Mécanisme de base dans les systèmes superviseurs : Conception et réalisation d'un système à accès multiples

Jacques Bellino

THESE

présentée à

L'Université Scientifique et Médicale de Grenoble

pour obtenir le grade de

DOCTEUR ES-SCIENCES APPLIQUEES

par

Jacques BELLINO

MECANISMES DE BASE DANS LES SYSTEMES SUPERVISEURS : CONCEPTION ET REALISATION D'UN SYSTEME A ACCES MULTIPLES

Soutenue le 28 septembre 1973 devant la commission d'examen

Président	N. GASTINEL
Rapporteur	L. BOLLIET
Examinateurs	J. Cl. BOUSSARD
	M. GRIFFITHS
	S. KRAKOVIAK

UNIVERSITE SCIENTIFIQUE
ET MEDICALE DE GRENOBLE

LISTE DES PROFESSEURS

Président : Monsieur Michel SOUTIF
Vice-Président : Monsieur Gabriel CAU

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
	BENOIT Jean	Radioélectricité
	BERNARD Alain	Mathématiques Pures
	BESSON Jean	Electrochimie
	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
	BONNIER Etienne	Electrochimie Electrométallurgie
	BOUCHERLE André	Chimie et Toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BOUSSARD Jean-Claude	Mathématiques Appliquées
	BRAVARD Yves	Géographie
	BRISSONNEAU Pierre	Physique du solide
	BUYLE-BODIN Maurice	Electronique
	CABANAC Jean	Pathologie chirurgicale
	CABANEL Jean	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
	CHENE Marcel	Chimie papetière
	COEUR André	Pharmacie chimique
	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
	DESRE Pierre	Métallurgie
	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
	FELICI Noël	Electrostatique
	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques Pures
	GALVANI Octave	Mathématiques Pures
	GASTINEL Noël	Analyse numérique
	GEINDRE Michel	Electroradiologie
	GERBER Robert	Mathématiques Pures
	GIRAUD Pierre	Géologie
	KLEIN Joseph	Mathématiques Pures
Mme	KOFLER Lucie	Botanique et Physilogie végétale
MM.	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	Biochimie pharmaceutique
	LAURENT Pierre-Jean	Mathématiques appliquées
	LEDRU Jean	Clinique médicale B
	LLIBOUTRY Louis	Géophysique
	LOUP Jean	Géographie
Mle	LUTZ Elisabeth	Mathématiques Pures
MM.	MALGRANGE Bernard	Mathématiques Pures
	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Seméiqlogie médicale
	MASSEPORT Jean	Géographie
	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
	PAUTHENET René	Electrotechnique
	PAYAN Jean-Jacques	Mathématiques Pures
	PEBAY-PEYROULA Jean-Claude	Physique
	PERRET René	Servomécanismes
	PILLET Emile	Physique industrielle
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	REULOS René	Physique industrielle
	RINALDI Renaud	Physique
	ROGET Jean	Clinique de pédiatrie et de puériculture
	SANTON Lucien	Mécanique
	SEIGNEURIN Raymond	Microbiologie et Hygiène
	SENGEL Philippe	Zoologie
	SILBERT Robert	Mécanique des fluides
	SOUTIF Michel	Physique générale

MM.	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLAND François	Zoologie
	VALENTIN Jacques	Physique nucléaire
	VAUQUOIS Bernard	Calcul électronique
Mme	VERAIN Alice	Pharmacie galénique
M.	VERAIN André	Physique
Mme	VEYRET Germaine	Géographie
MM.	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale
	YOCCOZ Jean	Physique nucléaire théorique

PROFESSEURS ASSOCIES

MM.	BULLEMER Bernhard	Physique
	HANO JUN-ICHI	Mathématiques Pures
	STEPHENS Michaël	Mathématiques appliquées

PROFESSEURS SANS CHAIRE

MM.	BEAUDOING André	Pédiatrie
Mme	BERTRANDIAS Françoise	Mathématiques Pures
MM.	BERTRANDIAS Jean-Paul	Mathématiques appliquées
	BIAREZ Jean-Pierre	Mécanique
	BONNETAIN Lucien	Chimie minérale
Mme	BONNIER Jane	Chimie générale
MM.	CARLIER Georges	Biologie végétale
	COHEN Joseph	Electrotechnique
	COUMES André	Radioélectricité
	DEPASSEL Roger	Mécanique des fluides
	DEPORTES Charles	Chimie minérale
	GAUTHIER Yves	Sciences biologiques
	GAVEND Michel	Pharmacologie
	GERMAIN Jean-Pierre	Mécanique
	GIDON Paul	Géologie et Minéralogie
	GLENAT René	Chimie organique
	HACQUES Gérard	Calcul numérique
	JANIN Bernard	Géographie
Mme	KAHANE Josette	Physique
MM.	MULLER Jean-Michel	Thérapeutique
	PERRIAUX Jean-Jacques	Géologie et Minéralogie
	POULOUJADOFF Michel	Electrotechnique
	REBECQ Jacques	Biologie (CUS)
	REVOL Michel	Urologie
	REYMOND Jean-Charles	Chirurgie générale
	ROBERT André	Chimie papetière
	DE ROUGEMONT Jacques	Neurochirurgie
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIBILLE Robert	Construction mécanique
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

Mle	AGNIUS-DELORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	AMBLARD Pierre	Dermatologie
	AMBROISE-THOMAS Pierre	Parasitologie
	ARMAND Yves	Chimie
	BEGUIN Claude	Chimie organique
	BELORIZKY Elie	Physique
	BENZAKEN Claude	Mathématiques appliquées
	BILLET Jean	Géographie
	BLIMAN Samuel	Electronique (EIE)
	BLOCH Daniel	Electrotechnique
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BOUCHET Yves	Anatomie
	BOUVARD Maurice	Mécanique des fluides
	BRODEAU François	Mathématiques (IUT B)
	BRUGEL Lucien	Energétique
	BUISSON Roger	Physique
	BUTEL Jean	Orthopédie
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHIAVERINA Jean	Biologie appliquée (EFP)
	CHIBON Pierre	Biologie animale
	COHEN-ADDAD Jean-Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie médicale
	CONTE René	Physique
	COULOMB Max	Radiologie
	CROUZET Guy	Radiologie
	DURAND Francis	Métallurgie
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	GENSAC Pierre	Botanique
	GIDON Maurice	Géologie
	GRIFFITHS Michaël	Mathématiques appliquées
	GROULADE Joseph	Biochimie médicale
	HOLLARD Daniel	Hématologie
	HUGONOT Robert	Hygiène et Médecine préventive
	IDELMAN Simon	Physiologie animale
	IVANES Marcel	Electricité
	JALBERT Pierre	Histologie
	JOLY Jean-René	Mathématiques Pures
	JOUBERT Jean-Claude	Physique du solide
	JULLIEN Pierre	Mathématiques Pures
	KAHANE André	Physique générale
	KUHN Gérard	Physique
	LACOUME Jean-Louis	Physique
Mme	LAJZEROWICZ Jeannine	Physique
MM.	LANCIA Roland	Physique atomique
	LE JUNTER Noël	Electronique
	LEROY Philippe	Mathématiques
	LOISEAUX Jean-Marie	Physique nucléaire
	LONGEQUEUE Jean-Pierre	Physique nucléaire
	LUU DUC Cuong	Chimie organique
	MACHE Régis	Physiologie végétale
	MAGNIN Robert	Hygiène et Médecine préventive
	MARECHAL Jean	Mécanique
	MARTIN-BOUYER Michel	Chimie (CUS)

MM.	MAYNARD Roger	Physique du solide
	MICHOULIER Jean	Physique (IUT A)
	MICOUD Max	Maladies infectieuses
	MOREAU René	Hydraulique (INP)
	NEGRE Robert	Mécanique
	PARAMELLE Bernard	Pneumologie
	PECCOUD François	Analyse (IUT B)
	PEFFEN René	Métallurgie
	PELMONT Jean	Physiologie animale
	PERRET Jean	Neurologie
	PERRIN Louis	Pathologie expérimentale
	PFISTER Jean-Claude	Physique du solide
	PHELIP Xavier	Rhumatologie
Mlle	RIERY Yvette	Biologie animale
MM.	RACHAIL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RENAUD Maurice	Chimie
	RICHARD Lucien	Botanique
Mme	RINAUDO Marquerite	Chimie macromoléculaire
MM.	ROMIER Guy	Mathématiques (IUT B)
	SHOM Jean-Claude	Chimie générale
	STIEGLITZ Paul	Anesthésiologie
	STOEBNER Pierre	Anatomie pathologique
	VAN CUTSEM Bernard	Mathématiques appliquées
	VEILLON Gérard	Mathématiques appliquées (INP)
	VIALON Pierre	Géologie
	VOOG Robert	Médecine interne
	VROUSSOS Constantin	Radiologie
	ZADWORNY François	Electronique

MAITRES DE CONFERENCES ASSOCIES

MM.	BOUDOURIS Georges	Radioélectricité
	CHEEKE John	Thermodynamique
	GOLDSCHMIDT Hubert	Mathématiques
	SIDNEY STUARD	Mathématiques Pures
	YACOUD Mahmoud	Médecine légale

CHARGES DE FONCTIONS DE MAITRES DE CONFERENCES

Mme	BERIEL Hélène	Physilogie
Mme	RENAUDET Jacqueline	Microbiologie

Fait le 30 mai 1972.

Je tiens à exprimer toute ma gratitude

A Monsieur le Professeur Noël GASTINEL, Directeur du Centre Interuniversitaire de Calcul de Grenoble, qui a bien voulu me faire l'honneur de présider le jury de cette thèse et qui n'a jamais ménagé son temps pour me prodiguer ses conseils et ses critiques.

A Monsieur le Professeur Louis BOLLIET, qui guida mes débuts en informatique et qui eut la clairvoyance, il y a plusieurs années déjà, d'aiguiller une équipe de chercheurs dont j'étais, vers le sujet encore peu exploré des systèmes en temps partagé.

A Monsieur Jean-Claude BOUSSARD, Professeur à l'Université de Nice, et à Monsieur Michaël GRIFFITHS, Maître de Conférences à l'Université Scientifique et Médicale de Grenoble, dont les encouragements m'ont incité à rédiger cette thèse et qui ont accepté de juger mon travail.

A Monsieur Sacha KRAKOWIAK, dont j'ai pu, en de nombreuses occasions, apprécier la compétence dans les domaines traités dans cette thèse et qui m'a fait l'amitié de participer au jury.

La recherche dans le domaine des systèmes d'exploitation, lorsqu'elle s'appuie sur des réalisations concrètes, est forcément un travail d'équipe. J'ai plaisir à remercier ici tous ceux avec qui j'ai eu l'occasion de travailler dans le cadre des divers projets de recherche auxquels j'ai participé.

Je citerai plus particulièrement :

Monsieur Philippe POTIN, dont la collaboration active et efficace a été déterminante pour mener à bien les travaux rapportés dans cette thèse ;

les chercheurs du Laboratoire d'Informatique de l'ENSIMAG et mes collègues de travail qui ont participé au projet GMS : Messieurs Alain AUROUX, Jean-Claude COEZ, Christian DEVILLERS, Luc FINET, Claude HANS, Xavier de LAMBERTERIE, Pierre SAUVAGE ;

Monsieur Max PELTIER, qui, en tant que Directeur du Centre Scientifique IBM de Grenoble, a su, par son enthousiasme et sa compétence, créer les conditions propices à l'aboutissement de ces travaux.

Je remercie également la Compagnie IBM-France au sein de laquelle j'ai pu, dans d'excellentes conditions, poursuivre les travaux de recherche concrétisés par cette thèse.

Monsieur Maurice BELLOT, par ses suggestions et critiques, m'a été d'une aide précieuse dans la préparation finale du manuscrit ; qu'il en soit chaleureusement remercié.

Je ne saurais oublier le Service de Reproduction du Laboratoire, qui a assuré avec le soin habituel la réalisation matérielle de cet ouvrage.

MECANISMES DE BASE DANS LES SYSTEMES SUPERVISEURS :

CONCEPTION ET REALISATION D'UN SYSTEME A ACCES MULTIPLES

A mes parents,

A Marie-Jane

TABLE DES MATIERES

CHAPITRE 1

INTRODUCTION

L'étude présentée ici est centrée sur la gestion des activités parallèles dans les systèmes d'exploitation. Un système réalise un ensemble complexe de fonctions dans le but d'atteindre des objectifs fixés. Pour les systèmes auxquels cette étude s'applique, l'objectif premier est de permettre l'exécution de programmes soumis par des utilisateurs.

Lorsque l'on essaie d'analyser un système existant ou d'en concevoir un nouveau, une démarche naturelle consiste à découper l'activité de ce système en un certain nombre de fonctions plus ou moins indépendantes les unes des autres et à établir des relations logiques entre ces fonctions. Cependant, un découpage purement fonctionnel n'est pas suffisant pour représenter l'évolution dynamique du système. Chacune des activités de ce système nécessite, pour être menée à son terme, l'emploi d'un certain nombre de ressources. A un instant donné, les besoins en ressources, pour le système lui-même et pour les programmes des utilisateurs, sont en général supérieurs aux ressources de l'installation et l'un des rôles du système est d'allouer les ressources aux divers demandeurs. Au niveau de la conception d'un système, nous pouvons nous placer dans le cas idéal où chacune des activités indépendantes dégagées disposerait de toutes les ressources nécessaires à son accomplissement et étudier l'évolution simultanée de ces diverses activités. La notion

fondamentale qui apparaît alors est celle de parallélisme, au sens macroscopique du terme. Nous pouvons observer deux types de parallélisme.

- Parallélisme technologique : par construction, certains organes de la machine peuvent fonctionner en simultanéité réelle. C'est le cas par exemple pour le processeur central et les canaux d'entrée-sortie.
- Parallélisme logique : dans un système pratiquant la multiprogrammation, deux programmes soumis par deux utilisateurs différents s'exécutent chacun de façon séquentielle, indépendamment l'un de l'autre. Il n'existe aucune interaction entre ces deux programmes dans l'hypothèse où chacun dispose des ressources nécessaires à son exécution.

  Ce parallélisme logique s'observe également pour un grand nombre des activités du système lui-même, par exemple entre la fonction de préparation de la file d'attente des travaux à exécuter et la fonction d'initialisation ou de supervision de ces travaux. Cependant, dans ce dernier exemple, ces deux activités bien que conceptuellemnt indépendantes nécessitent, en certains points de leur exécution, d'être ordonnées l'une par rapport à l'autre et supposent également le partage ou la communication d'informations (la file d'attente créée par l'une est exploitée par l'autre). La gestion de telles activités parallèles qui contribuent à la réalisation d'une tâche donnée, ici l'exécution des travaux, implique alors l'installation de mécanismes pour synchroniser ces activités et assurer la communication d'informations entre elles.

Dans la suite de ce texte, nous appellerons processus la mise

en oeuvre d'une activité particulière. A la différence de la notion de programme qui ne traduit que la description statique d'un algorithme exécutable par un processeur, la notion de processus est dynamique et représente l'exécution de cet algorithme dans le temps. A la notion de processus sont attachés l'algorithme du processus, les données sur lesquelles il travaille et les ressources nécessaires à son exécution.

Le déroulement d'un processus peut être suspendu pour deux raisons bien distinctes.

1- L'une au moins des ressources nécessaires à son exécution n'est pas disponible, bien qu'il n'existe pas une demande explicite pour cette ressource dans l'algorithme du processus. Il s'agit de l'une des ressources composant la machine logique dans laquelle s'exécute le processus. La détection de cette situation est à la charge d'une partie du système que nous appelons l'allocateur de ressources. Par exemple, lorsque cet allocateur décide de retirer le processeur à un processus, ce dernier est suspendu sans que cet état de fait n'apparaisse dans son algorithme. Il en est de même lorsqu'un processus disposant d'une mémoire virtuelle paginée accède à une page virtuelle non présente en mémoire centrale.

   Nous disons que nous sommes en présence d'un blocage technologique du processus [Saltzer 66].

2- Le processus attend, pour poursuivre son exécution, un signal en provenance d'un autre processus. Cette interaction est explicitement programmée et fait partie de l'algorithme du processus. Par exemple, le processus attend un message créé par un autre processus.

Nous disons alors que nous sommes en présence d'un blocage intrinsèque du processus [Saltzer 66].

De ce qui précède, nous pouvons déduire deux classes de problèmes rencontrés lors de la conception d'un système.

- L'allocation des ressources aux machines logiques dans lesquelles s'exécutent les processus ; cette fonction est mise en oeuvre par des techniques d'allocation.
- L'ordonnancement des activités des divers processus composant le système pour respecter les interactions intrinsèques de ces processus ; cet ordonnancement est mis en oeuvre par des mécanismes de synchronisation.

Nous étudions ces deux types de problèmes au chapitre 2 en exposant les solutions adoptées pour la construction d'un système particulier, le système GMS.

Nous ne prétendons pas proposer, à travers la présentation des choix effectués pour la construction de ce système, une règle générale pour l'élaboration de tout système ultérieur. En effet, il ne peut exister deux systèmes identiques si l'on considère la variété des objectifs visés et des contraintes de réalisation. De plus, même dans l'hypothèse où ces objectifs et ces contraintes seraient semblables, il apparaitrait toujours des différences dans la structure des systèmes produits, ne serait-ce que par la personnalité, l'expérience passée ou simplement les goûts personnels des concepteurs. Nous pensons cependant que l'exposé des méthodes employées pour bâtir un système réel, l'énoncé des choix et des compromis effectués, leur justification ainsi qu'une analyse critique a posteriori des décisions essentielles sont en mesure de contribuer à une approche rationnelle lors d'expériences

ultérieures.

Le thème central abordé ici est celui du parallélisme et de l'allocation des ressources. Nous n'avons pourtant pas voulu isoler ces problèmes de leur contexte réel, mais au contraire montrer comment les outils développés pour les résoudre sont utilisés pour réaliser de façon concrète les fonctions essentielles d'un système. Il apparaît en effet que la construction d'un système opérationnel en vue de sa mise à la disposition d'utilisateurs est la meilleure manière de vérifier la validité des principes mis en avant lors de la conception.

Pour approfondir la mise en application de ces principes, nous faisons porter l'étude sur le seul système GMS, le dernier en date à la construction duquel nous ayons participé. Il est clair cependant que les résultats d'autres expériences menées antérieurement ont influencé bon nombre de décisions prises lors de la conception ou de la réalisation de GMS. On trouvera en particulier dans les publications concernant les systèmes DIAMAG 1 et DIAMAG 2 [Auroux 66, Bolliet 67, Bellot 68, Verjus 68] une ébauche des solutions adoptées dans GMS concernant l'aspect externe du système tel qu'il apparaît à un utilisateur et sa structuration en fonctions indépendantes. Plus récemment, la construction du système LCMS [Bellino 69, Bellino 70 a] nous a conduit à mettre l'accent sur les problèmes liés à l'allocation des ressources et à la gestion d'activités parallèles dans un système à accès multiples.

Le système GMS [Bellino 72, Bellino 73, Finet 73 b, Potin 72] se prête bien à l'étude des problèmes qui nous occupent. Le point de départ en est un "système" existant, CMS [IBM 69 a], conçu pour

permettre à un utilisateur unique disposant d'une machine complète, de créer et d'exécuter des programmes. CMS fournit à cet utilisateur un ensemble de composants tels que des compilateurs et assembleurs ou des programmes de gestion pour ses fichiers. Un ensemble de commandes est ainsi mis à la disposition de cet utilisateur unique.

Nous pouvons considérer l'activité d'un tel utilisateur comme un processus séquentiel. Partant de cette base qu'est CMS, nous voulons autoriser, sur une machine donnée (un IBM/360-40), l'emploi simultané, par plusieurs utilisateurs, des facilités ainsi offertes.

Le système à créer est destiné à une utilisation effective et l'un des critères pour évaluer sa qualité est son efficacité. Nous verrons, tout au long de l'étude qui suit, que le souci d'efficacité est un paramètre fondamental de ce projet : il entre pour une bonne partie dans la plupart des décisions prises lors de la construction du système.

Nous adoptons pour cette étude le plan suivant.

- Les paragraphes 2.1 et 2.2 sont consacrés à la définition des objectifs et au choix d'une structure globale pour le système à construire. Nous présentons les deux solutions qui consistent, l'une à modifier CMS lui-même pour le rendre multi-utilisateurs, l'autre à placer au dessus de CMS un hyperviseur chargé de générer et de gérer plusieurs machines virtuelles dans lesquelles peuvent s'exécuter des systèmes CMS non modifiés.

- Ayant retenu la seconde solution, nous introduisons au paragraphe 2.3 la notion de processus dans GMS. Nous montrons comment l'ensemble des activités de l'hyperviseur est découpé en

processus, comment ces processus coopèrent et quels sont les mécanismes de synchronisation et de communication employés pour mettre en oeuvre cette coopération.

Nous présentons ensuite les problèmes liés à la réalisation et nous explicitons les mécanismes mis en place pour la gestion de ces processus. Nous examinons la manière dont ces processus sont décrits dans le système et l'implantation des primitives de synchronisation. Un point intéressant concerne l'utilisation qui est faite du langage dans lequel est écrit le système pour implanter la gestion des processus.

- Les problèmes posés par la gestion des ressources physiques de l'installation sont étudiés au paragraphe 2.4. Il ne s'agit pas là de développer des stratégies d'allocation pour ces ressources mais plutôt de montrer comment la notion de processus, précédemment introduite, est appliquée pour réaliser les mécanismes d'allocation et pour effectuer les opérations nécessitant l'utilisation de ces ressources.

L'étude de la gestion de la mémoire libre, pour les besoins de GMS lui-même, permet de préciser les relations entre l'allocateur et les processus demandeurs de mémoire.

L'étude de la gestion des opérations d'entrée-sortie montre comment les mécanismes de synchronisation sont employés pour traduire le parallélisme existant entre le déroulement d'une opération d'entrée-sortie et le processus qui demande l'exécution de cette opération.

Nous nous plaçons ensuite à un niveau plus général et nous définissons la structure des processus chargés de la gestion d'une unité d'entrée-sortie donnée. L'exemple de l'imprimante nous conduit à remettre en question le découpage initial du système en

processus et à envisager diverses solutions possibles.

- Nous avons défini la structure du système, les outils utilisés pour sa construction, la manière de s'en servir. Nous pouvons maintenant aborder le problème posé par l'exécution parallèle de plusieurs systèmes CMS, qui est l'objectif initial que nous nous étions fixé.

Nous développons au paragraphe 2.5 l'allocation des ressources au processus de chacun des utilisateurs et nous décrivons les processus de l'hyperviseur chargés de piloter l'utilisation de ces ressources.

Pour donner une vue d'ensemble du système GMS, nous précisons au paragraphe 2.6 les moyens de communication entre l'utilisateur et ce système.

Nous consacrons le paragraphe 2.7 à l'étude de problèmes spécifiques à cette réalisation. Nous étudions, en particulier, la stratégie d'allocation du processeur développée en tenant compte des contraintes propres à notre application.

Pour terminer, nous tentons dans le chapitre 3 de dégager les conclusions de cette expérience. Nous effectuons une analyse critique des solutions adoptées et nous proposons d'autres alternatives possibles.

Le projet GMS a été réalisé par une équipe composée de Luc Finet, Claude Hans, Philippe Potin et nous-même. Il a d'autre part servi de support au rapport de DEA de Luc Finet et Jacques Leroudier et aux projets de fin d'études de Jean-Paul Faure et Patrick Servant et de Jean-Pierre Ansart et Christian Devillers. Max Peltier a participé activement à la phase de conception. La

réalisation de la partie gestion de fichiers doit beaucoup à Xavier de Lamberterie et à Pierre Sauvage. La réalisation de l'environnement pseudo-terminal est l'oeuvre de Christian Devillers.

CHAPITRE 2

LE SYSTEME GMS

Le présent chapitre porte sur l'étude du système GMS [Bellino 72, Bellino 73 , Finet 73 b, Potin 72] réalisé au Centre Scientifique IBM de Grenoble pour permettre l'accès multiple au système mono-conversationnel CMS [IBM 69 a] sur un IBM/360 standard.

2.1 INTRODUCTION

Le système CMS (Cambridge Monitor System), qui sert de point de départ au projet, peut être utilisé sur une machine IBM/360 modèle 40 et au dessus disposant de 256 K octets de mémoire principale et des unités périphériques suivantes (figure 1) :

- un lecteur-perforateur de cartes,
- une imprimante,
- deux unités de disques, l'une servant de lieu de résidence aux composants de CMS (disque Système) et l'autre permettant à l'utilisateur de conserver ses fichiers (disque Privé),
- une console (la console de l'opérateur) qui est l'organe de communication entre l'utilisateur et le système,
- deux unités de bandes magnétiques (facultatives).

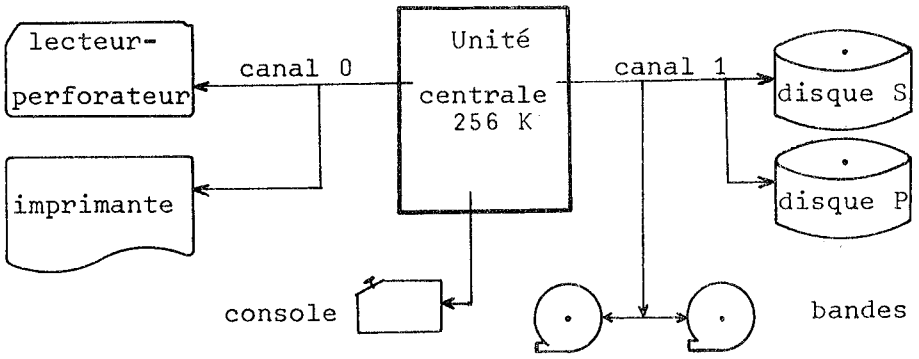


Figure 1. Configuration de base pour CMS

Dans cette version, CMS permet à un utilisateur unique de créer et d'exécuter des programmes en interaction directe avec le système. La contrepartie évidente de ces facilités d'interaction est que l'installation complète est monopolisée par un seul utilisateur. L'examen du comportement des programmeurs développant sous CMS des programmes d'application ou des composants de systèmes est significative : sur un 360-40, le système est en moyenne en attente d'une entrée-sortie sur la console pendant 85% du temps. Ce temps correspond aux périodes de réflexion du programmeur, au temps de frappe des lignes et au temps de sortie des messages. On peut alors raisonnablement penser qu'en utilisant ces temps d'attente, une telle machine est capable de supporter dans de bonnes conditions la charge due à environ 6 utilisateurs de ce type.

2.2 APPROCHES ENVISAGEABLES - CHOIX D'UNE SOLUTION

Deux solutions sont possibles pour permettre l'accès multiple au système CMS.

2.21 Inclusion dans CMS des mécanismes de l'accès multiple

La mise en place des mécanismes de l'accès multiple par modification du système CMS lui-même consiste à isoler dans CMS, si cela est possible, les fonctions d'accès aux ressources physiques (imprimante, console, ...) et aux ressources logiques (accès à un fichier, accès à un composant du système, ...) et à modifier ces fonctions d'accès pour tenir compte du partage. Cette solution permettrait d'avoir une copie unique du système partagée par les divers utilisateurs (mono-système/multi-utilisateurs) mais elle suppose l'existence d'un interface utilisateur-système rigoureusement défini et une protection du niveau système contre le niveau utilisateur (figure 2).

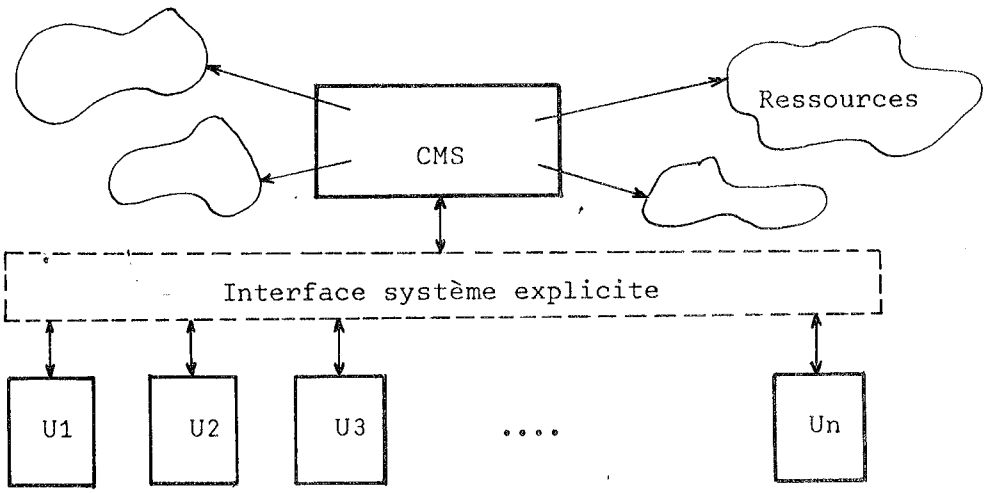


Figure 2. Un seul système multi utilisateurs

Dans CMS, les deux conditions précédentes ne sont pas réalisées : les deux niveaux d'accès aux ressources n'apparaissent pas et tous les programmes, qu'ils soient de l'utilisateur ou du système, s'exécutent avec les mêmes privilèges et ont accès sans contrôle à l'ensemble des ressources de l'installation. En particulier, tout programme peut s'exécuter en mode superviseur et a ainsi la possibilité de réaliser lui-même ses opérations d'entrée-sortie, de lire ou d'écrire dans n'importe quelle partie de la mémoire centrale ou encore de placer le processeur en état d'attente.

2.22 Emploi d'un interface de type machine virtuelle

Connaissant la configuration du 360 pour laquelle CMS a été écrit, l'accès multiple devient possible si l'on est en mesure, à partir des ressources réelles de l'installation, de générer plusieurs machines virtuelles ayant cette configuration. Cela revient à placer au dessus de CMS et non plus à l'intérieur les mécanismes de partage des ressources et à ne considérer que les ressources physiques [Goldberg 73]. Nous mettons ainsi en parallèle plusieurs systèmes CMS servant chacun un seul utilisateur (figure 3). Nous appelons hyperviseur le système chargé de générer des machines virtuelles (il peut "superviser" l'exécution de systèmes superviseurs).

Pour l'hyperviseur, les accès aux ressources physiques effectués par CMS sont des accès à des ressources virtuelles. Les ressources réelles, propriété de l'hyperviseur, sont utilisées par ce dernier pour créer des images de ces ressources virtuelles.

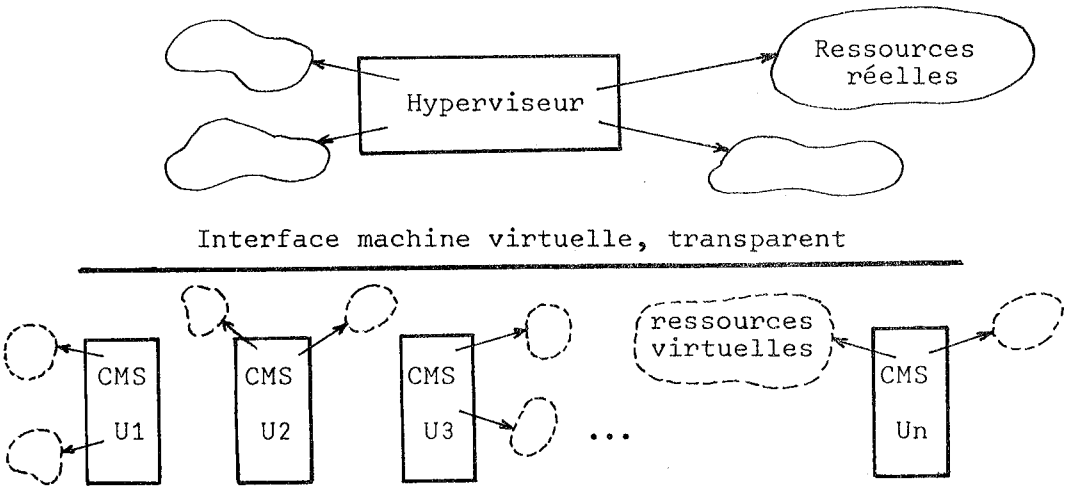


Figure 3. Plusieurs systèmes mono utilisateur

Cette deuxième solution est a priori plus tentante que la précédente du point de vue de sa réalisation. En effet, elle ne nécessite, en principe, aucune modification de CMS et est indépendante de la structure et du mode de fonctionnement de ce dernier. Toute modification ultérieure, aussi profonde soit-elle, apportée à la logique interne de CMS (lors d'un changement de version de CMS par exemple) n'entraîne aucune modification de l'hyperviseur, aussi longtemps que la configuration de base nécessaire à CMS reste la même.

Ce type d'accès multiple à CMS est celui réalisé sur IBM/360-67 au moyen du système générateur de machines virtuelles CP/67 [IBM 69 b].

2.23 Contraintes technologiques

La solution qui consiste à générer à partir d'un IBM/360 standard (autre qu'un modèle 67) des machines virtuelles "pures", c'est à dire fonctionnellement équivalentes à des IBM/360, n'est possible qu'à condition de recourir à l'interprétation des

programmes exécutés par ces machines virtuelles. Cela est dû, en particulier, au fonctionnement des dispositifs de protection de la mémoire qui ne permettent pas toujours de contrôler les accès à certaines informations (mots d'état en mémoire basse, horloge, ...). On trouvera dans [Goldberg 69, Bellino 73] une étude plus détaillée de ces problèmes.

2.24 Solution système virtuel

On remarque que l'approche machine virtuelle sur un 360 standard, tout en étant peu réaliste en raison des faibles performances qu'entraînerait l'interprétation des programmes, est en fait trop générale pour le cas qui nous occupe. En effet, à la différence du système CP/67 qui génère des machines virtuelles dans lesquelles doit pouvoir s'exécuter n'importe quel système d'exploitation, GMS doit seulement générer des "machines virtuelles" ayant toutes la même configuration (celle nécessaire à CMS) et dans lesquelles le système CMS puisse s'exécuter.

Nous avons choisi de modifier CMS de manière à ce qu'il n'effectue pas d'opérations non contrôlables par les "machines virtuelles dégénérées" (non interprétatives) que nous pouvons créer à partir d'un 360 standard. Ces modifications sont en fait mineures : elles consistent simplement à modifier les instructions qui référencent la partie basse de la mémoire, zone contenant les informations d'état de la machine réelle, sans signification pour une machine virtuelle. Après modification, la zone de mémoire référencée par de telles instructions est une image de la zone initiale. L'un des rôles de l'hyperviseur est d'assurer la validité des informations contenues dans cette zone image lorsqu'il active une machine virtuelle donnée.

Le système CMS modifié est obtenu automatiquement à partir du système CMS initial, sans modification de ses programmes source : tous les modules de CMS sont assemblés en utilisant un assembleur spécial qui détecte les références à la mémoire basse et applique lui-même la translation nécessaire.

A partir du moment où toute la généralité des machines virtuelles n'est plus nécessaire, il est possible d'affiner la solution retenue en mieux adaptant ces "machines" au système particulier pour lequel elles sont créées. Cette adaptation consiste à reconnaître dynamiquement l'accès à certaines ressources à un niveau logique et non plus au niveau de la ressource physique elle-même (rappelons que, vue de l'hyperviseur, cette ressource physique est virtuelle). Par exemple, si l'hyperviseur peut intercepter un appel au module de CMS qui lit une ligne sur la console de l'opérateur, il peut réaliser directement la simulation de la fonction "lire une ligne" en utilisant la ressource image de la console de l'opérateur (un terminal), plutôt que de simuler pas à pas la suite des opérations physiques exécutées par ce module. La simulation devient alors une simulation fonctionnelle ou macroscopique, par opposition à la simulation technologique. Nous appelons système virtuel une machine virtuelle de ce type, adaptée à un système particulier.

Nous avons utilisé la notion ci-dessus pour la construction de GMS. Nous dirons par la suite que nous générons des CMS virtuels.

2.25 Choix fondamentaux dans la conception de GMS

2.251 Structure interne

Les fonctions de l'hyperviseur GMS peuvent être classées en

deux catégories.

1) Fonctions permettant de fournir à chaque CMS virtuel les ressources qu'il utilise : ces fonctions sont assurées par l'interface générateur des CMS virtuels.
2) Fonctions de gestion des ressources physiques, initialisées par des demandes en provenance de l'interface ou en provenance d'autres parties de l'hyperviseur. Ces fonctions sont assurées par le noyau de l'hyperviseur.

La totalité des ressources physiques est sous le contrôle du noyau. Tout accès à une ressource physique effectué par CMS (par exemple l'initialisation d'une opération d'entrée-sortie sur l'imprimante) est traduit par l'interface générateur et peut donner lieu à une requête élémentaire pour la ressource physique image. Cette requête est soumise au noyau dont le rôle principal est de gérer les files d'attente pour chaque ressource physique.

La configuration machine sur laquelle GMS peut s'exécuter est représentée par la figure 4 ; elle correspond à la configuration de base nécessaire à CMS (figure 1) à laquelle ont été ajoutés une unité de contrôle de transmissions, des terminaux et des unités de disques supplémentaires.

