p)
        si AS_r = Ø alors
                pour r' ← toutes les ressources de r boucler
                tant que DISPONIBILITE (r', priorité(r))
                sinon DISPONIBILITE ← faux
                DISPONIBILITE ← vrai
        sinon
                si p > max(priorité(proj_U (AS_r))) alors
                        DISPONIBILITE ← vrai
                sinon
                        DISPONIBILITE ← faux
```

Pour qu'une telle information ait un sens, sa détermination doit entraîner un blocage de la commutation des articulations constituant la ressource considérée. Ce blocage doit être levé après la prise d'une décision découlant de ce test.

Remarque: Un tel mécanisme revient à simuler, pour une sollicitation donnée les mécanismes d'induction directe et inverse correspondant à une évolution possible du système hiérarchisé.

4/7. ARTICULATIONS MULTIPLES

La gestion optimisée de l'allocation des ressources équivalentes, regroupées par une articulation multiple, suppose la connaissance, par le module d'allocation, de la disponibilité de chacune de ces ressources équivalentes.

Cette information de disponibilité est utilisée dans l'opération PRENDRE appelée indirectement par l'opération S .

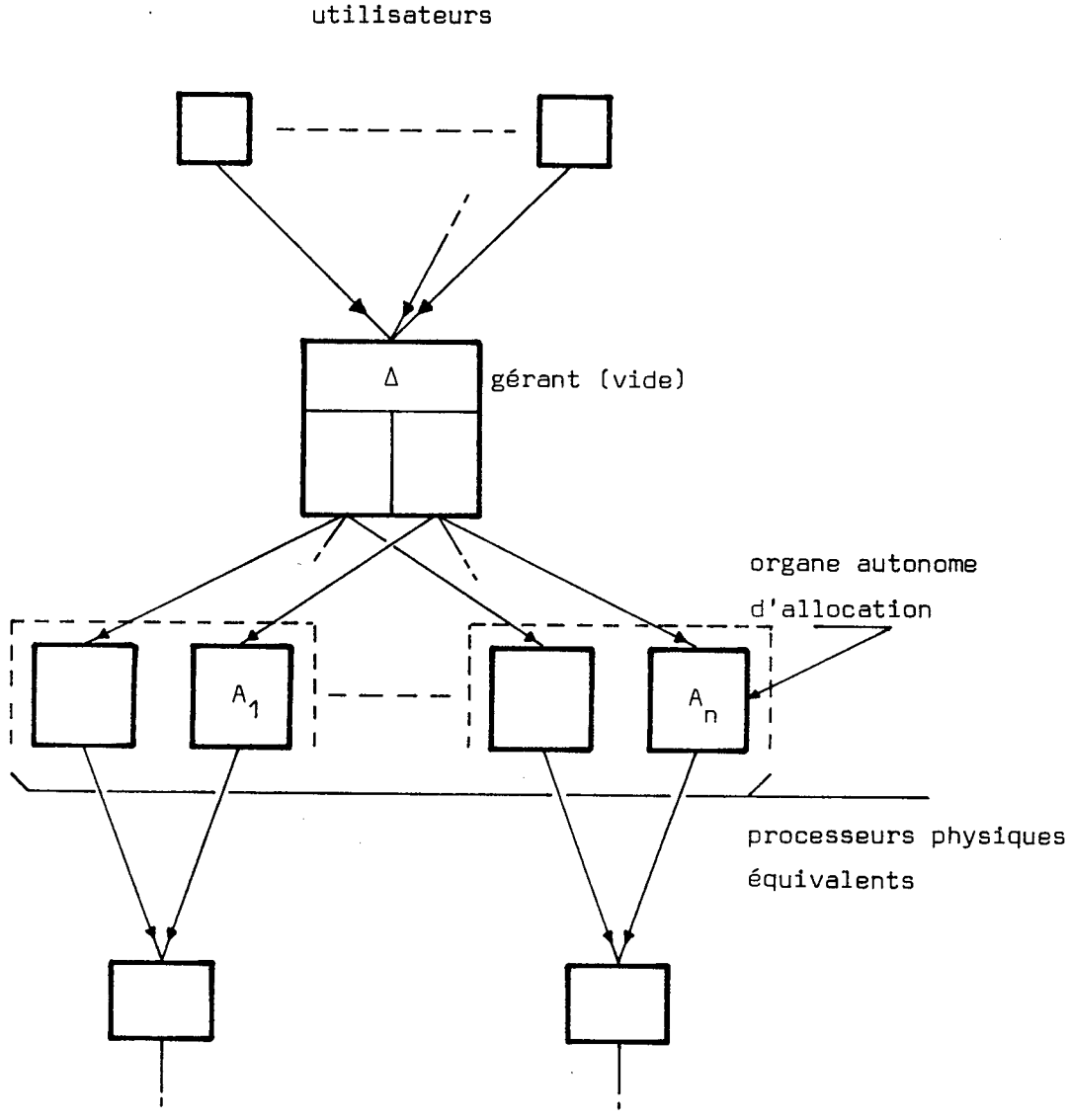
```
opération S(Δ, u)
   si  FR  =  Ø  alors
   sinon
        ATTENDU (Δ, u)

opération PRENDRE (r, FR)
      pointeur p
      p ← CHOIX (FR)
      r ← EXTRAIRE (p↑ , FR)
```

```
opération  ATTENDU (Δ, u)
           PRENDRE (r, FR)
           ASSOCIER (r, u)
```

```
pointeur fonction  CHOIX (FR)
  pour  r' ← élément(FR)  boucler
    tant que  ¬DISPONIBILITE (r', priorité(u))
    sinon  CHOIX ← ↓r'
```

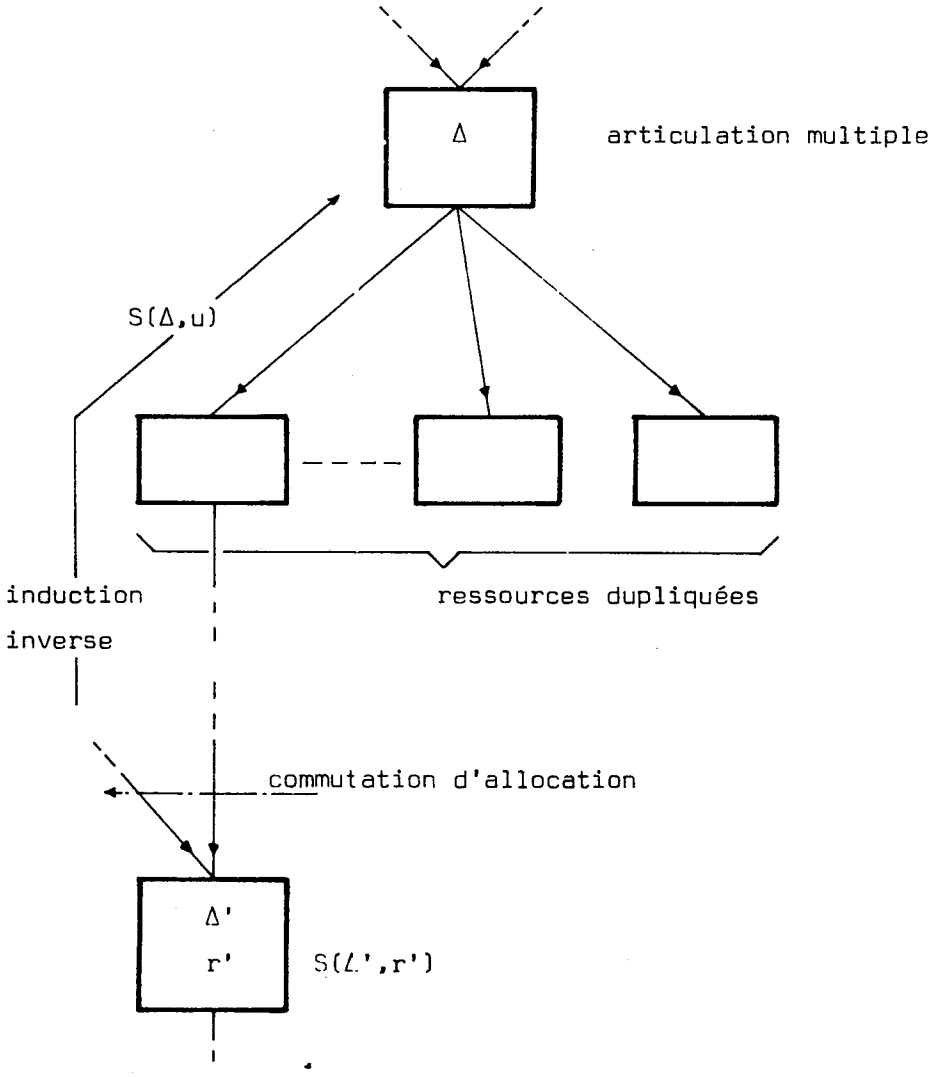
Une autre solution consiste à munir chaque ressource équivalente d'un mécanisme d'allocation autonome propre à cette ressource. Dans ce cas une opération S(Δ, u) ne correspond qu'au chargement de u dans la file FU de . Les articulations représentant les ressources équivalentes sont toujours candidates à leurs propres ressources et elles ne pourront exécuter les opérations S(Δ, r) pour solliciter de nouveaux utilisateurs que lorsqu'elles pourront progresser c'est-à-dire être disponibles. Cette solution est utilisée pour les configurations multi-processeurs obtenues à partir de machines munies individuellement d'un hyperviseur microprogrammé (T1600, GEOPROCESSEUR)



auto-allocation des ressources équivalentes.

Remarques:

- Dans tous les cas l'indisponibilité d'une ressource équivalente entraîne la cessation de son association avec son utilisateur courant. Cette indisponibilité résulte de la commutation de l'allocation de l'une des articulations constituant la ressource. Cette commutation déclenche une induction inverse qui provoque la re-soumission de cet utilisateur à la ressource multiple considérée par une opération S(Δ, U).

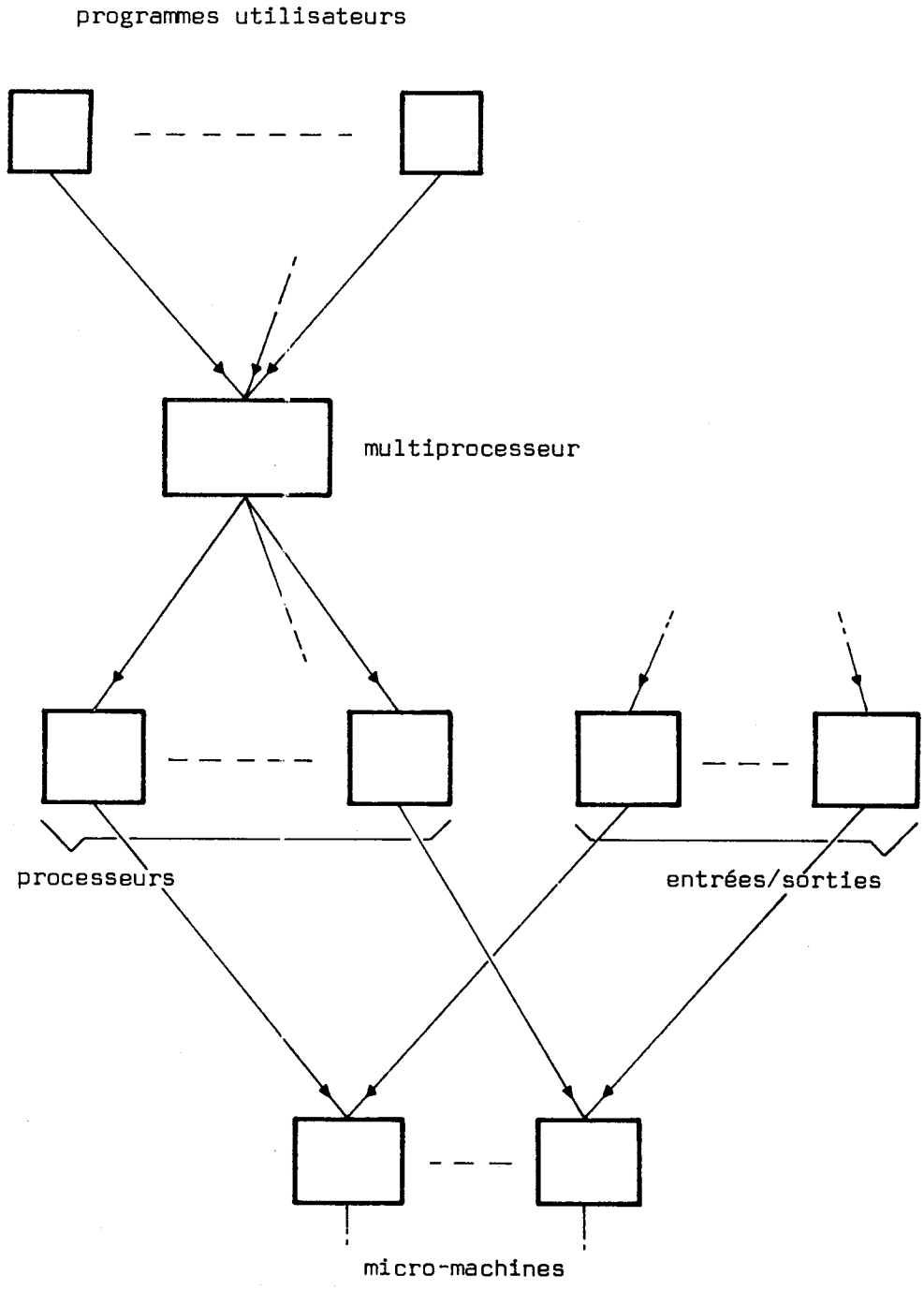


- Dans la majorité des cas pratiques, la disponibilité des ressources dupliquées est supposée toujours réalisée,

  . soit que ces ressources sont effectivement toujours disponibles,
  . soit parcequ'elles sont simplement supposées toujours disponibles:
    - parceque la durée de leurs indisponibilités est trop faible pour qu'il puisse en être tenu compte par une réallocation des utilisateurs,
    - soit parce que la prise en considération des indisponibilité entraînerait un coût supérieur à celui résultant de la perte de performance qui en résulte.

ex.:
Considérons le cas d'un multi-processeur dans lequel chacun des processeurs est réalisé à l'aide d'une micro-machine assurant également les fonctions d'entrées/sorties.

Chaque échange élémentaire entraîne une commutation de l'allocation de l'une des micro-machines de sa fonction de processeur à celle qui consiste à assurer cet échange. Cette commutation rend donc momentanément indisponible l'une des ressources processeur. Elle est toutefois trop courte pour qu'il soit raisonnable d'envisager de modifier l'allocation des programmes utilisateurs.


programmes utilisateurs
multiprocesseur
processeurs
entrées/sorties
micro-machines

4/9. FEUILLES DE LA HIERARCHIE

Deux solutions permettent de terminer le système au niveau des feuilles de la hiérarchie:

- Les feuilles de la hiérarchie représentent des utilisateurs élémentaires (I-processus ou processus) interconnectés par des RSC qui leur servent à échanger des informations nécessaires au démarrag et à l'arrêt des différents travaux à exécuter par ces utilisateurs.

- Les feuilles de la hiérarchie représentent des utilisateurs réduit à être de simples noms de travaux exécutables par les ressources représentées par les articulations de la couche immédiatement inférieu Les articulations représentant les noms de travaux doivent pouvoir être crées et détruites dynamiquement par les articulations du nivea inférieur de manière à ce que leur existence coïncide avec celle d' un travail.

Ces niveaux sont généralement réalisés par programmation. Les mécanismes ci-dessus correspondent à la réalisation des primitives utilisées pour lanc et contrôler la progression des travaux. Ce sont eux qui sont à l'origine de la plupart des évolutions du système.

Remarque:
Dans la suite de ce travail nous utiliserons la première solution sans tout fois mettre en évidence toutes les RSC impliquées dans ces mécanismes. Nous dirons simplement que les articulations terminales peuvent se soumettr mutuellement des travaux à exécuter.

4/10. ETAT INITIAL D'UN SYSTEME HIERARCHISE

Nous supposerons que les systèmes que nous décrirons sont clos, c'est-à-dir qu'ils contiennent ce qu'il est habituellement convenu d'appeler le milieu extérieur.

A l'origine des temps, les articulations peuvent être:

- actives, c'est-à-dire associées avec un utilisateur,
- en attente sur des RSC,
- en attente d'un travail, dans le cadre des conventions que nous avons établies au sujet des feuilles de la hiérarchie.

L'état initial correspondant du système doit être cohérent.

4/11. ARRET IMPROMPTU DE L'EVOLUTION D'UN PROCESSUS

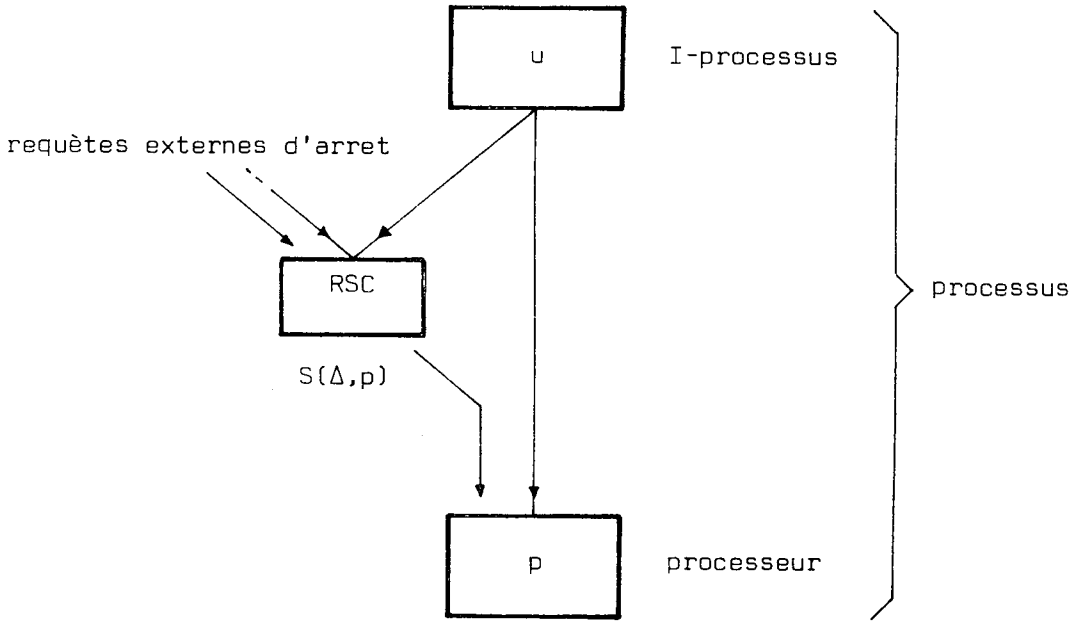
La possibilité d'arrêter de manière impromptue l'évolution d'un processus par un tiers est communément rejetée [19] .

La raison de ce choix tient dans le fait qu'un processus qui exécute un certain travail ne peut être arrêté n'importe où si l'on désire que l'état final du système soit cohérent. En particulier, un arrêt nécessite souvent l'exécution d'une séquence particulière qui ramène le système dans un état de repos. La nature de cette séquence dépend en général de l'instant, dans l'évolution du processus, où cet arrêt est décidé.

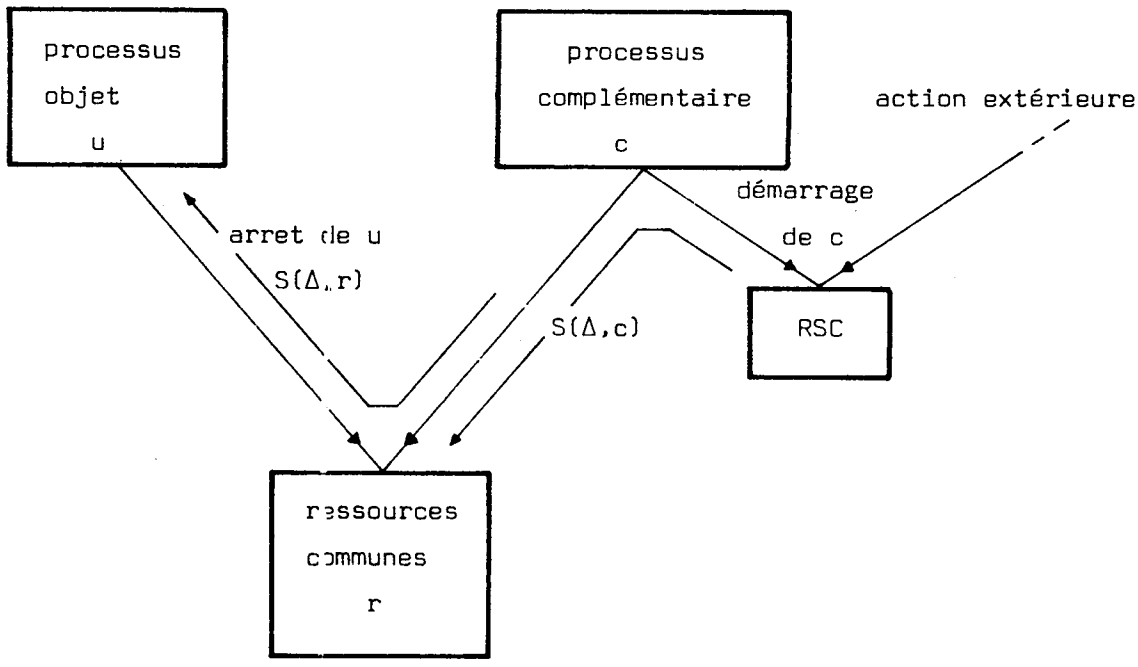
Remarque:
Les processeurs, qui sont souvent des ressources requérables, arrêtent de manière impromptue les I-processus qu'ils exécutent. Cet arrêt se situe toujours entre l'exécution de deux instructions pour que l'état du processe soit nul. Les I-processus peuvent en général émettre des requêtes particulières pour momentanément interdire leur arrêt impromptu.

Nous pouvons envisager deux solutions à ce problème:

- Le processus considéré peut être décomposé en un couple I-processu processeur tel que le I-processus se compose d'une suite d'instructions correspondant à des phases ininterruptibles de travail. L'ar impromptu consistera en une dissociation du couple I-processus-processeur. Il se situera donc en un point interruptible.



- Une autre solution consiste à associer au processus susceptible d'être interrompu un processus "complémentaire" qui au vu de l'état des ressources utilisées par ce premier processus sera compétent pour arrêter convenablement son travail. L'arrêt peut être simplement obtenu en démarrant le processus complémentaire qui, plus prioritaire, réquisitionne les ressources communes et provoque l'arrêt du processus objet.



DEUXIEME PARTIE

REPRESENTATION DANS LE FORMALISME DES SYSTEMES HIERARCHISES DES PRINCIPAUX MECANISMES UTILISES PAR LES MACHINES INFORMATIQUES

1. Gestion des organes de traitement
2. Adressage des ressources, mémoires virtuelles
3. Mécanismes d'échange de l'information
4. Exécution parallèle d'un I-processus.

1 - GESTION DES ORGANES DE TRAITEMENT

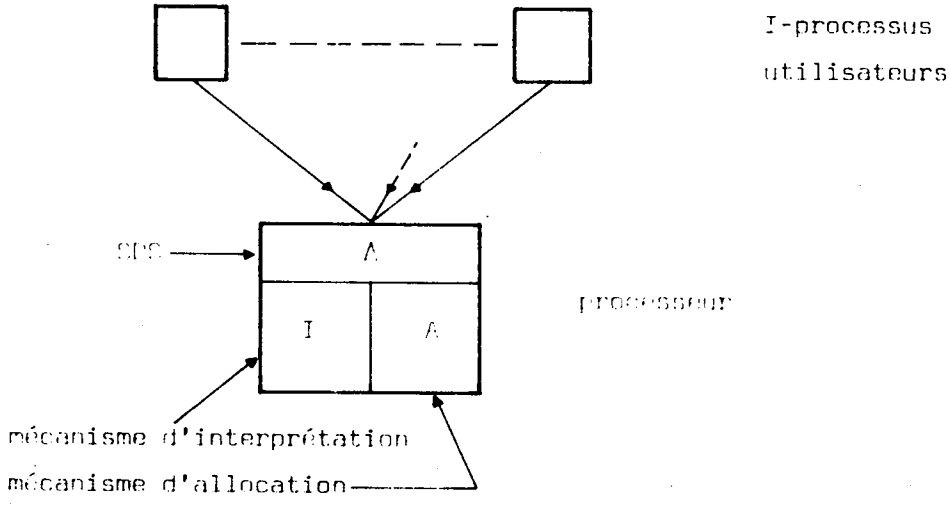
1/1. GESTION DE L'ALLOCATION D'UN PROCESSEUR A SES I-PROCESSUS UTILISATEURS

Très fréquemment le travail soumis à l'exécution d'une machine informatique se décompose en plusieurs processus représentés par autant de I-processus.

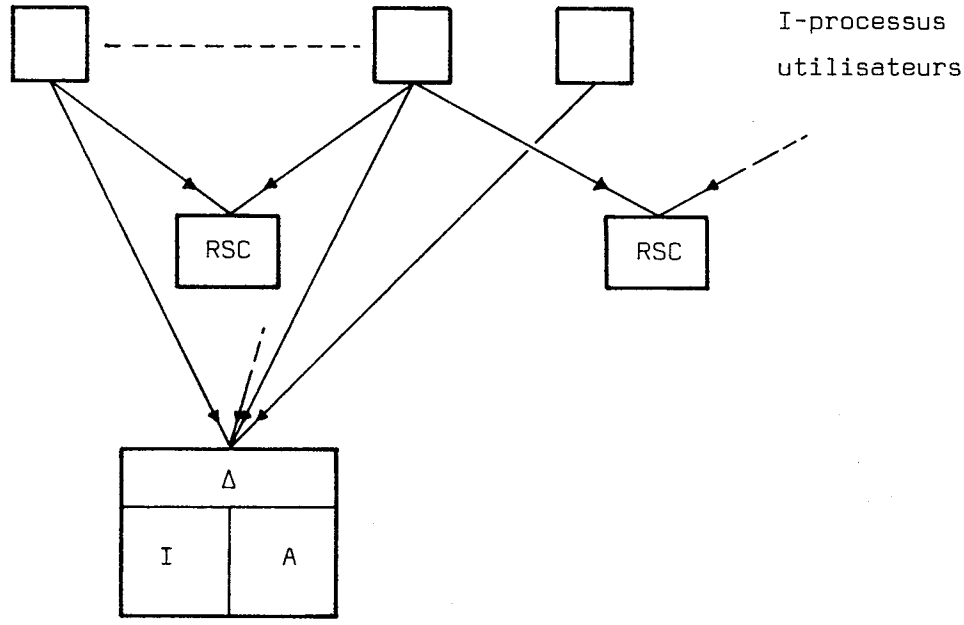
Les ressources offertes par cette machine devront donc être partagées entre ces utilisateurs soit temporellement (unité de traitement, organes d'entrée-sortie), soit statiquement (mémoires).

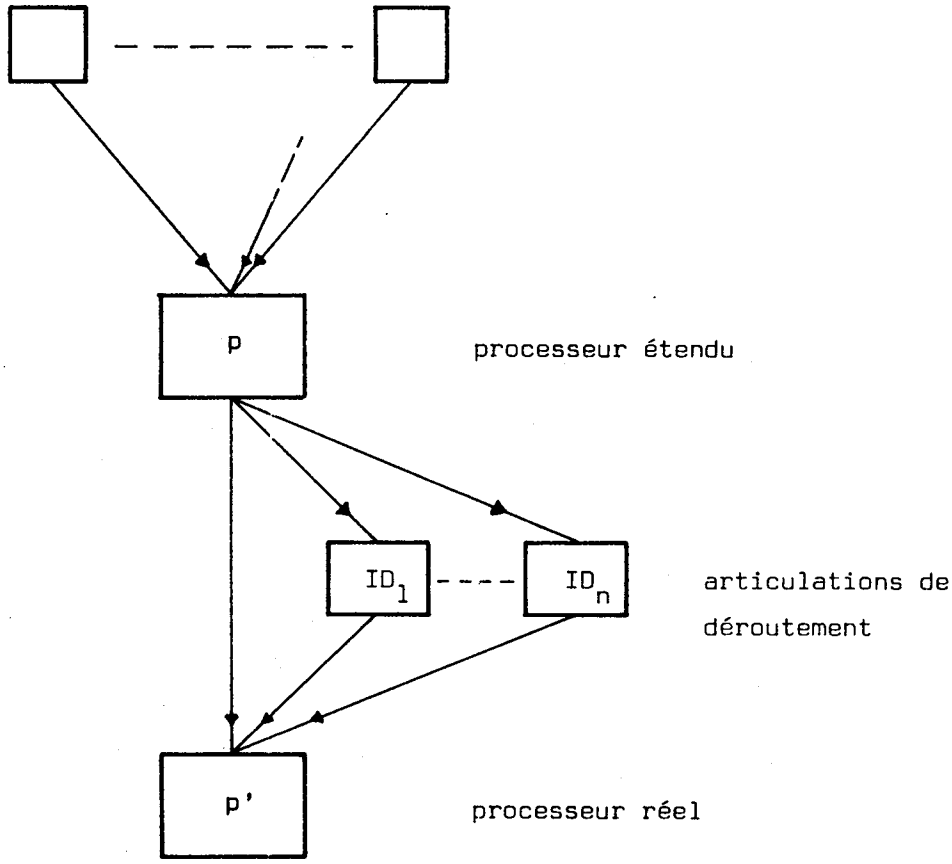
Considérons, dans un premier temps, le cas d'une machine informatique munie d'un processeur unique, c'est-à-dire non représenté par une articulation multiple. Ce processeur devra s'allouer à chacun des I-processus utilisateurs d'une manière propre à optimiser certains critères de fonctionnement (tels que le débit moyen de travail fourni ou le temps d'exécution de chacun des I-processus). Pour cette raison, l'articulation qui représente ce processeur doit être munie d'un mécanisme d'allocation très évolué qui peut être réalisé de différentes manières.

- par des programmes,
- dans le processeur lui-même (par micro-programmation),
- par des organes spécialisés.



L'allocation de ce processeur aux I-processus utilisateurs peut être modifiée à la suite de requêtes issues soit des I-processus eux-mêmes, soit du milieu extérieur. Ces requêtes peuvent être vues, dans un premier temps, comme des actions sur des RSC utilisées par les I-processus pour communiquer.




p
processeur étendu
ID1
IDn
articulations de
déroutement
p'
processeur réel

1/1.1. Déroutements

Il est fréquent que certains cas de l'exécution des instructions soumises à un processeur nécessitent soit la prise d'une décision au niveau du I-processus qui a soumis cette instruction (en général pour le traitement de erreurs telles que les débordements d'opérations arithmétiques ou les code opération invalides), soit l'extension des possibilités normales de ce processeur (en général l'extension de l'ensemble des instructions qu'il reconnait en associant, à un code opération inutilisé, l'exécution d'un algorithme particulier).

Ces ressources additionnelles sont en général réalisées à l'aide de programmes particuliers s'exécutant sur le processeur réel p' . Un processeur étendu p correspond au regroupement de la ressource offerte par p' et des ressources additionnelles en une articulation multiple (non dupliquée). Le processeur réel p' doit donc pouvoir s'allouer soit aux I-processus utilisateurs par l'intermédiaire du processeur étendu, soit aux articulations qui représentent les ressources additionnelles, que nous appellerons les articulations de déroutement. Le processeur réel p' déc de lui-même de la modification de son allocation au cours de l'exécution d instructions qui lui sont soumises par les I-processus utilisateurs. La soumission de ces instructions peut être vue comme déclenchant un mécanism d'induction directe. Vis-à-vis du mécanisme d'allocation du processeur réel p' les articulations de déroutement sont plus prioritaires que celle qui représentent le processeur étendu.

Ce mécanisme d'allocation doit répondre à certaines conditions, en effet:

- la commutation de l'allocation de p' vers les articulations ID_i au cours de l'exécution d'une instruction, doit se faire avant que ce processeur n'ait abordé l'exécution de l'instruction suivante.

- il convient de fournir à l'articulation de déroutement appelée l'identité et les privilèges du I-processus utilisateur qui a soumis l'instruction à l'origine de la commutation de l'allocation de p'

Remarques:

- Il n'est pas en général prévu que les programmes constituant les articulations de déroutement puissent utiliser les possibilités supplémentaires du processeur étendu. Cela signifie qu'ils ne pourront pas solliciter le service des ressources additionnelles.

- La file FU_i de la SDS de gestion de chaque articulation de déroutement ne peut contenir au plus que le nom d'un seul I-processus utilisateur. En effet, le démarrage d'une articulation de déroutement empêche l'interprétation, moins prioritaire, des I-processus utilisateurs au travers du processeur étendu. Tout nouvel appel est donc impossible puisque seuls ceux-ci peuvent les émettre.

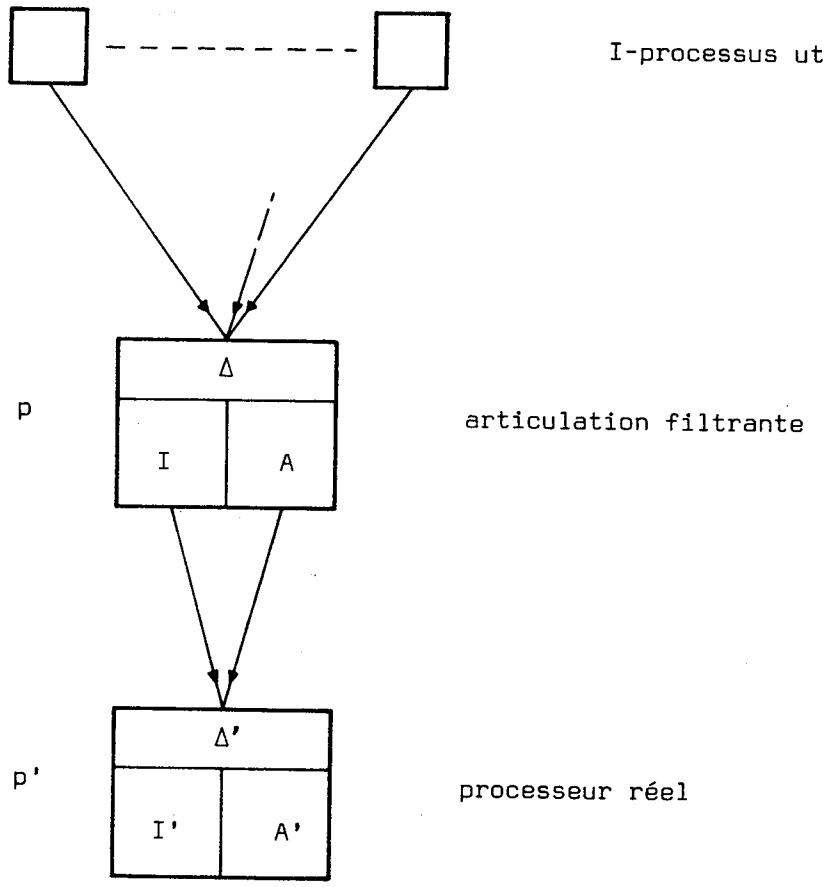
- Ces remarques entraînent que la commutation du processeur réel p' s'effectue toujours entre le processeur étendu et les articulations de déroutement.

1/1.2. Allocation programmée du processeur

Lorsque le mécanisme d'allocation A de l'articulation représentant le processeur est réalisé par un programme, il constitue ce qu'il est convenu d'appeler l'allocateur de travaux du système d'exploitation de cette machine.

Ce type de réalisation signifie que le processeur utilisé par les I-processus utilisateurs est représenté par une articulation transparente, ou filtrante, tandis que la ressource processeur proprement dite est réalisée par

une articulation p' située à un niveau inférieur d'où elle peut exécuter le programme correspondant à ce mécanisme d'allocation.



Le processeur réel p' pourra donc être alloué à l'un des deux utilisateurs suivants:

- le mécanisme d'interprétation I de p,
- le mécanisme d'allocation A de p .

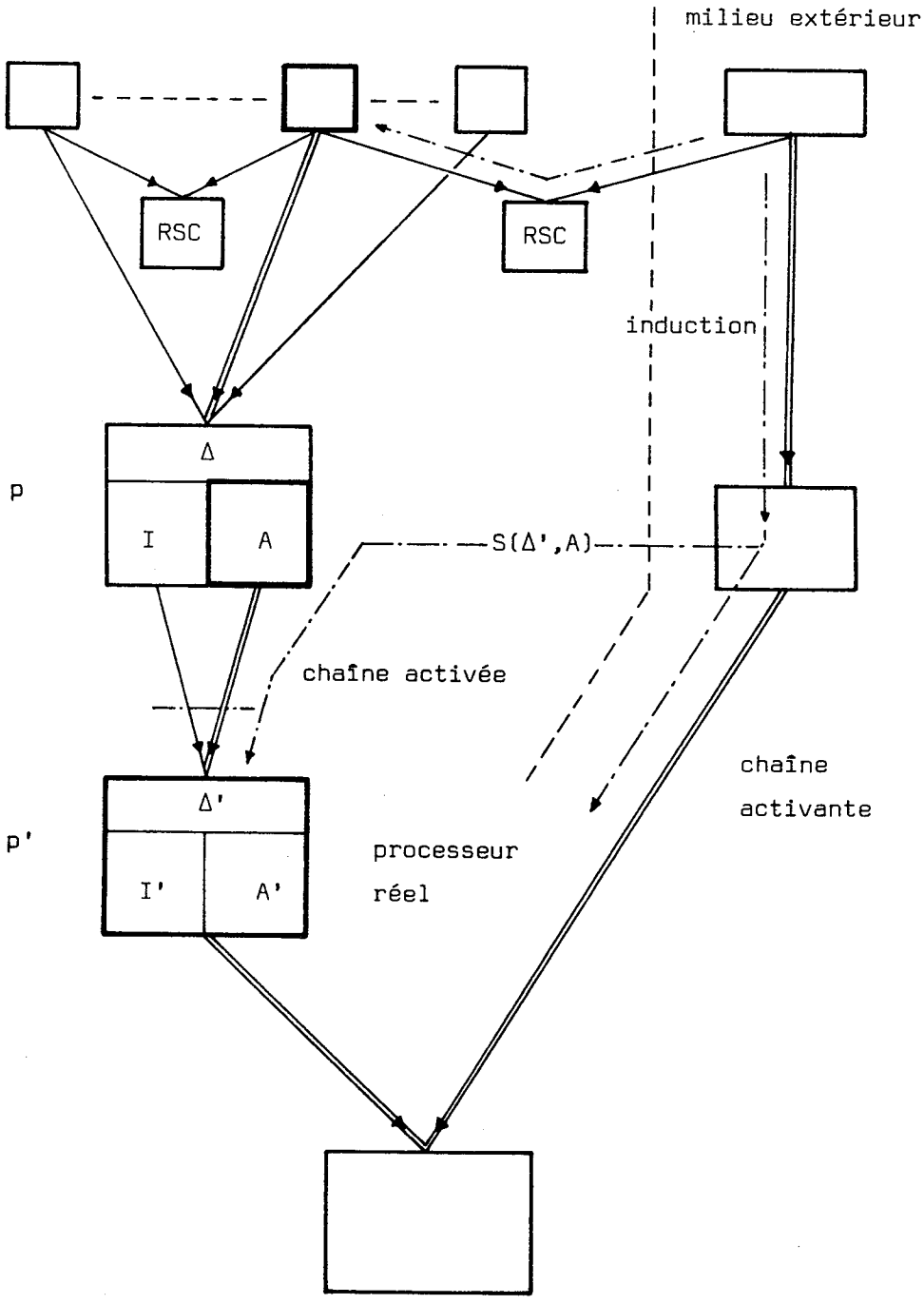
L'articulation p est filtrante en ce sens qu'elle offre à ses utilisateurs un sous-ensemble des instructions que reconnait p' (les instructions du mode utilisateur). Cette organisation sous-entend que l'articulation p'

dispose elle-même d'un mécanisme plus élémentaire d'allocation A' capable de l'allouer à l'un de ces utilisateurs. Nous verrons [annexe 3] comment ce mécanisme est réalisé au niveau du processeur lui-même.

Ce mécanisme doit commuter l'allocation du processeur réel chaque fois que le mécanisme d'allocation A doit intervenir pour modifier l'allocation des I-processus utilisateurs,

- soit sur une requête des I-processus utilisateurs eux-mêmes,
- soit sur une requête extérieure.

Ces requêtes provoquent l'exécution d'un mécanisme particulier voisin d'une induction directe. En effet, toute action d'allocation de p demandera que le mécanisme d'allocation A de cette articulation soit actif d'où que celui-ci se porte candidat à l'association avec p' par une opération $S(\Delta', A)$. L'articulation p' étant requérable et la priorité de A supérieure à celle de I entraînera la commutation d'allocation désirée.


milieu extérieur
RSC
RSC
induction
Δ
p
I
A
S(Δ',A)
chaîne activée
chaîne
activante
Δ'
p'
processeur
réel
I'
A'

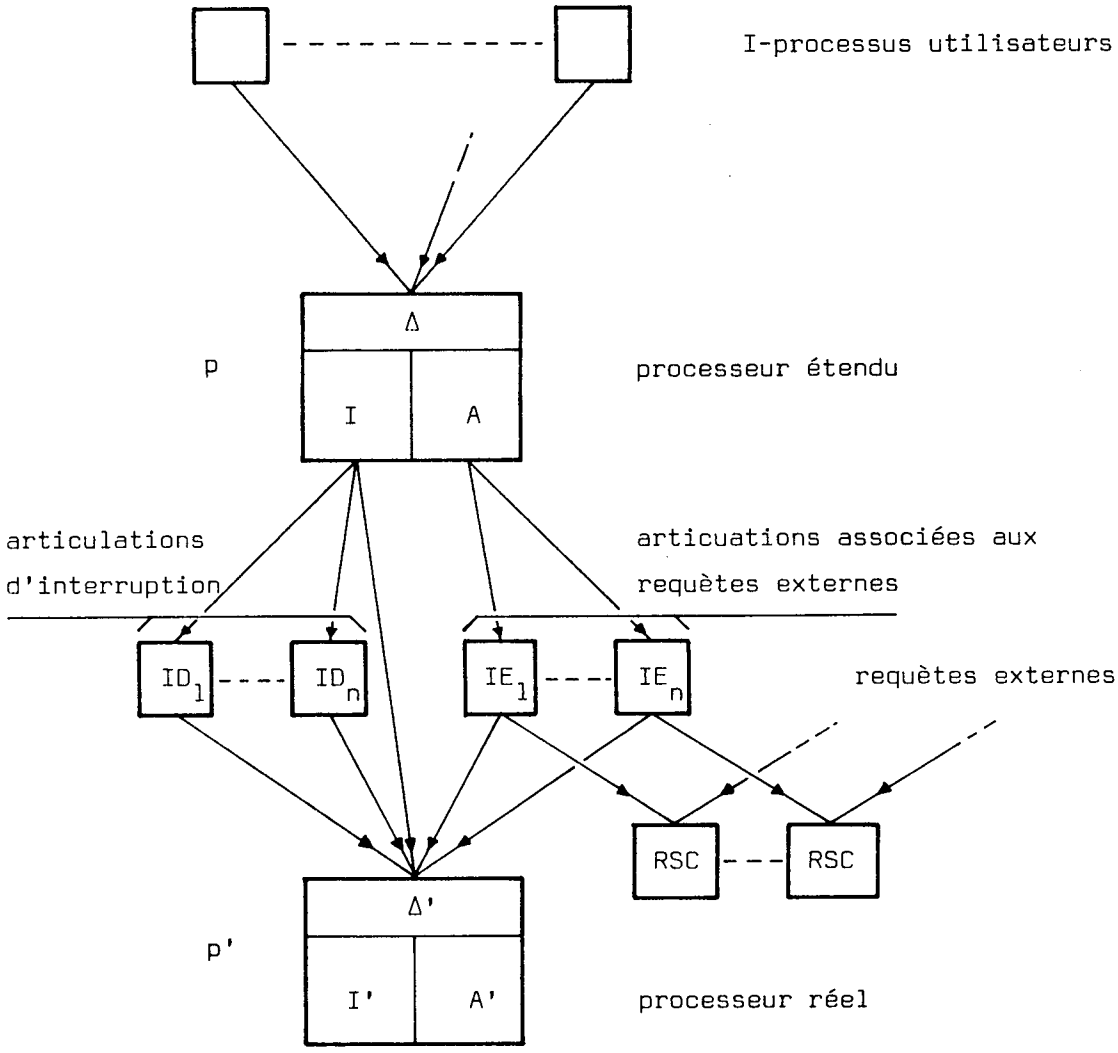
Suivant l'origine de la reqête de modification d'allocation, celle-ci sera déclenchée soit par l'interprétation par p' d'une instruction particulière par laquelle le I-processus en exécution demande une commutation d'allocation (ex.: une opération d'appel du superviseur), soit par une requête externe, dans ce cas la chaîne représentant le milieu extérieur se comporte alors en chaîne activante vis-à-vis de la chaîne activée constituée du mécanisme d'allocation A et du processeur p' . Le rôle de cette chaîne activante sera nul vis-à-vis des programmes (I-processus utilisateurs et mécanisme A) mais elle exécutera une opération $S(\Delta',A)$ pour soumettre le mécanisme d' allocation A comme candidat au processeur p'.

1/1.2.1. Mécanisme d'interruption

Le mécanisme d'interruption tel qu'il existe sur les ordinateurs courants consiste en un mécanisme technologique simple d'allocation du processeur réel p' à un ensemble fixe d'utilisateurs qui sont habituellement:

- le mécanisme d'interprétation I du processeur p offert aux I-processus utilisateurs (il s'agit du processeur étendu par le mécanisme de déroutement).
- le mécanisme d'allocation A de ce processus étendu.
- les articulations de déroutement ID_j .

Le mécanisme d'allocation du processeur étendu peut recevoir plusieurs directives du monde extérieur en représentant ces différentes requêtes par la sollicitation de l'activation de différentes articulations IE_i (niveaux d'interruptions externes).



Les articulations $ID_1 \ldots . ID_n$ et $IE_1 \ldots . IE_m$ représentent les I-processu d'interruption et sont appelées les articulations d'interruption.

Le mécanisme d'allocation A' du processeur réel p' doit répondre à certaines contraintes:

a) Les files FU_i des SDS de gestion des articulations d'interruption doivent pouvoir contenir les listes des appels relatifs

à ces articulations. Cette contrainte est requise par l'utilisation du mécanisme d'interruption pour transmettre les requêtes externes au mécanisme d'allocation A du processeur étendu par l'intermédiaire des articulations IE_j .

b) Ce mécanisme doit commuter l'allocation du processeur réel p' avant que celui-ci n'ait abordé l'exécution de la prochaine instruction. Cette contrainte est requise par l'utilisation du mécanisme d'interruption pour réaliser le mécanisme de déroutement de manière à ce que l'exécution d'un programme de déroutement prolonge effectivement l'exécution de l'instruction qui l'a sollicité.

remarque:
Un certain travail de préparation relatif à l'exécution de l'instruction suivante peut avoir été fait par p' avant que la commutation ne se produise. En aucun cas les organes de mémorisation de la machine à laquelle ce processeur appartient ne doivent avoir été modifiés de manière irréversible (compteur ordinal, registres programmables, mémoire).

c) Ce mécanisme doit fournir à l'articulation d'interruption appelée l'identité du I-processus utilisateur précédemment en exécution sur le processeur étendu. Cette contrainte est requise pour la réalisation du mécanisme de déroutement, de manière à ce que les articulations de déroutement aient les mêmes privilèges que les I-processus utilisateurs qui sont à l'origine de leur appel.

d) Il est quelquefois demandé à ce mécanisme d'être suffisamment rapide pour que les requêtes extérieures soient satisfaites relativement à certaines contraintes de temps (temps réel).

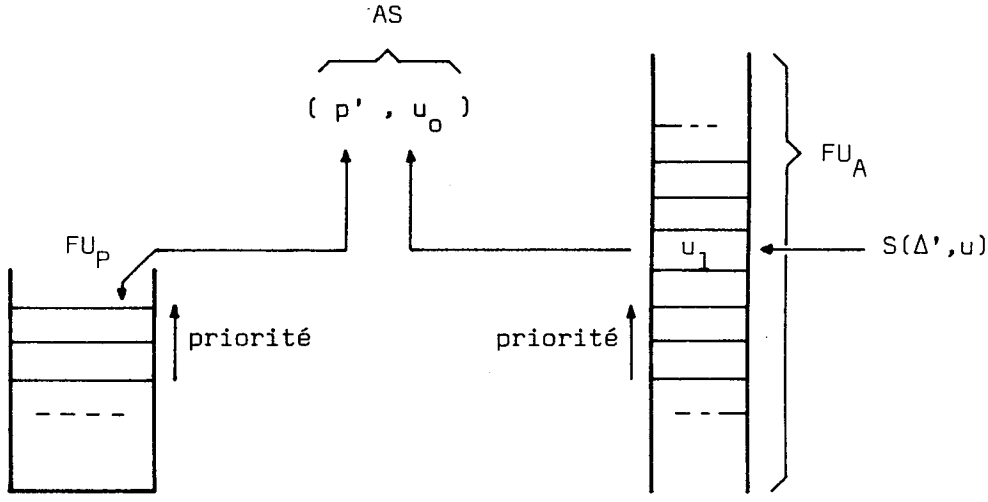
Remarques: Si nous ajoutons la contrainte que les programmes constituant les articulations d'interruption doivent s'exécuter rapidement, les contraintes b) et d) peuvent être utilisées pour éliminer la contrainte a) . En effet il est raisonnable de penser que dans ces conditions une articulation IE_i aura le temps de s'exécuter avant qu'une nouvelle requête ne lui parvienne.

Ces précautions seront souvent inutiles, car dans la majorité des cas les requêtes externes ne sont que des réponses indirectes à des actions du processeur sur le milieu extérieur.

1/1.2.2. Réalisation d'un mécanisme d'interruption

Le mécanisme d'allocation A' du processeur réel p' est souvent réalisé de manière autonome. La file FU de la SDS de gestion de ce processeur est souvent réalisée sous la forme:

- d'une file FU_A des utilisateurs de p' candidats à l'exécution, non encore démarrés et classés par priorité décroissante.
- d'une pile FU_P des utilisateurs de p' dont l'exécution a été suspendue à la suite de la requête de p' pour l'exécution d'un utilisateur de priorité supérieure.



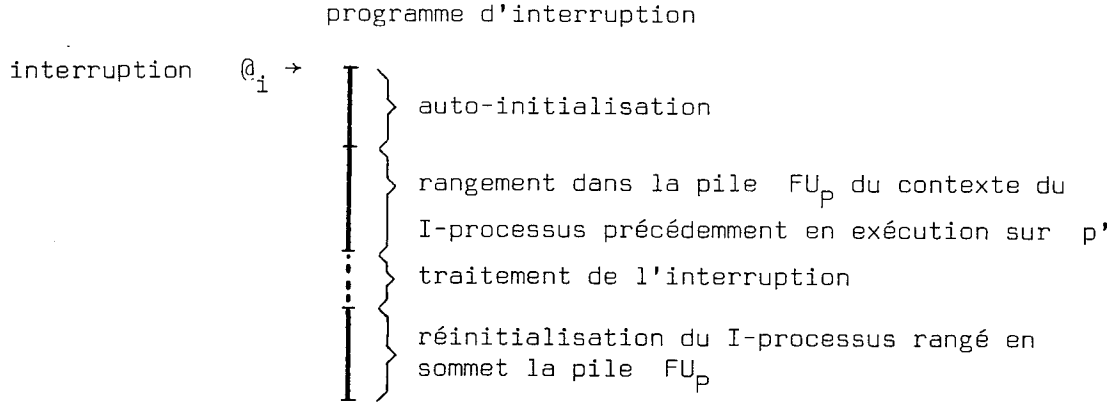
opération S (Δ', u)
METTRE (u, FU_A)

opération S(Δ', p')
u_o ← $proj_U$(AS)
DISSOCIER (u_o, p')
PRENDRE (u', FU_P) "extraire un élément de la pile"
ASSOCIER (u', p')

```
allocation autonome
E:   u_o ← proj_U (AS)
     u_1 ← max (FU_A)
           priorité (u_i)
     si priorité (u_o) < priorité (u_1) alors
          début
               DISSOCIER (u_o, p')
               METTRE (u_o, FU_P)  "ranger un élément dans la pile"
               EXTRAIRE (u_1, FU_A)
               ASSOCIER (u_1, p')
          fin
     aller a E
```

Il est aisé de montrer que les utilisateurs rangés dans la pile FU_P sont ordonnés par priorité croissante.

