Protection d'une mémoire virtuelle répartie par capacités implicites

Frederic Saunier

THESE

Présentée à

l'Institut National
Polytechnique de Grenoble

par

Frédéric Saunier

Protection d'une mémoire virtuelle répartie par capacités implicites

Thèse soutenue devant la commission d'examen le :

25 octobre 1996

Guy Mazaré	Président
Claude Bétourné	Rapporteur
Claude Kaiser	Rapporteur
Jacques Mossière	Directeur de thèse
Jacques Briat	Examinateur
Sacha Krakowiak	Examinateur

Je tiens à remercier l'ensemble des personnes qui, par leurs conseils, leurs encouragements ou leur aide, ont contribué à l'aboutissement de ce travail :

Jacques Mossière, Professeur à l'Institut National Polytechnique de Grenoble, qui a patiemment encadré ce travail, et plus particulièrement pendant la phase de rédaction ;

Guy Mazaré, Directeur de l'École Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble, qui m'a fait l'honneur d'accepter de présider le jury ;

Claude Kaiser, Professeur au Conservatoire National des Arts et Métiers, et Claude Bétourné, Professeur à l'Université Paul Sabatier, qui ont accepté d'être rapporteurs de ce document, et qui, par leurs critiques constructives, m'ont permis de l'améliorer ;

Sacha Krakowiak, Professeur à l'Université Joseph Fourier, pour la confiance qu'il m'a accordée et pour avoir bien voulu faire partie du jury ;

Jacques Briat, professeur à l'Université Joseph Fourier, pour les précieux conseils qu'il m'a apportés, et qui a bien voulu faire partie du jury ;

Roland Balter, Professeur à l'Université Joseph Fourier, Directeur du projet Sirac, qui m'a manifesté un soutien amical tout au long de ces années de recherches dans son équipe ;

Jay, Thierry, Pascal, Elizabeth, Fernando et Alain pour l'ensemble du travail qu'ils ont réalisé pour la mise œuvre de la plate–forme Arias ;

L'ensemble du personnel de feu l'Unité Mixte Bull–IMAG/Systèmes, et des projets Sirac et Opéra pour la chaleureuse ambiance qui a régné dans nos locaux. Je tiens en particulier à remercier sincèrement Béatrice et Joëlle pour les nombreux services qu'elles m'ont rendus, ainsi que leur présence et leur bonne humeur ;

Enfin, et non les moindres, Daniel, Christian et Christine pour leurs encouragements et leur amité qui m'ont été des plus précieux notamment au cours de la dernière année de cette thèse.

à Louisa...

Introduction

1 Les objectifs de l'étude

Avec l'arrivée des processeurs 64–bits qui rendent disponible un très large espace d'adressage, plusieurs projets de recherche se consacrent à construire des systèmes répartis qui offrent une mémoire virtuelle répartie avec une désignation uniforme. Pour fournir des mécanismes de contrôle d'accès sur une telle mémoire, ces systèmes répartis trouvent intérêt et avantage à utiliser des mécanismes à base de capacités. Ces mécanismes conçus à l'origine pour des systèmes centralisés et des architectures spécialisées, sont apparus à la fin des années 60, mais leur utilisation est restée marginale et limitée à quelques exceptions. Aujourd'hui, le concept de capacité et les mécanismes associés ont considérablement évolué et se sont adaptés aux architectures conventionnelles, ainsi qu'aux systèmes répartis. Les projets de mémoires virtuelles réparties uniformes utilisant des adresses de 64 bits ouvrent donc un champ d'application aux mécanismes de contrôle d'accès à base de capacités, et donnent un nouvel essor au concept. Néanmoins les systèmes répartis qui utilisent des capacités se déchargent en grande partie sur les applications et sur les utilisateurs pour la gestion des capacités, en invoquant un objectif de souplesse d'utilisation. Ainsi, les utilisateurs doivent être conscients qu'ils utilisent une mémoire virtuelle dont l'accès est contrôlé par des capacités, et ils doivent explicitement gérer les capacités à leur «compte» et les présenter systématiquement à chaque accès.

Les objectifs de cette étude sont de définir une alternative à ces mécanismes existants qui permette :

- de conserver leur souplesse d'utilisation,
- de rendre implicite toute utilisation de capacité lors d'un accès à la mémoire virtuelle répartie, afin de rendre uniforme la désignation de cette mémoire, quel que soit le type de contrôle d'accès mis en œuvre,
- d'intégrer toute la gestion du contrôle d'accès à la mémoire dans une interface d'administration de la protection indépendante de l'utilisation et de la désignation de la mémoire virtuelle répartie.

2 Le cadre du travail

Le travail de recherche que j'ai réalisé s'est déroulé au sein du projet SIRAC (**S**ystèmes **I**nformatiques **R**épartis pour **A**pplications **C**oopératives). C'est un projet commun aux instituts de recherche IMAG et INRIA, dont le but est la conception d'un environnement pour le développement d'applications coopératives réparties. Plus précisément, l'objectif du projet est double : il s'agit d'étudier les divers besoins des applications d'une part, et de fournir une plate−forme expérimentale pour valider et évaluer les résultats du projet, d'autre part.

En particulier, l'étude présentée ici se situe dans le cadre de la définition et de la réalisation d'une plate−forme offrant un service de gestion des données persistantes réparties et partagées. En effet, on souhaite également intégrer à cette plate−forme un service de protection d'accès aux données partagées. Les objectifs spécifiques de ce service de protection se traduisent par la volonté :

- d'offrir un service de contrôle d'accès qui ne nécessite pas de modifier les applications qui accèdent à la mémoire persistante répartie et partagée ; la plate−forme peut ainsi être configurée pour intégrer le service de contrôle d'accès ou pour ne pas l'utiliser du tout, sans qu'il soit pour cela nécessaire de modifier en quoi que ce soit les applications,
- d'offrir un service de contrôle d'accès configurable qui permette de réaliser différents modèles de protection dont les applications coopératives ont besoin.

Il convient de noter ici que cette plate−forme doit servir de base d'expérimentation. Par conséquent, un certain nombre de contraintes ou d'inconvénients potentiels apparaissent :

- la volonté d'effectuer un prototypage rapide oriente les choix de programmation vers l'utilisation d'interfaces standard et documentées. Ces choix d'architecture globale de la plate−forme ont été réalisés en vue d'un portage futur et d'une intégration rapide des différents modules et services développés. Ils imposent donc des contraintes supplémentaires à la réalisation des services, en particulier pour le service de protection qui nous intéresse ici ;
- la mise en avant de cet objectif de prototypage rapide implique également que la recherche de performances n'est pas le but principal de cette première réalisation de la plate−forme.

3 L'approche adoptée

Le travail présenté ici s'est déroulé en deux étapes. Pendant une première phase, je me suis attaché à définir et proposer un service de contrôle d'accès qui réponde aux objectifs fixés et à identifier les difficultés potentielles de réalisation de ce service.

Le principe directeur de cette proposition réside dans l'utilisation implicite de capacités dans la désignation des données réparties. En d'autres termes, une application utilise exclusivement des adresses virtuelles pour pouvoir s'exécuter de la même manière et sans modification de son code :

- dans différentes configurations de la plate−forme sous−jacente qui réalise la mémoire virtuelle répartie,
- dans différentes configurations des règles de contrôle d'accès définies pour les données réparties.

Cette proposition a également été consolidée par l'étude des solutions proposées par d'autres projets de recherche similaires, notamment pour identifier les convergences et les particularités des différentes solutions proposées, et leurs limites par rapport aux objectifs spécifiques de SIRAC.

La réalisation de cette proposition est intervenue dans une deuxième étape et elle s'est traduite par :

- la recherche de solutions qui a permis d'utiliser des techniques existantes répondant à nos besoins,
- l'étude de l'adaptation et de l'intégration de ces techniques dans l'architecture du premier prototype,
- la mise en œuvre et l'intégration effective des mécanismes proposés dans la première maquette de la plate−forme,
- le développement d'applications spécifiques pour le test et l'évaluation de la réalisation faite.

Cette mise en œuvre a permis de montrer que les objectifs souhaités d'un contrôle implicite d'accès à la mémoire virtuelle répartie sont tout à fait viables. La réalisation de nos objectifs concernant l'administration de la protection semble également atteinte, mais son évaluation n'est pas consolidée par l'utilisation par une application réelle.

4 Présentation du plan de la thèse

Ce document est constitué de trois parties. La première partie (chapitres I et II) présente les problèmes posés par la protection dans les systèmes à capacités, ainsi que les approches adoptées dans les systèmes répartis offrant une grande mémoire virtuelle. Une deuxième partie (chapitre III) présente ensuite les objectifs recherchés dans le cadre de cette étude. Enfin une troisième partie (chapitres IV et V) présente les principes de la solution que j'ai proposée pour réaliser ces objectifs, sa mise en œuvre et l'évaluation des résultats ainsi obtenus.

Chapitre I

Ce chapitre présente d'abord les concepts généraux des systèmes à capacités, puis un tour d'horizon de l'historique des réalisations de systèmes à capacités depuis leur première définition jusqu'à leur utilisation dans les systèmes répartis. Cette étude permet principalement, d'introduire dans un premier temps deux aspects des systèmes à capacités :

- les techniques de réalisation des informations de contrôle d'accès, c'est–à–dire des capacités et des enceintes de protection,
- les mécanismes de contrôle de l'évolution des droits d'accès définis à l'intérieur d'une enceinte de protection.

Ces deux premiers aspects du contrôle d'accès avec des capacités sont ensuite repris dans un état de l'art pour les systèmes offrant une grande mémoire virtuelle répartie et uniforme.

Chapitre II

Ce chapitre aborde un autre aspect du contrôle d'accès nécessaire dans un système à capacités : il s'agit du transfert d'exécution d'une enceinte de protection à une autre, et du contrôle des communications ainsi réalisées entre les différentes enceintes de protection définies dans un système à capacités. De la même façon qu'au chapitre précédent, il est d'abord présenté ici un tour d'horizon de l'historique et l'éventail des problèmes abordés et des solutions possibles. Ces définitions permettent ensuite d'exposer l'état de l'art en matière de contrôle de changement d'enceintes de protection dans les systèmes offrant une grande mémoire virtuelle répartie et uniforme.

Chapitre III

C'est le cadre de la thèse, le projet SIRAC, qui fait l'objet de ce chapitre. Les motivations et les objectifs généraux sont d'abord décrits pour définir le contexte de l'étude. Puis nous présentons les besoins et objectifs

spécifiques d'un service de protection dans le cadre d'un service global et général pour la gestion des données persistantes, réparties et partagées. L'ensemble de ces objectifs correspondent aux motivations de cette étude.

Chapitre IV

Ce chapitre présente dans une première partie les principes du service de protection que nous proposons pour répondre aux besoins décrits au chapitre précédent. Dans sa seconde partie, ce chapitre traite les différents aspects que la réalisation de ce service a permis d'aborder, et les solutions techniques que nous avons appliquées.

Chapitre V

Enfin, ce chapitre donne une évaluation du travail effectué et des résultats obtenus. Cette évaluation qualitative est également consolidée par une évaluation quantitative lorsque des mesures significatives la permettent sur le premier prototype de la plate−forme d'expérimentation.

Conclusion

Cette partie conclut ce document en rappelant les objectifs fixés ainsi que les résultats obtenus, et présente des perspectives de prolongement possible de ce travail.

Chapitre I

Capacités et mémoires réparties partagées

Dans les années 70, plusieurs systèmes d'exploitation centralisés sont basés sur l'utilisation de capacités, et [Levy 84] en présente de nombreux exemples. L'intérêt reconnu d'un tel mécanisme se situe dans son uniformité pour désigner tous les objets–mémoire du système, pour y accéder et pour en protéger le mode d'accès. Dans chacun de ces systèmes, les capacités sont entièrement gérées dans le code du noyau, et elles nécessitent souvent un support matériel spécifique.

C'est pourquoi l'utilisation de capacités comme mécanisme de base pour protéger les accès à la mémoire dans les systèmes répartis est longtemps restée marginale, et limitée à quelques exceptions comme le système Amœba [Tanenbaum 86]. Avec l'arrivée de processeurs 64–bits, de nombreux projets comme Angel [Murray 93a], Mungi [Heiser 93b] ou Opal [Chase 92] se sont consacrés à construire des systèmes répartis offrant une mémoire virtuelle et une désignation uniformes. De fait, ces systèmes répartis trouvent donc intérêt et avantage à utiliser des mécanismes à base de capacités, donnant ainsi un nouvel essor au concept.

I.1 Objectifs généraux de protection

Quelques définitions s'imposent pour cerner les différentes particularités des capacités et situer celles qui nous intéressent ici. Plusieurs rapports de recherche [Ferrié 75][Vivo 95] en donnent un aperçu. Il convient de définir notamment ce que représente exactement une capacité et ce qu'elle permet de protéger, ainsi que l'intérêt de ce mécanisme de protection dans le contexte ouvert et dynamique d'un système réparti. Enfin, nous verrons quelles sont les techniques usuelles pour réaliser un système à capacités, leurs motivations, leurs forces et leurs faiblesses.

I.1.1 La matrice de protection

La protection des entités d'un système est très couramment représentée par une matrice dont les lignes indiquent les utilisateurs du système – le terme utilisateur désigne indifféremment, ici et dans toute la suite, des personnes physiques, des applications ou des processus, – et les colonnes, les entités protégées. [Lampson 74] formalise cette notion de matrice qui représente à un instant donné la protection associée à chaque entité du système et à chaque utilisateur. Conformément à ces travaux, le terme de référence est celui de matrice de protection du système.

(Utilisateurs, Entités)	Fichier A	Coupleur B	C–liste C
Utilisateur X	Lecture	OUT	Ajout/Retrait, Recherche, Copie, Destruction
Utilisateur Y	–	IN	Recherche
Utilisateur Z	Lecture, Écriture	–	Ajout/Retrait

Fig. 1.1 : Matrice de protection

Comme l'indique l'exemple Fig. 1.1, pour connaître les droits associés à un utilisateur X, pour une entité A, il suffit de lire la cellule correspondante (X,A) de la matrice. Étant donné le nombre élevé d'entités et d'utilisateurs connus d'un système, il est inconcevable de stocker ces informations sous forme de matrice. De plus, cette matrice est essentiellement creuse. Pour ces deux raisons, l'information est donc factorisée :

- par listes de contrôle d'accès, c'est–à–dire que l'on stocke, pour chaque entité, l'ensemble des utilisateurs qui y ont accès et les opérations autorisées sous la forme de couples (opération, utilisateur). En résumé, cela consiste à lire la matrice par colonnes. Ainsi, dans l'exemple, l'information de protection sera conservée, entité par entité, sous la forme :
  - pour le fichier A : { (utilisateur X, opération Lecture), (utilisateur Z, opérations Lecture / Écriture) },
  - pour le coupleur B : { (utilisateur X, instruction OUT), (utilisateur Y, instruction IN) }, et ainsi de suite...
- par capacités, il s'agit alors de stocker par utilisateur l'ensemble des entités auxquelles il peut accéder, et les droits qui lui sont associés. Dans ce cas, la

matrice se lit par lignes, c'est–à–dire que, dans l'exemple, l'information sera conservée sous la forme :

- pour l'utilisateur Y : { (coupleur B, instruction IN), (C–liste C, opérations Ajout / Retrait / Recherche / Copie / Destruction) },
- pour l'utilisateur Z : { (fichier A, Lecture / Écriture), (C–liste C, Ajout / Retrait) }

I.1.2 Gestion de la protection et évolution des droits

En apparence identiques, les deux formes de protection que nous venons de voir n'offrent pas la même souplesse de manipulation des droits.

En effet, le modèle de protection par listes de contrôle d'accès est simple à réaliser, et il est très répandu dans les systèmes de gestion de fichiers, par exemple. Il permet d'associer à chaque fichier protégé la liste des utilisateurs susceptibles d'y accéder, ainsi que les droits accordés à ces utilisateurs. Cette liste fait intégralement partie (par inclusion ou par référence indirecte) du descripteur du fichier. À chaque référence d'accès au fichier, la vérification des droits s'effectue immédiatement dans les tables de gestion–mémoire (tables de pages ou de segments) de l'espace de l'utilisateur où les droits accordés auront été mis à jour lors du chargement du descripteur du fichier. Cependant, il convient de noter que cette réalisation de la protection est étroitement couplée avec l'identité de l'utilisateur. De ce fait, elle s'adapte plus particulièrement à une utilisation dans un contexte où la configuration de la protection reste statique. Par contre, dans un contexte tel que celui des applications coopératives et réparties qui requièrent une évolution très fréquente et un contrôle à grain fin des règles de protection, des listes de contrôle d'accès par utilisateur devraient être mises à jour constamment, avec un surcoût prohibitif.

Plusieurs opérations de protection participent à l'évolution des règles de protection. Ces opérations se définissent ainsi :

ajout/révocation c'est l'opération de configuration la plus élémentaire. Il s'agit tout simplement d'ajouter, dans un contexte de protection, un droit d'accès à une nouvelle entité,ou de modifier ou supprimer un droit existant. Dans le cas des listes de contrôle d'accès, cette opération consiste à insérer, modifier ou détruire un élément de la liste concernée. Dans le cas des capacités, l'ajout d'un droit d'accès s'effectue simplement par ajout ou modification d'une capacité dans le contexte de protection ; par contre, la révocation est plus délicate dans la mesure où, pour retirer un droit d'accès, elle

	nécessite de connaître l'ensemble des capacités qui confèrent ce droit d'accès à l'entité protégée.
amplification	c'est l'opération d'extension temporaire des droits d'un utilisateur. Il s'agit donc d'accorder à un processus ou une application des privilèges supplémentaires à ceux dont dispose l'utilisateur appelant. Ce processus exécute pour le compte de l'utilisateur appelant une action particulière sur une entité à laquelle l'utilisateur n'a pas accès autrement.
restriction	c'est l'opposée de l'amplification : il s'agit ici de réduire les privilèges de l'appelant.
délégation	c'est l'opération transposée de l'amplification : il s'agit en effet d'accorder à un processus un privilège délégué par l'utilisateur appelant. Le processus s'exécute dans son propre contexte, augmenté temporairement des droits délégués par l'appelant.
isolation	c'est l'opposée de la délégation : il s'agit d'exécuter une action dans un contexte confiné, indépendant de celui de l'appelant.

Il est difficile de réaliser un contrôle fin et systématique de l'amplification et de la délégation des droits avec un mécanisme de protection tel que les listes de contrôle d'accès, car celui−ci est contraint à une hiérarchie statique d'utilisateurs prédéfinis. C'est pourquoi les modèles de protection des applications coopératives réparties préfèrent souvent utiliser des mécanismes à base de capacités. La description des mécanismes d'amplification et de délégation dans les systèmes à capacités fait l'objet du chapitre suivant, et il convient auparavant de détailler la notion de capacité, et de définir les différentes techniques de réalisation.

I.2 Protection et capacités

I.2.1 Définition du concept

La première définition du concept de capacité a été proposée par Dennis et Van Horn en 1966 [Dennis 66]. Cette définition, détaillée plus loin, est purement abstraite : elle ne repose sur aucun modèle d'exécution particulier, et ne présume en rien des réalisations possibles. Cette définition a ensuite été reprise et utilisée dans divers contextes matériels ou logiciels, et a donné lieu à de nombreuses réalisations.

[Levy 84] en décrit un panorama historique complet que nous ne referons pas ici : une classification synthétique de ces réalisations nous permettra d'identifier les caractéristiques de visibilité qui nous intéressent plus particulièrement.

De manière générale, une capacité est un ticket, une clé qui confère à son détenteur le droit d'accéder à une entité dans le système (objet mémoire, ressource matérielle ou autre). Une capacité se compose :

1. d'un identificateur unique pour l'entité désignée
2. d'un ensemble de droits d'accès à cette entité

Chaque utilisateur, qu'il représente une personne, un programme ou encore une application, détient une liste de capacités ; puisqu'elles constituent l'unique moyen de désignation, ces capacités définissent l'ensemble des entités du système auxquelles l'utilisateur peut accéder. Ainsi, toute opération s'effectuant sur une entité particulière s'écrira :

```
opération (capa, param1,...)
```

où `capa` indique d'une part l'objet concerné par l'opération, et d'autre part si l'opération est autorisée ou non.

Les capacités constituent d'abord un moyen de désignation uniforme pour toutes les entités du système, pour tout objet–mémoire, qu'il soit temporaire ou permanent, et quelle que soit sa localisation en mémoire (objet résidant en mémoire principale ou en mémoire secondaire). Une capacité peut donc être considérée comme un nom pour une entité du système.

De plus, les capacités forment également la brique de base de la protection, et le système doit donc assurer qu'un utilisateur ne puisse ni modifier les informations contenues dans une capacité, ni forger de fausses capacités. Afin de préserver cette garantie, l'utilisateur n'est donc pas autorisé à modifier directement la liste des capacités où sont rangés les descripteurs contenant les informations de désignation et de protection pour chaque capacité. Seuls le système ou le matériel (dans le cas où les capacités sont prises en charge par un support matériel) sont habilités à le faire. Toutefois, sur certains appels système (ou via certaines instructions, dans le cas d'un support matériel), un utilisateur peut toujours :

- obtenir de nouvelles capacités, par exemple, lors de la création d'un fichier, l'utilisateur obtient une capacité pour le fichier créé,
- déplacer les capacités dans la liste de capacités,
- détruire une capacité dans la liste de capacités,
- restreindre les droits d'accès associés à une capacité et construire une version moins privilégiée de cette capacité,

- passer une capacité en paramètre d'un appel de procédure,
- transmettre une capacité à un autre utilisateur.

Ainsi, un utilisateur peut contrôler les mouvements des capacités, et leur partage (et donc celui des entités désignées) avec d'autres utilisateurs.

Il faut noter que, suivant les réalisations, un utilisateur dispose d'une ou de plusieurs listes de capacités. Dans ce dernier cas, il y a généralement une liste maîtresse qui contient des capacités sur des listes secondaires, formant ainsi une hiérarchie de listes. De plus, les capacités, tout comme les objets qu'elles désignent, peuvent être stockées de manière permanente.

Définition formelle selon Dennis et Van Horn

> La définition énoncée par Dennis et Van Horn [Dennis 66] utilise trois notions : le segment, le processus et l'activité. Un segment se définit comme une zone de mémoire physique contiguë, et une activité comme un flot d'exécution quelconque. Le processus représente un contexte d'exécution, une sphère de protection dans laquelle s'exécute une ou plusieurs activités, et il définit l'ensemble des segments accessibles à une activité, l'ensemble des entrées/sorties possibles, et de façon générale, l'ensemble des objets manipulables par une activité s'exécutant dans ce contexte.
>
> Un processus contient une liste de capacités, appelée C–liste. Une capacité désigne un objet quelconque dans le système – cet objet peut être par exemple un segment, ou bien une zone d'entrées/sorties. – et elle spécifie les droits d'accès accordés pour cet objet. Une capacité se compose donc de deux parties : un nom et des droits d'accès.
>
> - Le nom de la capacité est une chaîne binaire unique affectée à l'objet désigné lors de sa création. Ce nom est donc un pointeur qui peut être utilisé par le système pour localiser l'objet. Il faut noter que ce nom peut être complètement indépendant de l'adresse effective à laquelle se situe l'objet en mémoire.
> - Les droits d'accès de la capacité sont particuliers au type d'objet désigné. De plus, une capacité indique aussi si son détenteur est propriétaire de l'objet ou non. La propriété d'un objet confère des droits spécifiques comme celui de détruire l'objet.
>
> Dans cette définition, un processus contient une seule et unique C–liste, et les C–listes sont partageables entre plusieurs processus. Le modèle d'exécution permet de construire une hiérarchie de

processus, ainsi qu'un ensemble de sphères de protection et de sous−sphères, c'est−à−dire qu'un processus peut à tout moment :

- ajouter des capacités de sa C−liste dans la C−liste d'un processus inférieur,
- activer ou stopper un processus inférieur,
- supprimer des capacités dans la C−liste d'un processus inférieur.