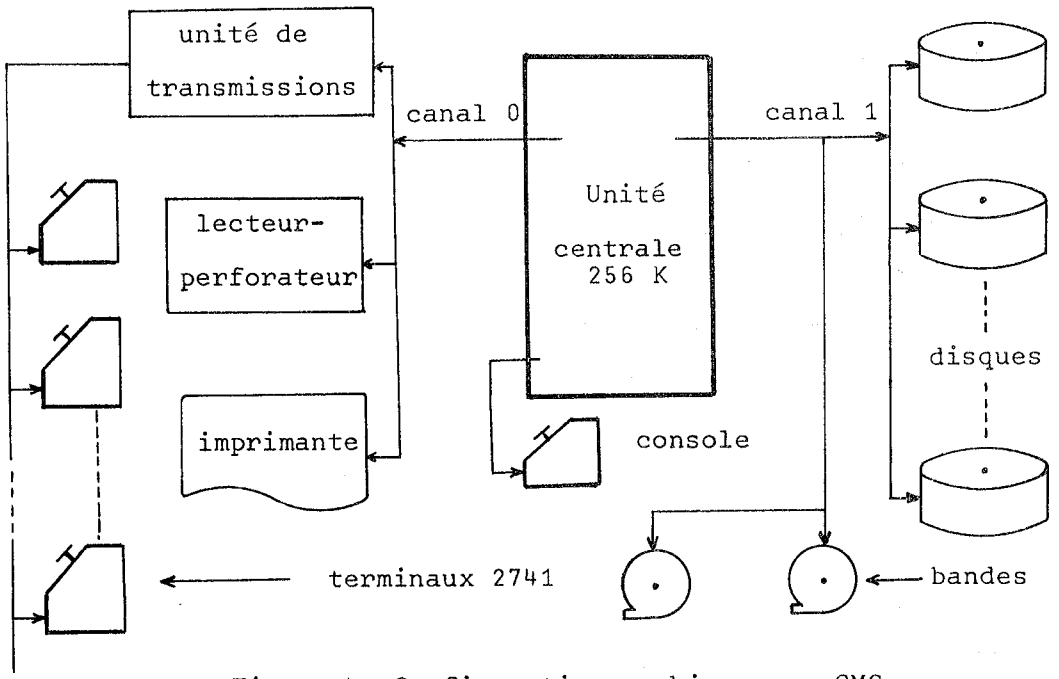


Figure 4. Configuration machine pour GMS

2.252 Mode de fonctionnement

Nous avons choisi de faire exécuter l'ensemble des fonctions de GMS en mode superviseur et toutes interruptions d'entrée-sortie interdites. Les CMS virtuels s'exécutent avec le processeur réel en mode programme et toutes interruptions permises, leur processeur virtuel pouvant, naturellement, être dans un autre état.

Les entrées dans l'hyperviseur résultent exclusivement d'interruptions : par définition, CMS ignore la présence de l'hyperviseur et n'émet donc pas d'appels explicites vers ce dernier.

En faisant fonctionner GMS avec interruptions interdites, nous renonçons à établir des priorités entre les événements qu'il contrôle. Nous obéissons alors aux priorités technologiques (priorités des interruptions les unes par rapport aux autres

lorsque plusieurs conditions d'interruption sont présentes simultanément).

Une autre alternative consistait à accepter les interruptions dans l'ensemble des fonctions de GMS sauf dans un noyau minimum chargé uniquement de placer ces interruptions dans des files d'attente (méthode utilisée par le système TSS notamment [IBM 69 c]). Cette dernière solution, associée à des règles de priorité, permet, soit d'interrompre l'exécution d'une fonction par l'arrivée d'un événement plus prioritaire, soit plus généralement de connaître, après avoir exécuté une fonction, tous les événements survenus pendant cette exécution et donc de choisir celui qui doit être traité le premier.

La technique de création de files d'attente avec exécution des fonctions en mode interruptible est utile lorsqu'il existe dans le système des événements associés à des unités ou à des fonctions qui exigent d'être pris en compte rapidement. Par exemple, la présence dans la configuration d'un tambour rapide implique souvent que la gestion de ce tambour soit prioritaire sous peine d'une dégradation de ses performances. De même, des fonctions telles que celles liées aux opérations de pagination nécessitent une priorité de traitement puisqu'elles conditionnent le plus souvent la possibilité d'activation des processus.

Dans notre cas, de telles particularités ne se présentent pas. De plus, les fonctions de GMS initialisées sur interruption nécessitent peu de temps de processeur. Ces deux aspects montrent qu'il n'est pas souhaitable d'exécuter, pendant le déroulement de ces fonctions, des opérations de changement de contexte qui elles sont généralement coûteuses.

Le choix qui consiste à faire exécuter toutes les fonctions de GMS en mode superviseur, c'est à dire avec les privilèges maxima,

se justifie par le fait que le nombre de modules du système est limité, les fonctions à réaliser sont bien définies et le travail est accompli par une équipe réduite. Ceci permet d'établir et de contrôler un certain nombre de conventions d'écriture des fonctions de GMS, ce qui n'exclut pas des contrôles dynamiques lors de l'exécution des modules du système.

L'hyperviseur consomme évidemment des ressources. Un choix fondamental est la manière dont ces ressources lui sont allouées : il est prioritaire pour l'utilisation du processeur par rapport aux CMS virtuels et la partie de la mémoire centrale dans laquelle il réside lui est affectée en permanence. Une autre partie de la mémoire lui appartient également qu'il gère à sa convenance pour des informations temporaires. Enfin, GMS dispose de façon permanente des ressources d'entrée-sortie non attribuées aux CMS virtuels, à savoir la console de l'opérateur, l'imprimante, le lecteur-perforateur de cartes et une unité de disque (disque système GMS).

2.253 Organisation des activités dans GMS

Toutes les activités à l'intérieur de GMS sont organisées en processus séquentiels. Nous ne tenterons pas de donner ici une définition générale d'un processus. Disons simplement que la notion de processus se précise d'elle-même lorsque l'on envisage la réquisition d'un processeur occupé à exécuter le programme correspondant à un algorithme donné opérant dans un environnement donné. Dire que l'activité interrompue par cette réquisition est un processus revient simplement à reconnaître qu'à l'exécution de ce programme sont associés les états de certains organes de la

machine (le mot d'état programme, les registres généraux, la mémoire elle-même, ...) et qu'il faut être capable de préserver ces informations - le vecteur d'état - si l'on veut plus tard reprendre l'exécution du processus à partir du point où il avait été arrêté.

En dehors des opérations liées à l'allocation des ressources, les actions du système sur un processus se traduisent par des opérations de création, destruction, mise en attente (blocage), réveil (déblocage) et activation.

Il n'y a pas contradiction entre le fait de définir des processus et celui de les faire s'exécuter sur un processeur avec interruptions interdites. En effet, nous verrons que dans GMS un processus peut de lui-même se mettre en attente d'un événement, ce qui revient à libérer le processeur. Remarquons simplement que l'exécution en mode non interruptible facilite l'implantation des mécanismes d'exclusion mutuelle lors de l'accès aux variables partagées.

Un point important de conception concerne la distinction qui est faite entre les processus de GMS et ceux correspondant à l'exécution des CMS virtuels (chaque CMS virtuel est un processus séquentiel). Il eut été possible, en considérant que dans le système ne s'exécutaient que des processus dotés de plus ou moins de prérogatives ("capabilities" [Dennis 66]), de prévoir un mécanisme unique de gestion des processus travaillant à partir d'une description paramétrée de ces processus. En fait, ce regroupement artificiel d'entités de nature aussi différentes ne peut que compliquer inutilement la gestion de ces processus. Au contraire, la séparation très nette entre les deux types d'activité du processeur permet de gérer chaque type de manière plus simple et plus efficace. En particulier, les ressources

nécessaires à l'exécution d'un processus sont distribuées et contrôlées par des mécanismes totalement différents selon qu'il s'agit d'un processus GMS ou d'un processus CMS. De même, les vecteurs d'état de ces processus ont des contenus non semblables : nous verrons que le vecteur d'état d'un processus GMS est réduit au strict minimum, alors qu'un CMS virtuel nécessite une description plus importante. Enfin, les mécanismes d'action d'un processus sur un autre, de coopération entre processus et de synchronisation sont très particularisés : pour un processus CMS, qui ignore l'existence des autres processus, les seuls mécanismes de synchronisation considérés concernent des événements technologiques, les interruptions, et ils sont mis en oeuvre par des opérations sur le mot d'état programme (chargement d'un masque d'interruptions, mise en attente du processeur, ...). GMS au contraire est construit comme un ensemble de processus coopérants qui peuvent agir les uns sur les autres et se synchroniser avec l'apparition d'événements technologiques ou programmés. Cette synchronisation est mise en oeuvre par l'emploi de primitives de synchronisation.

Dans la suite de ce chapitre, sauf mention contraire, chaque fois qu'il sera question de processus, il s'agira de processus GMS. Nous utiliserons de préférence le terme "CMS virtuel" pour désigner le processus associé à l'utilisation de CMS.

2.3 PROCESSUS ET SYNCHRONISATION

Dans ce paragraphe, nous définissons avec plus de précision ce qu'est, dans GMS, un processus, comment les processus sont implantés et quelles sont les primitives agissant sur les processus.

2.31 Découpage du système en processus

Dans une machine monoprocesseur, la notion de parallélisme réel entre processus n'existe pas. A un instant donné, un seul processus dispose du processeur. On peut cependant définir le parallélisme de deux processus de la manière suivante [Hansen 73] : deux processus sont parallèles si leur déroulement est susceptible de recouvrement ; en d'autres termes, si l'exécution d'un processus B peut démarrer avant qu'un processus A en cours d'exécution soit terminé, alors A et B sont parallèles.

Le découpage du système en processus présente une part d'arbitraire. A priori, il faut associer un processus à toute activité indépendante susceptible de s'exécuter en parallèle avec d'autres. C'est ainsi que les ressources physiques propriété de GMS à fonctionnement autonome, telles que l'imprimante ou le lecteur-perforateur de cartes, sont gérées chacune par un processus distinct. Dans ce cas, le découpage en processus parallèles résulte simplement du parallélisme technologique existant entre les unités contrôlées.

Certains processus de GMS sont créés par d'autres processus. C'est le cas du processus d'écriture d'un fichier sur l'imprimante réelle, créé par le processus qui détecte la "fermeture" d'une

imprimante virtuelle (cf. 2.431).

Plus généralement, chaque fois que, au cours du déroulement d'un processus apparaissent simultanément la possibilité de continuer à exécuter ce processus et la nécessité de déclencher une opération non immédiate (initialisation d'une entrée-sortie par exemple), alors il est intéressant de créer un nouveau processus chargé de contrôler cette opération non immédiate. Le processus initial peut ainsi continuer son exécution. Nous verrons dans la suite de ce chapitre un certain nombre d'exemples de création de processus parallèles correspondant à une telle situation.

Il est une autre catégorie de processus qui résultent directement d'une sollicitation en provenance d'un CMS virtuel : lorsqu'un CMS exécute une instruction privilégiée, cette instruction engendre une interruption qui redonne le contrôle à GMS. Un processus est ainsi activé qui contrôle la simulation de cette instruction privilégiée.

2.32 Coopération et communication entre processus

La réalisation d'une fonction particulière peut nécessiter l'action de plusieurs processus. Cela suppose que les processus coopérants ont une connaissance mutuelle les uns des autres, qu'ils peuvent partager entre eux certaines informations, qu'ils ont la possibilité de se communiquer de l'information et enfin qu'ils disposent de mécanismes pour synchroniser leur actions.

2.321 Connaissance mutuelle

Un processus agit sur un autre processus soit pour le créer

soit pour le réveiller. La création est assimilable à l'activation différée d'une procédure ; le processus créateur doit donc connaître le point d'entrée de cette procédure et la nature des paramètres à fournir. Le réveil d'un processus consiste à le placer dans la liste des processus prêts (cf. 2.3332). Le processus qui réveille doit alors seulement connaître le nom du processus à réveiller. Nous verrons que le "nom" d'un processus est simplement l'adresse en mémoire du descripteur du processus.

2.322 Partage d'informations

Un certain nombre d'informations du système sont partagées par plusieurs processus qui peuvent y accéder pour les consulter ou les modifier. Citons par exemple le catalogue des fichiers du système (il s'agit des fichiers de GMS créés en particulier pour simuler des ressources virtuelles telles que imprimante, lecteur de cartes, consoles, ...). Pour assurer la cohérence de telles informations, il faut inclure les opérations de modification dans des sections critiques. Cela est réalisé par l'emploi de primitives de verrouillage du type LOCK/UNLOCK qui garantissent à un processus l'accès exclusif à ces variables.

Par construction, la totalité des tables et variables de contrôle du système est accessible à l'ensemble des processus. Cela est dû au choix qui a été fait de placer tous les processus dans un même espace d'adressage. Cette solution suppose que les processus composant le système sont fiables. Elle permet, par contre, de résoudre de manière très simple les partages d'information, la seule condition de bon fonctionnement étant que les processus utilisent les mécanismes de verrouillage prévus (cf. 2.335).

2.323 Communication entre processus

En dehors des accès aux informations partagées, nous définissons dans GMS trois manières de communiquer des informations entre processus. Plutôt que de nous limiter à un type particulier de communication, nous pensons qu'il est préférable de prévoir des mécanismes divers adaptés chacun à des problèmes différents.

La première façon de communiquer de l'information à un processus consiste à lui fournir des paramètres. Nous verrons qu'un processus lorsqu'il prend le contrôle, qu'il soit créé ou réveillé, dispose d'un environnement constitué essentiellement par les registres du processeur. Le garnissage de ces registres est donc un moyen de transmettre de l'information à un processus. Un protocole doit alors exister entre les processus coopérants ; ce protocole est réduit à la connaissance des registres utilisables pour transmettre des paramètres et à la nature des paramètres transmis.

Une deuxième méthode de communication d'information utilise le principe des "boîtes aux lettres". Un processus voulant faire exécuter une action par un autre processus place une information dans une boîte aux lettres partagée par les deux processus. Si le processus récepteur est actif (en train de traiter une information placée précédemment dans cette boîte aux lettres) il viendra de lui-même rechercher et traiter ce nouveau message lorsqu'il aura traité tous ceux qui le précèdent dans la boîte aux lettres. Si le processus récepteur est en attente d'un message ou est inexistant, le dépôt d'un message s'accompagne de son réveil ou de sa création. Un tel type de communication correspond à une

"délégation de pouvoir" donnée au processus receveur. Il n'y a pas de synchronisation entre l'émetteur et le receveur lors de la fin de traitement d'un message de la boîte aux lettres. Nous expliciterons plus loin (cf. 2.431) l'exemple du transfert des fichiers sur l'imprimante qui met en jeu une communication de ce type : lorsqu'un fichier image de l'imprimante virtuelle d'un CMS est fermé, son nom est placé dans une liste des fichiers à imprimer qui joue le rôle d'une boîte aux lettres ; le récepteur en est le processus de gestion de l'imprimante.

Un troisième type de communication d'information mis en place dans GMS s'apparente à un mécanisme de messages [Hansen 70]. Nous verrons que l'environnement d'un processus est défini par le contenu des 16 registres de son processeur virtuel. Cette information est conservée dans une zone de mémoire qui est la représentation du processus lorsqu'il n'est pas actif. Ce descripteur de processus peut être complété par toute information jugée utile par le processus qui en crée ou qui en réveille un autre. Lorsqu'il devient actif, le processus créé ou réveillé peut accéder à son descripteur et récupérer ainsi l'information qui lui est destinée. Nous verrons qu'un tel type de transfert d'information est utilisé, en particulier, pour la réalisation de certaines opérations d'entrée-sortie.

2.324 Synchronisation

Dans GMS, un processus peut être actif, prêt ou en attente d'un signal en provenance d'un autre processus. Nous verrons également qu'un processus peut être inexistant : il s'agit alors d'un processus potentiel qui peut être créé par un autre processus (son environnement initial, son point d'entrée, les ressources

nécessaires sont connus du créateur).

Il existe entre les processus du système des relations d'ordre qui résultent du découpage adopté. Au cours du déroulement d'un processus, certains états qui apparaissent comme résultats de cette exécution peuvent conditionner l'exécution d'autres processus. Nous appelons l'apparition de ces états des événements. A titre d'exemple, la fin d'exécution d'une opération d'entrée-sortie, la fermeture d'un fichier image d'une imprimante virtuelle, la libération d'une zone de mémoire dans la mémoire allouée à GMS, la sortie d'une section critique sont des événements. Les événements peuvent avoir une origine technologique (résultat d'une interruption) ou programmée. Tout événement significatif pour le système est placé sous le contrôle d'une procédure (travaillant, bien entendu, pour le compte d'un processus) chargée de détecter son occurrence et d'appliquer alors le traitement prévu pour cet événement particulier.

Ici encore, comme nous l'avons fait pour la communication entre processus, nous n'avons pas voulu nous limiter à un mécanisme unique de synchronisation mais plutôt définir un ensemble d'outils pouvant s'adapter aux divers cas rencontrés.

Nous utilisons un premier type d'événement que nous appelons événement avec file d'attente. Un certain nombre de ces événements existent dès l'écriture du système, par exemple l'événement libération de mémoire ; d'autres peuvent être créés, par exemple l'événement attente de fin d'entrée-sortie, créé en même temps qu'est soumise une opération d'entrée-sortie et attaché à cette opération particulière.

La file d'attente associée à un tel événement est constituée par la liste des processus qui attendent son occurrence. Cette liste peut être limitée à un seul élément (cas d'un événement fin

d'entrée-sortie).

Lorsque l'événement arrive, le processus qui le détecte applique selon les cas l'une des deux primitives de synchronisation prévues, à savoir :

- réveiller le premier processus de la file,
- réveiller tous les processus de la file.

Réveiller un processus revient à le placer dans la liste des processus prêts. Un tel processus sera rendu actif lors d'un prochain passage dans le Contrôleur qui est la partie du système chargée de l'allocation du processeur aux processus.

Si la file est vide, l'occurrence de l'événement n'a aucun effet (la primitive de synchronisation ne trouve aucun processus à réveiller) et l'événement n'est pas mémorisé. C'est là une caractéristique très importante du mécanisme ; cela veut dire qu'un événement qui arrive alors qu'aucun processus ne l'attend n'empêchera pas un processus de se bloquer si ce dernier demande, plus tard, à être mis en attente de cet événement.

Il est essentiel, si l'on veut comparer ces mécanismes à d'autres, notamment aux mécanismes d'événements mémorisés [IBM 67] ou de sémaphores [Dijkstra 67], de préciser la manière dont ils sont utilisés pour construire un système. Une critique des événements non mémorisés consiste à remarquer [Dijkstra 71] que l'opération WAIT qui conduit un processus à se bloquer derrière la file d'attente d'un événement et l'opération CAUSE qui signale l'apparition de cet événement ne sont pas commutatives, leur effet dépendant de l'ordre dans lequel elles apparaissent. Cela est vrai, mais n'empêche nullement de bâtir un système en utilisant, de manière appropriée, ces types d'événements.

Prenons l'exemple de la gestion de la mémoire libre de GMS sur laquelle nous reviendrons ultérieurement. Nous avons associé un

événement à la libération d'une zone de mémoire par un processus quelconque. En faisant cela, nous permettons à tous les processus qui demandent de la mémoire et ne peuvent l'obtenir de se bloquer derrière cet événement "libération de mémoire". Comme nous l'avons dit, nous avons, lorsque cet événement apparaît, deux possibilités : soit débloquer le premier processus de la file, soit les débloquer tous (le déblocage conduit le processus à réémettre sa demande). C'est cette dernière solution qui a été choisie. Il est évident que si la file d'attente est vide, l'apparition de l'événement n'a aucune signification particulière et que la mémorisation de cet événement n'aurait aucun sens : en effet, à quoi peut servir à un processus qui demande de la mémoire et ne peut l'obtenir, le fait de savoir qu'un autre processus en a libéré quelques temps auparavant ? La relation d'ordre imposée par les événements non mémorisés est donc tout à fait adaptée ici. Nous pouvons remarquer au passage qu'un mécanisme de réveil des processus en attente de mémoire basé sur l'emploi des sémaphores serait difficile à mettre en oeuvre pour la raison simple que la mémoire (telle que GMS l'utilise) n'est pas une ressource quantifiée.

Prenons l'exemple moins favorable qu'est celui de l'exécution d'une opération d'entrée-sortie. Si un processus A soumet une requête d'entrée-sortie à un processus B et demande l'attente de la fin d'opération alors que celle-ci est déjà terminée, il attendra indéfiniment puisque l'événement n'a pas été mémorisé. Il est très facile d'éviter cette situation : l'opération d'entrée-sortie demandée est représentée par un descripteur dans lequel figure un indicateur "opération terminée" distinct de la représentation de l'événement correspondant. Le processus A peut examiner cet indicateur (mis à jour par B sur fin d'opération) et

ne demander l'attente que s'il n'est pas en fonction. Il faut évidemment que l'examen de l'indicateur et la demande éventuelle d'attente soient inclus dans une section critique.

Un second type d'événement utilisé dans GMS ne fait pas intervenir de file d'attente de processus mais est traité directement par le processus chargé de sa détection. Le plus souvent, le traitement consiste en la création d'un processus (processus que nous avons appelé "inexistant"). Par exemple, lors de l'occurrence de l'événement "fermeture d'une imprimante virtuelle" déjà cité, le processus qui détecte cet événement place le nom du fichier correspondant dans la boîte aux lettres qu'il partage avec le processus de gestion de l'imprimante, et il crée ce processus s'il n'existe déjà.

Nous pourrions nous demander, pour ce dernier exemple, pourquoi le processus de gestion de l'imprimante, lorsqu'il n'a plus de fichier à imprimer, n'est pas simplement en attente de l'événement "fermeture d'une imprimante virtuelle". Il eût effectivement été possible de réaliser un tel type de synchronisation mais des ressources sont allouées à ce processus, notamment les zones tampons par lesquelles transitent les enregistrements. Il est par conséquent préférable, lorsqu'il n'y a plus aucun fichier à imprimer, de détruire le processus de gestion de l'imprimante en récupérant les ressources qui lui sont allouées et de ne recréer ce processus et réallouer les ressources que lors de la prochaine arrivée d'un fichier à imprimer (nous verrons que la création d'un processus est une opération simple et peu coûteuse). Le traitement particulier appliqué par le processus détecteur de l'événement "fermeture d'une imprimante virtuelle" est donc bien justifié dans ce cas.

2.33 Implantation des processus et de la synchronisation

2.331 Utilisation du langage d'écriture et conventions de liaison

Le langage choisi pour développer GMS est le langage PL/S (Programming Language / Systems) [Brittenham 73, IBM 72, Potin 73]. C'est un langage dérivé de PL/I utilisé à l'intérieur d'IBM pour la programmation système. Les instructions de PL/S dérivées de PL/I sont les instructions d'affectation et de déclaration et les instructions CALL, RETURN, DO, ELSE, END, ENTRY, GOTO, IF et PROCEDURE. Une instruction GENERATE permet l'inclusion d'instructions ou de données exprimées directement en langage d'assemblage.

L'unité de programme manipulée par le compilateur est la procédure externe. Un programme est un ensemble de procédures externes liées entre elles par des instructions CALL/RETURN. La compilation d'une procédure fournit un programme en langage d'assemblage qui, une fois assemblé, est composé d'une seule section de contrôle.

Le programmeur peut contrôler la génération du programme résultant par l'utilisation d'options dans les en-tête de procédures. L'option REENTRANT spécifie que le programme généré doit être réentrant. D'autres options (SAVE, DONTSAVE, NOSAVAREA) permettent de définir les conventions de liaisons à utiliser. Enfin, il est possible de spécifier les registres de base à utiliser pour l'adressage du programme et pour l'adressage des données.

En l'absence d'option contraire (DONTSAVE, NOSAVAREA) les conventions de liaisons entre procédures, employées par les

programmes générés, sont celles conseillées dans le système OS/360. Rappelons que les méthodes de liaisons entre sections de programmes préconisées par OS/360 [IBM 67] consistent à associer à toute section de programme susceptible d'en appeler une autre , une zone de sauvegarde ("save area") pour les registres du processeur. La sauvegarde est effectuée par le programme appelé dans la zone de sauvegarde de l'appelant. L'appelé doit également disposer d'une zone de sauvegarde s'il effectue lui-même des appels. Les zones sont chaînées entre elles dans les deux sens et l'adresse de la zone courante, celle de la section de programme en train de s'exécuter, est contenue dans un registre particulier (le registre R13). Le point de retour au programme appelant est contenu dans l'un des registres sauvegardés (le registre R14). La figure 5 schématise les liaisons établies de cette manière entre trois programmes P0, P1 et P2 ; P0 appelle P1 qui appelle P2. Le schéma représente le contenu des zones de sauvegarde S0, S1 et S2 au moment où P2 s'exécute.

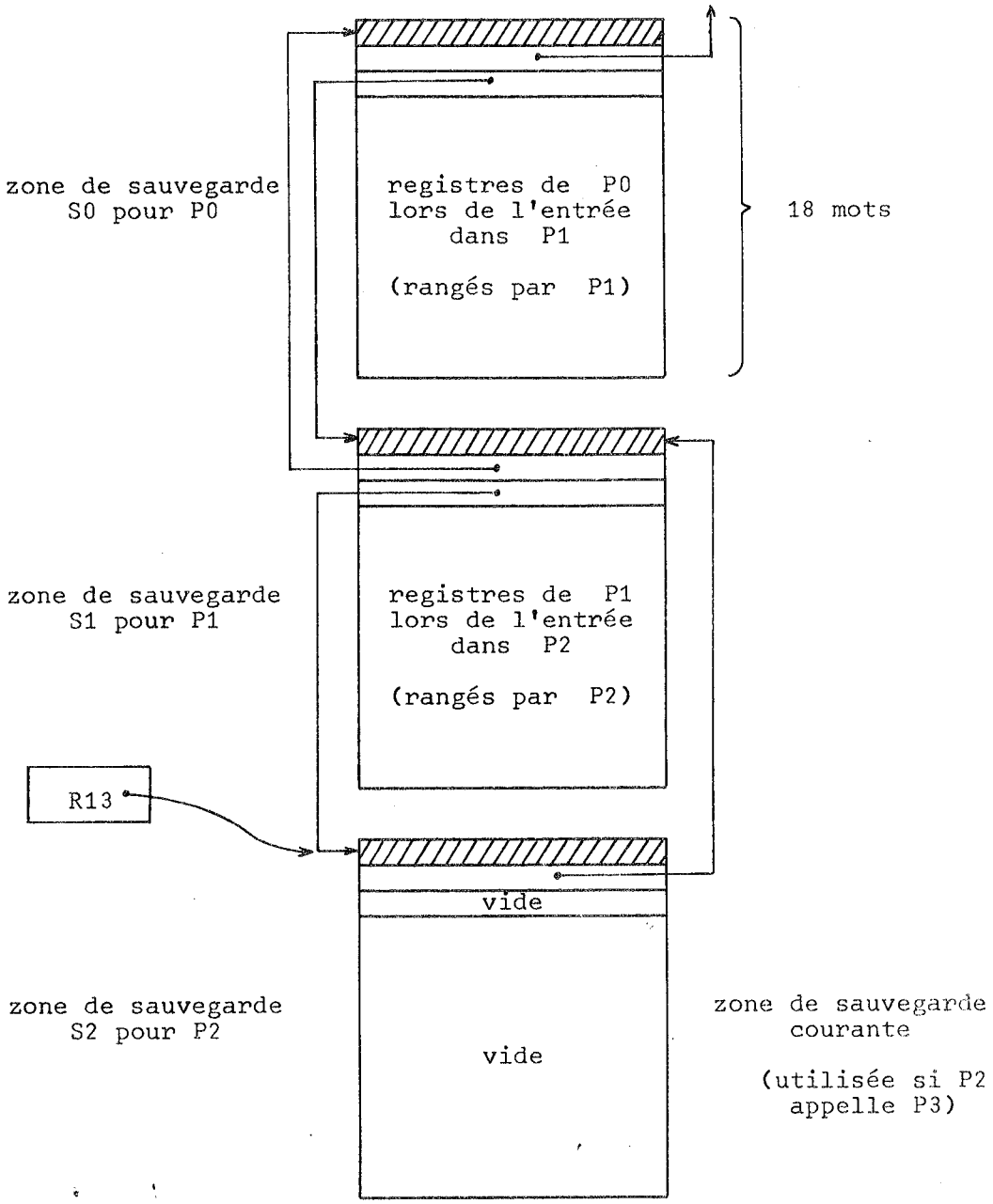


Figure 5. Mécanismes de liaisons type OS/360

Les zones de sauvegarde utilisées par un programme PL/S sont soit statiques, c'est à dire contenues dans les sections de programmes elles-mêmes, soit dynamiques c'est à dire acquises à l'entrée (prologue) et libérées à la sortie (épilogue) des

procédures. Une procédure compilée avec l'option REENTRANT acquiert dynamiquement sa zone de sauvegarde.

Nous pouvons noter une caractéristique importante de ces mécanismes de liaisons : la zone de sauvegarde courante, qu'elle soit statique ou acquise dynamiquement, est toujours disponible. Nous utiliserons cette particularité pour implanter la gestion des processus dans GMS. De même, la connaissance des conventions d'appel et de retour des procédures sera utilisée pour la réalisation des primitives de synchronisation.

2.332 Processus parallèles et réentrance

Dans GMS, le programme d'un processus particulier est composé d'un ensemble de procédures externes reliées par des instructions CALL/RETURN. Dans une procédure réentrante, la mémoire destinée à contenir la zone de sauvegarde et les variables locales est acquise par le prologue de la procédure. Pour que le programme d'un processus soit réentrant (nous parlerons par abus de langage d'un processus réentrant) il faut en principe que toutes les procédures qui le composent soient réentrantes. Nous savons d'autre part qu'il est préférable d'écrire et de mettre au point des procédures de petite taille réalisant chacune une fonction bien définie, plutôt que de regrouper dans de grosses procédures des actions distinctes. Cependant, la multiplication des procédures réentrantes a une incidence sur le temps d'exécution d'un processus puisque chaque appel de procédure entraîne une demande d'allocation de mémoire.

Nous pouvons envisager deux approches complémentaires pour lutter contre cette consommation excessive de ressources.

1- Une procédure n'a besoin d'être réentrante que si elle-même ou d'autres procédures qu'elle appelle risquent d'être interrompues. C'est en effet seulement dans ce cas qu'un autre processus, utilisant éventuellement les mêmes procédures, pourra être activé.

Dans GMS, du fait des options choisies (exécution avec interruptions interdites), seule une action volontaire programmée dans le corps de la procédure (par exemple l'emploi d'une primitive d'attente d'un événement) peut retirer le processeur au processus courant. Il est par conséquent possible, en considérant la chaîne des appels, d'utiliser des procédures non réentrantes en aval de la dernière procédure susceptible de se voir retirer le processeur. Par exemple, dans la figure 6, qui représente les liaisons entre procédures pour un processus donné, la procédure P3 utilise une primitive d'attente matérialisée par l'appel de la procédure WAIT ; P1, P2 et P3 doivent être réentrantes mais P4, P5, P6 et P7 peuvent ne pas l'être. Nous remarquons en particulier que la chaîne secondaire formée de P6 et P7, bien qu'appelée par P2 ne nécessite pas la réentrance.

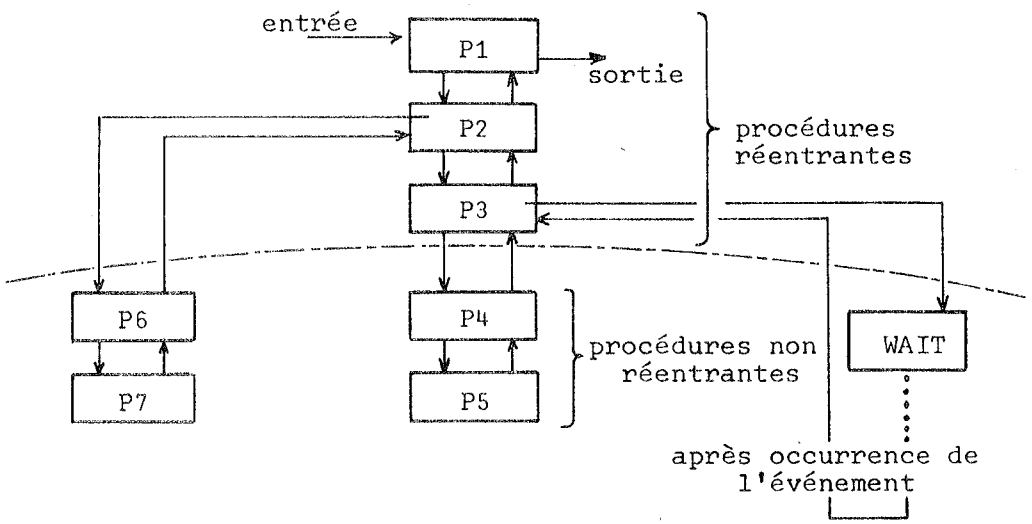


Figure 6. Exemple de chaîne d'appels de procédures

2- Une deuxième manière d'éviter de passer trop de temps à allouer des zones de mémoire pour les procédures réentrantes consiste à accélérer les mécanismes d'allocation. L'allocation des zones de tailles variables conduit à des algorithmes relativement lents. En effet, la recherche d'une zone de taille donnée impose de parcourir la liste des zones libres jusqu'à ce que soit rencontrée une zone de taille supérieure ou égale ; il faut ensuite mettre le résidu en bonne place dans la liste ; enfin, lors de la libération d'une zone, il est nécessaire de regrouper cette zone libre avec les éventuelles zones libres adjacentes pour lutter contre la fragmentation de la mémoire [Knuth 68, Crocus 73]. Toutes ces étapes disparaissent si l'on n'alloue que des zones de taille fixe et si une partie de la mémoire libre est prédécoupée en blocs ayant cette taille [Margolin 71].

La question qui se pose alors est la suivante : peut-on employer pour toutes les procédures réentrantes (ou au moins pour la plupart d'entre elles) une zone dynamique de taille fixe et quelle doit être cette taille ? Nous avons vu que cette zone

dynamique est destinée à contenir la zone de sauvegarde pour les registres du processeur (18 mots y compris les chaînages) et les variables locales à cette procédure. Nous nous apercevons en fait qu'en dehors des variables de travail, qui peuvent pratiquement toujours être contenues dans les registres eux-mêmes, l'utilisation de variables du type AUTOMATIC de PL/I n'est pas nécessaire. Cela est dû au fait que dans un système la plupart des variables utilisées sont des variables globales : ce sont les tables du système, partagées par l'ensemble de ses processus. Ceci nous montre que l'existence de processus parallèles n'implique pas automatiquement la réentrance des procédures composant leur algorithme. Nous pourrions cependant être tentés d'utiliser les mécanismes d'allocation automatique fournis lors de l'entrée dans une procédure pour allouer de la mémoire à un processus. Prenons l'exemple de la procédure hypothétique PRINT à qui l'on soumet une ligne à imprimer. PRINT utilise une zone tampon T pour y placer la ligne puis confie ce tampon à une autre procédure SORTIE (chargée d'en imprimer le contenu), qui s'exécute peut-être dans un autre processus. Si le tampon T est une variable automatique de la procédure PRINT, son allocation est demandée par le prologue de PRINT et sa libération est effectuée par l'épilogue de cette même procédure. Il y a de fortes chances pour que cette façon de gérer le tampon ne nous convienne pas ; notamment, si la procédure SORTIE ne s'exécute pas dans le même processus que la procédure PRINT, on peut vouloir laisser à SORTIE le soin de libérer le tampon une fois la ligne imprimée.

Cet exemple nous montre qu'une telle utilisation des variables automatiques doit être écartée car elle ne permet pas au programmeur de contrôler lui-même l'allocation de ressources. La solution normale consiste dans ce cas à programmer, dans le corps

de la procédure PRINT, la demande d'allocation de T et à programmer la demande de libération dans le corps de PRINT ou dans le corps de SORTIE, selon les choix effectués.

Nous avons décidé de préallouer pour les procédures réentrantes des zones de mémoire de taille fixe égale à 20 mots. La différence entre cette taille et la taille d'une zone de sauvegarde (18 mots) s'explique ainsi : bien que les variables automatiques ne soient pas utilisées, le compilateur peut générer des instructions nécessitant l'emploi d'une zone de travail, notamment pour réaliser des opérations de transferts entre mémoire principale et registres. Cette taille de 20 mots garantit dans tous les cas une place suffisante.

2.333 Représentation des processus et des événements

2.3331 Descripteur d'un processus non actif

Tout processus non actif (prêt ou en attente d'un événement) est complètement défini par les informations à charger dans les 16 registres généraux du processeur pour le rendre actif (après occurrence de l'événement s'il est en attente). Nous n'avons pas associé à chaque processus une liste des ressources qui lui sont allouées. Ce choix résulte simplement du fait que dans GMS il n'y a pas de processus privilégié chargé de contrôler l'exécution des autres processus. Chacun d'eux est responsable des ressources qu'il possède et les libère avant de disparaître. Il existe cependant certains cas où la récupération des ressources est effectuée par le processus qui détecte l'impossibilité pour un autre processus de continuer son exécution. Dans ces cas, la récupération est possible car le processus qui détecte cette

situation a connaissance des structures de données permettant de déterminer les ressources utilisées par le processus à détruire.