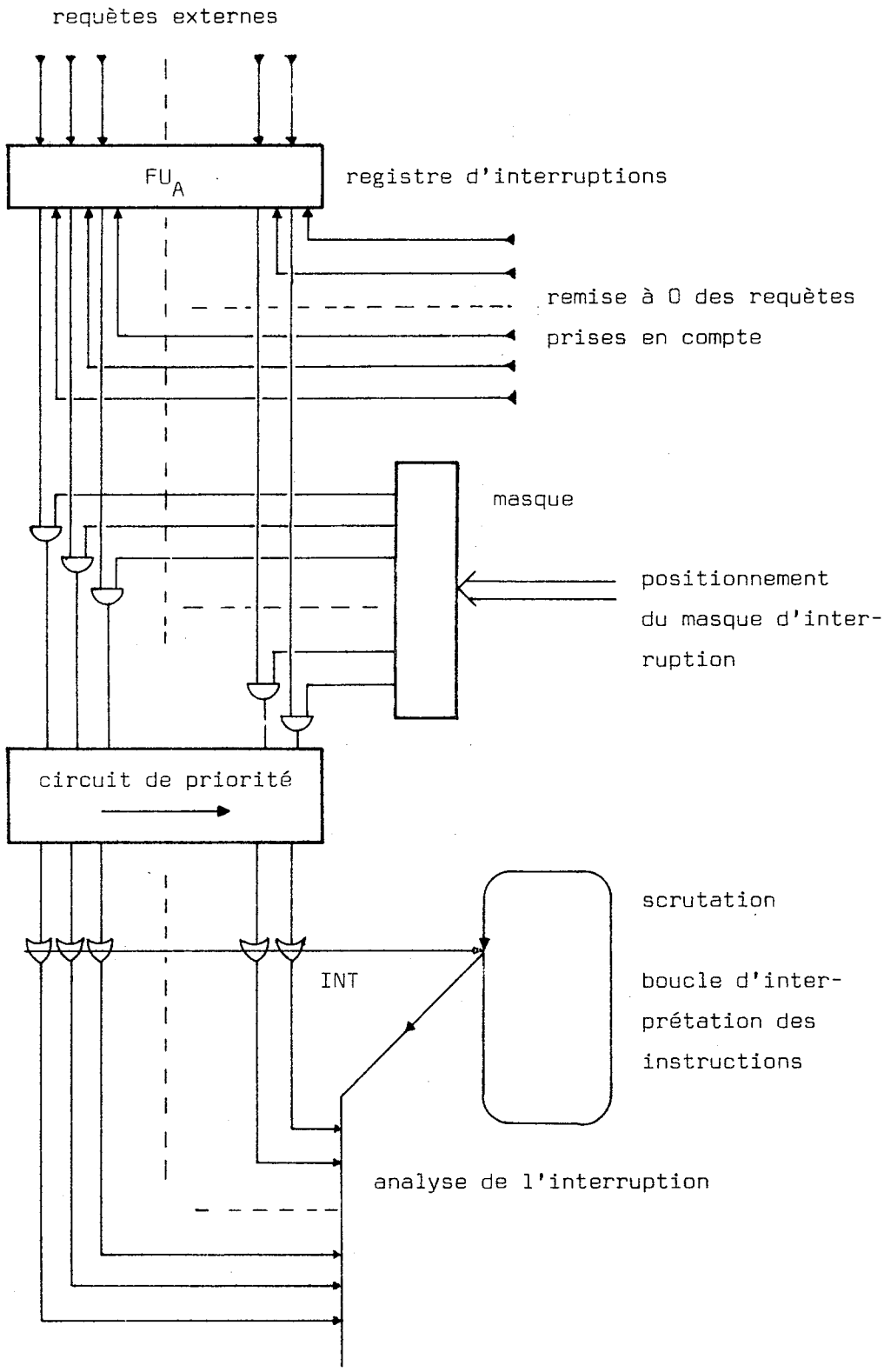
La raison pour laquelle de tels mécanismes furent définis est qu'ils ne nécessitent que des organes technologiques très simples complétés par des programmes adaptables aux conditions particulières de fonctionnement. En effet, l'arrêt d'un I-processus en cours d'exécution suppose la conservatio d'une certaine quantité d'informations constituant son contexte, tandis qu' peut s'auto-initialiser par programme. Il est relativement simple de gérer par programme l'ensemble des contextes des articulations d'interruption arrêtées au cours de leur exécution en simulant la pile FU_P . Dans ces conditions seul le démarrage des articulations d'interruption reste à la charge des organes technologiques. Il se fait généralement à une adresse implicitement contenue dans l'interruption sollicitée.



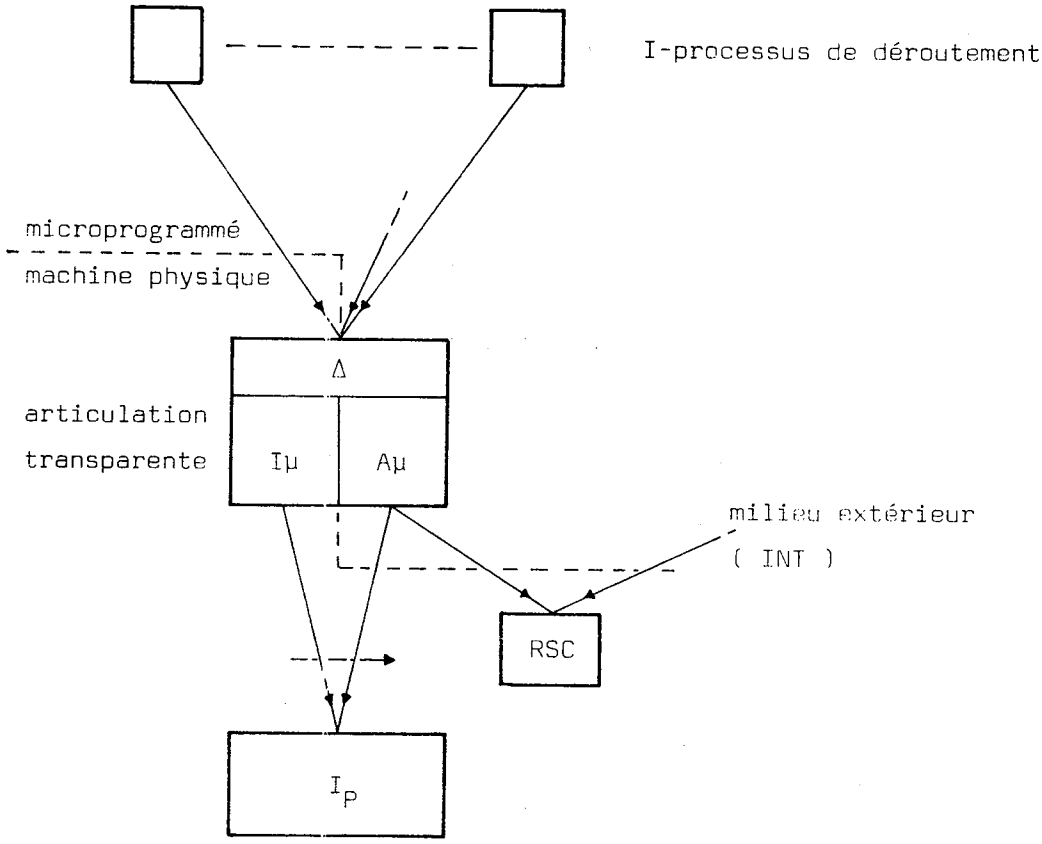
Un tel mécanisme n'est compatible qu'avec l'existence d'outils de synchronisation simplifiés entre les articulations d'interruption. En particulier la mise en attente de l'une de ces articulations accompagnée d'un redémarrage du candidat suspendu de priorité inférieure demande une modification du mécanisme précédent.

Le fait que le processeur réel est unique est largement utilisé pour obtenir un mécanisme de mutuelle exclusion simplifié en utilisant un masque technologique empêchant certains I-processus candidats de FU_A d'être pris en compte par le mécanisme d'allocation autonome.

La file FU_A est réalisée de manière technologique à l'aide d'un registre ayant autant de positions que p' admet d'articulations d'interruption différentes. Un registre de masque permet d'ignorer certains candidats. La sortie du registre d'interruption conditionnée par le contenu du registre de masque est connectée à un mécanisme de mutuelle exclusion ne transmettant que la requête la plus prioritaire. L'algorithme d'interprétation du processeur commande ce circuit de priorité et teste périodiquement sa sortie (entre chaque instruction) pour décider s'il doit continuer son travail courant ou passer à l'interprétation d'un programme d'interruption. Les commutations résultant d'évènements internes (déroutements, requêtes au superviseur) peuvent soit transiter par l'extérieur et utiliser le même mécanisme, soit déclencher directement la commutation.


requètes externes
FU A
registre d'interruptions
remise à 0 des requètes prises en compte
masque
positionnement du masque d'interruption
circuit de priorité
scrutation
INT
boucle d'interprétation des instructions
analyse de l'interruption

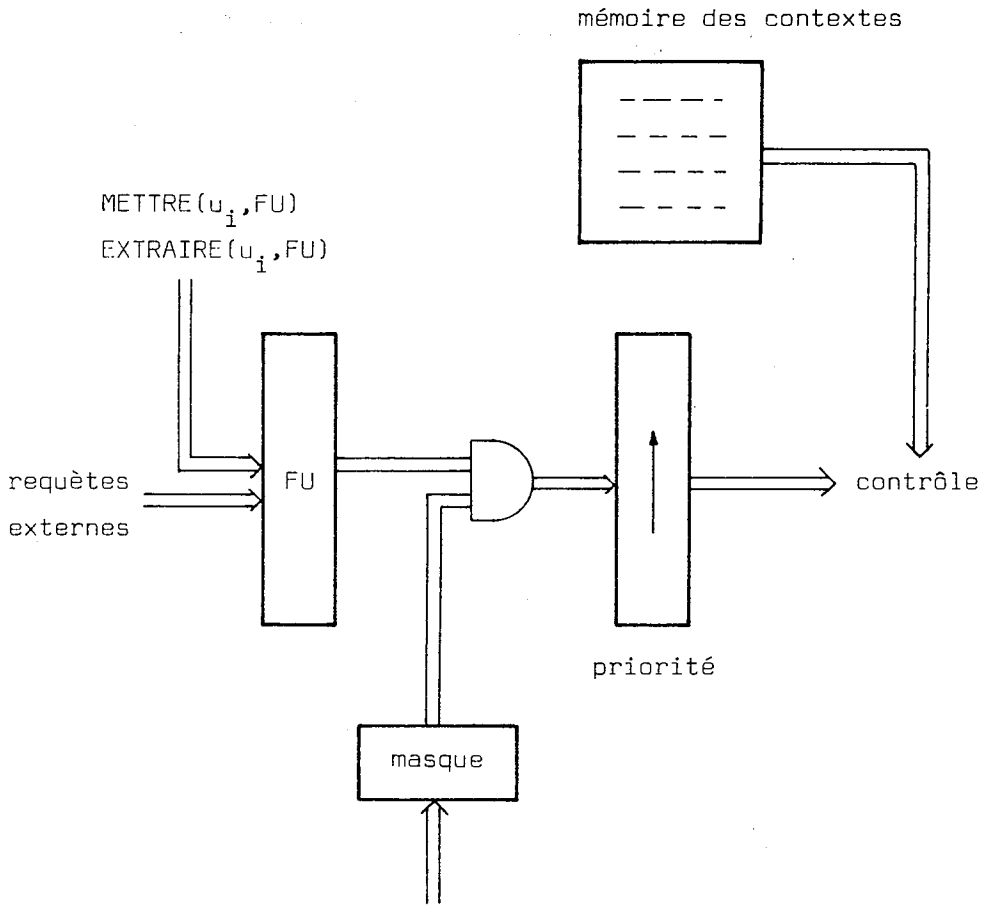
La représentation de ce mécanisme en tant que système hiérarchisé est:



L'interpréteur physique des micro-instructions peut se commuter vers le mécanisme d'allocation de la micro-machine , soit à chaque interprétation d'une instruction de test de la présence d'un appel externe (ligne int.), soit à l'exécution d'une instruction d'appel du superviseur, soit à la rencontre d'une condition de déroutement.

Remarque:

Si l'on ajoute au niveau de la machine technologique un organe de mémorisation capable de contenir ou de pointer sur le contexte initial ou courant d'une articulation d'interruption on obtient un mécanisme conceptuellement plus simple qui permet des mécanismes de synchronisation plus évolués et qui évite l'emploi d'une pile annexe [58].



opération S(Δ', u)
METTRE (u, FU)

opération S(Δ', p')
$u_o \leftarrow proj_U$ (AS)
$u_1 \leftarrow max_{priorité\ (u_i)}$ (FU)

allocation autonome
E : $u_o \leftarrow proj_U$ (AS)
$u_1 \leftarrow max_{priorité\ (u_i)}$ (FU)
si priorité (u_o) < priorité (u_1) alors
début
DISSOCIER (u_o, p')
METTRE (u_o, FU)
EXTRAIRE (u_1, FU)
ASSOCIER (u_1, p') "initialiser ou réinitialiser le contexte de u_1"
fin
aller à E

1/1.3. Gestion de l'allocation des utilisateurs par le processeur lui-même

Le mécanisme d'allocation du processeur aux I-processus utilisateurs peut être réalisé au niveau du processeur lui-même, généralement par micro-programmation [5, 31, 37] . Il peut être vu alors comme un développement du mécanisme d'interruption bien qu'il gère alors l'allocation des I-processus utilisateurs. A cause de ce développement, il n'est plus possible d'utiliser cet allocateur pour réaliser le mécanisme de déroutement qui devra donc être considéré à part.

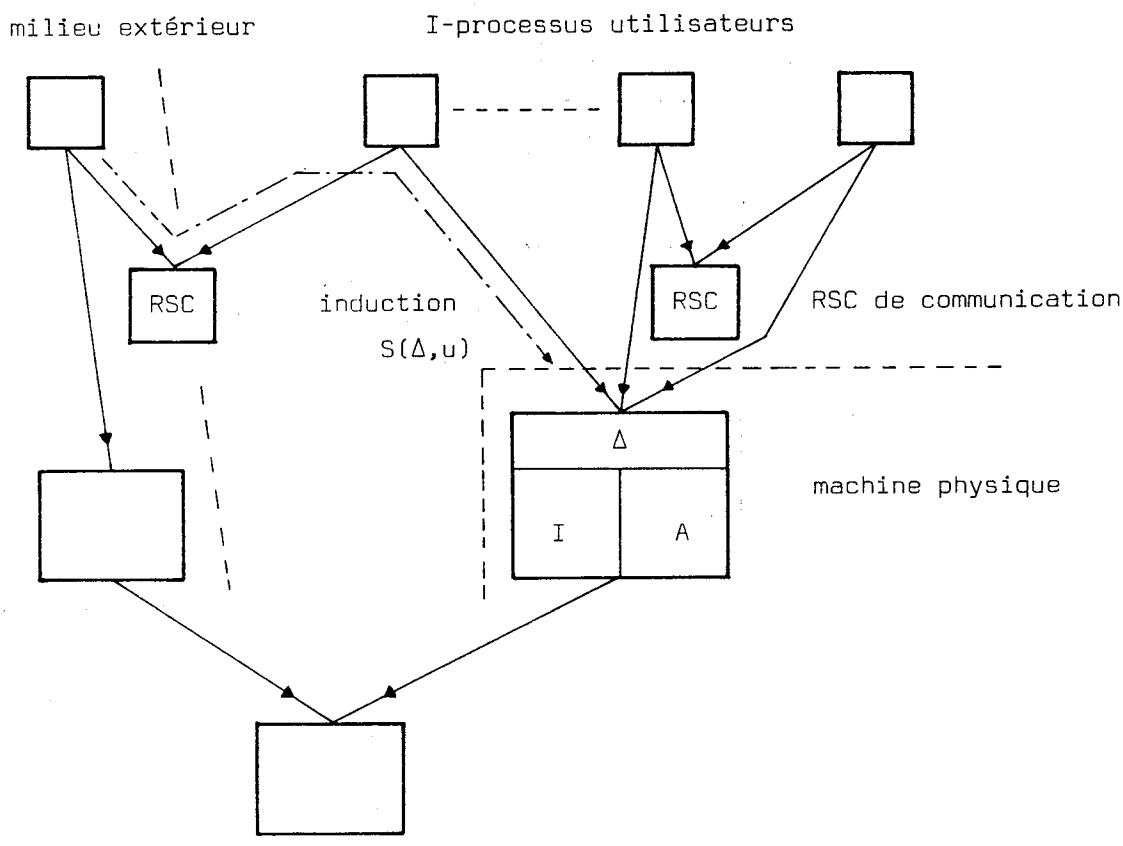
Un tel mécanisme réalise les algorithmes définis par les primitives S de gestion de la ressource requérable constituée par le processeur réel lui-même. Il est réalisé par l'organe de contrôle de ce processeur.

Les requêtes externes testées par l'algorithme d'interprétation des instructions sont assimilées à des demandes d'exécution des mêmes opérations S. La SDS de gestion du processeur est réduite à la file FU décrite dans la mémoire de la machine.

L'une des principales difficultés rencontrées lors de la réalisation de tels mécanismes consiste à gérer la file FU par priorité. Plusieurs solutions sont alors possibles:

- L'ensemble des I-processus auxquels peut être alloué le processeur est représenté par un tableau fixe, ordonné par priorité, de pointeurs vers leu contexte . Ceux qui sont candidats sont marqués comme tels dans le tableau. L'ensemble des entrées marquées constitue la file FU . Une telle organisation simplifie l'exécution de l'opération METTRE(u_i, FU) puisqu'elle se ramène à un simple adressage. En contrepartie l'opération PRENDRE (u_j, FU) nécessite un balayage pour trouver le candidat le plus prioritaire.

- La file FU est construite sous la forme du chaînage par ordre de priorité décroissante des contextes des I-processus candidats. Dans ce cas l'opération METTRE(u_i, FU) est beaucoup plus complexe puisqu'il faut balayer la liste pour trouver l'endroit où l'on doit insérer le nouvel arrivant. Par contre l'opération PRENDRE(u_j, FU) est triviale puisqu' elle se ramène à prendre le premier I-processus de cette liste.



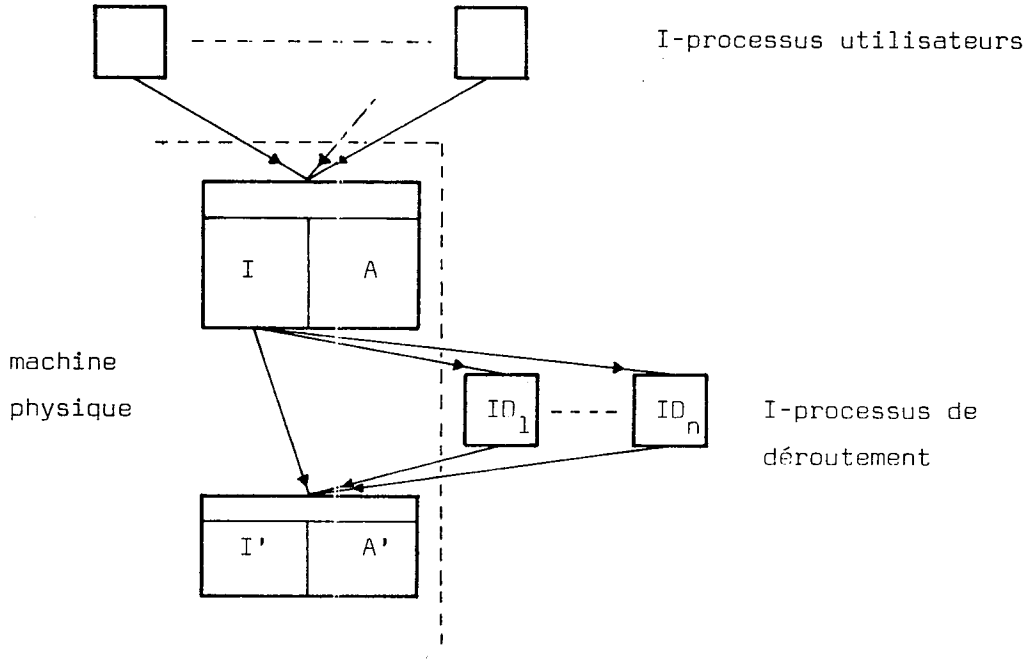
Il est évidemment possible de ne pas gérer la file FU par priorité mais par exemple comme une pile ou une queue.

1/1.3.1. Traitement des déroutements

Le traitement des déroutements peut être adjoint de différentes manières aux mécanismes d'allocation précédents:

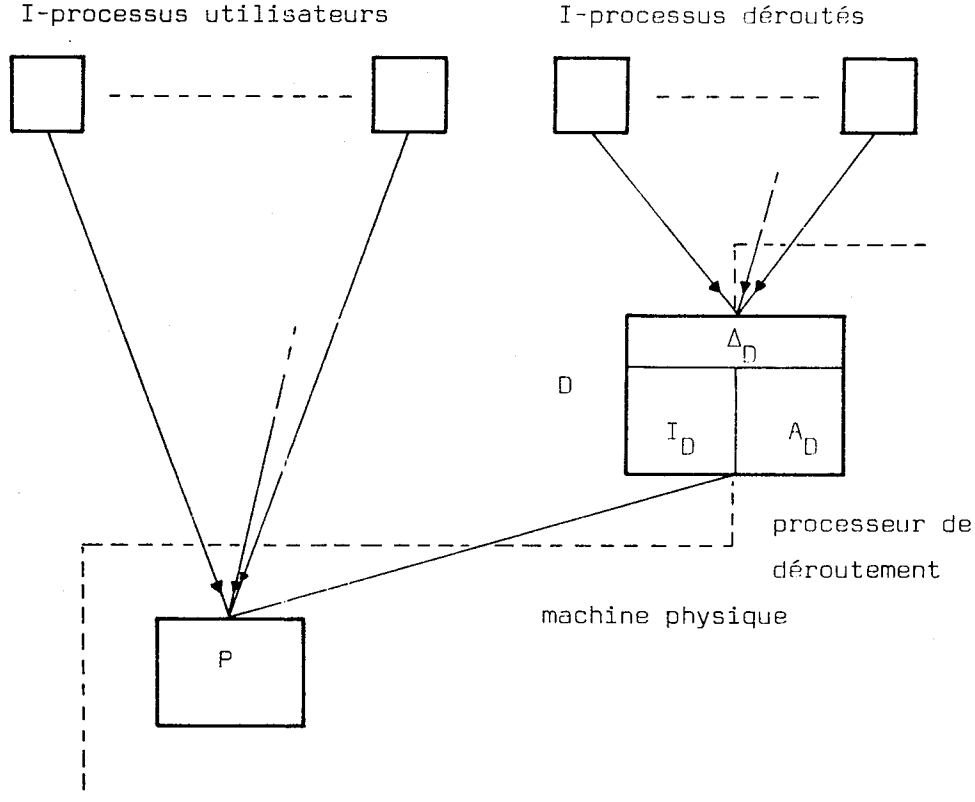
1/1.3.1.1. compléments programmés au processeur:

Le processeur connait l'emplacement dans la mémoire de programmes particuliers réservés à son seul usage, auquel il passe le contrôle par déroutement.



1/1.3.1.2. processeur programmé de déroutement:

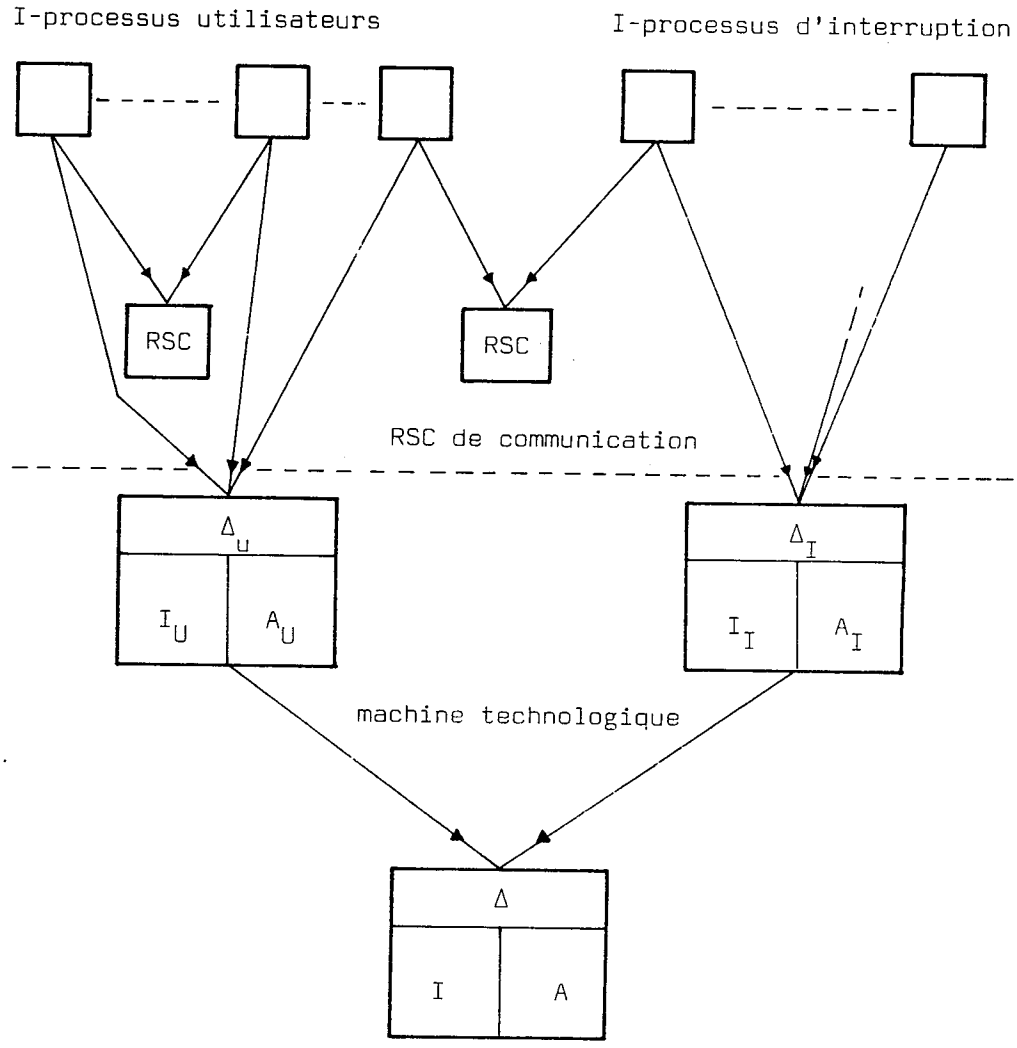
Un I-processus particulier D est géré par l'allocateur du processeur au même titre que les I-processus utilisateurs. Ce I-processus particulier peut se comporter comme un processeur particulier vis-à-vis des I-processus utilisateurs qui sollicitent son emploi lorsqu'ils déclenchent un déroutement. [§ chap.3 5].



Le mécanisme d'allocation de ce processeur complémentaire de déroutement est également réalisé par les circuits de contrôle de la machine.
Ce mécanisme a l'avantage de réduire la priorité des I-processus déroutés, donc fautifs. En contrepartie, cette particularité nuit à l'utilisation de ce mécanisme pour compléter l'ensemble des instructions reconnues par le processeur.

1/1.3.2. Découplage des requêtes externes

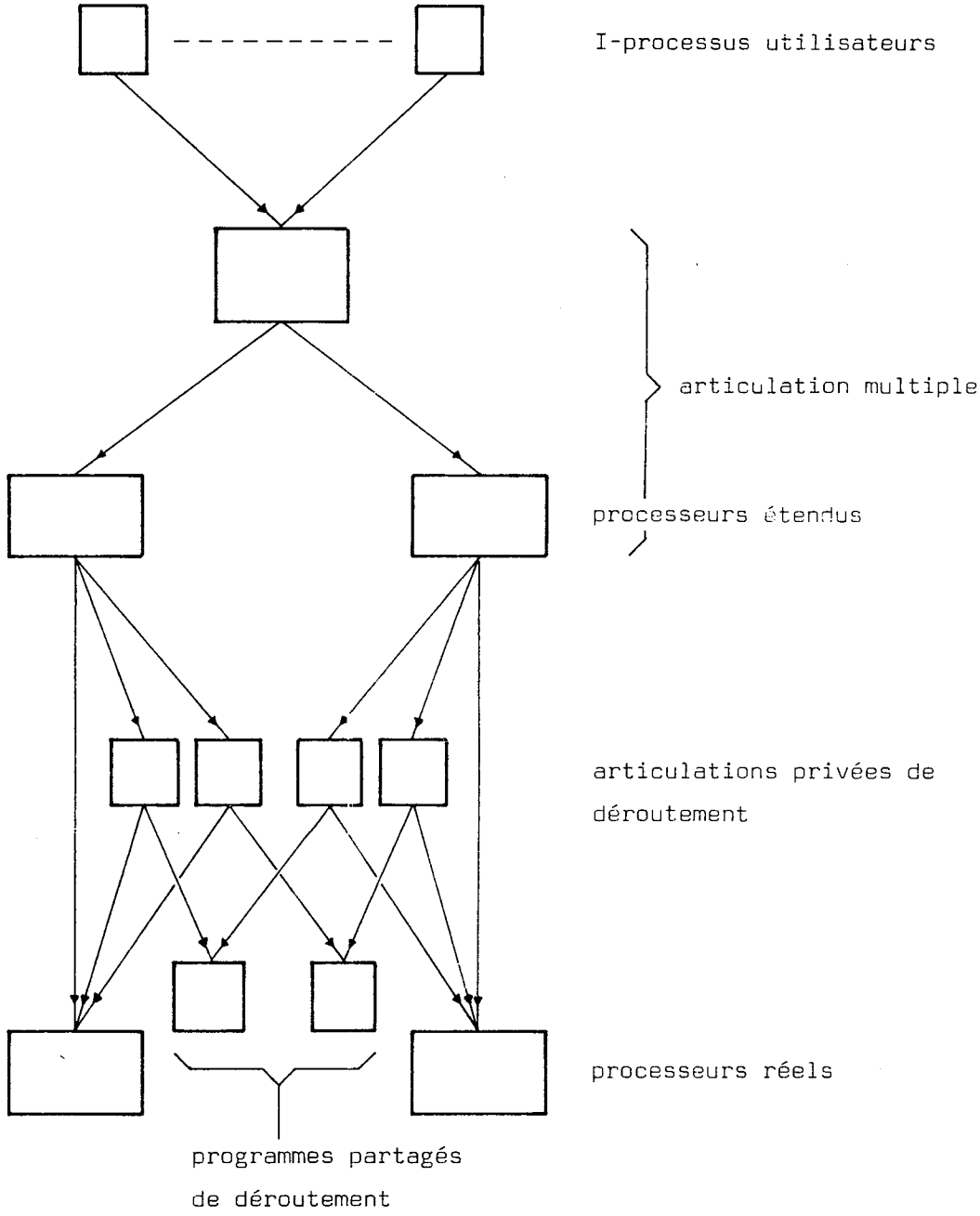
Il peut être intéressant de dissocier l'occurrence des reqêtes externes et l'exécution des opérations de synchronisation. Une solution à ce problème consiste à répartir l'activité du processeur réel entre deux processeurs virtuels, l'un pour les I-processus utilisateurs, l'autre pour un mécanisme d'interruptions externes. Les I-processus d'interruption doivent avoir la possibilité d'exécuter les opérations de synchronisation sur les RSC programmées qui permettent aux I-processus utilisateurs de se synchroniser. Pour les raisons déjà évoquées les I-processus d'interruption ne doivent pas pouvoir être arrêtés par l'exécution de ces opérations de synchronisation [31, 37] . Un tel mécanisme permet le traitement au niveau des I-processus d'interruption de certaines requêtes externes. Toutefois le temps nécessaire pour lancer l'exécution d'I-processus utilisateur se trouve accru par la présence de ce relai.


I-processus utilisateurs
I-processus d'interruption
RSC
RSC
RSC de communication
Δ_U
I_U
A_U
Δ_I
I_I
A_I
machine technologique
Δ
I
A

1/2. EXTENSION AU CAS DES MULTIPROCESSEURS

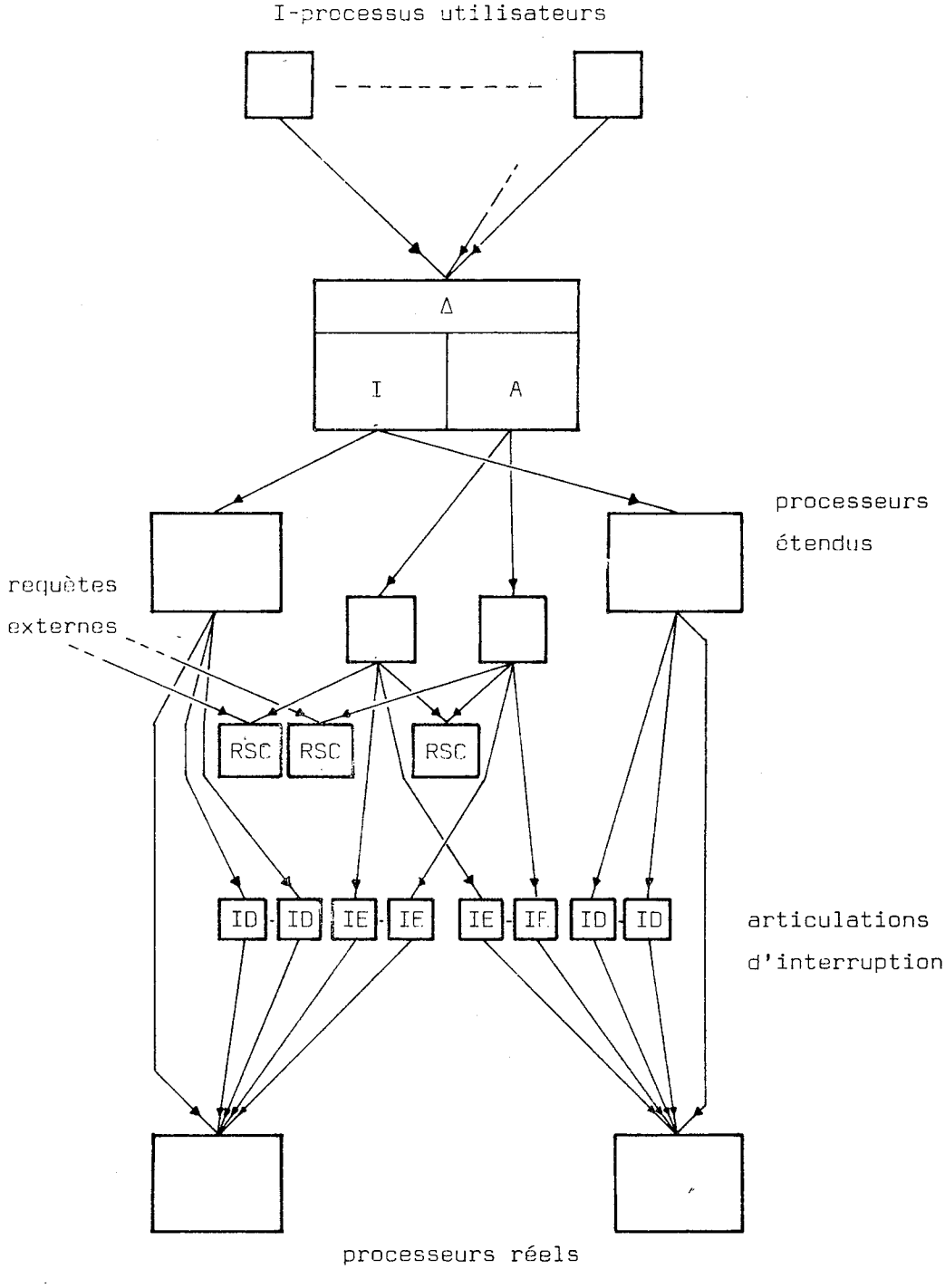
La transposition des mécanismes précédents au cas des multiprocesseurs se fait en remplaçant l'articulation simple qui représente le processeur par une articulation multiple dupliquée qui regroupe les différents processeurs équivalents.

Des précautions particulières doivent être prises concernant les mutuelles exclusions d'exécution entre les différentes articulations introduites. Alors que dans un mono-processeur un tel mécanisme peut aisément être obtenu en bloquant les modifications d'allocation du processeur, cette action n'entraîne évidemment pas la mutuelle exclusion d'exécution dans un multiprocesseur et d'autres mécanismes doivent être introduits. Nous devons également noter que la propriété des articulations de déroutement de n'être sollicitées que par, au plus, un seul I-processus utilisateur n'est plus vraie. Il faut toutefois remarquer que ces appels proviennent des différents processeurs étendus. Une solution consiste à dupliquer ces ressources additionnelles en leur faisant partager des programmes ré-entrants uniques.


I-processus utilisateurs
articulation multiple
processeurs étendus
articulations privées de déroutement
processeurs réels
programmes partagés de déroutement

1/2.1. Allocation programmée des processeurs

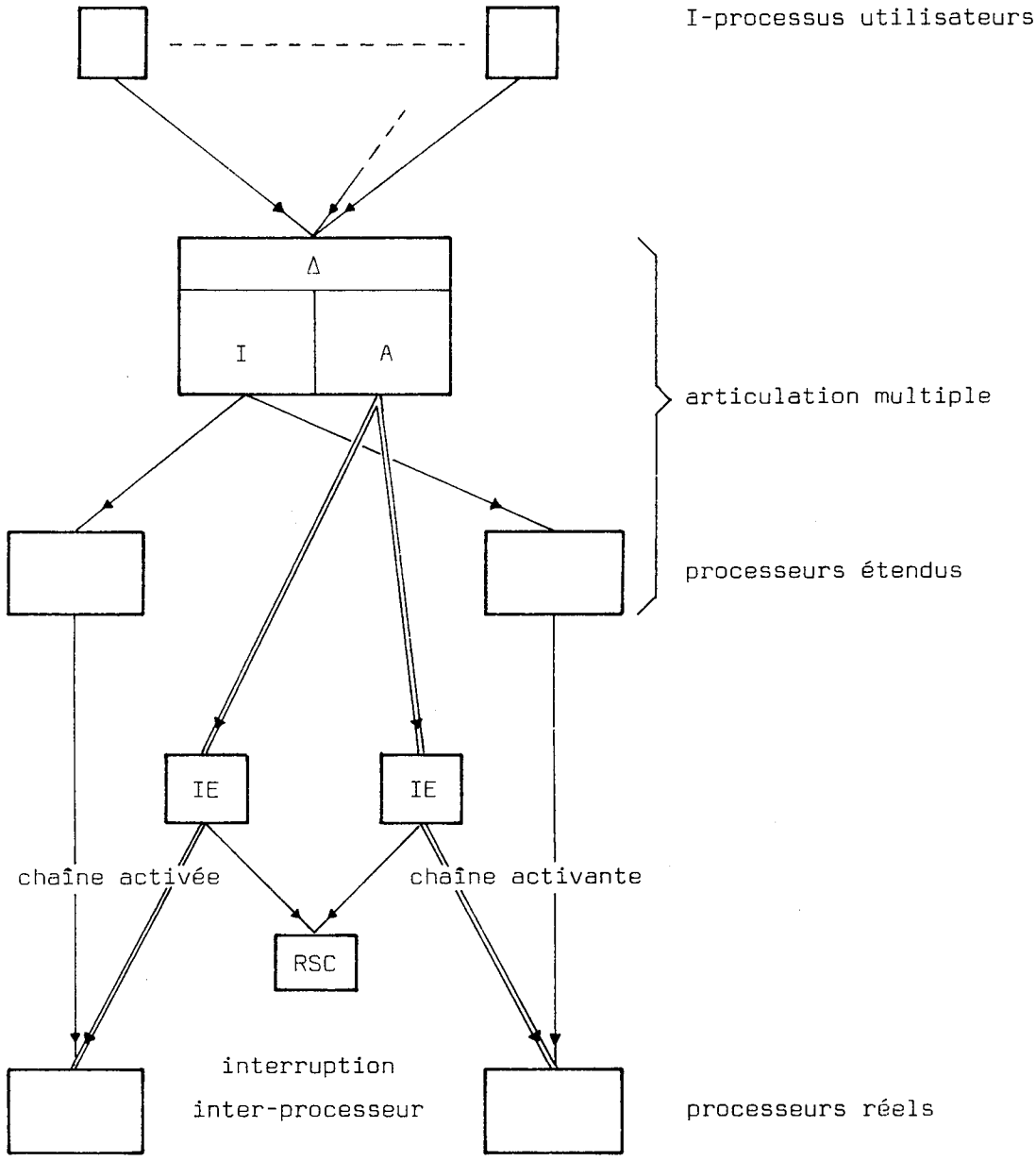
Lorsque le mécanisme d'allocation des ressources de l'articulation multiple est réalisé par un programme, son activation résulte de l'activité des mécanismes d'interruption propres à chacun des processeurs.



Comme rien n'empêche les différents processeurs d'être interrompus simultanément, un mécanisme de mutuelle exclusion doit être établi entre les articulations de déroutement. Les masques d'interruption de chaque processeur n peuvent évidemment plus remplir ce rôle puisqu'ils sont impuissants à arrête l'exécution d'une articulation d'interruption. Compte tenu des particularit des mécanismes courants d'interruption, la synchronisation établie entre ces articulations ne peut être que très simple et ne comporter que des attentes actives (par exemple par des instructions de test et de modification simultanée d'information en mémoire).

Le mécanisme d'allocation de l'articulation multiple peut avoir à modifier l'association de l'un des processeurs vis-à-vis des I-processus utilisateurs. Lorsque ce processeur est celui qui exécute le mécanisme d'allocation, l'opération ne présente aucune difficulté , par contre, lorsqu'il s'agit de modifier l'allocation d'un autre processeur il doit lui transmettre une information convenable en lui générant une interruption inter-processeur qui se reçue de la même manière qu'une requête externe particulière. Ce mécanisme est une induction directe dans laquelle il est possible de distinguer une chaîne activante et une chaîne activée.

Les interruptions externes qui constituent des requêtes d'allocation pour le système sont soit orientées de manière fixe vers certains processeurs soit aiguillées dynamiquement vers l'un de ceux-ci en fonction de leur instant d'occurrence ou de la priorité des I-processus associés. Si pour cette dernière solution les interruptions sont reçues par le processeur associé au I-processus en exécution de plus faible priorité, le mécanisme d'interruption inter-processeur n'a plus à être utilisé pour ce type de requêtes puisque le mécanisme d'allocation, appelé par la requête externe, fonctionne toujours sur le processeur dont on risque de changer l'allocation [39]. Cette amélioration est toutefois compensée par la nécessité de disposer d'un mécanisme externe de décision.



Le caractère privé des articulations d'interruption impose la duplication de la zone mémoire qui contient les informations qui leur sont relatives (files FU_i de leur SDS de gestion, cotextes) (problèmes relatifs à la duplication de la mémoire basse lorsque certains ordinateurs sont reliés en une configuration multiprocesseur).

Remarque:

Les interruptions inter-processeur ne servent qu'à prévenir un processeur qu'il doit s'occuper de son allocation. Un simple indicateur suffit pour cette fonction et il est inutile de prévoir une file pour y ranger les différentes sollicitations non encore traitées.

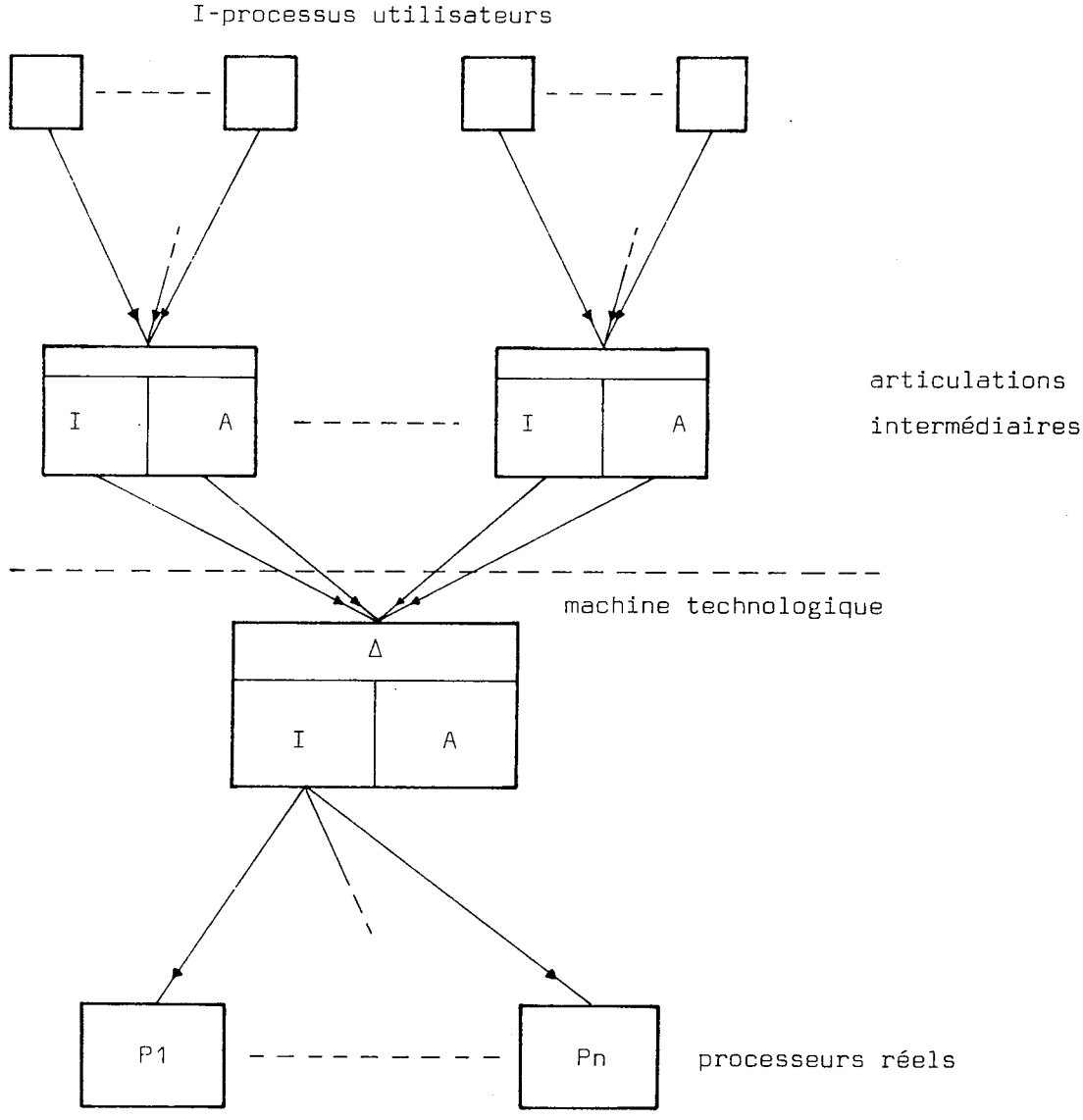
1/2.2. Allocation des processeurs réalisée par un organe indépendant

Le mécanisme d'allocation de l'articulation multiple regroupant un ensemble de processeurs équivalents peut être géré par un organe technologique distinct de ces processeurs [20].

De nombreuses solutions sont possibles, l'une d'entre elles consiste à réaliser à ce niveau un mécanisme voisin de celui habituellement utilisé pour gérer les articulations d'interruption. Il est en effet aisé de montrer qu'un tel mécanisme de gestion peut s'appliquer à l'allocation des ressources équivalentes d'une articulation multiple. Dans ce cas la comparaison doit alors s'effectuer entre le moins prioritaire des I-processus en exécution et le plus prioritaire des candidats.

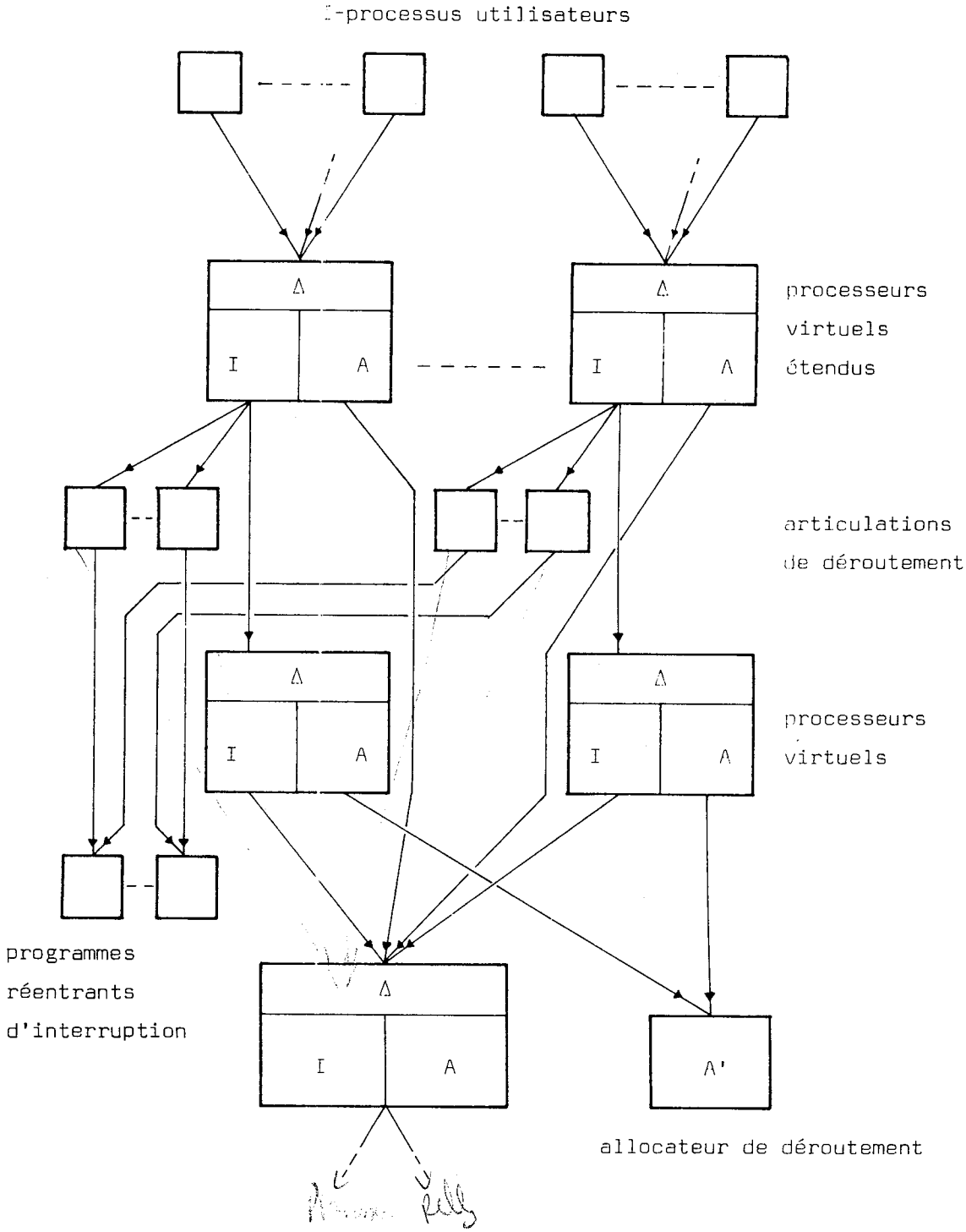
Mécanisme d'allocation

```
E :  (u_o, p_o) ← min_priorité (u_i) (AS)
     u_1        ← max_priorité (u_i) (FU_A)
     si priorité (u_o) < priorité (u_1) alors
          début
               DISSOCIER (u_o, p_o)
               METTRE (u_o, FU_p)
               EXTRAIRE (u_1, FU_A)
               ASSOCIER (u_1, p_o)
          fin
     aller a E
```



Ce mécanisme d'allocation est trop élémentaire pour pouvoir être utilisé pour les I-processus utilisateurs. Il ne sera, en général, utilisé que pour allouer les processeurs à un premier niveau d'articulations intermédiaires qui constituent des processeurs virtuels, et leurs mécanismes d'allocation, vis-à-vis des I-processus utilisateurs.

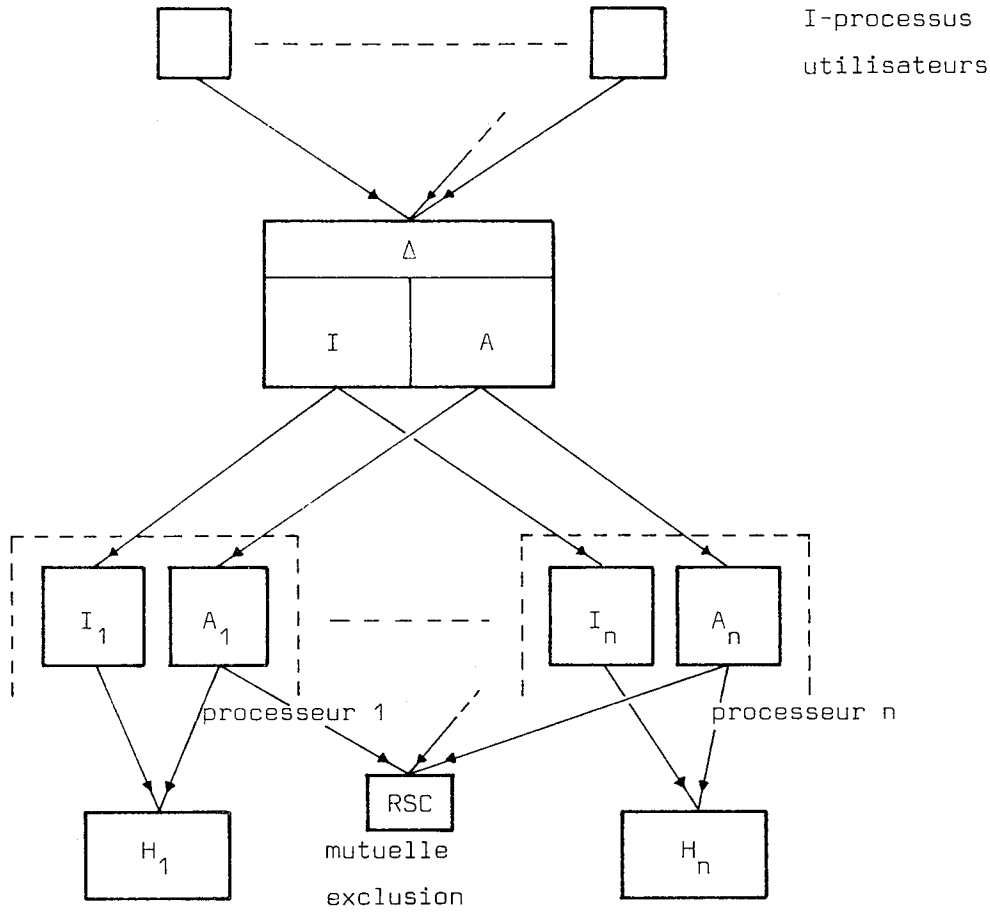
La réalisation d'un mécanisme de déroutement peut se faire en insérant un niveau supplémentaire d'articulations représentant des processeurs virtuels étendus. Le mécanisme d'allocation des processeurs virtuels aux articulations de déroutement doit être réalisé par un organe particulier de la machine puisque sa réalisation nécessite qu'il soit sollicité par le mécanisme d'interprétation des processeurs réels.