Partant de là, toutes les constructions sont possibles pour autant que l'on possède les capacités copiées. Les sous−sphères de protection permettent notamment d'exécuter une opération non fiable (déverminage par exemple) dans un contexte avec des droits moindres, et de limiter ainsi les effets possibles d'erreurs ou de malversations.

Enfin, il convient de signaler que les C−listes ne fournissent rien de plus qu'un service de désignation basique, c'est−à−dire qu'elles remplissent seulement une fonction d'adressage, et qu'elles ne sauraient constituer un service de noms symboliques. Outre le mécanisme de composition des C−listes, un service de noms plus complet doit donc être mis en place. Ce service assure notamment que les capacités soient individuellement privées ou publiques, même si elles se trouvent dans des C−listes partagées. Un répertoire constitue l'entité de base d'un tel service de noms. Un répertoire est une liste de triplets (nom symbolique, capacité, attribut privé/public), et de capacités sur d'autres répertoires. Ainsi, nous disposons par transitivité, à partir d'un répertoire racine, d'un graphe de répertoires, et les répertoires peuvent être partagés entre plusieurs graphes. Les répertoires, comme les segments, sont a priori permanents dans cette définition.

Capacités et désignation

Lorsque l'on compare les mécanismes de désignation par capacités à d'autres mécanismes d'adressage, l'analogie avec le schéma d'adressage d'une mémoire segmentée est frappante. En effet, comme indiqué Fig. 1.2, dans une mémoire segmentée, une adresse virtuelle désigne une adresse réelle de la façon suivante :

- la première partie de l'adresse indique un numéro de segment (appelé aussi VSN pour Virtual Segment Number). Le VSN donne directement ou indirectement via un registre, un index auquel se trouve, dans une table système, un descripteur de segment. Ce descripteur

contient notamment l'adresse réelle de base du segment, et les droits d'accès associés au segment,

- la deuxième partie de l'adresse indique un déplacement à l'intérieur du segment. Ce déplacement combiné à l'adresse réelle de base trouvée ci–avant, permet de retrouver l'adresse réelle désignée par l'adresse virtuelle de départ

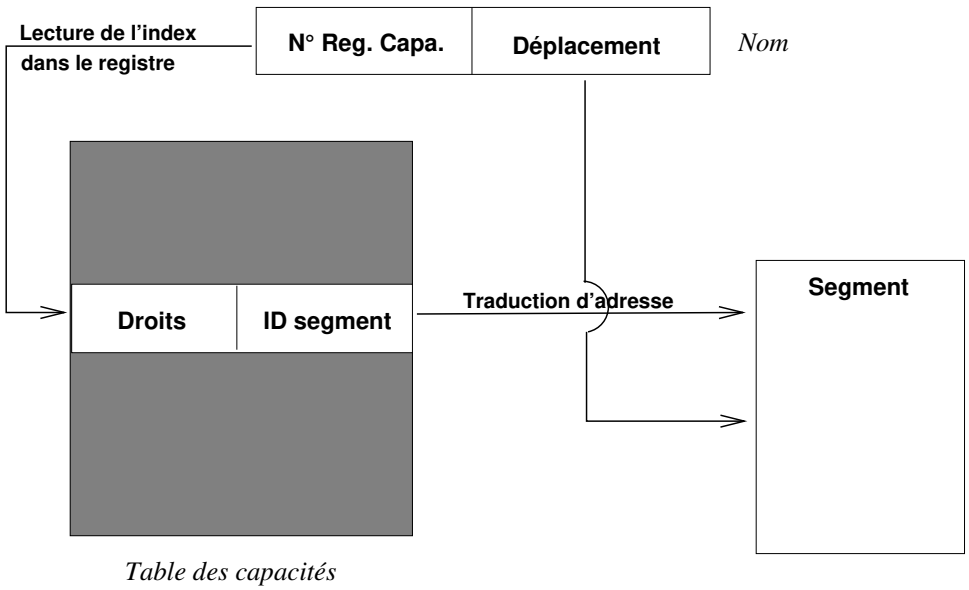


Fig. 1.2 : Désignation et protection par capacités dans le cas d'un support matériel

Ainsi, les mécanismes de désignation par capacités s'intègrent parfaitement à un schéma d'adressage segmenté, et c'est pourquoi les capacités sont très souvent utilisées dans ce contexte d'adressage. Il convient toutefois de distinguer ces deux mécanismes, et pour cela, de noter les propriétés particulières des systèmes à capacités :

- l'uniformité : une capacité permet de désigner n'importe quel type d'entité système
- la protection typée : une capacité contient des informations qui précisent les opérations permises sur une entité du système. La sémantique de ces informations dépend du type d'entité désignée. Par exemple, s'il s'agit d'un objet–mémoire, ces informations de protection indiquent le mode de couplage de cet objet en mémoire principale. S'il s'agit d'un coupleur d'entrées/sorties, elles donneront la liste des instructions valides pour accéder au coupleur.

Capacités et domaine de protection

La notion de sphère de protection introduite par Dennis et Van Horn est plus généralement connue sous le nom de domaine de protection, voire plus simplement de domaine. Un domaine constitue une enceinte de protection qui décrit l'ensemble des objets accessibles aux activités qui s'exécutent dans cette enceinte, et ainsi que les droits en vigueur au sein du domaine pour tout accès à ces objets.

Un domaine définit donc notamment l'ensemble de capacités ou de C–listes disponibles pour une activité à un instant donné, et une activité évolue au cours de son exécution d'un domaine de protection à un autre. Cependant, pour conserver l'étanchéité de ces sphères de protection, il est nécessaire de contrôler le passage d'un domaine à un autre. Ce contrôle se situe à deux niveaux : il s'agit d'abord de canaliser les entrées possibles dans un domaine. Ainsi une activité n'a pas le droit de franchir la sphère de protection d'un domaine par des points d'entrée arbitraires, mais seulement par des procédures définies. Ce sont exclusivement ces points d'entrée qui permettent à une activité de changer de domaine et qui peuvent ainsi servir de guichets de contrôle pour valider ou rejeter l'entrée d'un processus, et de cette façon, la communication réalisée par le processus entre deux domaines de protection. Ainsi, un domaine définit également l'ensemble de ces points d'entrée accessibles aux activités qui s'exécutent dans son enceinte.

Le principe du changement de domaine constitue ainsi une alternative à l'évolution des droits d'une activité, et il permet notamment de réaliser les opérations de restriction, d'amplification, de délégation et d'isolation que nous avons abordées précédemment. Mais le contrôle d'accès exercé au niveau des changements de domaines fait plus particulièrement l'objet du chapitre suivant, et auparavant, nous étudions les techniques de réalisation des capacités elles–mêmes et leurs différentes utilisations.

I.2.2 Techniques de réalisations

Les diverses réalisations de systèmes d'exécution à base de capacités s'appuient sur le modèle énoncé par Dennis et Van Horn. Elles se distinguent les unes des autres par leur apport spécifique et pratique à ce modèle abstrait. Notamment, les capacités dans tout système doivent être elles–même protégées contre les modifications accidentelles ou malveillantes. Plusieurs méthodes permettent d'y parvenir, et les réalisations de systèmes à capacités peuvent être classées suivant la méthode utilisée. Une description du principe de chacune de ces techniques nous

permet d'établir un bilan des avantages et des inconvénients de chacune, et des critères qui influencent le choix d'une technique dans une mise en œuvre.

I.2.2.1 Capacités confinées (segregated capabilities)

La première des méthodes pour protéger les capacités des modifications d'un utilisateur, consiste à stocker et à conserver les capacités dans des segments spécifiques, distinctement des autres données, de façon à les rendre inaccessibles par l'utilisateur. Un tel segment correspond dans de nombreuses réalisations à la notion de C–liste. Dans cette approche, les capacités ne peuvent absolument pas être manipulées comme les autres données qui peuvent être librement rangées, copiées ou transmises à un tiers.

Un utilisateur ne peut désigner et utiliser une capacité que par l'intermédiaire du nom de celle–ci. Toute manipulation, création, copie ou transfert se fait à travers une interface protégée, c'est–à–dire, soit à travers des appels système, soit à travers un jeu d'instructions privilégiées. [Levy 84] décrit plusieurs exemples de systèmes à capacités confinées, notamment :

- le système PDP–1 développé au MIT, ou la Chicago Magic Number Machine [Fabry 74], développée à l'Université de Chicago. Ces systèmes nécessitent un support matériel spécifique. Les capacités sont alors rangées et conservées dans une zone mémoire (C–liste) inaccessible à l'utilisateur. Les capacités sont désignées via un registre de capacités, et les noms des capacités sont des index permettant leur localisation dans la C–liste. Ces capacités ne sont accessibles et modifiables que par des instructions privilégiées.
- le système CAL–TSS [Lampson 76] développé à l'Université de Californie à Berkeley, et le système Hydra [Wulf 81] développé à l'Université Carnegie–Mellon. Ces systèmes à capacités sont entièrement réalisés par logiciel, et ne nécessitent aucun support matériel particulier. Pour les illustrer, regardons de plus près les capacités confinées du système Hydra.

Exemple du système Hydra

Hydra [Wulf 81] est un système à objets actifs, et les capacités sont conservées dans une table propre à chaque environnement d'exécution appelée LNS (Local Name Space). Cet espace local de noms est une C–liste, et constitue un contexte d'exécution pour un processus, c'est–à–dire son espace d'adressage alloué dynamiquement à l'entrée d'une procédure sur un objet. Lorsqu'un processus s'exécute dans une procédure donnée, les capacités sont désignées par leur index dans la C–liste LNS. Comme il n'y a

pas de support matériel pour l'adressage de la mémoire par capacités, toutes les entités (dans Hydra, ce sont des objets−mémoire) désignées par des capacités, et les capacités elles−même, doivent être confinées dans le noyau Hydra. L'ensemble des opérations s'effectuent alors par l'intermédiaire d'appels système. Un appel système comporte donc des capacités entre autres paramètres pour désigner l'objet concerné par l'appel. Plus exactement, chaque paramètre nécessitant une capacité revêt la forme d'un chemin d'accès (un index dans la C−liste de l'espace local LNS, ou une liste d'index à travers plusieurs C−listes) à une capacité. Ce chemin d'accès permet au noyau de retrouver la capacité correspondante (Fig. 1.3), et d'y lire :

1. les informations de localisation de l'objet qu'elle désigne, c'est−à−dire l'adresse de l'objet s'il est actif et déjà couplé en mémoire principale, ou bien son nom global s'il ne l'est pas. Les indicateurs contenus dans la capacité permettent entre autres choses de savoir si un objet est actif ou pas.
2. les informations de protection de l'objet. Ces informations se composent d'une part des droits auxiliaires, − ils sont spécifiques au type de l'objet, − et des droits génériques d'autre part. Ces derniers contrôlent non seulement les droits de lecture et de modification d'un objet, mais également le droit de détruire la capacité (et donc l'objet lorsque celui−ci n'est plus référencé par aucune capacité), ainsi que les droits de modifier la C−liste (espace local) de l'objet, et de copier la capacité dans une autre C−liste, c'est−à−dire le droit de partager un objet.

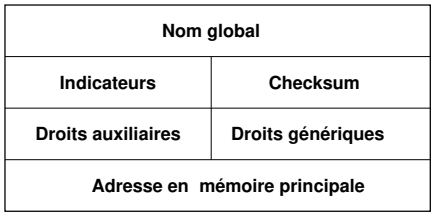


Fig. 1.3 : Représentation des capacités dans Hydra

En résumé, lorsque l'on utilise le terme «capacité» dans un système à capacités confinées, il peut être interprété de deux façons différentes suivant le contexte :

- dans un environnement utilisateur, une «capacité» se réduit à un nom de capacité, car c'est la seule partie visible de la capacité, et il n'existe aucun autre moyen de désigner un objet ou une capacité que par ce nom. De plus, toutes modifications d'une capacité ou d'une C−liste s'opèrent à travers une interface d'appels système (ou un jeu d'instructions) incontournable et parfaitement identifiée. Tout paramètre de cette interface (comme de toute

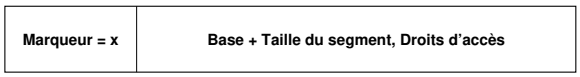
opération) nécessitant une capacité est donc décrit par le nom de cette capacité.

- dans l'environnement système confiné qui gère les capacités, une «capacité» est une structure de données à part entière, décrivant tous les éléments de désignation et les éléments de protection.

I.2.2.2 Capacités publiques (non–segregated capabilities)

Les capacités confinées, comme nous venons de le voir, offrent une réalisation simple à mettre en œuvre, mais rendent difficile toute manipulation de capacité dans la mesure où celles–ci ne peuvent pas être conservées, copiées ou transmises à un tiers comme toute autre donnée.

Pour pallier cet écueil, d'autres types de réalisation de systèmes à capacités s'efforcent d'éviter le confinement et la ségrégation des capacités par rapport aux autres données, et par opposition aux capacités confinées, ces réalisations se caractérisent par des capacités publiques.



Structure protégée d'une capacité

Structure d'une donnée «libre»

Fig. 1.4 : Identification des capacités à marqueur

Capacités à marqueur (tagged capabilities)

La réalisation de capacités à marqueur nécessite un support matériel. Un marqueur est une information (Cf Fig. 1.4) adjointe à chaque emplacement mémoire, et composée d'un ou de plusieurs bits ; cette information indique si cet emplacement de la mémoire contient une capacité ou non. Le positionnement et même la lecture de ce marqueur ne sont possibles qu'à travers un ensemble d'instructions privilégiées. De plus, si le marqueur indique la présence d'une capacité, les mécanismes matériels d'adressage garantissent que cette capacité ne sera manipulée que dans un mode d'exécution privilégiée, c'est–à–dire par des appels système. Ainsi ces capacités ne sont pas tout à fait «publiques» et, un utilisateur ne peut ni modifier les capacités, ni même en créer des factices. Par contre, les capacités à marqueur

pourront être conservées parmi les autres données, sans être confinées dans des emplacements spécifiques.

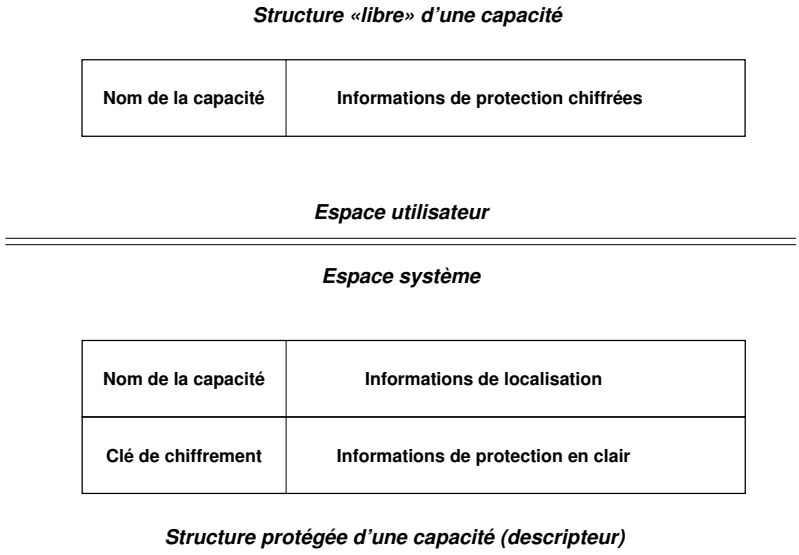


Fig. 1.5 : Vérification des capacités à mot de passe

Capacités à mot de passe (password capabilities)

Pour éviter le recours à un support matériel spécifique, d'autres systèmes utilisent des techniques de chiffrement des capacités pour prévenir les modifications accidentelles ou malveillantes des capacités. Avec ce type de réalisation, les capacités (intégralement chiffrées ou seulement partiellement et alors accompagnées d'un certificat) peuvent être manipulées sans aucune contrainte par l'utilisateur.

Les techniques de chiffrement permettent notamment :

- de vérifier qu'une capacité est valide, et n'a pas été forgée par l'utilisateur. En effet, seul le système connaît chaque clé (ou mot de passe) qui a servi au chiffrement d'une capacité et peut le réutiliser pour certifier conforme une capacité présentée par l'utilisateur.
- d'effectuer une révocation simple et immédiate quel que soit le nombre de capacités déjà dispersées, en renouvelant le mot de passe associé à une capacité.
- de permettre à l'utilisateur d'effectuer lui–même certaines modifications sur les capacités, telles que la copie et la restriction de droits, évitant ainsi le surcoût d'un appel système.

I.3 Premières utilisations dans un contexte réparti

Après de nombreuses réalisations expérimentales, les systèmes à capacités se sont développés pour gérer la protection de mémoires virtuelles réparties uniformes sur des architectures matérielles conventionnelles : un tel système a été développé dans ce sens à l'Université de Monash [Anderson 86] sur une plate−forme multiprocesseurs, et le premier système à capacités vraiment diffusé est le système réparti Amœba ([Tanenbaum 86], [Tanenbaum 91]). Notons que ces réalisations se basent toutes sur des capacités publiques à mot de passe.

I.3.1 Le système Amœba

Le système réparti Amœba permet d'offrir, sur un réseau local et un grand nombre de stations de travail hétérogènes, une plate−forme pour la programmation répartie et la programmation parallèle. Parmi ses objectifs généraux, il convient de signaler l'uniformité de la désignation et le masquage de la répartition, et la sûreté de fonctionnement de l'environnement de programmation répartie. De plus, en tant qu'environnement de programmation parallèle, Amœba doit impérativement offrir des performances respectables pour le parallélisme.

Par ailleurs, les systèmes répartis ont jusqu'alors toujours manqué d'un mécanisme qui permette d'uniformiser la désignation des objets−mémoire et leur protection. Ce constat de lacune a conduit les concepteurs d'Amœba à s'intéresser à l'utilisation de capacités, puisqu'un tel mécanisme incarne à la fois :

- un moyen de désignation : pour toute opération, l'utilisateur précise une (ou plusieurs) capacité(s) qui dénotent le(s) objet(s) au(x)quel(s) il souhaite accéder.
- un moyen de protection : une capacité contient des informations sur la protection qu'il convient d'appliquer à l'objet désigné. L'utilisateur n'a bien sûr pas accès à ces informations ou, du moins, il ne peut ni les modifier, ni les falsifier, ni les contrefaire dans le but d'acquérir des droits supplémentaires sur un objet.

Le choix du type de capacités utilisées est orienté par deux nécessités et motivations dans le contexte d'un système réparti hétérogène comme Amœba :

1. d'une part, Amœba doit s'affranchir des contraintes des noyaux monolithiques pour pouvoir offrir un support réparti et hétérogène de façon plus souple et portable. Dans ce contexte, il est donc exclu d'utiliser un type de capacités qui nécessiterait un support matériel spécifique.
2. d'autre part, un noyau n'est a priori pas fiable dans un système réparti hétérogène, car selon Tanenbaum, un utilisateur peut modifier un noyau sur sa propre machine et ensuite la connecter au système réparti pour

court–circuiter les règles de protection. Cette affirmation, quoique contestable, vient renforcer la précédente pour étayer le choix de gérer les capacités directement et entièrement au niveau utilisateur.

Fig. 1.6 : Une capacité dans Amœba

Ainsi le noyau n'a aucune connaissance des capacités, hormis quelques parties spécifiques que nous verrons plus loin. L'exemple Fig. 1.7 permet d'illustrer et de mieux comprendre l'utilisation des capacités telle qu'elle est faite dans Amœba. Les objets dans Amœba étant gérés par un serveur, les capacités ont donc le format Fig. 1.6, composé du nom (un port) du serveur gestionnaire de l'objet désigné par la capacité, d'un nom désignant l'objet, et interprétable seulement par le serveur gestionnaire, des droits associés à l'objet et d'une clé de contrôle pour vérifier l'authenticité et l'intégrité de la capacité. Cette approche permet de s'affranchir du support du noyau pour gérer les capacités. Toutefois, pour assurer la fiabilité de ce schéma de protection dans un contexte réparti, il convient de garantir l'authenticité et l'intégrité des capacités non seulement lorsqu'elles sont transmises d'un client vers un serveur, mais également lorsqu'elles sont transmises d'une machine à une autre. Un autre niveau de chiffrement doit alors intervenir, et trois solutions sont proposées dans Amœba. Elles sont basées sur :

1. l'intercalage matériel de modules de chiffrement entre chacun des processeurs et leur réseau de communication,
2. le chiffrement logiciel systématique sur tous les liens de communication,
3. l'utilisation d'algorithmes de chiffrement à clé publique pour les communications de capacités.

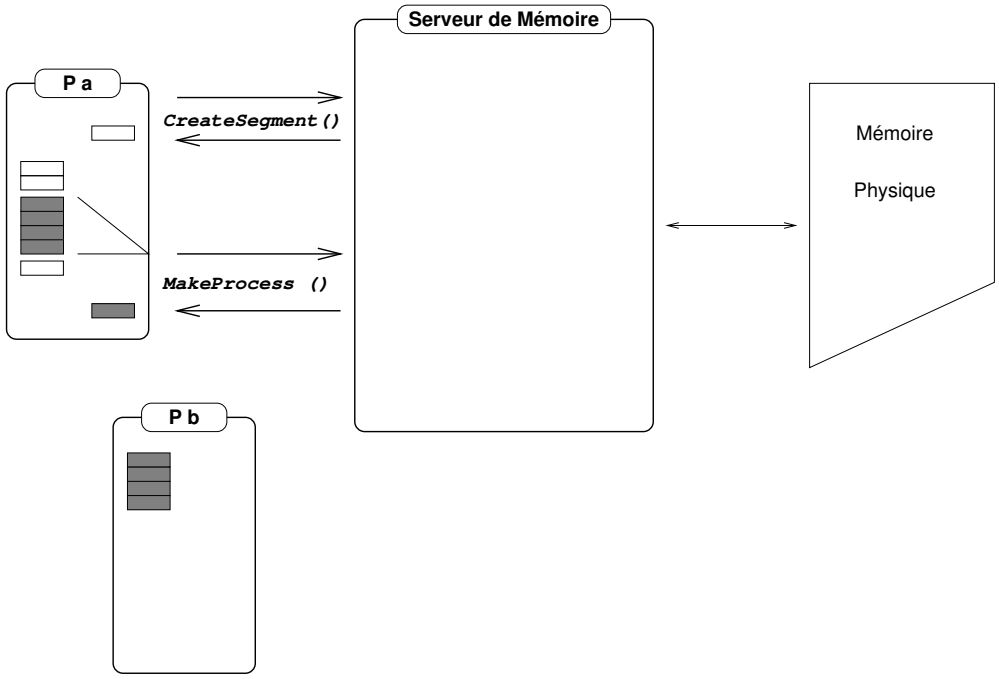


Fig. 1.7 : Réalisation d'un serveur de mémoire dans Amœba

Lorsqu'un processus P_a souhaite créer un segment de mémoire, il ne dispose pas de capacité lui permettant de gérer directement la mémoire physique. Il doit donc passer par l'intermédiaire d'un serveur de mémoire qui, lui, en a la possibilité : ceci est réalisé par l'appel de la procédure `CreateSegment()`, dont P_a reçoit en retour une capacité sur le segment créé par le serveur de mémoire. Avec cette capacité, le processus P_a peut alors écrire, lire et modifier le segment à sa guise, en utilisant les procédures `Read` et `Write` du serveur de mémoire. P_a peut également partager ce segment avec d'autres processus en leur donnant une capacité, et contrôler ce partage, par exemple en créant un processus P_b qui n'aura qu'un accès en lecture au segment créé, et à aucun autre segment. Pour cela, P_a doit typiquement créer un segment de pile, un segment de code pour le processus P_b et initialiser ce processus par un appel `MakeProcess`() au serveur de mémoire, en précisant les capacités sur les segments de code, de pile et l'ensemble des capacités auxquelles le processus P_b aura accès. Au retour de l'appel à `MakeProcess`(), P_a reçoit une capacité qui lui permet de manipuler et de contrôler le processus P_b.