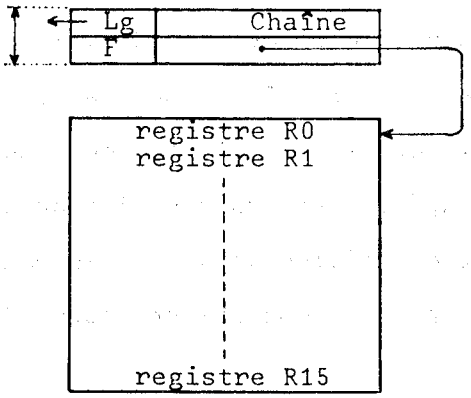
Un processus prêt ou en attente d'événement est représenté dans le système par un descripteur de processus qui a la structure suivante :

Lg	Chaîne
F	Registres

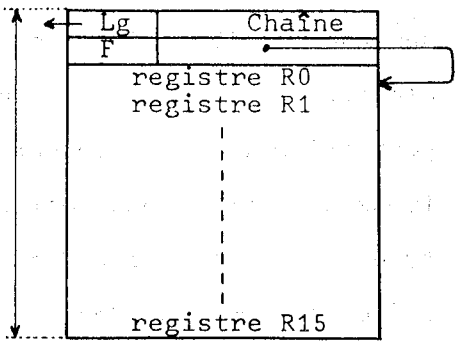
où : - Lg est la taille de ce descripteur, utilisée lors de la destruction du descripteur ;

- Chaîne permet de relier ce descripteur à une liste de descripteurs ;
- F permet d'indiquer au Contrôleur que ce descripteur doit être rendu à la mémoire libre du système lors de l'activation du processus correspondant ;
- Registres est l'adresse d'une zone contenant les 16 registres généraux du processeur virtuel de ce processus.

La zone contenant les registres peut être distincte du descripteur (nous appelons alors ce descripteur un descripteur de type 1) ou être contiguë (descripteur de type 2) ; dans le second cas, la partie longueur tient compte de la zone contenant les registres.



descripteur type 1



descripteur type 2

2.3332 Liste des processus prêts

Le Contrôleur ne connaît que les processus prêts. Les descripteurs des processus prêts sont rattachés à une liste que nous appelons la liste H-Activable (liste des processus Hyperviseur Activables). Le Contrôleur accède à la liste (figure 7) par l'intermédiaire d'une origine de liste donnant les adresses du premier et du dernier descripteurs de la liste.

Un processus P se trouvant dans cette liste peut y être parvenu de deux manières :

a) P était en attente d'un événement ; cet événement est survenu et le processus responsable de sa détection a rattaché P à la liste H-Activable. Dans ce cas, le vecteur d'état de P (qui se réduit aux 16 registres de son processeur virtuel) a été sauvegardé lors de la mise en attente dans la zone de sauvegarde de la procédure courante de P (cf. 2.331). Le descripteur de P est ici un descripteur de type 1.

b) P a été placé dans la liste comme résultat de sa création par un autre processus. Il a fallu pour cela créer le

vecteur d'état initial de P dans une zone acquise par une demande d'allocation de mémoire. Pour éviter d'avoir à demander de la mémoire pour ce vecteur d'état puis ensuite pour créer le descripteur à placer dans la liste, ces deux types d'informations sont regroupées dans un descripteur de type 2. L'indicateur F peut être utilisé dans ce cas pour faire libérer ce descripteur par le Contrôleur lors de l'activation de P.

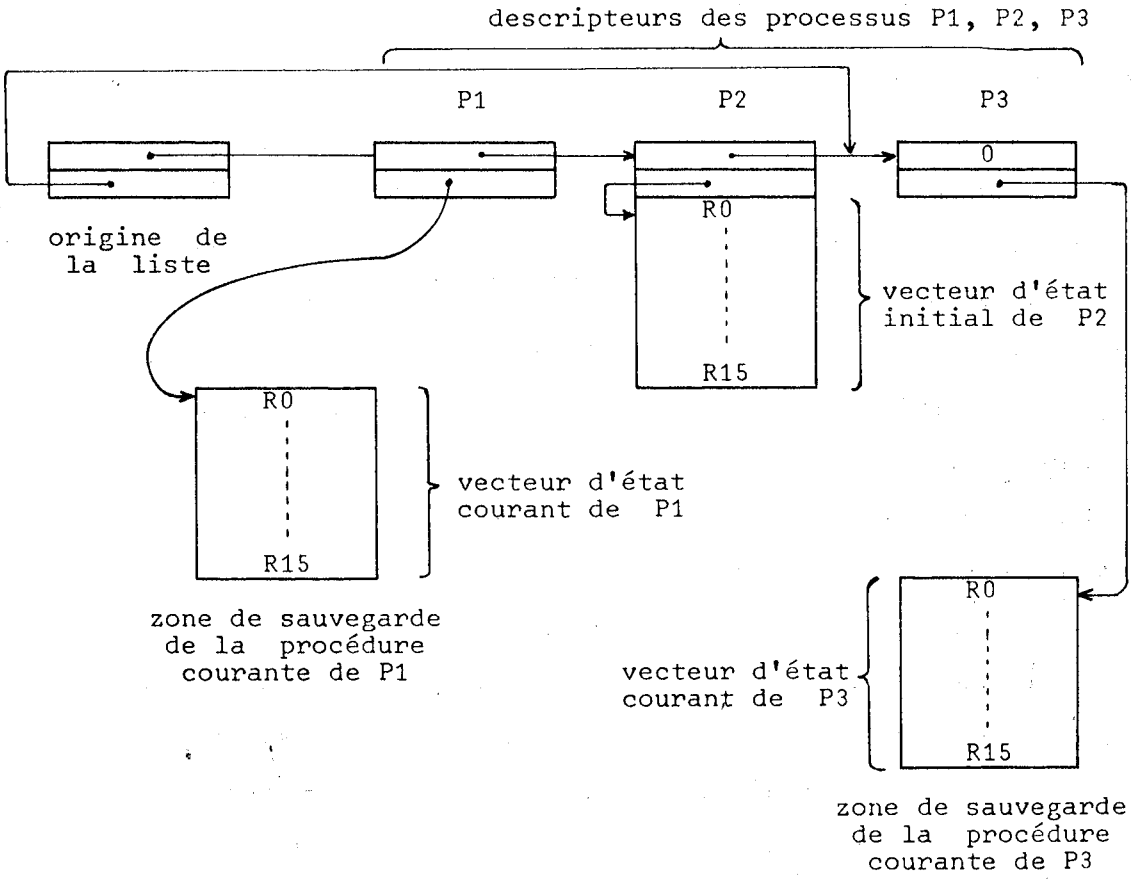


Figure 7. Liste des processus prêts

Sur cet exemple, P1 et P3 sont des processus pour lesquels un événement attendu est arrivé, alors que P2 est un processus que

l'on vient de créer.

2.3333 Processus en attente d'un événement

Un événement est représenté dans le système par descripteur d'événement constitué par deux mots permet d'accéder à la liste des descripteurs des processus en attent cet événement (figure 7 bis). Le premier mot adresse le pre descripteur de la liste ou contient 0 si la liste est vide second mot adresse le dernier descripteur de la liste.

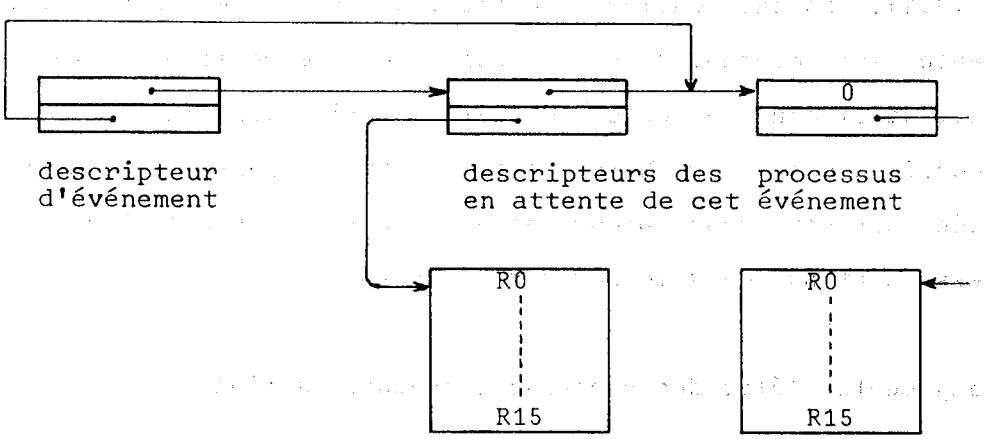


Figure 7 bis. Liste des processus en attente d'un événement

L'adresse du descripteur d'événement est connue du proce chargé de détecter son occurrence et des processus susceptible demander son attente.

2.3334 Processus actif

Le vecteur d'état d'un processus actif est contenu dans registres du processeur. Aucune autre représentation de processus n'existe en mémoire puisque nous avons limité le vec

d'état au contenu des 16 registres de son processeur virtuel.

2.3335 Point de reprise d'un processus

Après sa création, un processus est susceptible de passer alternativement et successivement par les états actif, en attente, prêt, actif, ... La mise en attente d'un processus (cf. 2.3342) résulte d'un appel à une procédure spécifique réalisant la primitive de synchronisation. Les conventions d'appel de cette procédure obéissent aux règles générales utilisées dans le système (cf. 2.331). En particulier, le point de retour à la procédure appelante est contenu dans un registre (le registre R14) et apparaît ainsi dans le vecteur d'état. L'activation d'un processus prêt consiste donc à recharger les registres à partir de la zone adressée par son descripteur et à effectuer un branchement à l'adresse contenue dans le registre de retour.

2.334 Changements d'état des processus, synchronisation

2.3341 Naissance d'un processus

a) Création par un autre processus

La création d'un processus consiste à créer un descripteur contenant son vecteur d'état initial et à rattacher ce descripteur à la liste H-Activable. La création d'un processus qui devrait attendre un événement avant d'être activé ne s'est pas avérée utile lors de la conception du système et du découpage en processus. Cependant, une telle création ne présenterait aucune difficulté : il suffirait de placer le descripteur créé dans la liste associée à l'événement attendu au lieu de le placer dans la

liste des processus prêts.

Lorsqu'un processus A crée un processus B, c'est en général que A a atteint un point de son exécution à partir duquel une activité parallèle peut être lancée. B devra dans la plupart des cas utiliser une partie du contexte de A tel qu'il était au moment de cette création. Par exemple, si A est un processus voulant imprimer un message sur la console de l'opérateur et B le processus chargé de contrôler l'impression, le contexte de A au moment où il crée B contient par exemple l'adresse et la longueur du message à imprimer. Il est donc naturel de transmettre à B ce contexte.

La convention que nous avons adoptée consiste, lors de la création d'un processus, à lui donner comme vecteur d'état initial celui de son créateur en surchargeant certaines parties de ce vecteur avec les informations propres au processus créé. C'est une façon de transmettre de l'information à un processus que l'on créé (cf. 2.323). Nous avons vu que le vecteur d'état est constitué par les 16 registres du processeur virtuel et que l'un de ces registres (R14) contient le point de reprise (ici le point d'entrée) du processus. Il nous faut donc, au minimum, surcharger dans le vecteur d'état créé le registre point de reprise avec l'adresse du point d'entrée du processus.

Plus généralement, la création d'un processus peut être notée symboliquement de la manière suivante :

CREER (Ri= ... , Rj= ... , R14= ... , ... , ...)

Le vecteur d'état du processus créé est alors constitué par les registres du processus actif surchargés par ceux mentionnés dans l'appel.

Si les conventions établies entre le processus créateur et le processus créé sont telles que les mêmes variables (par exemple la

longueur et l'adresse du message à imprimer) se trouvent dans les mêmes registres, alors la création s'écrit simplement :

CREER (R14= EP)

où EP est le point d'entrée du processus créé.

b) Naissance spontanée d'un processus

L'activité du processeur peut être considérée à un instant donné comme rattachée à l'un des deux domaines suivants (cf. 2.252).

- Domaine utilisateur U dans lequel s'exécutent les CMS virtuels. Le processeur est alors en mode programme et toutes les interruptions sont permises.
- Domaine hyperviseur H dans lequel s'exécutent les processus de GMS. Le processeur est alors en mode superviseur et les interruptions d'entrée-sortie sont interdites.

Le passage de U à H résulte exclusivement d'une interruption et le passage de H à U s'effectue par chargement du mot d'état programme (PSW).

Les interruptions redonnant le contrôle à GMS (passage de U à H) proviennent d'événements technologiques asynchrones (interruption d'entrée-sortie, interruption externe) ou d'événements synchrones (déroutement "programme" ou "appel superviseur" SVC). Dans la terminologie IBM/360, ces deux types d'événements sont regroupés sous le terme d'interruptions.

Nous savons qu'une interruption se matérialise par un échange automatique de mots d'état programme. L'instruction exécutée ensuite est celle dont l'adresse se trouve dans le "nouveau" mot d'état devenu mot d'état "courant". A chacune des quatre classes d'interruptions (entrée-sortie, externe, programme, SVC)

correspond un point d'entrée dans GMS ; ce point d'entrée est l'adresse contenue dans le nouveau mot d'état associé.

Au moment où la permutation des mots d'état a lieu, un processus naît. Ce processus, après avoir sauvegardé le contexte du CMS virtuel interrompu, initialise lui-même son propre contexte (garnissage des registres de base pour l'adressage notamment). Dès cet instant, plus rien ne le distingue d'un processus devenu actif par suite d'une création explicite ou d'un réveil. Il peut se placer en attente d'événement ou se terminer en utilisant les mécanismes communs. S'il se termine sans avoir demandé l'attente d'un événement, il n'aura jamais été matérialisé autrement que par le contenu des registres du processeur. C'est le cas par exemple lorsque l'interruption qui lui a donné naissance est une interruption programme qui peut être directement réfléchie au CMS virtuel dérouté (violation de la protection de la mémoire, code opération invalide, ...). Cette approche permet de traiter de façon efficace les événements qui peuvent être traités en un seul passage, tout en conservant la généralité des mécanismes mis en place.

Une autre alternative, fréquemment proposée [Denning 71], aurait consisté à considérer qu'à chacune des quatre classes d'interruptions était associé un processus en attente de cette interruption et à appliquer lors de son occurrence un primitive de réveil de ce processus. Cette méthode peut paraître a priori plus élégante en ce sens qu'elle uniformise les techniques de création et de réveil, mais il faut être conscient du fait que ce souci d'élégance conduit à raccrocher le processus ainsi réveillé à la liste des processus prêts, puis à appeler le Contrôleur qui décidera du processus à activer (il sélectera forcément le processus en question). Cela impose également d'avoir une

description permanente du contexte de ces processus. Pour des raisons d'efficacité, nous avons rejeté cette solution.

2.3342 Mise en attente d'un processus

Tout processus qui doit attendre, pour poursuivre son exécution, l'occurrence d'un événement dont la détection est à la charge d'un autre processus, peut se bloquer de lui-même en se plaçant dans la file d'attente de cet événement. Il utilise pour cela une primitive de synchronisation qu'il active par appel de la procédure INACT :

CALL INACT (Descr, E)

où Descr est une zone de mémoire (2 mots) destinée à recevoir un descripteur de processus et où E est le nom de l'événement attendu. Ce nom désigne l'emplacement en mémoire du descripteur de cet événement.

Par la suite, nous noterons simplement INACT (E) l'emploi de cette primitive.

La primitive INACT effectue les actions suivantes (figure 8).

a) Sauvegarder le vecteur d'état du processus courant dans la zone de sauvegarde de la procédure appelante.

b) Garnir le descripteur de processus fourni et le placer en queue de la liste accrochée au descripteur de l'événement E.

c) Effectuer un retour au Contrôleur qui choisira le prochain processus à activer.

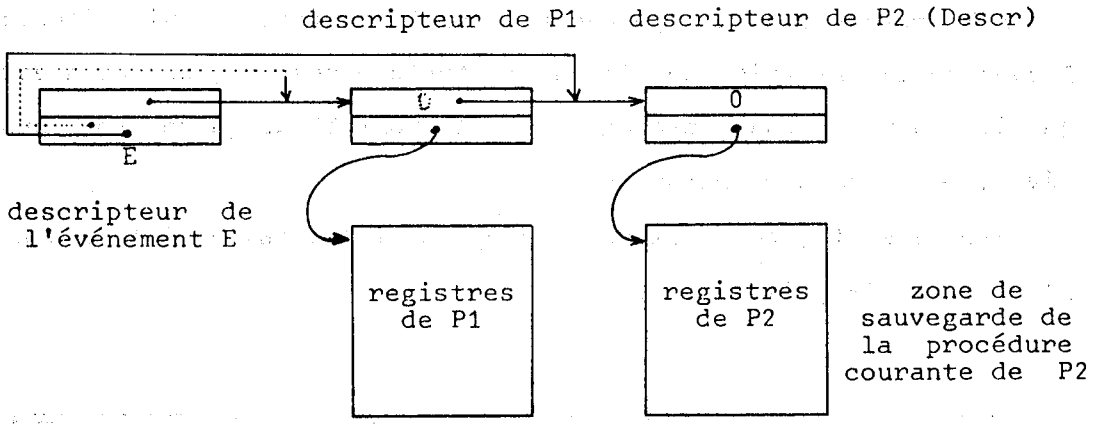


Figure 8. Résultat de INACT (E) exécuté par le processus P2

Rappelons que les événements dans GMS ne sont pas mémorisés ; en conséquence, un processus demandant l'attente d'un événement doit au préalable s'assurer qu'il n'est pas encore arrivé. Cette information n'est pas contenue dans le descripteur de l'événement mais dans des structures de données connues des processus que cet événement concerne.

2.3343 Réveil d'un processus

Lorsqu'un événement E avec file d'attente est détecté par le processus qui en a la charge, ce dernier exécute la primitive de synchronisation ACT par l'appel de procédure

CALL ACT (E, |ALL|)

Si le paramètre ALL est spécifié, tous les descripteurs de processus se trouvant dans la file d'attente de E sont raccrochés à la liste des processus prêts. Dans le cas contraire, seul le premier descripteur de la liste est rattaché à la liste des processus prêts (figure 9). Lorsque la file d'attente de E est vide, la procédure ACT n'a aucun effet.

ACT retourne à la procédure appelante (celle qui détecte l'événement) une fois les files d'attente mises à jour. Ce n'est que lors d'un prochain passage dans le Contrôleur qu'un processus réveillé pourra être rendu actif.

Par la suite, nous noterons simplement ACT (E) ou ACT (E,ALL) l'emploi de la primitive ACT.

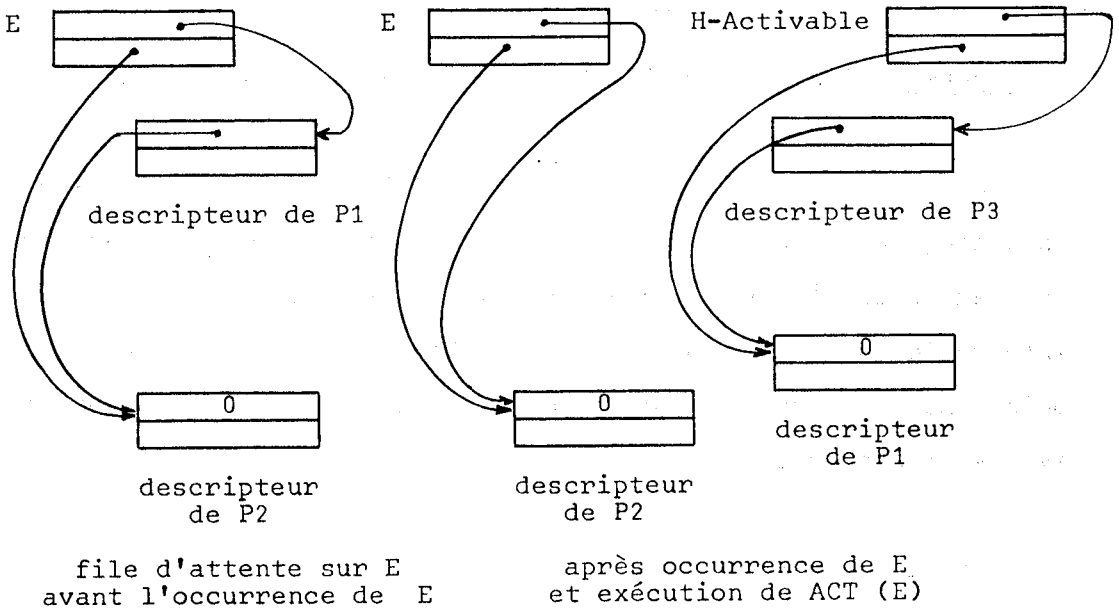


Figure 9. Réveil d'un processus

2.3344 Disparition d'un processus

Nous avons vu qu'un processus actif n'est matérialisé que par le contenu des registres du processeur. La fin d'exécution de ce processus est signalée au système par l'exécution, dans sa procédure courante, de l'instruction

RETURN TO DISPATCH.

(DISPATCH est le nom du point d'entrée du Contrôleur).

Lorsque l'exécution d'un processus met en jeu une chaîne d'appels de procédures réentrantes, ce retour au Contrôleur doit figurer dans la procédure la plus extérieure de manière à ce que les zones de mémoire acquises par le prologue de chacune des procédures de niveau inférieur aient été restituées par l'épilogue correspondant lors de l'exécution de l'instruction RETURN (figure 10). Nous avons en effet déjà signalé que ces ressources en mémoire allouées aux procédures réentrantes ne sont pas répertoriées dans le système ailleurs que dans les zones dynamiques de sauvegarde elles-mêmes.

Remarque. Il eût été possible de permettre le retour au Contrôleur depuis une procédure de niveau inférieur et de laisser le soin à ce dernier de récupérer les zones dynamiques de niveau supérieur en utilisant les chaînages garantis par le compilateur (cf. 2.331). Il est en effet possible de remonter, à partir de la zone dynamique de la procédure courante aux zones dynamiques de niveau supérieur en utilisant le chaînage inverse mis en place. Nous n'avons pas utilisé cette possibilité pour les processus qui se terminent par leur propre volonté, pour deux raisons.

a) La terminaison d'un processus décidée dans une procédure de niveau inférieur correspond au cas où cette procédure détecte une condition qui empêche la poursuite de l'exécution du processus. Cette procédure a en effet été appelée pour exécuter une fonction précise pour le compte de la procédure appelante. Il est dans ce cas plus logique de retourner à la procédure appelante en utilisant une sortie d'erreur (ou un code de retour spécial) et de laisser le soin à cette dernière procédure de traiter cette situation anormale.

b) Il eût fallu traiter à part le cas des procédures non réentrantes (en utilisant un point de retour spécial) et cela eût été une source d'erreurs possibles. En effet, on peut très bien décider, après qu'elle ait été écrite, qu'une procédure doit être compilée de manière à être ou non réentrante.

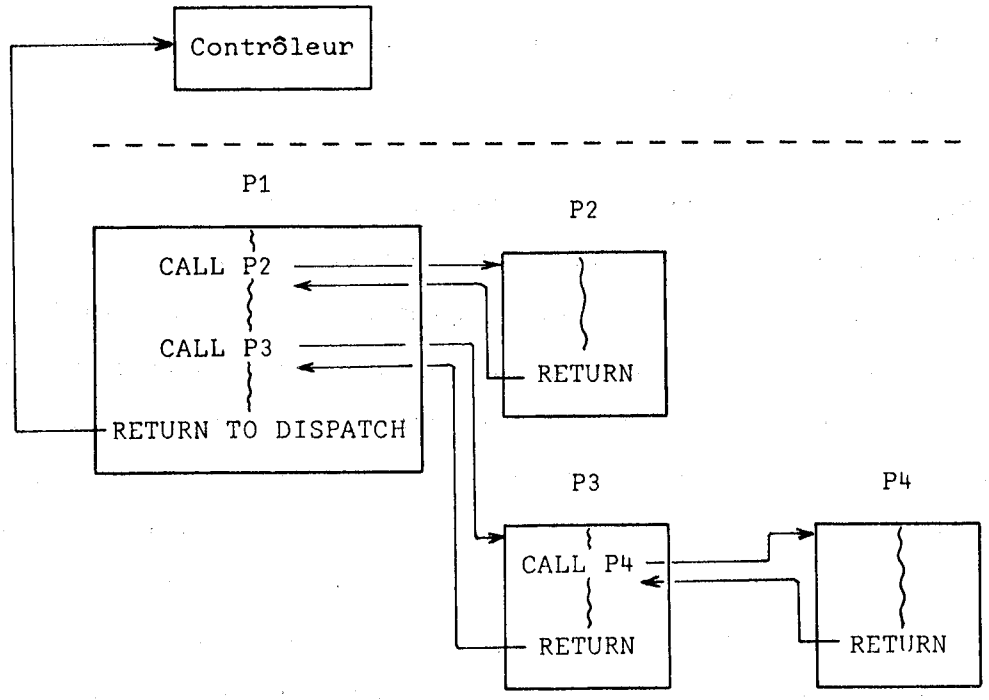


Figure 10. Liaisons entre procédures dans un processus

2.335 Sections critiques et accès aux informations partagées

Nous avons vu que les informations d'état du système sont accessibles par tous les processus. Ces derniers qui s'exécutent dans un même espace d'adressage peuvent, sans intermédiaire, consulter ou modifier toute information commune. Un problème classique dans la conception des systèmes basés sur l'exécution de

processus parallèles consiste à assurer la cohérence des informations partagées. Pour cela, nous définissons une section critique [Dijkstra 67] à l'intérieur de laquelle un processus, au plus, peut se trouver à un instant donné. A l'intérieur de cette section critique figurent les instructions qui lisent ou modifient une information partagée. L'entrée en section critique ou la sortie d'une section critique sont effectuées en utilisant des primitives du système.

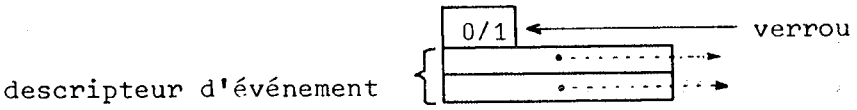
Dans GMS, la plupart des accès aux informations partagées ne nécessitent pas l'emploi explicite de primitives d'entrée en section critique. En effet, le réalisation de sections critiques résulte directement du mode de fonctionnement du processeur (monoprocesseur fonctionnant avec interruptions masquées). Un processus ne libère le processeur que de façon consciente, par l'emploi d'une primitive d'attente d'événement. Si le processus actif, pendant tout le temps où il consulte ou modifie une information partagée, n'emploie pas une telle primitive, il est certain que l'accès à cette information est automatiquement inclus dans une section critique.

Par contre, un processus doit utiliser explicitement des primitives d'entrée en section critique et de sortie de section critique chaque fois qu'il risque de libérer le processeur au cours d'une séquence d'instructions accédant à une information partagée. Ces primitives, LOCK et UNLOCK sont activées par les appels de procédures

```
CALL   LOCK   (verrou)
CALL   UNLOCK (verrou)
```

où verrou est un bloc de contrôle de section critique composé d'un indicateur binaire (le "verrou") mis à 1 lorsqu'un processus est dans la section critique, et d'un descripteur d'événement

permettant de placer en attente les processus qui ne peuvent entrer en section critique.



Lorsqu'aucun processus ne se trouve dans la section critique, le verrou est à zéro et aucun processus n'est rattaché à l'événement.

Lorsqu'un processus se trouve dans une section critique, le verrou associé à cette section est à 1 et au descripteur d'événement qui lui correspond sont rattachés, s'il y en a, les processus attendant de pouvoir entrer dans cette section critique.

LOCK et UNLOCK utilisent les mécanismes de mise en attente et de réveil. Leur algorithme est le suivant si l'on désigne par E l'événement associé au verrou concerné :

```
LOCK : si  verrou = 0 alors
           verrou := 1 sinon
           INACT (E)   fin ;

UNLOCK : si file d'attente vide
             alors  verrou := 0
             sinon  ACT (E)  fin ;
```

De tels mécanismes supposent une discipline concertée de la part des processus utilisant des informations partagées. En particulier, un phénomène d'interblocage peut apparaître si un processus se trouvant en section critique se place en attente d'un événement dont la détection est à la charge d'un autre processus utilisant la même section critique. Des outils théoriques existent pour prévenir ou détecter une telle situation, ou pour y remédier [Coffman 71, Haberman 69]. Leur utilisation obligerait à définir les sections critiques comme des ressources et à placer au dessus des processus un mécanisme centralisé d'allocation de ressources.

Dans le cas de GMS, l'ensemble des processus composant le système est limité, les relations entre ces processus sont bien définies, les ressources qu'ils utilisent également ; il est par conséquent possible, dans chaque cas particulier, de structurer ces processus de manière à éviter l'interblocage. Nous évitons ainsi l'emploi de mécanismes certes très généraux mais qui auraient l'inconvénient de consommer pour eux-mêmes une partie des ressources de l'installation (mémoire centrale et temps de processeur).

2.336 Allocation du processeur aux processus

Le Contrôleur est activé chaque fois qu'un processus se termine (RETURN TO DISPATCH) ou se place en attente (INACT). Le premier processus se trouvant dans la liste H-Activable est alors rendu actif. Rappelons que cette activation se résume au chargement des registres et au branchement à l'adresse contenue dans le registre de retour (R14).

Nous avons vu (cf. 2.3331) que le descripteur de processus contient un indicateur F. Si F est en fonction, alors la zone de mémoire occupée par ce descripteur est rendue à la mémoire libre du système par le Contrôleur. Cette possibilité est surtout utilisée à la suite de la création d'un processus. En effet, il a fallu acquérir une zone de mémoire pour loger le descripteur initial du processus. Cette zone de mémoire est utilisée lors de la première activation du processus mais doit en général être libérée à ce moment là.

Lorsque la file H-Activable est vide, le Contrôleur tente d'allouer le processeur à un CMS virtuel. A ce niveau là, les contraintes sont différentes, les techniques d'allocation aussi. Nous étudierons en 2.71 la stratégie d'allocation correspondante.

2.4 GESTION DES RESSOURCES PHYSIQUES

Toute ressource de l'installation (unité d'entrée-sortie, processeur, mémoire, ...) est soit affectée à un CMS virtuel, de façon permanente ou temporaire, pour servir de support à une ressource virtuelle, soit attribuée en permanence à GMS. Dans le premier cas se posent des problèmes d'allocation de ces ressources que nous évoquerons en 2.51. Dans les deux cas se posent des problèmes d'utilisation de ces ressources. Par exemple, il n'y a pas lieu de distinguer du point de vue de la gestion des entrées-sorties, une demande provenant d'un CMS virtuel d'une demande initialisée pour les besoins propres de GMS : au delà d'un certain traitement préliminaire, il est possible de présenter ces demandes au niveau inférieur sous une forme commune.

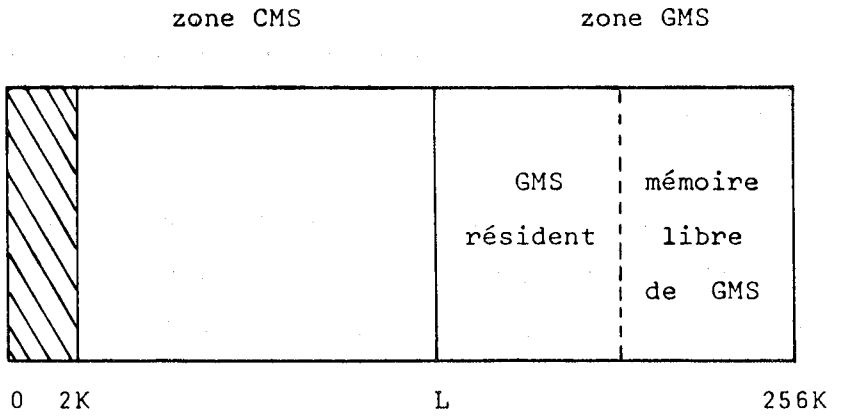
Dans ce paragraphe, nous exposons les techniques mises en oeuvre pour la gestion de la mémoire libre de GMS et pour la gestion des ressources d'entrée-sortie. Puis, nous plaçant à un niveau plus global, nous présentons, comme exemples d'utilisation des mécanismes généraux mis en place, la gestion de l'imprimante et celle de la console de l'opérateur.

2.41 Gestion de la mémoire libre de GMS

La mémoire centrale (256 K octets) est divisée en deux zones :

- la "zone CMS" qui s'étend depuis l'adresse 2K jusqu'à une limite L de l'ordre de 200K (L est un paramètre de génération du système) ; cette zone est multiplexée globalement entre les CMS virtuels prêts à être activés (cf. 2.515) ;

- la "zone GMS" qui s'étend depuis L jusqu'à la fin de la mémoire ; cette zone contient GMS lui-même, résident en permanence, et une partie de mémoire libre dans laquelle GMS puise pour ses propres besoins.



Les besoins en mémoire de GMS proviennent de deux origines principales.

1- Zones de mémoire acquises par le prologue des procédures réentrantes pour la sauvegarde des registres, les variables locales et les zones de travail. Nous avons vu (2.331) que dans la plupart des cas ces zones ont une taille comprise entre 18 et 20 mots. La demande d'une telle zone est matérialisée dans le programme généré par un appel au superviseur (SVC 10) correspondant à la fonction GETMAIN telle qu'elle est définie dans OS/360 [IBM 67]. La libération est matérialisée dans l'épilogue par l'appel au superviseur correspondant à la fonction FREEMAIN.

2- Zones de mémoire demandées explicitement dans le corps de la procédure par l'appel d'une procédure d'allocation et rendues par l'appel d'une procédure de libération. Les utilisations de zones de ce type sont multiples dans GMS.

Citons par exemple :

- les tampons utilisés pour les fichiers GMS (cf. 2.431) ou pour les entrées-sorties sur console (cf. 2.524) ;
- les zones acquises pour initialiser le vecteur d'état d'un processus que l'on crée ou pour contenir le descripteur d'un processus qui se place en attente ;
- la zone destinée à contenir les informations décrivant un utilisateur qui joint le système.

L'examen de l'ensemble des besoins du système nous conduit à subdiviser la totalité des demandes de mémoire en trois classes, relativement aux actions à entreprendre lorsque l'allocation est impossible.

Classe 1 : demandes inconditionnelles

Une demande inconditionnelle, lorsqu'elle ne peut être satisfaite entraîne un blocage du système. Les techniques d'allocation employées doivent être telles qu'une situation de ce genre ne puisse se produire. Il y a plusieurs cas de demandes inconditionnelles dans le système. Par exemple, si un processus désirant attendre un événement ne peut pas obtenir la zone nécessaire à la construction du descripteur d'attente, il ne peut pas libérer le processeur. Un autre exemple concerne la zone de sauvegarde des registres acquise par le prologue d'une procédure réentrante : s'il n'existe pas de zone de sauvegarde en amont de cette procédure, une mise en attente de la demande n'est pas possible. Nous avons vu en effet (cf. 2.3342) que la mise en attente effectuée par INACT consiste à sauvegarder les registres du processeur dans la zone de sauvegarde de la procédure qui exécute INACT. Nous avons choisi de traiter toutes les demandes

émises par le prologue des procédures comme des demandes inconditionnelles et avons prévu, en plus, de programmer explicitement de telles demandes lorsque cela était nécessaire.

Classe 2 : demandes conditionnelles avec attente

Les demandes conditionnelles avec attente sont émises par des processus dont la structure permet cette attente ; le processus ne reprend le contrôle que lorsque la demande a pu être satisfaite. Ce type de demande est très courant. Citons par exemple :

- la demande d'un tampon pour écrire un message sur la console de l'opérateur ou sur un terminal ;
- la demande de mémoire pour créer un enregistrement destiné à un fichier ou pour lire un tel enregistrement.

Classe 3 : demandes conditionnelles sans attente

Dans certaines situations, un processus qui ne peut obtenir une zone de mémoire demandée peut alors vouloir effectuer une activité complémentaire avant, éventuellement, de se placer de son propre gré en attente. Par exemple, lorsqu'un utilisateur joint le système par la commande LOGIN, le processus qui pilote cette commande doit acquérir une zone de mémoire (de taille relativement importante) dans laquelle prendront place toutes les informations utiles pour le système concernant cet utilisateur. Si une telle zone n'est pas disponible, le processus, avant de se placer en attente de mémoire peut écrire sur le terminal concerné un message demandant à l'utilisateur potentiel de patienter.

2.411 Schéma général de l'allocation

Une procédure unique d'allocation ALLOC traite à la fois les

demandes conditionnelles et inconditionnelles. L'appel au superviseur SVC 10 correspondant à une demande inconditionnelle est traduit par le programme de traitement des interruptions SVC en un appel de cette procédure ALLOC. Une demande conditionnelle, avec ou sans attente, est réalisée par appel de la procédure GETCORE. Cette dernière soumet la demande par appel de ALLOC.

La figure 11 illustre les deux types d'utilisation de la procédure d'allocation. On remarque que lorsqu'une demande conditionnelle entraîne une mise en attente du processus, cette mise en attente se situe à l'intérieur de GETCORE. Cette dernière est donc une procédure réentrante alors que ALLOC n'a pas besoin d'être réentrante. Cette particularité permet une allocation plus rapide pour les demandes inconditionnelles (partie gauche du schéma). Nous allons voir que la structure de la mémoire libre adoptée permet également d'accélérer les allocations pour ces demandes inconditionnelles.