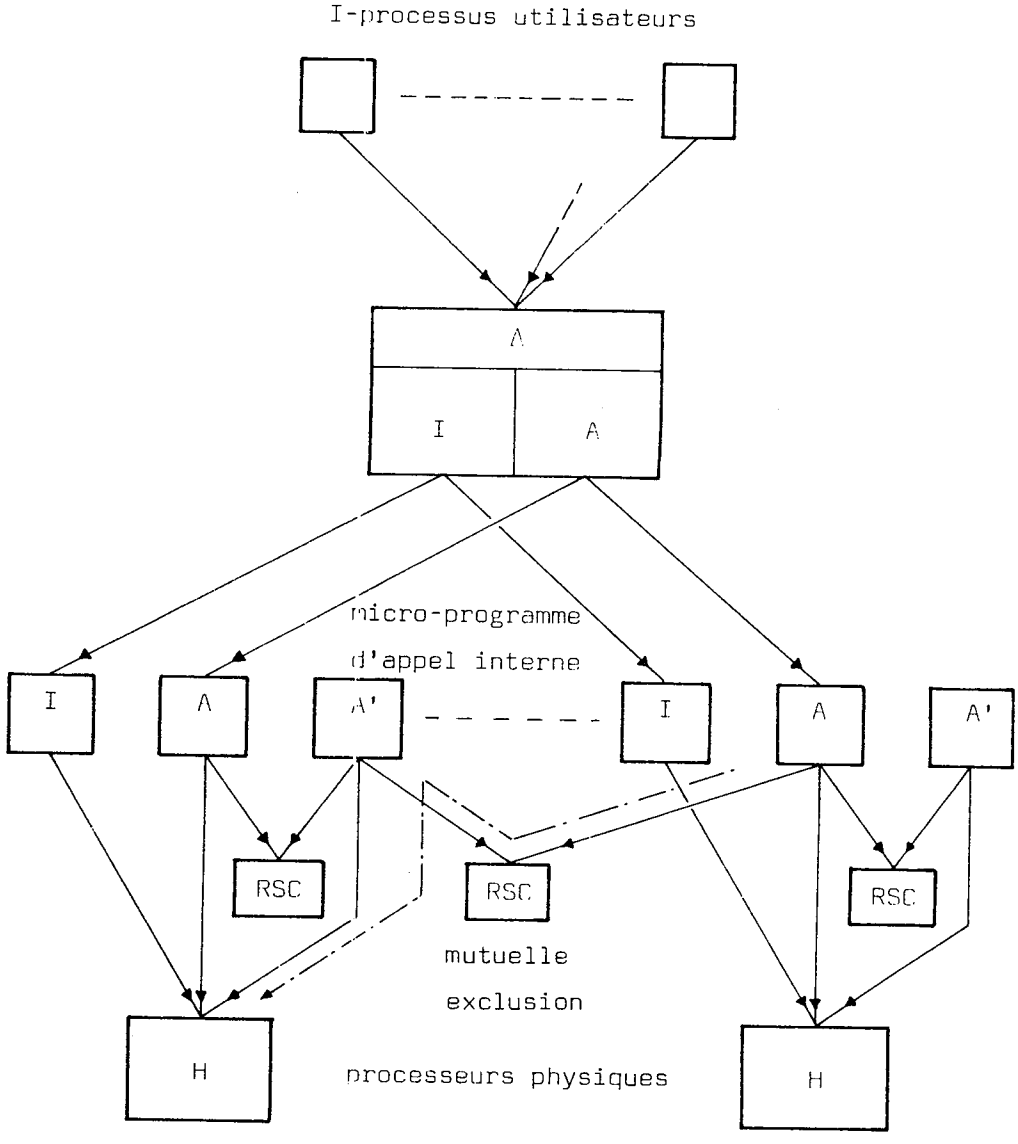
Dans un tel système, l'ensemble des êtres programmés (I-processus utilisateurs, articulations intermédiaires, allocateurs de processeurs virtuels, programmes de déroutement) n'a pas à connaître le nombre de processeurs réels en service. Ceci doit donc tendre à procurer une indépendance la plus complète possible entre le monde des programmes et les ressources réelles offertes par la machine.

1/2.3. Cas de processeurs munis de leur propre mécanisme d'allocation

Dans ce cas, chaque processeur possède son propre mécanisme d'allocation. Un mécanisme convenable de mutuelle exclusion fait qu'ils peuvent, tour à tour, jouer le rôle du mécanisme d'allocation de l'articulation multiple (au moins en ce qui concerne leurs processeurs respectifs). Les différentes SDS de gestion des processeurs doivent être confondues dans la SDS de gestion de l'articulation multiple. Cela est possible si la file FU_i de chacune de ces SDS est séparée de son ensemble AS_i (réduit d'ailleurs à au plus un couple). Toutes les files FU_i doivent être confondues en une seule file FU commune tandis que le produit des ensembles AS_i $AS_1 \times \ldots \times AS_n$ donne l'ensemble AS.



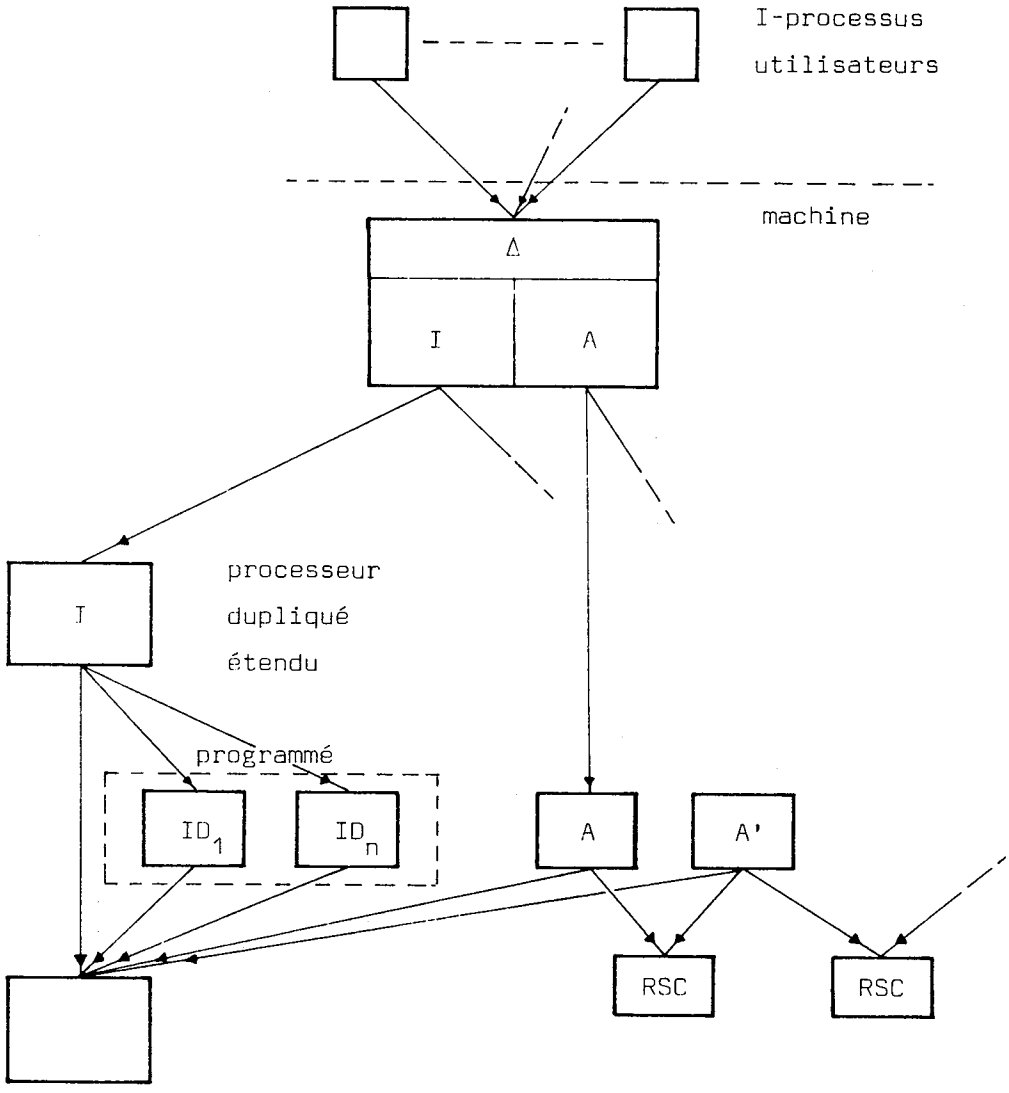
Le problème de l'allocation d'un processeur par le mécanisme d'allocation propre à un autre processeur se pose à nouveau. Il peut être résolu de différentes manières, en particulier par l'utilisation d'un mécanisme spécial d'interruption au niveau des processeurs physiques eux-mêmes (par exemple par la commutation de l'allocation de la micro-machine à un micro-programme spécifique). Comme dans le cas de l'interruption inter-processeur, il est inutile de prévoir une file pour y ranger les occurrences trop rapprochées.



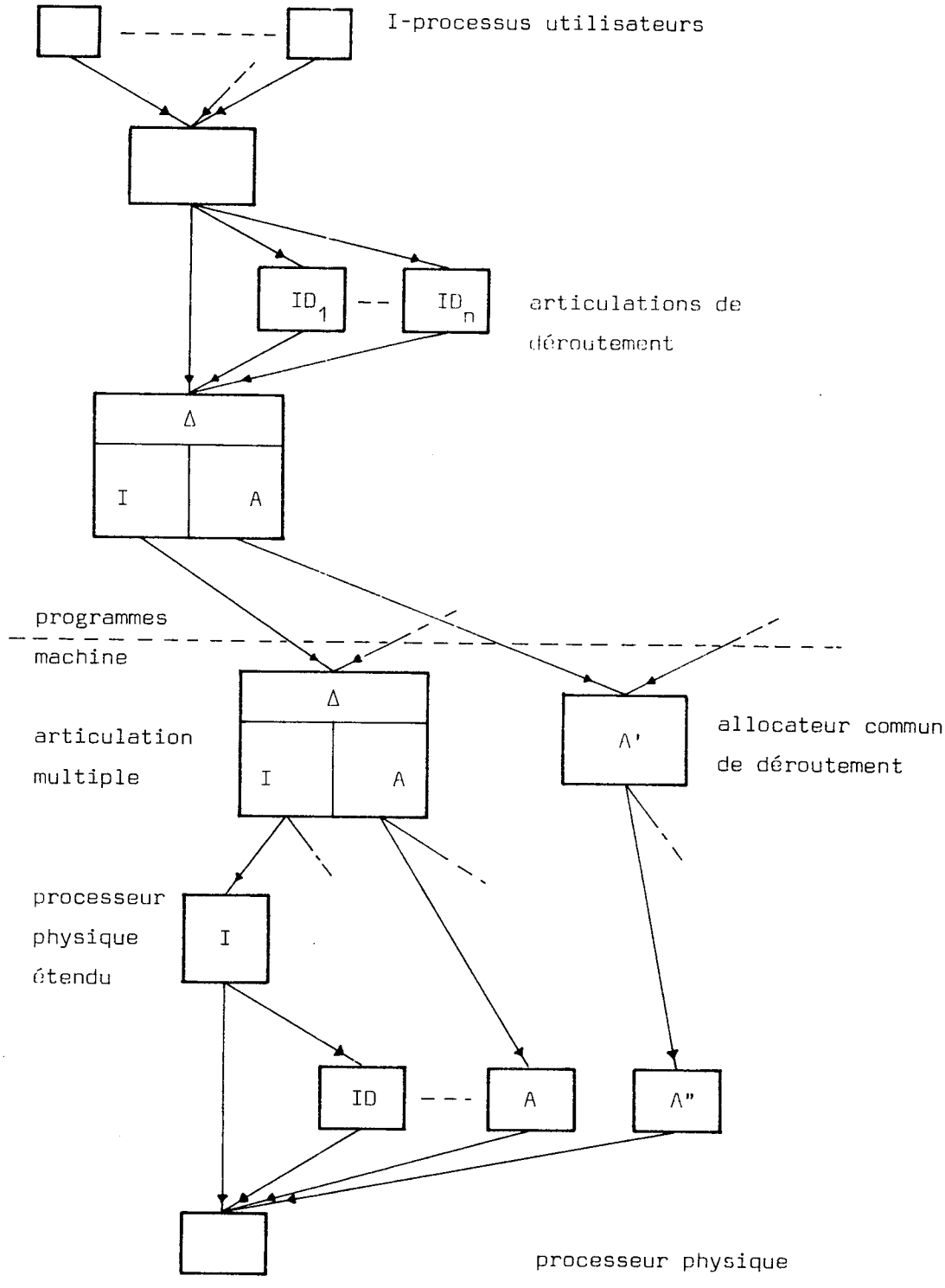
Les requêtes externes sont soit orientées d'une manière fixe vers les différents mécanismes d'allocation des processeurs, soit aiguillées vers celui du processeur qui exécute le I-processus de priorité minimale.
Les mécanismes de déroutement propres à ce genre de machines peuvent aisément être étendus au cas des multi-processeurs:

- le mécanisme utilisant un processeur programmé de déroutement s'étend immédiatement.

- un mécanisme de déroutement propre à chaque processeur peut également être étendu puisque chaque mécanisme d'allocation a la possibilité de bloquer toute commutation de l'allocation de son processeur durant l'exécution d'un programme de déroutement.



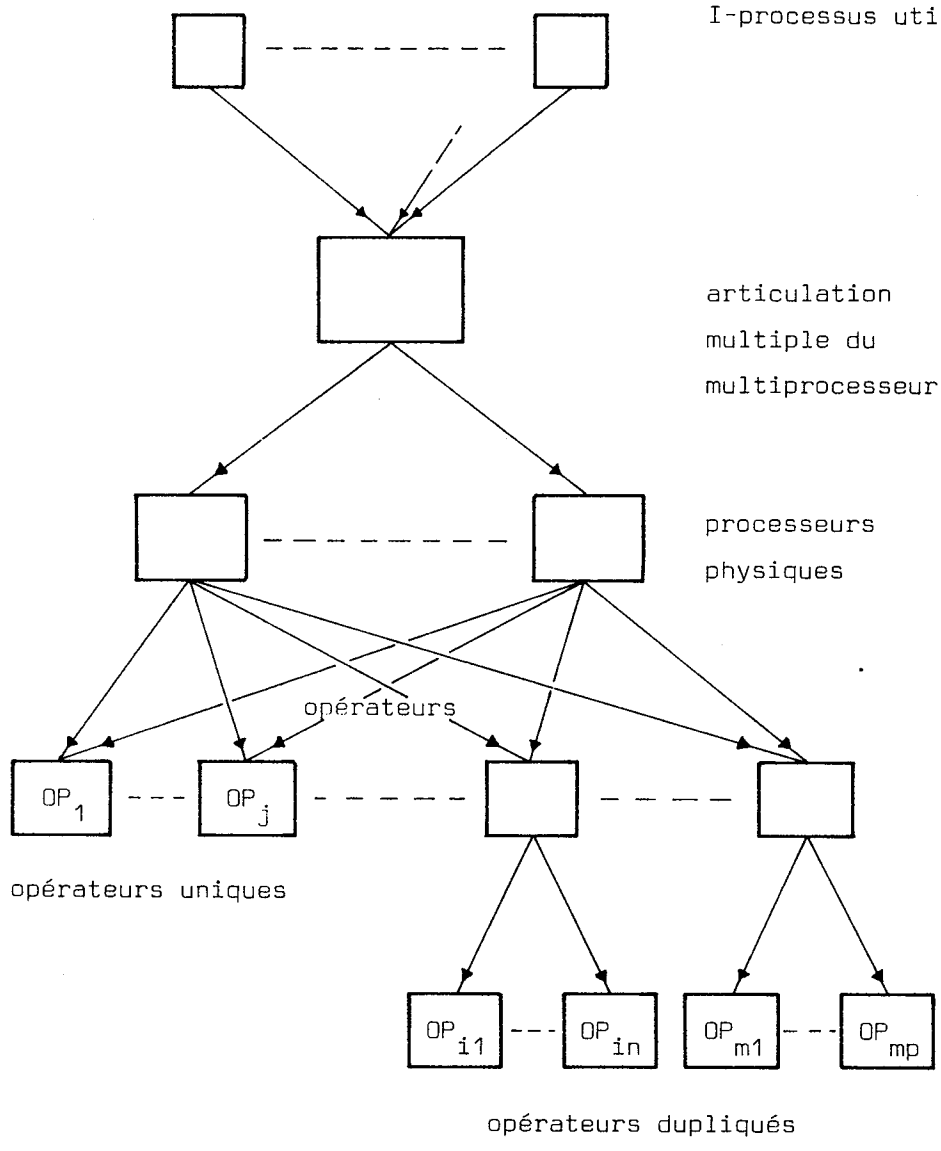
- Un niveau supplémentaire de processeurs virtuels peut être réalisé comme dans le cas d'un mécanisme d'allocation indépendant et unique. Dans ce cas l'allocateur de ces processeurs virtuels aux articulations de déroutement est réalisé au niveau de chaque processeur réel.



1/3. MULTIPROCESSEURS VIRTUELS

Plusieurs auteurs [8, 57] ont proposé une architecture originale de multiprocesseur basée sur la mise en commun de certains organes relativement peu fréquemment utilisés (multiplieurs rapides, diviseurs, opérateurs en virgule flottante...), chacun de ces organes existant en un nombre d'exemplaires proportionnel à son taux d'emploi.

Les processeurs physiques de cette machine disposent donc d'opérateurs partagés et n'ont plus qu'à assurer l'appel, l'enchaînement des instructions et les opérations les plus fréquemment employées.



Une telle architecture semble de nature à optimiser le compromis entre les performances et la complexité de la machine. Toutefois, sa principale difficulté réside d'une part dans la complexité des mécanismes d'allocation des différents opérateurs et d'autre part dans la performance et la complexité des organes de communication nécessaires.

2 - ADRESSAGE DES RESSOURCES MEMOIRES VIRTUELLES

2/1. REPRESENTATION DE LA NOTION DE PRIVILEGES

Il est intéressant de montrer comment la notion de privilèges peut s'exprimer dans le formalisme présenté.

Cette notion, familière aux concepteurs de systèmes d'exploitation, peut se transposer et trouver des applications au niveau de la réalisation technologique des machines informatiques.

ex.: deux organes technologiques peuvent avoir des privilèges d'accès différents à une mémoire commune:

- différence d'espace accessible,
- différence dans les opérations qu'ils peuvent solliciter de cette mémoire.

2/1.1. Environnement

Nous appellerons environnement d'un utilisateur l'ensemble des ressources qu'il peut utiliser.

2/1.2. Privilèges

Les privilèges qu'une ressource accorde à ses utilisateurs correspondront à l'ensemble des requêtes d'utilisation qu'admet l'articulation qui représente cette ressource.

Les privilèges d'un utilisateur correspondront à l'ensemble des privilèges que lui accordent les différentes ressources de son environnement.

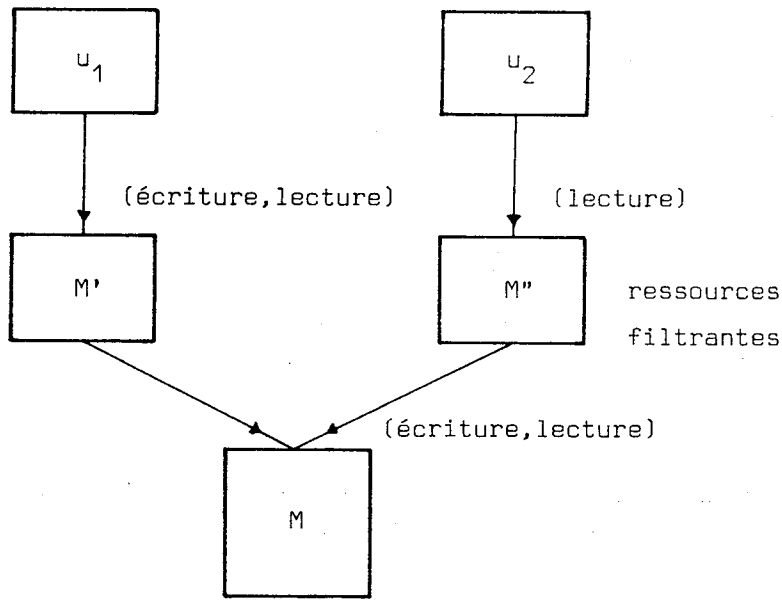
Remarques:

Ce point de vue est opposé à celui habituellement utilisé,qui considère les privilèges comme des propriétés des utilisateurs, alors que nous les considérons comme des caractéristiques des ressources. Sa conséquence directe du point de vue que nous adoptons, réside en la fréquente duplication des

ressources à l'aide d'articulations filtrantes [§ chap.2, 2/1.1.3.] caractérisées par le fait qu'elles n'admettent qu'un sous-ensemble des requêtes de la ressource qu'elles dupliquent. Ce sous-ensemble correspondant aux privilèges accordés par cette ressource à ses utilisateurs.

Un tel point de vue rejoint celui qui consiste à définir les privilèges accordés à un utilisateur à l'aide d'un ensemble de fonctions d'accès aux ressources qui lui sont allouées. Ces diverses fonctions d'accès représentent les articulations filtrantes dont nous nous servirons pour dupliquer formellement les ressources.

Exemple: Considérons la représentation du système constitué par deux êtres u_1 et u_2 pourvus de privilèges différents en ce sens que l'utilisateur u_1 peut lire et écrire les informations dans une mémoire tandis que u_2 ne peut qu'y lire.



ressource mémoire

2/1.3. Modification de l'environnement d'une articulation

Toute modification des privilèges d'un utilisateur revient à une modification de son environnement puisque cette opération revient à une modification de la nature de ses ressources accessibles.

La modification de l'environnement d'une articulation consiste en une modification de la structure du système hiérarchisé qui la contient par adjonction, suppression ou déplacement d'articulations.

2/2. MODIFICATION DE LA STRUCTURE D'UN SYSTEME HIERARCHISE

La modification de la structure d'un système hiérarchisé est une opération dont la complexité dépend de la nature de l'articulation crée ou supprimée.

Considérons les cas suivants concernant la nature de cette articulation:

- Articulation représentant une ressource non principale. Dans ce cas la décision d'insertion ou de suppression peut être prise par l'utilisateur de cette ressource.
ex.: modification, par un utilisateur, de son propre environnement (acquisition ou libération des ressources).

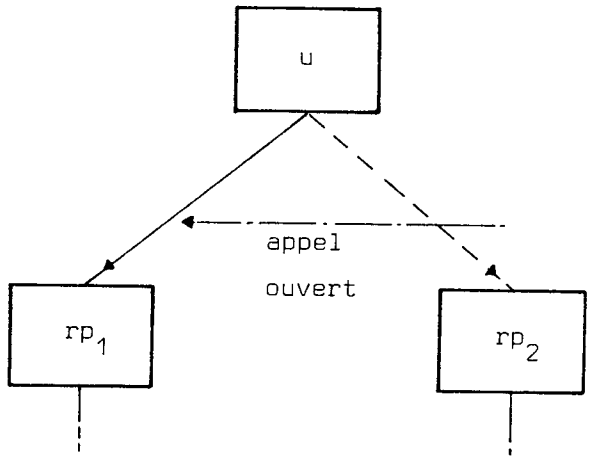
- Articulation principale. Dans ce cas, la ressource représentée par cette articulation pré-existe à ses utilisateurs. Elle n'est détruite qu'après le départ de tous ses utilisateurs, par des articulations appartenant à d'autres ressources ou plus proches de la racine.
ex.: le chargement d'un interpréteur programmé a lieu avant le chargement des programmes qu'il doit interpréter.

- Articulation représentant un utilisateur fictif situé au niveau des feuilles du système (un paramètre de fonctionnement). La décision de sa création ou de sa destruction peut être prise par les articulations qui se situent au niveau inférieur et auxquelles ce paramètre peut être associé.
ex.: création ou destruction de travaux à exécuter.

- Autres articulattions. Le problème devient alors plus complexe car il faut s'assurer, avant de supprimer une articulation, que la ressource

à laquelle elle participe ne sera plus sollicitée. Pour cette raison les décisions de modification de la structure du système ne peuvent être prises que par des articulations connaissant son état global.
ex.: L'opérateur humain qui pilote un ordinateur est en général seul habilit à assurer la mise en, ou hors, service des ressources physiques du système.

Remarques:
Nous pouvons citer l'appel ouvert [1] qui consiste en une opération permettant à un être programmé de changer de processeur. Cette opération correspond dans notre formalisme à une requête émise par une articulation pour changer la nature de sa ressource principale. Son exécution provoque donc une modification locale de la structure du système.



La création et la destruction des duplications virtuelles des ressources se réalisent souvent à l'aide d'un mécanisme simple. Dans le cas de la création d'une articulation filtrante,le seul mécanisme à réaliser consiste à concrétiser et à vérifier le sous-ensemble des requêtes de la ressource initiale accepté par l'articulation filtrante.
ex.: Le mot d'état d'un programme permet de déterminer certaines particularités de la ressource unité de traitement allouable à ce programme.

2/3. ADRESSABILITE

L'adressabilité d'un être par un autre consiste en la possibilité pour le second de nommer le premier.

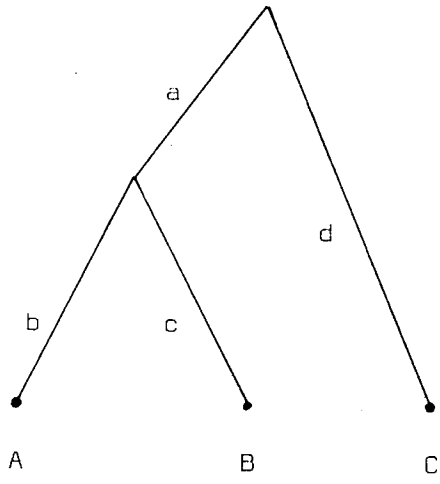
Le besoin pour un être de pouvoir en nommer un autre signifie qu'il existe entre eux une relation d'utilisateur à ressource,

- soit qu'un utilisateur désire nommer les ressources qu'il sollicite
- soit qu'une ressource désire nommer les utilisateurs qu'elle sert.

ex.: Une unité de traitement peut avoir besoin de nommer les programmes qu'elle interprète.

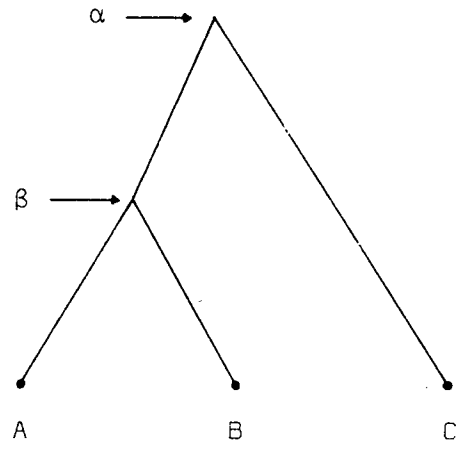
2/3/1. Structure d'adressage

Il est fréquent de considérer que les êtres adressables constituent les feuilles d'une arborescence. Le nom d'un être est assimilé au chemin qu'il faut parcourir pour l'atteindre depuis la racine. Les autres articulations de l'arborescence ne servent qu'à définir la structure de noms utilisée.



nom(A) = a.b
nom(B) = a.c
nom(C) = d

Les êtres habilités à utiliser cette structure (êtres adressants) pour en nommer d'autres se verront affecter un sommet dans cette arborescence, qui pourra éventuellement être la racine. Le choix de ce sommet signifie que l'être adressant ne pourra nommer qu'un sous-ensemble des feuilles dont les noms auront un préfixe commun.



l'être α peut nommer les êtres A,B,C .
l'être β ne peut nommer que les êtres A et B .

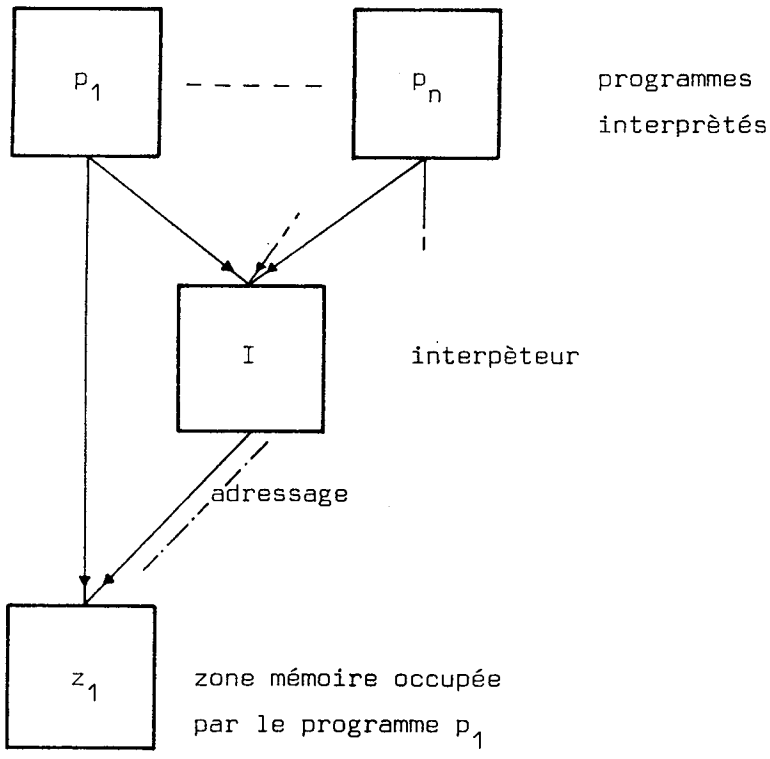
L'ensemble des feuilles adressables par un être donné constitue son espace d'adressage. De tels mécanismes seront utilisés pour unifier tous les espaces d'adressage des êtres d'un système en une arborescence unique. Les espaces d'adressage propres aux différents êtres pourront ainsi être évolutifs et inclus les uns dans les autres.

2/3.2. Relations avec le formalisme présenté

Nous pouvons imposer que les êtres nommés constituent les ressources des êtres adressants. Cette restriction ne nuit pas à la généralité du mécanisme d'adressage si nous admettons l'une des deux hypothèses suivantes:

- la nature d'un être peut varier au cours de sa vie, c'est-à-dire qu'il peut se déplacer dans le système hiérarchisé qui le contient.
ex.: un programme peut être considéré comme un fichier avant qu'il ne soit chargé, puis comme une zône de données en mémoire, puis comme un être interprétable.

- certains êtres ne sont pas nommables mais certaines de leurs ressources le sont.
ex.: sous cette hypothèse, nous pouvons supposer qu'un interpréteur ne nomme pas les programmes qu'il exécute mais la zône mémoire qu'ils occupent.

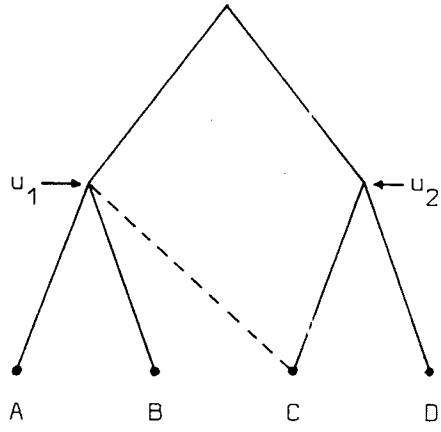


Sous cette condition l'espace d'adressage d'un être coïncide ou est inclus dans son environnement.

2/3.3. Partages de sous-espaces d'adressages

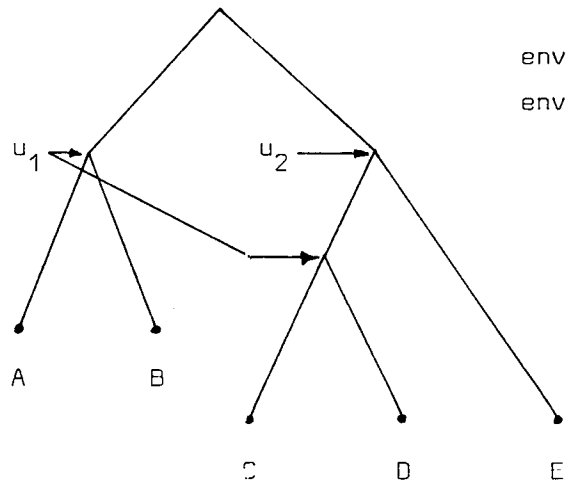
La mise en commun de sous-espaces d'adressages n'est pas toujours compatible avec la conservation de la structure arborescente d'adressage. Cette difficulté est généralement tournée:

- soit en établissant des liens de nature différente qui permettent de conserver à l'ensemble une pseudo-structure arborescente [26, 15].



environnement(u_1) = {A,B,C}
environnement(u_2) = {C,D}

- soit en permettant à un utilisateur d'adresser à partir de plusieurs articulations de la structure d'adressage.

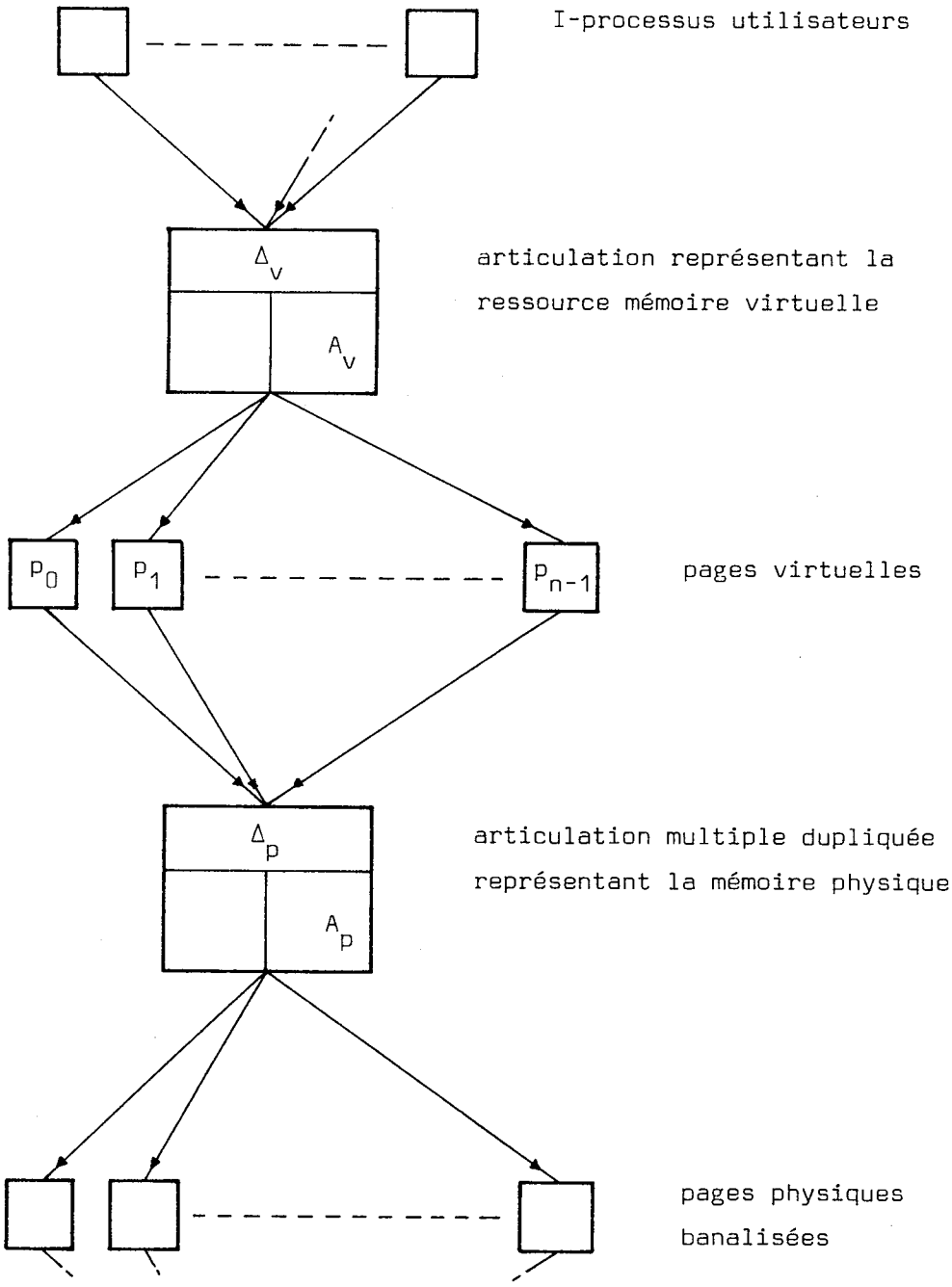


environnement(u_1) = {A,B,C,D}
environnement(u_2) = {C,D,E}

Le désir de maintenir la structure d'adressage arborescente permet de conserver une relation bi-univoque entre les noms et les êtres adressés ce qui facilite la mise à jour des êtres et de la structure elle-même.

2/4. MEMOIRES VIRTUELLES PAGINEES

La notion de mémoire virtuelle consiste à offrir à des utilisateurs la possibilité d'accéder à une ressource mémoire décomposée en n pages individualisées. Comme souvent une telle mémoire est irréalisable physiquement, l'articulation qui représente cette mémoire est transparente et fait appel à des ressources mémoire banalisées, de la taille d'une page et regroupées en une articulation multiple dupliquée.

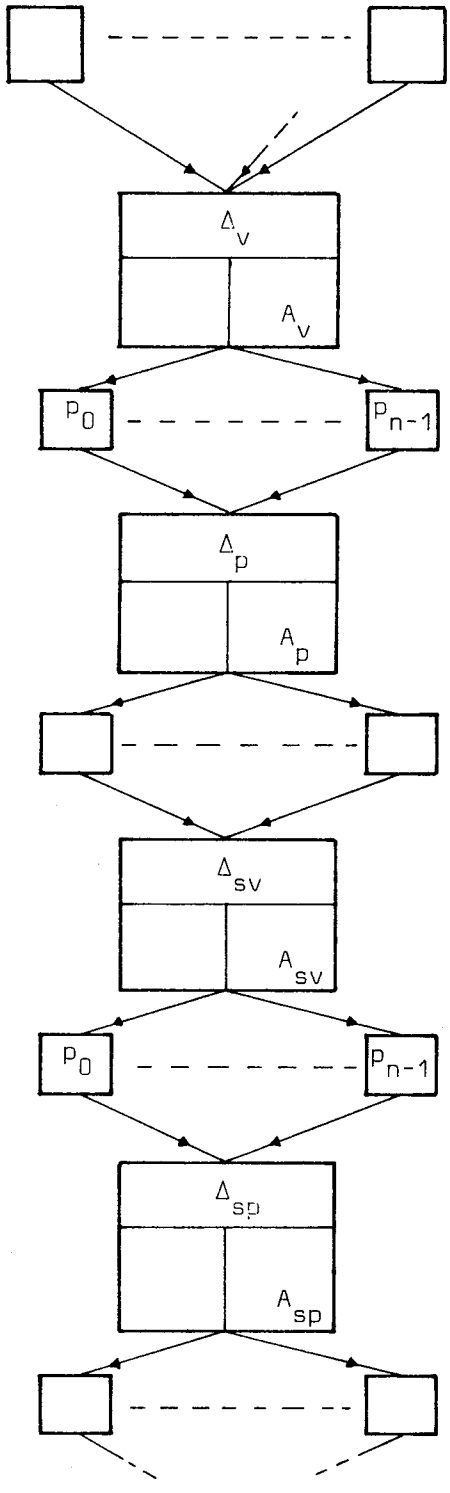


Cette organisation fait apparaître deux mécanismes d'allocation:

- le mécanisme d'allocation A_v de la mémoire virtuelle qui est chargée de l'allocation de cet espace aux différents utilisateurs.

- le mécanisme d'allocation A_p de la mémoire physique est plus important puisque c'est lui qui est chargé de l'allocation des pages physiques aux pages virtuelles (il inclut donc les mécanismes de translation dynamiques d'adresses). Les pages physiques constituent des ressources requerables,équivalentes lorsqu'elles sont libres mai de priorité différente lorsqu'elles sont allouées. La file constituée par $proj_R(AS)$ des pages allouées est gérée:
  - soit en queue,
  - soit ordonnée d'après la date de la dernière utilisation de chaque page physique,
  - soit ordonnée d'après la fréquence d'utilisation de chaque page physique,
  - soit non ordonnée dans le cas où l'on considère les pages allouées comme équivalentes.

Ce mécanisme peut se répéter car les pages physiques peuvent être à leur tour vues comme des utilisateurs qui sollicitent, avec une fréquence plus faible, des informations situées sur une mémoire secondaire plus vaste. Une telle requête apparait chaque fois que le mécanisme A_p décide de changer l'allocation d'une page physique à une page virtuelle. Il s'agit d'un mécanisme d'induction directe. Toute demande d'allocation émise (implicitement) par un I-processus utilisateur se concrétisera par la demande d'une page virtuelle particulière. Cette requête,éventuellement,induira l'allocation d'une page physique de la mémoire primaire à cette page virtuelle Celle-ci induira la requête de la page virtuelle dans l'espace secondaire pour son chargement. Une induction éventuelle sollicitera l'allocation d'une page physique de l'espace secondaire à cette page virtuelle. L'induction directe se propagera dans les niveaux inférieurs de mémorisation jusqu'à ce que l'on rencontre la page virtuelle désirée à laquelle est allouée une page physique.



I-processus utilisateurs

mémoire virtuelle

mémoire physique primaire

mémoire virtuelle secondaire

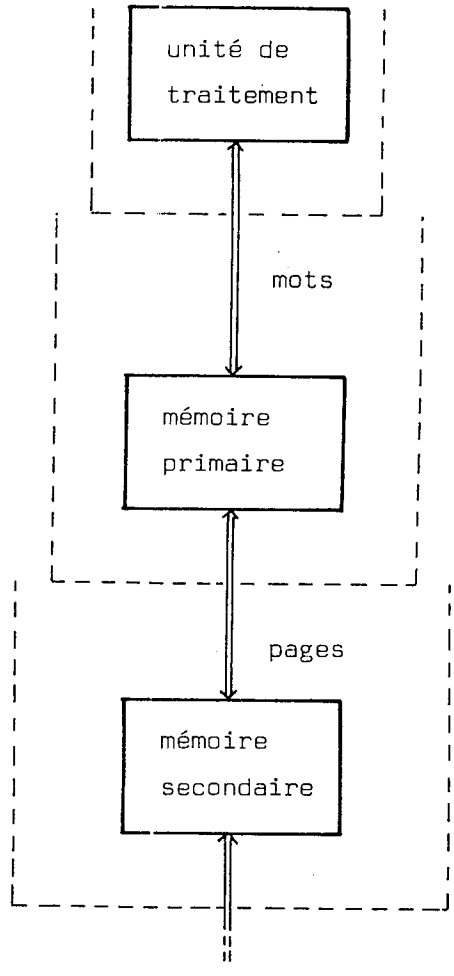
mémoire physique secondaire

niveaux inférieurs de mémorisation

Le mécanisme d'allocation A_{sv} de l'espace virtuel secondaire est trivial puisqu'il correspond à associer à une page physique de la mémoire primaire, la page virtuelle avec laquelle elle est chargée.

Le mécanisme d'allocation A_{sp} de l'espace physique secondaire est similaire dans ses fonctions à celui A_p de la mémoire physique primaire.

Nous voyons ainsi apparaître un empilement d'utilisateurs qui sollicitent à des fréquences très différentes des quantités d'informations également différentes (mots, pages, fichiers,...).



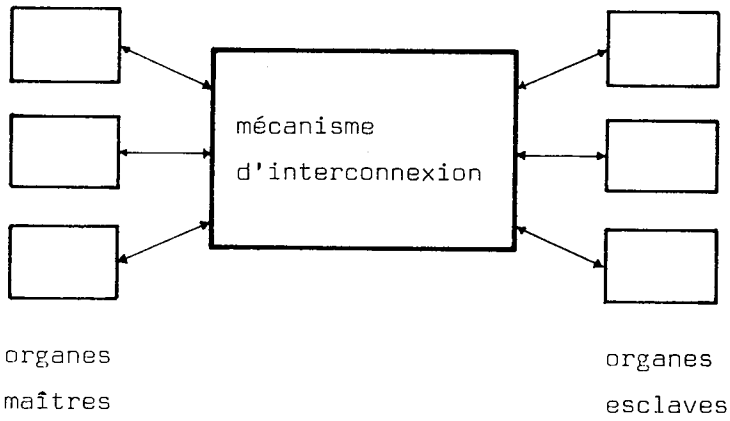
Les mécanismes d'allocation des différents niveaux de mémoire physique sont réalisés de manière technologique pour les niveaux supérieurs (mémoires associatives et micro-programmation) et programmés pour les niveaux inférieurs.

3 - MECANISMES D'ECHANGE DE L'INFORMATION

3/1. MECANISMES D'INTERCONNEXION

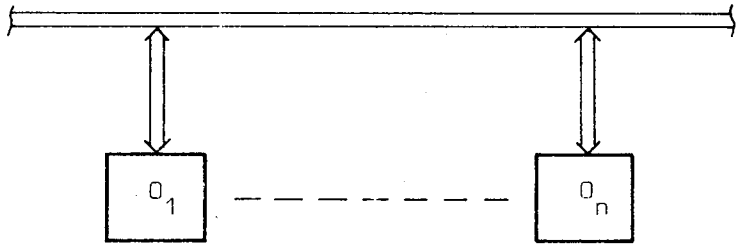
Un mécanisme d'interconnexion est chargé d'établir une communication entre des organes que nous supposerons, dans un premier temps, répartis en deux classes:

- les organes maîtres qui peuvent solliciter l'établissement d'une communication,
- les organes esclaves qui peuvent être sollicités comme interlocuteurs par les organes maîtres.



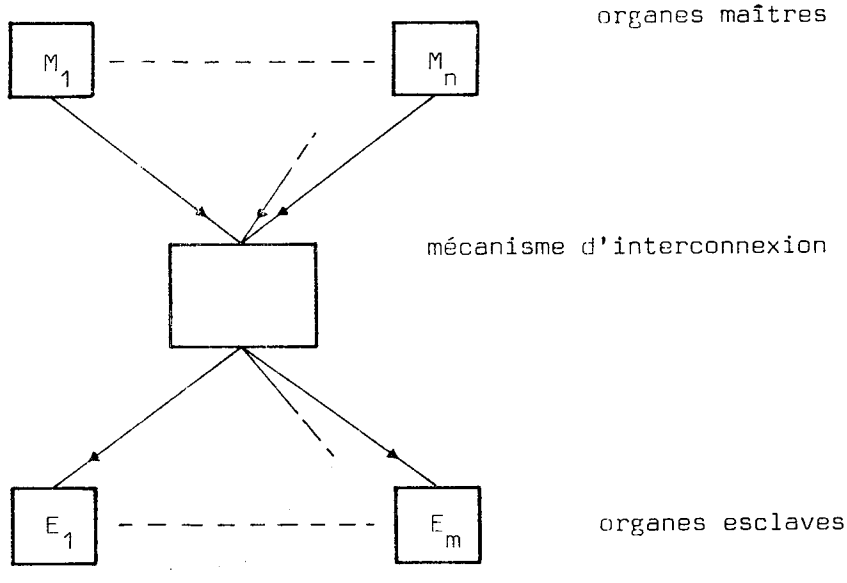
Nous supposerons, dans un premier temps, qu'un mécanisme d'interconnexion ne peut assurer qu'une communication élémentaire à la fois, celle-ci ne mettant en jeu qu'un organe maître et qu'un organe esclave.

ex.: Un bus est un mécanisme d'interconnexion constitué d'un ensemble de lignes d'interconnexion et de circuits annexes de gestion.



organes interconnectés

Sous ces conditions, un mécanisme d'interconnexion se comporte comme une ressource unique sollicitable par les organes maîtres et jouant le rôle d'un utilisateur vis-à-vis des organes esclaves. Il s'alloue, pour une communication élémentaire, à un organe maître et sollicite un esclave. La ressource qu'il constitue n'est pas requérable pendant le temps de cette communication.



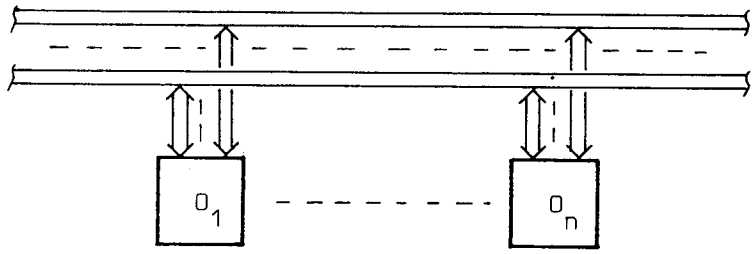
ex.: Un bus est muni d'un mécanisme d'allocation câblé, incluant sa SDS de gestion, réalisé à l'aide d'un circuit de mutuelle exclusion réparti le long du bus [§ annexe 3].

3/2. DUPLICATION DU MECANISME D'INTERCONNEXION

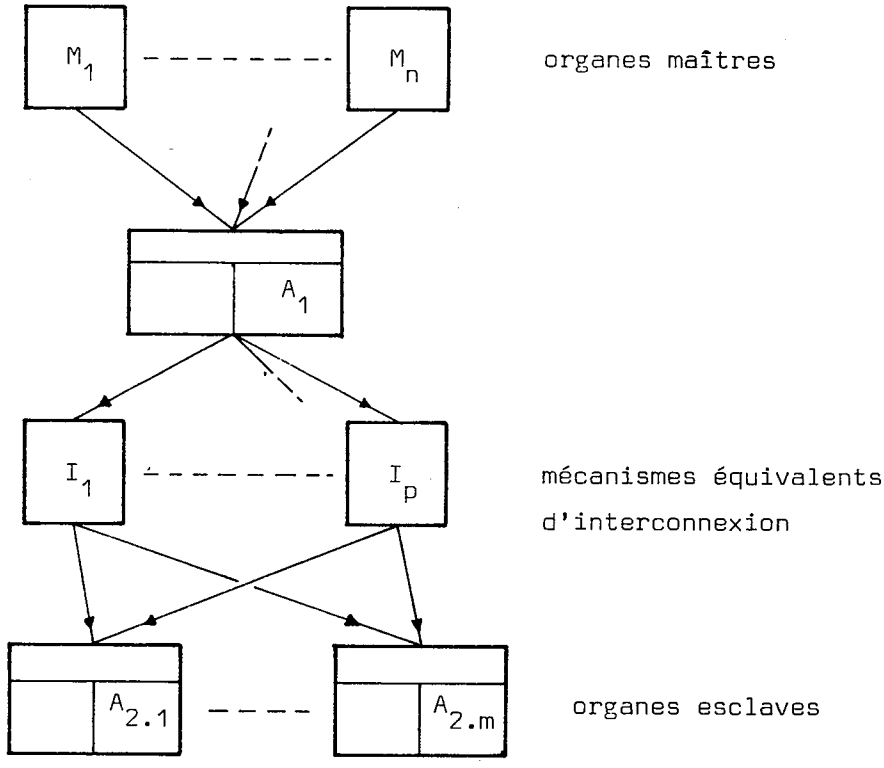
Certains mécanismes d'interconnexion sont conçus pour admettre plusieurs communications simultanées. Cette facilité peut être obtenue:

- soit par une duplication interne du mécanisme élementaire d'interconnexion.

ex.: plusieurs bus peuvent être groupés en parallèle pour obtenir un important débit d'échange d'information.



Le mécanisme d'interconnexion se comporte alors comme une articulation multiple.



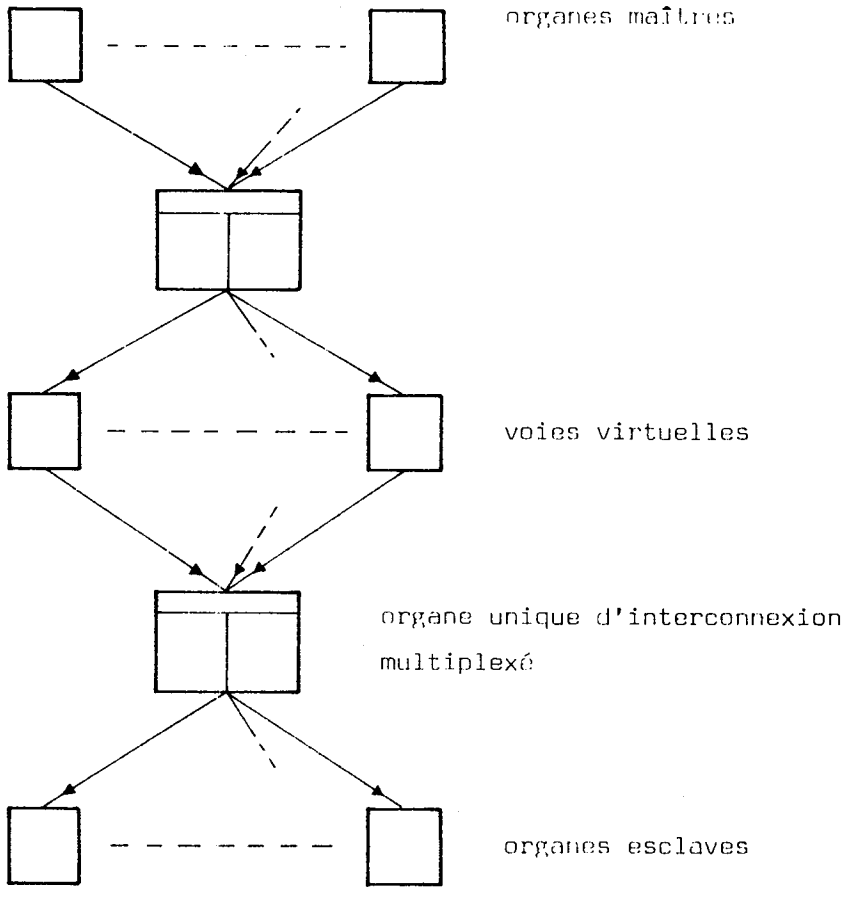
Cette organisation présente l'inconvénient de nécessiter la réalisation de deux niveaux d'allocateurs:

- le mécanisme A_1 qui correspond à l'allocation des différents mécanismes élémentaires de communication aux organes maîtres candidats à u communication.