D'autres services répartis sont implantés suivant ce modèle. Amœba fait donc la preuve de l'intérêt d'un système de protection à base de capacités dans un contexte réparti, et de sa viabilité. En s'affranchissant non seulement de la nécessité d'un support matériel, mais aussi de la nécessité d'un support noyau, Amœba permet de

découpler la gestion de la protection de la gestion de la mémoire ainsi que de la répartition.

I.3.2 À l'Université de Monash

Un projet de machine multiprocesseur a été développé à l'Université de Monash [Anderson 86]. Le schéma de protection offert par ce système est très proche de celui d'Amœba.

Fig. 1.8 : Une capacité dans la machine de l'Université de Monash

Dans cette approche, les capacités comprennent un nom d'objet, ainsi qu'un mot de passe qui sert de signature. Ce mot de passe n'est aucunement lié aux informations de protection. Seul le système peut retrouver ces informations, après avoir vérifié que le mot de passe était valide pour l'objet associé.

Dans ce système de capacités à mots de passe, un mécanisme de chiffrement des mots de passe permet de contrôler le partage des capacités, et de réaliser des processus mutuellement méfiants : chaque processus dispose d'un code−clé qui lui est propre, mais qui est inaccessible à ce processus. Lorsqu'un processus crée un objet, le mot de passe de la capacité retournée est chiffré (en fait, c'est un opérateur ou−exclusif qui lui est appliqué) à l'aide de ce code−clé. Lorsque ce processus essaie d'utiliser cette capacité, le système de gestion de la mémoire virtuelle décode (par un ou−exclusif) le mot de passe de la capacité avec le code−clé du processus, avant d'en vérifier la validité. Ainsi, lorsqu'un processus P_a souhaite exécuter un programme non fiable, il crée auparavant un nouveau processus P_b auquel il attribue une valeur arbitraire V qui sera utilisée pour dériver le code−clé de P_b à partir de celui de P_a et de V. Toutes les capacités que P_a souhaite passer à P_b sont codées avec la valeur V. Ainsi P_b pourra utiliser ces capacités, mais aucune autre de P_a. De plus, P_a devra explicitement décoder les capacités que P_b pourra lui retourner, avant de pouvoir les utiliser.

I.3.3 Conclusion

Les concepteurs de système admettent que les capacités fournissent un mécanisme de contrôle d'accès plus souple que le modèle des domaines de protection hiérarchiques. De plus, elles fournissent un moyen de désignation uniforme et non ambigu, car il ne dépend pas du contexte dans lequel un objet mémoire est visible. Cette orthogonalité des mécanismes de protection et des mécanismes d'adressage

est particulièrement intéressante dans le contexte d'une mémoire virtuelle répartie où un adressage uniforme, et indépendant de la répartition, est souvent l'objectif recherché. De même, les applications coopératives, dont les règles de protection évoluent constamment et fréquemment, bénéficient avec des capacités d'un support plus souple et plus adapté à ce dynamisme.

Enfin, des systèmes comme Amœba ont fait la preuve en «grandeur réelle» de la viabilité des systèmes répartis à capacités, et ils ont aussi aboli les contraintes matérielles des premiers systèmes centralisés à base de capacités.

I.4 Les grandes mémoires virtuelles réparties

Un certain nombre de projets se sont récemment intéressés à développer des systèmes répartis à espace d'adressage uniforme, appelés aussi SASOS[(1)] dans la littérature. Dans un système conventionnel comme Unix, les processus disposent d'espaces d'adressage distincts qui leur fournissent une protection intrinsèque. Dans un SASOS, les processus partagent tous le même espace d'adressage unique. Il y est donc nécessaire de disposer d'un mécanisme de protection indépendant de celui de la traduction des adresses virtuelles. C'est là l'une des principales motivations pour le choix de mécanismes à base de capacités que nous allons décrire plus en détail pour chacun des projets de SASOS que nous avons étudiés.

I.4.1 Projet Opal

Le projet Opal, développé à l'Université de Washington, à Seattle, propose également une grande mémoire virtuelle répartie destinée à faciliter le partage des informations entre utilisateurs, y compris les informations de type pointeur. Ceci motive le choix d'un système réparti à espace d'adressage unique (SASOS). Le modèle de protection réalisé pour cette mémoire répartie est décrit dans [Chase 94], et il est également basé sur des mécanismes de capacités à mot de passe. Les raisons du choix d'un système à capacités résident dans les restrictions et la rigidité qu'impose un modèle de protection avec des domaines de protection totalement isolés les uns des autres (ils sont souvent désignés au travers du terme de processus). Opal souhaite notamment offrir les possibilités :

- de réaliser des schémas de protection basés sur une confiance unilatérale et asymétrique, tels qu'un partage de données inconditionnel en lecture, mais protégé en écriture. Ou plus généralement de favoriser un partage de données direct, ne serait–ce qu'un accès en lecture seule, y compris entre des flots d'exécution mutuellement méfiants.

---

(1) SASOS est l'acronyme de Single Address Space Operating System

- de réaliser des schémas de protection de confiance mutuelle favorisant la coopération entre applications qui peuvent avantageusement s'exécuter dans un même domaine de protection pour réaliser des accès semblables aux données partagées. Les domaines de protection factorisent alors les types et les droits d'accès réalisés.

La protection dans Opal implique les entités suivantes du système :

- les segments : ce sont les entités d'allocation et de protection de la mémoire. Les segments Opal sont potentiellement persistants, mais ce n'est pas un comportement systématique. Notons que les segments Opal sont conçus pour effectuer une gestion de la mémoire allouée et sa protection à un gros grain, la protection à grain fin étant laissée à la charge des environnements de programmation et des langages.
- les flots (ou *threads*) : ce sont les entités d'exécution.
- les capacités : ce sont des capacités publiques à mot de passe, telles que celles utilisées par Amœba ou par Mungi. Elles sont codées sur des mots de 32 octets (256 bits). Il est intéressant de noter que les capacités peuvent être diffusées et partagées via un service de noms symboliques, qui les fournit au requérant suivant des listes de contrôle d'accès associées à la capacité demandée. Ainsi, le système de protection utilise conjointement les deux modèles de protection, capacités et listes d'accès, suivant leur adéquation et leur souplesse d'utilisation pour chacun des services de protection fournis.
- les domaines de protection : ce sont des contextes d'exécution pour les flots. Un flot peut coupler un segment dans son domaine de protection courant, par un appel explicite à la primitive de couplage *attach* (ou, de même, le découpler avec un appel à la primitive de découplage *detach*) en précisant une capacité suffisante. Le couplage s'effectue soit selon la méthode traditionnelle des systèmes à capacités, c'est–à–dire avec les droits spécifiés par la capacité, soit selon la méthode traditionnelle des systèmes Unix ou POSIX, c'est–à–dire avec les droits spécifiés par l'appelant de la primitive de couplage *attach* (équivalent alors au *mmap* d'Unix), à condition bien sûr que ces droits soient compatibles et autorisés par la capacité.

  Les domaines comme les segments sont désignés par des capacités qui permettent leur manipulation par l'utilisateur.

Dans cette réalisation, toute la gestion des capacités (création, modification et destruction de capacités ou de domaines, validation des mots de passe,...) est donc réalisée par un service privilégié, auquel l'utilisateur fait appel à travers les appels *attach* et *detach*.

Notons qu'un utilisateur peut également se passer des appels explicites aux primitives de couplage pour acquérir des segments partagés, et bénéficier plutôt d'une résolution et d'une protection implicite sur faute d'adressage. Mais cette ouverture ne semble pas prévaloir dans la philosophie d'Opal.

I.4.2 Projet Angel

Le projet Angel, développé à Londres à la City University et à l'Imperial College, réalise un système d'exécution expérimental qui fournit une mémoire virtuelle répartie dans un environnement multiprocesseur. Des expériences précédentes ont montré les faibles performances d'une réalisation de mémoire répartie au dessus d'un mécanisme d'échange de messages, aussi Angel s'intéresse à une réalisation par mémoire partagée qui offre des performances plus efficaces pour les appels de procédure à distance (RPC).

La mémoire virtuelle répartie d'Angel offre donc un espace d'adressage unique et cohérent couvrant l'ensemble des éléments qui composent une plate−forme multiprocesseur à mémoire distribuée. Les entités impliquées dans la protection de cette mémoire virtuelle répartie sont décrites dans [Murray 93a] et [Murray 93b]. Ce sont les objets, les processus et les capacités :

- les objets d'Angel correspondent à des segments de mémoire, c'est−à−dire à des zones de mémoire de taille fixe et composées d'adresses virtuelles contiguës. Ces segments sont persistants. Dans ce modèle, un domaine de protection est représenté par un segment. Il est géré par un service appelé le Gestionnaire d'Objets, et modifiable exclusivement par ce service. Le segment incarnant un domaine contient un ensemble de capacités qui définissent les segments accessibles, et les privilèges de manipulation associés dans ce domaine.
- les processus sont les unités de flot de contrôle et d'exécution. Le contexte d'exécution d'un processus est défini par le domaine courant du processus et se compose de l'ensemble des segments auxquels le processus peut accéder, notamment de ses segments de contrôle (un ou plusieurs segments contenant du code exécutable, un segment de pile, privée ou partagée) et de ses segments de données privées ou partagées avec d'autres processus. Un processus peut s'exécuter librement dans le domaine de son choix pourvu qu'il en ait le droit suffisant.

  Il convient de noter ici qu'Angel fournit un mécanisme d'interruption logicielle qui offre aux processus un mode de communication asynchrone indispensable en plus de la simple mémoire partagée (qui ne permet que des communications synchrones par attente active).

- les capacités donnent à un processus le droit d'accès à un segment, et ce sont des capacités logicielles publiques :
  - les capacités vues par un utilisateur sont désignées par le terme de *biscuits*. Elles se composent alors seulement d'une adresse virtuelle (le nom de la capacité), et de l'adresse du service qui les a engendrées, et peuvent donc être rangées dans n'importe quel segment. Toutes les informations de protection relatives à un biscuit (essentiellement droits d'accès et adresse du segment désigné) sont donc conservées dans l'espace du service qui a créé ce biscuit. Cet espace est protégé et seul ce service a l'autorité de déplacer ou de copier les capacités qu'il a engendrées.
  - les capacités vues par le service qui les gère sont représentés par des descripteurs. Un descripteur se compose de plusieurs attributs qui définissent les privilèges de manipulation associés. Dans Angel, une capacité comporte les attributs suivants : droit de lecture du segment, droit d'écriture d'un segment, droit d'exécuter un segment, droit de modifier les privilèges associés à la capacité, droit de consulter ces privilèges, droit de copier une capacité, droit de détruire une capacité, indicateur de segment en cours d'exécution par le processus, droit d'effectuer une interruption logicielle vers un domaine.

Toutefois, dans Angel, un *biscuit* ne correspond pas vraiment à une capacité traditionnelle dans la mesure où la possession d'un biscuit valide ne donne pas obligatoirement le droit de coupler le segment désigné et d'y accéder. En effet, ce droit est soumis à la réalisation de circonstances supplémentaires : à chaque *biscuit* le Gestionnaire d'Objets fait correspondre un descripteur de contrôle d'accès. Ce descripteur définit pour chaque type d'accès, l'ensemble des segments qui doivent préalablement être accessibles au processus. Dans l'exemple de la figure Fig. 1.9, le segment C ne peut être accédé en lecture par le processus qui présente le *biscuit* que si le segment A ou le segment B est déjà accessible en lecture dans le domaine courant du processus.

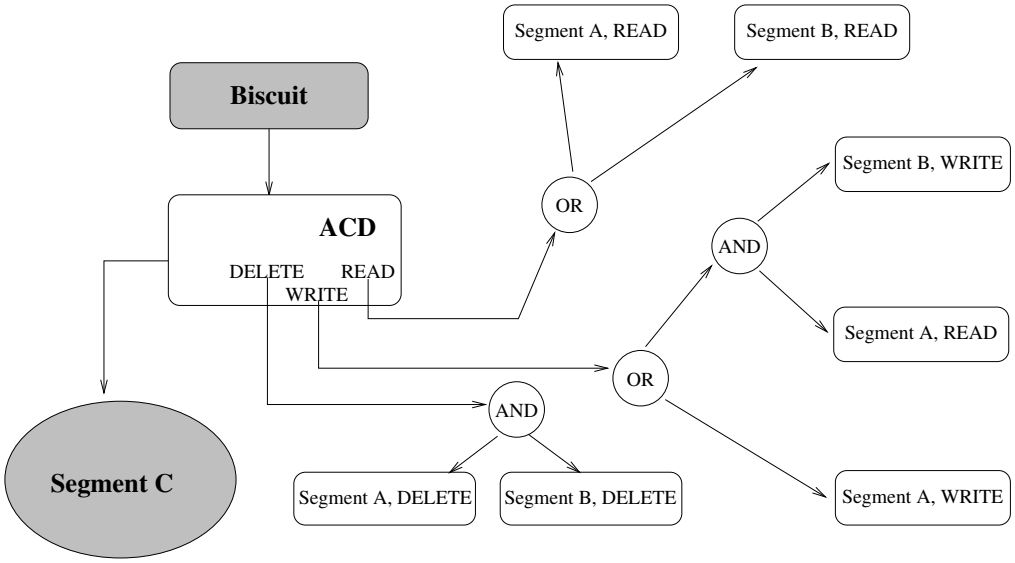


Fig. 1.9 : Schéma de protection dans Angel

Ainsi, un domaine Angel diffère de la définition classique (celle de Dennis et Van Horn) puisqu'il ne dispose pas de sa définition intrinsèque, en terme de C–listes, et de droits d'accès définis statiquement, mais qu'il constitue plutôt un contexte d'exécution où les capacités sont calculées dynamiquement suivant les privilèges déjà acquis, à partir du domaine représenté par le segment d'exécution initial.

I.4.3 Projet Mungi

Le projet Mungi adopte une approche beaucoup plus conventionnelle pour son système de protection. Les objectifs de ce SASOS développé à l'Université des Nouvelles Galles du Sud, sont d'offrir un service de mémoire virtuelle répartie englobant toutes les données disponibles sur un réseau d'échelle moyenne, soulageant l'utilisateur des soucis de localisation et de stockage de ces données et simplifiant le partage de données en le ramenant à un simple passage d'adresses.

Il convient toutefois de noter que si la répartition des données est cachée aux utilisateurs celles des processus ne l'est pas, car ce n'est pas le but recherché dans cet espace d'adressage uniforme.

Les entités impliquées dans la protection de la mémoire virtuelle répartie de Mungi sont décrites en détail dans [Vochteloo 93]. Il s'agit essentiellement :

- des objets : ce sont les entités d'allocation et de protection de la mémoire virtuelle, et ils sont persistants.
- des capacités : ce sont des capacités publiques à mot de passe. Elles fournissent un moyen de désignation indépendant de la localisation, et dans le

cas où un utilisateur présenterait une adresse brute (c'est–à–dire sans capacité), le système doit être capable de retrouver le mot de passe associé pour valider ou refuser l'accès à cette adresse. Dans Mungi, les capacités permettent d'associer à un objet le droit de destruction de l'objet (ce droit peut aussi être considéré comme le privilège du propriétaire de l'objet), le droit de lecture, le droit d'écriture et le droit d'exécution..

De plus, la méthode de chiffrement du mot de passe utilise des techniques analogues à celles d'Amœba¸ où l'utilisateur peut lui–même appliquer des restrictions sur les droits et dériver de nouvelles capacités moins privilégiées, sans modifier le mot de passe.

Enfin, l'utilisation de capacités à mot de passe permet également de résoudre le problème de la réutilisation ou non des adresses dans une grande mémoire virtuelle répartie : en effet, si un objet doit être détruit et si l'adresse virtuelle qu'il occupait doit être ensuite réutilisée, il suffit de changer le mot de passe associé à cette adresse pour invalider instantanément toutes les capacités résiduelles.

- des domaines de protection : ils répondent à la définition conventionnelle de Dennis et Van Horn, et sont constitués d'une arborescence de C–listes partagées.

Comme le représente la figure Fig. 1.10, l'ensemble des capacités dans Mungi est organisé autour d'un arbre partagé par l'ensemble des noyaux composant le système réparti. Chaque nœud de cet arbre (appelé aussi nœud–P ou P–node) pointe sur une C–liste. Il convient de noter ici que cette référence indirecte aux C–listes est due à leur nature publique : en effet, si l'arbre des capacités est une structure de données protégée et entièrement gérée par le système, les C–listes sont contrôlées par les utilisateurs, qui peuvent y ajouter ou retirer des capacités à leur gré. L'arbre est inversé dans le sens où la recherche d'une capacité associée à une adresse donnée s'effectue depuis les feuilles de l'arbre vers la racine. Ainsi les feuilles décrivent des C–listes qui sont scrutées en priorité, c'est–à–dire les plus privées, et la racine les C–listes les plus publiques qui donnent les capacités par défaut. Un domaine est représenté par un nœud–P de l'arbre, et il est défini par le sous–arbre dont ce nœud est la racine. De plus, chaque nœud–P pointe sur un traitant d'exception fourni éventuellement par l'utilisateur (ce pointeur est à nil sur l'exemple de la figure). Ce traitant permet à l'utilisateur d'appliquer une stratégie de recherche des capacités différente de celle adoptée par le système, notamment pour des raisons d'efficacité en fonction des besoins particuliers d'une application.

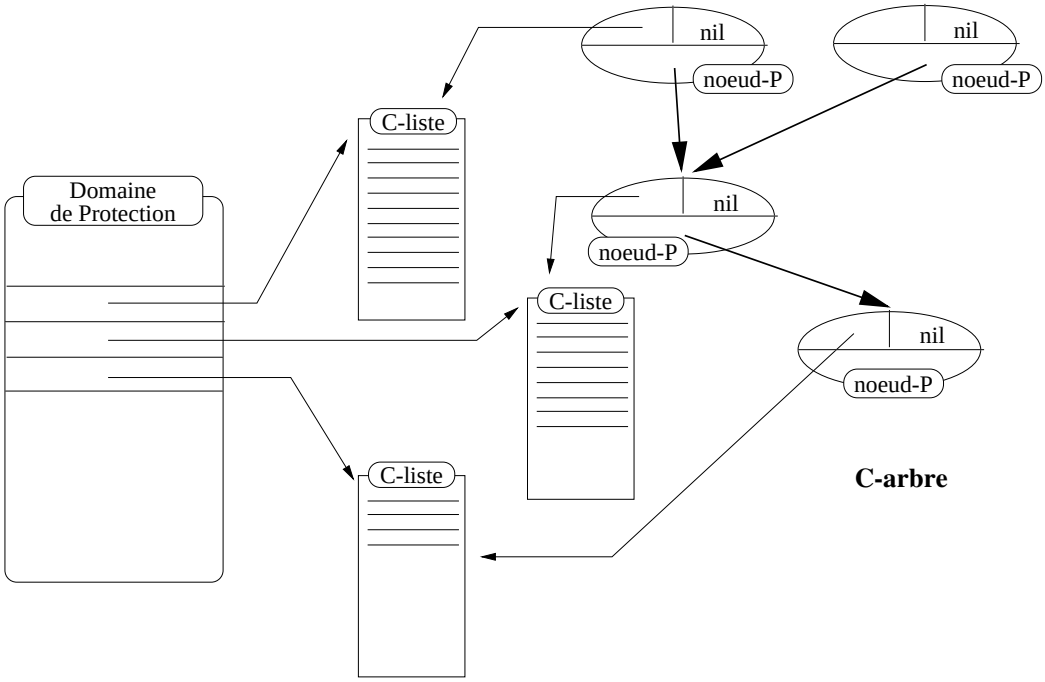


Fig. 1.10 : Arbre de capacités et domaines de protection dans Mungi

Enfin les capacités résidant dans les C–listes en espace utilisateur, il n'existe aucune garantie d'intégrité de ces données, c'est pourquoi elles sont vérifiées et validées par comparaison avec les mots de passe et les droits dont une copie est conservée dans une table du système.

I.4.4 Première synthèse

Après avoir abordé les définitions générales et les différentes classes de réalisation des capacités et des domaines de protection de façon statique, cette première partie étudie une première façon de faire évoluer les droits d'accès au cours de l'exécution d'une activité : il s'agit de faire évoluer les droits définis par le domaine de protection où s'exécute l'activité.

Par rapport à ces aspects, la description des services de protection des grandes mémoires virtuelles réparties d'Opal, d'Angel et de Mungi présentent tout d'abord des motivations communes ou similaires. Ces trois projets se soucient de :

- la généralité du modèle de protection. En effet, il n'y est pas question de restreindre les schémas de protection réalisables au seul schéma hiérarchique des processus que l'on rencontre dans le modèle d'exécution et le système de gestion de fichiers d'Unix par exemple. Suivant cet objectif, Opal, Angel et Mungi optent donc pour un système de protection à base de

capacités pour préserver l'intégrité des données de la mémoire virtuelle répartie.

- la souplesse du modèle de protection. Ce modèle de protection doit également permettre d'établir tous les types de relations de confiance entre utilisateurs : confiance ou méfiance mutuelle, réciproque ou non. Dans chacun des trois projets que nous venons de décrire, les deux entités qui interviennent sont les processus et les domaines : les processus constituent les entités d'exécution active, tandis que les domaines de protection constituent des contextes passifs. Ce sont d'une part les lieux de partage des données entre processus coopérant et, d'autre part, le moyen d'isoler deux processus l'un de l'autre.
- la maniabilité des informations de protection. Pour y parvenir, Mungi et Opal optent pour des capacités publiques, Angel pour des *biscuits* (assimilables à des capacités publiques), qui peuvent être manipulés en espace utilisateur comme n'importe quelle donnée.
- les performances. Cet aspect est particulièrement critique dans une mémoire virtuelle répartie de grande taille et avec un degré de partage élevé, car le nombre de capacités disséminées devient d'autant plus important. Quel que soit ce facteur d'échelle, la recherche d'une capacité dans un domaine doit néanmoins s'effectuer rapidement, pour valider et résoudre une adresse virtuelle.

  Il convient de remarquer ici que le problème de la gestion des capacités et des domaines (c'est–à–dire des C–listes) par l'utilisateur, ou plutôt par un programmeur d'applications utilisant la mémoire virtuelle répartie, ne semble pas faire partie des préoccupations ni d'Angel, ni de Mungi, ni d'Opal. Pourtant, suivant les besoins des applications, la gestion d'un environnement de protection dynamique peut également motiver, comme nous l'avons déjà vu, la préférence d'un modèle de protection à base de capacités, au lieu de listes de contrôle d'accès
- l'uniformité de la désignation vis–à–vis de la répartition. Pour y parvenir, Mungi réalise un arbre réparti global des capacités, Opal et Angel offrent un service réparti traitant la protection. La gestion des capacités se présente donc comme une application répartie à part entière.