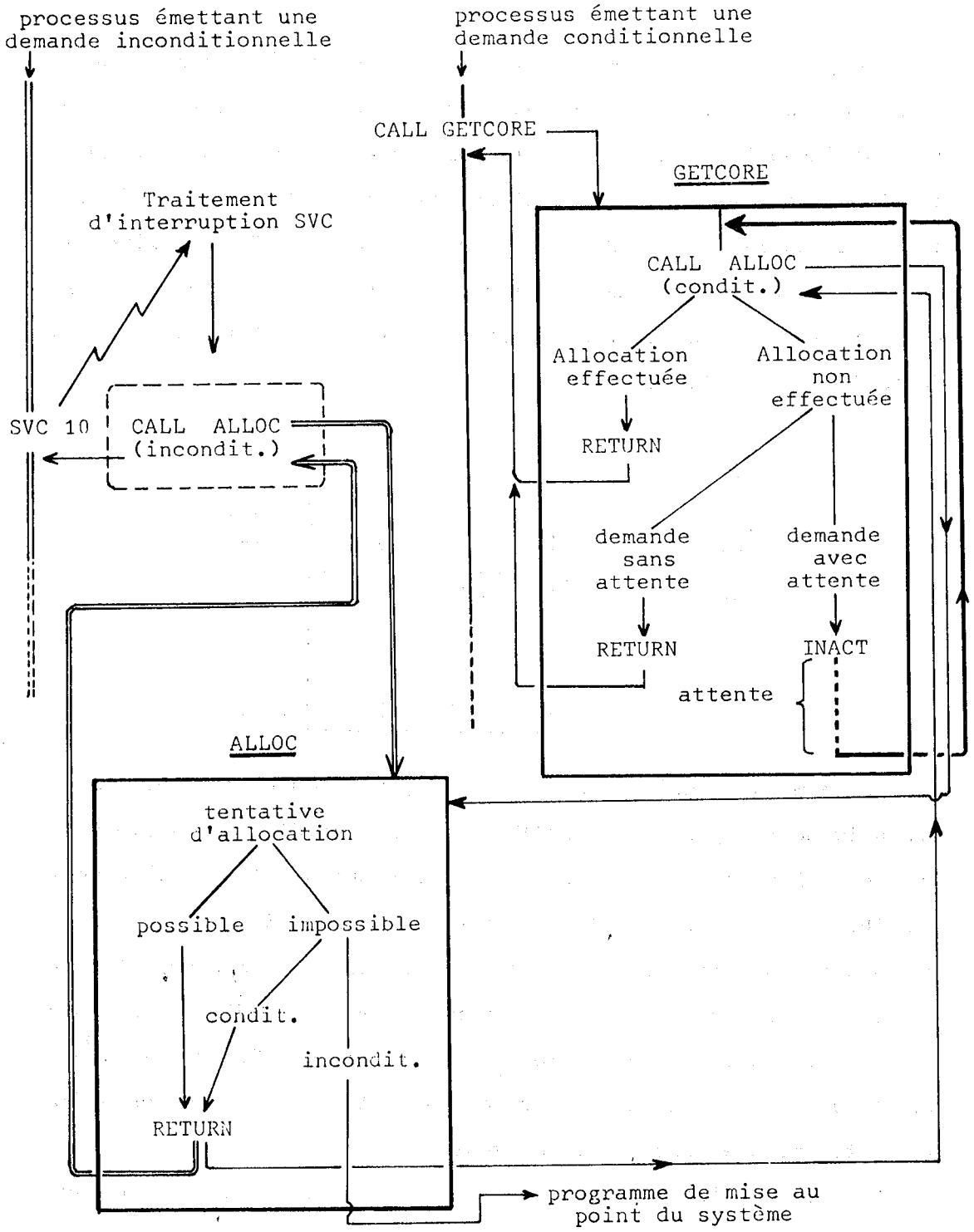
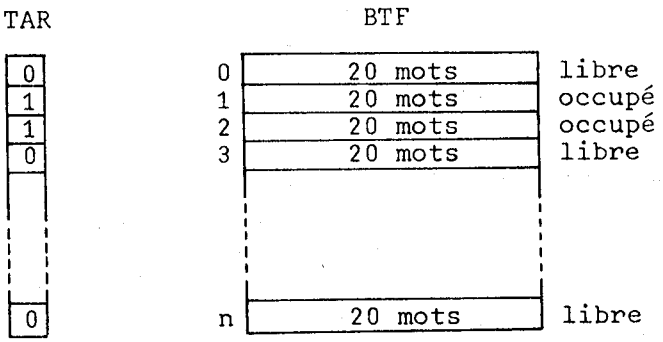


Figure 11. Schéma général de l'allocation de mémoire

2.412 Algorithme d'allocation

Une partie de la mémoire libre est découpée en blocs de 20 mots, cette taille correspondant à la majorité des demandes inconditionnelles émises par le prologue des procédures. Nous appelons ces blocs des BTF (Blocs de Taille Fixe). Une "Table d'Allocation Rapide" (TAR) permet de savoir si un BTF est libre ou alloué. Le reste de la mémoire libre est composé de zones chaînées par adresses croissantes. Nous appelons ces zones des ZOVAR (zones de tailles variables).



Une demande conditionnelle (émise par GETCORE) est satisfaite exclusivement à partir des ZOVAR. La recherche d'une zone libre de taille voulue est effectuée par la technique de la première zone possible avec liste circulaire [Crocus 73, Knuth 68].

Une demande inconditionnelle pour une taille comprise entre 18 et 20 mots est satisfaite soit par l'affectation d'un BTF libre s'il existe, soit par une allocation à partir des ZOVAR. Le nombre de BTF défini a priori n'est donc pas critique puisque les ZOVAR sont utilisées en cas de dépassement.

Si nous n'utilisions que des BTF pour les demandes inconditionnelles, il en faudrait un nombre égal au nombre maximum possible d'appels de procédures réentrantes en chaîne pour

l'ensemble des processus existant à un instant donné (actif ou en attente). Une proportion importante de ces BTF serait alors inutilisée dans la plupart des cas. Une estimation, complétée par des mesures effectuées sur les premières versions du système, a permis de fixer le nombre de BTF de manière à réaliser un compromis raisonnable entre la place occupée et la probabilité d'allocation sans débordement sur les ZOVAR. Ce nombre a été fixé à 25.

2.413 Mise en attente et réveil d'un processus demandeur

Nous avons vu (figure 11) que la procédure GETCORE contient la mise en attente du processus pour le compte duquel elle s'exécute, lorsque la demande ne peut être satisfaite. Tous les processus en attente de mémoire sont donc rattachés au même événement. La libération d'une zone de mémoire déclenche le réveil de tous les processus en attente de cet événement. La libération est réalisée par une procédure DESALLOC qui est appelée soit sur exécution de l'appel superviseur adéquat contenu dans l'épilogue des procédures réentrantes, soit par la procédure FREECORE lorsqu'il s'agit d'une libération programmée. DESALLOC se termine par l'exécution de la primitive ACT(Ev, All), où Ev est l'événement "attente de mémoire".

Chacun de ces processus, lorsque le Contrôleur l'active, réexécute (d'après la structure de GETCORE) l'appel de la procédure ALLOC et peut à nouveau se placer en attente de mémoire si la zone libérée, ou ce qu'il en reste, ne permet pas de satisfaire sa demande. Cette méthode permet d'assurer que lorsque tous les processus réveillés ont à nouveau émis leur demande, aucun de ceux qui sont revenus dans la file d'attente ne peut être

satisfait par la mémoire disponible.

Exemple.

Supposons que les processus P1, P2, P3, P4 se trouvent dans cet ordre dans la file d'attente avec des demandes respectives de 100, 200, 80 et 70 mots, et qu'un autre processus libère une zone de 240 mots. Les quatre processus sont réveillés par cette libération et, après une activation de chacun, P2 et P4 se retrouvent dans la file d'attente alors que P1 et P3 ont vu leur demande satisfaite. De la zone libérée il reste 60 mots.

Remarque. La méthode que nous venons d'exposer consiste à placer l'allocateur de mémoire dans chaque processus. Une autre alternative consiste à placer l'allocateur dans un processus à part. Le demandeur, au lieu d'appeler une procédure, active le processus allocateur en lui fournissant les paramètres de sa demande et se place en attente de l'événement "allocation réalisée". Si la demande ne peut être satisfaite, le processus allocateur la place dans une liste des demandes non satisfaites et se place lui-même en attente de l'événement "libération de mémoire". Lorsqu'une libération survient le processus allocateur, et lui seul, est réveillé. Il essaie alors de satisfaire avec la zone libérée les demandes se trouvant dans la liste. Lorsqu'il réalise une allocation, il réveille le processus demandeur correspondant.

Nous n'avons pas utilisé cette alternative car il est moins coûteux d'appeler une procédure que d'activer un processus. D'autre part, cette méthode ajoute une mise en attente et un réveil du processus demandeur lors de chaque demande, même si cette dernière peut être satisfaite immédiatement (ce qui est le

cas général).

Nous pouvons cependant remarquer que la méthode que nous avons employée (allocateur dans chaque processus) ne serait pas adaptée s'il existait en permanence une file importante de processus en attente de mémoire. Nous voyons en effet qu'un processus en attente de mémoire peut, avant que sa demande ne soit satisfaite, être tour à tour réveillé à chaque libération et replacé en attente lorsque la zone libérée n'est pas suffisante, et cela un nombre de fois indéterminé.

Un problème particulier se pose lors de la libération d'un descripteur de processus en attente de mémoire. Rappelons en effet que lorsque dans le descripteur d'un processus prêt est spécifié l'indicateur de libération F, ce descripteur est rendu à la mémoire libre par appel de la procédure FREECORE (qui elle-même appelle DESALLOC) lors de l'activation. La procédure GETCORE, lorsqu'elle veut placer le processus qui l'appelle en attente de mémoire, acquiert un descripteur (un double mot) par une demande inconditionnelle. Ce descripteur est rattaché à l'événement "libération de mémoire" et l'indicateur F est spécifié. Supposons alors que deux processus P1 et P2 soient en attente de mémoire ; une libération de mémoire par un autre processus déclenche le réveil de P1 et P2 qui se retrouvent ainsi dans la file H-Activable. Supposons encore que la zone libérée ne soit pas suffisante pour satisfaire la demande de P1 ni celle de P2. Lorsque P1 après avoir été activé se retrouve à nouveau dans la file d'attente, P2 est activé à son tour. Cette activation s'accompagnant de la libération du descripteur de P2, les processus en attente de mémoire (dans ce cas P1) sont réveillés. Après retour de P2 dans la file d'attente, P1 est à nouveau activé

et son activation réveille P2. Nous arrivons à une situation de bouclage dans laquelle chaque processus réveille l'autre avant de se placer en attente. Pour éviter ce problème, le Contrôleur, lorsqu'il libère le descripteur du processus qu'il active, utilise un point d'entrée spécial de la procédure de libération. Lorsque ce point d'entrée est utilisé, la libération ne s'accompagne pas du réveil des processus en attente de mémoire.

2.42 Gestion des opérations d'entrée-sortie

Toutes les opérations d'entrée-sortie, qu'elles proviennent, après une transformation virtuel-réel, d'une sollicitation d'un CMS virtuel ou qu'elles correspondent à un besoin propre de GMS, sont soumises à un programme unique, le superviseur d'entrée-sortie. Ce superviseur d'entrée-sortie assure la gestion de toutes les unités d'entrée-sortie et des chemins qui y mènent (canaux et unités de contrôle). Il exploite le parallélisme de fonctionnement des unités et optimise l'utilisation de l'ensemble des ressources d'entrée-sortie. Les problèmes de conception et de réalisation du superviseur d'entrée-sortie de GMS sont étudiés de manière très détaillée dans [Finet 73b]. On trouvera d'autre part [Bellino 70] une étude générale des problèmes de conception des superviseurs d'entrée-sortie dans un contexte de multiprogrammation. Dans ce paragraphe, nous mettons principalement l'accent sur les problèmes de synchronisation entre une opération d'entrée-sortie et le processus demandeur et sur la place du superviseur d'entrée-sortie dans le système. Nous exposons comment les mécanismes de base mis en place précédemment sont utilisés pour réaliser ces opérations d'entrée-sortie.

2.421 Conception globale

Une demande soumise au superviseur d'entrée-sortie est du type "Appliquer tel programme canal (commande élémentaire ou suite de commandes élémentaires chaînées) à l'unité qui a telle adresse". Dans la suite, nous appelons tâche d'entrée-sortie l'ensemble des opérations nécessaires pour l'exécution d'une demande. Par extension, nous utilisons également ce terme de tâche pour désigner dans le système le descripteur qui représente cette demande.

Nous pouvons distinguer deux aspects principaux dans le rôle du superviseur d'entrée-sortie.

1) Aspect utilitaire : l'exécution du programme canal soumis nécessite un certain nombre d'opérations de service réalisées par le superviseur d'entrée-sortie. Citons par exemple l'initialisation du programme canal lui-même, l'analyse des conditions de fin d'opération, l'application éventuelle de procédures de traitement d'erreurs ou encore la transmission en retour des conditions d'exécution de la tâche.

2) Aspect allocation de ressources : pour que l'exécution d'un programme canal puisse être initialisée, il faut que les ressources nécessaires à cette exécution soient disponibles. Ces ressources sont de trois types.

- L'unité d'entrée-sortie impliquée dans l'opération (par exemple un disque), qui n'exécute qu'un ordre élémentaire à la fois ; cet ordre lui est transmis par l'unité de contrôle.
- L'unité de contrôle à laquelle est rattachée l'unité et qui pilote l'exécution d'un ordre élémentaire sur une unité. Selon la nature de l'unité de contrôle, une seule ou

plusieurs des unités qui lui sont rattachées peuvent fonctionner en même temps. Les unités de contrôle sont indépendantes les unes des autres.

- Le canal qui va chercher un à un en mémoire centrale les ordres élémentaires composant un programme canal et les transmet, par l'intermédiaire de l'unité de contrôle, à l'unité d'entrée-sortie. Un canal dit "sélecteur" ne peut piloter l'exécution que d'un seul programme canal à la fois alors qu'un canal dit "multiplexeur" peut piloter l'exécution de plusieurs programmes canal en parallèle (chacun s'adressant à l'une des différentes unités rattachées au canal). Deux canaux peuvent fonctionner simultanément, le seul conflit possible résultant des accès à la mémoire centrale. Ce conflit étant réglé par des règles de priorité câblées, nous pouvons, au niveau d'observation qui est le notre, considérer qu'il y a parallélisme effectif entre les opérations de deux canaux.

Laissons de côté l'aspect utilitaire du superviseur d'entrée-sortie, et considérons ce dernier comme un allocateur de ressources. Nous devons résoudre le même problème de conception que celui rencontré à propos de la gestion de la mémoire (cf. 2.41), à savoir : quelle structure donner à cet allocateur, où le placer (dans un processus à part, dans chaque processus, ...), comment communiquer avec lui, quel type de synchronisation employer ?

Si nous essayons de transposer la technique adoptée pour la gestion de la mémoire, c'est à dire l'inclusion de l'allocateur dans chacun des processus demandeurs, nous nous heurtons au problème suivant : alors que dans le cas de la gestion de la

mémoire l'événement significatif pour les demandes en attente est unique (c'est une opération de libération provenant d'un processus quelconque), dans le cas des entrées-sorties cet événement n'est pas aussi facile à isoler. En effet, pour qu'une tâche ait la possibilité d'être initialisée, il faut qu'à la fois l'unité, l'unité de contrôle et le canal concernés soient disponibles et nous savons que chacun de ces éléments peut être libéré individuellement. De plus, alors que l'on peut espérer, à partir d'une zone de mémoire devenue libre, satisfaire plusieurs demandes de mémoire en attente (ce qui justifie le réveil de l'ensemble des processus demandeurs), dans la plupart des cas, l'apparition d'une condition "chemin disponible" ne permet de prendre en compte qu'une seule opération d'entrée-sortie, cette prise en compte rendant à nouveau le chemin indisponible. Prenons par exemple le cas des entrées-sorties sur un terminal. Une telle unité a sa propre unité de contrôle (en fait, une unité de contrôle à n voies d'accès dessert n terminaux) rattachée à un canal multiplexeur. La seule condition interdisant l'initialisation d'une tâche d'entrée-sortie sur un terminal est donc la condition "unité occupée", qui disparaît lorsque l'opération en cours se termine. A ce moment là, une seule tâche (par exemple la première se trouvant dans la file d'attente) peut être initialisée. Si nous prenons maintenant l'exemple des tâches en attente pour des unités attachées à un canal sélecteur, la libération du canal permet d'initialiser une seule tâche impliquant un transfert, même s'il existe pour ce canal plusieurs tâches pour lesquelles le chemin est libre (unité de contrôle et unité disponibles également). Par contre, s'il existe des tâches n'utilisant pas le chaînage des commandes et n'impliquant pas de transfert, plusieurs d'entre elles peuvent être prises en compte car chacune libère le canal

immédiatement et est pilotée ensuite par la seule unité de contrôle. Nous voyons donc que le réveil de toutes les tâches en attente lors de l'apparition d'une condition "chemin disponible", non seulement est inutile dans la plupart des cas, mais de plus ne permet pas d'optimiser l'emploi des ressources d'entrée-sortie puisqu'alors l'ordre de réémission des demandes est l'ordre dans lequel le processeur est alloué aux processus. Cet ordre n'est pas forcément celui permettant l'utilisation optimale des ressources.

Nous avons donc rejeté la solution qui consiste à placer l'allocateur dans chaque processus et avons choisi de mettre en place un mécanisme centralisé de gestion des tâches en attente.

Devons nous englober l'ensemble du superviseur d'entrée-sortie dans un processus à part ou bien certaines parties peuvent-elles être incluses dans les processus appelants ? Pour répondre à cette question, il nous faut examiner les différentes étapes de la vie d'une tâche d'entrée-sortie (figure 12) et les types de synchronisation souhaitables entre un processus et une tâche d'entrée-sortie qu'il veut faire exécuter.

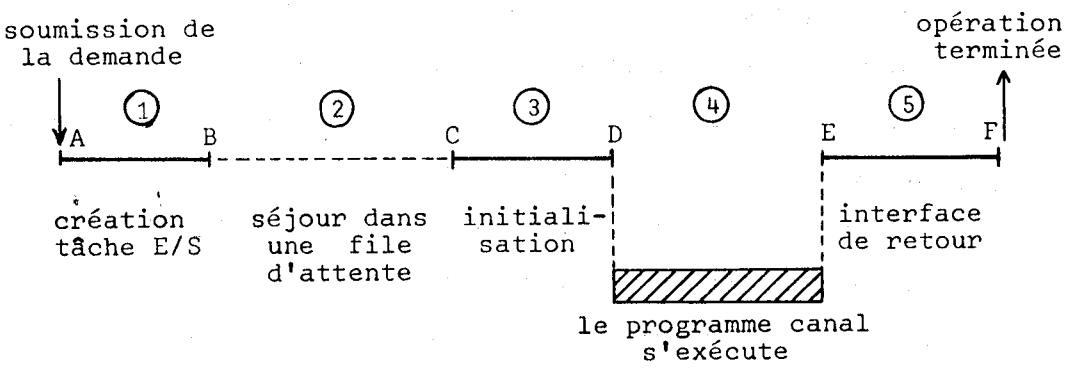


Figure 12. Diverses étapes de la vie d'une tâche d'entrée-sortie

En A, la demande est soumise au superviseur d'entrée-sortie. Considérons pour l'instant que cela est réalisé par l'appel d'une

procédure à qui l'on transmet en paramètres les caractéristiques de la demande. La partie ① consiste à prendre en compte la tâche. En B, deux situations peuvent se présenter : ou bien la tâche peut être exécutée immédiatement (toutes les ressources nécessaires sont disponibles), ou bien elle doit être placée dans une file d'attente. Dans le premier cas, la partie ② représentant le séjour en file d'attente est absente. En C, l'opération est initialisée (exécution de l'instruction SIO). Elle est maintenant sous le contrôle du canal (partie ④) et ne nécessitera l'intervention du processeur qu'en E, matérialisé par une interruption de fin d'entrée-sortie. La partie ⑤ consiste principalement à recueillir certaines informations d'état fournies par le canal en fin d'opération pour les rendre accessibles au demandeur de l'entrée-sortie.

Voyons maintenant les besoins des processus composant le système du point de vue de leur synchronisation avec les demandes d'entrée-sortie qu'ils soumettent.

Dans certains cas, un processus ne peut poursuivre son exécution tant que l'opération demandée n'est pas complètement terminée. Nous appelons les tâches d'entrée-sortie correspondantes des tâches synchrones. Une soumission de tâche synchrone contient donc implicitement une demande de mise en attente du processus, qui ne sera de nouveau prêt qu'en F, une fois la tâche complètement exécutée.

Exemple.

Les processus de l'interface générateur des CMS virtuels qui pilotent des opérations d'entrée-sortie sur disque ou bande en provenance d'un CMS (cf. 2.523) utilisent des tâches synchrones.

Dans d'autres cas, le processus demandeur peut poursuivre son exécution sans attendre la fin de l'opération. Nous appelons les tâches d'entrée-sortie correspondantes des tâches asynchrones. Le processus qui soumet une tâche asynchrone peut reprendre son exécution dès le point B ou le point D de la figure 12 selon que l'initialisation ③ a été différée ou a pu être effectuée immédiatement (partie ② non nécessaire).

Exemple.

Le processus de l'interface générateur des CMS virtuels qui pilote une écriture de message sur terminal en provenance d'un CMS (cf. 2.525) utilise une tâche asynchrone car il n'est pas nécessaire d'attendre la fin de l'opération (une zone tampon interne à GMS est utilisée) pour que ce processus se termine et rende possible la reprise du contrôle par le CMS concerné.

Il eut été possible, et nous l'avons fait dans certains cas, de n'employer que des tâches synchrones même lorsque l'on veut poursuivre l'exécution du processus sans attendre la fin de l'opération. Il suffit en effet qu'un processus P1 voulant exécuter une entrée-sortie sans attente crée un processus parallèle P2 qui pilotera cette entrée-sortie en utilisant une tâche synchrone. Rappelons que la création d'un processus (cf. 2.3341) consiste seulement à initialiser un descripteur de processus et à le rattacher à la liste des processus prêts. Les deux diagrammes suivants (figure 13) illustrent ces deux possibilités : dans la partie gauche le processus P1 soumet une tâche asynchrone, dans la partie droite il crée le processus P2 qui soumet une tâche synchrone.

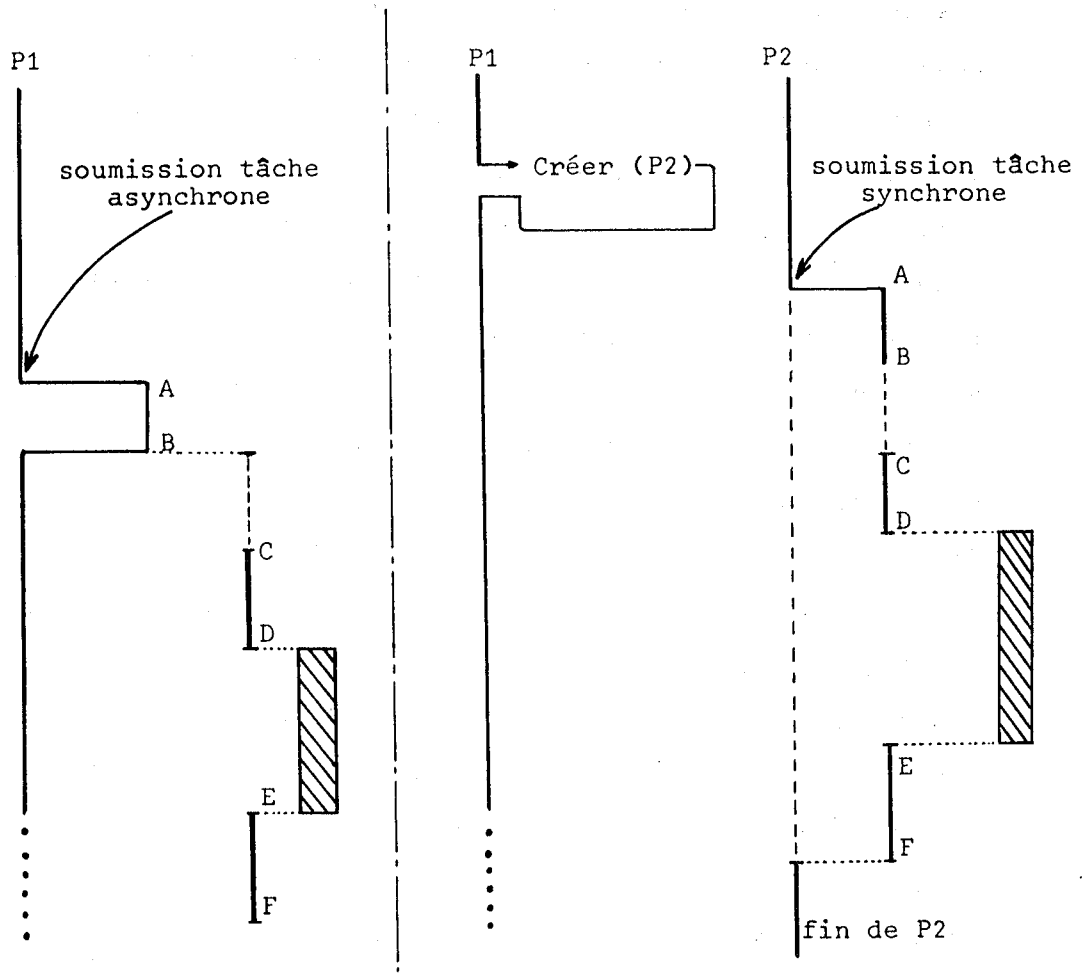


Figure 13. Tâche asynchrone et processus parallèle

Cette dernière approche, qui consiste à n'utiliser que des tâches synchrones pilotées par un autre processus que le processus émetteur, bien qu'a priori plus élégante, n'est pas dans tous les cas la plus efficace. Essayons d'expliquer pourquoi. Si nous reprenons le schéma donnant les étapes successives par lesquelles passe une tâche d'entrée-sortie au cours de son existence, nous constatons qu'au point E se situe l'interruption signalant la fin de l'opération. Si nous ne considérons que la demande en cours, il ne reste qu'à réaliser l'étape ⑤, c'est à dire la transmission

des informations en retour au processus demandeur. Mais si nous nous intéressons à l'ensemble des tâches d'entrée-sortie en attente, nous constatons que ce point E correspond à une libération des ressources utilisées et que c'est précisément à partir de ce point que peut être envisagée l'initialisation d'une autre tâche. L'étape (5) n'est donc pas suivie immédiatement de la réactivation du processus demandeur (dans le cas d'une opération synchrone) mais de l'exécution de la partie du superviseur d'entrée-sortie que nous appelons "relance de l'activité du canal" et qui consiste à initialiser une tâche en attente des ressources libérées.

Prenons alors comme exemple le cas réel suivant : lorsque le catalogue des fichiers de GMS (cf. 2.431) est modifié en mémoire centrale, une copie de ce catalogue modifié est écrite sur disque pour permettre un redémarrage après un incident éventuel. Ce catalogue occupe un certain nombre d'enregistrements sur le disque système de GMS, son écriture nécessite donc plusieurs demandes d'entrée-sortie successives. Si le processus chargé de l'écriture utilise des tâches synchrones, le superviseur d'entrée-sortie, lors de l'interruption signalant la fin d'écriture de l'un des enregistrements, trouve la file d'attente vide (en supposant qu'il n'y a pas d'autres demandes que celles relatives à l'écriture du catalogue). Il n'y a pas alors de relance de l'activité du canal mais le réveil du processus qui va alors soumettre la demande correspondant à l'enregistrement suivant. Le diagramme d'exécution est donné par la figure 14.

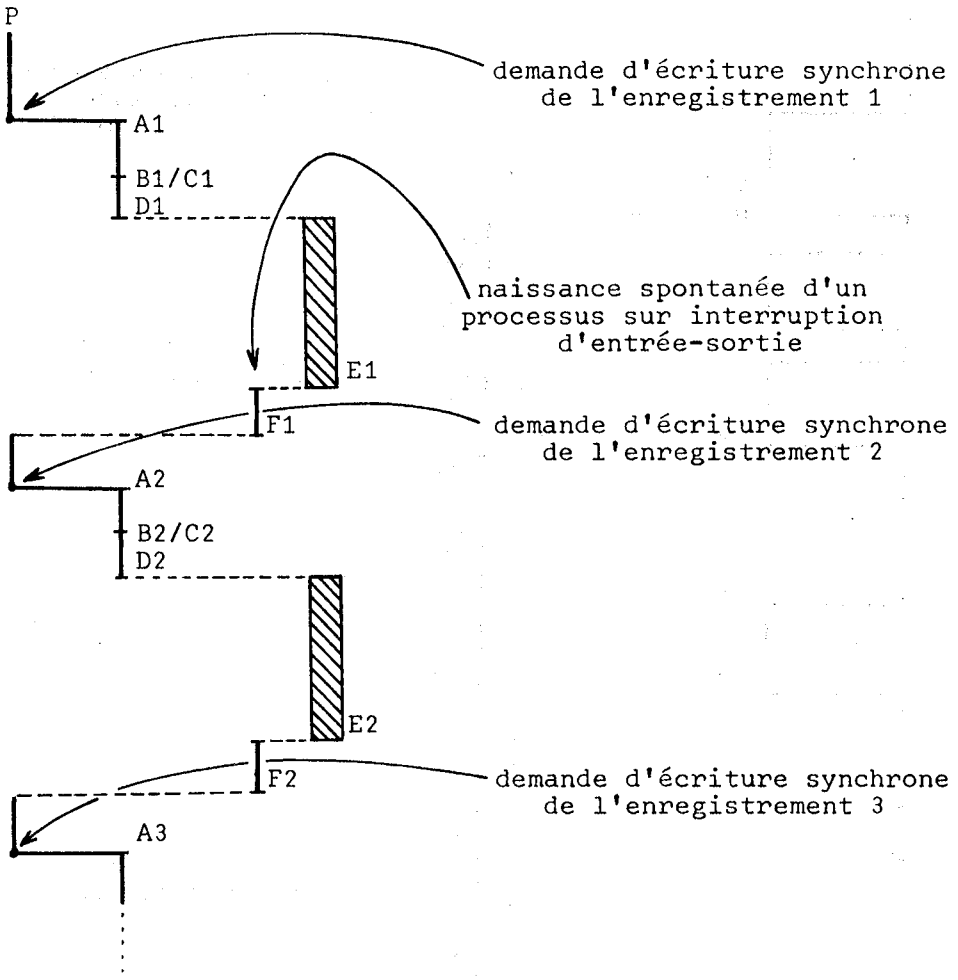


Figure 14. Ecriture de n enregistrements par tâches synchrones

Si maintenant le processus utilise des tâches asynchrones, la première sera vraisemblablement lancée immédiatement (si les ressources sont disponibles) et les suivantes mises en file d'attente. Lors d'une interruption signalant une fin d'opération, le superviseur d'entrée-sortie peut immédiatement initialiser la prochaine demande en attente. Le diagramme d'exécution correspondant est donné dans la figure 15.

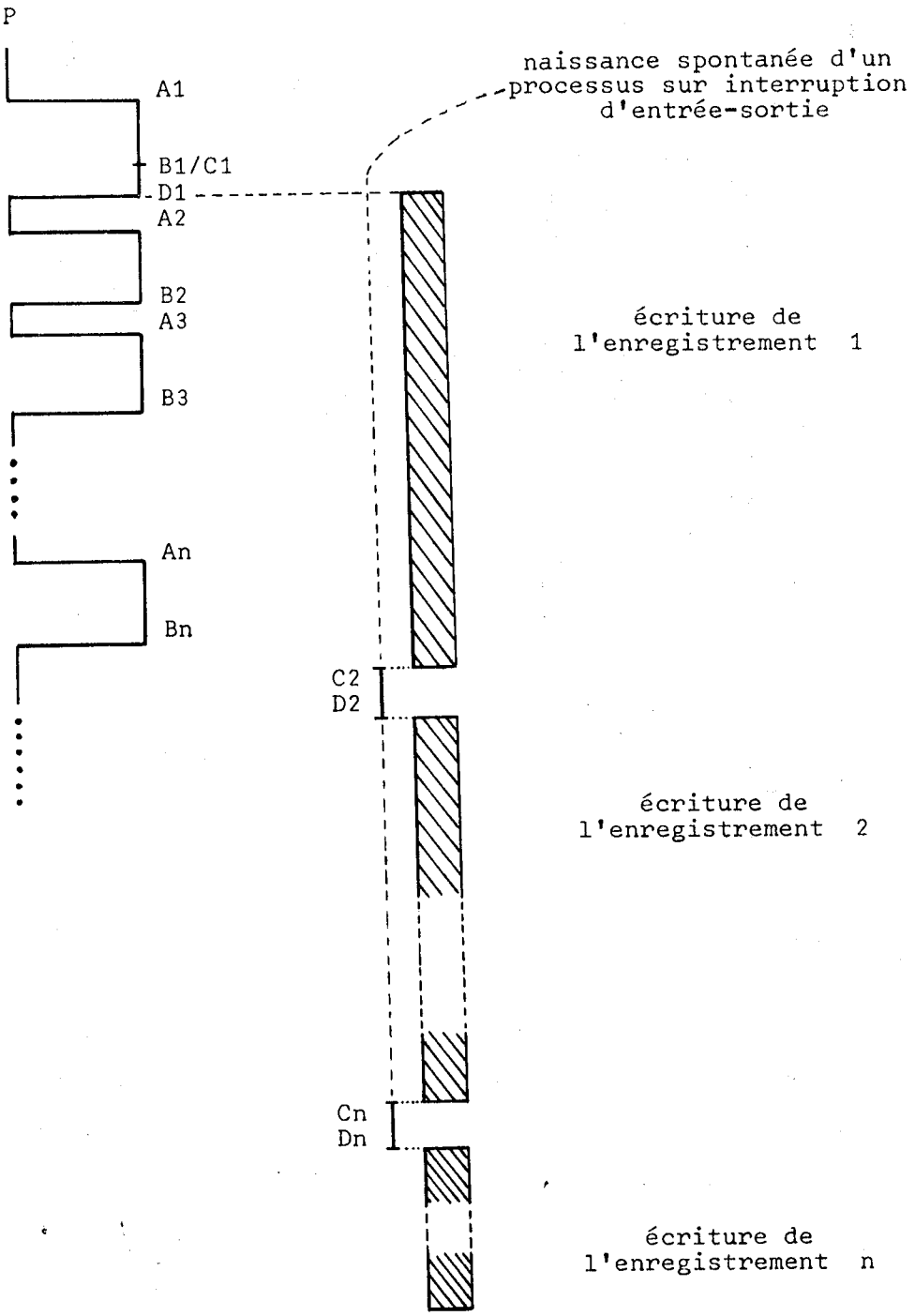


Figure 15. Ecriture de n enregistrements par tâches asynchrones

Cette deuxième solution permet de réoccuper le canal beaucoup plus rapidement après une libération. De plus, elle nécessite moins de changements de contexte ; nous pouvons en effet remarquer que dans le premier cas, en chacun des points Fi se situe une réactivation du processus demandeur alors que dans le second cas l'ensemble des tâches est soumis au moyen d'une seule activation du processus P.

La troisième possibilité consiste, comme nous l'avons vu, à créer des processus intermédiaires pilotant des tâches synchrones. Il faut, dans notre exemple, que le processus P crée n processus P1, P2, ... Pn, chaque processus Pi étant chargé d'écrire l'enregistrement i. Lorsque P, après avoir créé les Pi se termine, le Contrôleur active successivement P1, P2, ... Pn. Chaque Pi, arrivé au point Bi ou Ci, est placé en attente de fin d'opération. Nous obtenons donc, du point de vue de l'exploitation de la file d'attente, le même résultat qu'avec la solution du processus unique avec tâches asynchrones, mais le coût en temps de processeur est nettement plus élevé. Il faut en effet n créations de processus qui supposent l'acquisition de mémoire libre pour contenir leur vecteur d'état initial, et 2n changements de contexte (une mise en attente et un réveil pour chacun des Pi).

Nous pouvons retenir de cette discussion la conclusion principale suivante : les outils de synchronisation mis en place peuvent être employés de manières très variées selon la structure donnée aux composants du système. Plutôt que d'établir les relations entre processus pour l'ensemble du système sur un modèle unique et rigide, il est préférable d'adapter l'emploi de ces outils à chaque cas particulier rencontré.

Nous pouvons maintenant répondre à la question que nous nous étions posée, à savoir, dans quel processus s'exécute le superviseur d'entrée-sortie ? Les diagrammes d'exécution que nous venons de donner (figures 14 et 15) contiennent en eux-mêmes cette réponse.

1) La partie du superviseur d'entrée-sortie activée sur appel d'un processus s'exécute dans ce processus jusqu'à ce que la tâche d'entrée-sortie soit placée dans une file d'attente ou jusqu'à ce que le programme canal soit initialisé, selon que cette initialisation n'est pas ou est possible immédiatement (point B ou D figure 12).

S'il s'agit d'une tâche synchrone, la mise en attente a lieu en B ou D. Nous utilisons pour cela un événement lié à cette tâche.

S'il s'agit d'une tâche asynchrone, en B ou D le processus reprend son cours normal. Il n'y a eu dans ce cas activation d'aucun autre processus.