- les mécanismes A_{2i} qui correspondent au choix qu'il est nécessaire d'effectuer au niveau des organes esclaves en cas de conflit résultant de leur requête simultanée par plusieurs organes maîtres via des mécanismes d'interconnexion différents.

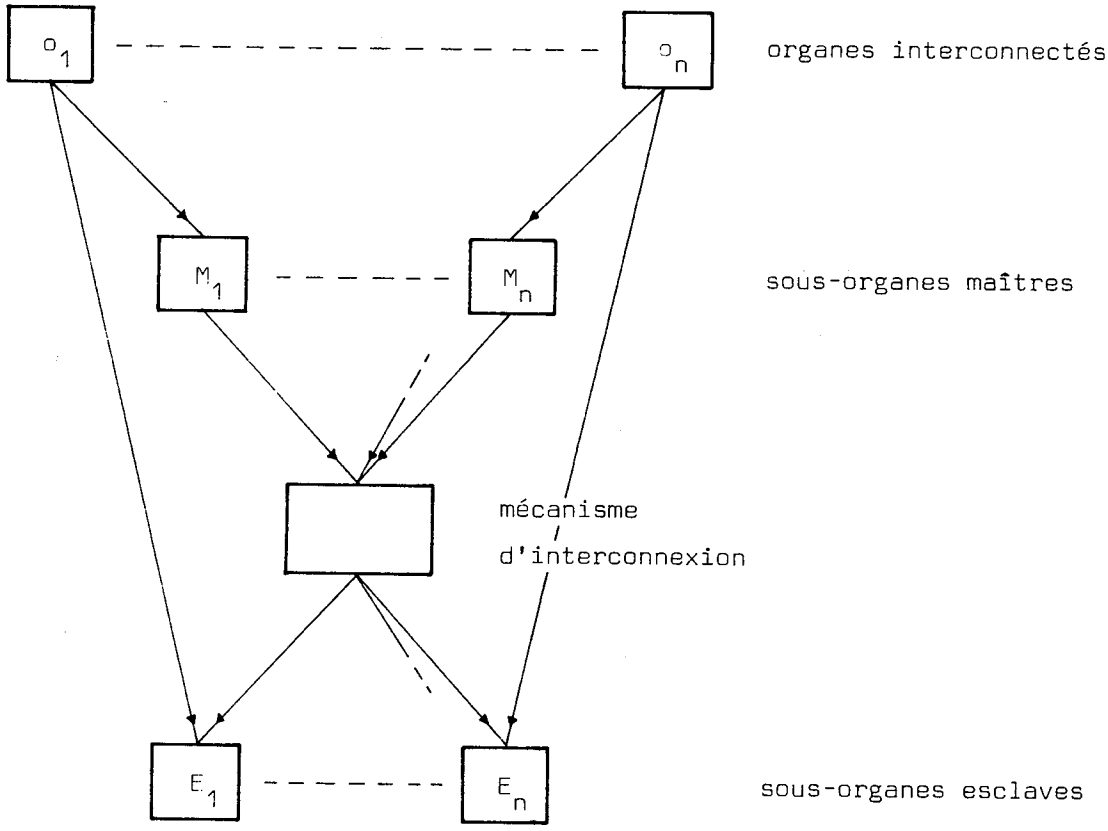
- Soit par le multiplexage temporel d'un organe unique d'interconnexion très rapide.

ex.: Les lignes téléphoniques sont quelques fois échantillonnées, digitalisé et multiplexées sur une ligne très rapide (système MIC)



3/3. ORGANES INTERCONNECTES ALTERNATIVEMENT MAITRES ET ESCLAVES

Lorsque les organes interconnectés sont alternativement maîtres et esclaves , il est simple de considérer chacun d'eux comme le regroupement d'un sous-organe maître et d'un sous-organe esclave.

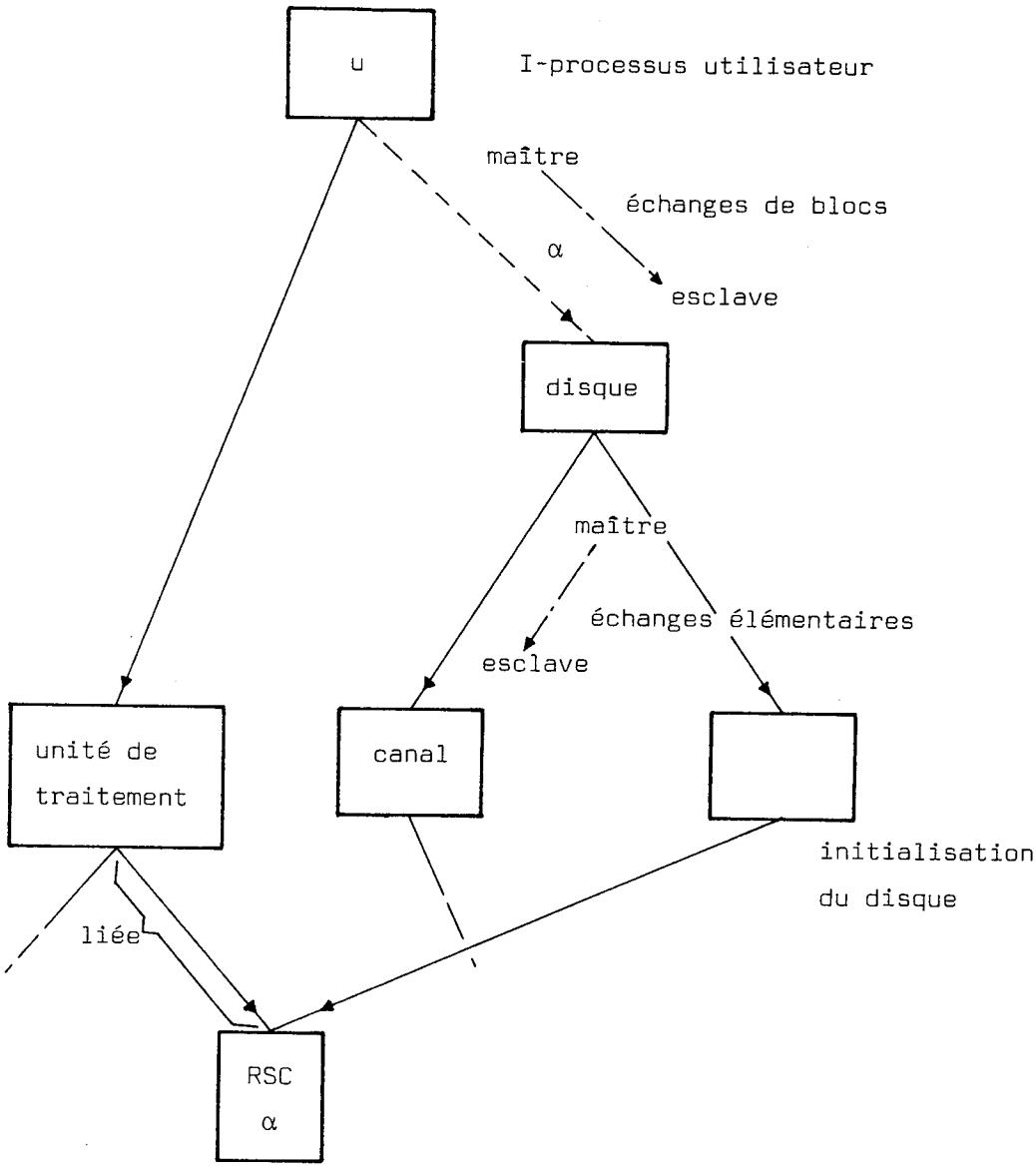


remarque: Cette séparation en sous-organes n'est pas toujours artificielle car souvent les mécanismes nécessaires au comportement de maître sont distincts de ceux utiles au comportement d'esclave.

ex.: Dans le cas d'organes dialoguant par un bus:

- les circuits d'admission d'une requête de communication appartiennent au sous-organe maître.
- les circuits de reconnaissance de l'adresse appartiennent au sous-organe esclave.

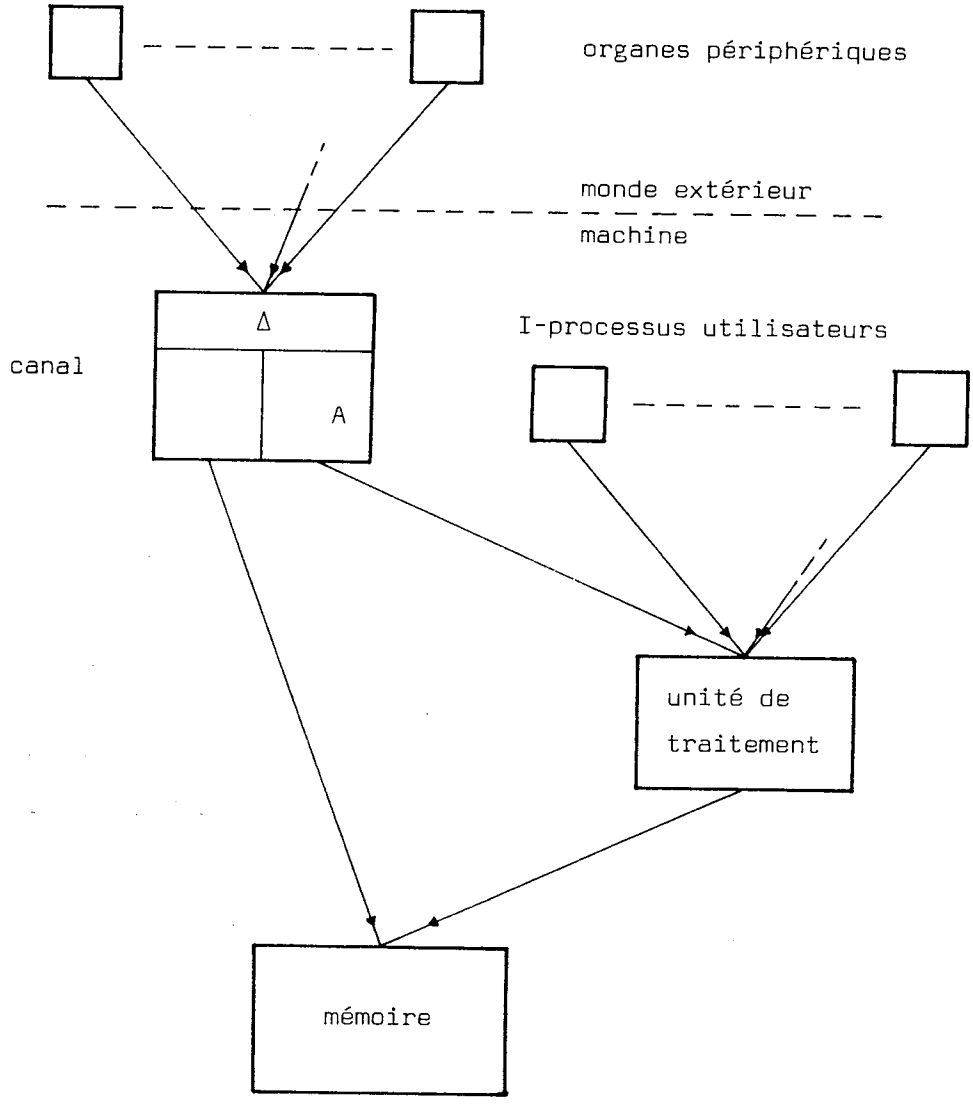
ex.: Dans l'exemple précédent, l'échange d'un bloc se situait entre un programme utilisateur et un disque tandis que les échanges élémentaires se situent entre ce disque et le canal de la machine.



La connexion α étant impossible à réaliser. Elle sera remplacée par une RSC α située entre le disque et l'unité de traitement à laquelle elle est liée.

3/5. CANAL

Dans un ordinateur un canal est un organe, fonctionnellement distinct de l'unité de traitement, spécialisé dans l'exécution des échanges de données avec l'extérieur et ayant accés à la même mémoire physique que cette dernière.

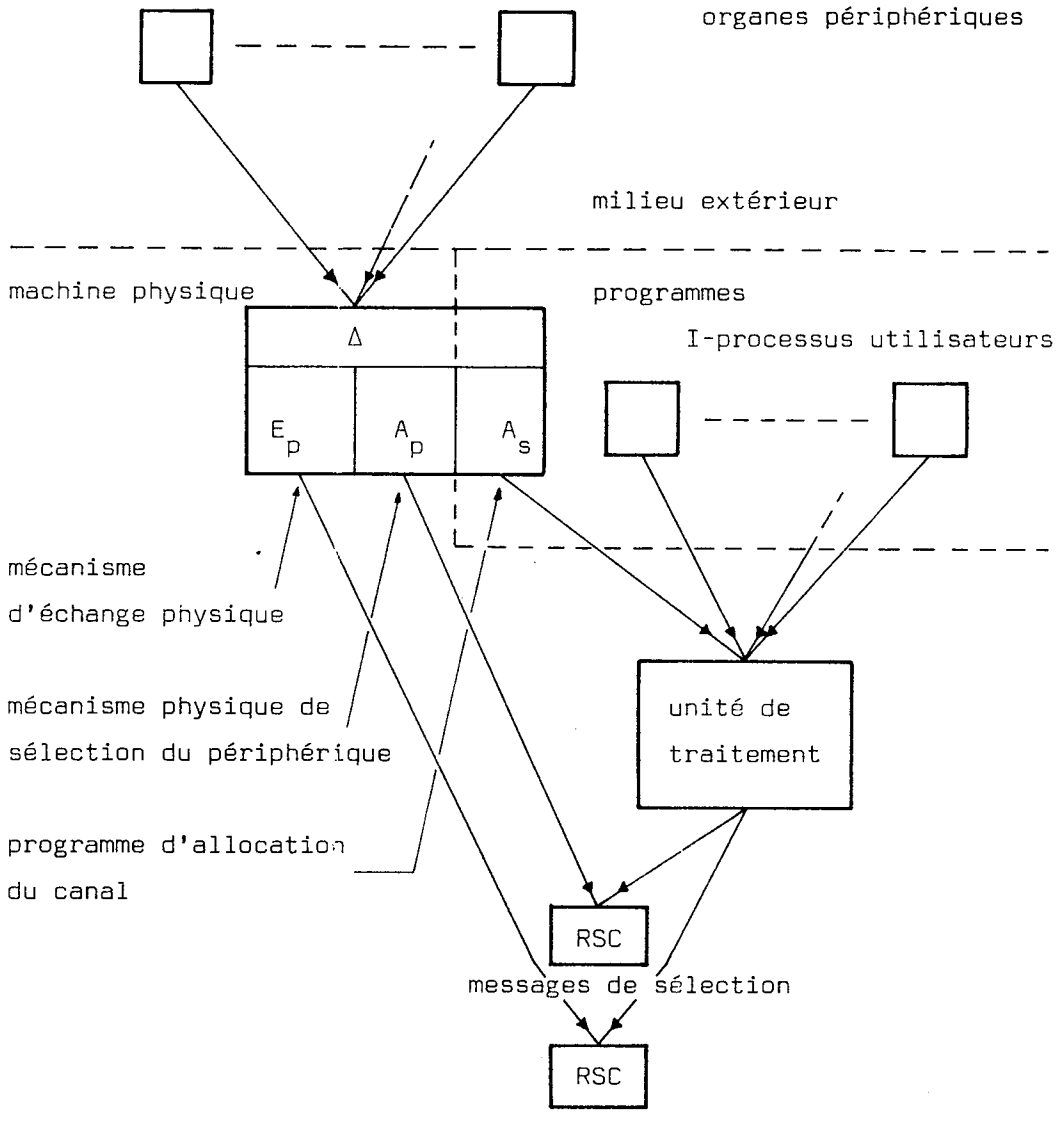


Le mécanisme d'allocation A du canal est chargé de sélectionner l'organe périphérique avec lequel le canal doit travailler. Ce mécanisme de sélection est souvent réalisé par un programme s'exécutant sur l'unité de traitement.

3/5.1. Commandes d'un canal

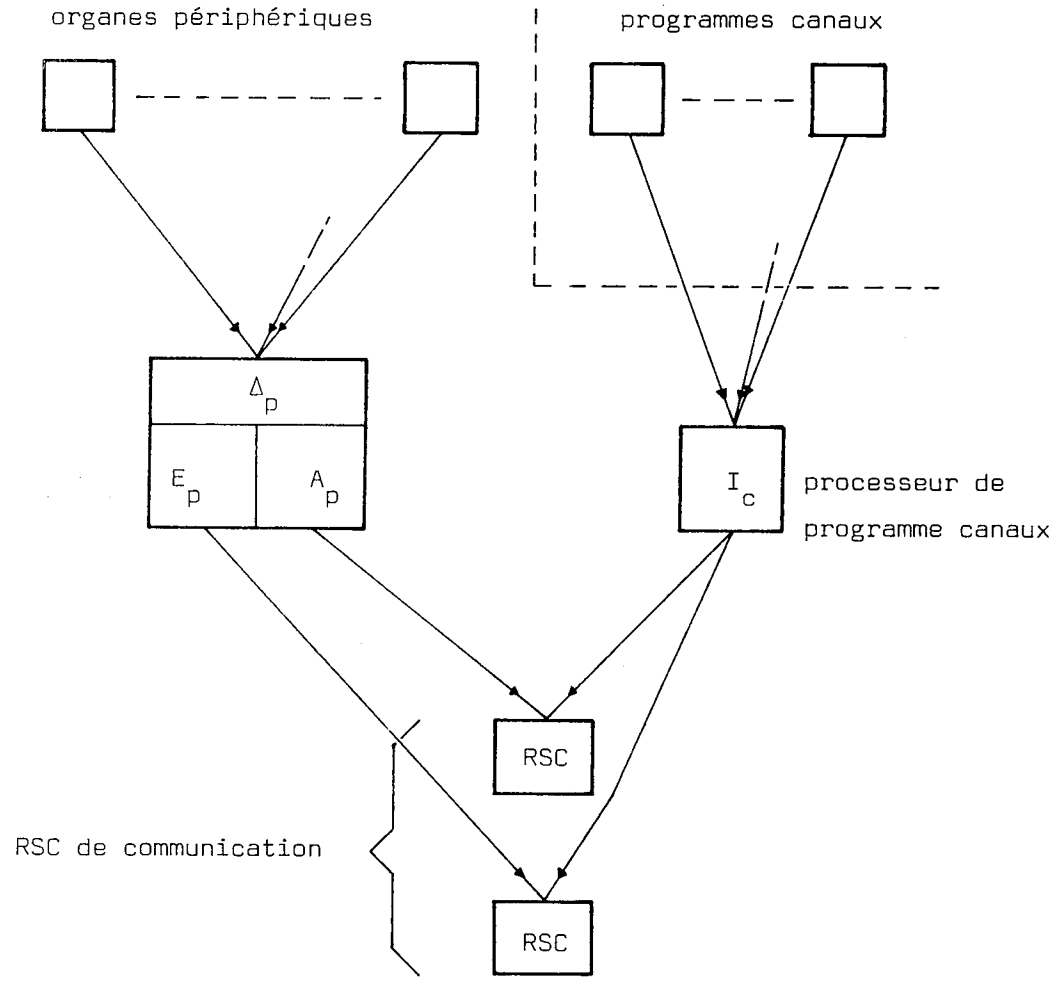
Un canal peut recevoir des requêtes de la part des utilisateurs de la machin de plusieurs manières:

- soit directement à partir de l'unité de traitement par des RSC particulières par lesquelles transitent les informations nécessaires sa sélection et à son fonctionnement.



Remarque: Cet exemple nous montre les mécanismes que l'on doit ajouter sous forme de RSC physiques pour établir la liaison entre les deux constituants mécanisme d'allocation du canal (le programme d'allocation et l'organe physiq de sélection).

- Soit en soumettant à l'exécution d'un processeur particulier un I-processus qui décrit le travail que doit exécuter le canal, ce I-processus est appelé un programme canal.



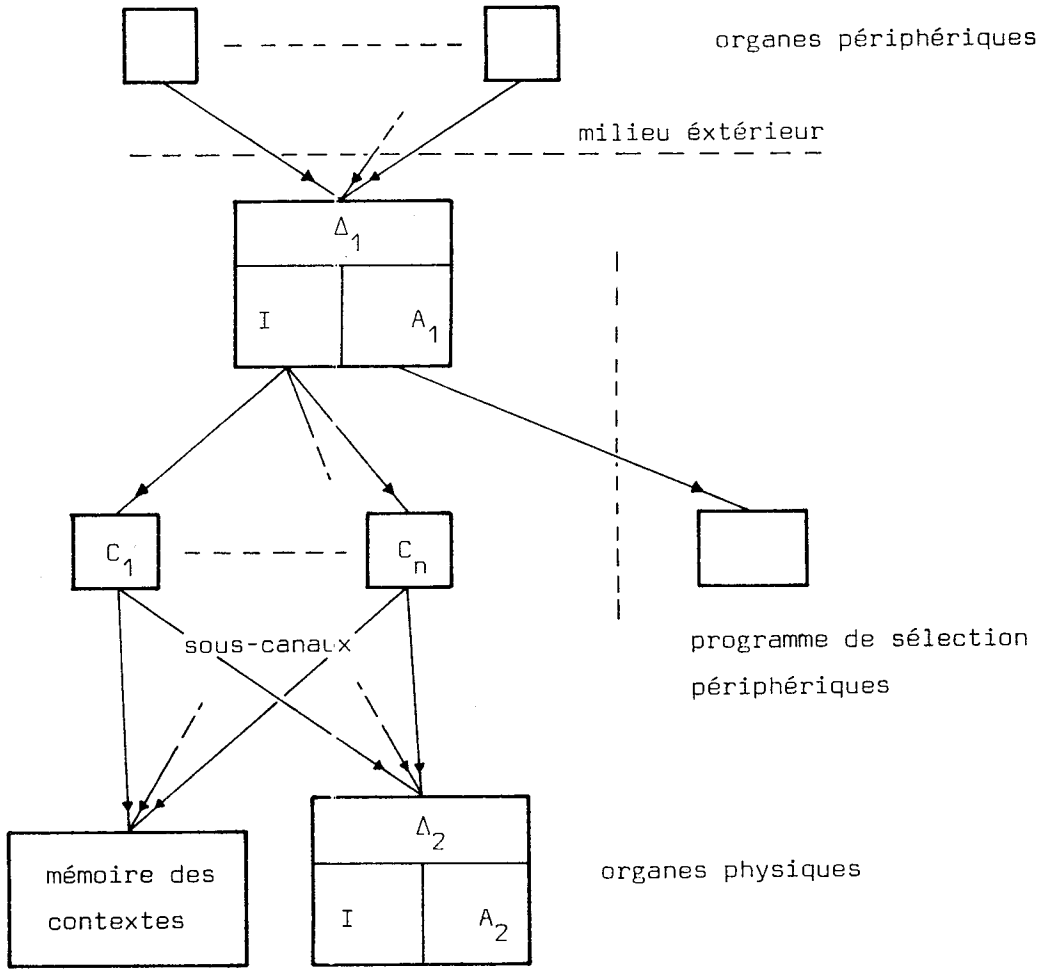
En retour, l'unité de traitement sera avisée de la fin du travail confié au canal,

- soit parceque ce dernier déclenche une interruption particulière du processeur utilisateur.

- soit par l'utilisation des mécanismes normaux de synchronisation disponibles entre les programmes canaux et les I-processus utilisateurs.

3/5.2. Canal multiplexé

Le canal représente un exemple de ressource non requérable. Durant chaque échange de bloc il contient un contexte (adresse mémoire courante, nombre de mots restant à transférer, état...) qui ne peut être annulé qu'à la fin de l échange. Un canal qui reste associé à un périphérique donné durant tout un échange de bloc s'appelle un canal sélecteur. Dans le cas d'échanges de blo avec des périphériques lents, les temps d'inactivité du canal entre chaque échange élémentaire, sont importants. Cette remarque incite à multiplexer l ressource offerte par le canal entre plusieurs échanges de blocs en pseudo-parallélisme. Les contextes de chacun de ces échanges élémentaires doivent alors être rangés dans une mémoire auxiliaire. La solution usuellement rete nue consiste à créer une articulation multiple regroupant un nombre fixe de duplications virtuelles, appelées sous canaux, de la ressource canal.

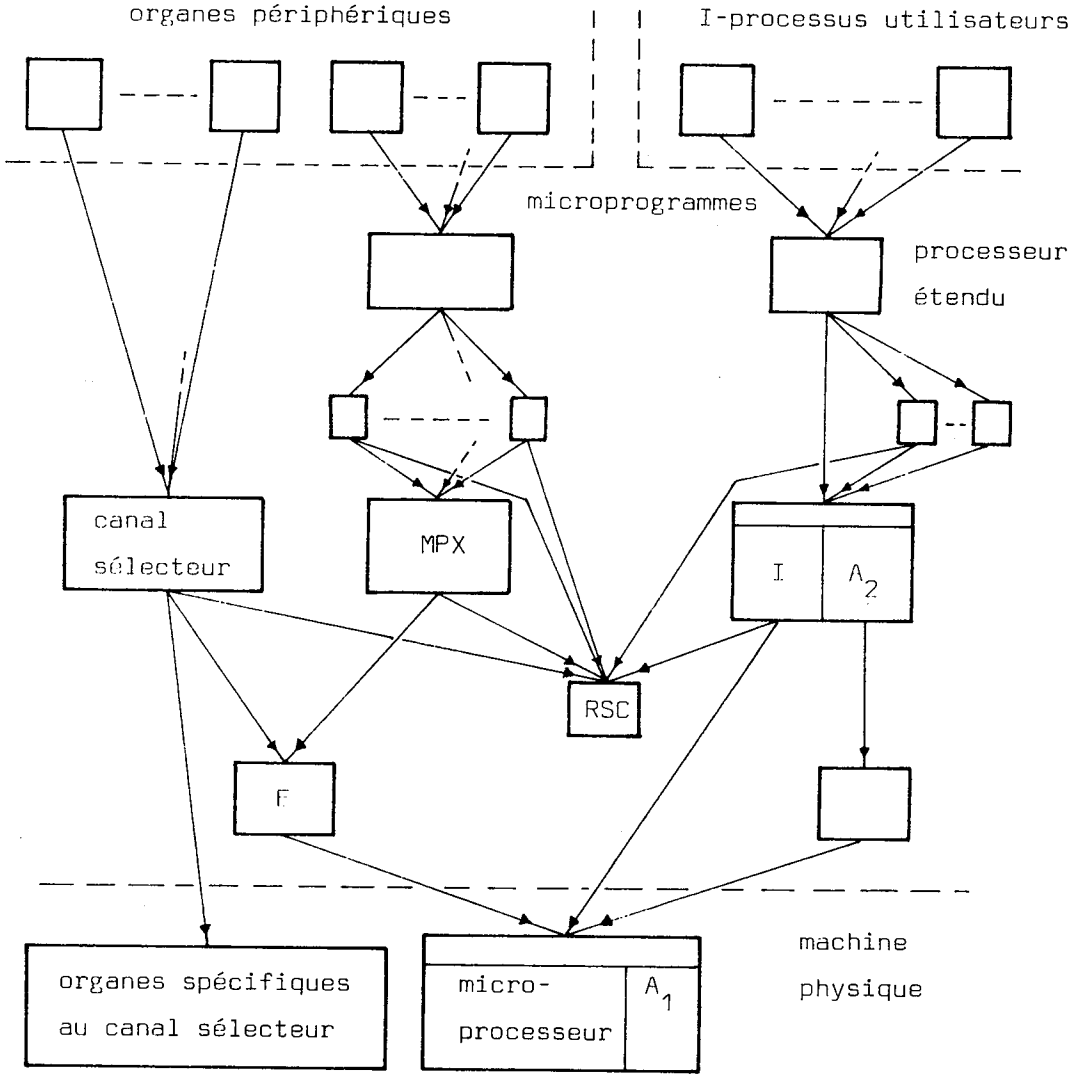


Le mécanisme A_1 associe les organes périphériques aux sous-canaux, il est réalisé par programme (programme de sélection des sous-canaux).

Le mécanisme A_2 alloue les organes technologiques aux sous-canaux pour les échanges élémentaires.

3/5.3 CANAUX INTEGRES

Sur certaines machines [5, 45] l'unité de traitement et les canaux sont réalisés par des micro-programmes différents, s'exécutant sur le même micro-processeur. Ces différents micro-programmes constituent donc les I-processus de ce micro-processeur. Son mécanisme d'allocation est soit réalisé technologiquement, soit micro-programmé.



Le mécanisme d'allocation A_2 gère la commutation de l'unité de traitement vers les différentes articulations d'interruption. Il est réalisé par un micro-programme d'échange des mots d'état programme.

L'allocateur A_1 gère la commutation de la micro-machine vers ses différents micro-programmes. Il est réalisé de manière technologique et agit au moment où le contexte du micro-processeur est minimal ou nul, c'est-à-dire entre l'exécution de deux micro-instructions.

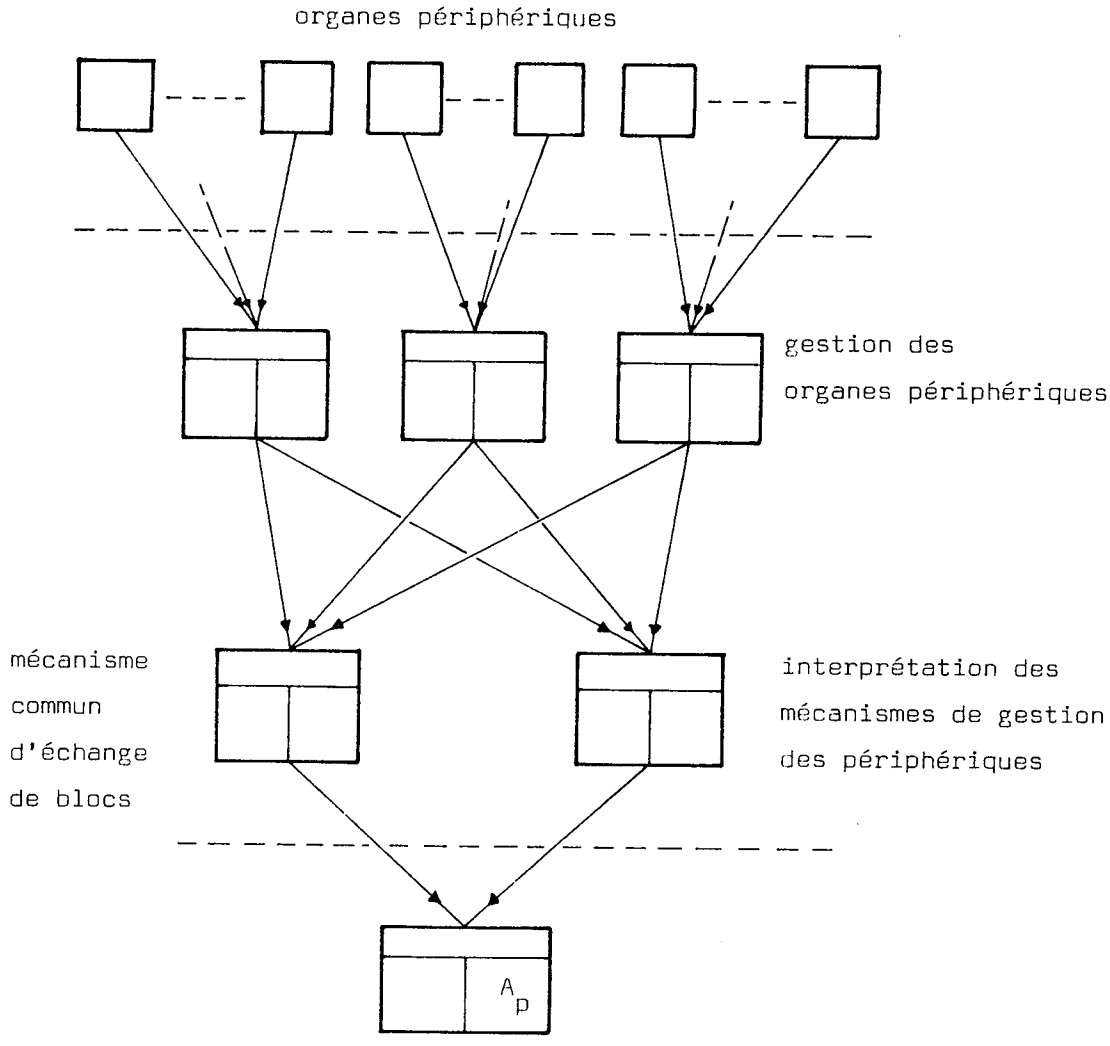
La file FU de la SDS est en partie réalisée par l'allocation dans une mémoire locale, de manière statique, des différents contextes des utilisateurs de cette micro-machine. La commutation est automatiquement réalisée par la modification technologique de la valeur d'un registre de base pointant dans cette mémoire et par rangement automatique dans une pile physique, du contexte résiduel du micro-processeur, s'il existe.

3/5.4. Réalisation sous la forme d'un groupement de processeurs

Certaines machines modernes [46] sont réalisées sous la forme d'un groupement de processeurs:

- l'un de ceux-ci est dédié à la réalisation de l'unité de traitement de l'ordinateur,

- les autres sont chargés des entrées-sorties et simulent le comportement des canaux et quelquefois des organes de gestion des organes périphériques.



processeur d'entrée-sortie

L'allocateur A_u le plus élémentaire du processeur, destiné à la réalisation de l'unité de traitement, servira à construire le mécanisme d'interruption. Il peut être très simple et se résumer au test périodique des conditions externes d'interruption venant du monde extérieur ou des processeurs d'entrée-sortie et des commutations dues aux conditions internes.

L'allocateur A_p le plus élémentaire de chaque processeur d'entrée-sortie doit quelquefois être plus élaboré et plus automatique lorsque ces processe sont soumis à des contraintes temporelles serrées dues au travail de gestio d'organes périphériques rapides. La réalisation de A_p sous la forme d'un mécanisme technologique permet à ces processeurs d'être très rapidement dis ponibles aux requêtes externes et en particulier à celles qui correspondent aux échanges élémentaires d'information.

4 - EXECUTION PARALLELE D'UN I-PROCESSUS

Dans le but d'accroître les performances des machines informatiques, il est fréquent de rechercher dans quelle mesure les instructions de la suite qui définit l'algorithme utilisé par un I-processus, peuvent s'exécuter en parallèle.

4/1. SUITE CHRONOLOGIQUE DES INSTRUCTIONS EXECUTEES

Considérons un I-processus u écrit à l'aide des instructions de l'ensemble $I = \{i_1, \ldots\ldots, i_n\}$. Pour une exécution particulière de ce I-processus, définissons la fonction EX(j) qui donne l'indice de l'instruction de I à exécuter au pas j . La suite $\{i_{EX(j)}\}$ correspond à la séquence chronologique des instructions soumises par u à son processaur.

Définissons récursivement la fonction V qui, pour chaque variable a manipulée par u , donne le nombre des valeurs différentes qu'elle a prises depuis le début de l'exécution jusqu'au pas j .

$$V(a, 0) = 0$$

$$V(a, j) = \left| \begin{array}{l} \underline{\text{si}} \text{ l'instruction } i_{EX(j)} \text{ écrit la variable a } \underline{\text{alors}} \\ V(a, j-1) + 1 \quad \underline{\text{sinon}} \quad V(a, j-1) \end{array} \right. \qquad j \neq 0$$

4/2. TRANFORMATION EN I-PROCESSUS INDEPENDANTS

Associons à chaque pas j de l'exécution particulière précédente de u , un I-processus particulier pj constitué de la seule instruction $i_{EX(j)}$.

Associons à chaque élément de la suite $v(a)_{V(a,j)}$ des valeurs que prend chaque variable durant cette exécution une RSC particulière $R_{V(a,j)}$.

Si au pas j l'instruction $i_{EX(j)}$ écrit dans la variable a , le I-processus pj qui lui est associé doit solliciter l'usage de la RSC $R_{V(a,j)}$ pour l'exécution d'une opération

$$S\ (SDS(R_{V(a,j)}),\ v(a)\ )$$

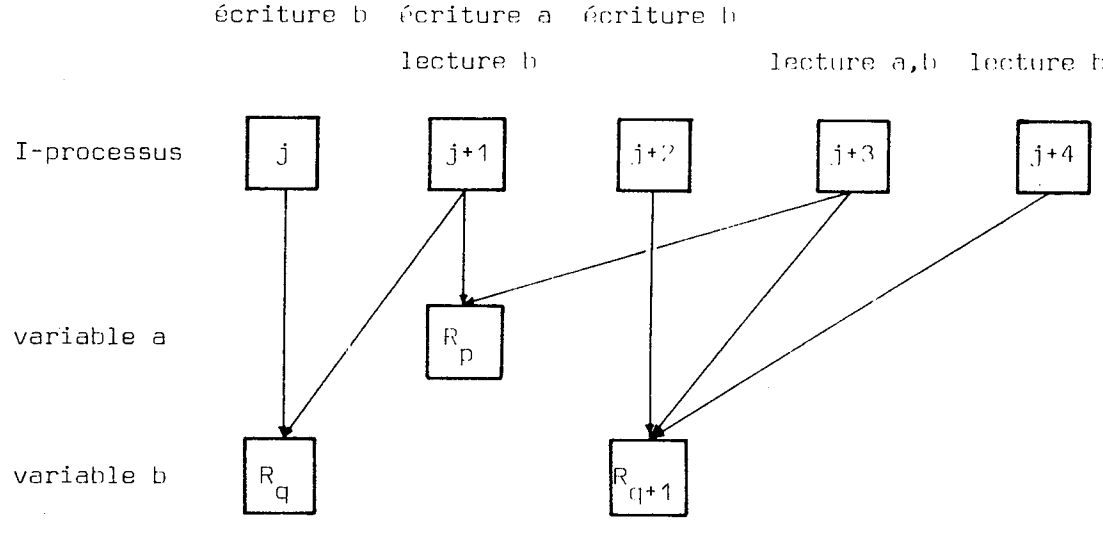
Si cette instruction lit la variable a le I-processus pj doit solliciter l'usage de la RSC $R_{V(a,\ j-1)}$ pour l'exécution de la suite d'opérations:

$$S\ (SDS(R_{V(a,\ j-1)}),\ pj)$$

$$S\ (SDS(R_{V(a,\ j-1)},\ v(a))$$

Remarque:

La seconde opération S de la suite précédente signifie simplement que la valeur de la variable a ne doit pas être extraite de la file Fval de la SDS de cette RSC mais y être lue. Cette seconde opération pourrait être remplacée par le rangement, dans cette file Fval , de m fois la valeur v(a) par une suite de m opérations S lorsque cette valeur doit ensuite être lue m fois.

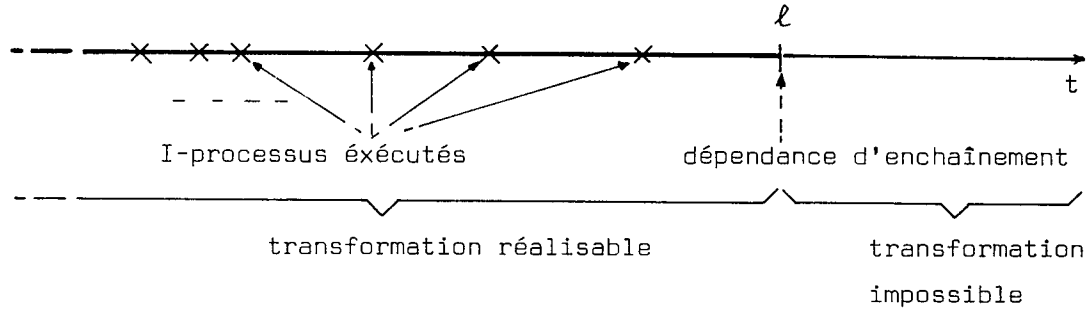


Le fait de lire la valeur d'une variable associée au pas précédent d'exécution, permet de résoudre simplement les problèmes liés aux instructions susceptibles de lire et d'écrire dans la même variable au même pas.

Les I-processus pj peuvent maintenant être candidats à un processeur dès le début de l'exécution du I-processus u . Les RSC introduites assurent la cohérence des résultats en ne permettant l'exécution d'un I-processus pj que lorsque les valeurs des variables qu'il utilise ont été définies (dépendances sur les valeurs des variables).

Remarque:
La suite $i_{EX(j)}$ des instructions soumises au processeur par une exécution particulière ne peut être à tout moment connue que jusqu'au pas P pour lequel la détermination de la prochaine instruction à exécuter après $i_{EX(P)}$ dépend de la définition de la valeur de variables par des instructions précédentes non encore exécutées (dépendance d'enchaînement).

La fonction V ne peut être établie que jusqu'à cette étape de même que la transformation en I-processus indépendants pj .



4/3. DEMARRAGE SEQUENTIEL DES I-PROCESSUS ELEMENTAIRES

Lorsque, d'une part, l'exécution des I-processus élémentaires résultant de la décomposition précédente est débutée dans l'ordre croissant de leurs indices et que, d'autre part, ces mêmes I-processus écrivent les différentes valeurs des variables dans ce même ordre, une importante simplification des mécanismes de synchronisation peut être réalisée. En effet:

- Soit à un instant donné t, un indice courant $m(a,t)$ tel que:

$$m(a,t) = \max_j \{\ pj \mid pj \text{ en exécution à l'instant } t \text{ et } pj \text{ écrit dans la variable } a\ \}$$

soit

$$\partial V(a,j,t) = \begin{cases} \text{si } j > m(a,t) \text{ alors } V(a,j-1) - V(a,m(a,t)) \\ \text{sinon } 0 \end{cases}$$

Pour un I-processus élémentaire pj $\partial V(a,j,t) = 0$ signifie qu'il a à sa disposition, à l'instant t, une valeur de la variable a correcte pour éventuelle lecture.

Soit $M(a,t)$ un autre indice courant tel que:

$$M(a,t) = \min_j \{\ pj \mid pj \text{ lit la variable } a \text{ et } \partial V(a,j,t) \neq 0\ \}$$

Cette définition entraîne que $M(a,t) > m(a,t)$ et que le I-processus $pM(a$ ne peut être en exécution puisque la valeur de a qu'il doit lire n'est p encore disponible.

Soit: $M'(t) = \min_a \{\ M(a,t)\ \}$

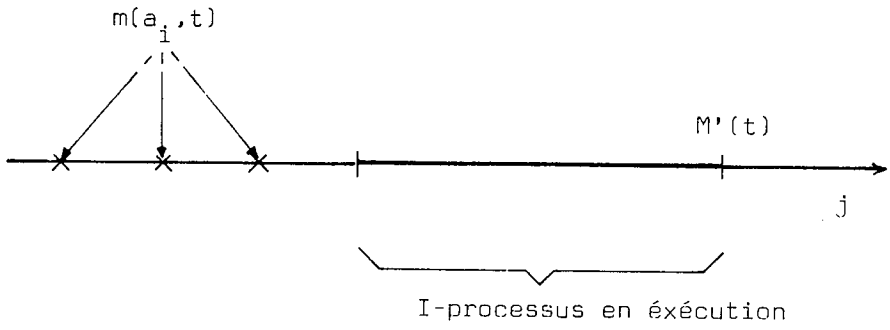
A chaque instant le I-processus $pM'(t)$ ne peut débuter son exécution puisque au moins l'une des variables qu'il doit lire n'a pas la valeur convena Cela entraîne que :

$$\forall\ a \quad M(a,t) \geqslant M'(t) > m(a,t)$$

Pour gérer les dépendances dans une telle machine il nous suffit de conse pour chaque variable a le couple $< v(a)\ ,\ \partial V(a,M'(t),t) >$.

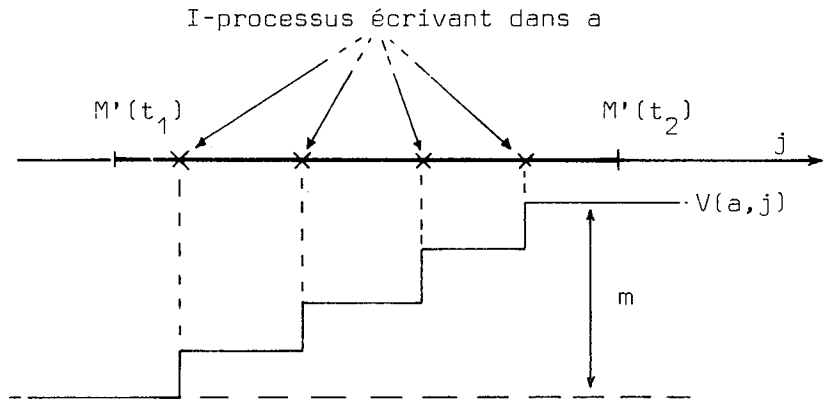
Le front d'exécution, c'est-à-dire l'indice maximal des processus en exéc tion, progressera au fur et à mesure de la progression de l'indice $M'(t)$ d la valeur est elle-même liée à celles des indices $m(a,t)$.

L'exécution du I-processus pj est démarrée lorsque d'une part le I-processus pj-1 est démarré et que d'autre part M'(t)>j. Pratiquement les I-processus élémentaires sont démarrés séquentiellement et chacun de ceux-ci peut arrêter momentanément cette progression si V(a,M'(t),t) ≠ 0 pour une variable a qu'il doit lire.



Dans une telle machine le calcul de chaque ∂V(a, M'(t),t) est fait de la manière suivante:

- le déplacement de l'indice M'(t) de M'(t_1) à M'(t_2) peut provoquer le démarrage de l'exécution de m I-processus élémentaires écrivant dans la variable a . Celà entraîne que:



$$\partial V(a, M'(t_2),t_2) \leftarrow \partial V(a,M'(t_1),t_1) + m$$

en effet

$$V(a, M'(t_2)-1) - V(a,M'(t_1)-1) = m$$

Pratiquement, il suffit d'incrémenter de 1 la valeur de $\partial V(a,M'(t),t$ au démarrage de chaque I-processus écrivant dans la variable a .

- La fin de l'exécution d'un I-processus qui écrit dans la variable a entraîne un déplacement de $m(a,t)$ d'où

$$V(a,m(a,t_2)) \leftarrow V(a,m(a,t_1)) + 1$$

et

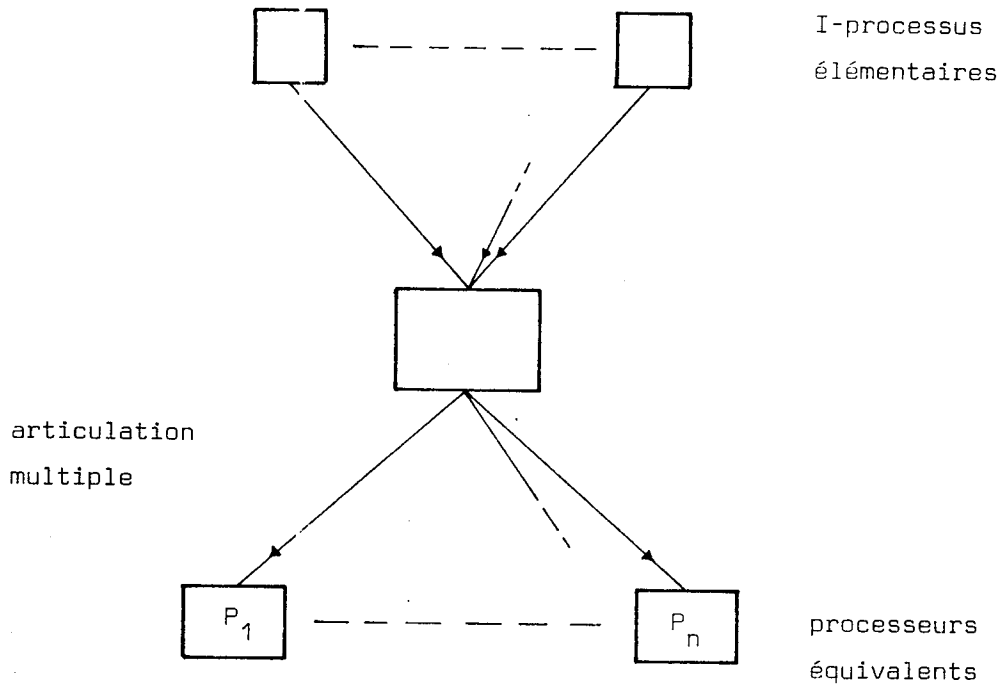
$$\partial V(a,M'(t_2),t_2) \leftarrow \partial V(a,M'(t_1),t_1) - 1$$

Il suffit donc de décrémenter de 1 la valeur de $\partial V(a,M'(t),t)$ à la fin de l'exécution de chaque I-processus écrivant dans la variable a .

4/4. EXECUTION PARALLELE

Moyennant la transformation précédente un I-processus peut être considéré comme décomposé en un grand nombre de I-processus élémentaires exécutable en parallèle dans la limite du respect des synchronisations dues aux dépendances.

Nous pouvons donc envisager l'exécution de ces I-processus élémentaires p des processeurs équivalents P_i regroupés en une articulation multiple.



Le nombre relativement important de dépendances entre les I-processus élémentaires limite fortement le nombre de ces processeurs susceptibles de travailler effectivement en parallèle, à une valeur moyenne de l'ordre de 1,4.

De telles machines démarrent séquentiellement l'exécution des I-processus élémentaires, de manière à bénéficier des simplifications qu'apporte cette solution, d'une part relativement aux mécanismes de synchronisation pour le respect des dépendances sur les variables, et d'autre part du fait qu'il est plus simple de repérer les instructions à exécuter par un pointeur plutôt que de les marquer dans le mécanisme général.

Le choix de cette solution peut être renforcé, si nous remarquons que la probabilité pour que des instructions éloignées soient exécutables,est relativement faible.

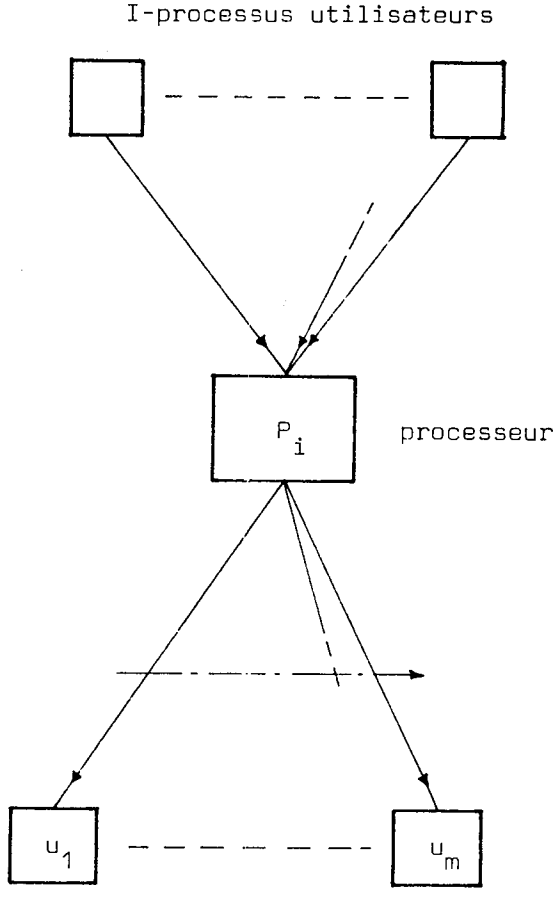
4/4.1. Organisation "leap-frog" [63]

Dans ce type de machine les processeurs équivalents P_i du schéma précédent sont réalisés par des machines physiques.

4/4.2. Organisation "pipe-line" [7]

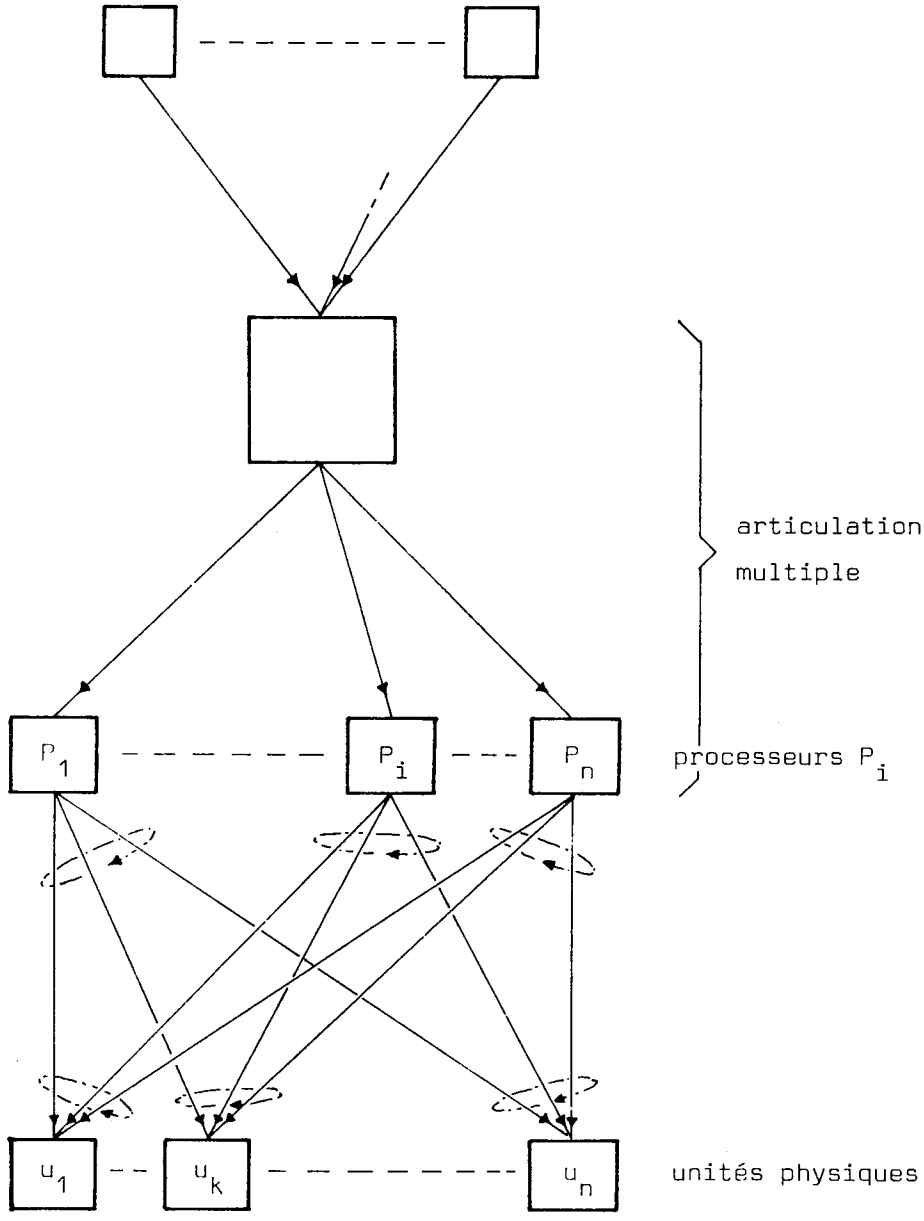
Dans cette organisation, les différents processeurs équivalents sont eux-mêmes décomposés en plusieurs unités U_h correspondant aux différentes étapes de l'exécution d'une instruction.