Outre ces convergences d'objectifs, certaines extensions proposées sont particulièrement intéressantes. Il s'agit notamment :

- de l'utilisation d'adresses virtuelles pour effectuer un couplage implicite. Dans Opal et Mungi, un utilisateur peut éventuellement ne pas présenter de capacités pour accéder à un objet. La capacité nécessaire pour cet accès est

alors recherchée dans le domaine courant lors du traitement du défaut que cette tentative d'accès direct à un objet non couplé va provoquer. Ce n'est cependant pas là le schéma d'accès «normal» reconnu et défendu par ces systèmes.

- des listes de contrôle d'accès associées aux noms symboliques qui désignent des capacités dans Opal. Cette approche permet de faire cohabiter la mémoire virtuelle répartie protégée par des capacités avec un monde exempt de capacités comme celui d'Unix, évitant ainsi de les disjoindre.
- de la définition dynamique des domaines dans Angel. Cette approche particulière offre une dynamique plus grande à la définition d'un domaine de protection. Il convient de noter cependant que ce modèle ne permet pas a priori de connaître statiquement par transitivité quelle sera la composition d'un domaine d'exécution puisque celle–ci s'étend de façon contextuelle. Il semble donc difficile dans ce modèle de garantir qu'un accès particulier sera vraiment restreint (en lecture seule par exemple), ou interdit.

En conclusion, il apparaît donc un consensus sur le fait que les capacités sont particulièrement adaptées à la protection d'une grande mémoire virtuelle répartie. Les qualités reconnues de ce modèle de protection résident notamment dans sa souplesse ou plus exactement sa généralité. De plus, il s'intègre dans une mémoire virtuelle répartie puisqu'il permet de conserver l'uniformité de la désignation et le masquage de la répartition et de la localisation des données.

Par contre, il convient de noter certains aspects qui ne sont guère ou pas traités tels que la souplesse d'utilisation pour un programmeur d'applications réparties qui souhaitent gérer ses capacités et ses domaines de protection, ou pour un utilisateur qui doit présenter explicitement ou non une capacité pour réaliser un accès à une entité protégée.

Enfin, cette étude de cas nous a montré comment les capacités pouvaient être utilisées pour préserver l'intégrité des données dans un domaine de protection et construire ces domaines et les faire évoluer ; mais il reste à considérer les mécanismes qui permettent à une activité de passer d'un domaine de protection à un autre, et les mécanismes de communication entre domaines qui en dépendent.

Chapitre II

Capacités et protection de services partagés

Au cours du chapitre précédent, nous avons envisagé les différentes méthodes disponibles et utilisées pour réaliser le contrôle d'accès aux données partagées d'une mémoire virtuelle répartie à l'aide du modèle des capacités. Ce contrôle d'accès concernait principalement les opérations de lecture et d'écriture des données, et le partage des données était réalisé de façon passive, par partage des capacités dans les C–listes. Une deuxième méthode de partage existe cependant : il s'agit de l'échange de capacités entre deux domaines de protection. Il convient donc d'étudier aussi le contrôle d'accès réalisé sur ces communications inter–domaines. Aussi nous ne parlons plus ici de protection des données partagées, mais plutôt de protection du code ou encore de services partagés. Ainsi, après avoir cerné l'ensemble des objectifs à atteindre pour ce type de contrôle, nous verrons quelles en sont les différentes réalisations existantes ou potentielles sur la base de capacités. Enfin nous nous intéresserons plus particulièrement aux projets qui offrent une telle protection sur de grandes mémoires virtuelles réparties.

II.1 Les objectifs

Si nous reprenons les notions de segment, de processus et d'activité présentées par Dennis et Van Horn [Dennis 66], le contrôle d'accès à des services partagés peut se réaliser ainsi :

- soit il s'agit tout simplement de contrôler le droit d'exécution d'un segment accordé par une capacité dans un processus. À l'intérieur de ce processus, toutes les activités peuvent ainsi exécuter le code contenu dans ce segment, ou bien aucune ne le peut. Cette protection élémentaire est réalisée avec les mécanismes de contrôle d'accès aux données que nous avons vus précédemment. Elle ne permet pas un contrôle à grain fin sur l'accès au code partagé, mais seulement un contrôle global pour toutes les activités d'un processus. De plus, le contexte d'exécution est le même pour toutes les

activités qui traversent ce code : c'est le processus qui possède une capacité accordant le droit d'exécution.

- soit il s'agit d'accorder à une ou plusieurs activités un droit contrôlé pour l'exécution d'un segment de code, et de distinguer le contexte d'exécution de ce code de celui de l'activité. C'est cet aspect de la protection de l'exécution que nous développerons ici.

En effet, comme nous l'avons vu précédemment, les domaines de protection (ou les processus de Dennis et Van Horn) définissent des contextes d'exécution isolés qui peuvent certes partager des capacités mais de façon statique. Il s'agit donc de permettre à une activité d'évoluer d'un domaine de protection vers un autre, dynamiquement au fil du code qu'elle exécute, et cela doit se faire sans rompre la séparation des domaines. En entrant ainsi dans un autre domaine, l'activité réalise une communication et une éventuelle coopération entre deux domaines de protection (celui qu'elle quitte et celui où elle entre), et très souvent, cette communication s'exprime en termes de service. Comme le montre l'exemple de la figure Fig. 2.1, le domaine appelé Domaine Serveur constitue une enceinte de protection pour le service offert par la fonction F().

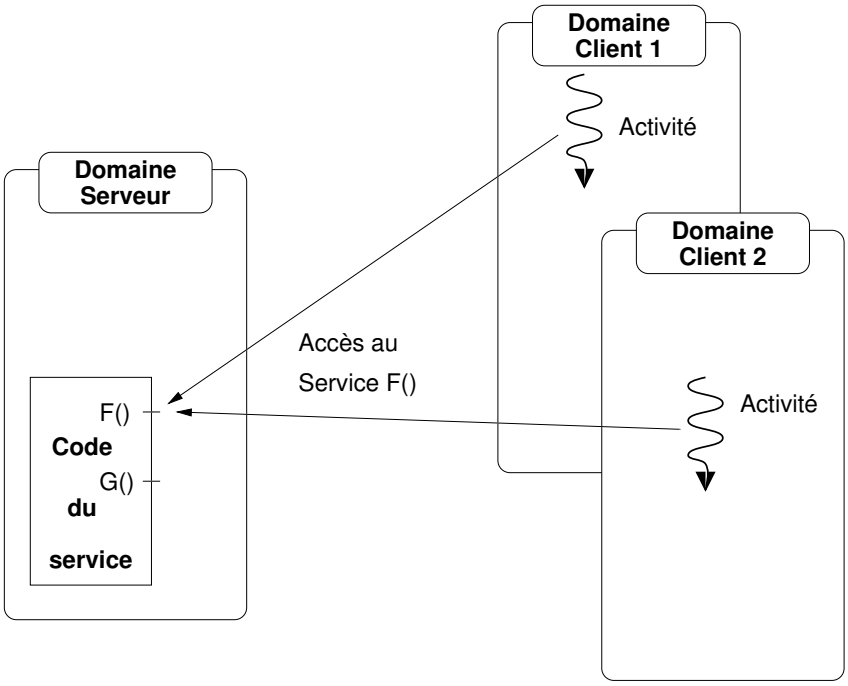


Fig. 2.1 : Enceinte de protection d'un service

Ainsi, une activité qui s'exécute dans un autre domaine de protection, par exemple le domaine Client 1, accède au service offert en entrant dans le domaine de protection de ce service, et le code correspondant à ce service est exécuté par

l'activité mais dans une enceinte de protection fixée et définie par le service. Réaliser ainsi un contrôle sur le domaine d'une activité pour exécuter le code d'un segment particulier est une nécessité motivée par :

- la volonté de contrôler les points d'entrée dans un contexte privilégié ou isolé. Ceci permet de garantir que le code qui y est exécuté, respecte d'une part la sémantique du service offert, et d'autre part le contrôle d'intégrité des données qui y est assuré. C'est typiquement l'objectif recherché par tout service sensible et toute application protégée.
- la volonté de contrôler non seulement les points d'entrée dans un contexte protégé, mais également de garder le contrôle des informations de protection qui sont échangées entre ce contexte et l'extérieur. En effet, une telle situation survient lorsqu'un point d'entrée du domaine nécessite des paramètres et que, parmi ces paramètres, se trouvent des pointeurs. Ces pointeurs peuvent notamment faire référence à des données dans des segments qui ne sont a priori pas accessibles dans le contexte protégé du service, mais qui l'étaient dans le contexte du domaine de provenance de l'activité. Aussi, si l'exécution du code du service nécessite de suivre de tels pointeurs, ces pointeurs doivent alors être accompagnés d'informations de protection nécessaires (et suffisantes) au bon déroulement du service.

  Ce type d'opération complète le précédent et il offre à deux contextes isolés la garantie d'une communication protégée.

La réalisation de ces objectifs dans un système à capacités dépend d'une part des mécanismes de communication mis en œuvre dans le système d'exécution, et d'autre part, de la sémantique du modèle d'exécution. Nous étudierons donc les réalisations possibles d'abord dans le cas d'une communication par messages, ensuite dans le cas d'une communication par appel de procédure à distance, et enfin les motivations et les choix effectués dans les projets de grandes mémoires virtuelles réparties que nous avons déjà abordés précédemment.

II.2 Réalisations de services protégés

Depuis la définition des procédures protégées que Dennis et Van Horn [Dennis 66] ont énoncée, plusieurs réalisations se sont attachées à mettre en œuvre un contrôle d'accès aux services protégés, et à la communication entre des contextes d'exécution isolés. Ces réalisations couvrent l'éventail des techniques de communication inter−processus disponibles, de l'appel de procédure à distance (ou RPC [Birrel 84]), à la communication par message utilisée dans les systèmes Amœba ou Mach.

II.2.1 Concept de procédure protégée

Le terme de «procédure protégée» a été introduit dans la définition de Dennis et Van Horn : une procédure protégée sert aux activités de point d'entrée dans un processus. Ce point d'entrée permet d'assurer un service protégé dans une relation de méfiance mutuelle entre deux contextes d'exécution, en protégeant de tout dommage d'une part les objets locaux nécessaires au service, et les objets de l'appelant d'autre part.

La construction d'un service protégé repose sur la notion d'arborescence de processus présente dans l'ensemble des systèmes d'exploitation et se résume ainsi : un processus supérieur définit le domaine (ou sphère) de protection d'un processus inférieur en lui accordant un sous−ensemble des capacités dont il dispose ; à tout moment, il peut exercer son contrôle sur l'exécution de ce processus inférieur, c'est−à−dire le démarrer, le suspendre, l'arrêter, etc. Partant de ce principe, la mise en place d'un service protégé s'opère selon la décomposition suivante :

a) dans un premier temps, il convient de définir le contexte dans lequel le service devra être assuré, c'est−à−dire l'ensemble des segments nécessaires et suffisants pour le déroulement correct de ce service. Ce contexte constitue le domaine ou processus serveur. C'est l'étape (1) représentée sur la figure Fig. 2.2. où le programmeur du service est représenté par une activité dans le domaine B, et il définit le domaine Serveur en décrivant les capacités qui le consituent. Ces capacités délimitent dans ce domaine B, un sous−ensemble des segments avec des droits d'accès suffisants pour l'exécution du service : notamment le segment de code contenant la procédure `p()` du service qui doit être accessible en exécution dans le domaine Serveur.

b) cette première définition doit ensuite être affinée pour que le serveur puisse contrôler l'accès aux points d'entrée, d'une part, et aux données non partagées d'autre part. Dans l'exemple, le seul point d'entrée valide dans le domaine Serveur est l'adresse du début de la procédure `p()`.

c) puis, ce sont l'interface et le mode d'accès au service qu'il faut détailler. En effet, l'ensemble des activités qui entrent dans une procédure protégée peuvent soit partager le même espace d'adressage, soit s'exécuter dans des espaces d'adressage distincts, évitant ainsi la possibilité de toute interaction entre deux activités servies simultanément. Par exemple, dans l'étape (2) l'exemple de la figure Fig. 2.2, l'activité qui s'exécute dans le domaine A ne peut pas y effectuer localement le traitement de la procédure `p()`, mais elle peut aller dans le domaine Serveur pour exécuter ce traitement. Si une deuxième activité A' effectue la même opération, elle peut s'exécuter soit

dans le même espace d'adressage que A pour traiter `p()`, soit dans un espace d'adressage distinct mais qui représentera le même domaine Serveur.

d) enfin, lors de l'exécution du service dans le contexte protégé, il convient de définir ce qu'il doit advenir des nouveaux objets et des segments créés par le serveur, notamment s'ils peuvent ou doivent être partagés avec les domaines A ou B ou au contraire demeurer privés au domaine Serveur.

e) pour terminer, il est éventuellement possible de livrer une capacité du contexte du serveur à l'activité appelante. À l'étape (3) représentée sur la figure Fig. 2.2, la capacité d'appel à la procédure protégée `p()` est diffusée, notamment au domaine A (directement ou par l'intermédiaire d'un service de noms symboliques)

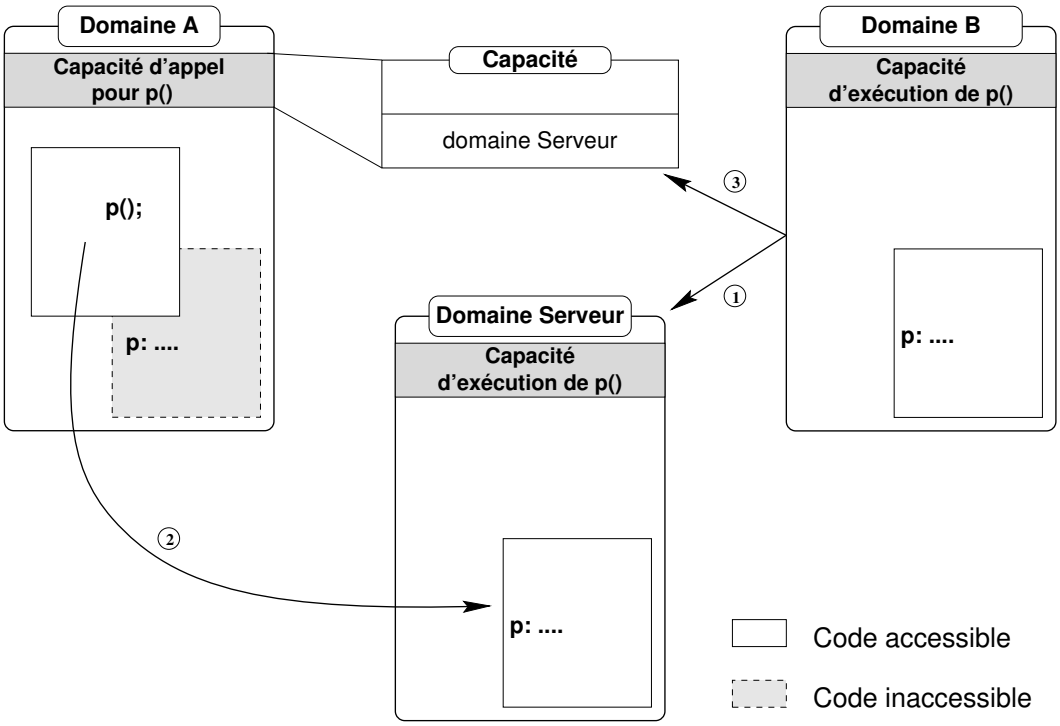


Fig. 2.2 : Principe d'exécution d'une procédure protégée

Enfin, ce mécanisme de procédure protégée est mis en œuvre par l'intermédiaire des capacités d'appel (*«entry capabilities»*). Une capacité d'appel confère le droit d'entrer dans un domaine particulier pour y exécuter une procédure donnée. Dans l'exemple précédent, le domaine A détient une capacité d'appel pour l'exécution de la procédure `p()` dans le domaine Serveur. L'activité qui s'exécute dans le domaine A peut ainsi changer de domaine pour exécuter `p()`, et cette capacité

d'appel constitue donc un point d'entrée pour le domaine Serveur. Dans le modèle de Dennis et Van Horn, une capacité d'appel contient :

- un pointeur sur la C–liste de son créateur, c'est–à–dire sur la sphère de protection où exécuter la procédure protégée.
- une table des points d'entrée dans cette sphère de protection.

Pour réaliser un appel, une activité doit donc réunir et présenter au système d'exécution : une capacité d'appel, l'index désignant la procédure protégée à appeler dans la table des points d'entrée, et une éventuelle capacité à passer en paramètre à la procédure. Cette dernière capacité permet notamment de transmettre au domaine Serveur :

1. les données correspondant aux paramètres effectifs du point d'entrée,
2. si ces paramètres contiennent des pointeurs vers des données accessibles depuis le domaine A, mais a priori inaccessibles depuis le domaine Serveur, les informations de protection nécessaires pour permettre à l'activité d'utiliser ces pointeurs et de faire référence aux données correspondant depuis le domaine Serveur.

Nous pouvons remarquer que le passage de paramètres dans l'appel d'une procédure protégée pose une difficulté similaire à celle rencontrée dans les appels de procédure à distance (ou RPC). Lors d'un RPC, le partage de données entre deux espaces d'adressage distincts ne peut pas être réalisé par la transmission d'un pointeur. En effet, un pointeur constitue une référence qui n'est valide que localement dans un espace d'adressage donné. De même dans l'appel d'une procédure protégée, le partage de données entre deux domaines de protection distincts ne peut pas être réalisé par la transmission d'un pointeur. En effet, la référence d'un pointeur désigne ici une capacité, et cette référence n'est valide que localement dans un domaine donné : dans deux domaines de protection distincts, la capacité référencée par un même pointeur désigne bien les mêmes données, mais elle désigne des droits d'accès qui ne sont valides que localement.

Cette définition d'une procédure protégée présente donc chaque étape de l'appel d'une procédure protégée et en établit les aspects configurables par le prestataire de service. Toutefois, elle laisse la liberté au modèle d'exécution de dérouler l'appel de la procédure protégée dans le contexte de son choix.

II.2.1.1 Cas n° 1 : exécution chez l'appelé

Dans ce premier cas, la procédure protégée s'exécute donc dans le domaine de protection de son créateur. C'est l'option retenue par le modèle de Dennis et Van Horn, et également par des réalisations comme Cal–TSS [Lampson 76].

Ici, le contexte de l'appelant et celui de l'appelé sont effectivement séparés comme sur la figure précédente Fig. 2.2. Pour assouplir la coopération et la communication entre ces contextes disjoints, l'appel d'une procédure protégée permet donc la transmission d'un nombre déterminé de données ou de capacités en paramètre. De façon symétrique, le retour d'une procédure protégée permet également la transmission de données et de capacités en paramètre. Le modèle de Dennis et Van Horn limite cette transmission à une seule et unique capacité, mais l'exemple du système Cal–TSS nous montre qu'il est possible de transmettre une ou plusieurs capacités en paramètre.

Cet échange de paramètres revêt une importance particulière pour le troisième point dans la mise en place décrite plus haut : en effet, si deux activités A_1 et A_2 appellent la même procédure protégée dans un contexte C_P avec respectivement c_1 et c_2 en paramètre, elles peuvent exécuter alors le service :

a) soit dans le même espace d'adressage défini par le contexte $C_P \cup \{c_1, c_2\}$. Dans ce cas, l'activité A_1 peut craindre que l'activité A_2 exploite, dans ce contexte, la capacité c_1 de façon malencontreuse ou malintentionnée. Trois situations sont alors possibles, suivant le degré de méfiance (ou de confiance) accordé par A_1 au contexte C_P :

   1. dans le cas d'une méfiance extrême, le risque que l'activité A_2 exploite la capacité c_1 que l'activité A_1 a passée à C_P, peut être considéré totalement inacceptable.
   2. dans une situation plus modérée, le code exécuté dans le contexte C_P est jugé fiable, l'exploitation de c_1 par un tiers comme A_2 peut être tolérée, à condition que cette exploitation soit effectivement confinée au contexte C_P, c'est–à–dire que l'activité A_2 ne pourra notamment pas recopier la capacité c_1 pour l'exploiter a posteriori, en dehors du contexte C_P et hors du contrôle du contexte d'origine de cette capacité et de l'activité A_1.
   3. enfin dans une situation de confiance totale, la capacité c_1 cédée peut circuler librement, mais cela signifie que la capacité c_1 est considérée comme publique, et que la protection ne motive plus le passage de cette capacité qui devrait alors être déjà connue du contexte C_P. Cette dernière hypothèse est donc à exclure.

b) soit dans des espaces d'adressage distincts, et définis respectivement par $C_P \cup \{c_1\}$ et $C_P \cup \{c_2\}$. Cette solution qui isole les contextes d'exécution résout le problème posé par l'exploitation incontrôlée des capacités passées en paramètre.

Le premier comportement est celui adopté notamment par le système Cal–TSS dans lequel on peut éviter la diffusion incontrôlée de capacités par la technique suivante : l'activité appelante peut chiffrer la capacité passée en paramètre pour la rendre inutilisable sauf pour le domaine cible. Ainsi, si cette capacité venait à être copiée, frauduleusement ou par inadvertance, elle demeurerait néanmoins inutilisable et inexploitable en dehors du domaine de protection du service.

Le deuxième comportement où les contextes d'appels multiples sont disjoints est celui que l'on rencontre notamment dans le système Hydra [Wulf 81] où, un nouvel espace d'adressage est créé à chaque changement de domaine.

II.2.1.2 Cas n° 2 : exécution chez l'appelant

L'autre possibilité pour dérouler l'appel de la procédure protégée réside dans l'utilisation du contexte de l'appelant. Dans ce cas, la procédure protégée dispose des droits d'accès à l'ensemble des objets de l'appelant. De plus, pour résoudre les étapes 1 et 2 de la mise en œuvre décrite plus haut, il convient de donner à l'appelant le droit d'accès aux objets locaux du service de la procédure protégée, par l'intermédiaire d'un segment de liaison.

En reprenant la description des deux activités A_1 et A_2 s'exécutant dans des contextes C_1 et C_2 qui appellent la même procédure protégée dans un contexte C_P, ces appels se dérouleront alors respectivement dans les contextes $C_P \cup C_1$ et $C_P \cup C_2$.

Dans cette solution, l'intersection des espaces d'adressage de deux activités exécutant simultanément un appel à la procédure protégée est strictement restreinte à C_P, et le problème d'interférences malencontreuses ou malveillantes que nous avons vu dans le cas précédent est ainsi supprimé. Par contre, il faut noter que les contextes d'exécution de l'appelant et celui de l'appelé ne sont pas disjoints. Il est donc impossible d'utiliser ce modèle d'exécution pour réaliser un appel entre deux contextes mutuellement méfiants.

II.2.2 Premières utilisations dans un contexte réparti

Dans un contexte réparti, la communication entre les domaines de protection potentiellement distants a du être adaptée. En effet, la communication ne peut plus s'effectuer sur la base d'un simple appel de procédure comme dans les réalisations centralisées que nous venons de voir : elle s'effectue alors sur la base d'un envoi de messages pour mettre en œuvre un appel de procédure à distance. Des exemples récents de réalisations nous sont fournis par les systèmes Amœba [Tanenbaum 86][Tanenbaum 91] et Mach [Acetta 86].