2) La partie du superviseur d'entrée-sortie qui prend le contrôle sur interruption s'exécute dans un processus à part, différent de celui qui a soumis la tâche qui se termine. Nous avons vu (cf. 2.3341) qu'une interruption entraîne la naissance spontanée d'un processus. Ce processus réveille le processus émetteur dans le cas d'une tâche synchrone et se contente de faire disparaître toute trace de la tâche terminée si c'est une tâche asynchrone (elle n'intéresse aucun processus). Dans les deux cas, la relance de l'activité du canal est tentée, puis ce processus se termine.

Nous pouvons alors donner les deux diagrammes montrant le traitement d'une tâche synchrone (figure 16 a) et d'une tâche

asynchrone (figure 16 b). Nous notons dans ces diagrammes par "Call IOSUP" l'appel de la procédure du superviseur d'entrée-sortie à laquelle est soumise la tâche et par "ACT" et "INACT" l'emploi des primitives de synchronisation agissant sur le processus demandeur. Pour simplifier les schémas, mous nous sommes placés dans le cas où la demande peut être lancée immédiatement ; s'il n'en est pas ainsi, la partie BD des diagrammes disparaît et est reportée lors d'une relance de l'activité du canal effectuée sur interruption.

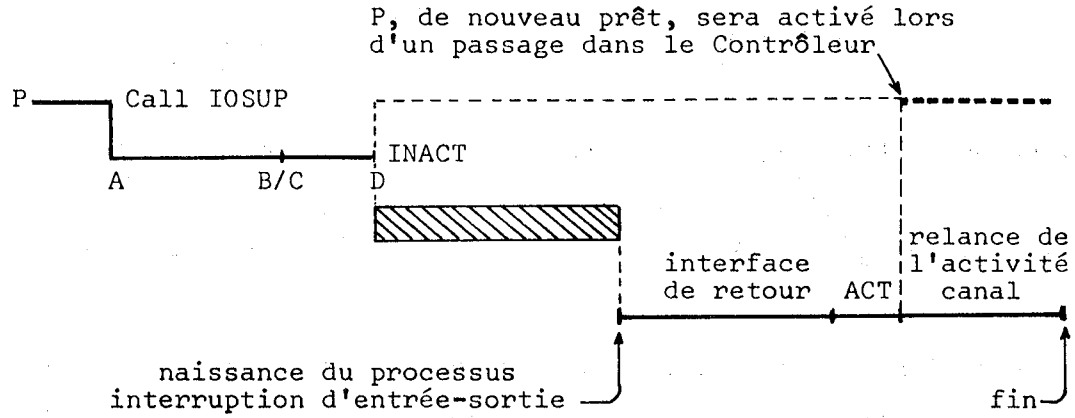


Figure 16 a. Traitement d'une tâche synchrone

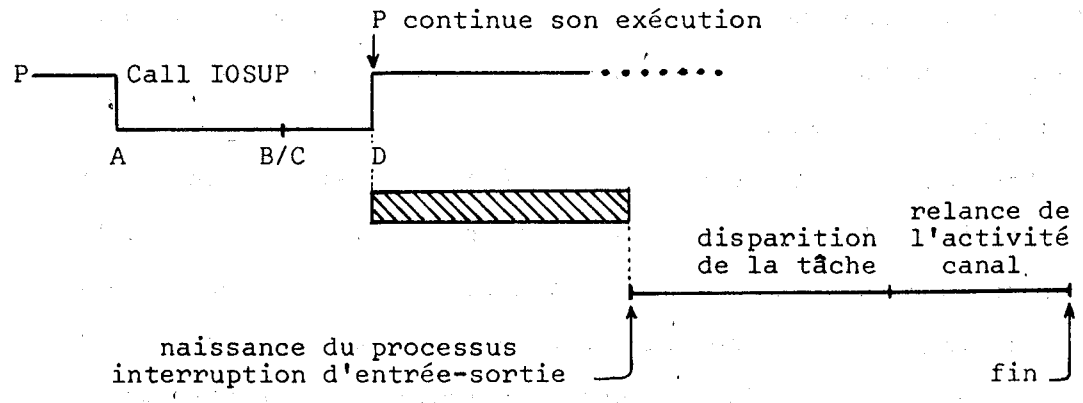
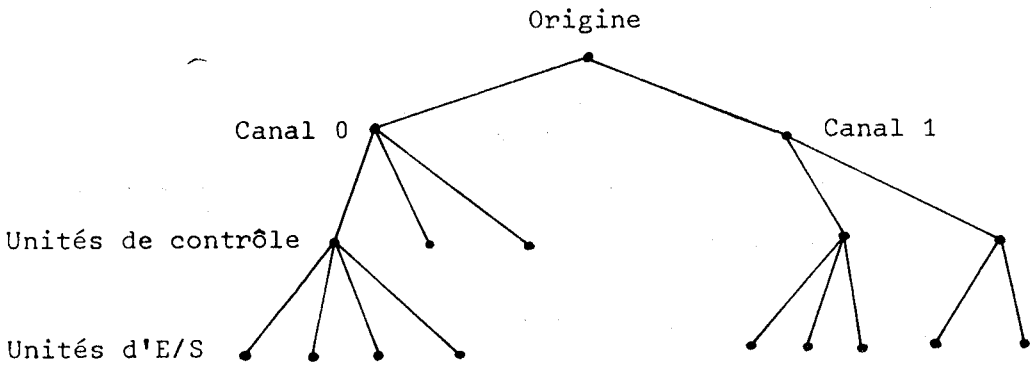


Figure 16 b. Traitement d'une tâche asynchrone

2.422 Structures de données associées aux entrées-sorties

2.4221 Configuration

Nous utilisons une représentation en arbre faisant apparaître les relations de dépendance entre unités, unités de contrôle et canaux.



Lors d'une interruption d'entrée-sortie, le canal fournit au processeur l'adresse de l'unité qui est à l'origine de l'interruption. Cette adresse est de la forme 'abc' où a est le numéro du canal, b celui de l'unité de contrôle et c celui de l'unité. Il faut pouvoir retrouver à partir de cette adresse le noeud correspondant à l'unité. Ce noeud permet d'accéder au descripteur de l'unité.

Lors de la relance de l'activité d'un canal, il faut pouvoir accéder à la liste des unités de contrôle qui y sont rattachées, et pour chacune d'entre elles à la liste des unités contrôlées.

La figure 17 donne la représentation de la configuration d'entrée-sortie de l'ordinateur sur lequel GMS a été implanté [Finet 73 b]. Nous voyons par exemple que sur le canal 1 sont connectées les unités de contrôle 3 et 8. A l'unité de contrôle 3

sont connectées 8 unités (qui sont des disques d'adresses 130 à 137) alors qu'à l'unité de contrôle 8 sont connectées 2 unités (qui sont des bandes d'adresses 180 et 181).

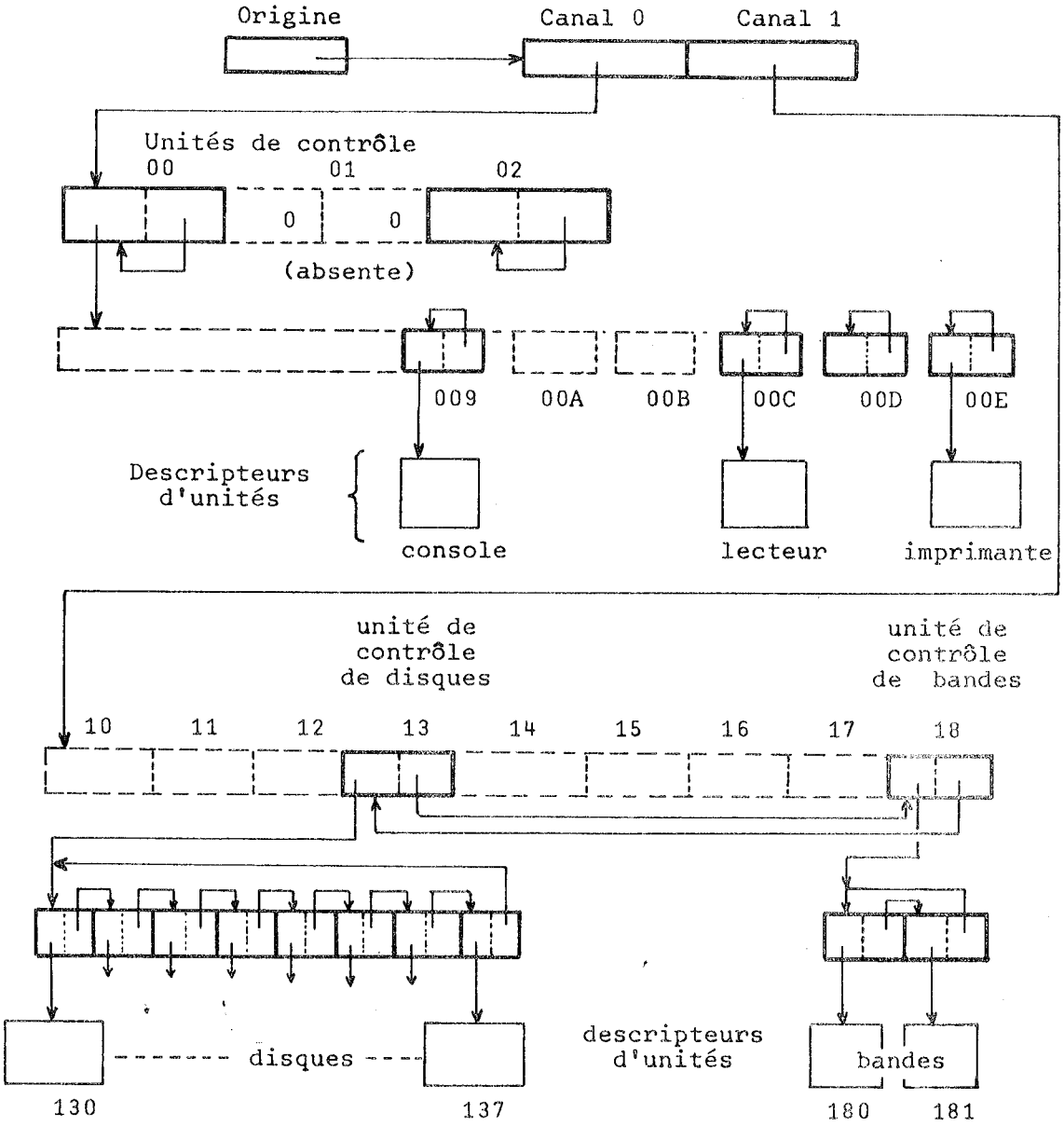


Figure 17. Représentation de la configuration d'E/S dans GMS

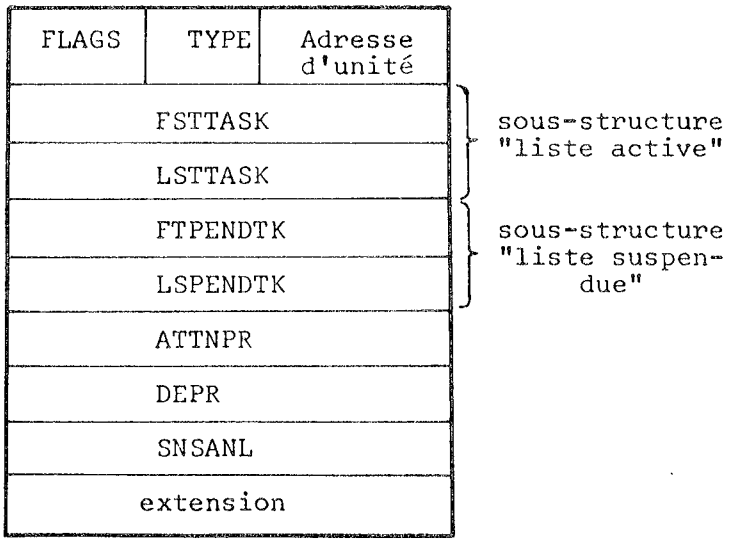
2.4222 Descripteur d'unité

Les actions à entreprendre pour exécuter une opération d'entrée-sortie dépendent dans une certaine mesure de l'unité concernée. Par exemple, les informations fournies par le canal sur interruption signalant la fin d'une opération ne sont pas les mêmes pour un disque et pour un terminal. Les traitements d'erreurs sont également spécifiques de chaque type d'unité. Pour ces raisons, nous avons adopté un fonctionnement "dirigé par tables" en plaçant dans le descripteur d'une unité les adresses des procédures à appeler chaque fois qu'un traitement particulier doit être appliqué.

Les tâches d'entrée-sortie relatives à une même unité sont rattachées au descripteur de cette unité. Nous nous réservons la possibilité d'utiliser deux listes de tâches par unité :

- une liste active qui est celle que le superviseur d'entrée-sortie examine lors de la relance de l'activité ;
- une liste suspendue qui correspond à des tâches dont l'exécution doit être différée. Cette liste est utilisée en particulier lors du changement d'environnement de l'utilisateur (cf. 2.62).

La figure 18 représente la structure adoptée pour ce descripteur d'unité.



FLAGS état de l'unité :
- occupée
- non prête
- analyse en cours
- utilise les services du canal
- arrêt d'opération en cours
- analyse en attente

TYPE type de l'unité :
- disque
- bande
- lecteur, perforateur, imprimante, console opérateur 1052
- terminal 2741

FSTTASK repère la première tâche active

LSTTASK repère la dernière tâche active

FTPENDTK repère la première tâche suspendue

LSPENDTK repère la dernière tâche suspendue

ATTNPR procédure de traitement des interruptions asynchrones

DEPR procédure de traitement des interruptions synchrones

SNSANL procédure de traitement des informations d'analyse

extension informations complémentaires propres à l'unité

Figure 18. Descripteur d'unité d'entrée-sortie

2.4223 Descripteur de tâche d'entrée-sortie

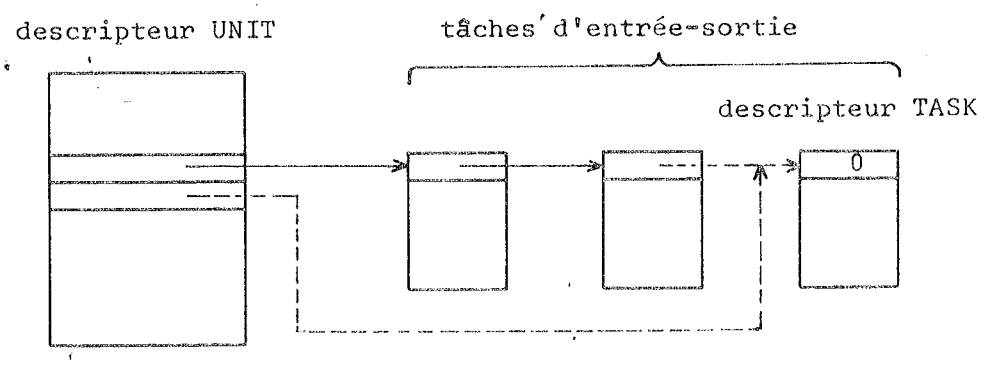
Un processus voulant soumettre une tâche d'entrée-sortie exécute l'appel de procédure

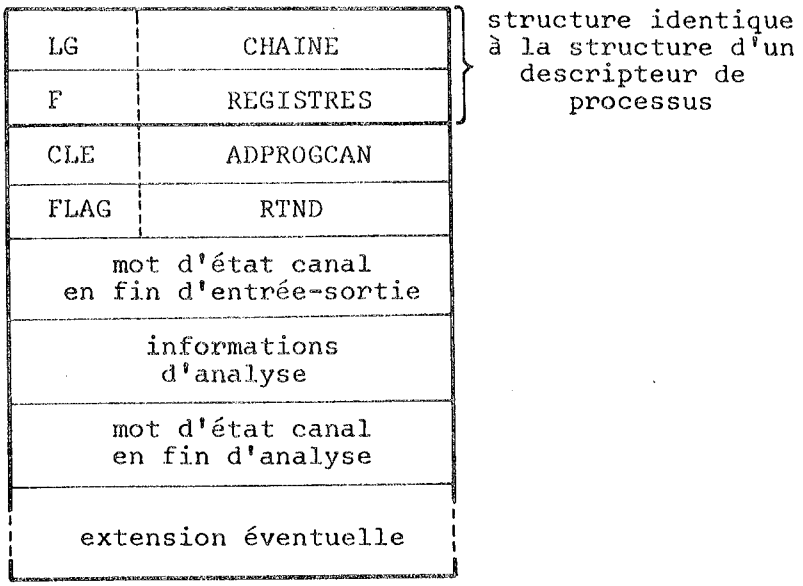
Call IOSUP (UNIT , TASK)

où UNIT désigne le descripteur de l'unité à laquelle s'adresse l'opération et TASK désigne un descripteur de tâche créé auparavant par le processus. Une partie du descripteur de tâche est initialisée par le processus (adresse du programme canal, indication de tâche synchrone ou asynchrone, ...), le reste est utilisé par le superviseur d'entrée-sortie pour placer ce descripteur dans une liste, pour noter l'état d'avancement de la tâche ou pour transmettre des informations en retour au processus appelant.

La figure 19 représente la structure adoptée pour ce descripteur de tâche.

Lorsqu'il est appelé par Call IOSUP (UNIT,TASK), le superviseur d'entrée-sortie chaîne le descripteur de tâche TASK au descripteur d'unité UNIT. C'est la partie (1) du diagramme dans la figure 12, notée "création tâche E/S", et qui se résume à cette mise en liste du descripteur de tâche fourni.





LG	longueur du descripteur
CHAINE	pointeur de chaînage dans les files d'attente
F	indique que ce descripteur est à libérer par le Contrôleur
REGISTRES	adresse de la zone de sauvegarde des registres
CLE	clé de protection à utiliser lors de l'exécution de l'instruction SIO
ADPRGCAN	adresse du programme canal
FLAG	options de la tâche et état d'avancement : - tâche autocorrectrice - descripteur à libérer par RTND - prétraitement achevé - tâche terminée - mot d'état canal rangé sur SIO - tâche synchrone
RTND	adresse d'une procédure spéciale

Figure 19. Descripteur de tâche d'entrée-sortie

Une caractéristique importante du descripteur de tâche est que la partie de tête a la même structure qu'un descripteur de

processus (cf. 2.3331). Ce choix résulte du fait que l'on veut pouvoir appliquer les mêmes primitives de synchronisation à une tâche d'entrée-sortie (en fait, par son intermédiaire, au processus émetteur) que celles appliquées à un processus.

Remarque. Le fait que les processus qui soumettent des tâches d'entrée-sortie aient à connaître l'adresse du descripteur d'unité ne soulève aucune difficulté. Les descripteurs des unités de la configuration sont définis dans la procédure qui les contient et toute autre procédure peut utiliser les noms de ces descripteurs en tant que symboles externes. L'avantage, par rapport à l'utilisation directe de l'adresse de l'unité, réside dans le fait qu'il n'est pas nécessaire d'effectuer la recherche du descripteur à partir de l'arbre décrivant la configuration (cf. 2.4221).

2.423 Synchronisation

Nous avons vu que l'exécution d'une tâche synchrone nécessite l'emploi des primitives de synchronisation en deux points (figure 16 a) :

- mise en attente du processus émetteur (point D de la figure) après initialisation ou mise en file d'attente de la tâche ;
- réveil du processus émetteur pár le processus d'interruption d'entrée-sortie, après que le descripteur de tâche ait été mis à jour (interface de retour).

La sous-structure "liste active" dans le descripteur d'unité (figure 18) est identique à un descripteur d'événement. Nous utilisons cette sous-structure comme descripteur de l'événement "tâche terminée".

La tâche soumise a été chaînée à la fin de la liste active ; la mise en attente consiste alors simplement à placer dans la zone REGISTRES du descripteur de tâche (figure 19) l'adresse de la zone de sauvegarde de la procédure appelante (celle qui exécute "Call IOSUP") et à retourner au Contrôleur.

Le réveil du processus émetteur par le processus d'interruption d'entrée-sortie s'écrit

ACT (FSTTASK (UNIT))

où nous notons symboliquement par FSTTASK (UNIT) l'adresse de la sous-structure liste active dans le descripteur d'unité UNIT. Cette primitive (cf. 2.3343) retire le premier descripteur de la liste attachée à l'événement FSTTASK (UNIT) et l'accroche à la liste des processus prêts. Nous vérifions que le mécanisme est cohérent, puisque la tête du descripteur de tâche ainsi rajouté à la liste des processus prêts représente bien le descripteur du processus qui était en attente de cette entrée-sortie. Lors de l'activation du processus, le Contrôleur ne distingue pas un tel descripteur d'un descripteur classique de processus prêt.

Remarque. Une procédure qui appelle IOSUP doit posséder, comme toute procédure en appelant une autre, une zone de sauvegarde pour les registres. La sauvegarde est effectuée par IOSUP lors de son prologue. On pourrait penser que le processus est placé en attente à l'intérieur de IOSUP en utilisant la propre zone de sauvegarde de cette dernière. Il n'en est rien car cela imposerait à IOSUP d'être réentrante, ce qui entraînerait lors de chaque appel l'acquisition dynamique d'une zone de sauvegarde. Or, le contexte de la procédure IOSUP au moment où elle effectue la mise en attente ne sera pas réutilisé : en effet, toutes les informations concernant la tâche se trouvent dans le descripteur et cette tâche

sera ensuite prise en charge par un autre processus (le processus d'interruption d'entrée-sortie). Il est donc inutile de placer l'attente à l'intérieur de IOSUP puisque la seule opération à effectuer après le réveil serait un retour à la procédure appelante. Nous utilisons donc directement la zone de sauvegarde de cette dernière dont l'adresse est placée dans la partie REGISTRES du descripteur de tâche avant de retourner au Contrôleur. Notons simplement qu'une remise en ordre des contenus de cette zone de sauvegarde est nécessaire en raison de l'ordre de rangement des registres utilisé par un prologue, qui diffère de l'ordre utilisé pour les descripteurs de processus.

2.424 Intervention pendant le déroulement d'une tâche asynchrone

Nous avons mis en évidence en 2.421 l'intérêt qu'il y avait dans certains cas à employer des tâches asynchrones. Cependant, lorsqu'un processus soumet une tâche asynchrone cette dernière lui échappe complètement. Or il est rare qu'un processus puisse se désintéresser totalement d'une entrée-sortie qu'il a soumise. Citons deux exemples, parmi d'autres, empruntés à GMS.

Exemple 1.

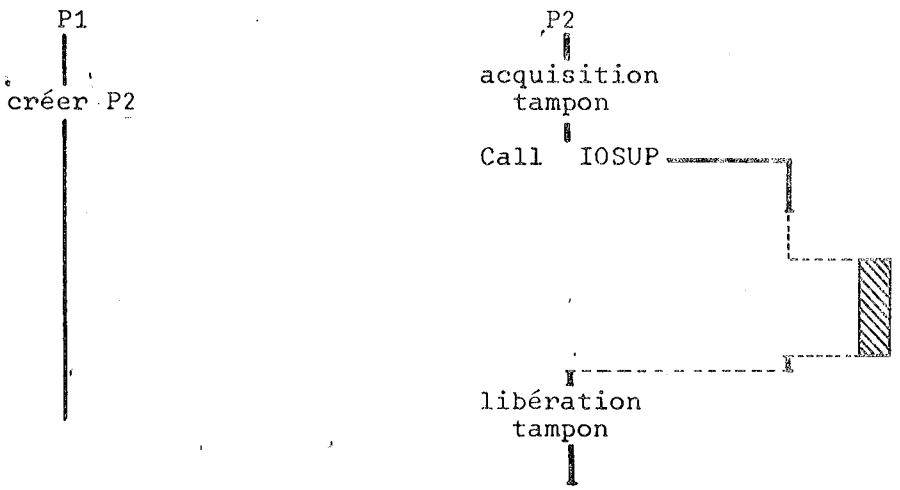
Le processus de l'interface générateur des CMS virtuels qui pilote une opération d'écriture sur terminal utilise une tâche asynchrone. Il acquiert une zone tampon dans laquelle il place le message à écrire, construit et soumet la tâche asynchrone correspondante. Qui va rendre le tampon à la mémoire libre de GMS une fois l'opération terminée ? Nous pourrions bien sur utiliser la solution de fortune qui consisterait à faire exécuter ce travail par le superviseur d'entrée-sortie lui-même. Nous rejetons

cette solution car elle revient à inclure dans la logique du superviseur d'entrée-sortie des éléments étrangers qui n'ont rien à y faire.

Exemple 2.

Nous avons donné en 2.421 l'exemple de la réécriture du catalogue des fichiers de GMS au moyen de tâches asynchrones. Un verrou est associé à ce catalogue de manière à ce qu'il ne soit pas modifié en mémoire par un processus B pendant qu'un processus A est en train de l'écrire [Faure 71]. Ce verrou doit être enlevé et un éventuel processus en attente réveillé lorsque le dernier enregistrement a été écrit. Qui va s'occuper de ce verrou ? Là encore, inclure de telles opérations dans la logique du superviseur d'entrée-sortie n'est pas une solution envisageable sérieusement.

Devons-nous alors revenir à la solution qui consiste, pour un processus P1 voulant effectuer une opération d'entrée-sortie sans attente, à créer un processus parallèle P2 utilisant une tâche synchrone ? Le schéma suivant illustre l'utilisation d'une telle technique dans le cas de l'exemple 1 :



Nous proposons dans GMS la solution suivante.

Dans tout descripteur de tâche d'entrée-sortie, nous pouvons inclure le point d'entrée d'une procédure spéciale (c'est la zone notée RTND dans la figure 19). Par convention, lorsque le processus d'interruption d'entrée-sortie s'aperçoit qu'une tâche est terminée, il appelle cette procédure si elle est mentionnée, en lui fournissant l'adresse du descripteur de tâche. C'est donc dans une telle procédure que nous pouvons placer la libération de la zone tampon de notre exemple 1 ou le traitement du verrou de notre exemple 2. Nous pouvons nous demander comment la procédure RTND connaît par exemple l'adresse et la longueur du tampon à libérer. Si nous nous reportons à la figure 19, nous voyons que le descripteur de tâche possède une extension pour contenir toute information jugée utile par le processus qui le crée. Dans notre exemple 1, le processus rajoute au descripteur la longueur et l'adresse du tampon. L'adresse du descripteur de tâche étant transmise à la procédure RTND par le processus d'interruption d'entrée-sortie, cette procédure a ainsi accès à ces éléments.

Nous pouvons justifier l'emploi d'une telle solution dans le cas de GMS compte tenu des autres options choisies relativement au mode d'exécution des processus. Nous pouvons également fixer les limites de validité de cette solution dans des contextes différents. La première remarque, fondamentale, est que la procédure RTND ainsi activée par le processus d'interruption d'entrée-sortie s'exécute dans ce processus et non dans le processus ayant soumis la tâche. Cela veut dire que son espace d'adressage est celui du processus d'interruption d'entrée-sortie et que ses prérogatives sont également celles de ce processus.

Dans GMS cette solution est satisfaisante puisque tous les processus utilisent un espace d'adressage commun, s'exécutent avec les mêmes prérogatives (tous fonctionnent en mode superviseur avec possibilité d'un contrôle complet sur toutes les ressources du système) et sont également prioritaires vis à vis des événements asynchrones (exécution avec interruptions interdites). De plus, nous avons délibérément choisi d'accorder le même degré de confiance à l'ensemble des procédures composant le système, ce qui nous conduit à considérer que l'inclusion du "corps étranger" qu'est la procédure RTND dans le déroulement du processus d'entrée-sortie ne perturbe pas ce dernier.

Cette dernière condition n'est satisfaite que si les procédures RTND respectent un certain nombre de conventions d'écriture. En particulier, une procédure RTND ne doit pas placer son processus en attente ni appeler d'autres procédures risquant de conduire au même résultat. Cela reviendrait en effet à introduire une attente à l'intérieur du superviseur d'entrée-sortie, à l'insu de ce dernier, ce qui, au minimum, empêcherait l'utilisation optimale des ressources d'entrée-sortie. Les possibilités des procédures RTND sont donc limitées ; cependant, elles ne se sont à aucun moment avérées insuffisantes lors de la construction de GMS.

Si maintenant nous envisageons le cas d'un système dans lequel les processus s'exécutent dans des espaces d'adressage disjoints ou avec des prérogatives différentes, la solution que nous avons adoptée dans GMS devient très criticable.

Supposons par exemple que chaque processus dispose d'un espace virtuel propre, mis en oeuvre par des mécanismes de pagination. Lorsque le processus d'interruption d'entrée-sortie a le contrôle, l'espace qu'il peut adresser est défini par la table de pages

courante. La procédure RTND doit adresser des informations créées dans l'espace d'adressage du processus ayant soumis la tâche d'entrée-sortie. Il faudrait donc changer de table de pages lors de l'appel de la procédure RTND et restaurer la table courante au retour.

Supposons maintenant qu'une hiérarchie soit établie entre les processus du système, certains s'exécutant avec plus de prérogatives que d'autres. Si le concepteur du système a choisi de faire s'exécuter le processus d'interruption d'entrée-sortie en mode superviseur (il doit pouvoir émettre les instructions privilégiées d'entrée-sortie) et les autres processus en mode programme, l'inclusion d'une procédure RTND dans le corps du processus d'entrée-sortie autorise un processus "programme" à faire exécuter des actions "superviseur". Là encore, il est en principe possible d'envisager un changement de prérogatives du processus au moment où il appelle la procédure RTND, mais c'est une solution qui met en oeuvre des mécanismes complexes et par là même fait décroître le rendement du système.

Remarque. Une solution analogue aux procédures RTND de GMS existe dans le système OS/360 [IBM 70]. En certains points de l'exécution d'une requête d'entrée-sortie, le superviseur d'entrée-sortie peut donner le contrôle à des sous-programmes appelés "appendages" dont les noms sont accessibles à partir des blocs de contrôle décrivant la requête. Nous avons personnellement utilisé cette technique dans le système LCMS [Bellino 69] pour gérer des sous-tâches à l'intérieur d'une tâche conversationnelle. Ces sous-programmes reçoivent le contrôle avec les prérogatives du superviseur d'entrée-sortie, c'est à dire qu'ils s'exécutent en mode superviseur avec la clé zéro de protection de mémoire (autorisant

l'accès à la totalité de la mémoire) et toutes interruptions interdites. Une faute de programmation de la part d'un utilisateur employant la technique des "appendages" peut entraîner une destruction du système. Les contraintes imposées aux utilisateurs, notamment l'obligation de placer ces sous-programmes dans une bibliothèque spéciale, ne sont pas toujours suffisantes pour se garantir contre de telles fautes.

2.425 Interruption de type asynchrone

Nous appelons ici interruption asynchrone une interruption d'entrée-sortie ne résultant pas de l'exécution d'une tâche d'entrée-sortie mais provenant, au contraire, d'un événement externe au système. Cet événement externe est créé par l'opérateur qui fait passer de l'état "non prêt" à l'état "prêt" une unité d'entrée-sortie (disque, bande, lecteur de cartes, perforateur, imprimante) ou qui presse la touche "Attention" sur la console maîtresse de l'installation.

S'il existe des tâches chaînées au descripteur de l'unité correspondante, la première d'entre elles attendait le passage de l'unité à l'état "prêt" pour être initialisée. C'est par exemple le cas des tâches destinées à l'imprimante, lorsqu'elles sont soumises au moment où l'opérateur change le papier. A la suite de l'interruption asynchrone signalant l'état "prêt", la "relance de l'activité canal" se produit et la première de ces tâches peut alors être exécutée.

Dans d'autres cas, le passage à l'état "prêt" est un moyen donné à l'opérateur pour initialiser une activité particulière du système. Cette solution est utilisée dans GMS pour gérer le lecteur de cartes : lorsque l'opérateur veut faire lire un paquet

de cartes par le système, il place ces cartes dans le magasin du lecteur et appuie sur la touche "prêt". Bien entendu, ce n'est pas le superviseur d'entrée-sortie qui va lire le paquet de cartes mais un processus spécialisé du système. Pour créer ce processus, nous procédons en deux temps. En nous reportant à la figure 18, nous voyons que dans un descripteur d'unité figure une zone notée ATTNPR pouvant contenir le point d'entrée d'une procédure traitant les interruptions asynchrones. Si une adresse est spécifiée, la procédure correspondante est appelée par le processus d'interruption d'entrée-sortie. Dans le cas du lecteur de cartes, c'est cette procédure qui va créer le processus "lecture d'un paquet de cartes". Cette technique qui consiste à fournir au niveau de chaque unité une procédure traitant un événement particulier (ici une interruption asynchrone) est à rapprocher de celle consistant à faire traiter, par une procédure RTND, un événement particulier au niveau de chaque tâche d'entrée-sortie (dans ce dernier cas la fin de l'opération).

Une question qui peut venir à l'esprit est la suivante : pourquoi ne pas faire créer directement le processus du système associé à une interruption asynchrone par le processus d'interruption d'entrée-sortie ? En fait, nous savons qu'un processus créé ne sera effectivement activé que lors d'un prochain passage dans le Contrôleur. En admettant même qu'il soit seul dans la liste des processus prêts, il ne prendra le contrôle qu'après la fin d'exécution du processus d'interruption d'entrée-sortie. Or, l'action à entreprendre peut impliquer un changement de comportement du processus d'entrée-sortie lui-même.

Exemple.

A un instant donné, le système connaît les identifications ("labels") de tous les disques montés sur des unités. Cela permet, en particulier, de vérifier lorsqu'un utilisateur joint le système, que le disque privé que l'on intègre à sa configuration lui appartient bien. Supposons qu'en cours de session quelqu'un (par erreur ou sciemment) démonte un disque et le remplace par un autre ; au moment où l'unité devient à nouveau prête, il peut exister des tâches d'entrée-sortie chaînées à son descripteur. Il ne faut pas les exécuter avant d'avoir contrôlé l'identification du disque qui vient d'être monté. Ce contrôle implique une lecture du disque réalisée par une tâche d'entrée-sortie qui doit être exécutée avant celles qui attendaient l'état "prêt". La création d'un processus "vérification d'identification" par le processus d'interruption d'entrée-sortie ne permet pas de satisfaire cette condition puisque la "relance de l'activité canal" entraîne la prise en compte de la première des tâches chaînées. L'appel direct de la procédure ATTNPR permet donc de résoudre ce problème.

Dans notre exemple, cette procédure exécute les deux actions suivantes :

1) décrocher la "liste active" de tâches attachée au descripteur d'unité et la rattacher à la "liste suspendue" ;
2) créer le processus "vérification d'identification".

Lorsque cette procédure s'est terminée, le processus d'interruption d'entrée-sortie ne trouve pas de tâche en attente pour cette unité et se termine aussi.

Lors d'un prochain passage dans le Contrôleur, le processus "vérification d'identification" est activé. Son algorithme se déroule ainsi.

- Soumettre une tâche d'entrée-sortie synchrone pour lire

l'identification du disque qui vient d'être monté.

- Vérifier la validité de cette identification.
- Lorsque la vérification est satisfaisante, appeler une procédure spéciale du superviseur d'entrée-sortie qui rattache la chaîne suspendue à la chaîne active et tente la relance de l'activité du canal. Au retour, le processus "vérification d'identification" se termine.

Remarque. La technique des deux chaînes de tâches peut être utilisée pour le traitement des erreurs d'entrée-sortie, lorsqu'à la suite d'une difficulté quelconque, certaines opérations doivent être effectuées sur l'unité avant que les autres tâches en attente ne soient initialisées.

2.43 Gestion des périphériques

2.431 Gestion de l'imprimante

Les CMS virtuels disposent d'une imprimante virtuelle constituée par un fichier GMS (cf. 2.514). Lorsqu'un CMS "ferme" son imprimante, l'interface générateur (qui intercepte l'opération) "ferme" le fichier correspondant ; ce fichier devient ainsi candidat à une sortie sur l'imprimante réelle.

Comme application des mécanismes généraux développés précédemment, nous étudions deux aspects du problème, à savoir, d'une part la communication-synchronisation entre le processus de l'interface et le processus de gestion de l'imprimante, d'autre part la structure de ce processus de gestion.

2.4311 Communication - synchronisation

La communication entre le processus de l'interface qui ferme un fichier image d'une imprimante virtuelle et le processus de gestion de l'imprimante réelle est du type "boîte aux lettres". Le nom du fichier est placé dans une liste que peut consulter le processus de gestion. Lorsque ce dernier n'a pas de fichier à imprimer, il n'est pas en attente de l'événement "fermeture" mais est tout simplement inexistant (cf. 2.324). Cette méthode évite de mobiliser inutilement des ressources pendant de longues périodes. A l'entrée de ce processus, un "témoin d'existence" est mis à 1. A la sortie, le témoin est remis à zéro. Les opérations de communication et de synchronisation sont alors réalisées simplement de la manière suivante :

processus fermeture de fichier	processus gestion de l'imprimante
début	début
Fermer le fichier ;	TEMOIN D'EXISTENCE := 1 ;
Rajouter son nom dans la boîte aux lettres ;	SUIVANT : si boîte aux lettres vide
si TEMOIN D'EXISTENCE = 0	alors aller a SORTIE ;
alors Créer (processus de gestion de l'imprimante)	Extraire nom de fichier ; Imprimer ce fichier ;
fin	aller a SUIVANT ;
	SORTIE : TEMOIN D'EXISTENCE := 0
	fin

Remarque 1. Les deux processus doivent être en exclusion mutuelle en ce qui concerne l'accès à la boîte aux lettres et au témoin d'existence.