- appel des instructions en mémoire,
- décodage des instructions,
- appel des opérandes,
- exécution.



Pour exécuter un I-processus élémentaire pj , correspondant à une instruction du I-processus initial, chaque processeur P_i sollicite séquentielle ces unités U_h dans un ordre immuable pour l'éxécution des travaux succes impliqués dans l'exécution de pj .

I-processus élémentaires



Comme un processeur P_i n'utilise, à un instant donné, qu'une seule unité U_h, l'ensemble de ces processeurs peut se partager le même ensemble d'unités physiques.

La contrainte d'exécution séquentielle des pj peut être réalisée en imposant que les unités U_h s'allouent au processeur P_i de telle manière que chaque unité voit, indirectement, défiler les pj dans l'ordre de leurs indices.

Pour cela, à la fin de l'exécution de la phase de travail qui lui a été soumise, une unité U_h exécute

$S(\Delta_{U_h}, U_h)$ pour se dissocier du processeur P_i

$S(\Delta_{U_{h+1}}, P_i)$ pour rendre le processeur P_i candidat à l'unité suivante.

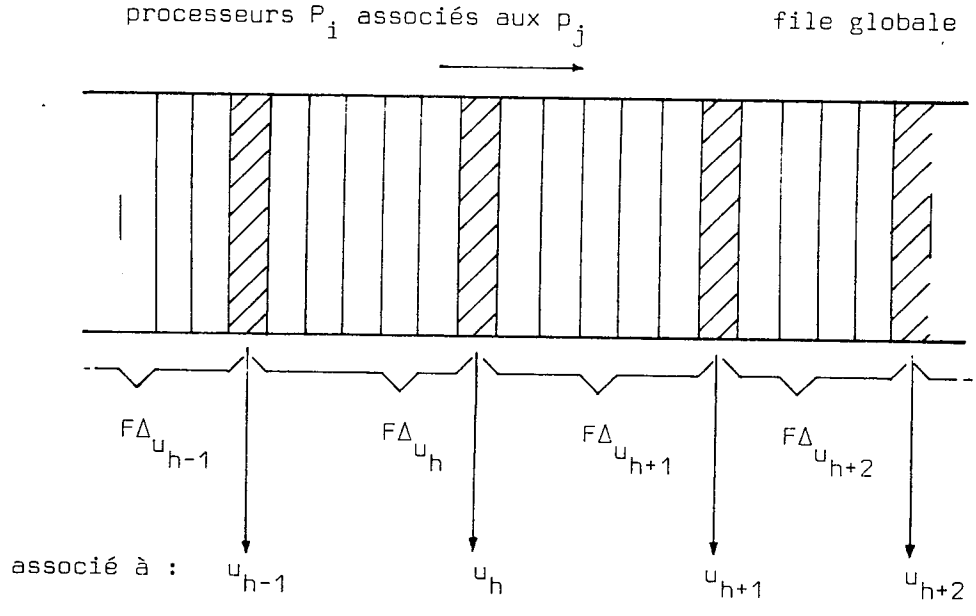
La dernière unité U_m exécutera:

$S(\Delta_{U_m}, U_m)$ pour se dissocier du processeur P_i

$S(\Delta_{P_i}, P_i)$ pour que P_i s'associe à un nouveau I-processus élémentaire pj

$S(\Delta_{U_i}, P_i)$ pour rendre P_i candidat à U_1.

Ce mécanisme est aisément réalisé en regroupant les files $F\Delta_{U_h}$ des SDS de gestion des unités U_h en une file unique (appelée chaîne de montage dans laquelle chaque position représente un processeur P_i. Ceux qui sont associés à une unité U_h sont repérés par un pointeur de ce dernier.



Chaque P_i étant lui-même associé à un I-processus élémentaire pj qui représente une instruction $i_{EX(j)}$ du I-processus initial, une confusion peut être faite en considérant que chaque position de la file globale contient une instruction à différents stades de son exécution.

4/4.3. Organisation barillet "pipe-line" [59]

Une amélioration de l'organisation pipe-line peut être obtenue par le raisonnement suivant:
Au lieu d'essayer de lancer en parallèle l'exécution de plusieurs instructions appartenant au même I-processus utilisateur, ce qui conduit inévitablement à rencontrer des dépendances, il semble plus intéressant de lancer l'exécution, en parallèle, d'instructions appartenant à plusieurs I-processus utilisateurs de manière à ce que chacun d'eux ait, à un instant donné, au plus une seule instruction en exécution.

Ce raisonnement appliqué à une organisation leap-frog, redonne un multiprocesseur classique, appliquée à une organisation pipe-line, elle est susceptible de donner une machine intéressante. En effet:

- les problèmes de dépendance disparaissent totalement, ce qui d'une part simplifie la structure de la machine et d'autre part permet d'en utiliser la pleine puissance, ce qui est rarement le cas pour une machine pipe-line.

- l'adaptation des organes physiques à leur fonction, atout principal de l'organisation pipe-line sur le leap-frog est conservée.

Pour réaliser une telle organisation, une solution consiste à utiliser autant de processeurs P_i qu'il peut y avoir d'instructions en exécution simultanée dans la machine. Chacun de ces processeurs est alloué, de manière statique, à un I-processus utilisateur comme dans le cas d'une organisation multiprocesseur conventionnelle. Chacun de ces processeurs P_i sollicite les unités U_h pour l'exécution d'une seule instruction à la fois d'une manière analogue au comportement des processeurs P_i d'une organisation pipe-line.

De la même manière que dans une machine pipe-line, les différentes unités utilisent une file commune dans laquelle défilent les différentes instructions que la machine exécute et qui appartiennent toutes à des I-processus utilisateurs différents.

Tous les mécanismes de gestion des dépendances (compteurs $\partial V(a,M'(t),t)$ associés aux différentes variables, mécanismes d'arrêt de la progression c I-processus élémentaires pj) sont inutiles.

En contrepartie, comme un multiprocesseur, elle doit comporter autant de contextes de processus associés qu'il y a de processeurs P_i (registres programmables, compteurs ordinaux, codes conditions, etc...).

Le nombre de processeurs virtuels, ainsi créés, doit être limité par leur utilisation et il ne semble pas raisonnable d'en créer un trop grand nombr

TROISIEME PARTIE

HYPERVISEUR MICROPROGRAMME DU GEOPROCESSEUR

HYPERVISEUR MICROPROGRAMME DU GEOPROCESSEUR

Le GEOPROCESSEUR est un calculateur de traitement des données géophysiques issu d'une collaboration entre l'INSTITUT FRANCAIS DU PETROLE (IFP) et l'ENSIMAG [5]. Les spécifications initiales de cette machine furent posées de manière à permettre son emploi sur le site même d'une campagne de prospection pétrolière .

Ce projet a débuté vers Pâques 1970. Après une première phase d'étude en commun de l'architecture générale de la machine physique, la réalisation du prototype fut entreprise à l'IFP tandis que l'ENSIMAG se chargeait plus spécialement de l'étude et de la réalisation des microprogrammes et de quelques programmes du système.

Les micro-programmes furent essayés sur le prototype vers septembre 1972. Leur mise au point, relativement rapide, fut due en grande partie aux nombreuses simulations effectuées préalablement à l'aide de divers modèles programmés de la machine.

Je dois ici, remercier Mr.P.DROUET et les ingénieurs et techniciens de l'IFP qui, en partant des idées initiales, souvent confuses, ont réussi à écrire et à mettre au point les microprogrammes de cette machine. Ils ont de ce fait, largement contribué à la précision et au développement des mécanismes de cet hyperviseur.

La présentation de la structure de cet hyperviseur, dans le cadre de ce travail, doit répondre à deux buts:

- situer dans un domaine concret et réaliste le formalisme présenté.
- proposer une stratégie de réalisation des mécanismes de synchronisation.

Il est en effet apparu qu'un formalisme général, tel que celui que nous présentons, ne devait servir qu'à la compréhension et qu'à l'étude des mécanismes mis en jeu. Les contraintes qui pèsent sur la réalisation qui en découle (contraintes technologiques, d'encombrement, de performance...) sont telles que la réalisation effective de chaque mécanisme local a intérêt à être très spécifique.

Il est à remarquer que très souvent les réalisations ainsi obtenues sont identiques ou voisines de celles qui seraient issues de raisonnements plus directs sans justifications précises. Ce fait, loin de rendre caduc un tel formalisme complexe, le renforce par la vérification qu'il conduit à des solutions connues, donc utilisables.

Cette présentation de l'hyperviseur microprogrammé du GEOPREOCESSEUR n'est pas une notice d'utilisation car certains mécanismes ajoutés par la suite pour étendre son emploi n'y sont pas présentés.

1 - PRESENTATION DE LA MACHINE PHYSIQUE

La machine physique sur laquelle est microprogrammé l'hyperviseur se présente sous la forme d'une unité centrale raccordée au monde extérieur par l'intermédiaire:

- des bus d'entrée-sortie,
- des bus de mémoire centrale,
- de signaux de contrôle dont le rôle sera précisé ultérieurement.

D'un point de vue interne, cette unité centrale est organisée autour de 4 bus internes:

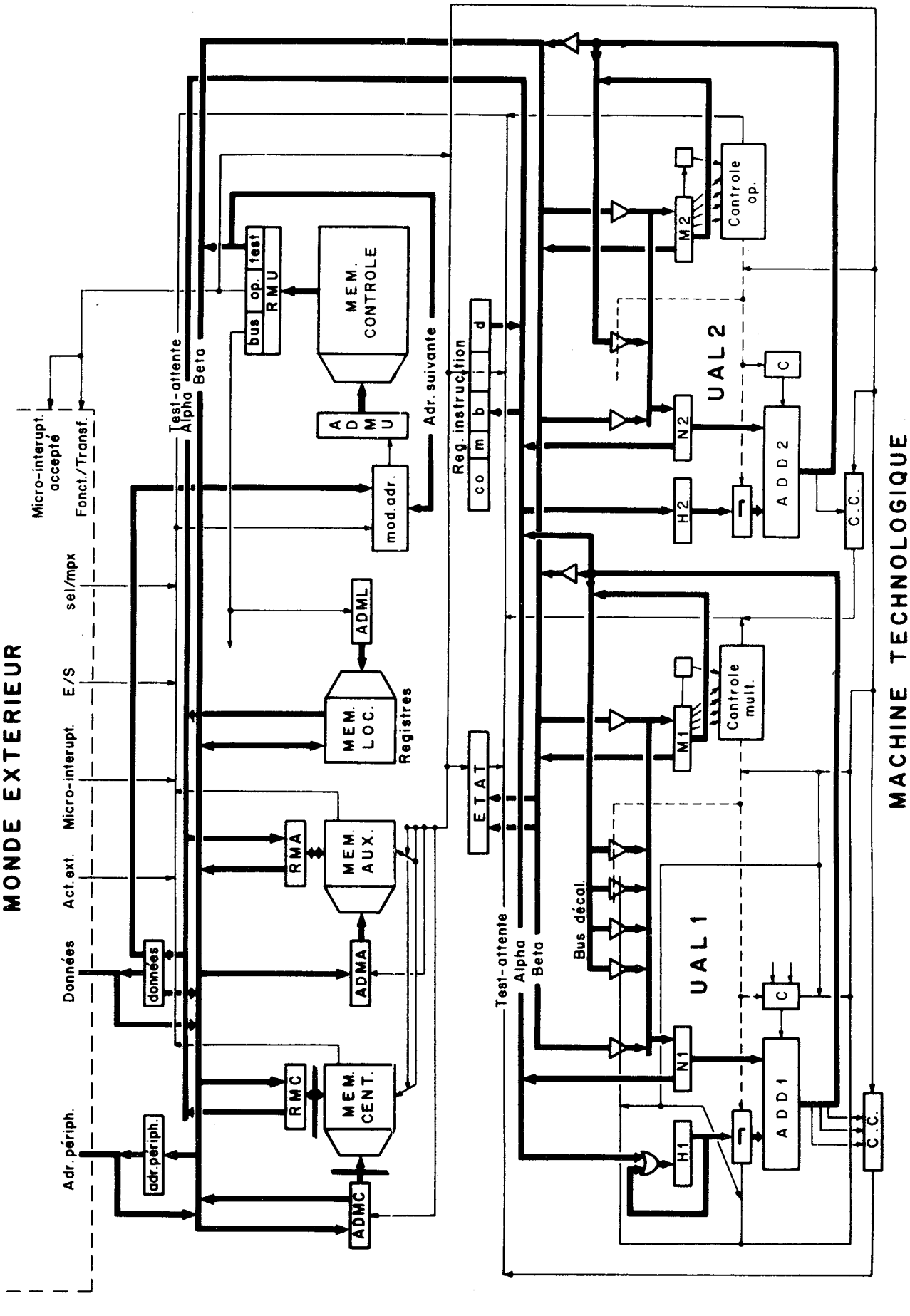
- deux bus de données (α et β),
- un bus de test,
- un bus d'attente.

Les bus α et β sont utilisés pour véhiculer les données entre les différents registres et organes de la machine.

Le bus de test est utilisé pour collecter des informations dans toute la machine. Ces informations sont utilisées par le mécanisme d'adressage de la mémoire de contrôle pour effectuer les tests nécessaires au déroulement du micro-programme.

Le bus d'attente est utilisé pour collecter des évènements dans toute machine (fin de cycle mémoire, fin d'opération arithmétique automatique, réponse du milieu extérieur etc...).

Le microprogramme a ainsi la possibilité d'attendre l'arrivée de l'un de ces évènements. Pour celà, il interrompt son horloge. Celle-ci sera redémarrée par l'occurrence de l'évènement sélectionné sur le bus d'attente.


MONDE EXTERIEUR
MACHINE TECHNOLOGIQUE
Micro-interupt. accepté
Fonct./Transf.
sel/mpx
E/S
Micro-interupt.
Act.ext.
Données
Adr.périph.
adr.périph.
données
Test-attente
Alpha
Beta
bus op. test
RMU
MEM. CONTROLE
ADMU
mod.adr.
Adr.suivante
Reg.instruction
co m b i d
ADML
MEM. LOC.
Registres
RMA
MEM. AUX.
ADMA
RMC
MEM. CENT.
ADMC
ETAT
Test-attente
Alpha
Beta
Bus décal.
UAL 1
UAL 2
M1
M2
Controle mult.
Controle op.
N1
N2
H1
H2
ADD 1
ADD 2
C
C.C.

Les principaux organes de cette machine physique sont:

Deux unités arithmétiques pouvant fonctionner en parallèle:

- la première de ces unités est bâtie autour d'un opérateur universel capable d'exécuter les opérations classiques (addition, soustraction, etc...) sous le contrôle direct du microprogramme.

Un contrôle local câblé permet à cette unité arithmétique d'exécuter des multiplications en virgule fixe, par l'emploi d'un algorithme de BOOTH (sur 3 bits). Seul le démarrage de cette opération est commandé par le microprogramme qui doit ensuite attendre sa fin.

- la seconde unité arithmétique est semblable à la première à quelques différences prés: elle ne peut exécuter sous le contrôle direct du micro-programme que relativement peu d'opérations mais son contrôle local est plus élaboré et permet l'exécution des opérations suivantes:

. multiplication en virgule fixe et flottante,
. normalisation des nombres en virgule flottante,
. conversion des nombres en virgule fixe en représentation virgule flottante et réciproquement,
. décalages complexes.

Quatre mémoires:

- une mémoire centrale d'une taille maximale de 128 k mots reliés à la machine par un bus d'adresses et un bus de données,

- une mémoire auxiliaire qui servira, comme nous le verrons, aux opérations spéciales (FFT, convolution, etc...) et au canal multiplexé,

- une mémoire locale rapide de 16 mots contenant les registres offerts à l'utilisateur, les registres de travail du microprogramme et les contextes d'échange des canaux rapides.

- une mémoire de contrôle de 2 K mots de 48 bits contenant le micro-programme. Le format de ces mots de micro-programme est de type long.
  (Ils sont divisés en un nombre fixe de champs de commande).

Cette machine est conçue pour exécuter un code d'ordre classique utilisant un accumulateur, une extension de cet accumulateur, un compteur ordinal, trois bases, trois index et un masque décrivant les privilèges d'accès à la mémoire centrale. En plus de ces possibilités, le programmeur dispose des facilités suivantes :

- Des opérations de traitement de signal:
  - convolution,
  - transformée de Fourier (FFT),
  - opérations vectorielles.
- Des entrées-sorties évoluées permettant:
  - des départs d'entrée de données sur l'occurrence d'un mot venant de l'extérieur,
  - d'opérations arithmétiques ou logiques entre un flux de données venant de l'extérieur et une table en mémoire.
- Un hyperviseur micro-programmé simplifiant la mise en oeuvre de ses programmes.

2 - STRUCTURE DE L'HYPERVISEUR MICRO-PROGRAMME

L'hyperviseur micro-programmé du GEOPROCESSEUR est organisé pour offrir aux utilisateurs de cette machine un système hiérarchisé de processeurs virtuels .

Ces processeurs virtuels sont:

- deux canaux rapides (sélecteurs),
- un processeur chargé de l'exécution des opérations spéciales,
- 62 canaux lents,

} regroupés en un canal multiplexé

- une unité de traitement.

Les canaux lents sont associés bi-univoquement aux organes périphériques qu'ils ont à gérer.

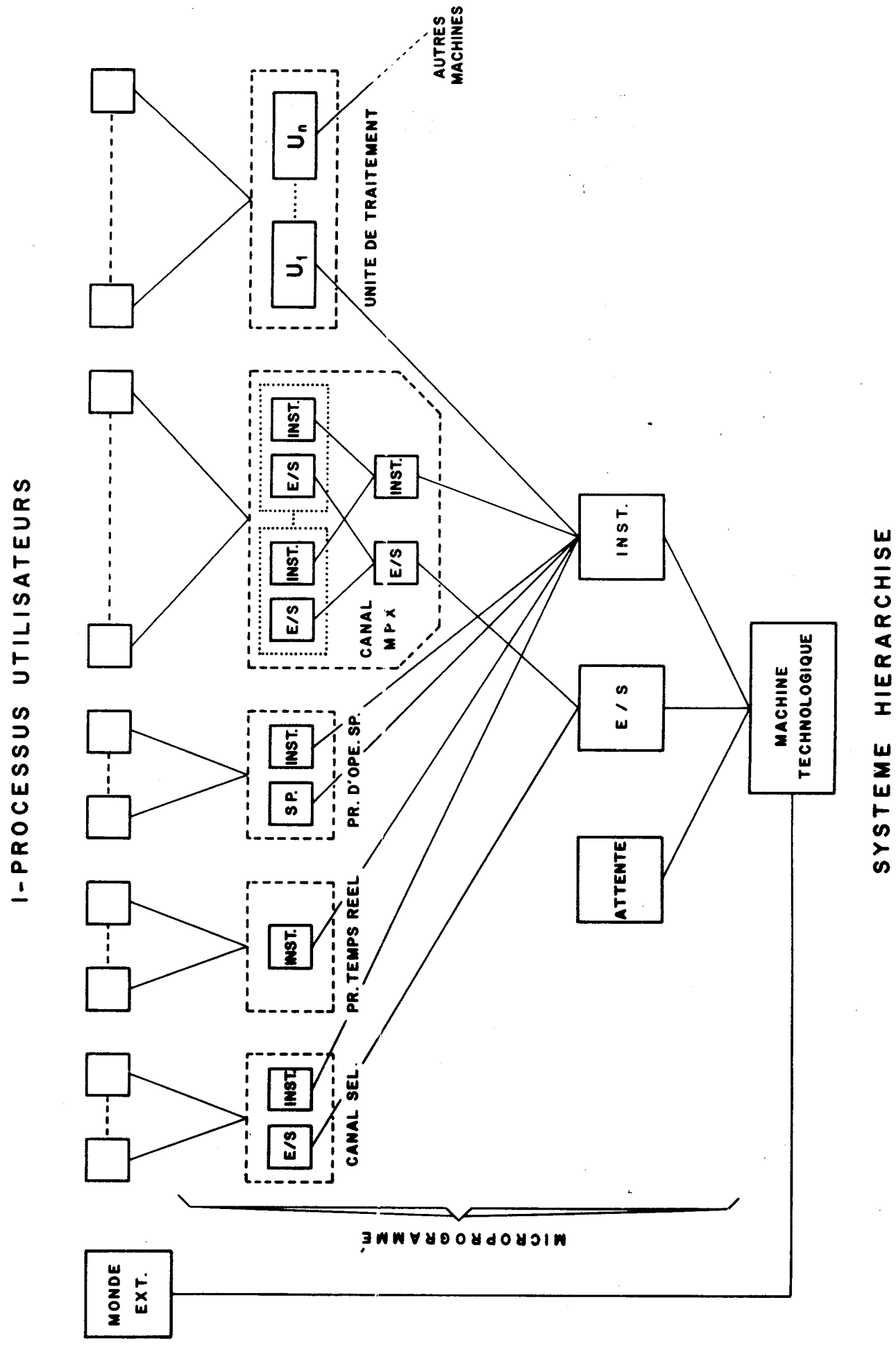
Chacun de ces processeurs virtuels dispose d'un accumulateur, d'une extension de cet accumulateur, de 3 bases et de 3 index. Il peut exécuter, d'une part les instructions d'un ensemble commun à tous les processeurs virtuels comprenant:

- des instructions de mouvement de données,
- des instructions arithmétiques en virgule fixe et flottante fournissant leur résultat dans l'accumulateur en mémoire,
- des instructions de contrôle de l'exécution du programme (tests et branchements),
- des instructions de synchronisation.

En plus de ces instructions, chaque processeur virtuel peut exécuter celles d'un ensemble particulier lié à sa fonction.

L'adressage de la mémoire par les instructions est de type :

(base + déplacement + index)


I-PROCESSUS UTILISATEURS
AUTRES MACHINES
UNITE DE TRAITEMENT
U_n
U_1
INST.
E/S
CANAL M P X
PR. D'OPE. SP.
S.P.
PR. TEMPS REEL
CANAL SEL.
INST.
E / S
ATTENTE
MACHINE TECHNOLOGIQUE
MICROPROGRAMME
MONDE EXT.
SYSTEME HIERARCHISE

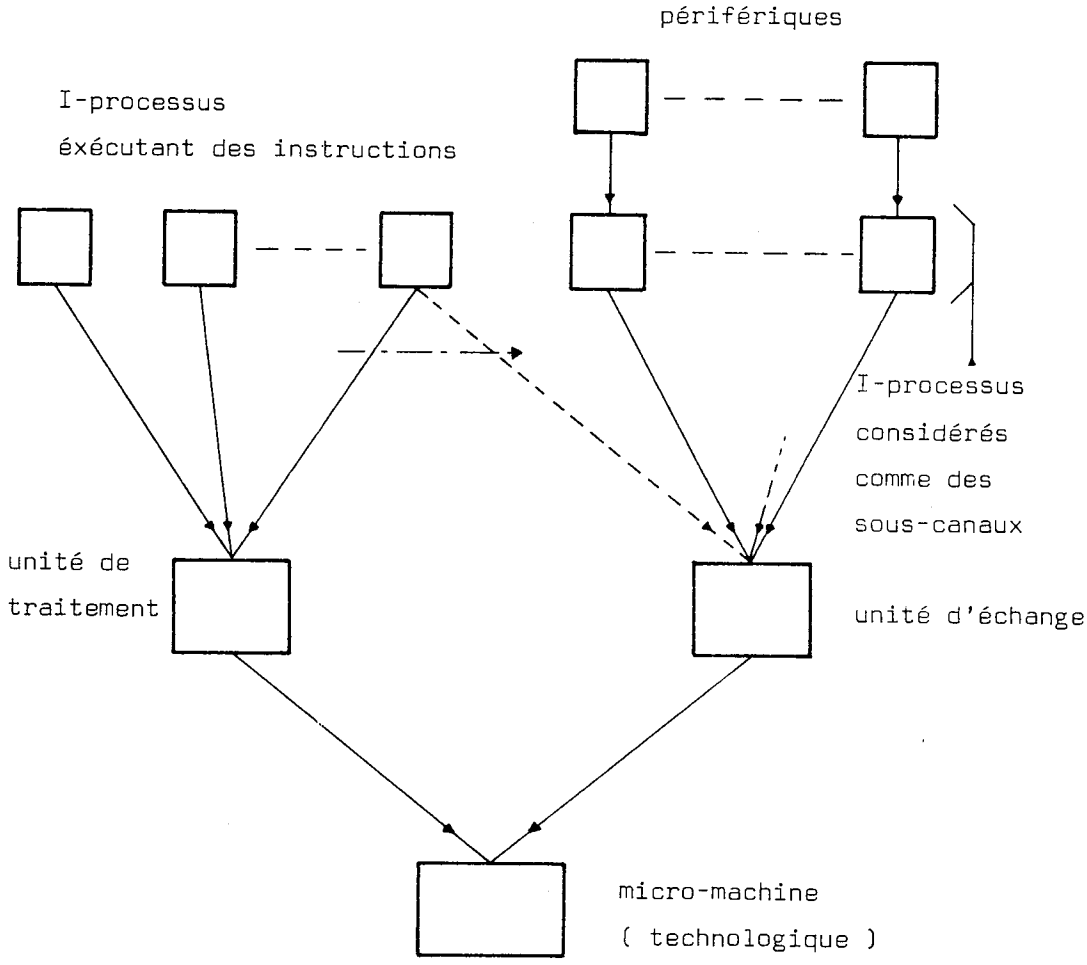
Les programmes devant s'exécuter sur cette machine sont organisés sous la forme de I-processus indépendants. Chacun de ces I-processus ne peut s'exécuter que sur l'un des processeurs virtuels dont l'identité est inscrite dans son descripteur.

Chaque processeur virtuel (canal rapide, sous-canaux multiplexés, unité de traitement) ne peut être associé, à un instant déterminé, qu'avec un seul I-processus. Toutefois leur file de candidats n'est pas bornée. L'unité de traitement est requérable tandis que les canaux ne le sont pas.

Cette duplication des processeurs trouve sa justification dans le désir d'offrir au programmeur une structure statique familière (canaux sélecteurs et multiplexés, unité de traitement). La présence de plusieurs canaux permet d'exécuter en quasi-simultanéité plusieurs échanges de bloc.

Une autre solution aurait été d'offrir seulement deux ressources au programmeur:

- une unité de traitement requérable,
- un mécanisme apte à exécuter les échanges élémentaires avec les organes périphériques.



Comme dans l'organisation réalisée, un seul I-processus aurait été associé à l'unité de traitement à un instant donné. Toutefois celle-ci aurait décodé les instructions d'échange qui auraient provoqué le transfert du I-processus sur le mécanisme d'échange durant lequel un I-processus se serait comporté comme un sous-canal vis-à-vis d'un périphérique particulier.

L'option prise,lors de la conception de l'hyperviseur du GEOPROCESSEUR, de considérer les canaux comme des processeurs particuliers pouvant être sollicités par plusieurs I-processus candidats ne semble plus,à postériori, intéressante, car il est apparu que la gestion d'un organe périphérique demande une certaine cohérence d'action incompatible avec la possibilité d'accueillir tout I-processus candidat sur le sous-canal qui le gère.

Une solution préférable aurait sans doute consisté à admettre que seul un I-processus puisse gérer un organe périphérique c'est-à-dire être lié à un canal particulier pour en étendre les possibilités. Cette insertion d'un I-processus dans le système hiérarchisé de ce canal devant être faite par une opération spéciale indépendante des opérations de synchronisation.

Sur la machine réalisée il est possible de limiter, par convention, à un le nombre des I-processus susceptibles d'être candidat à un canal. Cette solution n'est pas entièrement satisfaisante car rien ne peut empêcher un I-processus parasite de se porter candidat à l'association avec ce canal,entre deux associations du I-processus seul habilité à l'utiliser.

La modification de l'hyperviseur pour permettre une association fixe des processus sur les canaux est réalisable mais se heurte au double emploi de la mémoire auxiliaire (§ 4.2.1.).

2/1. CONFIGURATION MULTIPROCESSEUR

Plusieurs machines peuvent être reliées dans une configuration multiprocesseur à mémoire commune banalisée. Les bus d'adresse et de données des mémoires de ces machines sont alors mis en commun. Ces bus gèrent eux-mêmes la mutuelle exclusion de leur utilisation. Il en est de même pour chaque banc de la mémoire. Il est donc possible de les faire travailler en parallèle. Dans cette configuration les unités de traitement

virtuelles sont groupées en une articulation multiple dupliquée admettant un seul ensemble de processus à traiter. Les canaux restent liés à chaque machine physique dans le but d'étendre les possibilités de connection des périphériques.

3 - SYNCHRONISATION DU SYSTEME HIERARCHISE

Chaque articulation de ce système hiérarchisé de processeurs est gérée par une SDS Δ manipulée par des opérations S. La réalisation de ces SDS et de ces opérations S varie pour chaque niveau. Nous utiliserons la terminologie suivante pour les opérations de synchronisation:

- S au niveau des utilisateurs,
- μS au niveau des processeurs virtuels,
- nS au niveau de la machine physique.

Toutes ces opérations sont exécutées par le microprogramme. La mutuelle exclusion d'exécution correspond ici à une indivisibilité aisée à obtenir puisqu'il n'y a qu'un seul exécutant physique.

3/1 - SYNCHRONISATION DES I-PROCESSUS UTILISATEURS

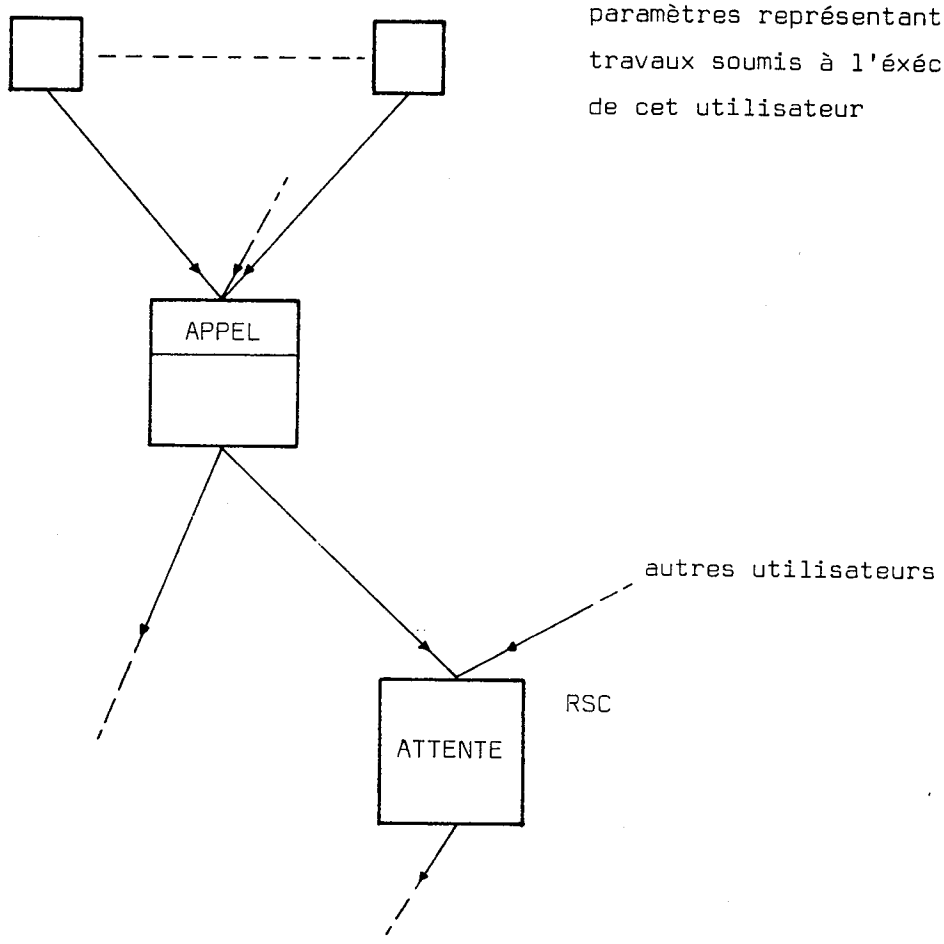
Les articulations qui représentent les I-processus utilisateurs reçoivent l'indication des travaux qu'elles ont à exécuter sous la forme de paramètres considérés comme les feuilles de la hiérarchie. Chacun de ces paramètres appartient à un utilisateur qui le crée, le soumet à l'exécution de l'articulation qui représente un autre utilisateur pour lui donner un certain travail à exécuter puis le détruit une fois ce travail achevé.

Remarque:

> Les utilisateurs de cette machine se comportent comme des processeurs de paramètres, ce qui implique que les I-processus écrits pour elle soient cycliques.

Une SDS qui lui est liée, gère l'association d'un utilisateur avec les paramètres qui lui sont soumis. Cette SDS est nommée APPEL et regroupe en une file unique une queue de paramètres FP et la file $proj_p(AS)$.

Chaque utilisateur peut également attendre par le biais d'une RSC qui lui est liée la fin d'un travail résultant d'un paramètre qu'il a soumis à l'exécution d'un autre utilisateur. La SDS privée de cette RSC est appelée ATTENTE.

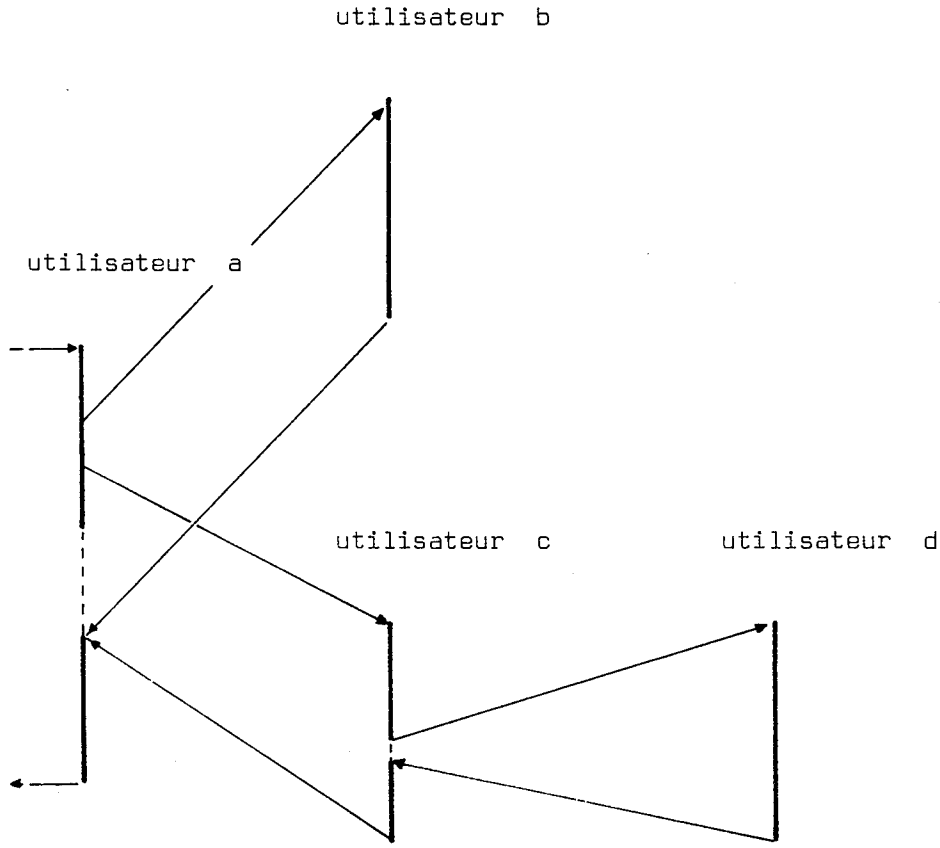


Les opérations de manipulation de ces SDS sont:

- S (APPEL$_i$,p) appelée: "activation du I-processus i avec le paramètre p" et équivalente à une opération v(APPEL$_i$,p) en assimilant la SDS APPEL$_i$ à un sémaphore.
- S (APPEL$_i$,-) appelée: "fin d'une exécution" (du I-processus lui-même) et équivalente à une opération p(APPEL$_i$).
- S (ATTENTE$_i$,-) appelée: "mise en attente" (du I-processus i lui-même) et équivalente à une opération p(ATTENTE$_i$) en assimilant la SDS ATTENTE$_i$ à un sémaphore.

- S(ATTENTE$_i$,q) appelée: "libération du I-processus i avec le rapport q" et équivalente à une opération v(ATTENTE$_i$,q).

Ces mécanismes de synchronisation sont volontairement restreints de manière à, d'une part simplifier leur réalisation et d'autre part à permettre une récupération de l'exécution en cas d'erreur. En particulier, il a été convenu qu'un utilisateur i devait, et ne devait que, libérer l'utilisateur j qui l'a appelé. Ce mécanisme est réalisé, d'une part,en utilisant pour cette libération le nom de l'utilisateur appelant qui a été rangé dans une place laissée libre du paramètre et, d'autre part en plaçant le rapport dans un autre emplacement, prévu à cet effet, de ce même paramètre. De telles restrictions font que les utilisateurs ne peuvent s'appeler que d'une manière arborescente.

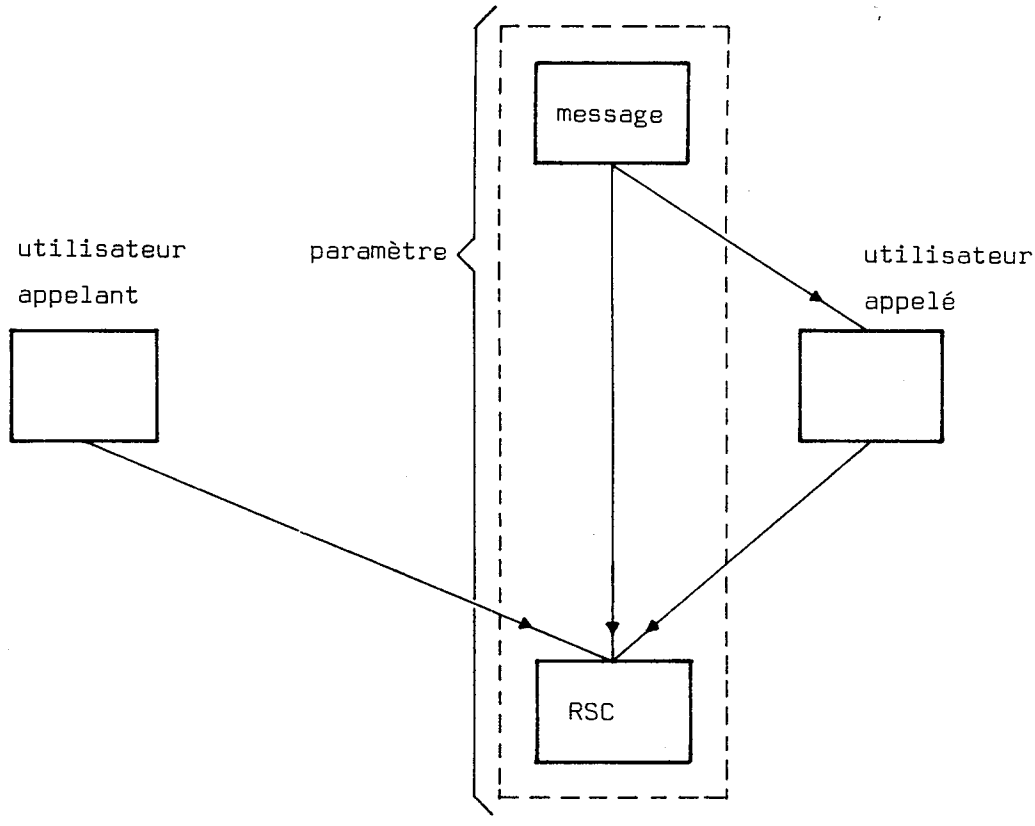


Remarque: Cette contrainte peut être levée pour certains utilisateurs particuliers constituant le système d'exploitation de la machine. Dans ce cas l'hyperviseur micro-programmé lui-même doi réserver la place nécessaire au rapport associé à la libératic d'un utilisateur quelconque.

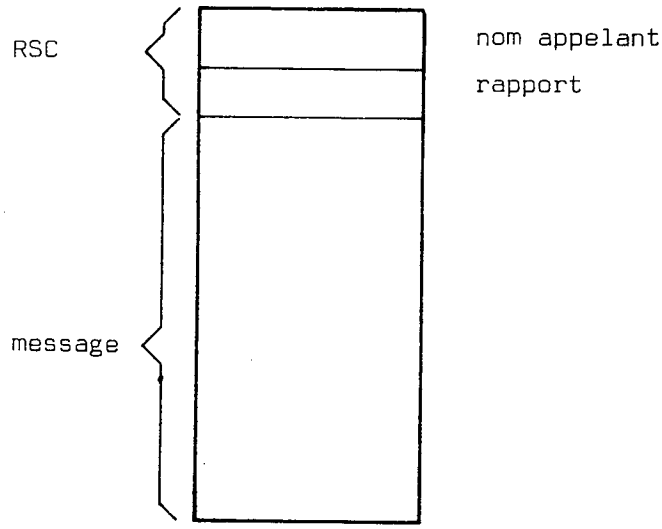
Un tel schéma de synchronisation n'est pas complétement satisfaisant car il interdit pratiquement à un utilisateur d'attendre spécifiquement la fin de l'une des séries de travaux lancés en parallèle. En effet , le premier de ceux-ci à être terminé libère l'utilisateur en attente qui doit, pour connaître l'identité de l'utilisateur appelé qui l'a débloqué analyser par programme le rapport reçu.

Une meilleure solution en cours de réalisation consiste à lier la RSC de réponse au paramètre qui représente le travail à réaliser. Dans ce cas:

- la file FQ ne serait susceptible que de contenir le rapport concernant l'exécution du travail représenté par le paramètre.
- la file FR' ne serait plus implicite et pourrait contenir le nom de l'utilisateur appelant lorsqu'il se met en attente du rapport.



Ces files seraient réalisées à l'aide des deux mots laissés libres dans le paramètre. L'opération d'attente correspondrait au chargement du nom de l'appelant dans le mot du paramètre qui lui est réservé.



structure d'un paramètre .

3/1.1. Actions du milieu extérieur

Le milieu extérieur peut déclencher des opérations S d'activation de I-processus. Le paramètre P est alors émis, par l'organe qui sollicite cette activation, sur les lignes d'entrée-sortie.

Nous avons donc, dans cette machine, remplacé le mécanisme d'interruption en ce qui concerne les appels de processus, par l'accès du milieu extérieur à certaines opérations de synchronisation.

3/1.2. Mécanismes programmés de synchronisation

De manière à pouvoir étendre le système hiérarchisé par des êtres programmés constituant des niveaux supérieurs, nous avons introduit une RSC particulière permettant de réaliser un mécanisme de mutuelle

exclusion entre les I-processus utilisateurs. Pour celà la SDS objet SOLO de cette RSC regroupe:

- une file de jetons FJ susceptible de contenir, au plus, un jeton qui peut être sollicité par un I-processus pour obtenir la mutuelle exclusion. Cette file est initialisée comme contenant le jeton. El est simplement réalisée à l'aide d'un bit particulier du registre d' état de la machine.

- une file FU des candidats à la possession du jeton. Toutefois, comme dans une configuration mono-processeur la machine ne dispose que d'un unique exécutant, cette file peut être supprimée par le blo cage des m différents mécanismes d'allocation (des processeurs sur la machine physique et des I-processus sur le processeur en exécutic

Cette SDS est manipulée par les opérations:

- S(SOLO, -) pour demander l'entrée dans une section critique,
- S(SOLO, δ) pour sortir de la section critique.

Ce mécanisme fournit la mutuelle exclusion indispensable à la construction, par programme, d'opérations de synchronisation plus élaborées que celles fournies par l'hyperviseur. Ces opérations doivent, pour déclencher les inductions nécessaires à leur fonctionnement, utiliser les opérations S accessibles au niveau du code d'ordre.

La SDS SOLO est également utilisée au niveau de l'hyperviseur lui-même pour l'exécution, en mutuelle exclusion, de certaines séquences de modification des données critiques concernant en particulier l'état des files d'utilisateur des SDS de gestion des processeurs virtuels.

Pour résoudre le problème de la levée de l'exclusion mutuelle en cas de mise en attente, il a été convenu que toute opération de synchronisatior exécutée dans une section critique levait automatiquement cette mutuell exclusion. Ceci impose de mettre ces opérations de synchronisation à la fin de la section critique.

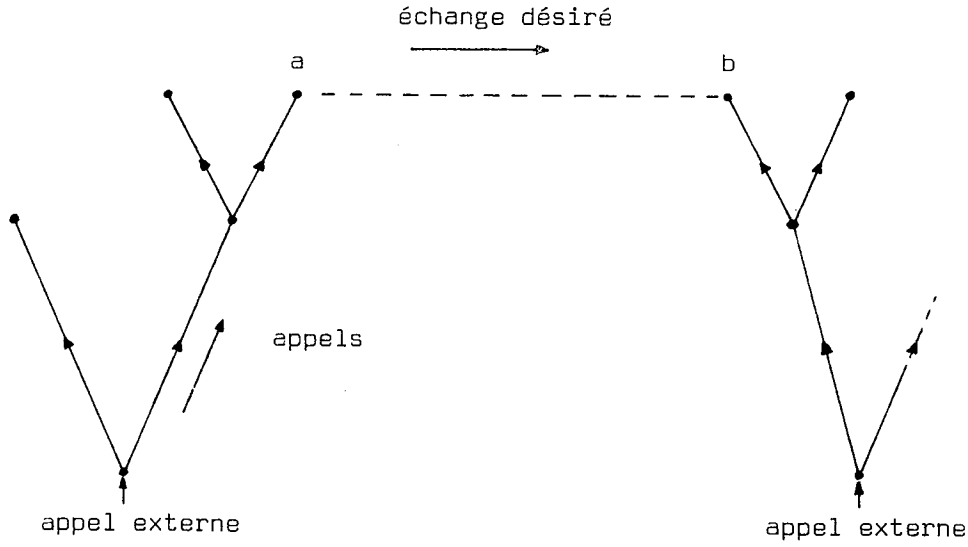
3/1.2.1. Configuration multiprocesseur

Lorsque plusieurs machines sont couplées pour obtenir un multiprocesseur à mémoire commune, une connection électronique permet de coupler les SDS SOLO de chaque machine en une seule SDS et d'étendre ainsi la mutuelle exclusion à tout le système.

3/1.2.2. Synchronisation inter-arborescences

Nous avons vu que les restrictions apportées au mécanisme de synchronisation des I-processus impose une structure arborescente des appels de ces processus. Pratiquement chaque appel externe peut être considéré comme la racine d'une telle arborescence.

Exemple:



synchronisation inter-arborescence .

Les opérations de synchronisation offertes par l'hyperviseur ne permettent pas d'établir directement un mécanisme d'échange d'information entre des articulations appartenant à des arborescences différente (par exemple entre les articulations a et b de l'exemple précédent) en effet:

- il n'est pas possible de considérer que l'information à transmettr est un paramètre au sens de ceux utilisés pour déclencher l'activité des I-processus car il s'agit plutôt ici de libérer le I-processus récepteur lorsque l'information est disponible.

- il n'est pas non plus possible de considérer cette information comme un rapport d'exécution car l'utilisateur émetteur n'a pas été appelé par le récepteur pour exécuter un travail. L'usage des opérations de libération qui ne correspondent pas à l'envoi d'un rappor se heurte à des difficultés:

  . il faut être sûr que le descripteur du I-processus à libérer existe sous peine de détruire les éléments fondamentaux au fonctionnement de l'hyperviseur.

  . l'usage de ces opérations est en général réservé aux seul I-processus du système d'exploitation car leur usage ne per met plus au mécanisme de récupération automatique de l'exécution de fonctionner en cas d'erreur .

Une solution consiste à construire par programme une RSC liée au récepteur de cette communication.

La SDS objet de cette RSC est chargée de gérer l'association du récepteur aux messages transmis. Elle est manipulée par des opérations programmées qui utilisent d'une part la SDS auxiliaire SOLO pour réal ser leur mutuelle exclusion, et, d'autre part, les opérations S fournies par l'hyperviseur vis-à-vis de la SDS ATTENTE du récepteur pour transmettre les inductions directes nécessaires.

Ces SDS sont manipulées par des opérations µS dont l'exécution est provoquée par les inductions directes dues à l'exécution des opérations S de synchronisation des I-processus utilisateurs.

Pour accroître la rapidité du mécanisme, l'état de la file implicite FP de processeurs virtuels d'une telle SDS est représenté par un indicateur binaire dans le registre d'état de la machine physique.

3/2.1. Gestion des unités de traitement

L'unité de traitement virtuelle d'un système résultant du couplage de plusieurs GEOPROCESSEURS est une articulation multiple, dont le mécanisme d'allocation est indépendant du nombre de machines connectées et est obtenu par le simple établissement de connections particulières entre elles, sans aucune adjonction de nouveaux organes technologiques ou de programmes particuliers. Les configurations ne comprenant qu'une machine sont considérées comme des multiprocesseurs dégénérés.

Les simplifications issues du caractère privé des SDS de gestion des processeurs virtuels ne peuvent plus être appliquées au mécanisme de gestion de l'articulation multiple constituant l'unité de traitement du système. Celle-ci sera gérée par une SDSΔu constituée de:

- une seule file FU de I-processus candidats au traitement
- un ensemble AS dont chaque couple (p,u), où p est une unité de traitement virtuelle équivalente et u un I-processus utilisateur, sera représenté par un pointeur sur le descripteur du I-processus u situé à une adresse caractéristique du processeur p .
- une file FP des unités de traitement candidates à l'association avec un I-processus. Cette file sera représentée par un bit dans le registre d'état de chaque machine physique.

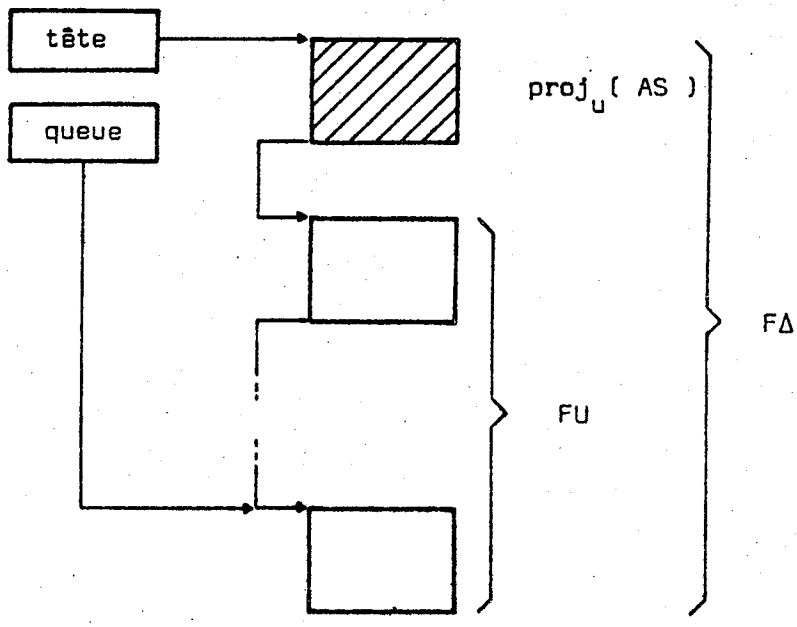
Remarques:

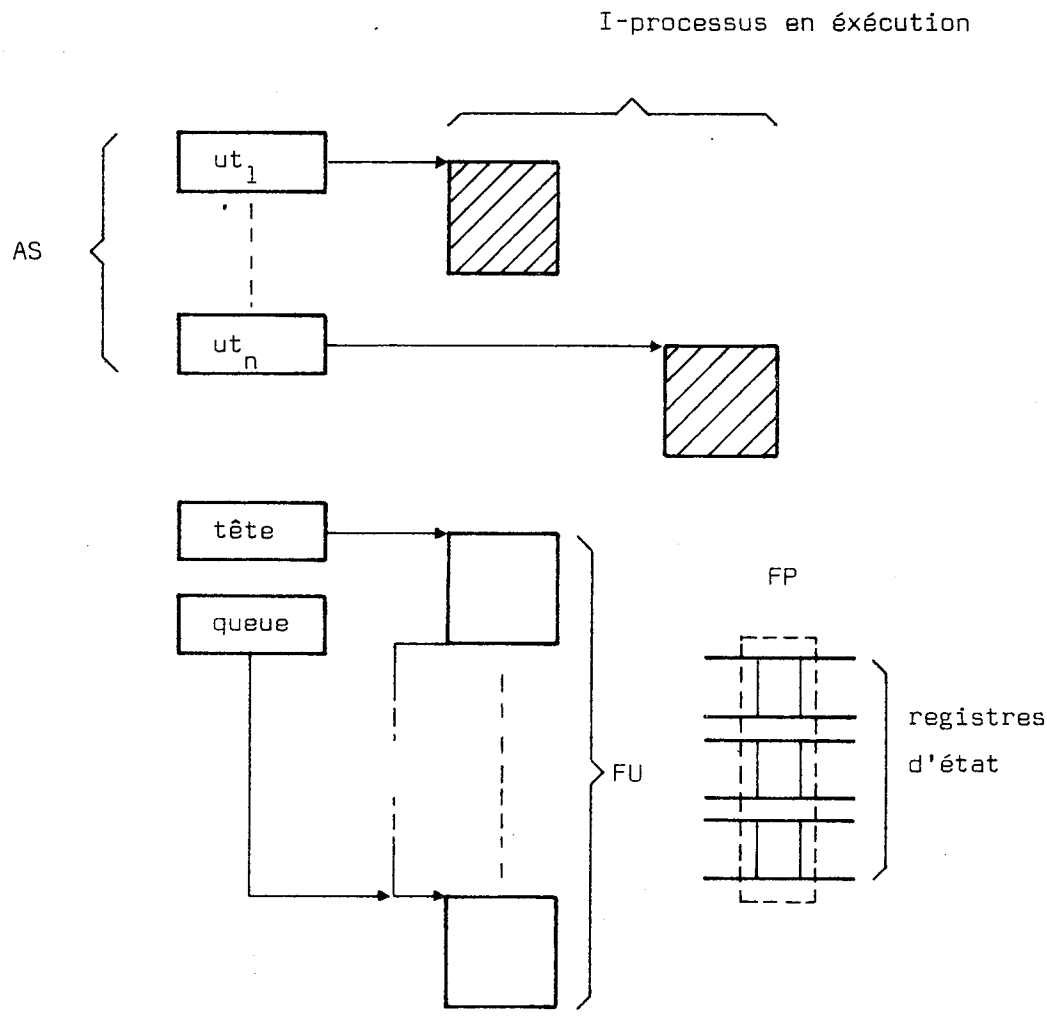
- Un mécanisme d'échange plus élaboré devrait comporter deux voies:
  - l'une pour l'envoi des messages eux-mêmes de a vers b
  - l'autre pour un retour de messages représentant les places libérées de FM de manière à éviter un débordement de cette file.