II.2.2.1 Amœba

Au chapitre précédent, nous avons vu comment le système Amœba, précurseur pour l'utilisation des capacités dans un contexte réparti, utilise les capacités pour protéger l'accès aux objets. Avant de reprendre ici la description de ce système pour détailler les mécanismes de contrôle d'accès liés au modèle d'exécution, il convient de rappeler le format d'une capacité dans Amœba (Cf illustration de la figure Fig. 1.6) : une capacité se compose du nom de l'objet qu'elle désigne, des droits qui lui sont associés, d'un champ de vérification d'intégrité, et du nom (aussi appelé port) du serveur qui gère cet objet.

Dans le modèle d'exécution client−serveur d'Amœba, chaque objet n'est accessible qu'à travers les opérations définies par le serveur qui le gère, et ces opérations constituent les seuls points d'entrée dans le domaine de protection du serveur. Une opération est un appel de procédure à distance réalisé par l'envoi d'un message de requête au serveur et l'attente d'un message de réponse. Ce mécanisme, détaillé sur la figure Fig. 2.3, se déroule suivant le schéma usuel d'un appel de procédure à distance, c'est−à−dire :

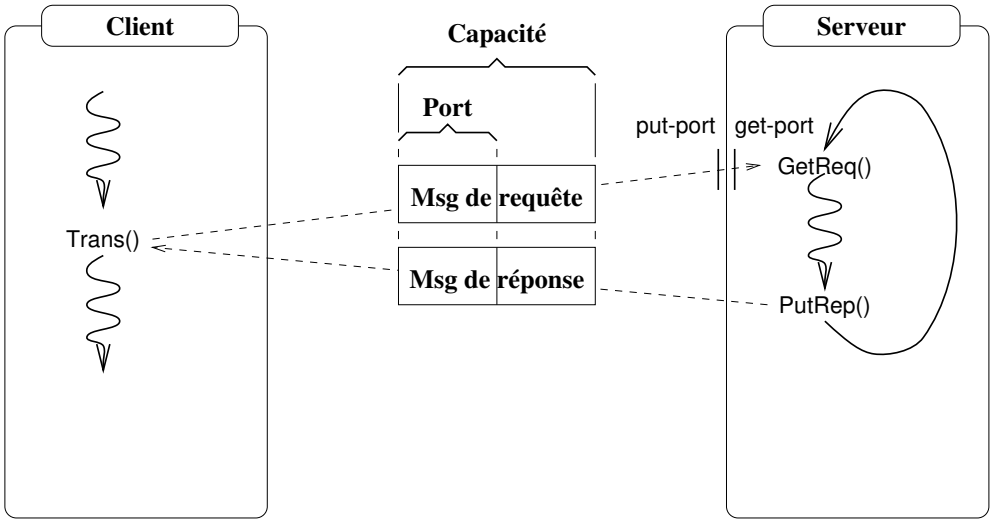


Fig. 2.3 : Appel de procédure protégée dans Amœba

- du côté du serveur : un flot d'exécution (*thread*) reste en attente de l'arrivée d'une requête sur le port externe du serveur (*get−port*), traite une requête, en retourne la réponse, et boucle sur l'attente active de nouvelles requêtes. Un message de requête se compose :
  1. d'une capacité qui désigne l'objet concerné par la requête. Cette capacité désigne notamment le serveur gestionnaire en indiquant le port où adresser les requêtes émises pour cet objet. Le reste de la capacité n'est

compréhensible que par le serveur qui vérifie systématiquement cette partie à la réception d'une requête avant d'en commencer le traitement.

2. d'une commande à effectuer sur l'objet. Cette commande est représentée par un indice dans la table des opérations offertes par le serveur.
3. d'un éventuel complément d'information. Cette partie contient les paramètres éventuels de l'opération requise. Au besoin, ces paramètres doivent être déballés du message (*unmarshalling*) avant de traiter l'opération requise.

- du côté du client : le flot d'exécution client doit d'abord composer le message de requête avec la capacité de l'objet et la commande désirée, et emballer (*marshalling*) les éventuels paramètres. Ensuite, la primitive `trans()` permet l'envoi du message sur le port destinataire (*put−port*) et l'attente d'une réponse à la requête (ou d'une erreur). Le flot client doit éventuellement déballer les paramètres de retour contenus dans le message de réponse.

Un programmeur d'applications peut évidemment écrire de bout en bout le code de la boucle du serveur, et les talons d'emballage/déballage des paramètres, mais le système Amœba offre un niveau d'abstraction procédurale à cette communication par messages permettant ainsi de s'affranchir du souci de programmation des mécanismes de communication sous−jacents. Il s'agit du langage AIL (Amœba Interface Langage) qui permet de décrire un service en termes de «classe», et chaque définition de classe contient la signature de la procédure telle qu'elle est vue par le client.

La description de ce langage sortirait du champ de l'étude qui nous intéresse ici ; en résumé, il convient de noter que le langage AIL est conçu pour masquer les mécanismes de communication et permettre de disposer d'une interface procédurale, plus simple à concevoir et plus maniable, pour réaliser des services protégés dans Amœba.

II.2.2.2 Mach

Le système d'exécution Mach [Acetta 86], développé à l'Université Carnegie−Mellon, n'est pas conçu à l'origine pour offrir à ses utilisateurs le service d'une mémoire virtuelle répartie comme l'est un système comme Amœba. Cependant, le cas particulier de ce système nous intéresse : en effet, le système Mach permet de protéger les communications sur un site d'un système réparti à l'aide de capacités de communication, plus communément désignées par le nom de *ports*. Dans le cadre du projet Mach, une étude [Sansom 86] montre comment étendre ce mécanisme de communications locales protégées pour fournir une abstraction répartie du concept

de port. Aussi, après une rapide présentation du mécanisme des ports dans Mach, nous verrons comment ce concept local peut être réparti, et les problèmes que cela engendre.

Le système Mach repose sur trois notions fondamentales. Il s'agit des tâches, des flots d'exécution et des ports :

- une tâche représente un ensemble de ressources du système comme, par exemple, la mémoire, le(s) processeur(s), etc. Une tâche constitue un espace d'adressage pour les flots d'exécution, et définit ainsi l'ensemble des ressources accessibles pour ces flots.
- un flot d'exécution représente l'entité d'exécution du système. Nous parlerons aussi d'activités. Un flot s'exécute à l'intérieur d'une tâche, et il partage les ressources de cette tâche avec tous les autres flots qui s'y exécutent.
- un port représente l'entité de communication du système et, comme indiqué sur la figure Fig. 2.4, il est réalisé par une file de messages gérée et protégée par le système : un port est seulement accessible et désigné par des capacités accordant des droits d'émission ou de réception sur la file de messages. Très souvent d'ailleurs, la notion de port Mach est confondue avec celle de capacité.

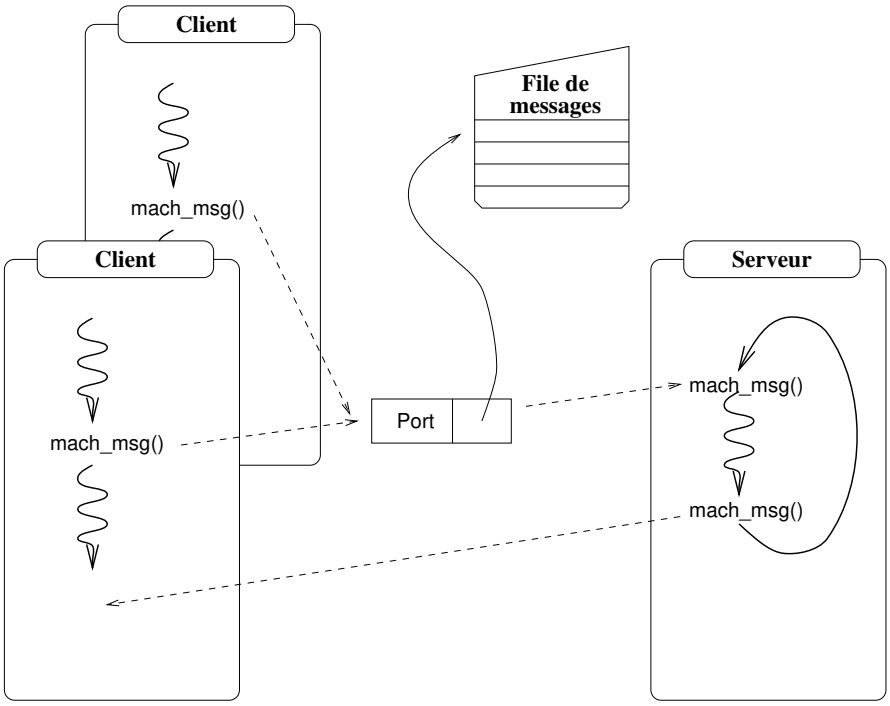


Fig. 2.4 : Communication protégée dans Mach

La réalisation des capacités dans Mach s'apparente donc à la famille des capacités confinées : un port n'est accessible que par des noms de «ports» qui désignent une capacité connue du noyau et toutes les entités du système sont désignées par des ports. Une capacité permet d'accorder plusieurs types de droits sur un port de communication. Il peut s'agir d'un droit :

- de réception, qui permet au détenteur d'une telle capacité d'attendre un message en provenance de ce port. Un port dispose d'un seul et unique récepteur.
- d'émission, qui permet au détenteur d'envoyer un message sur ce port. Il existe un cas particulier, le droit d'émission unique, qui permet au détenteur d'envoyer un seul et unique message sur le port désigné, et ce droit s'autodétruit après l'envoi du message.

Les mécanismes de communication de Mach permettent de réaliser notamment un modèle d'exécution client–serveur, comme dans Amœba. Un client et un serveur sont deux tâches distinctes dont les ports constituent le seul moyen de communiquer ; les communications sont entièrement réalisées par l'appel noyau `mach_msg()`. De plus, la primitive `mach_msg()` permet d'inclure dans un message des droits sur un port et de transférer ainsi des capacités de façon sûre et protégée, d'une tâche à une autre. C'est d'ailleurs l'unique moyen dont disposent deux tâches pour partager des capacités dans Mach. Pour initialiser de telles communications, toute tâche dispose d'un port de communication qui lui est associé, et du droit de réception sur ce port. Elle peut dès lors communiquer des droits d'émission sur ce port à toute autre tâche, par le biais d'un service de désignation symbolique par exemple.

L'exemple de la figure Fig. 2.4 décrit le principe d'un appel de procédure à distance (entre deux tâches locales ou distantes comme nous le verrons plus loin) que nous pouvons résumer ainsi : la tâche du client dispose d'une capacité avec des droits d'émission, sur un port de la tâche serveur (c'est–à–dire un port pour lequel la tâche serveur a le droit de réception). Un flot s'exécutant dans la tâche client peut alors envoyer un message sur ce port en appelant `mach_msg()`.

Pour réaliser un appel de type procédural avec ce moyen de communication, le message envoyé contient un droit d'émission unique sur un port de réponse créé par la tâche client pour cet appel. Après avoir envoyé le message, le flot dans la tâche client se met en attente de réception d'un message sur le port de réponse.

Du côté de la tâche serveur, un flot est en attente de réception de messages de requête de service. En recevant ce message, la tâche reçoit également un droit d'émission sur un port de réponse. Dès lors, le flot peut traiter le message selon la sémantique du service, envoyer si nécessaire une réponse sur le port pour lequel la

tâche a reçu un droit d'émission, et reboucler en attente pour traiter les messages suivants dans la file du port de communication de la tâche serveur.

De même que dans le système Amœba, la réalisation d'une interface client–serveur de type procédural est simplifiée par l'utilisation d'un langage d'interface Mig (*«Mach Interface Generator»*) qui permet de soulager le programmeur, non seulement de l'emballage et du déballage des paramètres, mais également de la transmission des droits accompagnant les données, en spécifiant le format des messages échangés et les ports qu'ils contiennent. Cette transmission de droits est une particularité essentielle qui distingue l'approche adoptée dans le système Mach de celle d'Amœba dans l'utilisation de capacités avec une communication par messages.

II.2.2.3 Sécurité des communications

Nous avons étudié jusqu'ici comment les systèmes à capacités permettaient d'assurer le contrôle de l'intégrité des données partagées dans une mémoire virtuelle répartie, d'une part de façon passive, à l'intérieur d'un domaine de protection, et d'autre part, lors d'un partage actif et dynamique lors des interactions (appels de procédure ou envois de message) entre deux domaines de protection. Il convient néanmoins de garder à l'esprit d'autres aspects de la communication qui interviennent dans la protection des données. Ces aspects resurgissent d'ailleurs particulièrement dans l'étude menée dans Mach [Sansom 86] pour étendre la protection de communications de Mach dans un contexte réparti.

En effet, le principe utilisé dans cette étude pour mettre en œuvre la protection des communications distantes consiste à donner une vision répartie des ports de communication. Comme l'indique la figure Fig. 2.5, ceci peut être réalisé par l'utilisation d'un serveur réseau qui assure la transmission des messages sur le réseau, et gère une table des correspondances entre le représentant local d'un port «réparti», et ses représentants distants sur chacun des sites connus du système réparti. Sur l'exemple de la figure, deux sites interviennent : lorsque le flot dans la tâche client souhaite envoyer un message à la tâche serveur sur le site B, il utilise le représentant local du port de la tâche serveur lors de son appel à `mach_msg()`. La tâche réceptrice de ce représentant local est le serveur réseau du site A qui effectue alors la traduction du port destinataire en son représentant local sur le site B, et qui fait suivre le message au serveur réseau du site B, qui a son tour le fera suivre à la tâche serveur. De la même façon, ce mécanisme permet également de traduire les représentants locaux de ports distants lorsque des droits sur ces ports sont transmis dans un message.

L'analyse de cette réalisation révèle plus particulièrement un aspect crucial dans la sécurité d'un système réparti à capacités : lorsque les capacités sont transmises

sur le réseau et que ces capacités ne sont pas des capacités publiques, mais comme dans Mach, des capacités confinées, il est alors nécessaire de protéger les communications réseau qui les contiennent. Sans cela, les capacités peuvent être détournées de leur utilisation, et compromettre ainsi l'intégrité des données qu'elles sont censées protéger, si l'un des cas suivants venait à se produire :

- un tiers effectue une écoute passive du réseau, et acquiert ainsi une connaissance illicite des données protégées et des ports contenus dans les messages,
- un tiers détourne ou réutilise les messages qu'il a interceptés,
- un tiers peut se faire passer pour un autre et utiliser sa connaissance des ports répartis pour usurper l'identité d'un représentant local,
- un tiers peut faire entrave au service en faisant croire à la mort d'un port réparti.

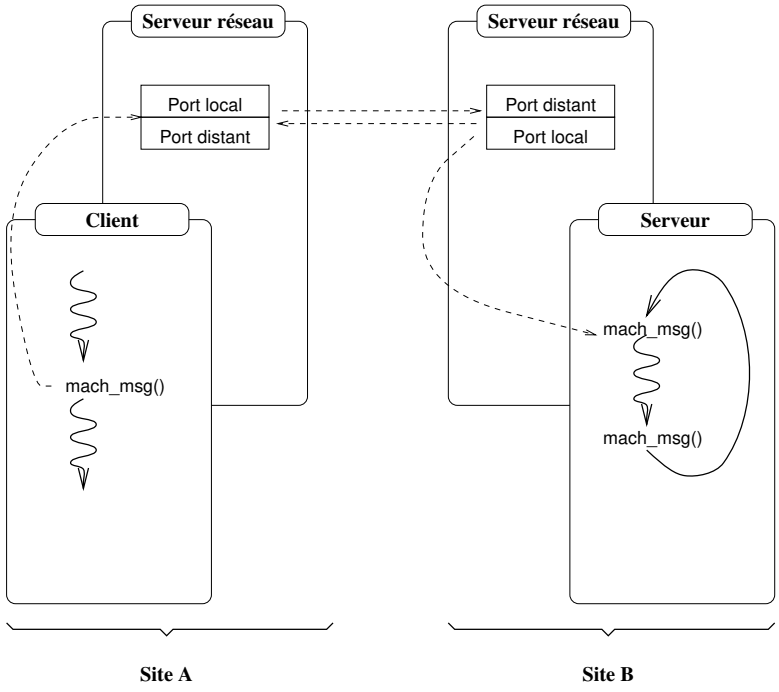


Fig. 2.5 : Appel à distance d'une procédure protégée dans Mach

Plusieurs techniques permettent de réduire ces risques de détournement des ports. Nous pouvons notamment citer :

- le chiffrement systématique de tous les messages échangés sur le réseau qui permet de protéger les messages contre toute écoute passive,

- l'adjonction d'estampilles, de numéros d'ordre, de «checksums» aux messages qui permet de les protéger contre des attaques actives telles que les modifications ou les réutilisations,
- les techniques d'authentification telles que les algorithmes à clé publique qui permettent de certifier l'identité du récipiendaire d'un port, et d'assurer ainsi un transfert fiable des droits sur le réseau.

Cette liste ne prétend pas constituer une étude exhaustive des problèmes de sécurité liés à l'environnement réparti, mais elle montre que dans un tel environnement, les techniques d'authentification qui sont utilisées dans les réalisations de systèmes à capacités publiques deviennent également indispensables pour une réalisation répartie à capacités confinées.

Une approche plus complète et générale des problèmes de sécurité liés au réseau est donnée dans [Bidan 95], et n'entre pas dans le cadre de cette étude où nous intéressons plus particulièrement aux aspects liés au contrôle de l'intégrité des données.

II.3 Les grandes mémoires virtuelles réparties

Après avoir vu quels sont les mécanismes mis en œuvre pour assurer le contrôle d'accès aux services protégés et le contrôle des communications inter–domaines dans les systèmes à capacités, nous nous intéressons maintenant au cas plus particulier des projets de grandes mémoires virtuelles réparties : ce sont les projets Opal, Angel et Mungi que nous avons déjà abordés au premier chapitre.

II.3.1 Projet Opal

Dans le projet Opal [Chase 94], la notion de procédure protégée suit un mécanisme similaire à celui des appels de procédure à distance (RPC ou LRPC) et correspond à la définition donnée par Dennis et Van Horn. Ici, ce mécanisme porte le nom de portail (*portal*) et réalise un point d'entrée et de contrôle d'accès dans un domaine de protection. Un portail Opal spécifie donc :

- le domaine de protection dont il constitue un point d'entrée,
- l'adresse du code que tous les flots qui rentrent dans le domaine de protection doivent exécuter.

Il est nécessaire ici de reprendre et de détailler la nature des capacités publiques utilisées dans Opal. Un portail est désigné par un nom unique dans l'espace de noms réparti et global des portails, et une capacité se compose : d'abord d'un nom de portail pour entrer dans un domaine serveur, ensuite d'une adresse virtuelle en mémoire partagée répartie, et enfin d'une signature de vérification de la capacité. La

spécification d'une adresse virtuelle, constitue un paramètre lors du passage par un portail, à l'attention du domaine serveur. Cette adresse permet de multiplexer le traitement de plusieurs objets ou ressources par le même portail d'entrée dans le domaine de protection du serveur.

Ainsi, lorsqu'un flot qui s'exécute dans un domaine de protection du client, fait référence au traitement d'une des ressources du serveur, la capacité associée à cette ressource permet de déclencher un appel de procédure à distance pour traiter la ressource dans le domaine de protection du serveur.

Ce mécanisme constitue d'ailleurs dans Opal, le seul moyen de lancer une exécution dans un autre domaine de protection donné : un flot qui s'exécute dans un domaine de protection donné peut effectivement créer un domaine de protection «fils» et lui passer des capacités pour y coupler un sous−ensemble des segments dont il dispose dans le domaine courant. Il contrôle ainsi le contexte du domaine «fils», notamment les segments de code, mais pour démarrer l'exécution de ce code, le flot doit d'abord enregistrer un portail en spécifiant une procédure de son choix comme point d'entrée dans le domaine «fils», puis appeler cette procédure en utilisant le nom de portail obtenu.

Comme dans les autres projets qui réalisent une communication inter−domaine par appel de procédure à distance comme Amœba, Mach ou Mungi, Opal utilise un support langage pour masquer les mécanismes sous−jacents de cet appel. Il s'agit ici du langage C++, et l'appel de procédure à distance prend la forme d'un appel de méthode sur un objet, permettant ainsi à un programmeur d'applications de traiter de la même façon les appels de procédure qui s'exécutent dans le domaine de protection courant, et ceux qui doivent s'exécuter dans un domaine de protection serveur.

II.3.2 Projet Angel

Avant toute autre chose, rappelons brièvement les caractéristiques du système développé dans le cadre du projet Angel. Les entités rencontrées dans l'utilisation de la mémoire virtuelle répartie d'Angel sont :

- les objets, c'est−à−dire des segments de mémoire dont les droits d'accès sont exclusivement gérés par le service du Gestionnaire d'objets,
- les processus, c'est−à−dire les flots d'exécution,
- les domaines, c'est−à−dire le contexte d'exécution des processus. Il est constitué de capacités qui décrivent les droits accordés sur les objets aux processus s'exécutant dans le domaine.

Parmi tous les droits qui peuvent être décrits par une capacité, le droit d'effectuer une interruption logicielle vers l'objet désigné est particulièrement intéressant ici. En effet, le mécanisme d'interruption logicielle disponible dans Angel complète le

service de la mémoire virtuelle répartie, en permettant un partage actif de données entre deux domaines de protection. Et afin de préserver le contrôle d'accès inhérent aux domaines, l'accès aux interruptions logicielles est lui−même contrôlé. Son fonctionnement, illustré sur la figure Fig. 2.6, se déroule selon les trois étapes suivantes :

1. le processus émetteur de l'interruption logicielle s'exécute ici dans un domaine que nous appellerons domaine Client. Ce processus déclenche le mécanisme d'interruption en précisant deux paramètres : d'abord, il convient de désigner le destinataire de cette interruption. Le premier paramètre est donc le nom du processus destinataire. Les processus étant représentés par des objets Angel, le domaine Client doit donc disposer avant tout d'une capacité lui conférant le droit d'effectuer l'interruption logicielle du processus visé. Un second paramètre peut intervenir lors du déclenchement d'une interruption : en effet, il est possible d'accompagner l'interruption d'un complément d'information sous la forme d'un pointeur. Ce pointeur désigne un objet partagé par les deux domaines, Client et Serveur, et dans lequel le processus émetteur dans le domaine Client a rangé les informations qu'il souhaite communiquer.
2. avant de signaler effectivement l'interruption au domaine Serveur, le système doit effectuer des vérifications supplémentaires pour s'assurer de la validité de l'interruption vis−à−vis du contrôle d'accès. Le pointeur précisé en paramètre localise les données sur lesquelles seront effectuées ces vérifications : le domaine Client doit disposer d'une capacité d'écriture sur l'objet partagé désigné par ce pointeur. Quant au domaine Serveur, il doit disposer d'une capacité de lecture de cet objet. Si l'un de ces critères n'était pas vérifié, la requête d'interruption serait immédiatement rejetée.

   Il convient également de noter ici, que la communication du pointeur n'est a priori pas fiable. Ainsi ce pointeur n'est utilisé qu'à titre d'indication.
3. l'interruption logicielle prend alors place dans le domaine Serveur (selon la terminologie anglo−saxonne, il s'agit d'un *upcall*), et le processus traitant cette interruption va rechercher les éventuels paramètres et informations liés à ce signal d'interruption, et disponibles dans l'objet partagé entre les deux domaines, et effectuer le traitement associé.

L'association du mécanisme d'interruption logicielle et de la mémoire virtuelle répartie et partagée d'Angel permet donc de réaliser des communications entre différents domaines de protection tout en préservant les contrôles d'accès définis par les domaines Client et Serveur.