Remarque 2. L'un des principaux problèmes d'implantation d'une communication par boîte aux lettres est celui de l'allocation d'espace à la boîte aux lettres. Dans GMS ce problème est résolu en utilisant une boîte aux lettres d'un type très spécial qui est un sous ensemble du catalogue des fichiers. Un fichier, dans GMS, est identifié par un nom à trois composants 'nom1.nom2.nom3'.

- 'nom1' est le "destinataire" du fichier. Si le fichier doit être écrit sur l'imprimante, nom1 a pour valeur 'PRT'.
- 'nom2' est le "créateur" du fichier, par exemple le nom de l'utilisateur CMS.
- 'nom3' est un numéro d'ordre donné automatiquement lors de la création du fichier. Pour un même destinataire, le numéro d'ordre augmente de 1 à chaque nouvelle création.

Lorsqu'un fichier est fermé, son nom figure dans le catalogue avec l'indication "fermé". Nous disposons d'une procédure recherchant dans le catalogue le nom du fichier fermé de plus petit numéro d'ordre ayant un destinataire donné. Il est alors inutile d'enregistrer ailleurs le nom de ce fichier. L'instruction "Extraire nom de fichier" que nous avons fait figurer dans l'algorithme du processus de gestion est réalisée par l'appel de la procédure de recherche du fichier 'PRT' fermé de plus petit numéro. L'absence, dans le catalogue, de fichier 'PRT' fermé (détectée par la procédure) est équivalente à la condition "boîte aux lettres vide".

2.4312 Processus de gestion

Nous nous intéressons ici à la partie "Imprimer ce fichier" de l'algorithme du processus de gestion.

Les fichiers sont constitués d'enregistrements sur disque ayant une taille fixe (496 octets) ; un enregistrement de fichier 'PRT' contient un nombre entier de lignes à imprimer (généralement 3 ou 4).

Le disque et l'imprimante pouvant fonctionner en simultanéité, l'impression d'un fichier semble bien se prêter à un découpage en deux processus dont les relations seraient du type producteur-consommateur, le processus de lecture des enregistrements du disque "produisant" des "messages" dans des tampons et le processus d'impression "consommant" ces "messages". C'est cette technique qui est utilisée, en particulier, dans le système ESOPE [Kaiser 73, Krakowiak 73]. La structure des deux processus et leurs relations mutuelles s'expriment alors par les algorithmes suivants dans lesquels N est le nombre total de tampons utilisés, NPLEIN le nombre de tampons remplis et non imprimés et NVIDE le nombre de tampons encore disponibles [Crocus 73].

```
                PRODUCTEUR                          CONSOMMATEUR

                    entier NPLEIN = 0, NVIDE = N ;

PROD : NVIDE := NVIDE - 1 ;          CONSOM : NPLEIN := NPLEIN - 1 ;
       si NVIDE = -1 alors                    si NPLEIN = -1 alors
          attendre ;                             attendre ;

  lire un enregistrement dans          écrire l'enregistrement contenu
  le prochain tampon libre;            dans le prochain tampon occupé;

       NPLEIN := NPLEIN + 1 ;                 NVIDE := NVIDE + 1 ;

si consommateur en attente alors      si producteur en attente alors
   réveiller consommateur ;              réveiller producteur ;

aller a PROD ;                       aller a CONSOM ;
```

Nous pouvons noter, avant de poursuivre, que les événements non mémorisés mis en place dans GMS sont tout à fait adaptés pour l'implantation de ces algorithmes. Nous pouvons en effet faire correspondre un événement à chacune des conditions "producteur en attente" et "consommateur en attente". La fonction "attendre" est réalisée par une primitive INACT sur l'événement correspondant et la fonction "réveiller ..." par une primitive ACT. Le test "si ... en attente" n'est même pas nécessaire puisque la primitive ACT est ineffective pour un événement à file d'attente vide.

Pourtant, dans GMS, nous n'avons pas employé ce modèle du producteur-consommateur car il n'eût pas permis une utilisation optimale des ressources. Traçons en effet le diagramme d'évolution dans le temps des deux processus (figure 20) ; pour simplifier, nous supposons qu'il n'y a pas, pour les unités ou canaux concernés, de charge autre que celle due aux entrées-sorties effectuées par ces deux processus. Le temps maximum de lecture d'un enregistrement depuis le disque est inférieur à 100 ms ; le temps d'impression des trois ou quatre lignes contenues dans cet enregistrement est de l'ordre de 300 ms. Deux tampons sont suffisants pour assurer le parallélisme des deux processus, le producteur lisant un enregistrement dans le tampon i+1 pendant que le consommateur imprime le tampon i. Nous désignons dans ces diagrammes par "soumission L(Ti)" et "soumission E(Tj)" la partie des processus qui consiste à préparer une tâche d'entrée-sortie (respectivement pour lire dans Ti et écrire à partir de Tj) et à la soumettre au superviseur d'entrée-sortie par l'intermédiaire de la procédure IOSUP. Nous démarrons le cycle du producteur avec le tampon T1 déjà plein pour obtenir un diagramme en régime permanent.

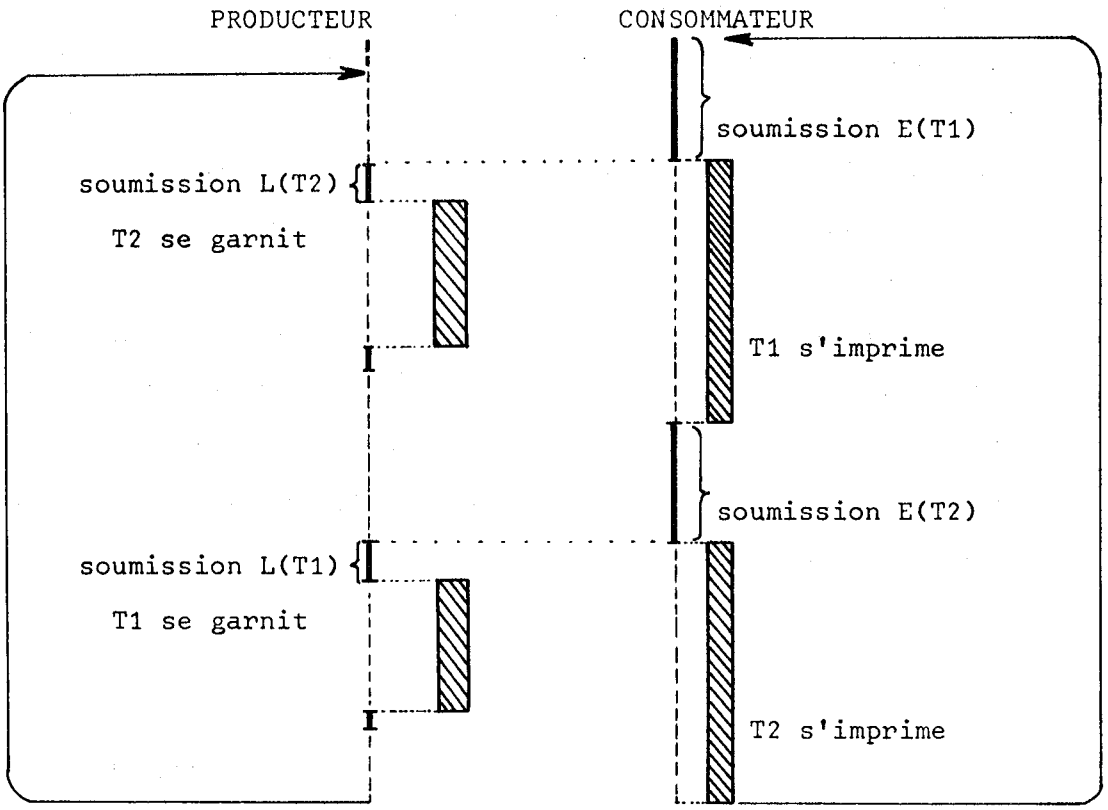


Figure 20. Impression d'un fichier en utilisant 2 processus

Nous voyons qu'il n'y a pas recouvrement total du temps de processeur par les opérations sur l'unité la plus lente qu'est l'imprimante. La partie "soumission E(Ti)" est en fait non négligeable : il faut créer la chaîne de commandes nécessaire pour imprimer les lignes contenues dans le tampon, interpréter certaines informations de contrôle propres à ces lignes puis enfin compléter et soumettre la tâche d'entrée-sortie. Nous obtenons un fonctionnement saccadé de l'imprimante qui imprime trois ou quatre lignes, marque un temps d'arrêt, imprime à nouveau, etc... .

Pour remédier à cela, il faut faire en sorte que lors de la fin d'impression du tampon Ti, la tâche correspondant à l'impression du tampon Ti+1 se trouve déjà dans la file d'attente

attachée à l'imprimante. Nous allons utiliser pour cela le mécanisme de tâches asynchrones et les procédures RTND à l'intérieur d'un processus unique. La figure 21 représente le diagramme d'évolution de ce processus. Nous démarrons le cycle alors que vient d'être lancée l'impression asynchrone du tampon T1.

Les lectures du disque sont réalisées par des tâches synchrones. Lorsqu'un tampon est plein, il est soumis pour être imprimé au moyen d'une tâche asynchrone de manière à pouvoir à nouveau, avant la fin de cette impression, garnir et soumettre le tampon suivant.

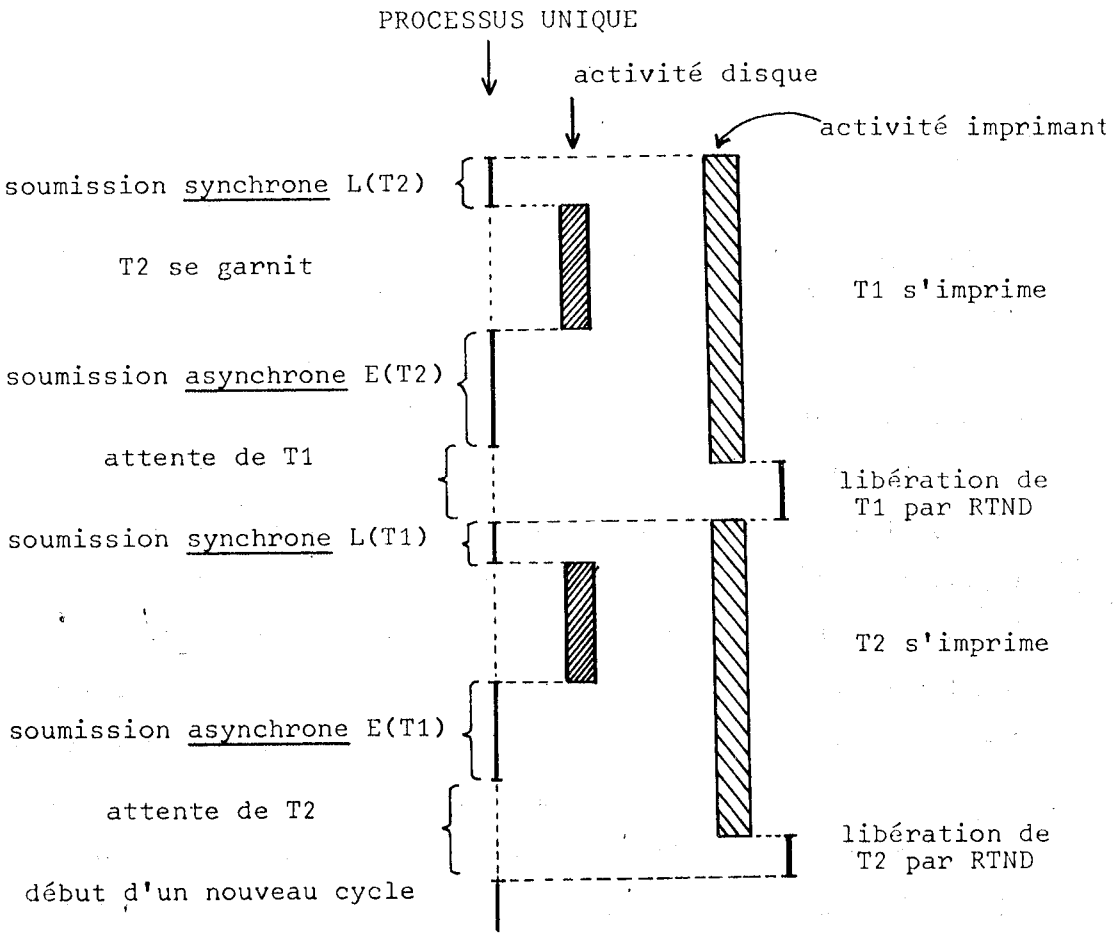


Figure 21. Processus unique de gestion de l'imprimante

La synchronisation ne pose aucun problème : nous définissons deux événements E1 et E2 associés à la libération des tampons T1 et T2. Si le prochain tampon Ti à garnir n'est pas libre (une structure de données décrivant chaque tampon donne cette information) le processus se place en attente de l'événement Ei par l'emploi de la primitive INACT (Ei). Le réveil est effectué par une procédure RTND spécifiée dans le descripteur de tâche asynchrone relatif à l'écriture sur l'imprimante. Cette procédure indique que le tampon que l'on vient d'imprimer est libre et applique la primitive ACT (Ei) qui place à nouveau le processus dans la liste des processus prêts.

2.432 Gestion de la console de l'opérateur

La console de l'opérateur est utilisée par GMS comme dispositif de sortie du journal de bord pour indiquer, par exemple, qu'un utilisateur joint ou quitte le système (commandes LOGIN/LOGOFF) et pour signaler certains incidents nécessitant l'intervention de l'opérateur (erreur permanente sur une unité, fin de papier sur imprimante, unité non prête, ...). La console est également utilisée comme moyen de commande et de contrôle par l'opérateur qui peut entrer un certain nombre de commandes et recevoir des réponses.

A la différence de l'imprimante pour laquelle les requêtes se situent au niveau global "imprimer un fichier", ici, chaque processus voulant utiliser la console n'émet que des requêtes élémentaires du type "écrire un message" ou "lire un message". Une telle requête peut être satisfaite par une tâche d'entrée-sortie et nous disposons avec le superviseur d'entrée-sortie d'une

gestion des files d'attente de tâches pour chaque unité. Il n'est donc pas nécessaire de rajouter une gestion de file d'attente pour les messages de la console.

Les messages sont pour la plupart créés dynamiquement. Pour cela, le processus acquiert une zone de mémoire dans laquelle est placé le message. La console étant une unité lente, l'écriture est effectuée par une tâche asynchrone. Dans le descripteur de cette tâche est spécifiée une procédure RTND chargée de rendre la zone contenant le message à la mémoire libre, une fois l'écriture terminée.

2.5 INTERFACE GENERATEUR DES CMS VIRTUELS

Dans ce paragraphe, nous expliquons d'abord comment les ressources physiques, images des ressources virtuelles, sont allouées aux CMS virtuels. Ensuite, nous montrons comment sont réalisées dans GMS les fonctions d'accès à ces ressources physiques lorsque CMS accède à une ressource virtuelle. Nous donnons pour cela la structure des principaux processus de l'interface qui pilotent la transformation virtuel-réel.

2.51 Allocation des ressources images des ressources virtuelles

Les ressources utilisées par un CMS virtuel peuvent lui être fournies de trois manières.

1) Affectation permanente : lorsqu'il existe suffisamment d'exemplaires soit d'une ressource donnée, soit d'une ressource au moyen de laquelle il est possible de réaliser, par simulation, des fonctions équivalentes, on peut allouer à chaque CMS un exemplaire de cette ressource. Nous disons alors que cette ressource est dédiée à un CMS. Nous utilisons cette technique pour le disque privé de l'utilisateur et pour son terminal. Les unités de bandes magnétiques sont également dédiées mais seulement sur demande explicite de l'utilisateur (leur emploi doit être, en principe, exceptionnel).

2) Multiplexage de ressources requérables : les ressources telles que le processeur et la mémoire sont multiplexées entre les différents CMS virtuels. Cela se traduit dans GMS par une stratégie d'allocation de ces ressources (cf. 2.71).

3) Simulation par des extra-mécanismes : lorsqu'une ressource non

requérable existe en trop peu d'exemplaires pour en dédier un à chaque CMS virtuel, la ressource virtuelle est simulée au moyen d'un mécanisme intermédiaire. C'est le cas de l'imprimante, du lecteur et du perforateur de cartes, simulés par une technique de "spooling".

2.511 Disque système et disque privé

Le système CMS connaît les adresses virtuelles Sv pour le disque système et Uv pour le disque utilisateur, et les diverses copies de ce système utilisées par les CMS virtuels sont identiques. Le disque système n'étant jamais modifié mais seulement lu par CMS, est partageable ; s'il est monté sur l'unité d'adresse réelle Sr, alors pour chaque CMS une opération d'entrée-sortie adressant l'unité virtuelle Sv est transformée par GMS en la même opération adressant l'unité réelle Sr.

Le disque utilisateur n'étant pas partageable, un tel disque doit être affecté à chaque CMS. Toute opération d'entrée-sortie adressant l'unité virtuelle Uv est transformée par GMS en la même opération adressant l'unité réelle Ur(i), i étant l'utilisateur actif et Ur(i) l'adresse réelle de l'unité sur laquelle est monté son disque privé.

Ainsi, une configuration réelle comportant 8 unités de disques permet de faire tourner 6 CMS, une unité étant réservée à GMS pour ses propres besoins et une autre recevant le disque système CMS.

Lorsqu'il joint le système GMS, l'utilisateur précise l'adresse réelle de l'unité sur laquelle est monté son disque privé. Une vérification d'identification est alors exécutée (cf. 2.425).

2.512 Console opérateur

La console opérateur est utilisée pour le contrôle de GMS. A chaque CMS est associé un terminal et GMS exécute sur ce terminal les opérations d'entrée-sortie logiquement équivalentes à celles émises par CMS et s'adressant à la console opérateur.

L'allocation proprement dite d'un terminal à un utilisateur consiste simplement à associer le nom de cet utilisateur à l'adresse réelle du terminal sur lequel il a effectué la procédure de "LOGIN".

2.513 Bandes magnétiques

Ici, les ressources réelles dont dispose GMS sont précisément les ressources virtuelles attendues par CMS ; leur utilisation étant relativement peu fréquente et la réquisition n'étant guère envisageable pour de telles unités, la solution adoptée est la suivante : une commande de GMS permet à un utilisateur depuis son terminal de demander l'attachement d'une unité de bande à sa configuration virtuelle. Si l'unité demandée est déjà attachée à un autre, un message le signale au demandeur ; sinon, l'attachement est réalisé. Lorsqu'une opération d'entrée-sortie concernant une unité de bande est émise par CMS, GMS vérifie que l'unité adressée est attachée à cet utilisateur et si oui exécute l'opération. Sinon, il réfléchit à CMS une condition d'unité non opérationnelle en même temps qu'il prévient l'utilisateur par un message sur son terminal.

2.514 Périphériques

Le lecteur de cartes, le perforateur de cartes et l'imprimante sont simulés pour chaque CMS au moyen de fichiers créés sur le disque GMS.

Une opération d'impression ou de perforation émise par CMS est transformée en une écriture dans le fichier correspondant. L'allocation de ces fichiers est dynamique : lors de la première opération d'écriture sur imprimante ou de perforation de carte émise par le CMS de l'utilisateur U, le fichier PRT.U.n ou PUN.U.n est créé (cf. 2.4311). Cette imprimante ou ce perforateur virtuels disparaissent lors d'une "fermeture" de l'imprimante ou du perforateur par CMS. Les fichiers ainsi créés sont alors transmis au processus de GMS gérant l'imprimante réelle ou le perforateur réel.

Une opération de lecture de carte émise par le CMS de l'utilisateur U est transformée en une lecture dans le fichier RDR.U.n créé automatiquement par GMS lorsqu'un paquet de cartes précédé de l'identification de U est placé dans le magasin du lecteur. Lorsque plusieurs fichiers RDR.U.n existent, c'est celui de plus petit numéro n qui joue le rôle de lecteur virtuel. Un tel fichier est détruit lorsqu'il a été lu entièrement ; une nouvelle lecture de carte émise par le même CMS utilise alors comme lecteur virtuel le fichier RDR.U.n suivant.

2.515 Mémoire et processeur

CMS est structuré grossièrement en deux parties : un noyau résident de 70 K octets environ et une zone de mémoire libre dans laquelle s'exécutent les composants du système et les programmes

de l'utilisateur (figure 22 a). Une partie de cette mémoire libre est retirée à CMS pour y loger GMS (figure 22 b) ; une tentative d'écriture dans cette région est réfléchie à CMS comme une interruption programme pour adressage hors limite.

La partie basse de la mémoire (à partir de l'adresse 0) est une zone privilégiée qui contient des informations exploitées par les processeurs ou rangées par eux (mots d'état programme anciens et nouveaux, mot d'état canal, ...). Cette zone ne doit par conséquent être rendue accessible qu'à GMS, mais CMS, qui ignore l'existence de GMS, utilise aussi cette zone pour lire ou modifier son contenu. Il faut donc pour chaque CMS en créer une réplique dans son domaine adressable, tenir cette réplique à jour et transformer toute référence à la zone originale effectuée par CMS en une référence à cette zone de remplacement.

La mémoire réservée à GMS est protégée par les verrous de protection de la mémoire associés à chaque bloc de 2 K octets. Nous faisons fonctionner CMS avec une clé de protection qui ne donne pas accès aux blocs appartenant à GMS. Les informations technologiques du bas de mémoire étant contenues dans le premier bloc Bo, ce bloc est rendu inaccessible à CMS et nous utilisons comme réplique le bloc B1 dont CMS ne se sert pas.

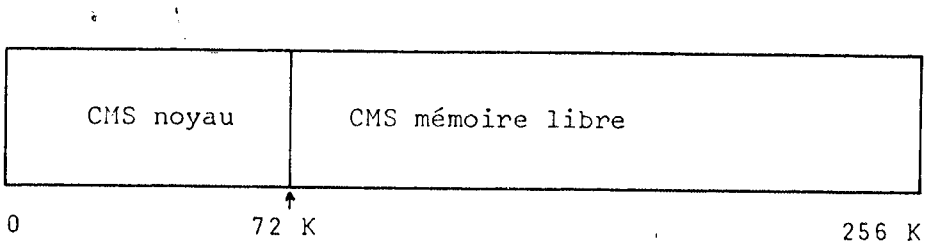


Figure 22 a. Configuration mémoire pour CMS standard

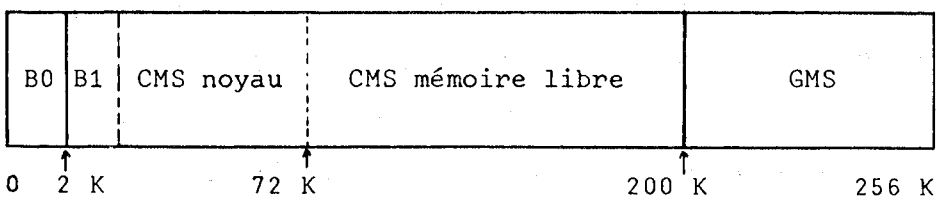


Figure 22 b. Configuration mémoire de CMS sous GMS

La zone de mémoire CMS est affectée successivement et globalement aux utilisateurs par la technique du va-et-vient [Crocus 73]. CMS n'étant pas conçu pour être partagé, chacun dispose de sa propre copie lue depuis support externe avant de recevoir le contrôle et recopiée à la fin de chaque tranche de temps d'utilisation du processeur. Ce va-et-vient est réalisé entre la mémoire et une zone réservée du disque privé de l'utilisateur.

Il existe sur le disque GMS une image mémoire correspondant à la copie initiale de CMS telle qu'elle serait obtenue par un "chargement initial" depuis le pupitre (IPL). C'est cette copie qui est amenée en mémoire lorsque l'utilisateur frappe initialement la commande 'CMS' sur son terminal, ce qui simule la fonction IPL du pupitre.

Le va-et-vient global impose d'allouer le processeur en même temps que la mémoire. Pour que GMS puisse effectuer le contrôle de l'utilisation des ressources physiques, le processeur, lorsqu'il est alloué à CMS est placé dans l'état programme bien que CMS utilise l'état superviseur. Les fonctions ainsi retirées à CMS doivent lui être rendues par simulation, le processeur virtuel d'un CMS disposant des mécanismes de contrôle. Ainsi, les instructions privilégiées exécutées par un CMS (manipulation du mot d'état programme, instructions d'entrée-sortie, ...) sont interceptées et simulées par des processus de l'hyperviseur.

L'utilisation du va-et-vient global pose des problèmes importants d'efficacité. Pour assurer à la fois des temps de réponse convenables et une efficacité raisonnable du processeur nous devons mettre en oeuvre une stratégie d'allocation du processeur aux CMS virtuels qui tienne compte des contraintes liées au va-et-vient. L'étude de la stratégie adoptée sera abordée, en même temps que d'autres problèmes particuliers, au paragraphe 2.7.

Chaque utilisateur est représenté dans le système par un descripteur donnant pour chaque ressource virtuelle de son CMS le support réel correspondant (figure 23). Ce descripteur contient également une zone de sauvegarde de l'état du processeur, utilisée lors de la suspension du CMS et des renseignements tels que le nom de l'utilisateur, l'environnement dans lequel il se trouve (CMS, GMS, ...) ou l'état de son CMS (attente de lecture console par exemple).

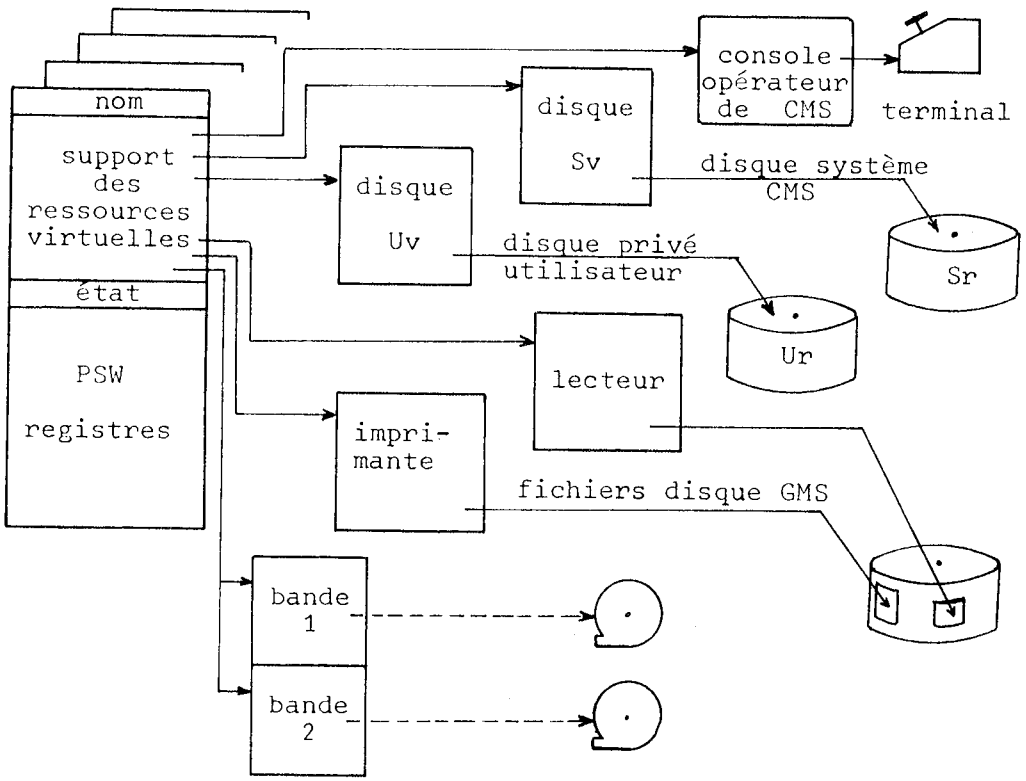


Figure 23. Description du support des ressources virtuelles

2.52 Réalisation des fonctions d'accès aux ressources

2.521 Réaction aux instructions privilégiées

Pour chaque CMS virtuel, GMS tient à jour un pseudo mot d'état (PSW). Les opérations privilégiées agissant sur le mot d'état (chargement d'un masque d'interruption ou chargement d'un nouveau mot d'état) sont interceptées et simulées à l'aide de ce pseudo mot d'état.

Les opérations dont l'exécution dépend du mot d'état sont interprétées en fonction de ce pseudo état. Par exemple, lorsque CMS exécute une instruction d'entrée-sortie (SIO), cette

instruction étant privilégiée n'est pas exécutée mais une interruption programme redonne le contrôle à GMS puisque le mot d'état réel indique le mode programme. Si le pseudo état indique aussi le mode programme, l'exécution de l'instruction est une erreur et GMS réfléchit l'interruption programme à CMS (il utilise pour cela les ancien et nouveau mots d'état du CMS courant pris dans B1). Si par contre le pseudo état indique le mode superviseur, l'instruction est légitime et GMS doit réaliser l'opération correspondante.

Nous allons détailler, dans ce cas particulier de l'instruction d'entrée-sortie SIO, les processus qui réalisent l'opération de simulation.

2.522 Interface d'entrée-sortie

Lorsque CMS veut effectuer une opération d'entrée-sortie, il procède de la manière suivante après avoir préparé le programme canal.

a) Exécution de l'instruction SIO dans laquelle est précisée l'adresse de l'unité.

b) Examen du "code condition" résultant de l'exécution de SIO. Une valeur nulle de ce code condition indique que le canal a pris en charge l'opération d'entrée-sortie. Une valeur différente de zéro signale une condition particulière qui, dans le cas général, n'a pas permis de prendre en charge l'opération (unité occupée ou canal occupé par exemple).

c) Dans le cas où l'opération a été démarrée (code condition = 0) CMS peut, éventuellement, exécuter d'autres instructions avant de se placer en attente de la fin d'entrée-sortie.

d) Les étapes a, b, c sont exécutées par CMS avec interruptions

d'entrée-sortie masquées. Pour se synchroniser avec l'opération d'entrée-sortie, CMS exécute une instruction de chargement du mot d'état programme (LPSW) qui place le processeur virtuel en état "attente" avec interruptions permises. Lorsque l'interruption se produit, CMS reprend le contrôle du processeur et traite cette interruption. Il dispose pour cela des informations contenues dans le mot d'état canal (CSW) écrit en mémoire basse par le canal au moment de l'interruption.

La figure 24 représente le diagramme de cette exécution virtuelle.

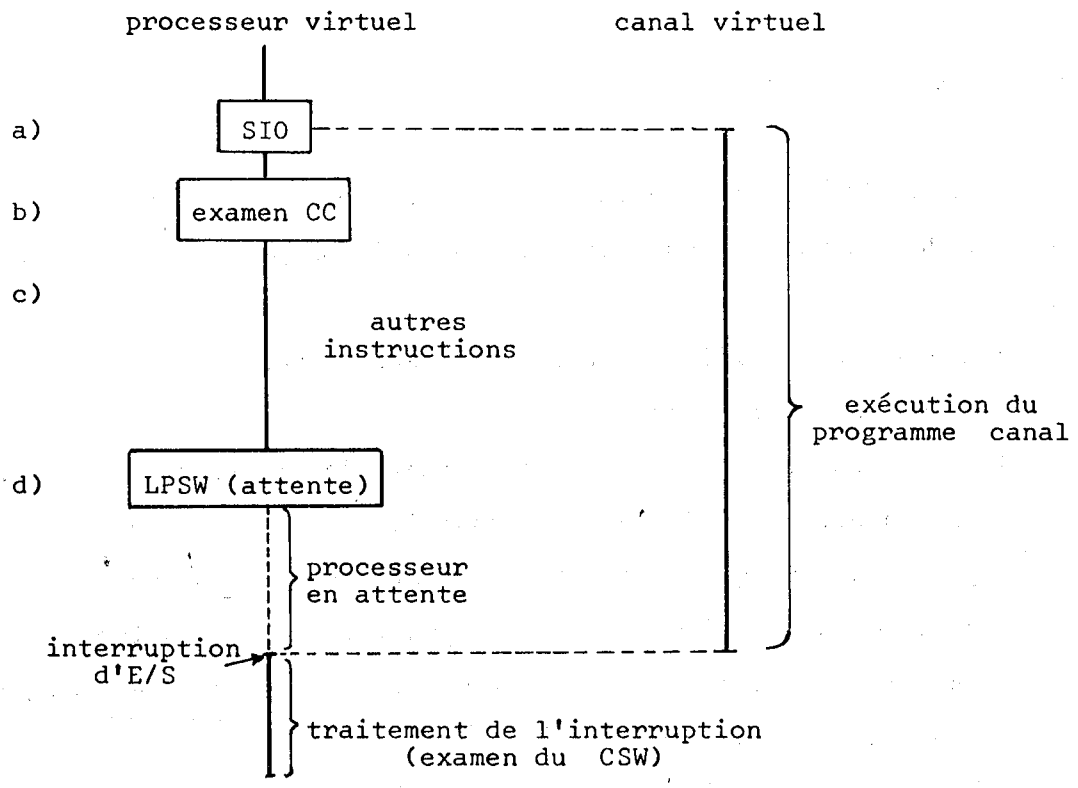


Figure 24. Les étapes d'une entrée-sortie virtuelle

Pour simuler une entrée-sortie virtuelle, GMS doit non

seulement faire exécuter les fonctions définies par le programme canal soumis, mais également réaliser la synchronisation virtuelle matérialisée dans CMS par l'emploi de l'instruction LPSW (attente).

Deux techniques sont utilisées pour simuler une entrée-sortie virtuelle, la simulation synchrone et la simulation asynchrone.

2.5221 Simulation synchrone

En simulation synchrone, lorsque CMS reprend le contrôle après exécution d'une instruction SIO, l'opération réelle correspondant à cette entrée-sortie virtuelle est complètement terminée. Le processus de GMS qui pilote cette simulation, réalise la fonction équivalente à l'entrée-sortie virtuelle, puis met en réserve un code condition de valeur nulle qui sera accessible à CMS lors de sa réactivation ; il crée, ou obtient depuis le canal, un mot d'état canal tel qu'il serait obtenu si l'entrée-sortie virtuelle s'exécutait directement sur une configuration réelle ; enfin, il mémorise une condition d'interruption dans la zone décrivant ce CMS. Lorsque ce dernier exécute l'instruction LPSW (attente), la simulation de cette instruction conduit à simuler le mécanisme d'interruption d'entrée-sortie en fournissant à CMS le mot d'état canal mis en réserve précédemment (figure 25).

Remarque. Lorsque l'opération correspondant à l'instruction SIO exécutée par CMS ne peut être réalisée (adresse d'unité invalide par exemple), la valeur du code condition mis en réserve n'est pas nulle mais reflète cette situation. Il n'y a pas mémorisation d'une condition d'interruption ni création d'un mot d'état canal.

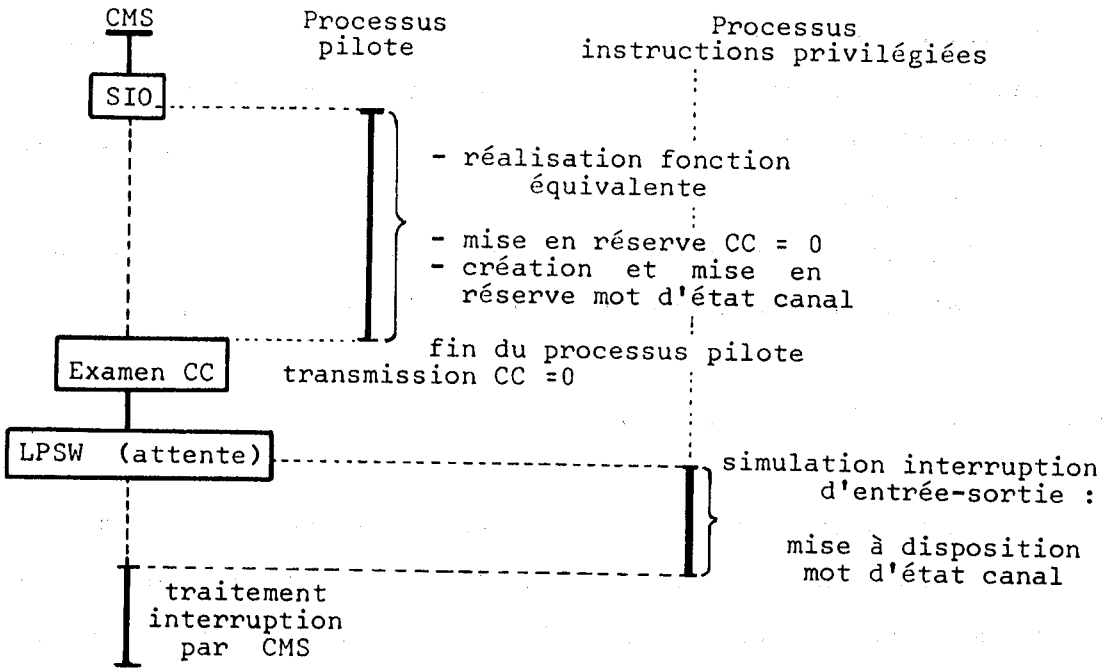


Figure 25. Simulation synchrone d'une entrée-sortie virtuelle

La simulation synchrone est utilisée pour toutes les unités, sauf pour la console lorsque l'opération sur cette dernière est une écriture.

2.5222 Simulation asynchrone

En simulation asynchrone, le CMS émetteur de l'instruction SIO peut à nouveau être activé avant que la fonction réelle correspondant à l'entrée-sortie virtuelle ne soit terminée. Le diagramme d'exécution est identique au précédent, à cela près que la fin du processus pilote ne correspond pas à la fin de la simulation : d'autres processus interviennent ensuite pour la poursuite de cette simulation.