- Dans le cas d'une réalisation pratique, il serait plus simple de faire transiter tous les messages dans une direction par une seule file qui regrouperait FM U $proj_M(AS)$.

3/2 SYNCHRONISATION DES PROCESSEURS VIRTUELS

Les articulations qui représentent le niveau supérieur de processeurs virtuels sont chacune munies d'une SDSΔ qui lui est liée, regroupant en une file FΔ une queue FU des I-processus utilisateurs candidats à l'association avec ce processeur et la file $proj_U(AS)$.





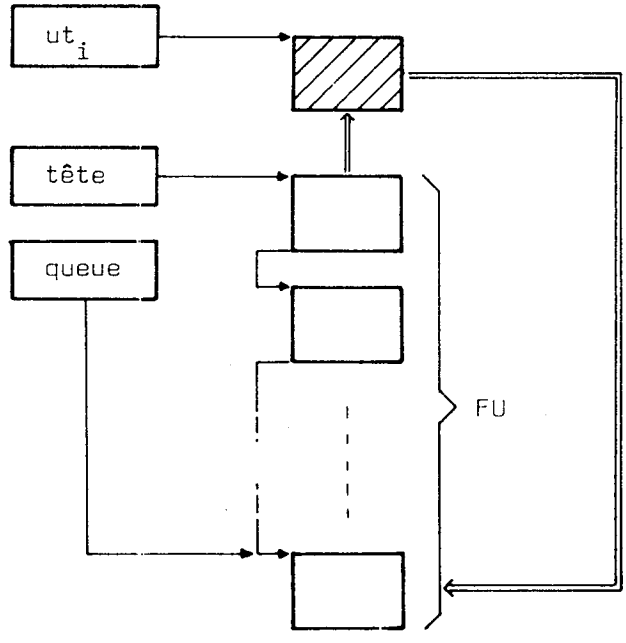
Chaque unité de traitement effectue sa propre allocation. Celle-ci nécessite d'obtenir une mutuelle exclusion d'accès à la file FU par l'intermédiaire de la SDS SOLO étendue à toutes les machines du système. Un tel mécanisme assure la disponibilité d'une unité de traitement qui s'associe avec un I-processus utilisateur puisque cette disponibilité lui est indispensable pour exécuter son mécanisme d'allocation.

Remarque:

Dans le but de permettre la réalisation d'un système d'exploitation en temps partagé, les processeurs virtuels canaux peuvent exécuter une instruction particulière qui a pour effet:

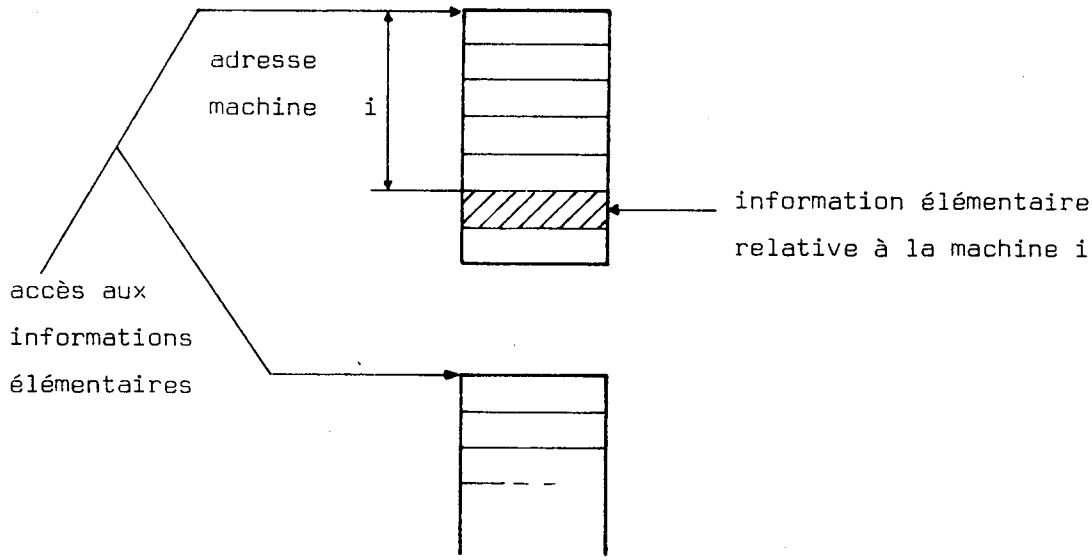
- d'arrêter le I-processus en exécution sur l'unité de traitement de la machine à laquelle appartient ce canal.
- de ranger ce I-processus suspendu en queue de la file FU des I-processus candidats à l'articulation multiple unité de traitement.
- d'associer à l'unité de traitement ainsi libérée le I-processus en tête de la file FU .

Un tel mécanisme a pour but de répartir l'activité de la machine sur les différents I-processus utilisateurs de l'unité de traitement. L'exécution de cette instruction est généralement confiée à un I-processus particulier s'exécutant sur un canal lent et appelé par une horloge extérieure.



3/2.2. Séparation des informations relatives aux hyperviseurs

Chaque hyperviseur microprogrammé propre à une machine physique manipule un certain nombre d'informations en mémoire centrale (pointeurs). Lorsque plusieurs machines sont couplées chaque hyperviseur peut lire sur un organe périphérique particulier l'adresse de la machine physique sur lequel il s'exécute. Cette adresse est utilisée pour indexer tous les accès qu'il fait à ses informations en mémoire centrale. Ceci signifie que chacune des informations élémentaires de chaque machine se trouve regroupée en un tableau ayant autant d'entrées qu'il y a de machines connectées ensemble.



3/3 ALLOCATION DE LA MACHINE PHYSIQUE

En ce qui concerne l'exécution des instructions, la machine physique se présente comme une articulation simple, requérable, pouvant s'allouer à l'un des processeurs virtuels candidats à l'exécution.

Cette articulation est gérée par une SDS Δ_H qui lui est liée et réduite à sa file $F\Delta_H$ des processeurs virtuels candidats ou associés. L'ensemble de ceux-ci étant fixe et connu, cette file pourra être réalisée par une simple suite de bits dans le registre d'état de la machine physique. Chacun de ces bits est affecté, de manière statique, à l'un des processeurs virtuels. Cette SDS sera manipulée par des opérations nS .

Nous pouvons toutefois remarquer que l'information relative à l'état de la file FR_i de la SDS Δ_i de gestion d'un processeur virtuel p_i et l'appartenance de ce processeur de la file $F\Delta_H$ peuvent être regroupées en une seule information binaire. En effet, si nous utilisons les opérations CONCURRENT' et ATTENDU' données en exemple, nous pouvons montrer que:

$$FR_i = \emptyset \leftrightarrow p_i \in F\Delta_H$$

en effet, à l'initialisation du système

$$P_i \in FR_i \quad \text{et} \quad F\Delta_H = \emptyset$$

l'exécution d'une opération $\mu S(\Delta_i, u)$ entraîne:

si $FR_i = \emptyset$ l'opération n'a aucun effet sur les éléments de la relatio

si $FR_i \neq \emptyset$ l'opération PRENDRE (p_i, FR_i) est exécutée et entraîne:

$FR_i = \emptyset$

l'opération $nS(\Delta_H, p_i)$ est induite et entraîne $p_i \in F\Delta_H$

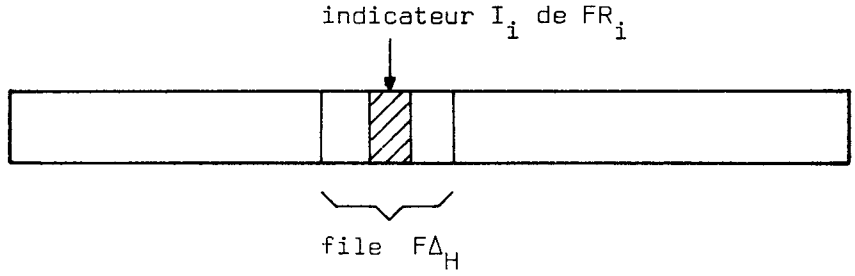
l'exécution d'une opération $pS(\Delta_i, p_i)$ entraîne:

si $FU_i = \emptyset$ l'opération METTRE (p_i, FR_i) est exécutée et entraîne:

$FR_i \neq \emptyset$

l'opération $nS(\Delta_H, h)$ est exécutée et entraîne $p_i \in F\Delta_H$

si $FU_i \neq \emptyset$ l'opération n'a aucun effet sur les éléments de la relatic



registre d'état de la machine j

A ce niveau de l'étude nous admettrons que la file FH de la SDS Δ_H utilisée pour ranger le nom de cette articulation n'a pas besoin d'exister car son mécanisme d'allocation boucle s'il n'a aucun candidat.

3/3.1. Cas d'une configuration multiprocesseur

Lorsque une configuration multiprocesseur est utilisée, il n'est plus toujours possible d'exécuter les inductions $\mu S \rightarrow nS$ par le simple appel du microprogramme correspondant. En effet, l'opération μS peut être exécutée par une machine pour soumettre un I-processus à l'exécution d'un processeur virtuel spécifique à une autre machine (par exemple à l'un de ses canaux) dont le registre d'état ne lui est pas accessible. Pour pallier à cette difficulté un mot mémoire est affecté à chacune des machines dans un tableau réservé en mémoire centrale et le mécanisme suivant est utilisé:

- la première de ces machines inscrit dans le mot réservé à la seconde la modification souhaitée de son registre d'état, en l'ajoutant

par union à celles déjà inscrites.

- un indicateur technologique de la seconde machine est positionné par les voies de sortie de la première.

- la seconde machine, prévenue par l'indicateur, vient, dès qu'elle le peut, relever les requêtes de modification qui lui sont destinées, remettre à zéro le mot qui les contenait et mettre à jour son registre d'état.

Tous les accès à ce tableau devront être faits après l'établissement d'une mutuelle exclusion à l'aide de la SDS SOLO.

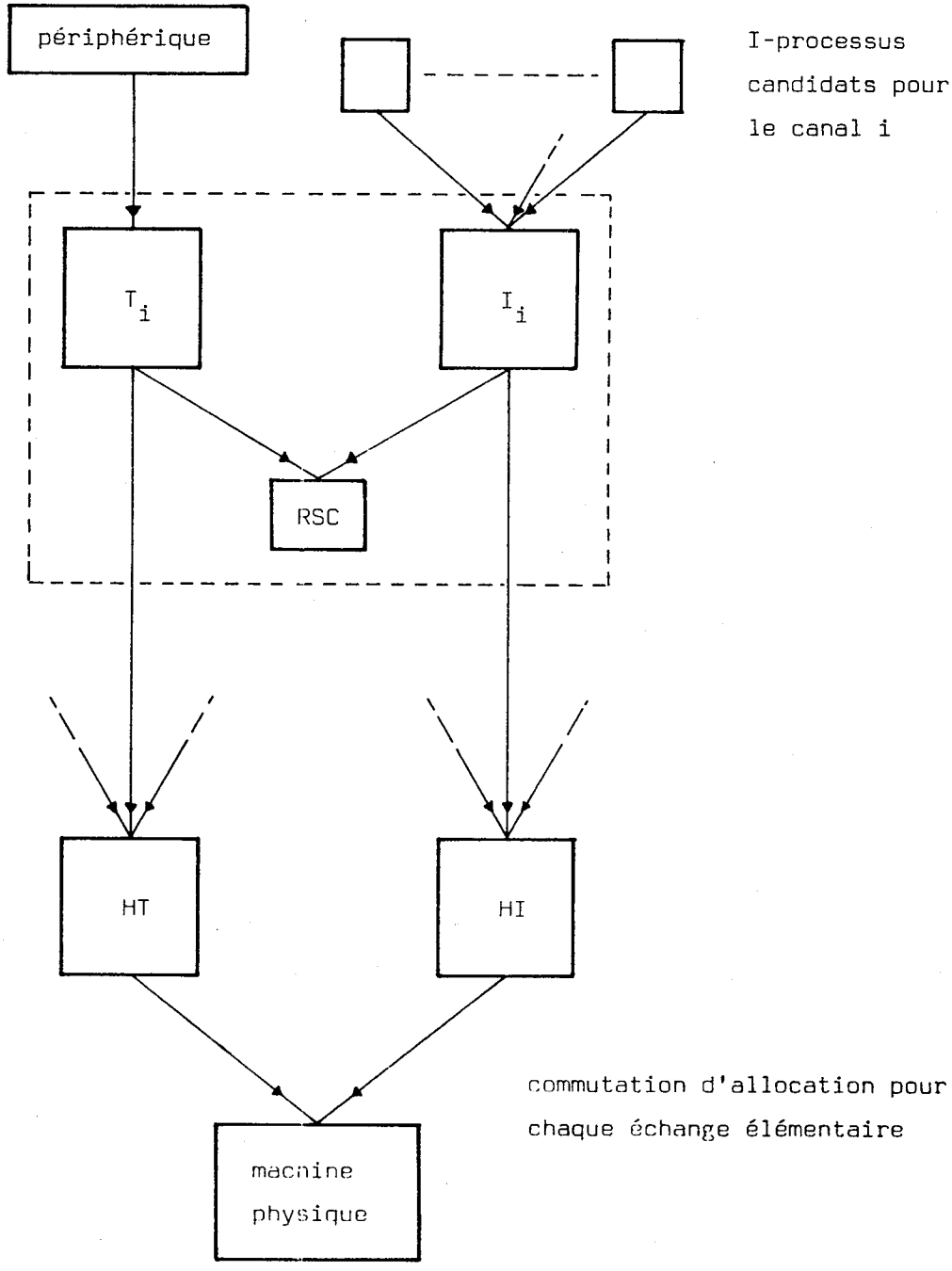
La rapidité de réaction de la seconde machine n'est pas critique. En effet le but de cette requête n'est que de permettre la fourniture de la ressource constituée.

4 - CANAUX

Chaque processeur virtuel canal est constitué de deux articulations:

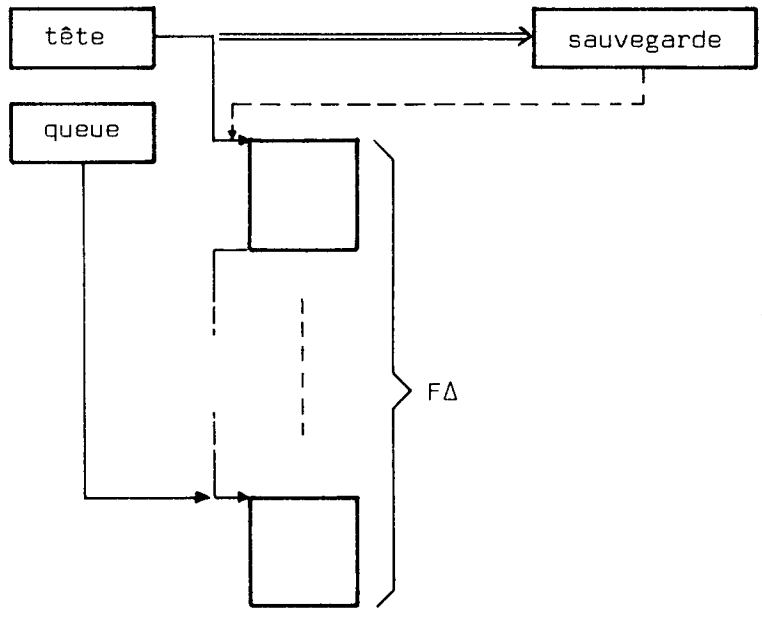
- une articulation I_i chargée de l'exécution des instructions des I-processus candidats à ce canal.
- une articulation T_i chargée de l'exécution des échanges d'information avec les périphériques connectés. Cette articulation est reliée à la précédente par une RSC.

Toutes les articulations I_i utilisent une ressource unique représentée par une articulation HI , qui a pour rôle l'interprétation centralisée des instructions pour les différents processeurs virtuels. Cette articulation est réalisée par un microprogramme unique d'interprétation permis par la grande similitude des ensembles d'instructions reconnus par les différents processeurs virtuels. De même, les articulations d'échange T_i utilisent une ressource unique représentée par l'articulation HT qui a pour rôle la réalisation d'un échange élémentaire. La raison pour laquelle nous avons adopté ce découpage tient au fait que dans cette organisation un I-processus exécuté par un processeur virtuel canal se met en attente sur la RSC qui relie l'articulation I_i qui l'interprète, à celle, T_i qui est activée par le décodage d'une instruction d'échange. Par un mécanisme d'induction directe, l'articulation I_i ne se trouve plus candidate au traitement par l'articulation HI pendant tout l'échange d'un bloc d'information entre un périphérique et la machine. Lors de chaque transfert élémentaire l'articulation T_i est activée et induit une réquisition par HT de la machine physique. La commutation qui en découle ne concerne pas les I-processus en exécution, se trouve simplifiée par le fait que les contextes des articulations HI et HT sont simultanément présents dans les mémoires internes (locale et auxiliaire) de la machine physique.


périphérique
I-processus candidats pour le canal i
T_i
I_i
RSC
HT
HI
commutation d'allocation pour chaque échange élémentaire
machine physique

Les mises en attente et les libérations de l'articulation I_i sur la RSC qui le relie à l'articulation T_i sont réalisés simplement en trompant les mécanismes normaux d'allocation de l'articulation HI aux processeurs virtuels. En effet, un processeur virtuel n'est supposé candidat à l'articulation HT que si la file FΔ de sa SDS de gestion Δ n'est pas vide. Nous utilisons ce mécanisme lors de la mise en attente de I_i en simulant une file FΔ vide. pour celà la valeur du pointeur de tête de cette file est sauvegardée à un emplacement réservé à cet usage et il est chargé avec l'indication d'une file vide.

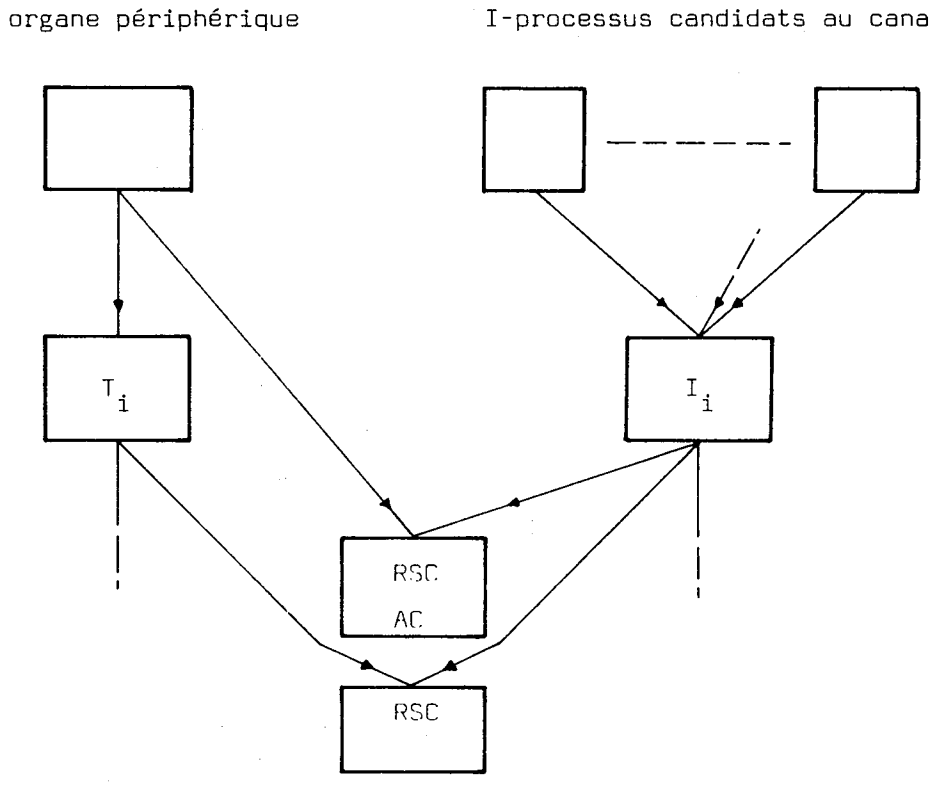
Cette mise en attente induit l'exécution d'une opération nS qui extrait ce processeur de la file $F\Delta_H$ de la SDS de gestion Δ_H de l'articulation HI en positionnant convenablement l'indicateur binaire du registre d'état associé à ce canal. La fin du transfert d'un bloc provoque la libération de l'articulation I_i qui entraîne la restitution de la valeur convenable du pointeur sur la tête de FΔ .



Remarque: Cette substitution ne perturbe pas la soumission de nouveaux I-processus à ce canal par des opérations S(Δ,u) car la file FΔ est une queue et elles n'utilisent que le pointeur, inchangé, sur son autre extrémité.

4/1 - SYNCHRONISATION COMPLEMENTAIRE AVEC LE MILIEU EXTERIEUR

Il est apparu nécessaire de permettre aux I-processus en exécution sur les canaux , d'attendre l'occurrence d'un évènement extérieur (par exemple le déplacement du bras mobile d'un disque). Plutôt que de permettre au milieu extérieur l'usage des opérations de libération sur les SDS ATTENTE des I-processus, il est apparu plus simple, et moins dangereux, d'utiliser des RSC particulières AC_i permettant une synchronisation entre les organes périphériques et les articulations I_i des canaux.

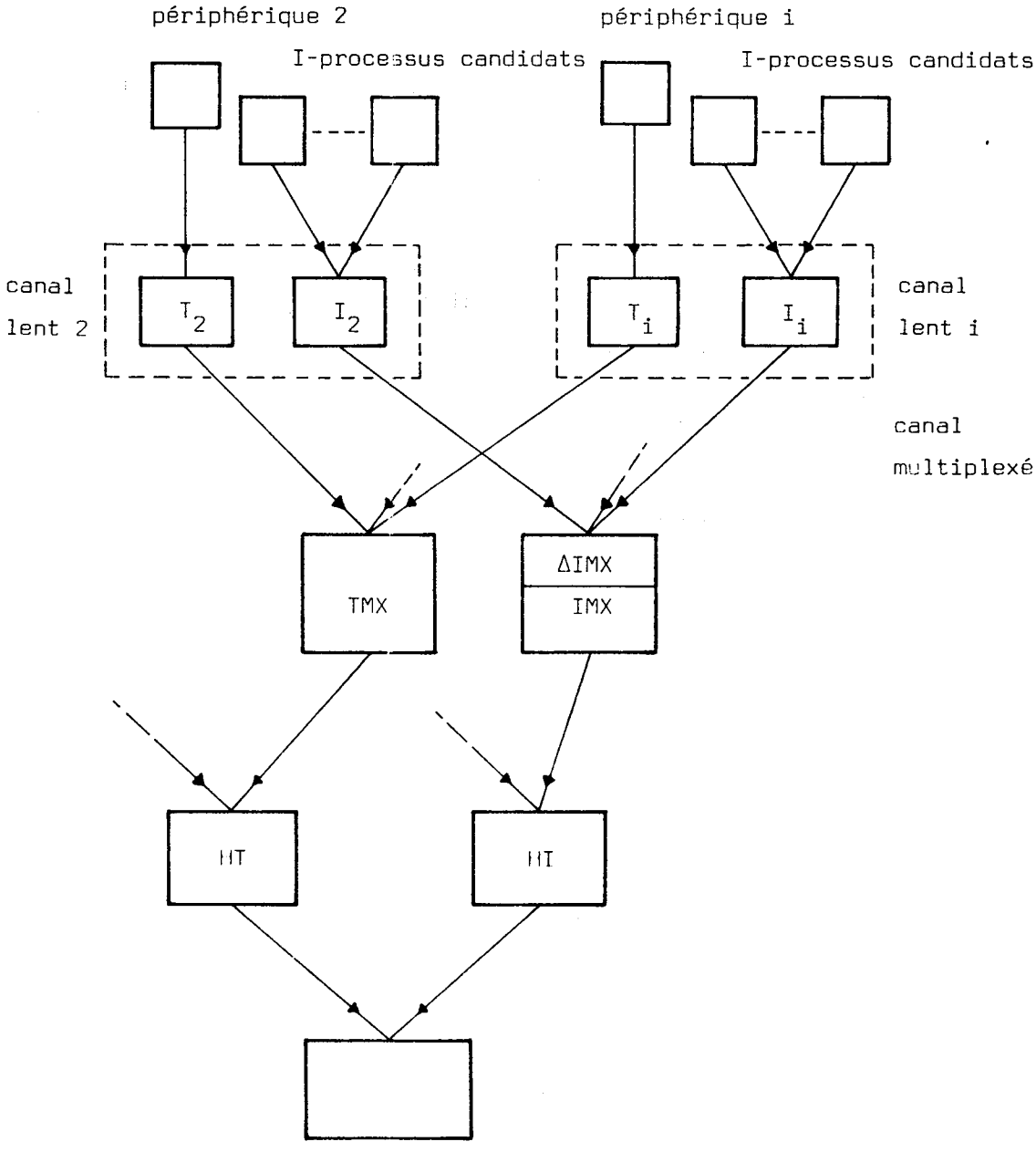


L'articulation I_i d'un canal peut exécuter une opération S(SDS(AC),-) pour se mettre en attente jusqu'à ce que le périphérique exécute S(SDS(AC),δ) pour la libérer.

4/2 - CANAL MULTIPLE

Les articulations I_i et T_i des canaux lents n'utilisent pas directement les ressources représentées par les articulations HI et HT mais des articulations IMX et TMX qui représentent les ressources d'interprétation des instructions et de transfert pour le canal multiplexé. Cette organisation permet de ne pas compliquer le mécanisme d'allocation de HI par les 64 canaux lents qui n'y sont représentés que par l'articulation IMX. Celle-ci dispose par contre d'un mécanisme d'allocation adapté aux canaux lents et constitué d'une SDS Δ_{IMX} privée dont la file $F\Delta_{IMX}$ est représentée par une chaîne de bits dans la mémoire auxiliaire. Chacun de ces bits représente un canal lent. Une opération PRENDRE $(c, F\Delta_{IMX})$ microprogrammée permet de rechercher, dans cette file le canal lent le plus prioritaire candidat à l'exécution.

Chaque canal lent ne peut être lié qu'à un seul organe périphérique pour éliminer un niveau de mécanismes d'allocation.


périphérique 2
I-processus candidats
périphérique i
I-processus candidats
canal
lent 2
T_2
I_2
T_i
I_i
canal
lent i
canal
multiplexé
TMX
ΔIMX
IMX
HT
HT

4/2.1. Conflit d'accés à la mémoire auxiliaire

La mémoire auxiliaire est une mémoire interne à la machine, relativement rapide, à laquelle sont affectées les fonctions suivantes:

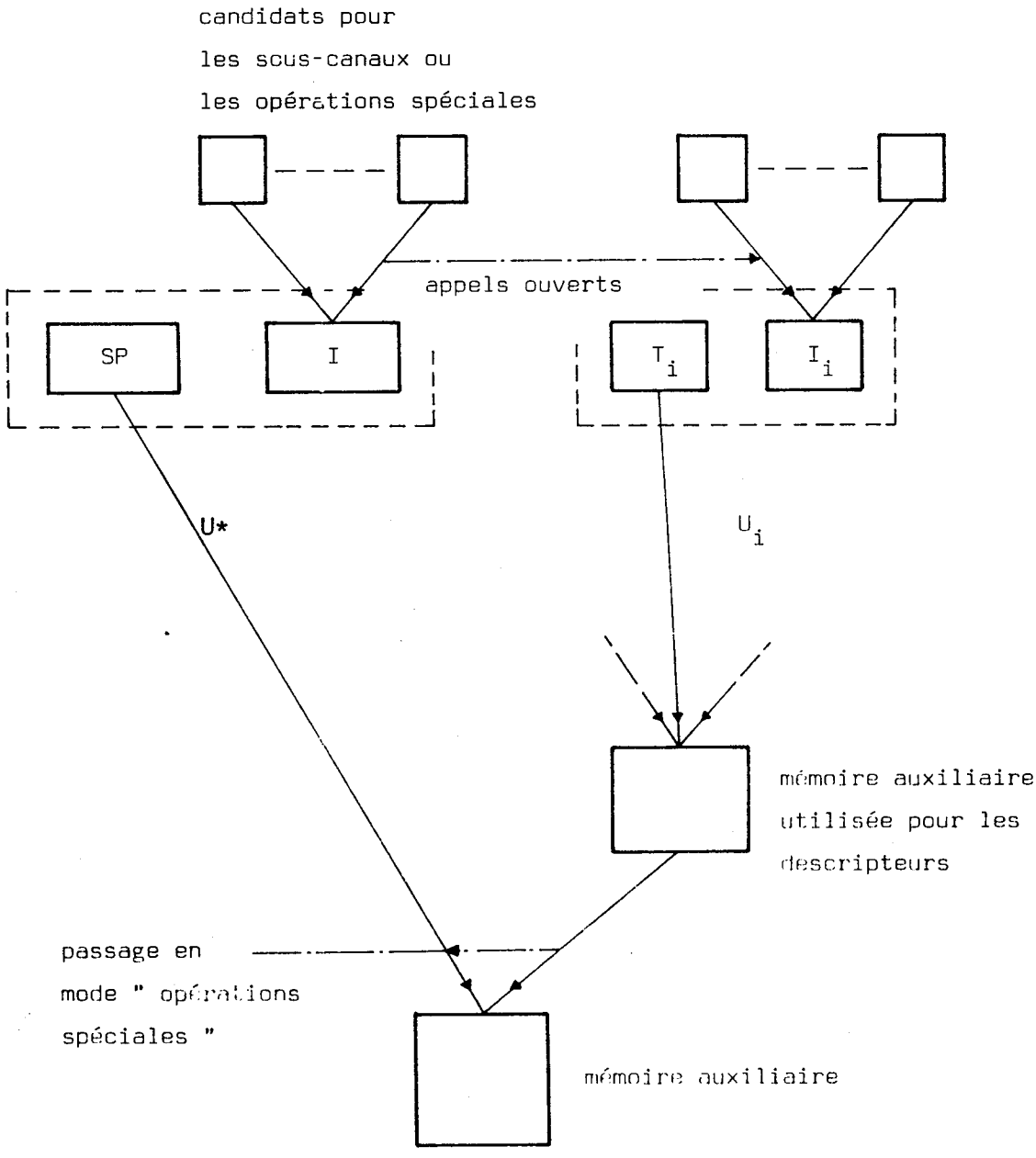
- contenir les informations relatives à l'exécution des opérations spéciales,
  - . opérateur d'une convolution,
  - . table trigonométrique d'une transformée de Fourier rapide.
- contenir les descripteurs des canaux lents multiplexés.

Ces deux fonctions sont évidemment mutuellement exclusives. Le mécanisme utilisé pour réaliser cette exclusion est le suivant:
Un canal lent particulier U*(n° = 64) du canal multiplexé possède les propriétés suivantes:

- son descripteur est en mémoire centrale,
- il ne peut pas exécuter de transfert de données avec l'extérieur,
- il peut exécuter les opérations spéciales (convolution, FFT,...),
- tous les I-processus candidats à l'exécution sur un quelconque canal lent, doivent primitivement être candidat à l'exécution sur U* qui les aiguillera ensuite, un à un, vers le canal lent réclamé. Cette opération suppose une transformation dynamique de la structure du système hiérarchisé puisqu'il y a changement dynamique du processeur allouable à un I-processus (appel couvert [1]).

Un I-processus désirant exécuter une opération spéciale doit se porter candidat à l'exécution par le canal lent U*. Ce dernier devra, avant d'entreprendre l'exécution de l'opération spéciale, s'assurer de la disponibilité de la mémoire auxiliaire, c'est-à-dire solliciter son allocation de manière exclusive. Celle-ci n'est possible que lorsque la totalité des canaux lents ont terminé leur travail courant.

Les I-processus transitant par U* pour rejoindre leur canal lent arrivés après celui qui désire exécuter une opération spéciale restent bloqués derrière lui dans la file $F\Delta_{U*}$ de la $SDS\Delta_{U*}$ de cette articulation et ne peuvent pas venir redémarrer les autres canaux lents, ce qui maintient la mutuelle exclusion entre U* et les autres canaux lents.


candidats pour
les sous-canaux ou
les opérations spéciales
appels ouverts
SP
I
T_i
I_i
U*
U_i
mémoire auxiliaire
utilisée pour les
descripteurs
passage en
mode " opérations
spéciales "
mémoire auxiliaire

5 - MECANISME DE DEROUTEMENT

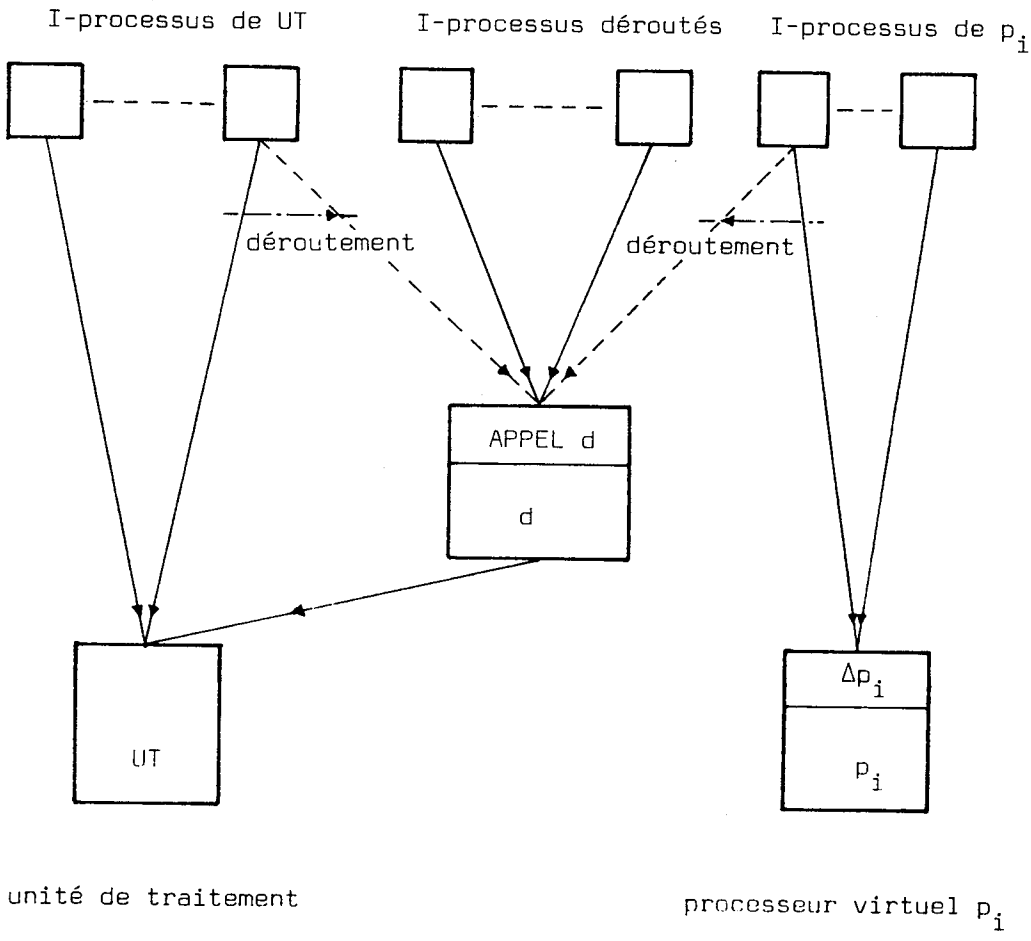
Le mécanisme de déroutement du GEOPROCESSEUR est réalisé en particularisant un I-processus utilisateur d pour jouer le rôle d'un processeur programmé de déroutement. Le I-processus d s'exécute sur l'unité de traitement.

Lorsqu'un I-processus utilisateur u commet une faute il est rendu candidat avec l'indication de la faute qu'il a commise au processeur de déroutement par une opération $S(APPEL_d, \langle u, faute \rangle)$.

Une induction de libération doit alors être émise à l'attention du processeur virtuel pi qui exécutait u sous la forme d'une opération $\mu S(\Delta_{pi}, pi)$ qui extrait u de la file $F\Delta_{pi}$ de la SDS Δ_{pi} de gestion de ce processeur virtuel.

A chaque I-processus utilisateur peut éventuellement être associée une séquence particulière de code exécutable dans certains cas d'erreur, pour permettre une récupération de l'exécution après que le processeur de déroutement ait analysé l'incident et déterminé si la faute n'est pas fatale.

Ce mécanisme de déroutement est compatible avec le fonctionnement de l'hyperviseur microprogrammé. Il a pour effet de réduire la priorité pratique des processus fautifs. Cette particularité n'est pas une restriction dans la limite où l'on ne désire pas utiliser ce mécanisme pour étendre l'ensemble des instructions reconnues par la machine.


I-processus de UT
I-processus déroutés
I-processus de p_i
déroutement
déroutement
APPEL d
d
UT
Δp_i
p_i
unité de traitement
processeur virtuel p_i

6 - DESTRUCTION DES I-PROCESSUS FAUTIFS

Il est normal, dans un système d'exploitation de chercher à éliminer les I-processus fautifs. Dans le cas de cette machine, il est nécessaire de prendre certaines précautions pour les raisons suivantes:

- le I-processus fautif peut avoir appelé d'autres I-processus. Dans ce cas leurs réponses doivent avoir lieu avant que son descripteur ne soit détruit.

- le I-processus fautif peut avoir été appelé par un I-processus qui attend, ou attendra, une réponse sous la forme d'un rapport sur l'exécution du travail qu'il a soumis.

Ces raisons amènent souvent à différer la destruction d'un I-processus fauti Celui-ci étant d'abord mis dans un état qui ne lui permet pas de poursuivre ses exécutions (il peut être considéré comme candidat à un processeur virtuel de destruction).

La première difficulté est résolue en associant à chaque I-processus utilisateur un compteur, appelé compteur d'appels, qui est incrémenté de 1 chaque fois que ce I-processus en appelle un autre et, décrémenté de 1 à chacune de leurs réponses. Un I-processus fautif ne pourra être éliminé que si son compteur d'appel vaut 0 , c'est-à-dire que s'il a reçu toutes les réponses aux appels qu'il a émis. La seconde difficulté est aisément résolue en émettant, à la place du I-processus fautif , une réponse vers celui qui l'a appelé sous la forme d'une opération $S(ATTENTE_{appelant}, f)$ dans laquelle f est un rapport universellement connu comme émis par l'hyperviseur en cas d'erreur d'I-processus appelé.

L'utilisation des opérations de libération explicites c'est-à-dire qui ne sont pas des réponses à des opérations d'appel, est incompatible avec ces mécanismes de récupération du fonctionnement de la machine. En effet, il n'est pas possible de prévoir, avant la mise en attente, qu'une libération explicite sera émise pour un I-processus ou que celui-ci devra ultérieurement en libérer explicitement un autre. Compte tenu de ces difficultés, l'usage de ces opérations de libération explicite est limité au cas où elles

sont indispensables et, leur exécution est confiée à certains I-processus constituant le système d'exploitation et supposés à l'abri des erreurs.

C O N C L U S I O N

Le formalisme présenté dans les pages précédentes est orienté vers la description de certains aspects de l'organisation interne des ordinateurs et de leurs systèmes d'exploitation.

Nous avons du, dans un premier temps, définir ce que nous entendions par les termes de processus et de processeur et travailler ainsi à préciser la nature et les propriétés de ce que nous pourrions appeler les êtres fondamentaux de l'informatique.

De manière à rester pragmatique et à convaincre le lecteur de l'utilité de ce travail, une importante partie est consacrée à l'application de ce formalisme à la description des principaux mécanismes rencontrés lors de l'étude des machines informatiques. La présentation d'une architecture originale pour une machine prototype actuellement réalisée, est destinée à montrer les développements qu'il est possible d'en attendre.

Il ressort nettement des exemples présentés que ce formalisme tend à unifier dans quelques modèles simples, la majorité des mécanismes d'allocation de ressources, de synchronisation et d'interprétation, utilisés dans les machines informatiques.

D'un point de vue plus général, il peut être considéré comme une contribution à l'unification de deux mondes de l'informatique concernés, l'un par la structure technologique des machines utilisées, l'autre par celle des programmes qu'elles exécutent. Deux domaines qui tour à tour s'ignorent puis se redécouvrent. Loin d'être épuisé, ce sujet semble très vaste et nous n'avons pas la prétention d'en avoir achevé l'exploration. La meilleure contribution que nous pourrions apporter à l'ensemble des travaux dans ce domaine, serait de susciter des discussions et d'ouvrir de nouvelles voies de recherche.

Ce travail est incontestablement inductif, c'est-à-dire qu'il est plus orienté vers la suggestion de nouvelles tournures d'esprit que vers la présentation rigoureuse d'une structure formelle aux régles précises.

Nous pensons qu'une telle attitude est indispensable lorsqu'il s'agit de jeter les grandes lignes de nouveaux concepts, mais qu'il y aurait toutefois un risque évident de "virtualiser" jusqu'au raisonnement lui-même si il n'était pas, par la suite, étayé par des travaux plus déductifs.

Nous pouvons, dans une premier temps, rappeler les grandes lignes du point de vue que nous adoptons, relativement à la notion de processus.

Un processus est, par définition, un être très simple déjà bien connu sous le nom de machine séquentielle. Nous pensons que sa définition ne doit pas tenir compte de sa réalisation. Elle ne doit donc pas contenir les notions de programme, processeur, allocation de processeur et de ressources.

Il est possible d'objecter que la notion usuelle de processus s'apparente plus à ce que nous appelons I-processus. Même dans ce cas, la distinction de deux états correspondant respectivement à la candidature d'un I-processus vis-à-vis d'un processeur et à son association avec celui-ci, n'a aucun sens pour ce I-processus (et encore moins pour le processus qu'il représente En effet l'écoulement de son temps ne peut raisonnablement qu'être compté relativement à sa progression, et son association avec son processeur ne doit concerner que ce dernier. Le fait qu'un processeur partage ou non ses services entre plusieurs I-processus ne concerne que sa réalisation et les modalités de ce partage sont décidées par son mécanisme d'allocation chargé d'organiser son travail.

L'origine de ce qui nous semble être des confusions est due au fait que souvent la réalisation de ces êtres ne suit pas leur décomposition fonctionnelle. En effet, il est fréquent de rencontrer des êtres dont certains

mécanismes ou parties de mécanismes, sont câblés tandis que le reste est programmé, de manière souvent non distincte des mécanismes appartenant à d'autres êtres.

Pour pousser plus avant notre critique des points de vue habituels, nous pouvons remarquer que la finalité profonde de l'existence des différents mécanismes de synchronisation concerne la réalisation des différents mécanismes d'allocation de ressources. Certaines de ces ressources étant utilisées pour la communication de messages entre les êtres considérés.

Si nous admettons ce point de vue, les états de fonctionnement MARCHE et ARRET habituellement reconnus à une articulation doivent également être banis. En effet, le seul rôle conscient d'une articulation ne consiste plus qu'en la sollicitation et la libération de ressources. La disponibilité de celles-ci ne la concerne pas. Les mécanismes d'induction déclenchés pour libérer une ressource requérable, déjà associée à une articulatior qui ne peut compléter l'ensemble des ressources qu'elle a sollicitées, ne concernent finalement que les mécanismes d'optimisation de l'allocation des ressources et non la logique du comportement de l'articulation elle-même.

Il peut paraître paradoxal qu'après cette prise de position nous maintenions ces états dans la définition de l'articulation. Celà est du au fait qu' elles peuvent être munies d'un mécanisme d'interprétation autonome, ressource interne, à qui il faut signifier un arrêt par une sorte de mécanisme d' induction.

Nous pouvons, dans un second temps, faire une critique de ce travail de manière à orienter de futurs chercheurs. Ses points faibles, qu'il partage d'ailleurs avec d'autres moyens de description, se situent d'une part au niveau de la définition même des objets manipulés, et d'autre part aux niveaux de quelques points de détail du formalisme que l'expérience de son emploi nous suggère d'améliorer.

La description d'un mécanisme à l'aide de cet outil suppose d'une part la connaissance des êtres mis en jeu et d'autre part, l'établissement de leurs relations hiérarchiques.

La définition même des êtres mis en jeu peut être sujette à caution. En effet, lors de l'étude de réalisations pré-existantes, elle est souvent subjective et résulte de la seule volonté de celui qui opère l'analyse, qu'il s'agisse de programmes, qui ne sont finalement que des ensembles de symboles codés, ou de machines qui ne sont elles-mêmes que des assemblages de composants technologiques.

De même, la relation utilisateur-ressource découle toujours de la constatation d'une certaine dissymétrie dans le dialogue, souvent bi-directionnel, entre deux êtres. La "valeur" de la hiérarchisation introduite dépend de celle de la dissymétrie invoquée. Elle peut être évidente, discutable, voire artificielle, si elle est issue uniquement d'un point de vue dissymétrique.

Le formalisme présenté doit être vu comme un outil de description dont le but est atteint s'il permet à celui qui l'utilise de définir des êtres et de les structurer en accord avec ses idées. Il n'est pas universel, c'est-à-dire qu'il ne peut pas tout décrire. Son domaine préférentiel semble être de donner un point de vue macrocospique des mécanismes d'allocation et d'empilement des ressources réelles ou virtuelles. Comme beaucoup d'autres outils de description, il cesse d'être efficient et ses limites sont atteintes lorsque les mécanismes décrits sont trop bien précisés pour ne plus être directement représentables par ceux normalement inclus dans le formalisme. Comme tout outil de description, il ne sert qu'à bâtir un modèle qui ne doit pas être confondu avec ce qu'il prétend représenter.

Parmis les améliorations que nous pourrions apporter, à postériori, au formalisme, nous ne retiendrons que celle concernant l'adjonction d'un troisième mécanisme à la définition d'une articulation. Ce nouveau mécanisme

aurait pour rôle la centralisation et la gestion des requêtes émises par les différents mécanismes de l'articulation vers ses ressources. Dans le formalisme actuel, ces requêtes sont directement émises par les mécanismes d'interprétation et d'allocation. La mise en évidence d'un nouveau mécanisn permettrait de mieux appréhender les problèmes de gestion de ressources rencontrés dans les articulations multiples, les structures pipe-line, la gestion des processus concurrents de lecture et d'écriture etc...

Au terme de ce travail nous constatons que la tournure d'esprit qui en constitue l'essentiel, nous semble maintenant indispensable à la compréhension des mécanismes utilisés dans les machines informatiques et qu'il comble une lacune importante des formalismes précédemment utilisés.

Nous constatons que son emploi, dans la définition de l'architecture des machines informatiques, s'avère particulièrement puissant.

Il permet entre autre une conception descendante de ces machines en ramenant leur définition architecturale à la construction d'un système hiérarchisé particulier à partir de la donnée des fonctions devant être réalisées au niveau des feuilles, et en obéissant à certaines contraintes technologiques de coût et de performances.

Chaque étape de la démarche consiste à définir un niveau supplémentaire du système hiérarchisé, en progressant vers la racine. Cette représentation précise de la machine à réaliser nous permet d'en simuler le comportement à chacune de ces étapes pour en déduire le taux de charge des divers constituants de manière à guider la définition des prochains niveaux du système hiérarchisé de manière à l'adapter au mieux aux performances désirées. Cette adaptation est réalisée soit par la démultiplication de certaines ressources très sollicitées en des articulations multiples, soit par le multiplexage de ressources faiblement utilisées sur des ressources qui leur sont communes.

ANNEXES

1. Problème de la synchronisation de processus concurrents de lecture et d'écriture.
2. Hiérarchies de forêts connectées.
3. Synchronisations élémentaires
4. Mécanismes globaux d'allocation de ressources.

1 - PROBLEME DE LA SYNCHRONISATION DE PROCESSUS CONCURRENTS DE LECTURE ET D'ECRITURE DANS UNE RESSOURCE COMMUNE

1/1. SOLUTION ADMETTANT UNE PRIORITE DES PROCESSUS DE LECTURE SUR CEUX D'ECRITURE DANS LA MESURE OU, AU MOINS, UN PROCESSUS DE LECTURE A ACCES A LA RESSOURCE.

Solution publiée

Processus de lecture

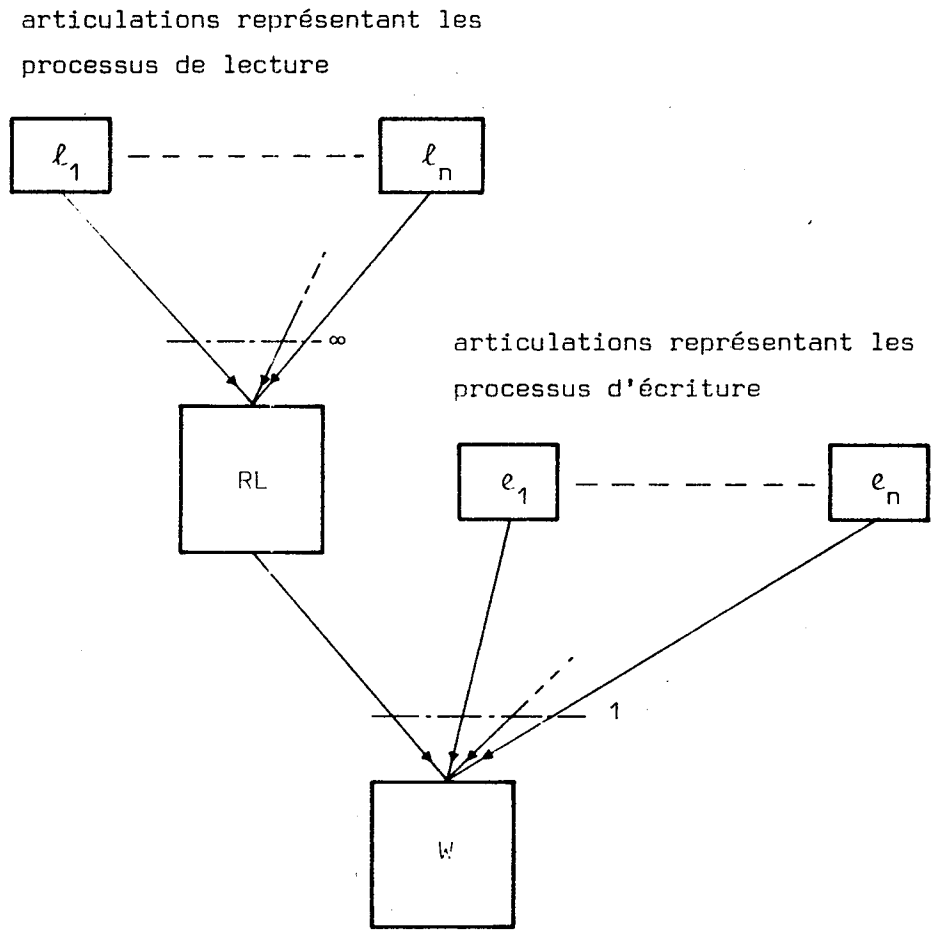
```
P(mutex)
        NL ← NL + 1
        si  NL  =  1  alors  P(w)        } A
V(mutex)
          |
       lecture
          |
P(mutex
        NL ← NL  - 1
        si  NL  =  0  alors  V(w)        } B
V(mutex)
```

Processus d'écriture

```
P(w)
  |
écriture
  |
V(w)
```

Le système hiérarchisé correspondant à cette solution est tel que les processus de lecture sont représentés par des articulations ℓ_i candidates à une ressource commune représentée par l'articulation multiple et dégénérée elle-même candidate, avec les articulations représentant les processus d'écriture e_i , à une ressource commune W qui assure une mutuelle exclusion entre les articulations e_i et vis-à-vis de l'ensemble des articulations ℓ_i via RL .