Cette association permet notamment de réaliser un appel de procédure à distance selon le mécanisme classique du RPC [Birrel 84] ou de sa version optimisée pour une mémoire virtuelle partagée, le LRPC [Bershad 90] : pour cela, le domaine Serveur cité dans l'exemple de la figure Fig. 2.6, doit simplement effectuer une réponse au domaine Client en déclenchant une autre interruption logicielle symétrique, dont le domaine Client est le destinataire, et où l'objet partagé est le vecteur des paramètres de retour.

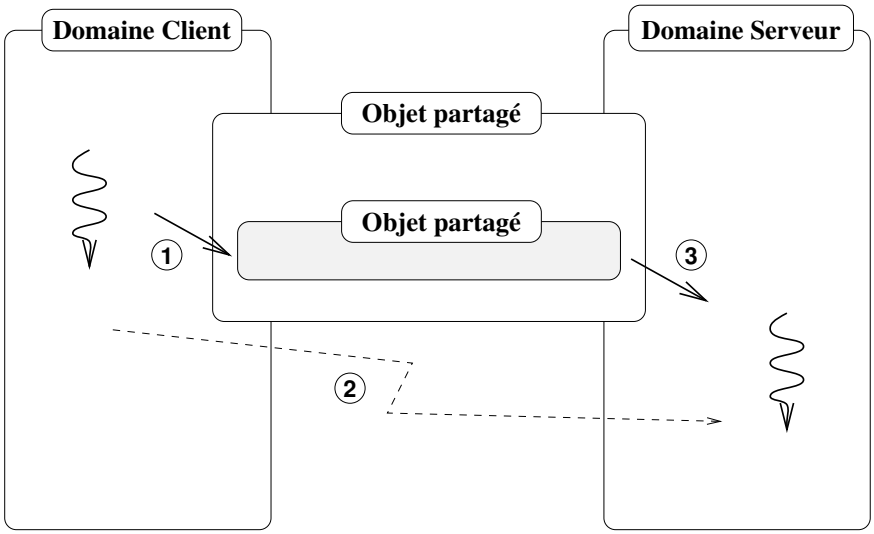


Fig. 2.6 : Communication inter−domaine dans Angel

II.3.3 Projet Mungi

Le projet Mungi [Vochteloo 93] offre un contrôle d'accès aux services partagés qui ne correspond pas tout à fait à la définition des procédures protégées énoncée par Dennis et Van Horn. En effet, le principe de la communication entre deux domaines de protection de Mungi repose exclusivement sur l'évolution dynamique du contexte d'exécution, notamment pour étendre ce contexte et englober ainsi les objets à partager nécessairement pour réaliser une communication, ou bien encore pour réduire ce contexte afin d'exécuter de façon sûre le code d'un programme suspect.

Mais avant de détailler les mécanismes d'extension et réduction des contextes d'exécution, il est nécessaire de revenir sur la définition d'un domaine de protection dans Mungi. En effet, il existe dans la terminologie adoptée dans les articles et rapports sur Mungi deux types de domaines de protection qu'il convient de distinguer :

- les domaines réguliers (appelés aussi RPD, pour *Regular Protection Domain*) : ils sont définis de façon statique par un nœud−P de l'arbre des capacités. Ce sont eux qui constituent les domaines de protection au sens

courant du terme, et chaque nœud−P de l'arbre partagé des capacités constitue un domaine régulier potentiel. Par exemple, chaque utilisateur du système dispose de son domaine régulier

- les domaines actifs (appelés aussi APD, pour *Active Protection Domain*) : ce sont plus exactement les contextes qui peuvent évoluer dynamiquement au fil de l'exécution d'un processus.

  Chaque processus dispose donc de son propre domaine actif, que l'on peut décrire par une liste de C−listes. À sa création, un processus peut, soit hériter du domaine actif du processus qui le crée, soit initialiser son domaine actif à partir d'un domaine régulier spécifié par son créateur.

Cette évolution des domaines actifs peut bien sûr résulter de la modification des C−listes par un processus qui dispose de capacités suffisantes sur ces C−listes, mais alors les domaines réguliers qui contiennent cette C−liste sont également affectés. Mais ce qui nous intéresse ici, ce sont les modifications apportées par des appels système qui permettent à un processus Mungi de réduire la liste de C−listes de son domaine actif. L'exemple de la figure Fig. 2.7 permet d'illustrer cette distinction : le domaine actif représenté a été initialisé à la création du processus avec le domaine régulier décrit par le sous−arbre dont le nœud−P_1 est la racine, et au cours de l'exécution du processus, le domaine actif a subi une restriction : la suppression de la C−liste pointée par le nœud−P_1. Ce mécanisme permet à un processus de lancer une exécution non fiable en créant un processus dont le domaine actif est un sous−ensemble des C−listes du domaine actif courant. Cette première facette de l'évolution d'un domaine actif est donc semblable aux contrôles exercés par un processus sur la sphère de protection des processus inférieurs qu'il a créés.

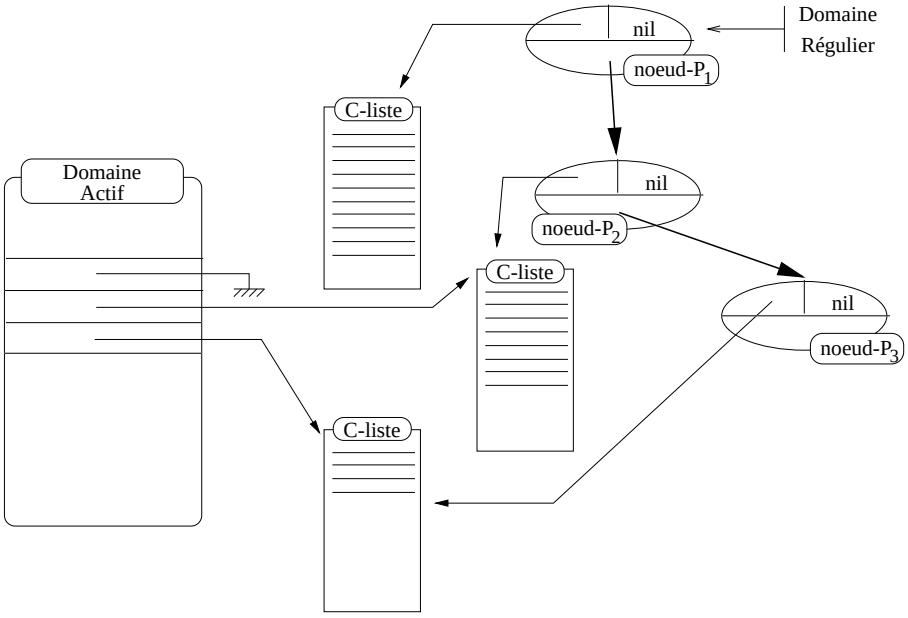


Fig. 2.7 : Évolution du domaine de protection actif

Par ailleurs, Mungi offre la possibilité à deux domaines de protection de communiquer par un mécanisme d'extension temporaire de domaine. Dans l'étude que nous avons faite au début de ce chapitre, ce mécanisme s'apparente au deuxième cas de communication par procédure protégée : celui où l'exécution de cette procédure a lieu dans le contexte de l'appelant.

En effet, le mécanisme d'extension de domaine de Mungi est mis en œuvre à l'appel de procédures particulières appelées procédures PDX (pour *Protection Domain eXtension*), et se déroule ainsi : lorsqu'un processus effectue un appel à une procédure PDX, cet appel est dérouté comme tout appel de procédure pour retrouver parmi les C−listes du domaine actif du processus, une capacité associée à l'objet contenant le code de cette procédure, et en vérifier le droit d'exécution. Si la capacité trouvée identifie une procédure PDX, alors cette capacité désigne non seulement l'objet qui contient le code de cette procédure, mais également la liste des points d'entrées valides dans cet objet, et une C−liste qui constitue le contexte protégé de cette procédure PDX. L'exemple de la figure Fig. 2.8 présente une telle capacité. Le domaine actif du processus contient les C−listes 1 et 2, dans lesquelles la recherche ne donne pas de capacité «normale» permettant de coupler directement l'objet, mais une capacité sur une procédure PDX.

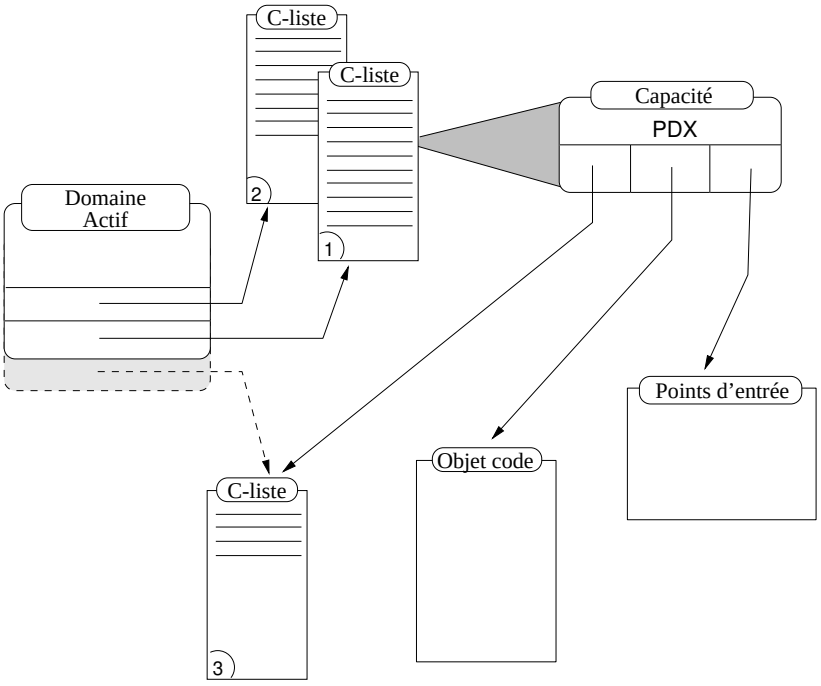


Fig. 2.8 : Appel d'une procédure protégée dans Mungi

Si l'appel correspond à un point d'entrée valide, alors le système ajoute au domaine actif, la C−liste de la procédure PDX (la liste 3 dans l'exemple) et modifie la pile du processus pour dérouter au retour cet appel sur une adresse qui génère une faute de protection. Maintenant, le processus peut exécuter le code de la procédure protégée dans le nouveau contexte ainsi formé par extension de son domaine actif. Au retour de l'appel, la faute de protection est interceptée par le système qui supprime alors la C−liste de la procédure PDX du domaine actif du processus.

Ce mécanisme présente donc deux atouts : d'abord, un processus utilisateur ne distingue pas l'appel d'une procédure PDX de celui d'une procédure normale. Ensuite, l'extension du domaine de protection évite tout transfert de capacités du contexte appelant vers le contexte appelé, puisque lors de l'exécution de la procédure PDX, le processus dispose de l'accès intégral à l'environnement du contexte appelé. Tout cela évite donc les charges d'emballage et de déballage des paramètres rencontrées dans les mécanismes d'appel de procédure à distance utilisés dans d'autres projets

Par contre, il convient d'ajouter que l'accès à l'environnement du contexte appelant n'est pas toujours souhaitable, notamment dans un scénario où le code de la procédure PDX est suspect pour le processus appelant. Certes, le processus appelant peut effectuer une restriction préliminaire de son domaine actif, mais cela ne permet pas un contrôle aussi fin et systématique sur les droits d'accès échangés,

que celui qui est effectué, par exemple, dans l'échange d'un message entre deux tâches Mach.

II.4 Synthèse récapitulative

Au cours du premier chapitre, l'étude de la protection des données partagées dans les systèmes Opal, Angel et Mungi nous a montré que ces réalisations visent des objectifs communs qui sont :

- la généralité du modèle de protection,
- la souplesse de ce modèle de protection,
- la maniabilité des informations de protection,
- les performances,
- l'uniformité de la désignation et le masquage de la répartition.

Dans le cas de la protection des services partagés, l'étude des mêmes systèmes Opal, Angel et Mungi nous montre que les objectifs communs sont légèrement différents. Notamment, il n'est déjà plus possible de parler de généralité, ni de souplesse du modèle de protection dans la mesure où le mécanisme de changement de domaine qui met en œuvre la protection des services partagés est avant tout, intégré au modèle d'exécution et dépendant du modèle de communication du système. Ainsi Opal n'offre cette protection que sur la base d'un appel de méthode C++, Mungi ne l'offre que sur la base d'un appel de procédure. Cependant Angel permet d'effectuer un changement de domaine de protection sur un éventail plus large de mécanismes de communication, et démontre ainsi que cette généralité est un objectif viable et accessible. Les performances des mécanismes de changement de domaine correspondent à celles des mécanismes de communication sous−jacents, et même si Opal et Angel fournissent des optimisations de type LRPC, les performances ne constituent pas ici un problème inhérent aux mécanismes de changement de domaine.

Rappelons ici quels sont les objectifs généraux pour la réalisation d'un changement de domaine qui est le mécanisme de base de la protection des services partagés. Il s'agit avant toute autre chose de contrôler les points d'entrée dans un domaine donné. Cet objectif primordial est bien sûr atteint par Opal, Angel et Mungi qui contrôlent respectivement et selon le modèle d'exécution qu'ils offrent, l'accès aux méthodes, aux interruptions logicielles ou aux procédures qui constituent les points d'entrée es domaines de protection. Notons ici que l'accès à ces points d'entrée se fait par présentation explicite d'une «capacité d'appel» pour ce point d'entrée. Mungi offre cependant la possibilité d'effectuer la liaison à ce point d'entrée de façon implicite, c'est−à−dire que c'est le système qui aura la charge de

retrouver la «capacité d'appel» (capacité de procédure PDX) correspondant à l'adresse invoquée.

Il convient ensuite de contrôler également les informations de protection qui sont échangées entre les deux domaines concernés par ce changement de domaine. Le cas de Mungi est particulier car, dans la mesure où l'exécution de l'appel de procédure protégée s'effectue dans le contexte de l'appelant, nous nous trouvons dans le cas décrit au paragraphe II.2.1.2, et le contrôle des informations de protection en cas de méfiance se limite à une éventuelle restriction du domaine de protection actif, préalable à l'appel. Par contre, Opal et Angel qui permettent également le transfert de droits (capacités publiques) lors d'un appel, se placent dans le cas décrit au paragraphe II.2.1.1. Dans ces deux systèmes, le contrôle des informations échangées est entièrement confié à la responsabilité du domaine appelé. Cette approche permet au domaine appelé d'assurer un contrôle particulier pour chaque appel entrant. En cas de méfiance mutuelle, le domaine appelant doit effectuer une restriction préalable des capacités publiques confiées au domaine appelé.

Ainsi, dans ces trois systèmes, tout contrôle des droits échangés entre les domaines de protection doit être réalisé de manière explicite, et reste indépendant du mécanisme de changement de domaine. Le seul contrôle inhérent au mécanisme est celui du point d'entrée dans un nouveau contexte.

Afin de disposer d'une vue d'ensemble des particularités de chacun des trois systèmes de grandes mémoires virtuelles que nous avons étudiés, le tableau, ci après, récapitule les mécanismes de protection offerts et mis en œuvre par ces systèmes.

Projet / Particularités	Opal	Angel	Mungi
Type de capacités	publiques de type (serveur + identificateur)	publiques de type (serveur + identificateur)	publiques et chiffrées
Désignation	mixte (expl./implicite)	explicite (biscuits)	mixte (expl./implicite)
Mécanisme de communications	RPC	interruption logicielle	RPC
Liaison	explicite	explicite	expl./implicite

Projet / Particularités	Opal	Angel	Mungi
Interface offerte	appel de méthode C++	interruption logicielle OU appel de procédure OU appel de méthode C++	appel de procédure
Contexte d'exécution des appels	chez l'appelé	chez l'appelé	chez l'appelant
Transfert de droits à l'appel	oui	oui	– (inutile)
Contrôle des droits transférés	non	non	non

Chapitre III

Sirac : présentation, objectifs et particularités

Ce chapitre présente le cadre dans lequel cette étude a été menée. Il s'agit du projet Sirac, projet commun aux instituts de recherche IMAG et INRIA.

Le projet Sirac a démarré en fin 1994 ; il est devenu projet INRIA en fin 1995. Il fait suite au projet Guide, mené dans le cadre d'une unité mixte Bull–IMAG de 1990 à 1994. Ce projet a abouti à une plate–forme industrielle de développement d'applications pilotes réalisées par des partenaires extérieurs. L'expérience du projet Guide a permis d'acquérir une compétence dans le domaine des systèmes et applications répartis, et d'identifier des problèmes pertinents, qui sont abordés dans Sirac.

La description [Balter 95] des motivations et objectifs généraux du projet Sirac permet en particulier d'introduire la définition [Dechamboux 95] d'un service de gestion des données persistantes et réparties, et les différents aspects abordés dans la réalisation de ce service qui m'a servi de support d'expérimentation pour cette étude.

Enfin, il convient d'identifier et de présenter les motivations spécifiques qui ont orienté les choix effectués pour définir la gestion de la protection d'accès au sein de ce service de gestion des données persistantes et réparties.

III.1 Présentation générale de Sirac

L'évolution des systèmes informatiques vers des configurations réparties est un phénomène général qui concerne tous les secteurs d'activité. Elle est favorisée par la généralisation des réseaux de communication et par la tendance des entreprises à décentraliser leurs structures de production et de décision.

Les systèmes pour le développement des applications réparties doivent fournir d'une part l'infrastructure nécessaire au partage d'information entre les machines d'un réseau et à l'exécution répartie de programmes, d'autre part les outils nécessaires à la construction, à l'installation et à l'administration des applications. Ces

deux aspects définissent le champ du projet Sirac, dont l'objectif général est de concevoir et de réaliser un environnement pour le développement et l'exécution d'applications réparties. Le rapport technique [Balter 95] donne une description plus détaillée de ces objectifs.

La démarche suivie s'appuie sur la réalisation de prototypes expérimentaux et sur leur validation au moyen d'applications réelles.

Le projet développe des actions de recherche dans les deux domaines suivants :

- construction d'applications réparties : l'objectif est de fournir des outils répondant à deux besoins : a) construire des applications réparties en combinant des techniques de programmation à base d'objets et des techniques d'intégration de composants ; b) faciliter l'administration, la configuration et l'évolution de ces applications.
- services pour le support d'objets partagés persistants répartis : l'objectif est de fournir un support générique et efficace utilisable pour la construction de plates−formes à objets répartis et de serveurs d'objets, en utilisant une mémoire virtuelle partagée répartie. Sa conception tient compte des nouvelles infrastructures matérielles (grandes mémoires virtuelles et réseaux rapides). Dans ce cadre, l'étude présentée dans ce mémoire constitue une action spécifique menée sur le service de la protection.

La classe d'applications considérée en priorité est celle des applications coopératives, caractérisée par la coopération autour d'une tâche commune d'équipes géographiquement réparties, avec partage étroit d'informations et interaction en temps réel. Partage et interaction devront prendre une forme beaucoup plus intégrée que les modes actuels (transfert de fichiers et courrier électronique). Les domaines d'application du travail coopératif sont très variés et intéressent de nombreux secteurs d'activité parmi lesquels on peut citer : la formation (télé−enseignement), la conception assistée par ordinateur, et plus généralement la bureautique (téléconférence, édition coopérative de documents, etc.) et les systèmes d'aide à la décision.

Il s'agit donc d'un domaine très étendu qui fait l'objet d'une attention croissante de la communauté scientifique. Ces dernières années, un effort particulier a porté sur la puissance et la convivialité des services de coopération offerts aux utilisateurs de ces logiciels pour un secteur d'application donné. En comparaison, il y a eu beaucoup moins de travaux portant sur la conception et la mise en œuvre d'infrastructures pour le développement de ces applications. La plupart des applications issues de ces travaux ont été réalisées à l'aide de mécanismes ad hoc sur des plates−formes d'usage commun (Unix ou Windows). Ceci se traduit en général par une grande

hétérogénéité des solutions et par une certaine difficulté à faire évoluer ces applications pour prendre en compte l'évolution des besoins et à réutiliser tout ou partie d'un système existant pour construire une nouvelle classe d'application. Cette absence de généricité se traduit globalement par un coût de production et de maintenance élevé.

L'objectif du projet Sirac n'est pas d'innover dans le domaine des applications. La prise en compte des applications répond à un double objectif : étude des besoins en vue d'établir un cahier des charges pour l'environnement de développement d'une part, fournir une base expérimentale pour valider et évaluer les résultats du projet d'autre part.

On trouve dans la littérature plusieurs tentatives de classification des applications coopératives. Ces études utilisent divers critères (type de service offert, critère espace–temps, etc.) pour caractériser les besoins des classes d'application en termes d'architecture, de méthodologies et outils de conception et de services systèmes nécessaires à leur mise en œuvre. Ces classifications ne sont pas totalement satisfaisantes dans la mesure où les applications coopératives considérées dans le projet combinent les besoins de plusieurs classes d'applications. C'est pourquoi il est préférable dans ce projet d'utiliser un mode de caractérisation plus large, dans lequel une application coopérative peut être représentée par une séquence d'interactions entre un ensemble d'entités (appelées des agents dans la suite), qui représentent des utilisateurs ou des composants logiciels. En utilisant ce modèle simple, une application coopérative est donc définie par les éléments suivants :

- l'espace de coopération, représenté par les données partagées et réparties, qui permet une communication indirecte entre les agents coopérants.
- les canaux de communication, qui permettent des échanges directs entre les agents (par exemple un canal audio dans un collecticiel synchrone).
- le module de coordination, qui permet de contrôler globalement les interactions entre agents via l'espace de coopération et les canaux de communication.

Une étude plus approfondie des applications conçues selon ce modèle permet d'identifier un certain nombre de besoins, parmi lesquels on peut citer :

- la réutilisation d'applications existantes, combinée avec le développement de logiciels additionnels. En effet, un nombre significatif d'applications coopératives sont conçues à partir d'applications mono–usager (existantes) pilotées par un module de coordination (à développer).

- la dynamicité des applications : la composition d'une application évolue pendant l'exécution de l'application, par exemple par suite des connexions et déconnexions dynamiques de participants à l'application.
- la mise en œuvre de l'espace de coopération met en évidence l'importance du partage des données, le plus souvent géographiquement distribuées. Le contrôle du partage requiert une gamme plus ou moins étendue de mécanisme de synchronisation (exclusion mutuelle, lecteurs–rédacteurs, transactions, etc.)
- la gestion des canaux de communication directs requiert des mécanismes de gestion des ressources physiques, ainsi que des services de plus haut niveau permettant de mettre en œuvre des schémas de coordination divers entre les agents. On distingue principalement un service de communication asynchrone (événementielle), assorti d'un service de désignation associative, permettant une communication anonyme entre les agents participants.
- l'administration de l'application : le déploiement de l'application dans un environnement réparti donné nécessite que certaines parties de l'application (placement des données et des agents, utilisation des canaux de communication, etc.) puissent être réglées en fonction de cet environnement. La fonction d'administration doit, en particulier, prendre en compte les contraintes imposées par l'autonomie des sites participants.

III.2 Définition d'un service de gestion des données persistantes et réparties

III.2.1 Motivations d'un tel service

Les applications visées ont besoin de services fournis par le système sous–jacent pour la gestion d'objets persistants répartis. Des services analogues sont requis par d'autres environnements de construction d'applications telles que les SGBDOO ou les services de persistance de CORBA. En outre, un service de partage d'objets peut être utilisé par des sous–systèmes logiciels qui ont besoin de partager des données persistantes, sans avoir nécessairement l'usage de toutes les fonctions d'un SGBD : environnements d'exécution de langages à objets persistants, serveurs pour le World Wide Web, bases de documents hypermédia. Il est possible d'espérer qu'un grand nombre d'environnements déjà existants pourront, moyennant un effort réduit de portage et d'adaptation, bénéficier d'un tel service.