Dans ces conditions, CMS traite une condition de fin d'entrée-sortie alors que l'opération est en cours. Cette méthode

n'est applicable qu'aux opérations de sortie à la condition que les zones de mémoire participant au transfert soient recopiées dans une zone de mémoire libre appartenant à GMS. En effet, la fin d'opération signifie, entre autres, pour CMS, que la zone de mémoire qui vient d'être écrite est maintenant disponible. Nous aurions pu utiliser la simulation asynchrone pour les opérations d'impression ou de perforation virtuelles (c'est d'ailleurs ce que nous avions fait dans une première version du système) mais la faible quantité de mémoire disponible dans GMS nous a conduit à y renoncer. Par contre, nous utilisons cette simulation asynchrone pour les écritures sur le terminal (console virtuelle).

2.523 Entrées-sorties sur disques ou bandes

Pour les disques et les bandes magnétiques, les unités réelles sont identiques aux unités virtuelles dont elles sont l'image. Le programme canal soumis par l'instruction SIO pour une telle unité peut donc être exécuté sans transformation et les réponses du canal peuvent être transmises telles quelles en retour. Le processus qui pilote la simulation prépare la tâche d'entrée-sortie synchrone correspondante et la soumet au superviseur d'entrée-sortie. Lorsque ce processus est réactivé, il récupère dans le descripteur de tâche le mot d'état canal qui sera, plus tard, transmis à CMS lors de l'exécution de l'instruction LPSW (attente). Pendant toute la durée de la simulation, CMS est bloqué en mémoire et n'est pas activable. L'adresse de l'unité réelle impliquée dans le transfert est obtenue à partir du descripteur des ressources virtuelles pour ce CMS. Dans le cas d'une bande magnétique (cf. 2.513), si une unité de bande n'est pas attachée à cet utilisateur, une condition

"unité non opérationnelle" est simulée (code condition 3).

2.524 Entrées-sorties sur imprimante et lecteur-perforateur

Les supports des imprimantes, lecteurs et perforateurs virtuels sont des fichiers GMS (cf. 2.43). Nous pouvons expliquer sur l'exemple de l'imprimante le fonctionnement des processus qui pilotent la simulation.

Lors de la première écriture sur imprimante en provenance du CMS de l'utilisateur U, le fichier PRT.U.n est créé. Une zone tampon est acquise dans la mémoire libre de GMS. Sa taille (496 octets) est celle des enregistrements de la gestion de fichiers. Tant que cette zone n'est pas pleine, la simulation de l'instruction SIO sur l'imprimante virtuelle se traduit simplement par un transfert de la ligne à imprimer dans cette zone. CMS reprend le contrôle immédiatement après. Les lignes étant de longueur variable, la détection de la condition "tampon plein" intervient alors qu'il y a dans la mémoire de CMS une ligne à imprimer dont la longueur est supérieure à la place disponible restante. La faible quantité de mémoire utilisable par GMS ne nous permet pas d'acquérir un second tampon qui permettrait une simulation asynchrone. Le tampon plein est donc écrit dans le fichier PRT.U.n au moyen d'une tâche synchrone et la ligne en attente est transférée en tête de ce même tampon au retour. C'est seulement après ce transfert que CMS peut à nouveau être activé.

2.525 Entrées-sorties sur console

Lorsque CMS exécute une instruction SIO sur l'unité d'adresse '009' qui est l'adresse de sa console opérateur virtuelle, GMS

examine le programme canal soumis pour différencier les lectures des écritures.

Une opération de lecture est traitée par simulation synchrone puisque CMS a en général besoin immédiatement du message qu'il lit. La frappe du message étant une opération lente, on indique que ce CMS peut être vidé de la mémoire pour laisser la place à d'autres CMS activables. La lecture doit donc être exécutée non pas dans la zone spécifiée par le programme canal (cette zone est dans la mémoire de CMS) mais dans un tampon acquis dans la mémoire libre de GMS. Lorsque le message sera lu et que CMS aura été ramené en mémoire pour réactivation, GMS devra transférer le contenu du tampon dans la zone de lecture initiale.

Le terminal image de la console opérateur virtuelle a un fonctionnement très différent de celui de cette dernière, il utilise des codes internes différents et transmet des informations différentes en fin d'opération. Le processus pilote concerné effectue donc une véritable simulation de la fonction "lire une ligne". Il construit pour cela une tâche synchrone d'entrée-sortie comportant un programme canal qui réalise sur le terminal la lecture qui s'adresse à la console opérateur virtuelle. Lorsque cette lecture est terminée, le processus pilote effectue les conversions de codes nécessaires et crée pour CMS le mot d'état canal tel que l'aurait produit un canal contrôlant la même opération sur la console de l'opérateur. Il indique enfin que ce CMS est à nouveau activable.

Une opération d'écriture est traitée par simulation asynchrone. Il est facile de justifier ce choix : en effet, l'asynchronisme permet de conserver CMS en mémoire et de le réactiver immédiatement. En simulation synchrone, il faudrait de toutes façons vider le CMS de la mémoire, l'écriture sur terminal

étant une opération lente ; nous introduirions ainsi, lors de l'écriture de plusieurs messages successifs, un coût très important dû aux opérations de vidage et de restauration.

Pour réaliser la simulation asynchrone, deux méthodes étaient envisageables.

1) Création d'un processus parallèle utilisant une tâche synchrone : le processus pilote acquiert un tampon dans la mémoire libre de GMS et y transfère la ligne à écrire depuis la mémoire de CMS. Il crée un processus "écriture sur terminal" (qui sera activé lors d'un passage dans le Contrôleur) à qui il transmet l'adresse du tampon. Le processus pilote se termine en créant les conditions de fin d'opération attendues par CMS. Le processus "écriture sur terminal", lorsqu'il est activé, soumet une tâche d'entrée-sortie synchrone. Lorsque cette tâche est exécutée, il libère le tampon et le descripteur de tâche et se termine.

   Remarque. L'acquisition du tampon et le transfert du message pourraient être réalisés par le processus "écriture sur terminal" ; en effet, ce dernier est, dans tous les cas, activé avant CMS, du fait de la priorité donnée par le Contrôleur aux processus de GMS par rapport aux CMS virtuels.

2) Utilisation d'une tâche asynchrone avec procédure RTND (cf. 2.424) : le processus pilote acquiert le tampon, y transfère la ligne à écrire et soumet une tâche d'entrée-sortie asynchrone dans laquelle est spécifiée une procédure RTND qui libérera le tampon et le descripteur de tâche en fin d'opération. Dès la soumission effectuée, le processus

pilote se termine en créant les conditions de fin d'opération pour CMS.

Nous avons adopté la seconde solution pour la raison suivante : dans la première solution, chaque opération d'écriture est gérée par un processus différent ; bien qu'il s'agisse du même programme, il y a autant de processus en attente que de messages à imprimer à un instant donné. Les descripteurs de ces processus occupent une partie de mémoire non négligeable : un calcul rapide montre qu'avec 6 terminaux ayant chacun 10 messages en attente, les descripteurs de processus occupent de l'ordre de 5 K octets de mémoire. Dans la seconde solution, il n'y a pas de processus en attente et cette place est ainsi économisée.

Le problème le plus délicat en simulation asynchrone est lié à la gestion de la mémoire libre : si un CMS n'effectue que des opérations d'écriture sur sa console, il peut, pendant la durée d'une activation, saturer complètement la mémoire libre de GMS. Il faut donc à la fois limiter le nombre maximum de messages en attente d'écriture sur terminal et prévoir une procédure d'attente pour le cas où il n'y a plus assez de mémoire pour acquérir un tampon.

Pour limiter le nombre de messages, nous avons défini un compteur de messages à écrire qui est incrémenté par le processus pilote et décrémenté par la procédure RTND. Le processus pilote, après avoir pris en charge le message, désactive le CMS si la valeur du compteur est égale à une limite NMAX (nous avons fixé NMAX = 10). Ce CMS peut alors être vidé de la mémoire. Il deviendra à nouveau activable lorsque le compteur passera par la valeur NMIN (nous avons fixé NMIN = 2). C'est la procédure RTND

qui détecte que cette limite est atteinte et indique que ce CMS est activable. Le choix d'une valeur différente de zéro pour NMIN permet, compte tenu de la stratégie d'allocation du processeur aux CMS virtuels (cf. 2.71), de réactiver le CMS avant que la file d'attente des messages à écrire soit épuisée et donc de réalimenter cette file dans le cas d'écritures répétées. Le terminal travaille ainsi sans discontinuité.

La mise en attente du processus lorsqu'il n'y a plus de mémoire disponible est un peu plus difficile à réaliser. En effet, le processus pilote naît lorsque CMS exécute l'instruction SIO sur la console virtuelle et meurt lorsqu'il a acquis un tampon, transféré la ligne depuis la mémoire de CMS dans ce tampon et soumis une tâche d'entrée-sortie asynchrone. S'il n'est pas possible d'allouer une zone libre, il semble raisonnable de permettre le vidage de ce CMS de la mémoire pour pouvoir attribuer mémoire et processeur à un autre CMS virtuel. L'utilisation directe des mécanismes de mise en attente et de réveil d'un processus demandeur de mémoire (cf. 2.413) n'est pas suffisante. En effet, lorsqu'à la suite d'une libérazion de mémoire le processus pilote est réactivé, le CMS n'est plus là et la ligne à écrire ne peut pas être transférée dans le tampon obtenu. Nous pourrions inclure dans le processus pilote une séquence de va-et-vient pour vider le CMS courant et ramener le CMS initial. Nous ne le faisons pas car cela reviendrait à introduire, à l'intérieur d'un processus dont la fonction est d'écrire une ligne sur terminal, une décision concernant l'allocation de processeur (et de mémoire) aux CMS virtuels. Une telle interférence n'est pas souhaitable car, à la limite, les stratégies d'allocation de ressources pourraient se trouver diluées, parfois de manière incohérente, dans l'ensemble du système.

Nous pouvons alors considérer que l'exécution par CMS d'une instruction SIO destinée à la console virtuelle est une demande indirecte d'allocation de ressource. Cependant, les mécanismes d'allocation de ressources que nous avons défini pour les processus de GMS ne tiennent pas compte des CMS virtuels. La seule solution satisfaisante consiste alors à "découpler" les deux activités (processus pilote et CMS virtuel) et à faire réémettre la demande par CMS (exécuter à nouveau l'instruction SIO) après que le processus pilote ait acquis le tampon. Nous arrivons à la solution suivante.

L'acquisition du tampon par le processus pilote est réalisée par une demande conditionnelle sans attente (procédure GETCORE). Si l'allocation est effectuée, le processus pilote se poursuit normalement par le transfert de la ligne à écrire dans le tampon et la soumission d'une tâche d'entrée-sortie asynchrone. Sinon, il indique dans le descripteur du CMS virtuel que ce dernier est en attente et peut être vidé de la mémoire. Il modifie le compteur ordinal de ce CMS de manière à ce que la prochaine instruction qu'il exécutera ne soit pas celle suivant le SIO mais le SIO lui-même. Enfin, il émet une demande conditionnelle avec attente pour le tampon. Lorsque, à la suite d'une libération de mémoire par un autre processus l'allocation du tampon est réalisée, le processus pilote indique que le CMS correspondant est à nouveau activable et inscrit l'adresse du tampon dans son descripteur. Ainsi, lorsque le processus pilote naît, il regarde si un tampon est dejà attaché à ce CMS ; si oui, il l'utilise, sinon il l'acquiert par la méthode que nous venons de développer. La figure 26 résume ces opérations.

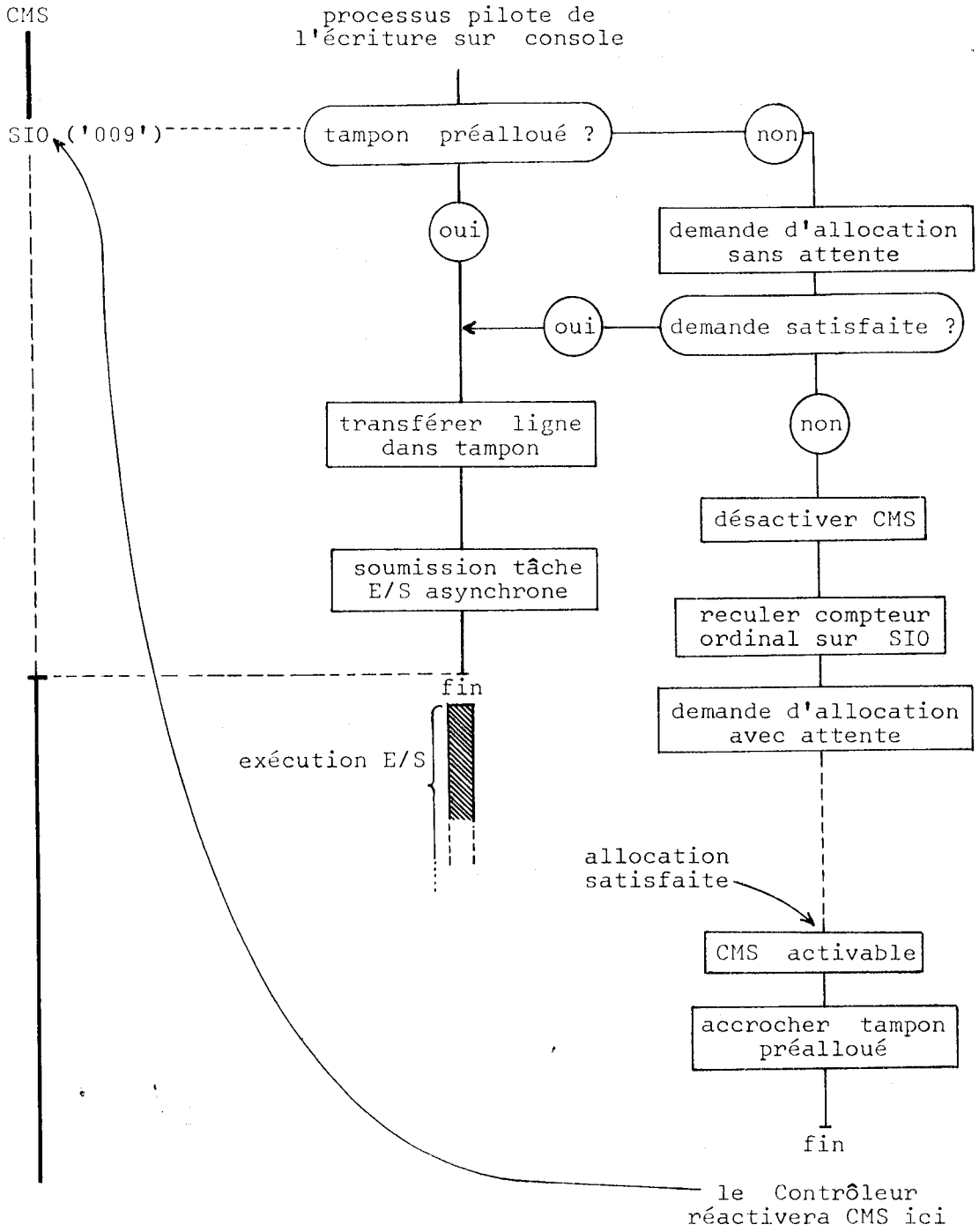


Figure 26. Processus pilote de l'écriture sur console

2.6 L'UTILISATEUR ET LE SYSTEME

En plus du support des CMS virtuels, GMS fournit à l'utilisateur un certain nombre de commandes pour réaliser les fonctions d'accès au système, l'initialisation ou la reprise d'un CMS, l'allocation des bandes magnétiques, la communication par messages ou par fichiers avec d'autres utilisateurs ou encore le contrôle des opérations sur certaines unités [Potin 72].

2.61 Transitions entre environnements

Nous appelons environnement le système ou la partie de système à qui sont destinés, à un instant donné, les messages en provenance d'un terminal. Les transitions entre environnements sont contrôlées par l'utilisateur au moyen de son terminal.

Nous définissons trois environnements possibles.

- L'environnement GMS dans lequel l'utilisateur frappe des commandes ou des données destinées à GMS. A la fin d'exécution de l'une de ces commandes, l'utilisateur se retrouve dans l'environnement GMS, sauf s'il s'agissait d'une commande de changement d'environnement. L'entrée dans l'environnement GMS est commandée par la touche "Attention" du terminal.
- L'environnement CMS dans lequel l'utilisateur frappe des commandes ou des données destinées à CMS. Toutes les commandes prévues sous le système CMS sont disponibles. L'entrée dans l'environnement CMS est effectuée au moyen de la commande 'CMS' ou de la commande 'EXTERN' (simulation de la touche interruption externe du pupitre de commande du

360) frappées dans l'environnement GMS.

- L'environnement TFILE dans lequel les lignes frappées sur le terminal sont utilisées pour créer un fichier qui pourra servir ensuite de "pseudo terminal" pour CMS. On entre dans l'environnement TFILE par la commande 'TFILE' frappée dans l'environnement GMS. L'intérêt de cet environnement est qu'il permet à l'utilisateur d'anticiper les lectures demandées par CMS, en créant une suite de lignes qui seront ultérieurement absorbées par CMS sans nécessiter le va-et-vient de ce dernier pour chacune d'elles.

La figure 27 résume les transitions entre environnements. Dans cette figure, "Attn" représente l'utilisation de la touche Attention du terminal et 'COM' la frappe de la commande de nom COM.

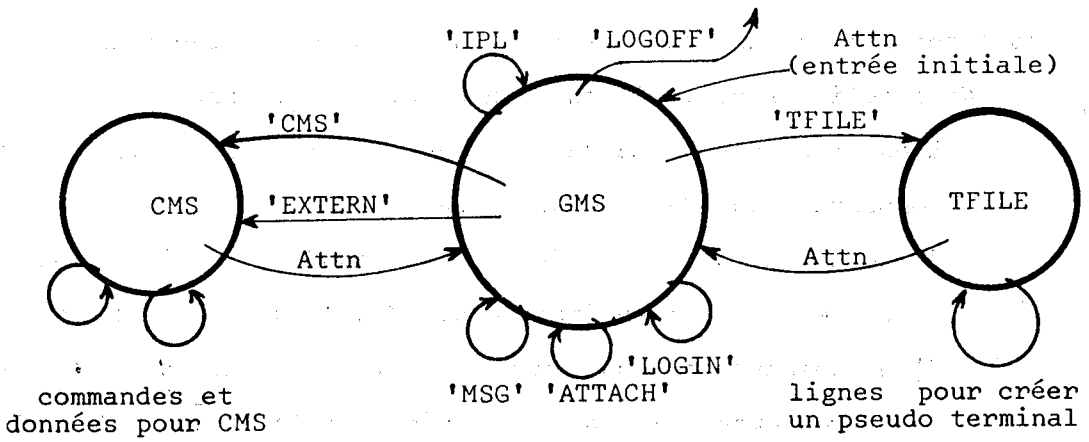


Figure 27. Transitions entre environnements

Nous ne détaillerons pas les commandes de l'environnement GMS ni le fonctionnement de l'environnement TFILE. Nous évoquons certains problèmes liés à la mise en oeuvre des environnements.

2.62 Entrée dans l'environnement GMS

L'entrée dans l'environnement GMS s'effectue en appuyant sur la touche Attention du terminal. Supposons que l'utilisateur se trouve dans l'environnement CMS. Au moment où la touche Attention est utilisée, le terminal peut être soit inactif soit en cours de lecture ou d'écriture. Dans chaque cas, il nous faut créer le processus GMS qui interprète les commandes de cet environnement (nous appelons ce processus SYSCOM), mais il faut aussi quitter "proprement" l'environnement CMS.

Quand un utilisateur est dans l'environnement GMS, son CMS est non activable ; le processeur et la mémoire peuvent ainsi être alloués à d'autres CMS. Ce CMS redevient activable lorsque la commande 'CMS' est frappée, à moins que d'autres conditions telle qu'une attente de lecture sur terminal n'empêche l'activation.

Lorsque le terminal est inactif, l'utilisation de la touche Attention est détectée comme une interruption asynchrone (cf. 2.425). En nous reportant à la figure 18 donnant la structure d'un descripteur d'unité d'entrée-sortie, nous voyons qu'il est possible de mentionner le point d'entrée d'une procédure ATTNPR traitant les interruptions asynchrones. Dans le cas du terminal, cette procédure crée le processus SYSCOM et indique que ce CMS est non activable. Lors d'un prochain passage dans le Contrôleur, SYSCOM sera activé.

Lorsque le terminal est en cours de lecture ou d'écriture, l'utilisation de la touche Attention termine le programme canal en cours. Cette situation est détectée par la séquence du superviseur d'entrée-sortie qui traite la fin d'opération ; or, la lecture ou l'écriture n'a pas été exécutée (ou exécutée de façon incomplète). S'il s'agit d'une écriture, il peut également exister d'autres

tâches d'écriture chaînées au descripteur d'unité. L'utilisateur va maintenant vouloir frapper des commandes pour GMS et ce dernier va lui transmettre des réponses. Il faut donc préserver les tâches chaînées au descripteur d'unité, y compris la première sur laquelle a eu lieu la détection. Plus tard, lorsque la commande 'CMS' sera frappée, il faudra exécuter ces tâches. Cela est réalisé par l'emploi de la "liste suspendue" dans le descripteur d'unité (figure 18). Pour ne pas introduire dans le superviseur d'entrée-sortie une partie de la logique du changement d'environnement, nous rajoutons au descripteur de tâche d'entrée-sortie une extension (figure 19) dans laquelle est noté le point d'entrée d'une procédure traitant une situation particulière survenant pendant le déroulement de la tâche. Cette procédure ne fait pas double emploi avec la procédure ATTNPR notée dans le descripteur d'unité : en effet, dans un autre environnement que CMS l'emploi de la touche Attention ne conduit pas aux mêmes actions ; nous spécifions alors pour les tâches d'entrée-sortie sur terminal, dans cet autre environnement, une autre procédure pour traiter l'attention sur lecture ou écriture. Nous pouvons ainsi considérer que la procédure ATTNPR, associée à l'unité, traite uniquement les interruptions asynchrones (aucune tâche d'entrée-sortie) alors que la procédure notée dans le descripteur d'une tâche traite l'attention survenant au cours du déroulement de cette tâche.

2.63 Contrôle des opérations de sortie sur terminal

Dans le système CMS fonctionnant seul sur une machine, l'utilisateur peut annuler la sortie sur terminal des messages transmis par une commande (cas d'une commande erronée ou de la

liste d'un fichier qui ne présente plus d'intérêt par exemple). Pour cela il utilise la touche "Request" de la console de l'opérateur et frappe la commande à effet immédiat 'KT'. Le traitement par CMS de cette commande consiste à annuler toutes les écritures actuelles et à mettre en fonction l'indicateur "kt" pour supprimer les écritures futures. L'indicateur ne sera remis à zéro que lorsque la fin de la commande en cours aura été atteinte. La commande en cours peut évidemment continuer à appeler la procédure de sortie sur terminal, mais cette dernière trouvant l'indicateur "kt" en fonction ignore l'écriture et se termine immédiatement.

Lorsque CMS fonctionne sous GMS, qui doit traiter la commande 'KT' ? Ce ne peut plus être CMS seul puisque GMS effectue, à son insu, des sorties différées. L'utilisateur ayant frappé 'KT' verrait alors s'imprimer ensuite sur son terminal les lignes en attente de sortie (celles correspondant aux tâches chaînées au descripteur du terminal). Le traitement ne peut pas être effectué par GMS seul puisque les écritures ultérieures demandées par CMS doivent également être annulées jusqu'à ce que la commande en cours soit terminée (et GMS n'a aucun moyen pour détecter la fin d'une commande de CMS). La seule solution utilisable, qui il faut le reconnaître est contraire au principe d'indépendance des deux systèmes, consiste à faire émettre la commande 'KT' dans l'environnement GMS en donnant à ce dernier accès à l'indicateur "kt" qui se trouve dans la mémoire de CMS. GMS annule les tâches d'écriture chaînées au descripteur du terminal et met en fonction l'indicateur "kt" dont il connaît l'adresse dans la mémoire de CMS. Il effectue alors un retour automatique à l'environnement CMS.

Pour pouvoir réaliser ces opérations de manière plus élégante, il faudrait que CMS, au lieu de mettre directement en fonction ou

hors fonction l'indicateur "kt" utilise deux fonctions "désactivation terminal" et "réactivation terminal" dont l'appel puisse être intercepté par GMS. Cela ne changerait pas la logique de CMS et permettrait la construction d'un hyperviseur complètement indépendant de cette logique.

2.7 PROBLEMES PARTICULIERS

Dans les paragraphes précédents, nous avons volontairement passé sous silence, pour ne pas distraire de l'essentiel, un certain nombre de problèmes liés aux notions présentées. Nous reprenons ici l'étude de quelques uns de ces points particuliers qui se rapportent à la gestion des ressources ou à la structure du système.

2.71 Stratégie d'allocation du processeur aux CMS virtuels

2.711 Etats d'un CMS virtuel

En ce qui concerne la possibilité d'allocation du processeur, un CMS virtuel peut se trouver dans cinq états différents :

- activable,
- non activable mais verrouillé en mémoire,
- en attente de lecture sur terminal,
- en attente pour dépassement du nombre de lignes non encore écrites sur terminal,
- suspendu par suite du passage dans l'environnement GMS.

L'état "activable" ne suppose pas la présence du CMS en mémoire : lorsque le Contrôleur choisit de rendre actif un CMS activable, il l'amène en mémoire s'il n'y est déjà.

L'état "non activable mais verrouillé en mémoire" correspond au cas où un processus GMS pilote une action initialisée par CMS et nécessitant la présence de ce dernier en mémoire. Il en est ainsi des entrées-sorties sur disques ou bandes magnétiques et,

comme nous l'avons vu (cf. 2.524), des écritures sur imprimante virtuelle. Dans tout état autre que "non activable mais verrouillé en mémoire", le vidage du CMS est possible.

Dans le descripteur de chaque CMS (figure 23) se trouve un indicateur d'état constitué par une chaîne de bits, chacun d'eux indiquant lorsqu'il est à 1 une condition particulière avec les conventions suivantes :

NOSWAP	: CMS verrouillé en mémoire
LECTER	: CMS en attente de lecture sur terminal
MAXLINE	: CMS en attente d'écriture (limite NMAX atteinte)
DISKIO	: CMS en cours d'entrée-sortie sur disque
TAPEIO	: CMS en cours d'entrée-sortie sur bande
GMS	: le terminal est dans l'environnement GMS
TIMEOUT	: interruption d'horloge signalant une fin de tranche de temps

La condition "CMS activable" correspond à une valeur nulle de cet indicateur d'état.

Nous avons introduit dans cet indicateur la condition "interruption d'horloge", plutôt que de la faire traiter directement par le processus créé par une interruption d'horloge ; en effet, une fin de tranche de temps n'entraîne pas forcément l'allocation du processeur à un autre CMS ; c'est au Contrôleur d'en décider. De plus, cette interruption peut survenir alors que CMS est verrouillé en mémoire. Il faut donc seulement noter l'occurrence de l'interruption et laisser le soin au Contrôleur de traiter la condition au moment opportun.

2.712 Algorithme du Contrôleur [Bellino 72, Finet 73 a]

Le Contrôleur reçoit le contrôle lors de la fin ou de la mise en attente de tout processus GMS. Comme nous l'avons vu, il active en priorité les processus GMS de la liste H-Activable. Cette liste est gérée par la technique du "premier arrivé, premier servi" (FIFO). Lorsque la liste est vide, le Contrôleur tente d'allouer le processeur à un CMS virtuel. C'est cette allocation que nous étudions ici.

Dans tout système interactif, on peut avoir à tenir compte de deux exigences contradictoires :

1) la garantie d'un temps de réponse convenable pour les utilisateurs qui émettent des commandes interactives ;
2) un emploi rationnel des ressources tendant à améliorer le rendement global de l'installation.

Dans GMS, la contradiction entre ces deux objectifs est accentuée. En effet, la satisfaction d'un bon temps de réponse conduit à envisager l'activation d'un CMS en priorité lorsque son utilisateur a frappé une commande, mais cette activation nécessite en général un vidage et une restauration de la mémoire ; ces opérations sont longues et pendant qu'elles se déroulent le processeur reste inactif (sauf pour exécuter des processus GMS). A titre indicatif, le temps de commutation d'un CMS à un autre est de l'ordre de 2 secondes.

Des mesures [Leroudier 72] du comportement de l'utilisateur de CMS permettent d'obtenir la distribution de probabilité cumulée du temps d'exécution des commandes (figure 28). Ces mesures ont été effectuées avec un CMS actif dans une machine virtuelle générée

par le système CP/67 sur 360-67. Nous avons ramené les temps d'exécution à ce qu'ils seraient sur un 360-40 qui est la machine sur laquelle fonctionne GMS.

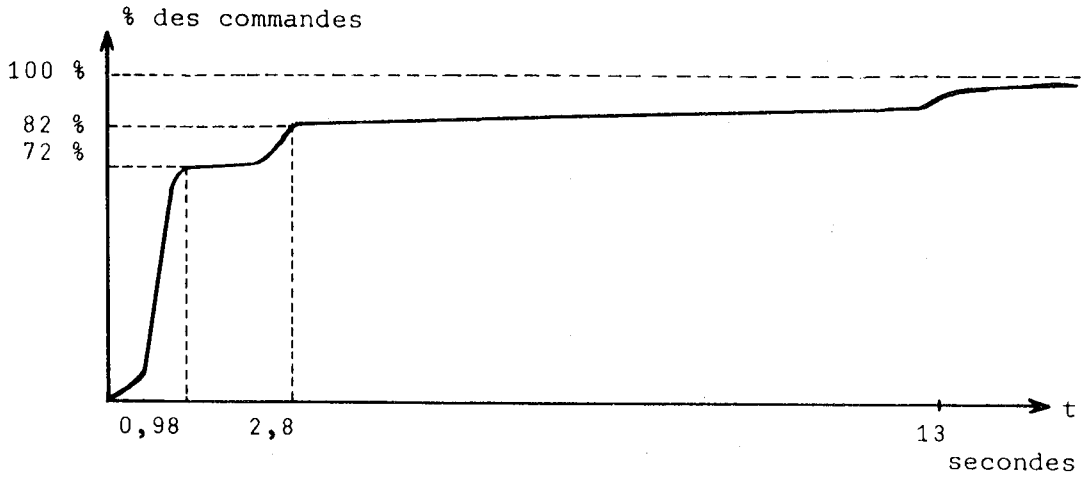


Figure 28. Temps d'exécution des commandes de CMS

Nous distinguons sur cette courbe deux paliers significatifs : 72 % des commandes sont exécutées en moins de 0,98 secondes et 82 % en moins de 2,8 secondes.

Pour respecter le principe d'indépendance des deux systèmes CMS et GMS, nous ne voulons pas faire connaître à GMS l'identité des commandes se situant au dessous de l'un ou l'autre des paliers. Cela serait d'ailleurs illusoire car une même commande peut se placer en des points différents de la courbe selon les caractéristiques des données qu'elle traite. Par exemple, une commande de copie d'un fichier de CMS dure d'autant plus longtemps que le fichier est plus volumineux. Nous pouvons cependant dire que si nous allouons le processeur à un CMS qui vient de recevoir une commande, il y a une probabilité de 0,72 pour que son exécution soit terminée en moins de 1 seconde. Ce temps étant inférieur au temps de commutation de deux CMS, nous pouvons en

déduire une première caractéristique de l'algorithme d'allocation : lors de la soumission d'une commande, le processeur est alloué au CMS concerné pour une durée au moins égale à 1 seconde (ou jusqu'à la fin d'exécution de la commande). De cette manière, 72 % des commandes ne nécessitent qu'un seul va-et-vient de CMS.

Si une commande utilise complètement ce premier quantum d'allocation t1 sans se terminer, nous ne faisons pas de supposition supplémentaire sur son temps estimé d'exécution ; le second palier n'augmente en effet que de façon marginale le pourcentage de commandes exécutées. Nous appelons commandes interactives les commandes dont le temps d'exécution se situe au dessous du premier palier et commandes non interactives toutes les autres. Toute nouvelle commande frappée est supposée interactive jusqu'à ce que, éventuellement, le dépassement de son premier quantum d'allocation t1 indique le contraire. Nous arrivons alors à une stratégie à deux files d'attente :

- FPRIOR (file prioritaire) dans laquelle se trouvent les CMS ayant soumis une commande supposée interactive ;
- FNPRIOR (file non prioritaire) dans laquelle se trouvent les CMS en cours d'exécution de commandes non interactives.

La file FPRIOR est gérée par la méthode FIFO et la file FNPRIOR par une technique de tourniquet ("round robin").

A partir du moment où une commande n'est pas interactive, il est inutile de hâcher trop finement son exécution par des va-et-vient successifs. Il ne serait pas cependant réaliste de lui laisser le contrôle du processeur pour un temps indéterminé (cas d'une exécution très longue), ce qui pénaliserait les CMS ayant soumis des commandes non interactives de durée inférieure. Nous

pouvons alors prendre en compte le second palier de la courbe en allouant le processeur pour une durée t2 telle qu'elle englobe les commandes de ce second palier. En fait, pour minimiser le coût du va-et-vient, nous avons choisi de prendre t2=10 secondes.

Le Contrôleur n'active un CMS de la file FNPRIOR que lorsque la file FPRIOR est vide. Nous donnons ci-dessous (figure 29) le diagramme des transitions entre files. Dans chaque file, la tête de file est située au sommet.

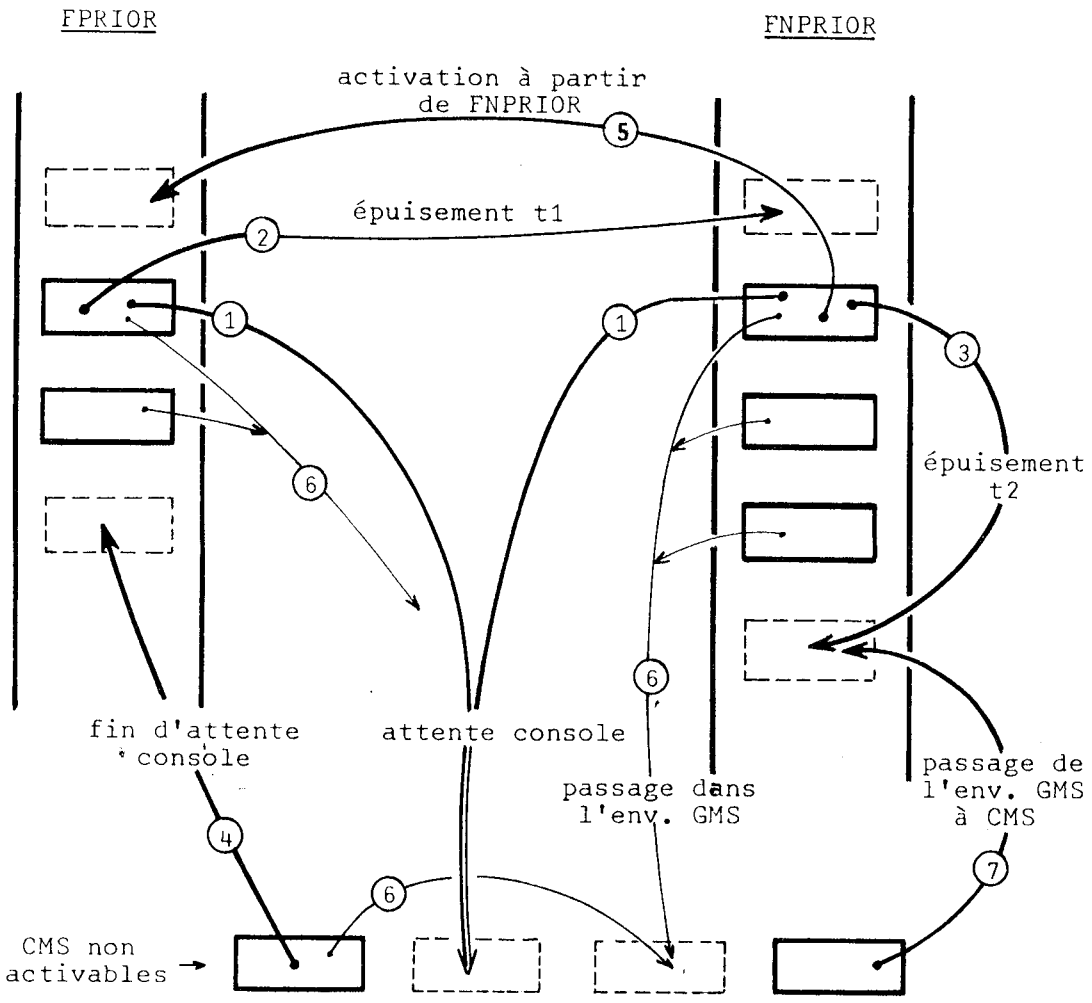


Figure 29. Schéma d'allocation du processeur aux CMS

① Le CMS actif, de l'une ou l'autre file, exécute une lecture sur sa console ou dépasse le nombre maximum de messages en attente d'impression sur terminal : il est enlevé de sa file.

② Le CMS actif, en tête de FPRIOR, épuise le quantum t1 : il est placé en tête de FNPRIOR.

③ Le CMS actif, en tête de FNPRIOR, épuise le quantum t2 : il est recyclé en queue de FNPRIOR.