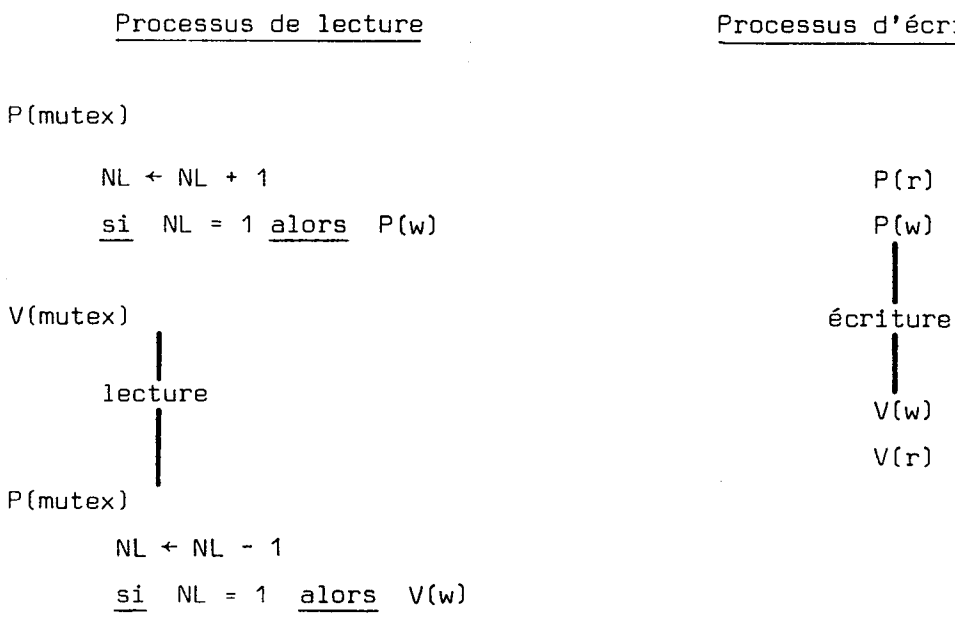
Dans la solution publiée, les groupements d'opérations A et B correspondent exactement à des opérations S(SDS(RL), ℓ_i) et S(SDS(RL), RL) compte tenu du caractère multiple et dégénéré de l'articulation RL .



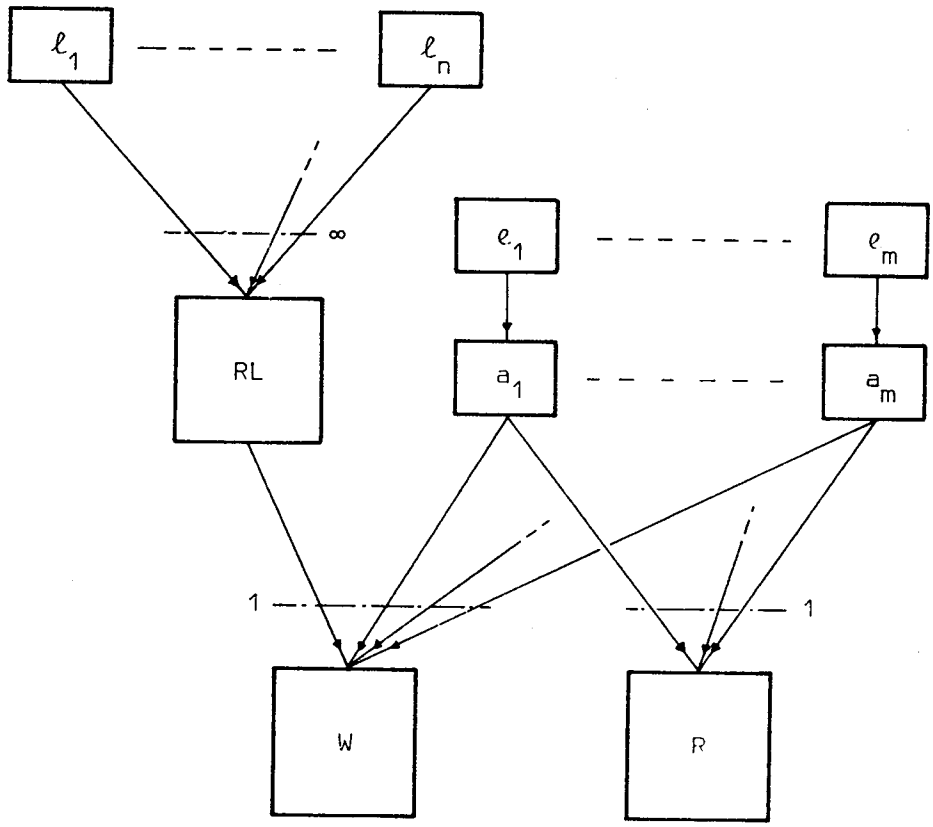
articulation simple
non requérable

1/2. SOLUTION ADMETTANT UNE PRIORITE DES PROCESSUS DE LECTURE SUR CEUX D'ECRITURE DANS TOUS LES CAS

Solution publiée



Il suffit d'adjoindre, à la solution précédente, une ressource commune R aux articulations e_i représentant les processus décriture pour réaliser la mutuelle exclusion supplémentaire entre ces processus, de manière à ce qu'il n'y ait, au plus, qu'une articulation e_i candidate à W .



Les articulations a_i sollicitent l'usage de l'articulation R avant celui de W . De même elles libèrent W avant R . Ces actions peuvent apparaître dans la suite des opérations S induites par les opérations

DEMARRER et ARRETER appelées indirectement par les opérations S utilisées pour activer et arrêter les processus d'écriture.

opération DEMARRER (a_i)

S(SDS(R), a_i) "a_i sollicite l'association avec R"
S(SDS(W), a_i) "a_i sollicite l'association avec W" inductions

opération ARRETER (a_i)

S(SDS(W), W) "l'articulation a_i se dissocie de W"
S(SDS(R), R) "l'articulation a_i se dissocie de R" inductio

1/3. SOLUTION ADMETTANT UNE PRIORITE DES PROCESSUS D'ECRITURE SUR CEUX DE LECTURE

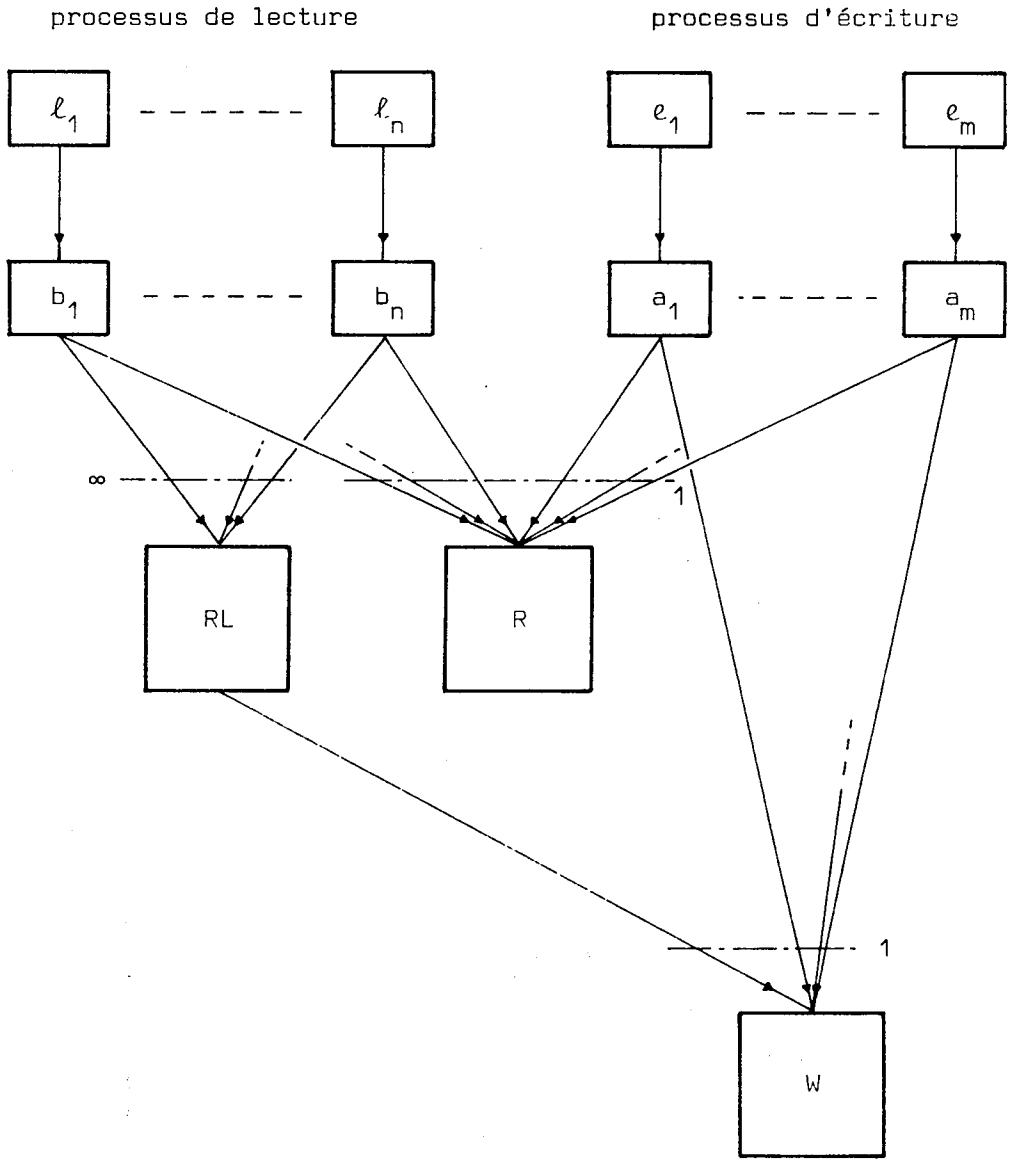
Solution publiée

Processus de lecture

```
P(s)
   P(r)
      P(mutex1)
         NL ← NL + 1
         si NL = 0 alors P(w)
      V(mutex1)
   V(r)
V(s)
        |
     lecture
        |
      P(mutex1)
         NL ← NL - 1
         si NL = 0 alors V(w)
      V(mutex1)
```

Processus d'écriture

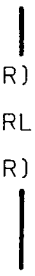
```
P(mutex2)
   NR ← NR + 1
   si NR = 1 alors P(r)        } C
V(mutex2)
 P(w)
   |
 écriture
   |
 V(w)
P(mutex2)
   NR ← NR - 1
   si NR = 0 alors V(r)        } D
V(mutex2)
```


processus de lecture
processus d'écriture
ℓ_1
ℓ_n
e_1
e_m
b_1
b_n
a_1
a_m
∞
1
RL
R
1
W

Les articulations a_i sollicitent R de la même manière que dans le cas précédent. Les articulations b_j ne sollicitent R qu'uniquement au moment de se porter candidates à l'articulation RL .

Inductions réalisées au niveau d'une articulation b_i :

opération DEMARRER (b_i)



S(SDS(R), b_i)
S(SDS(RL), b_i) "b_i sollicite RL"
S(SDS(R), R)

} allocation de R à b_i

opération ARRETER (b_i)

S(SDS(RL), RL) "b_i libère RL"

1/4. SOLUTION ADMETTANT UNE PRIORITE EGALE DES PROCESSUS DE LECTURE ET D'ECRITURE

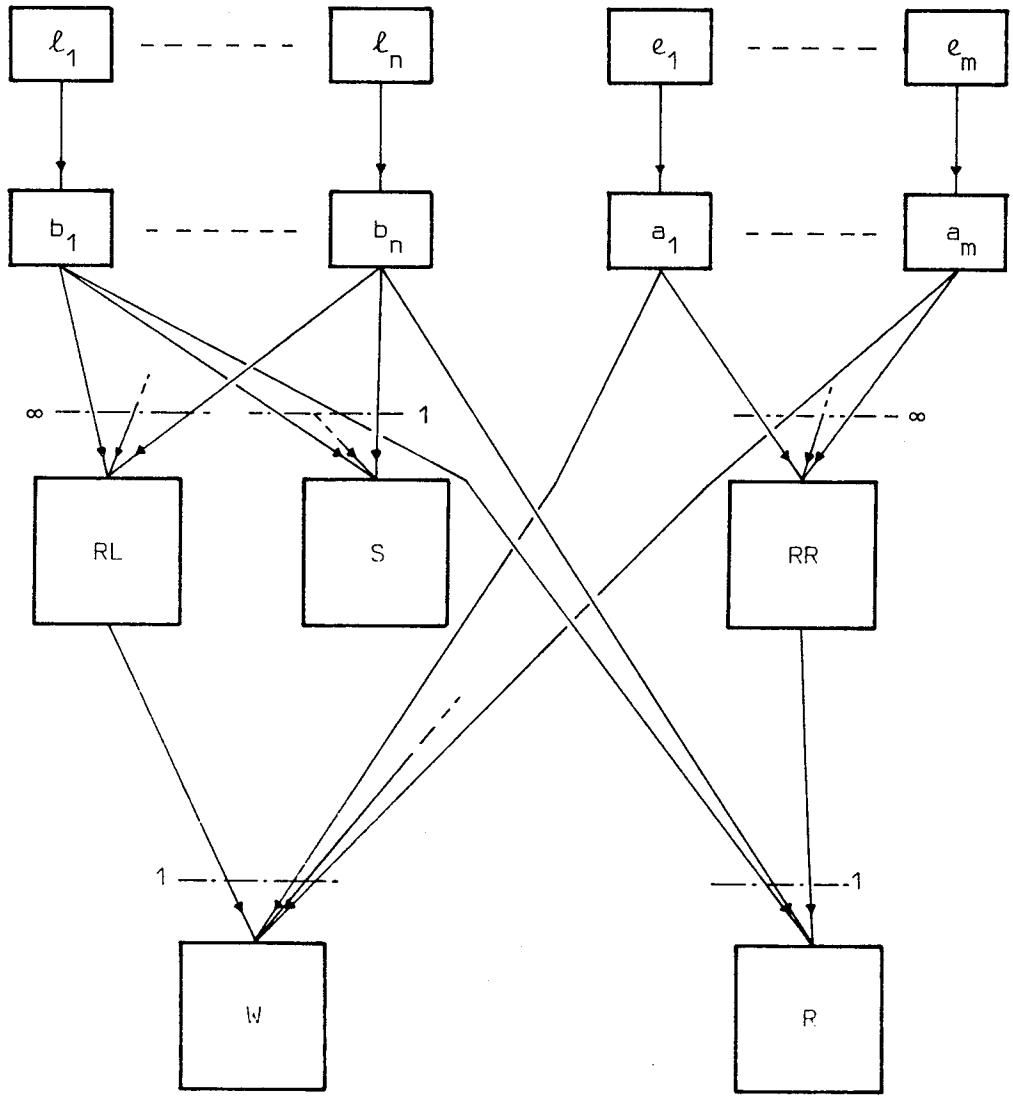
Solution publiée

Processus de lecture	Processus d'écriture
P(r)	P(r)
P(mutex)	P(w)
NL ← NL + 1	▏
si NL = 1 alors P(w)	écriture
V(mutex)	▏
V(r)	V(w)
▏	V(r)
lecture	
▏	
P(mutex)	
NL ← NL - 1	
si NL = 0 alors V(w)	
V(mutex)	

Par rapport à la solution précédente, l'accès à la ressource R est étendu à l'ensemble des articulations représentant les processus de lecture et d'écriture de manière à ce qu'une articulation ℓ_i ne puisse pas, tout en étant activée après une articulation e_j, obtenir l'association, via RL, à W avant cette dernière.

Dans cette solution, une articulation RR multiple et dégénérée, est chargée de regrouper les articulations a_i pour solliciter R. Les articulations RR et R jouent alors un rôle presque symétrique à celui de RL et W. De cette manière, les articulations e_j peuvent garder le contrôle de R tant que des processus d'écriture sont candidats au fonctionnement et ainsi bloquer sur R les articulations b_j candidates.

Il s'avère alors nécessaire d'adjoindre un mécanisme de mutuelle exclusion entre les articulations b_j . Ce mécanisme est réalisé par l'articulation S dont le rôle est d'éviter que plusieurs articulations b_j ne soient candidates à R et ne soient servies avant RR .
Les groupements d'opérations C et D correspondent exactement à des opérations S(SDS(RR), a_i) et S(SDS(RR), RR) compte tenu du fait que l'articulation RR est multiple et dégénérée.



Les articulations a_i sollicitent l'usage de l'articulation RR avant de solliciter celui de W .

Inductions réalisées au niveau d'une articulation a_i :

opération DEMARRER(a_i)

S(SDS(RR), a_i) "a_i sollicite RR"
S(SDS(W), a_i) "a_i sollicite W"
} inductions

opération ARRETER(a_i)

S(SDS(W), W) "a_i se dissocie de W"
S(SDS(RR), RR) "a_i se dissocie de RR"
} inductions

Les articulations b_j sollicitent l'usage de S puis de R uniquement au moment de se porter candidates à l'usage de l'articulation RL .

Inductions réalisées au niveau d'une articulation b_j :

opération DEMARRER(b_j)

S(SDS(S), b_j)
S(SDS(R), b_j)
S(SDS(RL), b_j) " b_j sollicite l'usage de RL"
S(SDS(R), R)
S(SDS(S), S)

} R est alloué à b_j
} S est alloué à b_j

opération ARRETER(b_j)

S(SDS(RL), RL) "b_j se dissocie de RL"

2 - HIERARCHIES DE FORETS CONNECTEES

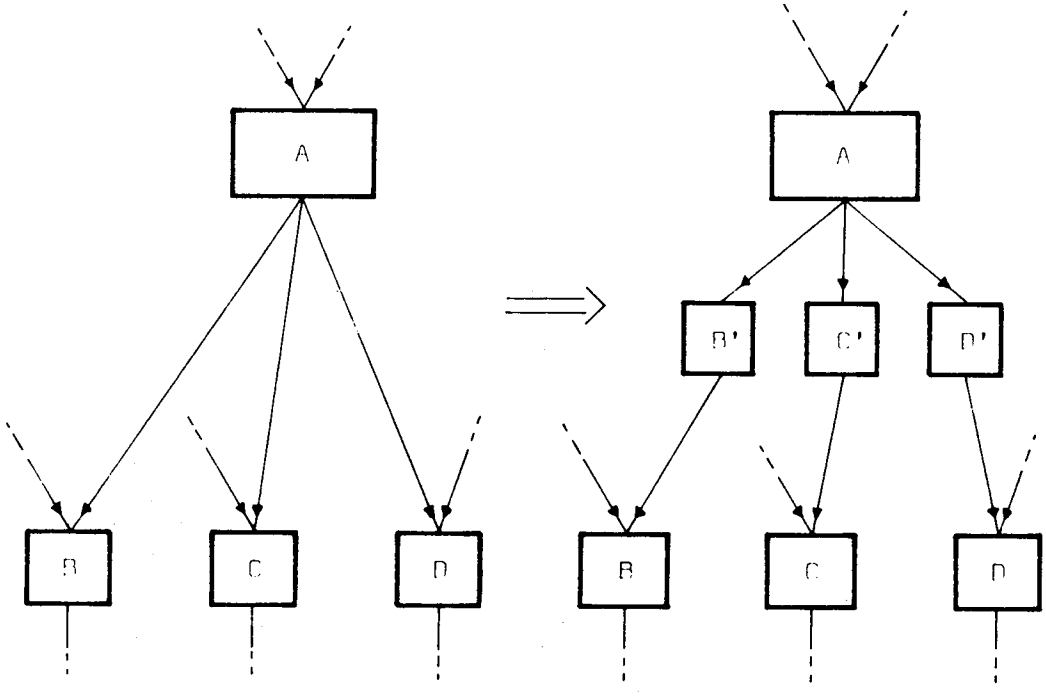
Nous allons, dans le but de simplifier la représentation graphique des systèmes hiérarchisés, restreindre la généralité des hiérarchies utilisées. Dans un premier temps, nous allons classer les articulations suivant que leur mécanisme d'allocation est inexistant ou non:

- la première classe correspondra aux articulations partagées,
- la seconde classe correspondra aux articulations privées (c'est-à-dire non partagées).

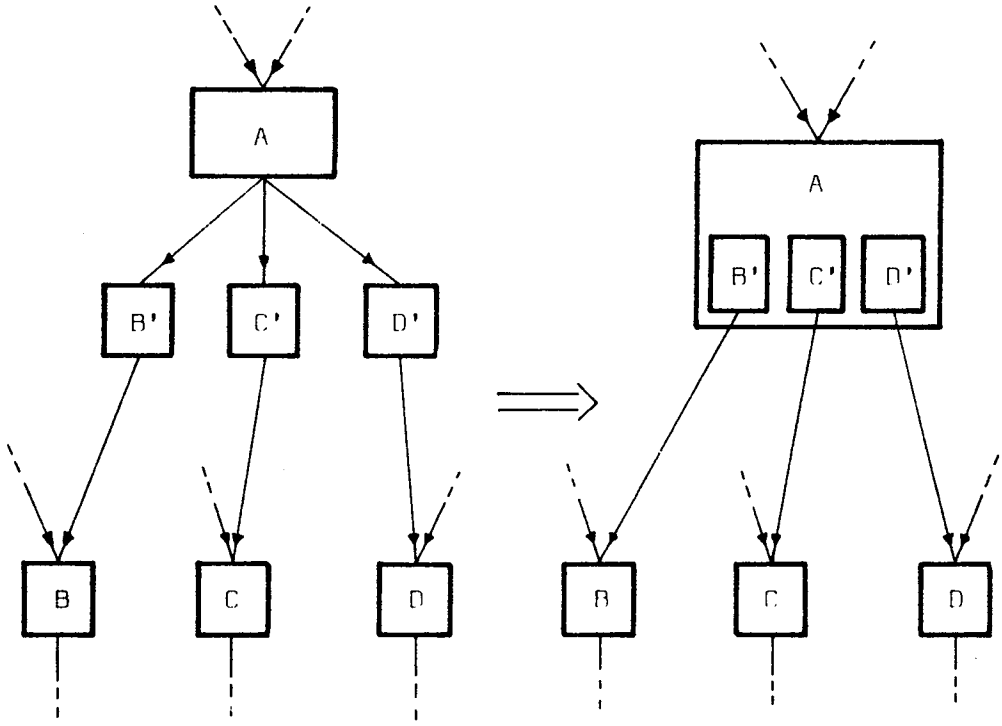
Nous introduirons les restrictions suivantes au niveau de la relation de définition de la hiérarchie:

- une articulation ne peut utiliser
  - . soit qu'une articulation partagée,
  - . soit plusieurs articulations privées.

Remarque:
Cette restriction ne réduit pas le domaine d'emplois du formalisme car il est en effet toujours possible d'intercaler des articulations transparentes privées entre une articulation donnée et celles partagées qu'elle utilise.



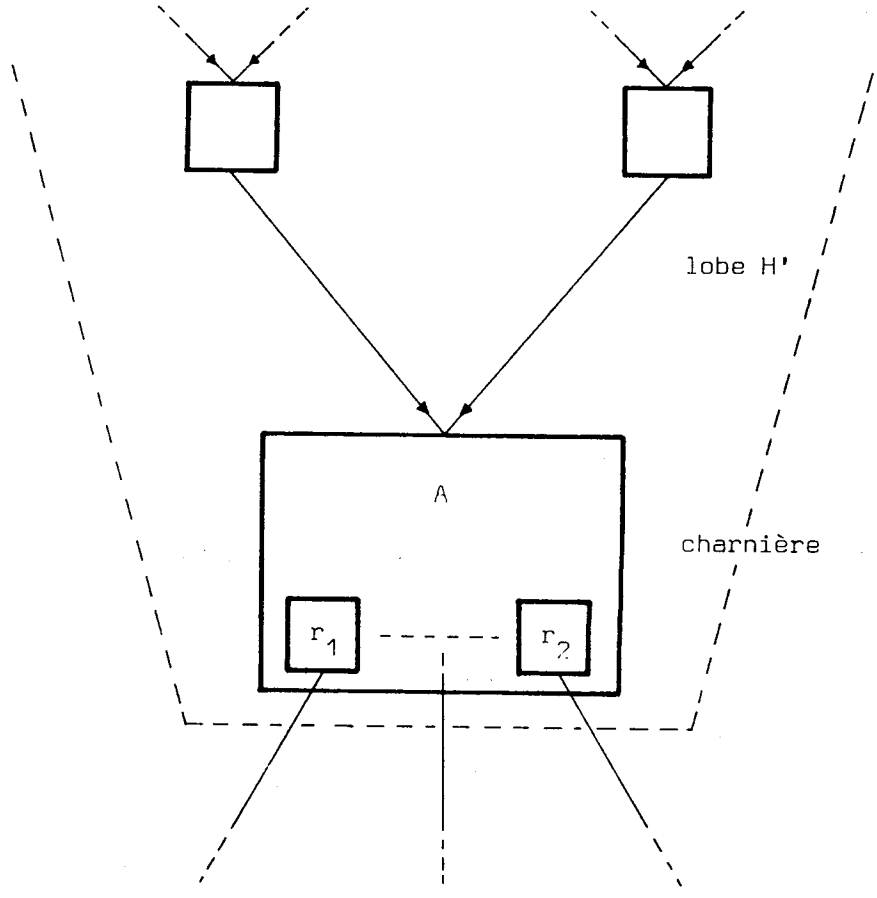
D'un point de vue graphique, cette restriction nous permettra de représent
les articulations de la seconde classe comme incluses dans celles qui les
utilisent.



2/1. INTERET DE CETTE RESTRICTION

Outre la clarté graphique apportée, cette restriction introduit de manière plus apparente la notion d'environnement dans les systèmes représentés.

En effet, l'introduction systématique de ressources virtuelles privées dupliquant les ressources partagées permet de définir l'ensemble des ressources qu'utilise chaque articulation. Cette propriété permet de noter l'environnement utilisé par un lobe de la hiérarchie sans en préciser la réalisation.



Le lobe H' utilise les ressources $r_1, \ldots\ldots, r_n$.

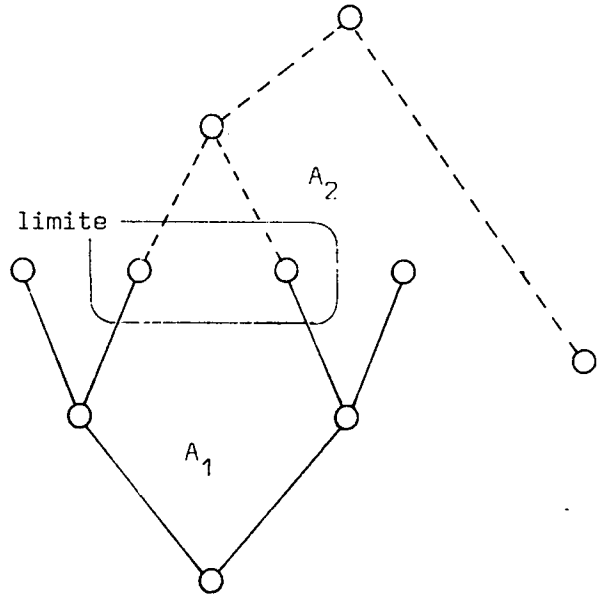
2/2. ARBORESCENCES CONNECTEES PAR LES FEUILLES

2/2.1. Définition

Deux arborescences A1 et A2 sont dites connectées par les feuilles si

- l'intersection de leurs ensembles de feuilles n'est pas vide,
- les ensembles de leurs autres articulations sont disjoints.

Ex.:



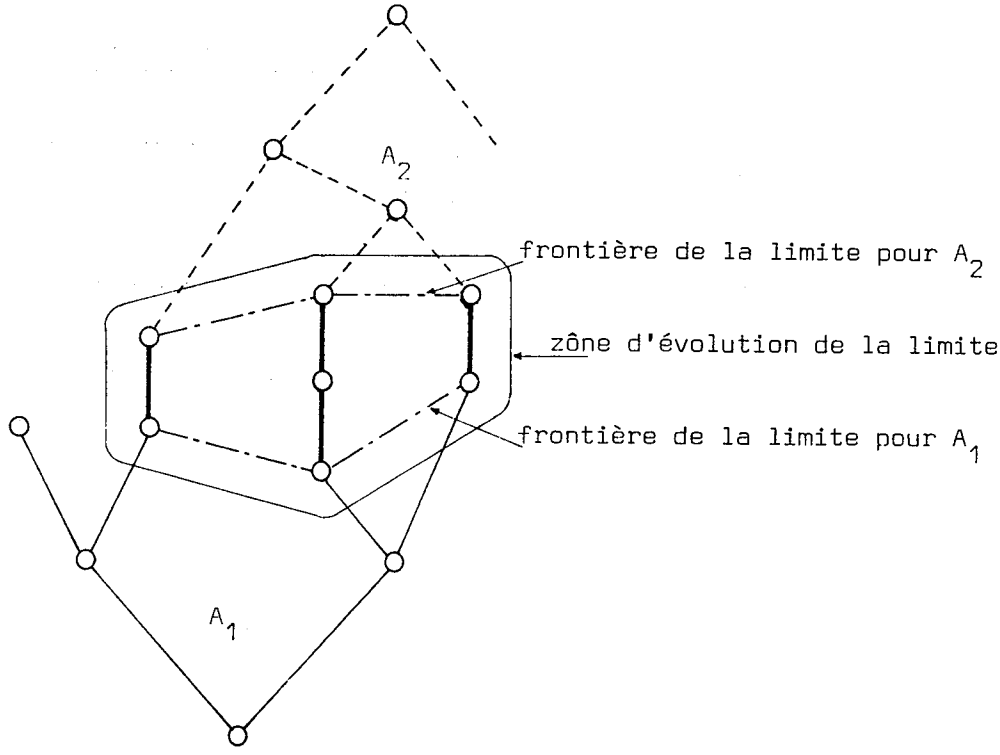
2/2.2. Limite

Nous appelons limite, l'ensemble des feuilles communes à A1 et A2 .

2/2.3. Zône d'évolution de la limite

Considérons l'ensemble des couples d'arborescences qui, connectées par les feuilles, donnent le même réseau, avec la même disposition des racines, à l orientation de quelques connexions près, que celui obtenu par l'opération précédente à partir de deux arborescences A1 et A2.

L'ensemble des articulations appartenant à une limite dans au moins une de ces connexions par les feuilles est appelé la zône d'évolution de la limite.

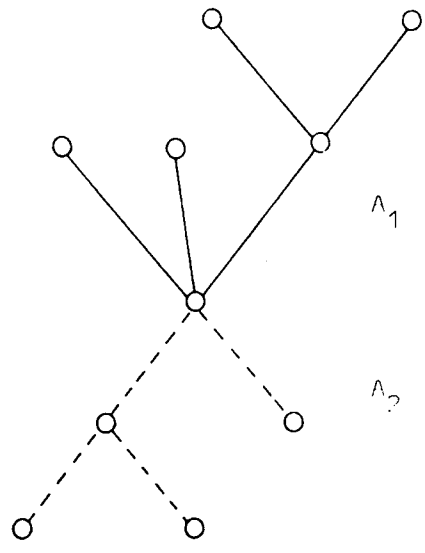


2/3. ARBORESCENCES CONNECTEEES PAR LA RACINE

Deux arborescences sont dites connectées par la racine si

- elles ont la même racine,
- les ensembles de leurs autres articulations sont disjoints.

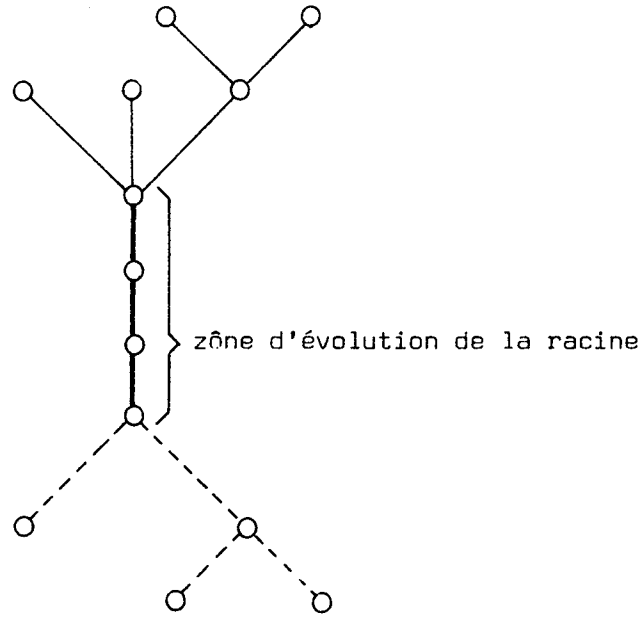
ex.:



2/3.1. Zône d'évolution de la racine

Considérons l'ensemble des couples d'arborescences qui, connectées par la racine, donnent le même réseau, avec la même disposition des feuilles, à l'orientation de quelques connexions près, que celui obtenu par l'opération précédente à partir de deux arborescences A1 et A2.

L'ensemble des articulations jouant le rôle de racine commune dans au moins l'une de ces connexions est appelé la zône d'évolution de la racine.



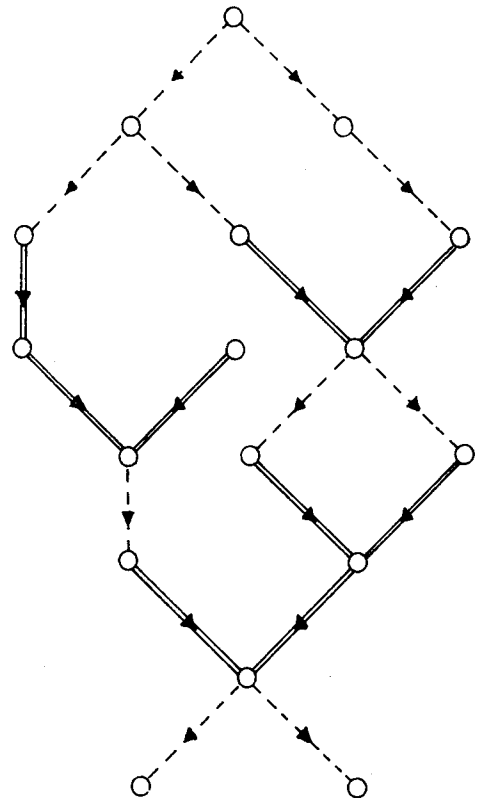
2/4. HIERARCHIES DE FORETS CONNECTEES

Appelons D1 la relation de définition d'une forêt 1 (dite directe).
Appelons D2 la duale de la relation de défintion d'une autre forêt 2 (dit inversée).

Le réseau constitué des arborescences de la forêt directe connectées à celles de la forêt inverse, à l'aide des deux opérations précédentes, aura pour relation de définition:

$$D = D1 + D2$$

Si ce réseau est sans circuit, alors il sera une hiérarchie, car toutes ses connexions seront orientées.



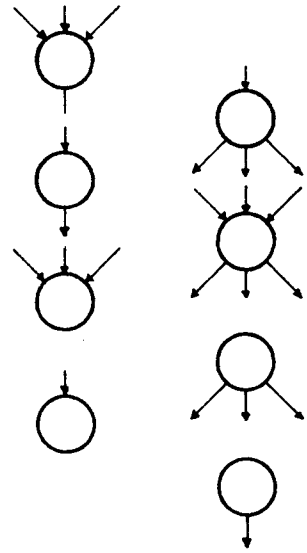
Remarques:

- En conséquence de la remarque précédente, les feuilles et les racines des arborescences ne peuvent être au plus communes qu'à une arborescence directe et à une arborescence inverse.
- Dans ce qui suit, nous admettrons que la forêt inverse ne contient pas d'arborescences réduites à de simples chaînes.

2/4.1. Type des articulations

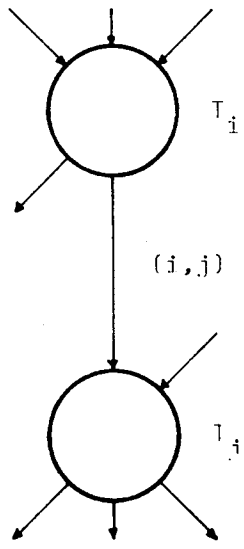
Nous pouvons classer les articulations en huit types suivant leurs demi-degrés entrant et sortant.

type	demi-degré entrant	demi-degré sortant
T_1	> 1	= 1
T_2	= 1	> 1
T_3	= 1	= 1
T_4	> 1	> 1
T_5	> 1	0
T_6	0	> 1
T_7	= 1	0
T_8	0	= 1



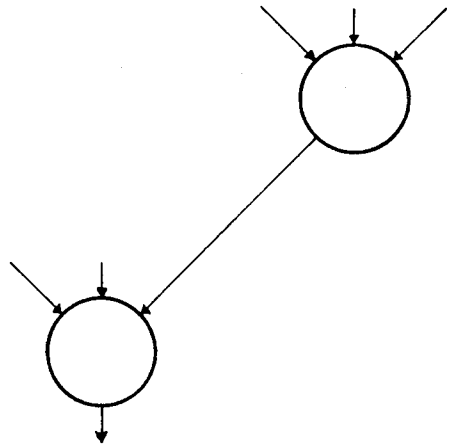
2/4.2. Type d'une connexion

Nous dirons qu'une connexion est de type (i, j) si elle relie une articulation de tupe T_i à une articulation de type T_j .



2/4.3. Propriété

Une hiérarchie de forêts connectées est caractérisée par le fait que pour tout couple d'articulations jointives, l'un des deux demi-degrés correspondants vaut 1.



Ceci entraîne que le réseau ne possède pas de connexions de type (2,1) (2,5) (6,1) (2,4) (4,1) (6,4) (4,5) (4,4) (6,5).

Nous appellerons "interdits" ces types de connexions.

2/4.3.1. Remarques

- Seules les articulations de type T_3 peuvent appartenir à la limite de deux arborescences, c'est-à-dire être des feuilles pour chacune d'elles.
- Seules les articulations des types T_1, T_2 et T_4 peuvent être des racines connectées d'arborescences.
- Seules les articulations des types T_5 et T_7 peuvent être des racines non connectées d'arborescences directes.
- Seules les articulations des types T_6 et T_8 peuvent être des racines non connectées d'arborescences inverses.
- Seules les articulations de type T_8 peuvent être des feuilles non connectées d'arborescences directes.
- Seules les articulations de type T_7 peuvent être des feuilles non connectées d'arborescences inverses.
- Une hiérarchie à élément infimum ne possède qu'une seule articulation de types T_5 ou T_7 qui est sa racine.

2/4.3.2. Démonstration

Nous démontrerons la propriété énoncée en deux temps:

a) Une hiérarchie de forêts connectées ne contient pas de connexions des types interdits.

b) Une hiérarchie ne contenant pas de connexion des types interdits est une hiérarchie de forêts connectées.

a) Supposons qu'une hiérarchie de forêts connectées contienne des connexions de types interdits.

Le tableau ci-après donne, compte tenu des remarques précédentes, l'appartenance des articulations reliées par les connexions interdites.

connexion	T_i	T_j	
(2,1)	I	D	I
(2,5)	I	D	
(6,1)	I	D	
(2,4)	I	R	II
(4,1)	R	D	
(6,4)	I	R	
(4,5)	R	D	
(4,4)	R	R	III
(6,5)	R	R	

D signifie que l'articulation appartient à une arborescence directe.

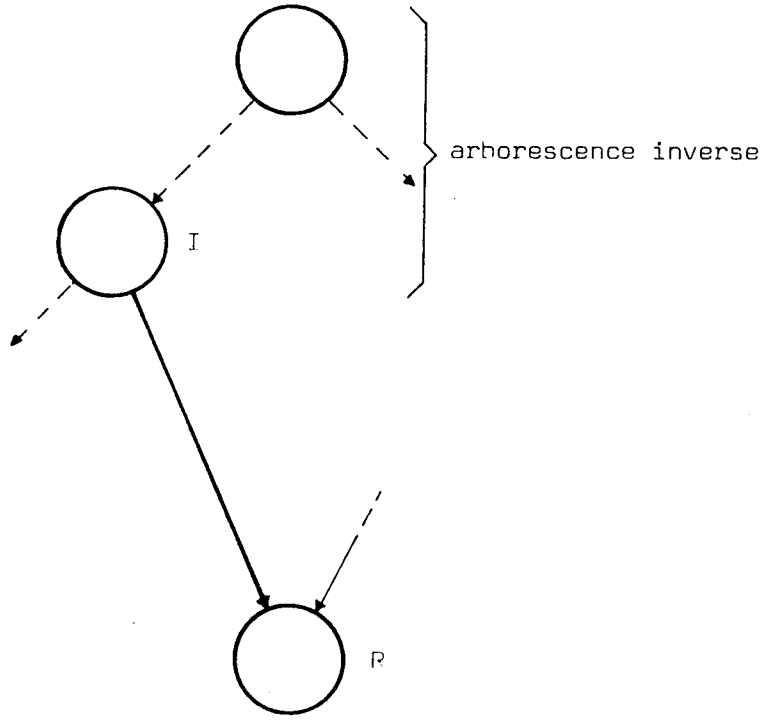
I signifie que l'articulation appartient à une arborescence inverse.

R signifie que l'articulation est une racine connectée.

Les articulations reliées par ces connexions interdites sont de types T_1, T_2, T_4, T_5, T_6. Elles ne peuvent donc pas être des feuilles d'arborescences

Les articulations interdites se classent en trois groupes:

- Le premier groupe I relie deux articulations appartenant à des arborescences de type différent qui ne peuvent être des feuilles, ce qui est impossible.
- Le second groupe II relie une racine à une articulation appartenant à une arborescence qui ne peut avoir pour racine cette première articulation. Ce cas est également impossible puisque ces deux articulations ne peuvent être des feuilles.



- Le troisième groupe III relie deux racines entre elles, ce qui est impossible pour la même raison.

b) Soit un réseau satisfaisant la propriété énoncée.

Remplaçons, dans ce réseau, toutes les chaînes d'articulations de type 3 par une seule de ces articulations. Coupons ce réseau, ainsi transformé, en toutes les articulations de type 3 et 4. Chaque morceau, ainsi créé, est:

. soit une arborescence (de type I ou D),

. soit deux arborescences connectées par la racine.

Cette transformation revient à remplacer toutes les articulations de la forme:

(3, i) par (8, i)

(i, 3) par (i, 7)

(4, i) par (6, i)

(i, 4) par (i, 5)

L'ensemble des connexions permises:

(1,1) (1,4) (1,5) (3,1) (3,4) (3,5) (8,1) (8,4) (8,5) (1,7) (3,7) (2,2 (4,2) (6,2) (2,3) (4,3) (6,3) (2,7) (4,7) (6,7) (8,2) (8,3) (1,2) (1,3 (3,2) (3,3) (8,7)

devient après cette transformation,

(1,1) (1,5) (8,1) (8,5) (1,7) (8,7) (2,2) (6,2) (2,7) (6,7) (8,2) (1,2).

Les assemblages possibles sont manifestement:

- des arborescences de type D

  $8^n\ 5$

  α) $8^n\ 1^*\ 7$ (considéré comme une arborescence de type D)

  β) $8\ \ 7$ (considéré comme une arborescence de type D)

- des arborescences de type I

  δ) $8\ \ 2^*\ 7^m$ (considéré comme une arborescence de type I)

  $6\ \ 2^*\ 7^m$

- des arborescences de types D et I connectées par la racine

  $8^n\ 1^*\ 2^*\ 7^m$

Remarque:

Les morceaux α β δ pourraient être également considérés comme des arborescences de type D et I connectées par la racine.

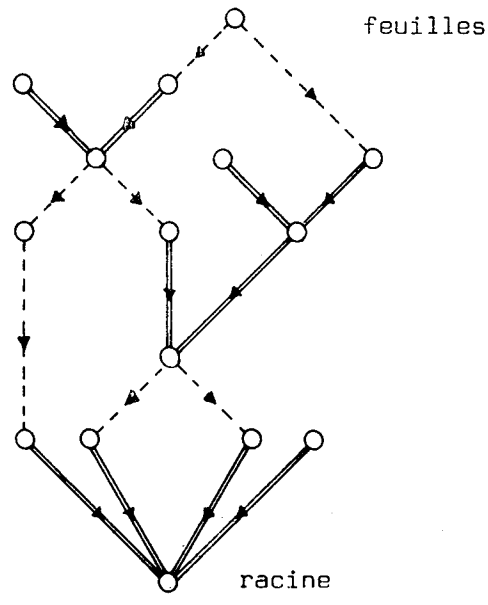
Les opérations de coupures du réseau en les articulations de type 3 correspondent donc à la libération d'arborescences connectées par les feuilles ou par la racine, celles en les articulations de type 4,à la libération d'arborescences connectées par la racine.

La suppression des chaînes d'articulation de type 3 n'a fait que supprimer des zônes d'évolution de limite et de racine entre des arborescences connectées par les feuilles ou par les racines.

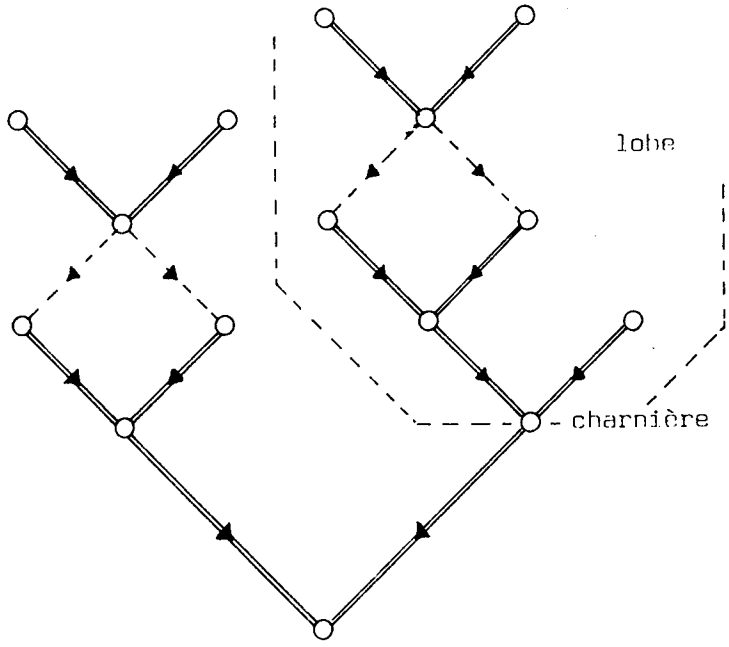
Ceci nous permet d'affirmer que le réseau initial peut être obtenu par la connexion de deux forêts.

2/5. HIERARCHIES DE FORETS CONNECTEES A ELEMENT INFIMUM

Si une hiérarchie obtenue par les opérations précédentes possède un élément infimum alors celui-ci sera la racine d'une arborescence de la forêt directe (compte tenu de la restriction énoncée) et ses feuilles seront soit celles d'arborescences de la forêt directe, soit des racines de la forêt inverse.



Tout lobe de cette hiérarchie est lui-même une hiérarchie de forêts connectées pour lequel la charnière qui le relie au reste de la hiérarchie est soit une feuille, soit l'élément infimum.



3 - SYNCHRONISATIONS ELEMENTAIRES

3/1. EMPILEMENT DES SYNCHRONISATIONS

Les principaux mécanismes évolués de synchronisation utilisent dans leur réalisation d'autres mécanismes de synchronisation plus élémentaires.

Ex.: Les opérations S présentées dans ce travail supposent pour leur réalisation, l'utilisation d'un mécanisme de mutuelle exclusion destiné à gérer l'accès aux SDS utilisées. Dans le cas d'une réalisation programmée des opérations S cette mutuelle exclusion pourrait être obtenue par l'usage des instructions indivisibles de test et de modification d'information en mémoire. Il faut alors remarquer que de telles instructions utilisent elles-mêmes implicitement, pour leur réalisation, la mutuelle exclusion d'accès aux éléments de mémorisation fournie par les mécanismes d'allocation de ces derniers.

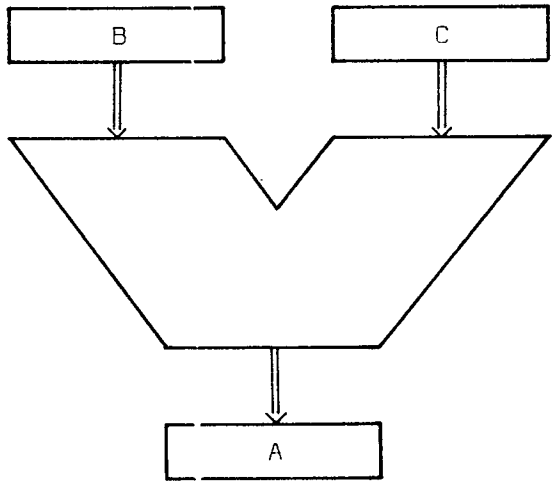
Nous devons, dès lors, nous poser la question de savoir comment peut être réalisé le dernier niveau de synchronisation.

3/2. EVENEMENTS TECHNOLOGIQUES

Les organes technologiques utilisés pour la réalisation des machines informatiques sont le siège d'évènements dont nous allons étudier l'enchaînement. Nous noterons un évènement par une instruction qui le décrit et qui comporte toujours une affectation explicite ou implicite. L'instant d'occurrence de l'évènement est celui où a lieu cette affectation.

Ex.: $A \leftarrow B + C$

représente l'évènement qui consiste à affecter à la variable A la valeur fournie par un réseau combinatoire d'addition connecté aux sorties des variables B et C .



additionneur combinatoire

Remarque:

Un évènement demande, pour se réaliser, la spécification d'un instant particulier au moyen de la transition de l'état d'une variable particulière appelée une horloge.

Nous en omettrons la donnée, contrairement aux langages spécialisés dans ce type de description [6] , car nous ne nous intéresserons qu'à l'enchaînement des évènements.

3/3. ENCHAINEMENT DES EVENEMENTS

Les évènements peuvent être séquentiels c'est-à-dire que leurs instants d'occurrence se succèdent dans un ordre précis dans le temps, collatéraux, c'est-à-dire sans aucune contrainte sur leurs instants relatifs d'occurrenc ou synchrones c'est-à-dire que leurs instants d'occurrence sont rigoureusement simultanés.

Nous adopterons les notations suivantes pour noter ces divers enchaînements

<enchaînement synchrone> ::= | <évènement>
| <évènement>, <enchaînement synchrone>

<ensemble d'évènements> ::= | <enchaînement synchrone>
| (<enchaînement séquentiel>)

<enchaînement collatéral> ::= | <ensemble d'évènements>
| <ensemble d'évènements>;<enchaînement collatéral>

<enchaînement séquentiel> ::= | <enchaînement collatéral>
| <enchaînement collatéral>;<enchaînement séquentiel>

Ex.:

La notation: E; (A, B; C).K;U signifie que les groupes d'évènements

E

(A, B; C).K

U

se suivent séquentiellement.

Le second de ces groupes se décompose en deux sous-groupes collatéraux

A, B; C

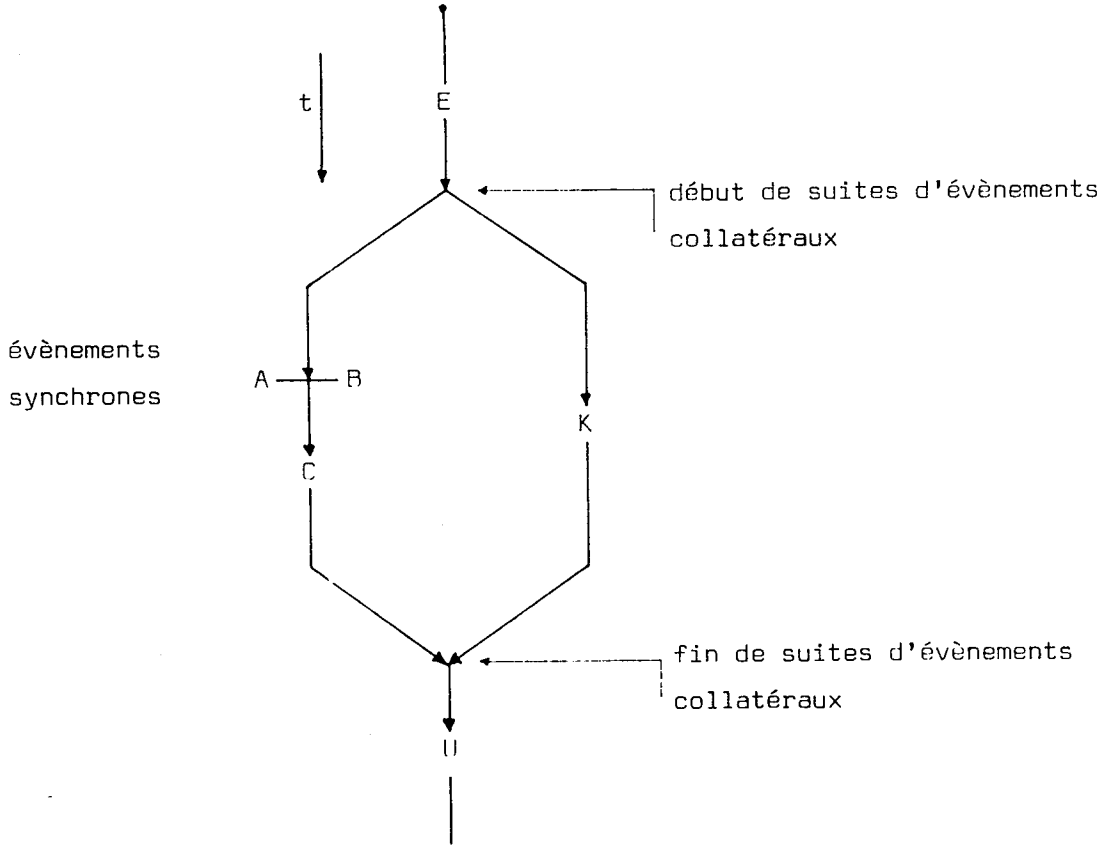
K

Le premier de ces sous-groupes se décompose lui-même en la suite séquentielle de

A, B

C

Un tel enchaînement peut se représenter sous la forme d'un graphe d'enchaînement:

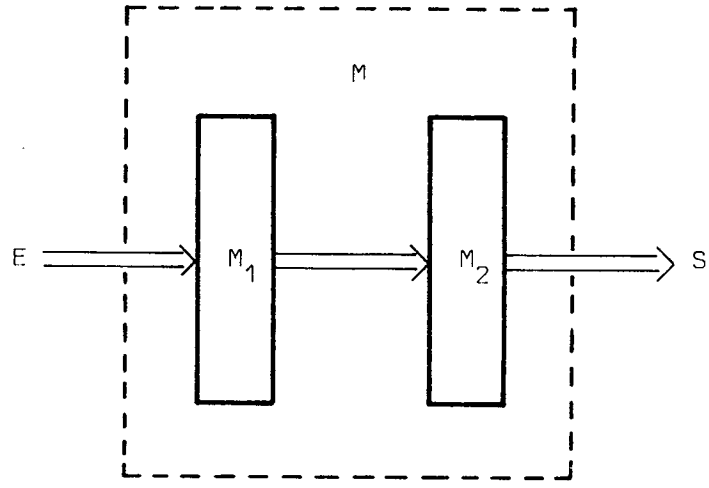


Remarque:

Le synchronisme entre les évènements n'est qu'une approximation qui permet de simplifier la représentation de certains mécanismes. Les évènements élémentaires doivent toujours être considérés comme collatéraux à cause des incertitudes sur les instants désignés par les horloges et sur les vitesses de réponse des organes mis en jeu.