Le partage d'objets persistants est identifié comme un besoin commun à une large classe d'applications et d'environnements d'exécution répartis, allant des

grappes de machines au World Wide Web. L'action «gestion de données partagées persistantes réparties» a pour objectif de fournir des services pour réaliser ce partage de données.

Un tel service doit avoir les qualités suivantes :

- Adaptabilité : le service doit pouvoir être configuré pour répondre aux besoins variables des applications, et même à des besoins différents à l'intérieur d'une même application. La généricité est un moyen d'assurer l'adaptabilité, en fournissant un ensemble de fonctions élémentaires pour la construction des services à la demande.
- Portabilité : le service doit pouvoir aisément être intégré à un nouvel environnement d'exécution. Ceci peut être assuré en le construisant comme une extension modulaire d'une plate–forme largement diffusée, comme Unix.
- Sécurité : le service doit fournir les moyens de mettre en œuvre les politiques de protection d'accès aux données partagées définies par les applications et de rendre les données résistantes aux pannes.
- Efficacité : les performances d'un service conditionnent son acceptation par les utilisateurs. Elles doivent donc être bonnes, et en rapport avec les services requis (en particulier, une application ne doit pas payer le coût d'un service qu'elle n'utilise pas).

III.2.2 Problèmes posés

Pour répondre aux exigences citées, la conception d'un service de gestion d'objets persistants répartis implique la résolution des problèmes examinés ci après :

1. les besoins d'efficacité et de disponibilité conduisent à dupliquer les données sur plusieurs sites, sous forme de caches logiciels. Il se pose alors le problème de garantir la cohérence mutuelle entre les données contenues dans ces caches. Diverses politiques de gestion de cache ont été définies, qui réalisent des compromis divers entre efficacité et fraîcheur des données. Les politiques les plus élaborées s'appuient sur une connaissance fine, par l'application, des schémas d'accès aux données (voir section III.2.3). Différentes applications (ou même différentes parties d'une application) peuvent nécessiter des politiques différentes. Il paraît dès lors intéressant de permettre à chaque application de réaliser ses propres protocoles de cohérence à partir de fonctions élémentaires fournies par le service. Le choix de ces fonctions doit concilier l'efficacité, la commodité d'utilisation, et la «complétude» (possibilité de programmer une gamme étendue de services).

2. la recherche d'efficacité conduit en outre à effectuer un placement dynamique des objets et de l'exécution, en exploitant les indications fournies par les applications et les observations faites par le système. Ces informations peuvent aussi être exploitées pour la gestion efficace des communications, au moyen de protocoles spécialisés.
3. les exigences de sécurité liées à celles d'efficacité des mécanismes proposés nécessitent une protection des données avec un grain moyen. Ceci peut être obtenu par l'usage de capacités logicielles associées à des segments (ensemble de pages contiguës). Un problème important est la mise en œuvre efficace de ces mécanismes de protection.

Les problèmes ci−dessus se posent dans divers types de systèmes de partage de données : systèmes répartis de gestion de fichiers, mémoire virtuelle répartie, service d'accès à grande distance comme le World Wide Web. Un des objectifs visés est d'examiner si des solutions développées dans l'un de ces environnements peuvent se transposer aux autres.

Le premier champ d'expérimentation choisi dans Sirac se traduit par la réalisation d'un service de gestion de données persistantes sur une grappe de machines homogènes reliées par un réseau local à haut débit, en utilisant une mémoire virtuelle répartie de grande taille. Ce choix est motivé par les arguments suivants [Balter 95] :

- le modèle de programmation fourni par la mémoire virtuelle partagée est plus simple à utiliser que le modèle client−serveur dès lors qu'il s'agit de mettre en œuvre des applications gérant des données partagées. Ce fait est maintenant largement reconnu tant par la communauté des systèmes pour le parallélisme que par celle des systèmes répartis.
- les outils pour la réalisation de la cohérence, de la persistance, et de la tolérance aux fautes d'une mémoire virtuelle répartie peuvent être rendus génériques, et réutilisables pour d'autres environnements.
- la mémoire virtuelle répartie peut servir de champ d'expérience pour les travaux d'expérimentation sur la construction d'applications coopératives L'adaptation à ce nouveau support des techniques de coordination, de mise au point et d'observation des applications est un domaine encore peu exploré où on peut attendre d'importantes retombées.
- les progrès récents aussi bien dans les méthodes de mise en œuvre des mémoires virtuelles réparties que dans les techniques de réseaux laissent espérer l'obtention de bonnes performances.

Cette première application, décrite dans la section qui suit, sert de support d'expérimentation pour cette étude.

L'objectif du travail est de réaliser sur un réseau local de machines homogènes une mémoire répartie de segments persistants, fournissant des services génériques pour la cohérence, la protection et le stockage des données.

L'architecture de base est constituée d'un ensemble de machines clients et de machines serveurs homogènes, connectées par un réseau local. Ces machines sont des stations Unix munies des services de partage, persistance et répartition. Une telle configuration constitue une unité d'administration désignée sous le terme de «*cellule*». Les problèmes posés par la mise en œuvre d'une cellule sur un réseau général et par l'interconnexion entre cellules seront considérés dans une étape ultérieure du projet.

Les machines serveurs sont chargées de la conservation à long terme des données permanentes utilisées par les clients. Clients et serveurs communiquent au travers d'une mémoire virtuelle partagée persistante. Cette mémoire ne constitue qu'une partie de l'espace adressable par une machine client, l'autre partie étant réservée aux données privées, et notamment aux informations de protection. La mémoire virtuelle persistante sert de support d'adressage aux données permanentes gérées par les serveurs ; elle peut contenir en outre des données partagées n'ayant pas d'image sur disque. Les structures d'exécution sont les processus du système Unix dans lequel ces services sont intégrés. L'architecture globale du système est décrite dans le rapport [Dechamboux 95] et illustrée sur la figure Fig. 3.1.

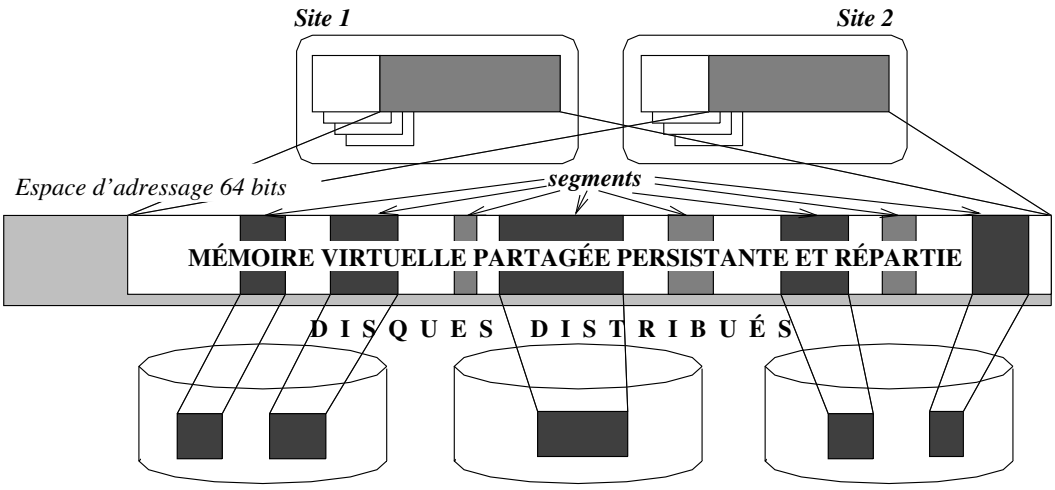


Fig. 3.1 : Support de la mémoire répartie partagée

Les informations sont organisées en segments, de taille fixée à la création, couplés dans la mémoire à des adresses fixes. Ainsi, les données partagées sont vues par tous les processus à la même adresse virtuelle, ce qui permet d'utiliser les adresses comme identifiant unique. Cette méthode de désignation devrait se traduire par un gain de performances significatif dans la manipulation des données.

Les segments se situent à un niveau intermédiaire entre les objets, entités logiques construites par les environnements des utilisateurs, et les pages, entités physiques manipulées par les plates−formes sous−jacentes. Sauf le cas particulier de certains objets de grande taille (par exemple des objets multimédia), les segments sont des regroupements d'objets. Les segments sont également des unités de protection.

Les principaux problèmes à résoudre pour la gestion de la mémoire virtuelle sont énumérés brièvement ci−dessous, avec le principe des solutions envisagées pour chacun. Trois aspects particuliers, liés à la cohérence, à la survie en cas de panne d'un site et à la protection des données partagées (qui nous intéressent plus particulièrement ici) sont présentés plus en détail plus loin.

- Allocation de la mémoire aux segments. Le principe consiste à ne pas réallouer les adresses de segments, pour garantir leur unicité. Ceci est rendu possible par l'étendue d'adressage fournie par les 64 bits. Chaque serveur gère des collections de segments, et les zones d'adresse de mémoire virtuelle sont réparties entre les serveurs (statiquement, avec possibilité d'extension).
- Structuration de l'espace de recherche des segments. Lors d'une faute de page, le système doit retrouver les données constituant le segment. La taille des espaces virtuels interdit la gestion classique par tables hiérarchisées. Il est prévu d'utiliser des méthodes de recherche des segments utilisant le hachage.
- Récupération des «méta−données» (informations de gestion des segments). Si on ne réalloue pas les adresses virtuelles des segments détruits (explicitement détruits ou récupérés), il faut être en mesure de collecter les méta−données associées aux segments. Cette récupération sera réalisée par un ramasse−miettes.
- Gestion de la cohérence des mémoires locales des machines clientes, contenant des portions des segments partagés. Ce point est détaillé en III.2.3.
- Résistance aux pannes. Dans un premier temps, il s'agit essentiellement de rendre permanente l'image des segments persistants. Ce point est détaillé en III.2.4.
- Protection des segments dans un espace virtuel partagé unique. C'est le point auquel cette étude s'intéresse. Il est abordé en III.3, et le chapitre suivant le décrit plus en détail, notamment les choix effectués et les problèmes rencontrés lors de la réalisation.

- Grain du partage. L'expérience tend à montrer que la taille moyenne des objets usuels est de l'ordre de quelques centaines d'octets, alors que la taille des pages (qui tend à augmenter) est de plusieurs milliers d'octets. Cette différence cause les problèmes dits de «faux partage» dans la mesure où plusieurs entités logiques (objets) peuvent être physiquement localisées sur la même page. La solution envisagée, en cas de conflit, consiste à abandonner la page comme unité de verrouillage et de gestion de cohérence, au profit d'unités logiques de plus petite taille appelées «zones» (zones contiguës de taille quelconque).

III.2.3 Gestion de la cohérence

La conception du service de gestion de la cohérence repose sur deux principes de base :

1. associer la gestion de la cohérence des informations partagées à la gestion de la synchronisation ; le service de gestion de cohérence peut ainsi disposer d'informations qui lui permettent de réduire le nombre des transferts de données ;
2. définir une interface générique restreinte comportant des fonctions élémentaires, et permettant de construire, à la demande, différents schémas de synchronisation.

Le premier principe permet d'améliorer l'efficacité de la gestion de cohérence, pour des applications qui utilisent un schéma de synchronisation. Un exemple simple permet de comprendre son principe : supposons qu'un processus modifie une copie locale d'une donnée protégée par un verrou exclusif ; tant que le verrou est en place, il n'est pas nécessaire de propager les modifications aux copies distantes de cette donnée puisqu'aucun autre processus ne peut y accéder. Il suffit de propager les modifications lors de la levée du verrou, ou même seulement lors du prochain verrouillage. En contrepartie, il est évident que seules les applications qui respectent les règles de synchronisation pour l'accès aux données bénéficient d'une garantie de cohérence.

Le deuxième principe permet à tout environnement, ou même à toute application, de spécifier un schéma spécifique de gestion de cohérence. Une application ne supporte que le coût des mécanismes qu'elle utilise effectivement.

Les unités d'information manipulées (unités de synchronisation et de mise en cohérence) sont des zones de taille quelconque. Pour chaque zone z, le système gère une copie de référence $m(z)$ appelée copie «maîtresse» dont il contrôle le site de localisation $s(z)$. L'interface générique fournie permet à un autre site autre que $s(z)$ d'obtenir une copie locale de $m(z)$ ou bien de modifier $m(z)$. Des appels

ascendants peuvent être attachés à ces actions, pour pouvoir leur associer des opérations spécifiées et programmer ainsi les protocoles de cohérence. Les services de synchronisation utilisent cette interface afin d'intégrer la gestion de la cohérence au schéma de synchronisation.

Des services de synchronisation par défaut sont fournis aux applications n'ayant pas de besoins particuliers. Les services suivants sont notamment souhaitables :

- un service de verrouillage de zones gérées en cohérence faible selon le schéma dit «*entry consistency*». La synchronisation des accès aux zones est assurée par des verrous et la mise en cohérence des données d'une zone (report des modifications effectuées sur des copies distantes) est réalisée, si nécessaire, lors du verrouillage de cette zone.
- un service de verrouillage transactionnel.

III.2.4 Permanence et résistance aux pannes

Comme nous l'avons déjà noté, la permanence envisagée consiste d'abord à assurer la permanence des segments en cas de panne d'un site. Il existe deux solutions :

- imposer un mécanisme unique de sauvegarde atomique sur disque d'un ensemble de zones appartenant à des segments différents,
- fournir les outils de base permettant de réaliser un tel mécanisme.

Dans les deux cas, on a besoin de l'équivalent d'un système de fichiers pour gérer les images sur disque des segments permanents et d'un mécanisme de journalisation pour assurer l'atomicité des opérations de recopie de l'espace d'exécution dans l'espace de stockage. Dans la première solution, le service de journalisation est caché aux utilisateurs, le système ne leur fournissant qu'une primitive de recopie atomique des zones modifiées. Le choix de cette solution obligerait donc les applications qui gèrent leurs propres services de journalisation à reprogrammer ces services alors que les mécanismes existent déjà dans le système. C'est donc la deuxième solution qui est retenue : un service de stockage qui permet de créer des images de segments sur disque, et un service générique de journalisation.

Le service de stockage fournit des opérations de création ou de destruction d'une image permanente de segment et de copie atomique d'une page de segment sur son image permanente,

Le service de journalisation générique permet à l'utilisateur de créer un journal en lui associant un protocole spécifique de mise à jour. L'utilisateur crée dans son propre journal des enregistrements dont l'en–tête a un format fixe prédéfini et dont le corps n'est interprétable que par le protocole de mise à jour. Les enregistrements

d'un même journal peuvent concerner des zones de segments différents. L'utilisateur peut valider ou invalider un journal. Les journaux invalidés sont détruits. Les journaux validés sont traités par un processus de fond, qui exécute les protocoles de mise à jour associés pour libérer l'espace qu'ils occupent sur disque. Un tel service permet d'implanter indifféremment des journaux contenant des images avant, des images après, ou un mélange des deux. De la même façon, le processus de reprise après panne exécute tous les journaux validés concernant un segment donné avant de rendre le segment disponible.

III.3 Objectifs du service de protection

Plusieurs besoins et objectifs généraux présentés ci–dessus ont une incidence sur le service de la protection. Il s'agit notamment :

- des besoins exprimés par les applications coopératives tels que la réutilisation d'applications existantes pour concevoir de nouvelles applications, et la configuration dynamique de la coopération mise en œuvre par ces applications.
- des qualités requises pour l'ensemble du service de gestion des données persistantes et réparties, telles que l'adaptabilité, la portabilité, la sécurité et l'efficacité.

Répondre à ces besoins et tenir ces objectifs soulève plusieurs problèmes pour la conception d'un service de protection, et motive les choix effectués. L'inventaire de ces choix et des problèmes abordés est décrit ci–dessous pour chacun des aspects du service de protection, notamment la gestion du contrôle d'accès aux données partagées, c'est–à–dire la gestion des domaines de protection, des capacités, ainsi que la gestion du contrôle d'accès au code et aux services partagés à travers les changements de domaine.

L'objectif d'adaptabilité et de généricité exige que les mécanismes de protection permettent de mettre en œuvre différents modèles de protection. En particulier, le choix d'un modèle de protection hiérarchique serait contraire à cet objectif, et il est donc à rejeter. Ces considérations conduisent donc naturellement à opter pour des mécanismes de protection basés sur des capacités logicielles.

La proposition présentée ici, s'appuie donc sur les notions «classiques», au sens de Dennis et Van Horn, de capacité et de domaine de protection. Une capacité représente un droit d'accès à un segment, soit directement en autorisant le couplage du segment, soit indirectement en permettant un changement de domaine de protection pour réaliser l'accès. Un domaine de protection est défini comme un ensemble de capacités.

Le principe de réalisation d'un service de protection réalisé par des capacités doit cependant rester en accord avec les objectifs de réutilisation, d'adaptabilité et d'efficacité fixés globalement dans le cadre du projet. En particulier, il semble impératif de rendre indépendant des capacités le code des applications qui utilisent le service de gestion des données persistantes et réparties. Ces motivations s'expriment de plusieurs façons au niveau des applications utilisatrices :

- une même application doit pouvoir s'exécuter de la même manière (c'est–à–dire sans qu'il soit nécessaire de modifier son code) quand elle utilise le service de la protection pour ses données persistantes réparties, ou quand elle ne l'utilise pas.
- une application qui n'utilise pas le service particulier de la protection – le service de protection fait partie des services disponibles à la carte dans le cadre de la gestion de données persistantes et réparties, – ne doit en aucun cas payer le coût de ce service.
- une application qui utilise le service particulier de la protection pour ses données réparties, doit pouvoir s'exécuter de la même manière dans différentes configurations de schémas de protection de ses données.

Dans la suite, l'ensemble de ces motivations est regroupé en un seul terme : l'utilisation implicite de capacités. Cet objectif distingue Sirac des réalisations précédentes, ainsi que celles effectuées dans les projets Angel, Mungi et Opal que nous avons abordés au début de cette étude.

Utilisation implicite de capacités dans un domaine de protection

Cet objectif d'utilisation exclusive d'adresses virtuelles et d'utilisation implicite des capacités écarte les solutions dans lesquelles une application doit explicitement demander au système de lui accorder l'accès à un segment en désignant une capacité pour ce segment. L'application doit donc manipuler exclusivement des adresses virtuelles, et le système doit implicitement associer les capacités concernées par chaque adresse virtuelle.

Ceci est valable non seulement pour les accès aux données persistantes, et le couplage des segments qui les contiennent, mais également pour l'accès au code des services partagés exécutés : en d'autres termes, le changement de domaine de protection doit également être caché au code qui le provoque, donc l'application ne doit manipuler que des adresses virtuelles.

Changement de domaine

Comme nous venons de le voir, un appel de procédure dans un segment de code doit pouvoir s'exécuter dans un même domaine de protection en ne manipulant que des adresses virtuelles, ou changer de domaine de protection

sans utiliser explicitement de capacité, et en manipulant exclusivement et de la même façon des adresses virtuelles.

De plus, lors d'un tel changement de domaine, une application peut nécessiter le passage de paramètres dans le nouveau domaine. Dans la mesure où l'application ne manipule que des adresses virtuelles dans un espace d'adressage global et unique, ces paramètres sont directement utilisables dans le nouveau domaine, à moins que ce ne soient des pointeurs sur des segments partagés. Dans ce cas, le nouveau domaine doit disposer d'une capacité adéquate pour ces segments afin d'utiliser et de suivre ces pointeurs ; et là aussi, l'objectif d'utilisation implicite des capacités écarte toute solution où les capacités accompagnant les paramètres «pointeurs» seraient passées explicitement.

Administration et gestion des domaines

La gestion des capacités composant un domaine de protection est une opération de configuration et elle est donc indépendante de l'exécution des applications pour lesquelles l'usage ou non de capacités demeure caché. L'administration des domaines de protection constitue une opération (optionnelle) de configuration d'un schéma de protection pour les données persistantes partagées. Si cette configuration peut se dérouler totalement à part, en prélude à l'exécution d'une application, les applications coopératives, auxquelles nous nous intéressons dans le cadre de Sirac, peuvent également nécessiter une reconfiguration dynamique des schémas de protection au fil de l'exécution. Par exemple, les participants d'une application coopérative de documents structurés sont amenés à se concéder ou à se retirer mutuellement des droits pour la rédaction d'une ou plusieurs parties du document qu'ils éditent simultanément.

De plus, la possibilité de gérer les capacités composant un domaine de protection est également un privilège dont il convient de contrôler l'accès. Les domaines de protection constituent donc des données persistantes partagées et réparties qui doivent être protégées au même titre et de la même façon que les segments.

Gestion de la répartition

Il convient de considérer la répartition sous deux aspects différents :

1. la répartition de l'exécution : un domaine de protection est un espace d'adressage qui définit un contexte et un ensemble de droits associés pour un ensemble d'exécutions indépendamment de la localisation de ces exécutions. Ainsi ces exécutions peuvent être réparties pour profiter notamment du parallélisme. Pour respecter le critère d'indépendance

des mécanismes qui nous est fixé dans le cadre de Sirac, la gestion de la protection et la gestion de la répartition peuvent être considérées indépendamment l'une de l'autre, en gérant :

a) des domaines de protection répartis : un domaine de protection peut alors être représenté sur plusieurs machines et des exécutions peuvent partager le contexte de ce domaine et se dérouler en parallèle,

b) des changements locaux de domaines de protection : une exécution amenée à changer de domaine effectue cette opération sur le site courant, et l'application peut donc contrôler indépendamment des mécanismes de protection la répartition de son exécution.

2. la répartition des données : les domaines de protection et les capacités sont des entités réparties, partagées et potentiellement persistantes. Lorsqu'intervient une reconfiguration dynamique d'un schéma de protection, impliquant une modification de domaines de protection ou de capacités, il est donc nécessaire de garantir la cohérence, la permanence et la sécurité de ces données de protection. La gestion des informations de protection constitue alors une application qui utilise l'ensemble des services offerts par la mémoire virtuelle partagée persistante et répartie de Sirac : une forte dépendance apparaît ainsi entre le service de la protection et les autres services de gestion des données réparties. La réalisation doit donc prendre garde à ne pas compromettre l'objectif d'adaptabilité du service global d'une mémoire virtuelle répartie protégée.

Enfin, outre l'objectif d'utilisation implicite des capacités qui a de fortes implications sur les mécanismes de protection, il est essentiel de fournir un service de protection qui soit efficace, afin qu'il puisse répondre au dynamisme des applications coopératives visées et être viable dans ce contexte d'utilisation.

III.4 Situation dans l'état de l'art

Plusieurs projets de recherche explorent actuellement l'utilisation des processeurs 64 bits pour la gestion d'un espace virtuel unique dans un système réparti. Il s'agit notamment des projets Opal, Mungi, et Angel que nous avons abordés dans les chapitres précédents. Si l'utilisation des adresses virtuelles comme désignation uniforme est un acquis, peu de choses ont été proposées pour l'allocation des adresses virtuelles dans l'espace virtuel, la localisation ou la migration des segments dans cet espace. C'est un des aspects sur lesquels le projet Sirac compte

apporter de nouvelles solutions. De plus, ces projets fournissent tous des solutions incompatibles avec Unix, dans la mesure où les processus ne peuvent pas adresser directement leurs données privées.