④ Une condition d'attente sur terminal est satisfaite (lecture ou écriture) : le CMS correspondant rejoint la queue de FPRIOR.

⑤ Le CMS en tête de FNPRIOR va être activé (FPRIOR vide) : il est placé en tête de FPRIOR.

⑥ Un utilisateur dont le CMS se trouve dans l'une des files passe dans l'environnement GMS : ce CMS est enlevé de la file dans laquelle il apparaît.

⑦ Un utilisateur rejoint l'environnement CMS (commande 'CMS') : il est placé en queue de FNPRIOR.

Les transitions ②, ⑤ et ⑦ méritent quelques commentaires.

Transition ② : lorsqu'il s'avère qu'une commande supposée interactive ne l'est pas, il semblerait normal de placer le CMS correspondant en queue de FNPRIOR. Cependant, ce CMS est en mémoire et la décision de le placer en tête de FNPRIOR conduit à lui allouer au prix d'un seul va-et-vient un quantum t1+t2, sauf s'il y avait d'autres CMS derrière lui dans FPRIOR, auquel cas c'est le premier d'entre eux qui sera activé. Si, par contre, le CMS qui épuise t1 est seul dans FPRIOR, nous faisons l'économie d'une opération de va-et-vient.

Transition ⑤ : lorsque FPRIOR est vide, le CMS se trouvant en tête de FNPRIOR est activé. Entre l'instant de la décision et celui de la prise de contrôle par ce CMS, il s'écoule le temps d'un va-et-vient pendant lequel d'autres CMS peuvent rejoindre FPRIOR. Sans cette transition ⑤, nous nous trouverions alors dans la situation où, ayant payé le prix du va-et-vient, nous devrions vider ce CMS avant même de lui donner le contrôle. Pour rentabiliser cette opération de va-et-vient, nous accordons à ce CMS un minimum de temps de résidence en mémoire égal à t1. Au bout de ce quantum t1, la transition ② intervient : ce CMS garde alors le contrôle si la file FPRIOR est vide, il cède la place dans le cas contraire.

Transition ⑦ : lors d'un retour depuis l'environnement GMS dans l'environnement CMS, ce CMS rejoint la queue de FNPRIOR. Cela permet d'éviter qu'un utilisateur ayant soumis une commande à temps d'exécution long ne fausse le jeu des priorités par passages rapides et répétés d'un environnement dans un autre.

Une caractéristique de l'algorithme n'apparaît pas sur le diagramme des transitions : lorsque le CMS actif en tête de FNPRIOR est suspendu par suite d'une arrivée dans FPRIOR, il reste en tête de sa file mais avec un quantum résiduel égal à t2-t si t est la partie du quantum qu'il a utilisée. Cependant, si ce quantum résiduel est inférieur à une valeur t3, ce CMS est placé automatiquement en queue de FNPRIOR (③) de manière à ne pas réaliser plus tard une opération de va-et-vient pour un temps de résidence trop faible. Nous avons choisi de prendre t3 égal à la durée d'un va-et-vient.

Signalons enfin un problème lié à l'utilisation de la file

prioritaire : des utilisateurs interactifs peuvent se coaliser pour empêcher toute activation d'un CMS se trouvant dans la file non prioritaire. Il suffit pour cela qu'ils répondent suffisamment vite aux demandes de lecture de manière à ce que la file prioritaire ne soit jamais vide. Cette situation est détectée en comptant le nombre d'activations consécutives à partir de la file prioritaire. Si ce nombre dépasse une valeur limite N, alors le CMS en tête de la liste non prioritaire est activé sans tenir compte de la liste prioritaire.

Notons pour terminer que les paramètres du Contrôleur (t1, t2, t3 et N) peuvent être modifiés par une commande de l'opérateur, soit pour évaluer l'algorithme dans divers cas de figure, soit pour adapter la stratégie d'allocation à des circonstances particulières.

2.72 Accès logique aux ressources

Comme nous l'avons signalé en 2.4, l'hyperviseur étant conçu pour diriger le seul système CMS, il est possible de l'adapter à ce système [Bellino 73]. Examinons plus en détail la manière dont CMS utilise certaines ressources. Prenons pour cela l'exemple de l'imprimante. Lorsque CMS ou un programme s'exécutant sous son contrôle veut imprimer une ligne, il appelle le sous programme d'impression PRINT en lui fournissant l'adresse de la ligne à imprimer et certaines informations de contrôle. PRINT construit un programme canal, exécute l'instruction SIO puis se place en attente de la fin d'opération (en exécutant LPSW). Lorsque survient l'interruption signalant la fin d'opération (nous nous plaçons dans le cas où il n'y a pas d'hyperviseur au dessus de CMS), le contrôle est donné au module général d'interruptions

d'entrée-sortie, qui retourne à PRINT, qui lui-même retourne au programme initial.

Si nous utilisons l'approche machine virtuelle classique, l'hyperviseur, dans ce cas, n'est pas concerné jusqu'à ce que soit exécutée l'instruction SIO et cette instruction doit alors être simulée comme si elle commandait une imprimante réelle connectée à un canal réel. En particulier, l'hyperviseur doit créer et présenter à CMS les conditions d'interruption correspondantes.

Le traitement que CMS croit appliquer à la ligne à imprimer a peu de rapports avec ce que l'hyperviseur en fait réellement : ce dernier, par inspection du programme canal associé à l'instruction SIO, localise la ligne en mémoire et l'écrit dans un fichier intermédiaire. Dans notre cas, le principal problème pour CMS n'est pas de gérer une imprimante mais d'imprimer des lignes. Il est alors préférable de faire intervenir l'hyperviseur directement au niveau logique de la demande d'impression, c'est à dire lors de l'appel superviseur, particulier au sous-programme PRINT, que GMS peut intercepter. Ainsi, la détection de l'accès aux ressources à un niveau logique est possible : GMS peut filtrer les appels superviseur émis par CMS, traiter ceux correspondant à des accès logiques aux ressources et renvoyer les autres à CMS. Nous avons utilisé cette approche chaque fois que l'accès logique était détectable sans ambiguïté. Cela revient à prendre en considération dans CMS un niveau utilisateur et un niveau superviseur, le premier soumettant des demandes de ressources logiques que le second traduit en accès à des ressources physiques. La solution machine virtuelle classique consiste à attendre l'accès physique virtuel pour le transformer en accès physique réel. La solution système virtuel consiste à transformer directement l'accès logique en un accès physique réel (figures 30 a et 30 b).

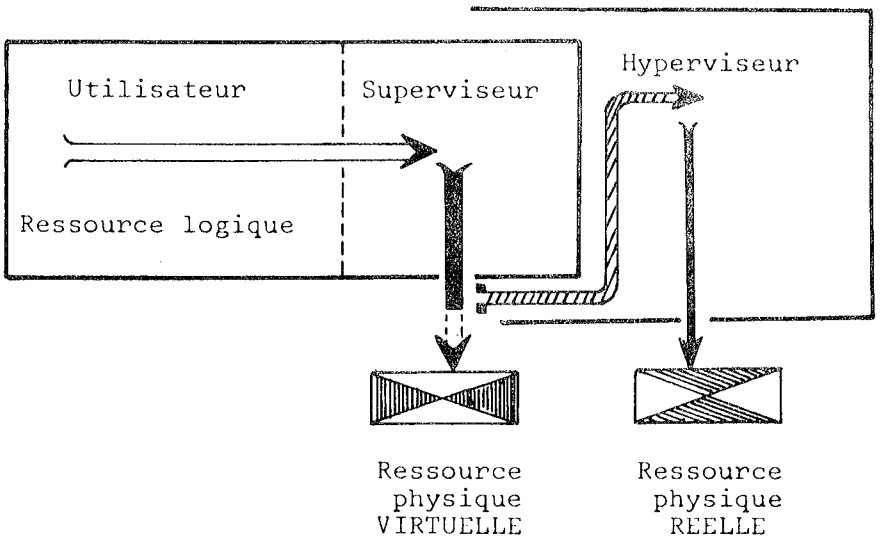


Figure 30 a. Simulation type machine virtuelle

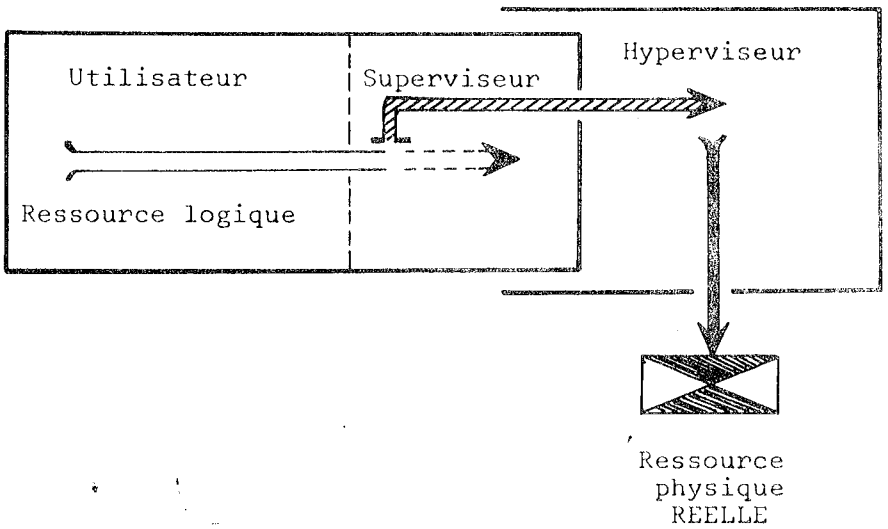


Figure 30 b. Simulation type système virtuel

2.73 Actions liées à l'activation d'un CMS

Le CMS actif quitte la mémoire dans deux cas :

1- il est dans la file non prioritaire et un autre CMS rejoint la file prioritaire (attente console satisfaite) ;

2- il épuise son quantum alors qu'il n'est pas seul dans sa file (quelle qu'elle soit).

Une première solution consiste à choisir le prochain CMS à activer au moment où ces situations apparaissent. Ce choix est simple : il s'agit de sélecter, dans le premier cas, le CMS qui vient de rejoindre la file prioritaire, dans le second cas le CMS suivant, dans la même file, celui qui a épuisé son quantum.

Entre l'instant de ce choix et celui du début de chargement en mémoire, il s'écoule le temps nécessaire à la recopie du CMS que l'on désactive, c'est à dire de l'ordre de une seconde. Pendant cet intervalle, le contenu des deux files a peut-être évolué : l'utilisateur du CMS que l'on s'apprêtait à activer a pu quitter l'environnement CMS (passage dans l'environnement GMS par utilisation de la touche Attention de son terminal), ou un CMS a pu rejoindre la file prioritaire alors que celui que l'on s'apprête à activer se trouve dans la file non prioritaire.

Il est donc préférable de différer la décision d'allocation le plus longtemps possible, c'est à dire jusqu'à la fin de la recopie du CMS courant. C'est cette seconde solution qui a été adoptée.

Lorsqu'un CMS a été rechargé en mémoire, il peut être nécessaire d'effectuer des opérations complémentaires avant de lui redonner le contrôle. Ces opérations sont de deux types.

1- Ecriture dans la mémoire de CMS : une opération de lecture

sur terminal correspondant à une lecture sur console exécutée par CMS s'effectue dans un tampon pris dans la mémoire libre de GMS (cf. 2.525). Ce tampon est transféré dans la mémoire de CMS avant activation ; il est ensuite libéré.

La mise en fonction de l'indicateur "kt" (lorsque la commande 'KT' a été frappée dans l'environnement GMS) a également lieu à cet instant.

2- Réflexion d'interruption : lorsque survient une interruption en provenance d'une unité contrôlée par un CMS, elle est mémorisée et lui sera réfléchie au moment de l'activation. Cette réflexion consiste à simuler la permutation des mots d'état programme et à inscrire dans la mémoire de CMS, à l'emplacement correspondant au mot d'état canal, le mot d'état canal obtenu lors de l'interruption (éventuellement complété et modifié par le processus qui pilotait l'opération en cours sur cette unité).

Ces opérations qui précèdent l'activation de CMS sont effectuées par le Contrôleur lui-même qui trouve les renseignements nécessaires dans le descripteur du CMS virtuel. Nous aurions aussi pu créer des processus de GMS pour les réaliser. Il aurait alors fallu rendre l'activation de ces processus dépendante de la présence en mémoire du CMS concerné. Cela aurait été un exemple de création de processus non rattachés à la liste des processus prêts, mais placés en attente d'un événement "CMS en mémoire" déclenché par le Contrôleur avant l'activation de CMS.

CHAPITRE 3

CONCLUSIONS

Il nous reste à dégager les enseignements de ce travail. Comme toute recherche en ce domaine, ce projet se termine en laissant à ses auteurs tout à la fois des sentiments de satisfaction et d'insatisfaction : satisfaction d'avoir pu mener cette étude jusqu'au bout et de voir ensuite fonctionner le système mis à la disposition de ses utilisateurs ; insatisfaction, lorsque, le projet arrivant à son terme, un examen critique des solutions adoptées suggère d'autres approches et fait découvrir des possibilités d'amélioration alors qu'il est trop tard pour les réaliser. Mais il faut savoir mettre un terme à un projet tel que celui qui vient d'être décrit ; les enseignements que l'on en tire trouvent alors leur application dans d'autres projets pour lesquels les domaines abordés ne sont pas forcément les mêmes.

Donnons pour commencer quelques éléments quantitatifs concernant cette réalisation.

Les phases de conception et de réalisation se sont étendues sur une période de deux années, de janvier 1970 à décembre 1971. Compte tenu d'autres travaux menés en parallèle par les personnes composant l'équipe, nous pouvons estimer que pendant ces deux années le projet a occupé deux personnes et demie à temps complet. Les deux mois suivants, janvier et février 1972, ont été consacrés à la mise au point définitive du système. Pour la réaliser, GMS a été mis à la disposition d'utilisateurs de bonne volonté pendant

deux à trois heures chaque jour. En mars 1972, GMS est devenu le système unique utilisé sur la machine de notre centre. Il l'est resté jusqu'en avril 1973, date à laquelle un changement de matériel nous a contraint à l'abandonner.

Pendant ces 13 mois d'emploi, le nombre moyen d'utilisateurs connectés pendant les heures de la journée a été de 3 ou 4. Le nombre d'incidents nécessitant un rechargement du système a été relativement réduit : de un par jour au début, ce nombre est rapidement passé à un par semaine. Plusieurs de ces incidents provenaient d'un mauvais ajustement des paramètres du système, notamment de la taille de la zone de mémoire laissée à la disposition de GMS. D'autres incidents provenaient de fautes de conception ou de programmation, par exemple d'une mauvaise récupération des ressources utilisées par un processus devant être détruit par suite d'une situation exceptionnelle (fonctionnement défectueux des entrées-sorties par exemple).

Les performances atteintes ne s'écartent pas des estimations initiales. Nous avons simplement dû, après quelques jours de fonctionnement, reconsidérer le problème de l'allocation du processeur pour faire intervenir les "facteurs humains", à la lumière des premières réactions des utilisateurs. Le nouvel algorithme du Contrôleur, tel que nous l'avons développé en 2.71, a alors permis d'augmenter le degré de satisfaction de ces derniers en favorisant les travaux interactifs.

Le système GMS occupe de l'ordre de 48 K octets de mémoire. Il se compose de 82 procédures. Le développement a été effectué en utilisant comme support le système CMS fonctionnant seul sur notre machine.

Un point important est l'utilisation d'un langage de niveau

élevé, dans notre cas PL/S, pour la réalisation du système. Nous avons depuis longtemps oublié les quelques réticences initiales et il nous apparaît que l'emploi d'un langage de plus haut niveau qu'un simple assembleur est une condition essentielle pour mener à bien un tel projet. La plus grande clarté des programmes produits, le gain de temps de programmation et de mise au point, sont des arguments en regard desquels les inconvénients éventuels, et nous l'espérons provisoires (augmentation de la taille et du temps d'exécution des programmes créés), peuvent être considérés comme secondaires.

Un langage tel que PL/S est bien adapté à l'écriture de systèmes. Parmi les caractéristiques qui nous paraissent les plus intéressantes, citons :

- la possibilité de définir et d'utiliser des structures de données élaborées ;
- la définition des variables de type pointeur ;
- la gestion automatique des registres associée à la faculté laissée au programmeur d'agir sur cette gestion (en indiquant par exemple qu'un registre est réservé pour une variable donnée) ;
- la possibilité de génération de toute instruction du répertoire de la machine.

L'expérience acquise en utilisant PL/S pour ce projet a d'ailleurs conduit d'autres chercheurs à définir un langage de même nature, GSL [Berthaud 72], dont l'implantation permet une meilleure vérification des actions sur les objets manipulés et réalise des contrôles de cohérence au moment du chargement pour les procédures compilées séparément.

Essayons maintenant de préciser ce qui, dans cette étude, est

transposable à d'autres applications, ce qui l'est moins, quels sont les domaines non abordés ou peu approfondis.

Dans GMS, la tâche d'un utilisateur de CMS est considérée comme un processus séquentiel unique. Il n'y a donc pas à ce niveau de notion de coopération et de synchronisation entre les processus d'un même utilisateur. Ces notions, nous les avons développées pour les processus du système lui-même et il y a là une différence essentielle : en effet, le schéma général de coopération et de synchronisation pour l'ensemble des processus du système peut être établi une fois pour toutes. Les adresses des variables utilisées pour réaliser la synchronisation sont connues des processus coopérants. Les outils employés pour réaliser la synchronisation ne nécessitent pas l'adjonction de mécanismes de protection. La totalité des processus a accès à l'ensemble des variables de contrôle du système. Si nous voulions prendre en compte des tâches utilisateur composées d'une famille de processus, il faudrait envisager la création dynamique de processus et la définition dynamique des variables utilisées pour la synchronisation (par exemple les événements). Il ne nous semble pas qu'une tentative d'uniformisation des outils employés au niveau des processus du système et au niveau des processus de l'utilisateur constitue un apport fondamental.

Nous n'avons pas eu à développer les problèmes de l'accès aux informations par les processus. Cela résulte de l'option initialement prise qui consiste à placer tous les processus dans un même espace d'adressage. Une telle option suppose que les processus sont fiables. Ici encore, cette solution n'est pas forcément celle qu'il faudrait employer pour les processus des utilisateurs.

Nous n'avons pas non plus abordé le problème de la protection

entre les processus de GMS ; les procédures composant les processus sont écrites et mises au point une fois pour toutes et la taille relativement réduite du système autorise cette approche, qui permet un gain d'efficacité. Par contre, le type d'interface développé entre l'utilisateur et le système est tel qu'il réalise une protection absolue de ce système contre toute action de l'utilisateur. Pendant toute la période de fonctionnement de GMS, il n'est apparu à aucun moment d'incident ayant pour origine une action des programmes des utilisateurs.

Compte tenu de l'exécution des processus du système en mode non interruptible, la plupart des accès aux variables partagées sont effectués sans faire intervenir explicitement des primitives d'entrée en section critique et de sortie de section critique. Un processus n'abandonne le processeur que sur sa propre initiative et l'accès exclusif aux variables lui est ainsi garanti lorsqu'il est actif. Lorsqu'un processus doit se bloquer (attente d'un événement) au cours de l'exécution d'une séquence d'instructions qui accèdent à des variables partagées, alors il utilise les primitives de section critique LOCK et UNLOCK.

A la réflexion, cette option est sans doute la plus criticable parmi celles prises lors de la conception de GMS. En effet, nous lions ainsi artificiellement un problème intrinsèque (les conditions d'accès aux variables partagées) à un problème technologique (l'allocation du processeur aux processus). Si nous décidions, une fois le système écrit, de changer l'algorithme d'allocation du processeur aux processus de GMS, en introduisant un quantum de temps par exemple, nous devrions alors réexaminer tous les processus et introduire l'emploi des primitives pour toutes les sections critiques.

Examinons maintenant les outils utilisés pour la synchronisation des processus. Nous avons introduit l'événement non mémorisé et l'avons utilisé à deux fins.

- Pour signaler l'apparition d'une situation qui n'a d'intérêt que si elle est attendue par un ou plusieurs processus. Nous avons étudié l'exemple de la libération de mémoire qui n'est un événement significatif que si des processus sont en attente de mémoire.

- Pour signaler l'apparition d'une situation qui concerne forcément un processus. La principale utilisation de ce type d'événement se trouve dans la réalisation des tâches d'entrée-sortie synchrones : un processus initialise une activité à laquelle est associé un événement qui sera déclenché lorsque l'activité sera terminée ; la demande de mise en attente est contenue de façon implicite dans la tâche soumise et se situe ainsi toujours avant l'occurrence de l'événement, ce qui permet l'utilisation d'événements non mémorisés. Dans d'autres circonstances, un processus est tenu d'examiner des informations attachées à l'événement (mais distinctes du descripteur d'événement lui-même) pour ne pas se placer en attente d'un événement déjà survenu. Il est clair qu'un événement mémorisé serait mieux adapté en pareil cas. Nous pouvons cependant remarquer que la solution employée est plus efficace puisqu'elle évite l'emploi d'une primitive de synchronisation dans le cas où l'événement est déjà survenu : le processus se contente d'examiner une information associée à cet événement. Cet argument perd sa valeur si nous devons employer des primitives de section

critique pour consulter cette information.

Aurions-nous pu utiliser d'autres mécanismes de synchronisation que ceux employés ? Certainement, et nous aurions également abouti à un système opérationnel mais dans lequel les relations entre processus auraient été sensiblement différentes, le découpage en processus également. Nous pouvons envisager deux types de synchronisation bien connus, la synchronisation à base de sémaphores [Dijkstra 67] et celle à base de messages [Hansen 70].

Le mécanisme des sémaphores diffère de celui des événements mémorisés par le fait essentiel qu'un sémaphore peut enregistrer un nombre quelconque de signaux d'activation alors qu'un événement n'en enregistre qu'un seul. Cependant, nous pouvons faire un certain nombre de remarques à ce propos. La première consiste à reconnaître que la nécessité d'utiliser des signaux d'activation multiples apparaît peu fréquemment dans les processus d'un système. Il est significatif de constater à cet égard que le modèle du producteur-consommateur utilisé pour gérer n tampons partagés entre deux processus est pratiquement le seul exemple qui figure abondamment dans la littérature concernant ce sujet. Dans GMS, une situation de ce type n'est apparue qu'une seule fois : il s'agit de la gestion des deux tampons dans le processus d'impression des fichiers images d'imprimantes virtuelles. Remarquons qu'il est toujours possible d'utiliser une variable globale pour mémoriser plusieurs signaux d'activation (c'est ce que nous avons fait dans l'exemple cité ci-dessus), à condition d'inclure les opérations sur cette variable dans une section critique. Cependant, lorsque la section critique doit être réalisée par l'emploi de primitives, le sémaphore est mieux adapté puisque la primitive d'activation (opération V) réalise

automatiquement la section critique dans laquelle est modifiée la variable (ici le sémaphore lui-même) et que l'on fait l'économie de la primitive de sortie de section critique.

Une seconde remarque, plus fondamentale celle-là, concerne la gestion des files d'attente. Les sémaphores réalisent une correspondance un pour un entre les signaux de blocage et d'activation (opérations P et V). Ils ne fournissent aucun moyen pour réveiller plusieurs processus lors de l'occurrence d'un événement qui les concerne tous (signalons cependant l'extension définie dans le système ESOPE par l'emploi de la primitive D [Kaiser 73]). Nous avons développé l'exemple de la gestion de la mémoire libre de GMS qui fait intervenir l'événement libération de mémoire, dont l'occurrence réveille tous les processus en attente de mémoire. Nous pouvons imaginer d'autres cas de réveil multiple. Par exemple, lorsqu'un processus particulier du système doit s'exécuter sans activation parallèle des autres processus, ces derniers sont tous placés en attente de l'événement "fin du processus particulier" et seront tous réveillés lors de l'occurrence de cet événement. Ce processus particulier crée par exemple une image globale du système à un instant donné ou modifie les variables de contrôle du système.

Dans le même ordre d'idées, les sémaphores ne permettent pas de lier le déblocage d'un processus à l'apparition de plusieurs signaux d'activation. Supposons par exemple que nous voulions réaliser l'allocation des pages dans une mémoire centrale paginée en utilisant un sémaphore NP représentant le nombre de pages libres et qu'à un instant donné un processus ait besoin de n pages. La primitive P(NP) ne permet que d'acquérir une page à la fois, alors que l'option prise lors de la conception de l'allocateur de ressources consiste peut-être à allouer ces n

pages en une seule fois.

L'exemple de la gestion de la mémoire libre de GMS et ce dernier exemple nous montrent que l'emploi des sémaphores pour la gestion des ressources n'est possible que si ces ressources sont quantifiées et sont acquises une unité à la fois.

La synchronisation par messages est une extension du mécanisme des sémaphores qui consiste à inclure, dans les primitives de synchronisation, la transmission d'informations qui seront accessibles au processus réveillé lorsqu'il sera rendu actif. Ainsi, lorsqu'un processus P active un processus Q, il peut par exemple lui transmettre son identité, la nature de l'action requise, ou encore toute autre information utile à Q. L'allocation de mémoire pour les messages à transmettre est un problème difficile à résoudre : cette allocation est en effet réalisée à l'intérieur de la primitive d'activation. Dans un système tel que GMS, où la coopération des processus est établie dès la conception, l'utilisation de variables communes suffit en général à résoudre le problème de la communication et une synchronisation par messages n'est pas absolument nécessaire. Remarquons cependant que nous avons utilisé une variante de cette méthode en permettant à un processus qui en crée ou en réveille un autre d'utiliser une extension de son descripteur pour lui transmettre de l'information.

Au cours du développement de GMS, nous nous sommes trouvés en présence de situations pour lesquelles les mécanismes mis en place n'étaient pas totalement satisfaisants. Nous aurions par exemple pu résoudre de façon plus élégante certains problèmes par l'utilisation d'événements masqués. Ce type d'événements a été utilisé dans MULTICS [Organick 72] et plus récemment dans le système GEMAU [Briat 73]. L'événement masqué est un événement

mémorisé auquel est associé un masque d'événement. Lorsqu'un signal d'activation est émis, il n'y a réveil d'un ou de plusieurs processus (selon les options choisies) que si le masque est hors fonction. Si le masque est en fonction, l'événement est mémorisé mais le réveil (passage dans la liste des processus prêts) n'interviendra que lorsque le masque sera remis hors fonction (par l'emploi d'une primitive). Cette technique permet de différer l'activation d'un processus qui n'est plus bloqué intrinséquement. Elle permet également d'établir des priorités d'activation pour les processus indépendantes de l'ordre des signaux d'activation. Les processus rejoignent la liste des processus prêts dans l'ordre où sont exécutées les primitives de mise hors fonction des masques.

En conclusion de cette discussion concernant les mécanismes de base employés pour la construction d'un système, nous dirons qu'il n'est pas souhaitable de définir a priori ces mécanismes et d'adapter ensuite les objectifs pour les rendre réalisables au moyen des outils définis. Nous préconisons au contraire l'approche qui consiste à mettre en place divers mécanismes, adaptés chacun aux situations particulières rencontrées. Il ressort en effet de ce qui précède qu'il est essentiel, de disposer de mécanismes permettant de gérer de diverses manières les files d'attente de processus.

Abordons enfin un point important souvent passé sous silence dans la littérature, celui des entrées-sorties.

Nous avons montré la difficulté qu'il y avait à considérer les tâches d'entrée-sortie comme autant de processus indépendants. Cette difficulté résulte du fait que l'initialisation d'une

opération d'entrée-sortie nécessite, en général, la disponibilité de plusieurs ressources. Un emploi rationnel de ces ressources suppose une gestion centralisée de ces demandes. Nous avons défini la notion de tâche d'entrée-sortie, différente de celle de processus, pour représenter une demande d'entrée-sortie tout au long de son exécution. Nous avons établi les relations de synchronisation entre une tâche et le processus qui la soumet. Nous avons mis en évidence l'intérêt des tâches asynchrones et proposé une solution pour résoudre le problème de l'intervention du demandeur pendant le déroulement d'une telle tâche. Nous avons également montré comment les interruptions de type asynchrone pouvaient être traitées dans une organisation centralisée telle que celle mise en place, par l'appel immédiat d'une procédure créant un processus à activation différée.

Pour terminer, nous pouvons évoquer quelques voies de recherche se situant dans le prolongement de l'étude que nous avons présentée.

Nous constatons que les outils de synchronisation examinés se trouvent placés à un niveau très bas ; les primitives sont assimilables à des instructions de la machine. Une mauvaise utilisation de ces primitives peut, entraîner l'incohérence des informations partagées ou un blocage du système. Il n'existe pas non plus de mécanisme qui oblige les processus à n'utiliser des informations partagées qu'à l'intérieur de sections critiques. Des propositions en ce sens ont été formulées récemment [Hoare 73, Hansen 73]. Elles consistent, en particulier, à définir des "régions critiques" et à leur associer l'identification des variables partagées qu'elles manipulent. Il reste à approfondir ces concepts et à montrer leur applicabilité à la construction

d'un système.

Une seconde direction est suggérée par la remarque suivante. Nous mettons en place des mécanismes destinés à régler les conflits résultant du partage des ressources dans une situation de concurrence. Les primitives de section critique en sont une illustration. Dans certaines applications, les conflits sont rares : par exemple, il n'y a pas, en général, de processus dans la section critique ou en attente d'y entrer. Nous pourrions alors imaginer des primitives telles que leur emploi ne coûte rien (ou presque) en l'absence de conflit. La proposition suivante constitue une approche possible. Elle est basée sur l'emploi de deux instructions à rajouter au répertoire de la machine, SET (verrou) et RESET (verrou), dans lesquelles 'verrou' est l'adresse d'un mot de mémoire (ou peut-être d'un registre particulier sur lequel seules sont permises les opérations SET et RESET). Un verrou peut se trouver dans trois états différents, matérialisés par trois valeurs différentes de son contenu :

- 0 : le verrou est ouvert ;
- 1 : le verrou est fermé ;
- 2 : le verrou est fermé et des demandes d'ouverture existent.

Les instructions SET et RESET réalisent les opérations suivantes :

```
SET :   si VERROU = 0 alors VERROU := 1
        sinon début  VERROU := 2 ;
                     interrompre le processus courant
                     (interruption technologique)
              fin
```

```
RESET : si VERROU = 1 alors VERROU := 0
        sinon début VERROU := 0 ;
                    interrompre le processus courant
                    (interruption technologique)
              fin
```

Nous voyons qu'un tel mécanisme permet, par exemple, de réaliser l'entrée en section critique et la sortie de section critique au moyen d'une seule instruction lorsqu'il n'y a pas conflit. En cas de conflit, mais seulement dans ce cas, le système est alerté et peut alors régler ce conflit.

Un mécanisme tel que celui-ci semble particulièrement adapté lorsque l'emploi des primitives nécessite un changement de contexte, généralement coûteux (appel au superviseur pour réaliser la primitive). Son emploi pourrait être très utile pour régler, au moindre coût, les conflits d'accès dans une base de données.

Ceci n'est évidemment qu'une proposition initiale, qui peut être développée. Une réalisation de ce type, utilisée à d'autres fins, est exposée dans [Carver Hill 73].

BIBLIOGRAPHIE

Auroux 66 — Auroux A., Bellino J.
Système en temps partagé 1401/7044 en mode moniteur et mode conversationnel
Monographie d'informatique n° 2, 1969

Bellino 69 — Bellino J., Potin Ph.
Local Conversationnal Monitor System
SEAS meeting proceedings, 9/69

Bellino 70 a — Bellino J., Potin Ph.
Sous-système conversationnel fonctionnant sous OS/360
Etudes n° FF2-0106 et FF2-0114
Développement Scientifique, IBM France

Bellino 70 b — Bellino J.
Superviseurs d'entrées-sorties dans un contexte de multiprogrammation
Cours C4 maîtrise d'informatique, IMAG Grenoble 1970

Bellino 72 — Bellino J., Potin Ph.
Mécanismes d'un hyperviseur
Congrès AFCET, Grenoble 1972

Bellino 73 — Bellino J., Hans C.
Virtual Machine or Virtual Operating System ?
ACM Workshop on Virtual Computer Systems, Harvard University, march 1973

Bellot 68 — Bellot M., Siret L., Verjus J.P.
DIAMAG 2 : Système conversationnel à accès multiples
Séminaires de programmation, IMAG Grenoble 1968

Berthaud 72 — Berthaud M., Clauzel D., Jacolin M.
GSL : définition du langage
Etude n° FF2-0133
Développement Scientifique, IBM France

Bolliet 67 — Bolliet L., Auroux A., Bellino J.
DIAMAG : a multi access system for on line Algol programming
Proceedings AFIPS vol. 30, SJCC 1967

Briat 73 — Briat G.
Processus et synchronisation dans le système GEMAU
A paraître

Brittenham 73 — Brittenham W.R.
The Systems Programming Language Problem
IFIP TC/2 Working Conference on Machine Oriented Higher Level Languages, Trondheim, August 1973

Carver Hill 73 — Carver Hill J.
Synchronizing Processors with Memory Content Generated Interrupts
CACM 16,6 1973

Coffman 71 — Coffman E.G., Elphick M.J., Shoshani A.
System Deadlocks
ACM Computer Surveys, vol. 3 n.2 1971

Crocus 73 — Ouvrage collectif
Systèmes d'exploitation des ca'culateurs, principes de conception
A paraître chez Dunod

Denning 71 — Denning P.J.
Third Generation Computer Systems
ACM Computer Surveys, vol. 3 n 4 1971

Dennis 66 — Dennis J.B., Van Horn E.C.
Programming semantics for multiprogrammed computations
CACM 9,3 1966

Dijkstra 67 — Dijkstra E.W.
Cooperating Sequential Processes
in Programming Languages, NATO advanced study institute, F. Genuys editor, Academic Press

Dijkstra 71 — Dijkstra E.W.
Hierarchical Ordering of Sequential Processes
Acta Informatica, vol. 1 1971

Faure 71 — Faure J.P., Servant P.
Gestion de fichiers du système GMS/40
INP Grenoble, projet de fin d'études, juin 1971

Finet 73 a — Finet L.
Gestion des ressources physiques dans un système avec contraintes de swapping
Séminaires de programmation, IMAG Grenoble janvier 1973

Finet 73 b — Finet L.
Entrées-Sorties dans les Systèmes d'exploitation
Thèse de troisième cycle, USMG Grenoble 1973

Goldberg 69 — Goldberg R.P.
Virtual Machine Systems
MIT Lincoln Lab. Report MS-2687

Goldberg 73 — Goldberg R.P.
Architectural Principles for Virtual Computer Systems
Ph. D. Thesis, Harvard University 1973

Haberman 69 — Haberman A.N.
Prevention of System Deadlocks
CACM 12,7 1969

Hansen 69 — Hansen P.B.
RC4000 Software Multiprogramming System
A/S Regnecentrallen, Copenhague 1969

Hansen 70 — Hansen P.B.
The Nucleus of a Multiprogramming System
CACM 13,4 1970

Hansen 73 — Hansen P.B.
An Approach to Multiprogramming
A paraître dans ACM Computer Surveys

Hoare 73 — Hoare C.A.R., Perrot R.H.
Towards a theory of parallel programming
in Operating Systems Techniques
Academic Press, New york, 1973

IBM 67 — IBM Corporation
IBM System/360 Operating System, Supervisor and Data Management Services
Form C28-6646

IBM 69 a — IBM Corporation
Control program 67/Cambridge Monitor System
User's Guide
Form GH20-0859

IBM 69 b — IBM Corporation
Control Program 67
Program Logic Manual
Form GY20-0590

IBM 69 c — IBM Corporation
IBM System/360 Time Sharing System
System Logic Summary PLM
Form GY28-2009

IBM 70 — IBM Corporation
IBM System/360 Operating System
Input/Output Supervisor PLM
Form GY28-6616

IBM 72 — IBM Corporation
Guide to PL/S Generated Listings
Form GC28-6786

Kaiser 73 — Kaiser C.
Conception et réalisation de systèmes à accès multiples : gestion du parallélisme
Thèse d'état Paris 1973

Knuth 68 — Knuth D.
The Art of Computer Programming, volume 1
Adison Wesley Reading Massachussets

Krakowiak 73 — Krakowiak S.
Conception et réalisation de systèmes à accès multiples : allocation de ressources
Thèse d'état Paris 1973

Lampson 68 — Lampson B.W.
A Scheduling Philosophy for Multiprocessing Systems
CACM 11,5 1968

Leroudier 72 — Leroudier J.
Analyse d'un système à partage de ressources
Séminaires de programmation, IRIA décembre 1972

Margolin 71 — Margolin B.H., Parmelee R.P., Schatzoff M.
Analysys of free storage algorithms
IBM Systems Journal vol. 10 n 4 1971

Organick 72 — Organick E.I.
The MULTICS System : an experimentation of its structure
MIT Press, 1972

Potin 72 — Potin Ph.
Système GMS - Guide de l'utilisateur
Note interne GMS/27 Centre Scientifique IBM Grenoble

Potin 73 — Potin Ph.
An Experience of Systems Programming in a High Level Language
Etude N° FF2-0146
Développement Scientifique, IBM France

Saltzer 66 — Saltzer J.H
Traffic Control in a Multiplexed Computer System
MAC TR-30 (Ph.D. Thesis), MIT 1966

Verjus 68 — Verjus J.P.
Etude et réalisation d'un système Algol conversationnel
Thèse Docteur-Ingénieur, Grenoble 1968