L'approximation qui permet de définir des évènements synchrones peut être obtenue en adoptant un point de vue macrocospique, d'une part sur les évènements eux-mêmes et d'autre part sur les éléments de mémorisation concernés.

Ex.: Un élément de mémorisation M de type maître-esclave est une telle approximation. Il regroupe deux éléments réels de mémorisation M_1 et M_2.



Son fonctionnement nécessite deux évènements successifs $M_1 \leftarrow E$; $M_2 \leftarrow M_1$ que l'on assimile à un seul évènement $M \leftarrow E$ dont l'occurrence pourra être synchrone avec des évènements semblables.

Les évènements synchrones $A \leftarrow B$, $C \leftarrow D$ se décomposent, si A et C sont des organes de mémorisation de type maître-esclave, en l'enchaînement suivant d'évènements élémentaires:

$$A_1 \leftarrow B_2 \ . \ C_1 \leftarrow D_2 \ ; \ A_2 \leftarrow A_1 \ . \ C_2 \leftarrow C_1$$

3/4. ACCES SIMULTANES A UN ELEMENT DE MEMORISATION

Il est utile de préciser les hypothèses concernant les accès collatéraux ou synchrones à un élément de mémorisation.

Soit m un élément de mémorisation. Il peut prendre sa valeur dans un ensemble V et il est supposé initialisé à la valeur $x_o \varepsilon V$. Nous noterons par Φ une valeur quelconque de V .

évènements	état final après l'occurrence des deux évènements:
1 $A \leftarrow m \{'\}_{\cdot} B \leftarrow m$	$m = x_o$, $A = x_o$, $B = x_o$
2 $A \leftarrow m$, $m \leftarrow x_1$	$m = x_1$, $A = x_o$ (par définition)
3 $A \leftarrow m$. $m \leftarrow x_1$	$m = x_1$, $A =$ si $x_1 = x_o$ alors x_o sinon
4 $m \leftarrow x_1 \{'\}_{\cdot} m \leftarrow x_2$	$m =$ si $x_1 = x_2$ alors x_1 sinon ϕ
5 $m \leftarrow$ si C alors x_1 sinon x_2, $C \leftarrow \overline{C}$	$m =$ si C alors x_1 sinon x_2 (par définition)
6 $m \leftarrow$ si C alors x_1 sinon x_2. $C \leftarrow \overline{C}$	$m = \phi$

Remarques:

- Les cas 2 et 5 se situent dans le cadre des approximations permett de définir des évènements synchrones.

- Les cas 3, 4 et 6 se simplifient si V = {0 , 1}

évènements	état final après l'occurrence des deux évènements:
$A \leftarrow m$. $m \leftarrow x_1$	$m = x_1$, $A = x_1$ ou bien x_2
$m \leftarrow x_1 \{'\}_{\cdot} m \leftarrow x_2$	$m = x_1$ ou bien x_2
$m \leftarrow$ si C alors x_1 sinon x_2 . $C \leftarrow \overline{C}$	$m = x_1$ ou bien x_2

3/5. INCERTITUDES DANS LA PROGRESSION DES PROCESSUS

La progression de l'algorithme d'un processus doit être vue comme une suite d'affectations à un variable qui représente son état algorithmique (celà signifie que nous ne nous intéresserons qu'aux processus représentés par des machines séquentielles dites "synchrones", c'est-à-dire des machines dont les progressions individuelles sont rythmées par des horloges. Celles-ci peuvent d'ailleur être locales aux processus).

Ces affectations obéissent aux règles précédentes, en particulier un enchaînement conditionnel tel que:

aller à si C alors L1 sinon L2

risque d'avoir pour résultat le chargement d'un état algorithmique aléatoire voire incohérent (c'est-à-dire ne correspondant à aucun des enchaînements souhaités) s'il se produit simultanément, de manière collatérale, à une modification de la condition C .

Parmi les solutions envisageables pour remédier à cet inconvénient, nous pouvons citer la décomposition des évènements d'enchaînement conditionnels en deux évènements successifs:

- test de la condition ;
- affectation d'une nouvelle valeur à l'état algorithmique (indépendamment de la valeur courante des opérandes de condition).

L'incertitude ne se traduira alors que par un résultat aléatoire du test. L'enchaînement qui en résultera s'effectuera donc obligatoirement vers l'une des alternatives possibles.

3/6. SYNCHRONISATIONS ELEMENTAIRES

Les synchronisations élémentaires entre des organes physiques réalisant des processus peuvent être:

- des synchronisations élémentaires correspondant à des envois de messages binaires destinés à permettre à un processus d'attendre l'occurrence d'un évènement exécuté par un autre.

- des mutuelles exclusions destinées à gérer l'accès à une ressource commune à plusieurs processus.

Remarque:

A ce niveau de réalisation les critères d'optimisation sont très différents

de ceux utilisés au niveau des réalisations programmées.

- Le nombre des processus n'est pas limité. Il est naturellement grand car chaque organe technologique a un comportement indépendai de celui de son voisin, sauf si un mécanisme particulier le lui interdit.

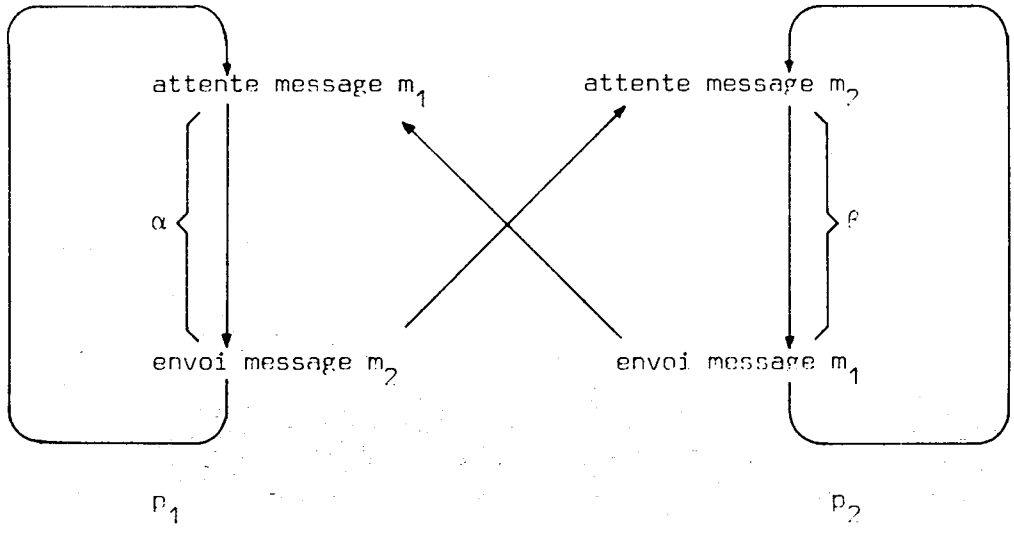
- Une attente peut être, sans inconvénient, active, c'est-à-dire réalisée par une boucle dans l'algorithme associé au processus co sidéré. Cette solution est utilisable car les organes de ce nive n'utilisent pas de ressources communes principales et requérables En effet, s'ils en utilisaient, celles-ci pourraient leur fournir un niveau supplémentaire de mécanismes de mutuelle exclusion et i ne seraient plus chargés d'assurer les synchronisations élémentai res.

3/6.1. Echanges élémentaires de messages

La construction de mécanismes élémentaires d'échanges de messages revient à réaliser technologiquement des opérations S . Elle est relativement simple dans la limite où l'on restreint fortement la généralité des mécanismes:

- Les SDS Δ utilisées sont bi-liées, c'est-à-dire que chaque clas d'êtres à synchroniser ne comprend qu'un élément.
  soit M = {m} la classe des messages et P = {p} celle des processus.

- Les échanges de messages sont toujours bi-directionnels et se situent toujours dans le cadre de relations demandes-réponses.



Cette restriction permet d'assurer la mutuelle exclusion des sections α et β .

- Si un mécanisme d'attente passive est employé, l'opération S de mise en attente devra toujours strictement précéder celle de libération.

S (Δ, p) ; S(Δ, m)

3/6.1.1. Attente passive

La progression de l'algorithme du processus p considéré est supposée arrêté pendant une attente.

```
opération S(Δ, m)
   ATTENDU (m) :
   PRENDRE (p, FP)
   ASSOCIER (m, p)
   DEMARRER (p)
```

```
opération S(Δ, p)
si FM = Ø alors
   CONCURRENT (p)
   METTRE (p, FP)
   ARRETER (p)
```

Compte tenu des restrictions énoncées, la file FM est toujours vide et celle FP peut être représentée par un simple indicateur binaire BP tel que:

$$BP = 1 \leftrightarrow p \in FP$$

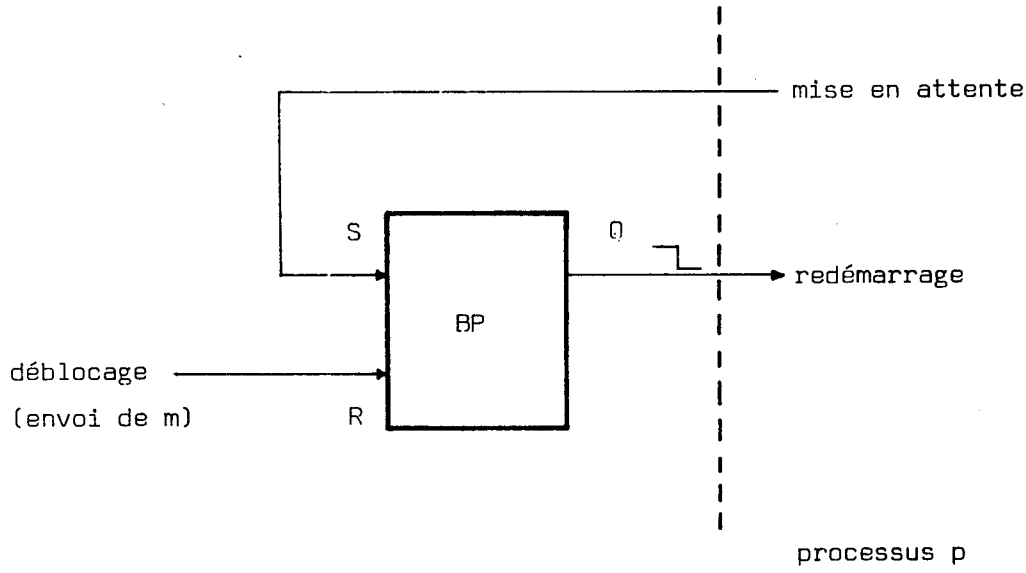
La stricte séquentialité dans l'exécution de ces opérations S assure leur mutuelle exclusion d'exécution.
Ces opérations deviennent:

```
opération S(Δ, m)
   BP ← 0
```

```
opération S(Δ, p)
   BP ← 1
```

La transition de 1 à 0 de la valeur de l'indicateur BP relance la progression de p .

Ex.: Un tel mécanisme de synchronisation peut être réalisé par une simple bascule de type RS .



3/6.1.2. Attente active

Dans ce cas, l'attente d'un processus est réalisée par une boucle dans son algorithme. Dans ces conditions, les opérations S deviennent:

```
opération S(Δ,-) "exécutée par p"
boucler tant que FM = Ø
        PRENDRE (m, FM)
        ASSOCIER (m, p)
```

```
opération S( Δ, m)
    METTRE (m, FM)
```

La file FP est inexistante à cause de l'attente active et celle FM peu être représentée par un simple indicateur binaire BM tel que:

$$BM = 1 \leftrightarrow m \in FM$$

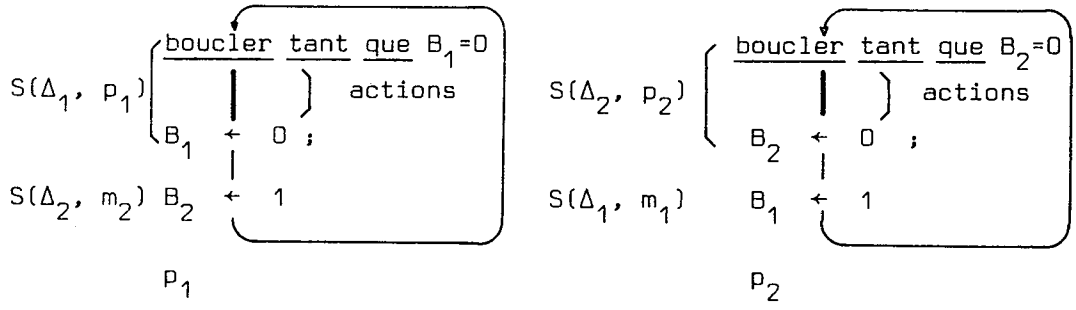
Les opérations S deviennent:

```
opération S(Δ,-) "exécutée par p"
boucler tant que BM = 0 ;
                  α
      β  BM ← 0
```

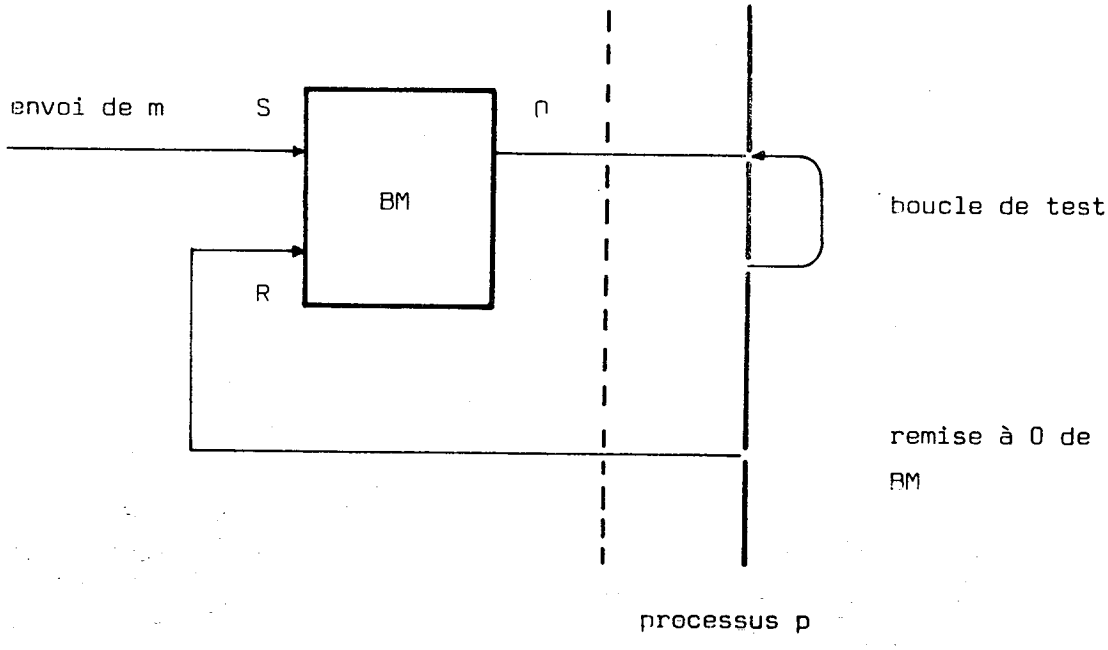
```
opération S(Δ, m)
γ  BM ← 1
```

L'exécution de ces opérations ne nécessite pas de mécanismes supplémentaires de mutuelle exclusion. En effet,

- les évènements α et γ peuvent s'exécuter de manière simultanée collatérale car BM ne peut prendre que les valeurs 0 et 1. Si le résultat du test est 0, p exécutera un pas de plus dans sa boucle d'attente au terme duquel il obtiendra un résultat correct. Si ce résultat est 1 , cette valeur est compatible avec l'occurrence de γ et p sortira de sa boucle d'attente ;
- la restriction de n'utiliser de telles opérations que dans des dialogues bi-directionnels, assure que les évènements β et γ sont mutuellement exclusifs. Cette propriété permettra de déplacer l'occurence des évènements jusqu'à précéder l'occurrence des opérations $S(\Delta,m)$ de réponse.



Ex.: Un tel mécanisme de synchronisation peut être réalisé à l'aide d'une simple bascule de type RS .



3/6.2. Allocateurs élémentaires

Il s'agit de mécanismes d'allocation de ressources simples et élémentaires du système. Ces mécanismes peuvent soit être locaux à cette ressource, soi être répartis entre ses utilisateurs.

Soit r une ressource élémentaire gérée par une SDS Δ qui lui est liée. Un utilisateur u_i exécutera une opération $S(\Delta, -)$ pour se porter candic à r et $S(\Delta, r)$ pour s'en dissocier.

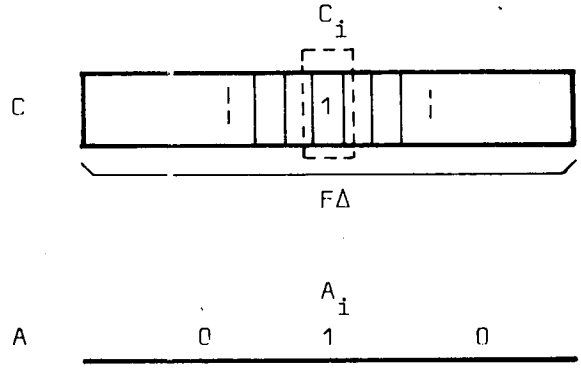
Les attentes impliquées par ces opérations sont obligatoirement actives puisqu'il s'agit du dernier niveau de mécanismes utilisable.

Ces allocateurs élémentaires seront également dits assurer la mutuelle exclusion entre les utilisateurs u_i de r. A ce titre, ils obéissent à la condition énoncée par BREDT [14]:

> Un organe de commande susceptible d'assurer la mutuelle exclusion entre n processus doit avoir au moins $n+1$ états internes.

3/6.2.1. Allocation autonome d'une ressource élémentaire

La ressource r est supposée munie d'un mécanisme d'allocation autonome. La fille $F\Delta$ de sa SDS sera représentée par un ensemble de bascules C_i liées à chaque utilisateur:



La désignation dans $F\Delta$ de l'utilisateur associé est faite grâce à un autre vecteur A de mêmes dimension que C dont la composante d'indice correspondant à l'utilisateur associé est égale à 1.

Les opérations suivantes correspondront donc:

METTRE(u_i, F) $\Rightarrow$ $C_i \leftarrow 1$

EXTRAIRE(u_i, F) $\Rightarrow$ $C_i \leftarrow 0$

ASSOCIER(u_i, r) $\Rightarrow$ $A_i \leftarrow 1$

DISSOCIER(u_i, r) $\Rightarrow$ $A_i \leftarrow 0$

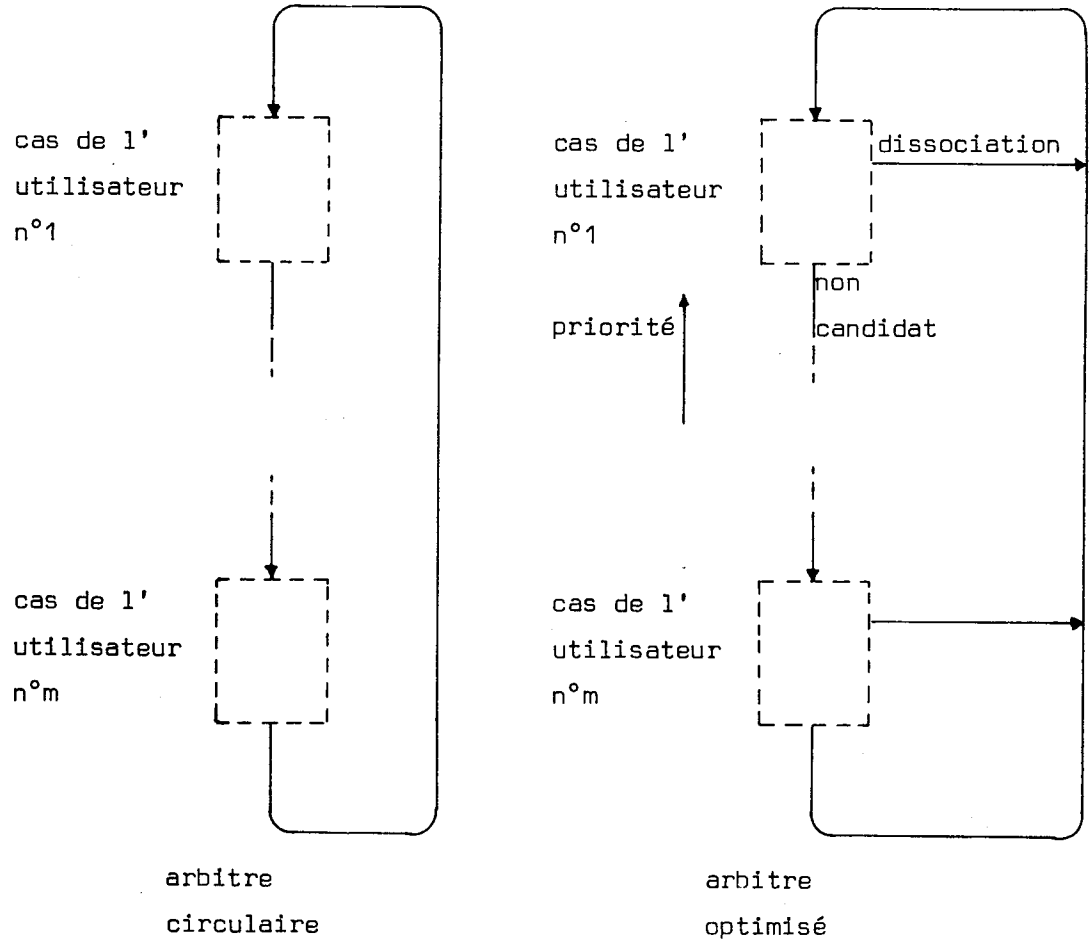
3/6.2.1.1. Arbitre temporel

Le mécanisme d'allocation MA de la ressource r explore alternativement les différents candidats u_i, analyse leurs éventuelles demandes d'association, les associe éventuellement puis attend qu'ils désirent se dissocier.

utilisateur u_i	ressource r
opérations S(Δ,-)	mécanisme autonome d'allocation
$C_i \leftarrow 1$; α	
boucler tant que $A_i = 0$;	Li : si $C_i \neq 0$ alors aller à Li+1 ; β
	$A_i \leftarrow 1$;
opération S(Δ, r)	boucler tant que $C_i = 1$;
$C_i \leftarrow 0$;	$A_i \leftarrow 0$;
	Li + 1 : - - - - -

La simultanéité entre les opérations collatérales α et β provoquera un enchaînement aléatoire soit vers Li + 1 soit vers l'évènement suivant. Dans le premier cas l'allocateur ne prend pas la requête en compte à ce tour mais la trouvera au suivant tandis qu'elle est prise en compte dans le second cas.

Remarque: La structure de l'allocateur peut être telle qu'elle favorise les utilisateurs les plus prioritaires.



3/6.2.1.2. Arbitre resynchronisant

Un mécanisme élémentaire d'allocation rapide peut être obtenu en resynchronisant les demandes avant de prendre une décision indépendante de leur évolution après la resynchronisation.

Cette solution suppose l'existence d'un troisième vecteur D jouant avec le rôle de FΔ . Le contenu de C sera transféré dans D au moment de la resynchronisation.

utilisateur u_i

opération S(Δ, -)

$C_i \leftarrow 1$;

boucler tant que $A_i = 0$;

opération S(Δ, r)

$C_i \leftarrow 0$;

ressource r

mécanisme autonome d'allocation :

L : si $\bigcup_i C_i = 1$ alors D ← C

sinon aller à L ;

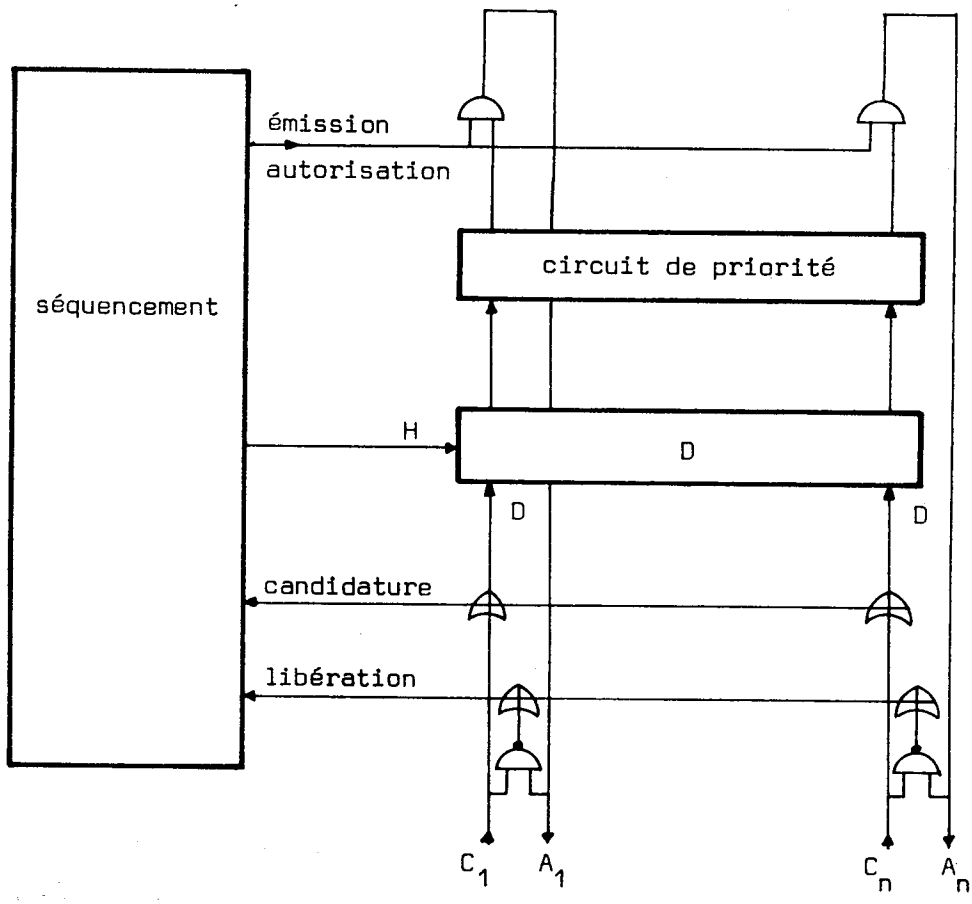
$A_{min(j = i \text{ tel que } D_i = 1)} \leftarrow 1$;

boucler tant que $C_j = 1$;

$A_j \leftarrow 0$;

aller à L ;

Un tel mécanisme peut aisément être câblé de la manière suivante:



3/6.3.1.3. Arbitre de BREDT [14]

Il s'agit de l'arbitre susceptible de gérer n candidats en ayant le minimum d'états internes.

Les opérations exécutées par les utilisateurs sont les mêmes que précédemmer Le mécanisme autonome d'allocation de la ressource est:

Lo: A ←0 . aller à si U_i C_i = 1 alors $L_{min(i \text{ tel que } C_i = 1)}$ sinon Lo ;

L1: ----------

Li: A_i ← 1 . $A_{\neq i}$ ← 0 . aller à si C_i = 1 alors L_i

sinon si U_j C_j = 1 alors $L_{max(j \text{ tel que } C_j =}$

sinon Lo ;

Ln: ----------

3/6.2.2. Solution répartie

La ressource r n'est plus supposée disposer d'un mécanisme d'allocation en propre. Sa gestion sera effectuée par ses utilisateurs lors de l'exécution des opérations de synchronisation.

3/6.2.2.1. Solution de DEKKER [28]

Il s'agit d'opérations de synchronisation permettant à deux utilisateurs u_1 u_2 de s'allouer de manière exclusive une ressource élémentaire r . C mécanisme nécessite un indicateur binaire C_i par utilisateur et un indicateur binaire T commun aux deux et susceptible de contenir leur identité codée par 0 et 1.

opération S(Δ, -) "exécutée par un utilisateur u_i"

L1: C_i ← 1 ;

L2: aller à si C_j = 1 alors si T = i alors L2 sinon L3 sinon sorti

L3: C_i ← 0 . si T = j alors aller à L3 ;

opération S(Δ, r) "exécutée par l'utilisateur u_i"

T ← j . C_i ← 0 ;

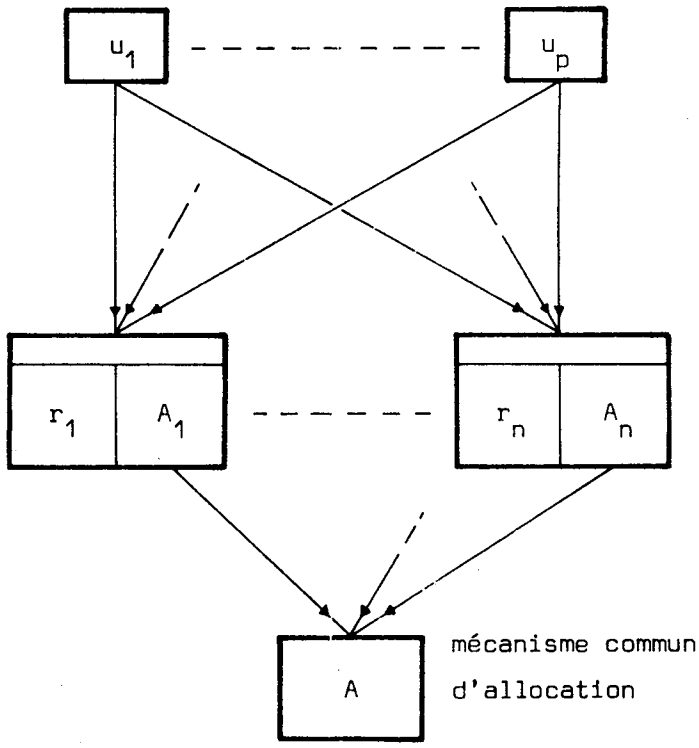
Remarque: L'algorithme présenté par DIJKSTRA[27] n'est pas utilisable au niveau élémentaire car il suppose l'existence d'une variable commune aux différents processus à synchroniser et pouvant être chargée par leurs indices respectifs. Compte tenu des hypothèses précédentes concernant les affectations simultanées à une variable, celle-ci peut alors prendre une valeur qui ne correspond pas à l'indice de l'un des processus candidat et permettre ainsi un blocage dynamique de la décision pour certaines vitesses relatives d'exécution de ceux-ci.

4 - MECANISMES GLOBAUX D'ALLOCATION DE RESSOURCES

Le point de vue que nous avons adopté consiste à associer à chaque ressou son propre mécanisme d'allocation. Une telle organisation conduit à intr duire dans le système ainsi défini des risques d'interblocage et d'attent infinie de la disponibilité d'un ensemble de ressources.

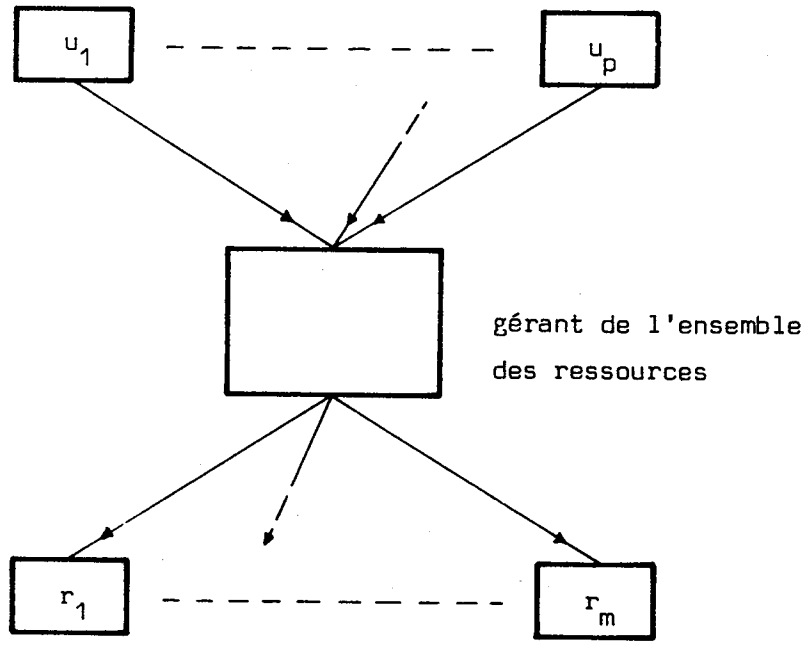
4/1. ALLOCATION CENTRALISEE DES RESSOURCES

Une solution pour pallier à ces inconvénients consiste à regrouper les mécanismes d'allocation des différentes ressources dans un mécanisme unic capable de prévenir de telles situations.



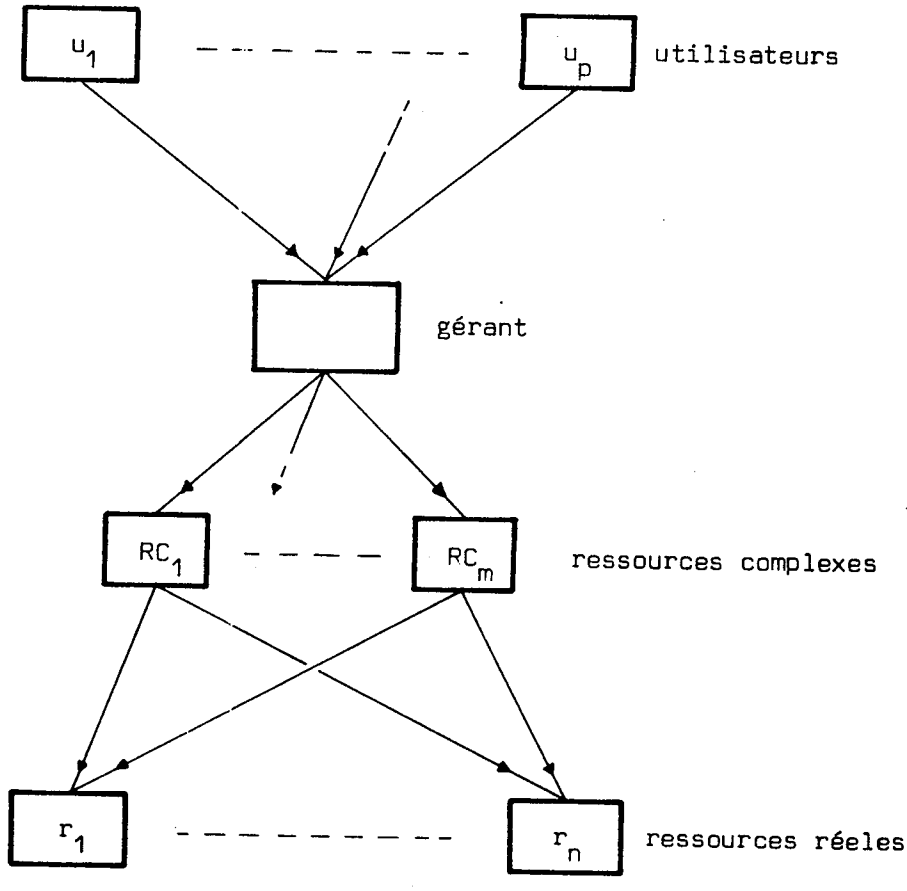
L'articulation A est informée de toutes les requêtes d'allocation et peut donc décider de la manière dont les différentes ressources doivent s'associe à leurs candidats, à l'aide d'algorithmes particuliers de décision (algorith me du banquier [40] permettant la gestion de ressources non requérables, (algorithme de conseravtion de l'espace suffisant de travail dans le cas de la gestion de ressources mémoires).

Une autre solution consiste à regrouper toutes les ressources du système en une articulation multiple par laquelle transitent toutes les requêtes issues des différents utilisateurs. Le gérant de cette articulation multipl ré-émet les requêtes vers les ressources réelles dans un ordre tel qu'il évite les interblocages.



ressources du système

Une solution très fréquemment utilisée consiste à sélectionner à tout inst et au vu des requêtes, un sous-ensemble des utilisateurs qui peuvent utili les ressources sans risque d'interblocage. Le système peut alors être vu comme le regroupement, par une articulation multiple, d'un certain nombre de ressources complexes représentant une certaine partition de l'ensemble ressources du système. L'articulation qui gère cet ensemble associera un utilisateur privilégié à chacune de ces ressources complexes.



4/2. Décentralisation des mécanismes d'allocation de ressources

Nous pouvons chercher à montrer sous quelles conditions une certaine décentralisation des mécanismes d'allocation de ressources peut être réalisée.

4/2.1. Cas de ressources non requérables (Interblocages)

Définissons la relation d'équivalence suivante entre les ressources d'un système:

$r_1 \overset{u}{\frown} r_2$ <=> l'articulation u utilise les ressources r_1 et r_2

Le système présentera un risque d'interblocage s'il existe une chaîne:

$r_1 \overset{u}{\frown} r_2 \overset{u' \neq u}{\frown} \ldots\ldots\ldots\ldots\ldots\ldots u^n \neq u^{n-1} \frown r_k$

telle que:

$$r_k = r_1$$

Il est en effet aisé d'imaginer, pour un système répondant à cette condition, une situation telle que toutes les articulations $u, u', \ldots\ldots u^{n-1}$ n'obtiennent chacune qu'une partie des ressources qu'elles sollicitent sans qu'aucune ne puisse les obtenir toutes.

Ex.: Le problème des cinq dîneurs philosophes [30].

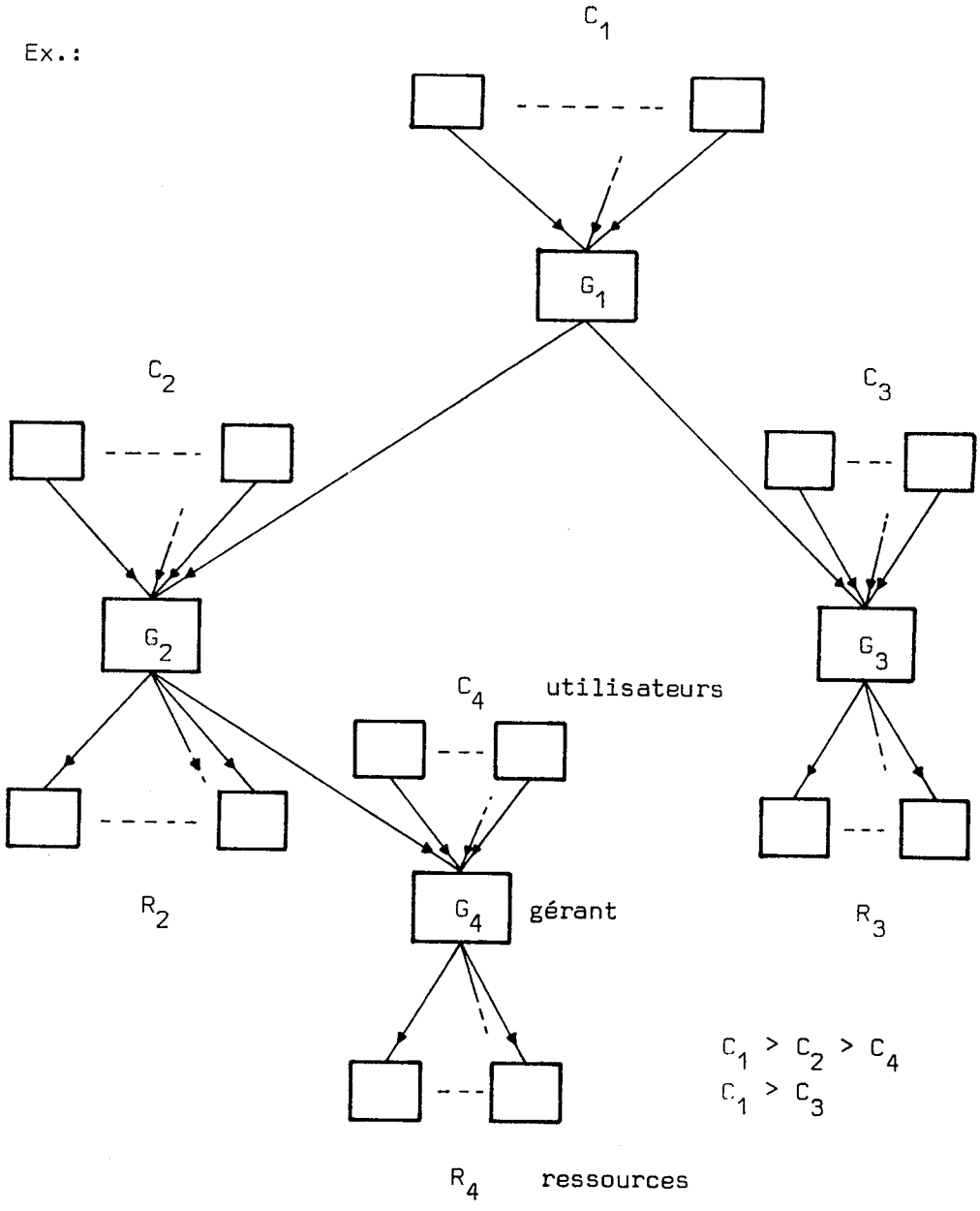
4/2.1.1. Arborescence d'articulations d'allocation

Partitionnons l'ensemble des utilisateurs u_i des ressources de telle façon que nous puissions structurer les classes obtenues de manière arborescente par la relation suivante:

soit C_i, C_j des classes d'utilisateurs

$C_i > C_j$ <=> l'ensemble des ressources utilisées par les articulations de la classe C_j est contenu dans celui de la classe C_i .

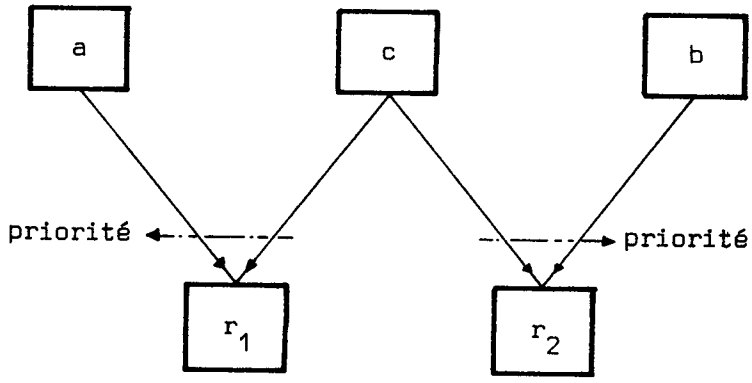
Nous pouvons déduire de cette partition et de cette relation, une arborescence d'articulations gérant chacune, de manière centralisée, un sous-ensemble des ressources.



Chaque articulation de gestion gère une classe d'utilisateurs et reçoit des requêtes des articulations de gestion des classes supérieures à la sienne. Cette articulation gère sa file FU de manière à servir ces demandes dans un ordre tel qu'il n'y ait, à son niveau, aucun risque d'inte blocage.

4/2.2. Cas des ressources requérables (Risque d'attente infinie)

Il serait possible de chercher à éliminer un tel rique d'un système hiérarchisé. Il suffit pour celà d'éliminer tout sous-réseau de la forme:



En effet, un tel sous-réseau comporte des risques d'attente infinie d'allocation. Une coalition des articulations a et b, respectivement plus prioritaires que c relativement à r_1 et r_2, peut retarder infiniment c sans que ni a ni b ne sollicitent r_1 et r_2 pour des périodes d'allocation supérieures à une valeur donnée.

REFERENCES BIBLIOGRAPHIQUES

|1| ABRIAL J. R.
Description d'un modele synthetique de fonctionnement
d'un systeme moderne
Séminaire IRIA mars 71

|2| ANCEAU F.
Microprogrammed system for tasks management
NATO Advanced Institute on Microprogramming aout 71 Hermann p 2

|3| ANCEAU F.
synchronisation dans les systemes hierarchisés
de processus/processeurs
Congrès AFCET Grenoble novembre 72

|4| ANCEAU F.
Hierarchical structures in computer architecture
Ecole CEE Alpe d'Huez décembre 73

|5| ANCEAU F. , BEAUDUCEL C. , COURBOULET P. , CRETIN J. , DROUET P.
GEOPROCESSEUR : a computer for geophysical researchs
IFIP congress Stockolm aout 74

|6| ANCEAU F. , LIDELL P. , MERMET J. , PAYAN C.
A language to describe digital systems .
Application to logic design
Third Symposium on Computer and Information Science
COIN Miami décembre 69 Academic Press vol 1 p 179

|7| ANDERSON D. W. , SPARACIO F. J. , TOMASULO R. M.
The IBM System / 360 model 91 : machine philosophy
and instruction handling
IBM journal janvier 67 p 8

|8| ASCHENBRENNER Richard A. , FLYNN Michael J. , ROBINSON Georges A.
Intrinsic multiprocessing
Spring Joint Computer Conference 67 p 81

|9| BASIJI Jamshid , BERGH Arndt B.
Central bus links modular HP 3000 hardware
Hewlett-Packard Journal vol 24 n.5 janvier 73

|10| BATAILLE M.
The GAMMA 60 , the computer that was ahead of its time
Honeywell Computer Journal vol 5 n.3 71 p 99

|11| BELPAIRE G. , WILMOTTE J. P.
Correctness of realisation of levels of abstraction
in operating systems
Congrès sur les aspects théoriques et pratiques
des systèmes d'exploitation
IRIA avril 74

|12| BETOURNE C. , BOULENGER J. , FERRIE J. , KAISER C. , KRAKOWIAK S. , MOSSIERE J.
Présentation génerale du système ESOPE
Congrès d'informatique AFCET Paris septembre 70

|13| BLEASE Thomas A. , HEWER Alan
Single operating system serves all HP 3000 users
Hewlett-Packard Journal vol 24 n.5 janvier 73

|14| BREDT Thomas H.
The mutual exclusion problem
Technical report n.9 Digital systems laboratory
STANFORD university aout 70

|15| BRIAT J. , GUIBOUD RIBAUD S.
Espace d'adressage et espace d'exécution du système GEMAU
Congrès sur les aspects théoriques et pratiques
des systèmes d'exploitation
IRIA avril 74

|16| BRINCH HANSEN P.
The nucleus of a multiprogramming system
Communication of the ACM vol 13 n.4 avril 70 p 238

|17| BRINCH HANSEN P.
Multiprogramming with monitors
Carnegie Mellon University novembre 71

|18| BRINCH HANSEN P.
A comparison of two synchronisizing concepts
Acta Informatica n.1 72

|19| BRUNET claude
Etude d'une structure biprocesseur banalisé
Thèse INPG Grenoble mars 74

|20| CII
Documemtation de l'ordinateur IRIS 80

|21| CONWAY Melvin E.
A multiprocessor system design
Fall Joint Computer Conference 63

|22| COURTOIS P. J. , HEYMANS F. , PARNAS D. L.
Concurrent control with reader and writers
Communications of the ACM vol 14 n.10 octobre 71 p 667

|23| CREECH B. A.
Architecture of the B6500
Third Symposium on Computer and Information Science
COIN Miami décembre 69 Academic Press vol 1 p 29

|24| DALEY R. , DENNIS J. B.
Virtual memory , processes and sharing in MULTICS
Communication of the ACM vol 11 n.5 mai 68 p 306

|25| DARONDEAU P. , GUIBOUD RIBAUD S. , OTRAGE C. , SMIT H.
Architecture de calculateur orientés vers les systèmes
RAIRO B2 74

|26| DENNIS Jack B. , HORN Earl C.
Programming semantics for multiprogrammed computation
Communication of the ACM mars 66

|27| DIJKSTRA E. W.
Solution of a problem in concurrent programming control
Communication of the ACM vol 8 n.9 septembre 65 p 569

|28| DIJKSTRA E. W.
Cooperating sequential processes
THE Eindhoven EWD 123 septembre 65

|29| DIJKSTRA E. W.
The structure of the THE multiprogramming system
Communication of the ACM vol 11 n.5 mai 68 p 341

|30| DIJKSTRA E. W.
Hierarchical ordering of sequential processes
THE Eindhoven EWD 310-0

|31| DOUSSY J.
Un premier pas vers une machine temps réel : Le T1600
Journées AFCET Machines Orientées Langage ,
Machines Orientées systèmes
Alpe d'Huez mai 74

|32| EASTON William B.
Process synchronisation without long term interlock
Third Symposium on Operating Systems Principles
Stanford University octobre 71

|33| FALKOFF A. D. , IVERSON K. E. , SUSSENGUTH E. H.
Formal description of system / 360
IBM System journal vol 13 64 p 198

|34| FLYNN Michael J.
Very high speed computing systems
Proceedings of the IEE vol 54 n.12 décembre 66 p 1901

|35| FORBES Bert E. , GREEN Michael D.
An economical full-scale multipurpose computer system
Hewlett-Packard Journal vol 24 n.5 janvier 73

|36| GAGLIARDI Ugo O. , GOLDBERG Robert P.
Virtualizeable architectures
Proceeding ACM AICA Venise 72

|37| GAUDET B.
Système a processus paralleles sur une petite machine
Séminaires IMAG janvier 72

|38| GAVARINI P.
Design et realisation d'une machine multiprogrammee de
moyenne puissance : Le HP 3000
Journees AFCET Machines Orientées Langage ,
Machines Orientées Systèmes
Alpe d'Huez mai 74

|39| GOUNTANIS R. J. , VISS N. L.
A method of processor selection for interrupt handling
in a multiprocessor system
Proceedings of the IEE vol 54 n.12 décembre 66 p 1812

|40| HABERMANN Arie N.
On the harmonious cooperation of abstract machines
Thèse THE Eindhoven octobre 67

|41| HOARE C. A. R.
Monitors : an operating system structuring concept
Séminaire IRIA 73

|42| HORNING J. J. , RANDELL B.
Process structuring
ACM Computing Survey vol 5 n.1 mars 73 p 5

|43| HUBERMAN B
Principles of operation of the VENUS Microprogram
MITRE Corporation contract n F 19 (628) 68 C 0365 mai 70

|44| HUSSON Samir S.
Microprogramming : Principles and Practices
Prentice Hall 70

|45| IBM
Documentation de l'ordinateur 360/25

|46| IBM
Documentation de l'ordinateur 370/125

|47| KAMRAN Said
Allocation de ressources dans un réseau d'ordinateurs
Thèse université Claude Bernard Lyon septembre 72

|48| KUNTZMANN J.
Théorie des réseaux
DUNOD Paris 72

|49| KURTZBERG Jerome M. , VILLANI Raymond D.
A balanced pipelining approach to multiprocessing on an instruction stream level
IEE transaction on computers vol C22 n.2 février 73 p 143

|50| LAMPSON B.
A scheduling philosophy for multiprocessing systems
Communication of the ACM vol 11 n.5 mai 68 p 347

|51| LAWSON Harold W. jr , MALM Bengt
A flexible asynchronous microprocessor
BIT n.13 73 p 165

|52| LAWSON Harold W. jr , SMITH Burton K.
Functional characteristics of a multilingual processor
IEE transaction on computers vol C20 n.7 juillet 71 p 73

|53| LEE John A. N.
Computer semantics
Van Nostrand Reinhold Company 72

|54| LESSER Victor R.
Dynamic control structures and their use in emulation
SLAC report n. CS 309 octobre 72

|55| LEVY John V.
Computing with multiple microprocessors
SLAC report n.161 avril 73

|56| LINDSAY Bruce
Suggestions for an extensible capability-based machine architecture
International workshop on computer architecture GRENOBLE juin 73

|69| ULLMANN P.
Cooperation entre taches et dispatching
EMP Fontainebleau janvier 71

|70| WEGNER P.
The Vienna definition language
ACM Computing Surveys mars 72

|71| WIRTH N.
On multiprogramming , machine coding and computer organization
Communication of the ACM vol 12 n.9 septembre 69 p 489

|57| MAC NAUGHTON P. C.
The virtual multiprocessor , a new computer architecture
Thèse Toronto 67

|58| MADSEN Ole Brun
BPL a hardware and software description language
RECAU Université d'AARHUS 72-15

|59| MITRANI E.
Réseau d'opérateurs adaptés au traitement du signal
Journées IRIA St Pierre de Chartreuse novembre 73 p 61

|60| NOGUEZ Gerard
L'itération : un mécanisme d'exécution
Thèse université Paris 6 mars 71

|61| ROHMER J.
Hierarchical systems . Simulation and design
IERE Joint Conference on Computers Londres novembre 72 p 375

|62| ROHMER J.
Etude et réalisation d'un simulateur de systemes hierarchisés
Rapport final contrat CRI 72.21/ 1-41 décembre 73

|63| ROSENFELD Jack L. , VILLANI Raymond D.
Micromultiprocessing : An approach to multiprocessing at the level of very small tasks
IEE transactions on computers vol C22 n.2 février 73 p 149

|64| SAAL H. J. , RIDDLE W. E.
Communicating semaphores
Stanford university CCTM 117 décembre 71

|65| SCHOELLKOPF J. P.
Microprogramming : A step of a top down design methodology
7 th Annual Workshop on Microprogramming Palo Alto octobre 74

|66| SCHROEDER Michael D. , SALTZER Jerome D.
A hardware architecture for implementing protection rings
Communication of the ACM mars 72 p 157

|67| THOMAS Robert H.
A model for process representation and synthesis
Thèse projet MAC MIT mai 71

|68| TOMASULO R. M.
An efficient algorithm for exploiting multiple arithmetic units
IBM Journal janvier 67 p 25