L'état de l'art nous montre que les systèmes fondés sur la gestion d'un espace virtuel unique fournissent tous une protection de cet espace sous forme de capacités logicielles, et de domaines de protection. L'approche adoptée dans Sirac suit ce choix, et elle est plus particulièrement comparable à celle adoptée par Mungi ou Opal (voir le récapitulatif, section II.4). Par rapport à ces projets, l'apport de cette étude dans le cadre du projet Sirac se situe à deux niveaux :

- la désignation des capacités dans Sirac est implicite afin de permettre une désignation uniforme, que le service de protection soit sélectionné ou non par les applications : cet objectif interdit la manipulation de capacités publiques pour réaliser le couplage des segments partagés et accéder aux données qu'ils contiennent, ou pour réaliser la liaison d'un segment dans un autre domaine. Et comme la désignation est exclusivement réalisée par adresses virtuelles, le système doit prendre en charge la désignation des capacités qui correspondent à ces adresses.

  Cet objectif distingue Sirac des projets Opal et Mungi où la désignation des capacités est explicite pour les applications, que ce soit pour le couplage de segments de mémoire partagés, ou pour le changement de domaine à travers un appel de procédure.
- le contrôle des droits transférés dans Opal et Mungi est effectué de façon explicite par l'application qui doit au besoin manipuler les capacités et restreindre «manuellement» son domaine de protection courant pour mettre en place un contexte d'exécution fiable en cas de méfiance avec le code qu'elle exécute. Or cette approche ne permet pas à une même application de s'exécuter dans plusieurs situations de confiance ou de méfiance, puisqu'il faut en modifier le code pour configurer les domaines de protection en fonction de la situation. La proposition présentée ici dans le cadre de Sirac, doit permettre de contrôler systématiquement et implicitement les droits passés d'un domaine de protection à l'autre lors d'un appel et d'un changement de domaine

Chapitre IV

Mise en œuvre du service de protection pour Sirac

Le chapitre précédent présentait les objectifs généraux du projet Sirac et plus particulièrement les besoins spécifiques d'un service de protection. Conformément à cet objectif, la réalisation décrite ici met en œuvre des capacités désignées implicitement lors de l'accès aux données et aux services partagés et protégés. Ce service de protection intègre également le contrôle de droits échangés entre deux domaines dans le mécanisme de changement de domaine.

De plus, dans le cadre du projet Sirac, une plate–forme est développée pour offrir un ensemble de services pour le partage de données persistantes et réparties à des processus Unix qui s'exécutent sur des stations de travail homogènes. Cette plate–forme, appelée Arias, m'a donc servi de base d'expérimentation dans le cadre de cette étude pour la mise en œuvre du service de protection de la mémoire virtuelle répartie de Sirac [Hagimont 96][Saunier 95]. Après avoir présenté et défini les principes de ce service de protection, nous verrons comment il peut être mis en œuvre sur la plate–forme Arias, et comment les particularités de la plate–forme ont influencé cette mise en œuvre.

IV.1 Description du service

IV.1.1 Principes directeurs de la proposition

Comme nous l'avons vu précédemment, les objectifs énoncés pour l'ensemble du projet Sirac, notamment l'adaptabilité et la généralité des services offerts, conduisent naturellement à choisir des mécanismes de protection basés sur la mise en œuvre de capacités logicielles.

Le service proposé s'appuie donc sur les concepts de domaine de protection (dans la suite, le terme raccourci de domaine est également utilisé) et de capacités. Ces concepts sont proches de ceux rencontrés dans les projets Mungi et Opal et définis par Dennis et Van Horn :

Activité

Une activité dans Arias correspond à la définition de Dennis et Van Horn : c'est un flot d'exécution quelconque. Elle est donc naturellement réalisée par un flot d'exécution (ou *thread*) Unix sur un site donné.

Segment

Un segment dans Arias correspond comme dans la définition de Dennis et Van Horn à une zone de mémoire contiguë et le système AIX sur lequel Arias est réalisé fournit les mécanismes d'adressage d'une mémoire segmentée.

Domaine

Un domaine dans Arias correspond à la définition que donnent Dennis et Van Horn pour un processus : il s'agit d'un contexte d'exécution dans lequel s'exécutent une ou plusieurs activités. Le support d'exécution d'un domaine Arias est donc naturellement fourni sur un site donné par un processus Unix.

Comme les processus de Dennis et Van Horn, un domaine Arias est défini par une liste de capacités et les listes de capacités sont partageables entre plusieurs domaines.

Capacité

Dennis et Van Horn distinguent deux parties dans une capacité : son nom et les droits d'accès qu'elle confère. Dans Arias, l'objectif de désigner implicitement les capacités et d'utiliser exclusivement des adresses virtuelles impose le nom d'une capacité : pour une capacité qui désigne un segment, son nom doit correspondre à l'adresse virtuelle du segment désigné. Plusieurs types de capacités sont introduits dans la suite, notamment les capacités d'appel qui autorisent le changement de domaine de protection ; elles doivent obéir à cette règle de désignation afin de permettre aux applications de ne manipuler que des adresses virtuelles.

Rappelons que l'utilisation exclusive d'adresses virtuelles est motivée par l'objectif d'Arias d'intégrer des services «à la carte» dans une plate−forme adaptable aux besoins des applications. Par conséquent, le principe directeur de cette proposition réside dans une gestion des capacités réalisée intégralement par le système. Les applications ne manipulent que des adresses virtuelles et les capacités restent confinées dans le système. Chacun des mécanismes d'adressage où les capacités interviennent doit respecter ce choix. Ceci concerne essentiellement le couplage des segments protégés et l'appel de procédure protégée.

IV.1.2 Couplage des segments protégés

Dans la mesure où les capacités restent cachées à l'utilisateur, toutes les requêtes de couplage d'un segment ne précisent que l'adresse virtuelle de ce segment.

Pour traiter cette requête de couplage, le service de protection doit donc :

- identifier le domaine courant de l'activité qui demande le couplage d'un segment,
- retrouver dans ce domaine une capacité correspondant à l'adresse virtuelle de ce segment,
- vérifier que les droits accordés par la capacité trouvée sont suffisants, et valider ou rejeter la requête suivant le résultat.

Ces vérifications sont effectuées au sein d'un domaine de protection par un type spécifique de capacités, appelées capacités de couplage. Comme son nom l'indique, ce type de capacités autorise le couplage d'un segment, et il doit également préciser :

- l'adresse du segment qui sert également de nom pour la capacité,
- le mode de couplage autorisé, combinaison de droits de lecture, d'écriture, et d'exécution,
- un indicateur de droit de destruction du segment (voir § IV.3.4)

De plus, le modèle de mémoire virtuelle partagée de la plate–forme Arias effectue un couplage à la demande lorsqu'une activité accède à un segment et déclenche une exception pour défaut de segment. Ainsi, l'intervention du service de protection pour traiter une requête de couplage doit s'intégrer au traitement d'exception mis en place par le service de gestion des données partagés.

IV.1.3 Appel d'un service protégé

De la même façon, nos objectifs indiquent que le mécanisme de changement de domaine doit rester caché à l'utilisateur. Ainsi l'appel d'une procédure doit avoir une visibilité identique quand il s'effectue à l'intérieur d'un domaine de protection ou quand la procédure est protégée et nécessite un changement de domaine.

C'est pourquoi le changement de domaine est réalisé lors du traitement d'exception pour défaut de segment :

- si le domaine courant détient une capacité de couplage en mode d'exécution pour ce segment, alors la requête de couplage est immédiatement résolue. L'activité déroutée reprend alors son exécution pour effectuer l'appel de procédure dans le domaine courant.

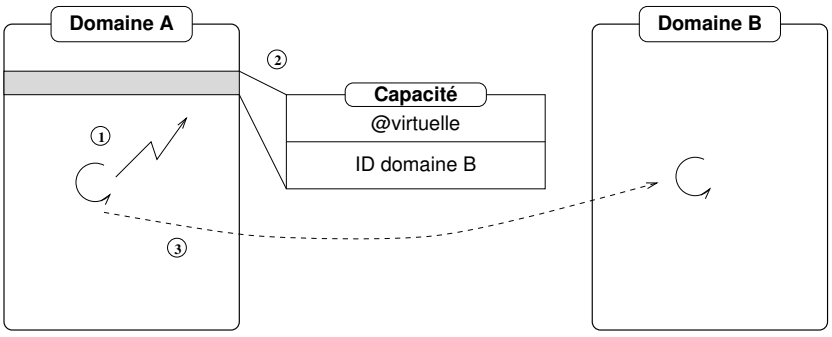
- si le domaine courant ne détient pas de capacité de couplage, mais permet d'appeler la procédure dans un autre domaine, alors l'activité déroutée change de domaine pour y exécuter la procédure appelée.

Pour permettre la mise en œuvre de ce mécanisme, il convient d'introduire un deuxième type de capacités constituant des points d'entrée ou *«entry capabilities»* pour les domaines de protection. Dans la suite, ces capacités sont désignées par le terme de capacités d'appel. Ces capacités sont uniquement utilisées lorsqu'un segment de code ne peut pas être couplé en mode d'exécution dans le domaine de protection courant. Elles permettent alors le transfert de l'activité dans un autre domaine de protection dans lequel la même procédure est appelée. Elles permettent également de spécifier quels paramètres désignent implicitement des capacités qui doivent être transférées à l'aller ou au retour du changement de domaine et de contrôler les droits ainsi transférés.

La composition d'une capacité d'appel est la suivante :

- un identificateur unique du domaine dans lequel toute activité doit exécuter la procédure protégée,
- une adresse qui est un point d'entrée d'appel de procédure,
- une spécification des règles de transfert des paramètres et des capacités qui doivent les accompagner.

Le modèle d'exécution de l'appel d'un service protégé et d'un changement de domaine est le suivant :



① Faute d'adresse

② Recherche de capacité et détermination du nouveau domaine

③ Reprise de l'activité dans le nouveau domaine

Fig. 4.1 : Principe du changement de domaine

- Une activité s'exécute dans un domaine A, et produit une faute d'adresse sur du code (c'est–à–dire branchement dans un segment non couplé en exécution).
- Le traitement du déroutement associé à cette faute procède à la recherche (voir § IV.3.3) d'une capacité de couplage du segment avec des droits d'exécution, ou d'une capacité de changement de domaine pour l'adresse en défaut. Dans le premier cas, le segment est couplé dans le domaine A ; dans le second cas, le segment ne peut pas être couplé dans le domaine courant (A), et l'activité doit aller s'exécuter dans un autre domaine (B) où l'on suppose que le segment de code référencé pourra être couplé en exécution. Le domaine B est parfaitement déterminé par la capacité d'appel.
- Avant d'effectuer le transfert de contrôle, il convient de passer les paramètres de l'appel (voir § IV.3.6).
- L'activité reprend son exécution à l'instruction fautive, dans le domaine B : ainsi, le changement de domaine s'est déroulé de façon totalement invisible pour le code de l'activité.

IV.1.4 Administration des domaines de protection

Un domaine est défini par une liste de capacités : la liste des capacités sur les segments accessibles aux activités s'exécutant dans ce domaine. Pour des applications ou des utilisateurs qui veulent gérer la protection, le service offert doit permettre d'associer des capacités aux domaines de protection.

Cette gestion peut se faire capacité par capacité, mais elle paraît bien plus facile avec des listes de capacités. Une liste de capacités peut représenter un ensemble de droits correspondant à un service. Ces droits sont accordés en donnant simplement au domaine souhaité l'accès à la liste. Cette opération accorde implicitement et en une seule opération, l'ensemble des droits contenus dans la liste.

Les listes de capacités sont désignées par des identificateurs uniques.

Puisque ces listes sont manipulées par les utilisateurs et les applications, et partagées entre eux, leur accès est contrôlé et conditionné par la possession d'une capacité. Un troisième type de capacités doit être introduit : ce sont les capacités de listes. Ces capacités permettent d'accorder des droits d'administration sur une liste et sur les capacités contenues dans cette liste. De plus, ces capacités servent de nœuds pour construire des graphes de listes qui seront utilisées pour spécifier le contenu d'un domaine de protection.

Les droits qui peuvent être définis dans une capacité de liste sont :

- le droit de consultation : ce droit autorise la consultation et l'utilisation des droits spécifiés par les capacités de la liste. Les droits inclus dans la liste

concernent les capacités de couplage, d'appel ou les capacités de listes. Ce droit permet de valider ou d'invalider tout un sous−graphe de capacités pour la recherche de capacités dans ce graphe,

- le droit de modification : ce droit permet de modifier la liste pour y ajouter ou en retirer une capacité, quel que soit son type,
- le droit de copie : ce droit permet de créer des copies des capacités de la liste avec les mêmes droits ou des droits moindres,
- le droit de destruction : ce droit permet de détruire une capacité sur une liste. Une liste de capacités est également détruite quand il n'y a plus de capacité qui la référence. Un simple compteur de références sur une liste de capacité permet de réaliser cette destruction : ce compteur est mis à jour à la création de la liste (et de la première capacité sur cette liste), lors de la recopie d'une capacité sur cette liste, et lors de la destruction d'une capacité sur cette liste.

Comme nous l'avons vu précédemment, la réalisation de ce service de capacités peut se classer parmi les systèmes à capacités confinées que nous avons vus au chapitre I. Cependant, il convient de rappeler ici que la finalité des capacités confinées et cachées consiste à offrir la possibilité d'exécuter une même application dans plusieurs configurations ou schémas de protection différents, voire sur une mémoire répartie partagée, protégée ou non, sans qu'il soit pour cela nécessaire de la reprogrammer ou de la recompiler. Cet objectif ne prend pas en compte l'administration des domaines pour configurer tel ou tel schéma de protection. En outre, cette opération est ici considérée comme une étape indépendante de l'exécution du cœur de l'application, et elle doit être réalisée par une application (ou une partie de l'application, un prologue par exemple) consciente et soucieuse de la protection. La structuration et l'organisation en listes des domaines est donc rendue visible par l'interface du service de protection

Il reste cependant deux options : manipuler et désigner ces listes par des capacités cachées ou bien par des capacités publiques. Les listes sont désignées par des identificateurs uniques, alloués dans un espace de noms homogène à celui des adresses virtuelles des segments (ce sont des adresses virtuelles en quelque sorte), mais qui est a priori distinct. Les capacités de listes pourraient donc tout à fait être réalisées par des capacités publiques. Cependant, par souci d'uniformité, la solution retenue pour la réalisation consiste à les confiner de la même manière que les capacités de couplage et les capacités d'appel : une capacité de liste est donc désignée et référencée par une pseudo−adresse virtuelle à partir de laquelle le système (et uniquement le système) pourra l'identifier et la retrouver.

En résumé, dans le modèle de protection proposé ici, une capacité peut donner des droits sur un segment (capacité de couplage), sur un domaine (capacité d'appel) ou encore sur une liste de capacités. En définissant un domaine à partir d'une liste qui peut notamment contenir des capacités sur d'autres listes, on obtient par transitivité, non plus une définition par une liste plate, mais par un graphe de capacités. Une activité peut donc facilement donner à un domaine un sous–ensemble d'une liste de capacités si celle–ci est structurée en sous–listes adéquates.

De plus, avec les listes de capacités, on peut associer une liste à une application ou à un groupe d'utilisateurs, cette liste représentant les droits de protection partagés par les utilisateurs de l'application, ou les membres du groupe.

Les listes de capacités constituent donc l'outil privilégié pour l'administration des domaines de protection, et l'entité de partage entre ces domaines.

IV.1.5 Spécification et contrôle des droits échangés

IV.1.5.1 Situation du problème

Lors d'un changement de domaine, le code exécuté correspond à celui d'un appel de procédure, le même que si l'appel avait été local. Afin de transférer l'exécution de l'activité dans un nouveau domaine, le problème est d'une part de transférer les paramètres de la procédure (présents sur la pile) dans le domaine destination, et d'autre part de permettre le transfert des capacités éventuellement nécessaires. La spécification de la protection doit permettre de contrôler les capacités autorisées à entrer et à sortir d'un domaine de protection. Dans ce but, il apparaît nécessaire de disposer d'un langage de définition d'interface, appelé aussi IDL (*Interface Definition Language*), étendu par des clauses de définition de la protection. L'annexe A donne une description plus complète de ce langage. Il permet de définir des interfaces protégées, appelées dans la suite PPI (pour *«Protected Procedure Interface»*). Une PPI définit non seulement la signature d'une procédure, mais aussi pour chaque paramètre les règles de manipulation des capacités à appliquer lorsque cet appel de procédure met en œuvre un changement de domaine de protection. Cette PPI est attachée à la capacité de changement de domaine de protection.

Ainsi, lorsqu'une exception intervient sur un appel de procédure, et si le traitement de l'exception trouve une capacité d'appel associée à l'adresse provoquant l'exception, l'exécution de cet appel doit être transféré dans le nouveau domaine (notons que le même traitement doit être appliqué de façon symétrique au retour de l'appel), et la préparation de ce transfert d'exécution nécessite :

1. de récupérer la PPI associée à la capacité d'appel,
2. d'analyser cette PPI pour identifier les paramètres nécessitant le passage d'une capacité, et de localiser ces paramètres,
3. pour chacun des paramètres concernés, de rechercher la ou les capacités associées dans le domaine courant (le domaine appelant),
4. de vérifier que le type de la capacité et les droits qu'elle accorde sont compatibles avec ceux attendus et décrits dans la PPI,
5. de mettre en place la capacité attendue dans le domaine cible

La liste exhaustive des contrôles sur l'échange des capacités peut s'exprimer par le tableau suivant :

	pour le domaine appelant	pour le domaine appelé
à l'appel	droits offerts = capacités sortant	droits nécessaires = capacités entrant
au retour	droits nécessaires = capacités entrant	droits offerts = capacités sortant

Chacun des deux domaines impliqués dans cet échange de capacités soit également définir dans quelle liste les capacités reçues doivent être insérées, et dans quelle mesure les capacités confiées peuvent être à nouveau partagées et déléguées par le nouveau domaine. La mise en place d'une interface protégée s'effectue donc par la participation du client et du serveur, chacun spécifiant une partie des contraintes, selon le schéma suivant :

- le prestataire de service spécifie :
  - les capacités requises à l'aller
  - leur placement dans l'appelé
  - les capacités qu'il est prêt à donner au retour
- le client spécifie
  - s'il accepte de donner ces capacités à l'aller et si les capacités données en retour lui suffisent (ce qui revient à accepter la PPI du prestataire, et à insérer telle quelle la capacité d'appel fournie par le prestataire)
  - le placement des capacités en retour dans l'appelant

Lorsqu'un prestataire de service crée une capacité d'appel, il lui associe une PPI correspondant à son service. Un client peut alors a) accepter la capacité telle quelle ou b) copier la capacité d'appel pour surcharger la définition de la PPI (pour changer le placement des capacités reçues). Dans le cas a), c'est le serveur qui spécifie la règle de placement des capacités en retour.

Notons que cette gestion de la protection est indépendante de la compilation de l'application. Cette opération de spécification de la PPI est une opération d'administration/configuration de l'application.

IV.1.5.2 Solution proposée

La PPI utilisée permet de spécifier le type des paramètres passés. Lorsque ces paramètres sont de type pointeur, un formalisme va permettre de :

- dire si une capacité doit être passée
- dire de quel type de capacité il s'agit

  On permet seulement le passage de capacités de couplage de segment et de changement de domaine. On pourrait autoriser également le passage de capacités de listes. La PPI permettrait de spécifier qu'un paramètre est un identificateur de liste. Mais cette possibilité briserait le principe selon lequel le code qui déclenche un changement de domaine est indépendant de la protection.
- donner les attributs de la capacité

  Principalement pour les capacités de couplage des segments, il s'agit du mode de couplage en lecture, écriture et exécution.
- spécifier le placement des capacités

  La PPI pourra utiliser des variables d'environnement du domaine qui pourront être affectée par l'administrateur de la protection pour désigner des listes de capacités.

  Par défaut, une capacité reçue en paramètre est insérée dans une liste de capacités privée à l'activité. Cette liste n'existe que pour le temps de l'appel de procédure dans le domaine de protection appelé et elle est détruite à la sortie du domaine.

  Un ordre de placement provoque la copie de la capacité passée en paramètre dans la liste spécifiée. Si aucun ordre n'est donné, la capacité est retirée du domaine à la fin de la procédure ou à la mort de l'activité.

Considérons l'exemple de spécification de la PPI suivante :

```
procedure Biblio_Register (
    in capa_seg_read TDOC *doc
        install BIB_DOC_LIST,
    out String[10] refbib
    );

procedure Biblio_LookUp (
    in String[10] keywords,
    out capa_seg_read TLISTREF *listref
    );
```

Avec la première PPI, le domaine prestataire du service `Biblio_Register()`, a besoin d'une capacité en lecture sur le segment contenant *doc*. Le domaine client doit alors disposer au moins d'une capacité en lecture pour que l'appel puisse avoir lieu. Si tel est le cas, une capacité en lecture sur le segment contenant *doc* sera passée au domaine prestataire, et installée dans la liste spécifiée pas *BIB_DOC_LIST*. C'est ici le domaine prestataire et lui seul qui peut préciser ce placement.

Avec la deuxième PPI, une capacité en lecture sur un segment est retournée. Si le client accepte la capacité d'appel avec cette PPI, la capacité retournée sera déposée dans la liste temporaire de l'activité. Il peut toutefois copier la capacité d'appel et surcharger la PPI associée à la nouvelle capacité, pour préciser le placement de la capacité reçue, avec par exemple :

```
procedure Biblio_LookUp (
    in String[10] keywords,
    out capa_seg_read TLISTREF *listref
        install BIB_LIST_REF,
    );
```

La PPI peut également offrir la possibilité de spécifier le passage d'un ensemble de capacités associées au passage d'une structure de données complexe.

Par exemple, la PPI suivante permet de passer les capacités des segments désignés par le tableau passé en paramètre,

```
procedure proc (
    in (capa_seg_read char *)tab[10]
    );
```

et la PPI suivante permet de passer l'ensemble des capacités associées à une liste chaînée de chaînes de caractères :

```
type TLISTE =
record
    capa_seg_read char *string;
    TLISTE *next;
end;

procedure proc (
    in TLISTE l,
    );
```

Enfin, un dernier mécanisme visant à contrôler le passage de capacités en paramètre est envisagé : il s'agit de permettre dans une PPI de limiter la recherche d'une capacité à une liste. On peut ainsi restreindre les capacités pouvant sortir d'un domaine en associant à un passage de paramètre une liste (ou plutôt un arbre) de capacités.

Un exemple d'utilisation est l'interface d'un serveur d'impression de fichier. Le serveur exporte la capacité d'appel correspondant à la procédure protégée *Print (char *file)*, et le client peut restreindre la liste des fichiers pouvant être imprimés en spécifiant la liste des capacités pouvant être passées en paramètre. Ceci est réalisé par surcharge de la PPI dans le domaine d'où doivent sortir les capacités. Par exemple, avant la surcharge, le client voit la PPI suivante pour l'appel à Print() :

```
procedure Print (
    in capa_seg_read char *file
    );
```

En appliquant une surcharge, une nouvelle PPI est associée à l'appel :

```
procedure Print (
    in capa_seg_read char *file
        from PRINTABLE_FILE_LIST
    );
```

On peut souligner que cette fonction permet également de rendre plus efficace la recherche des capacités lors du passage de paramètre, car on restreint cette recherche.

IV.2 Architecture générale de la plate–forme Arias

IV.2.1 Organisation logicielle

L'objectif de cette présentation générale d'Arias, est de décrire les décisions prises pour la conception du prototype actuel de la plate–forme, sur des machines AIX 4.1, et de décrire l'ensemble des services offerts par cette réalisation. Cette présentation montre tout d'abord l'architecture d'Arias en terme de modules, puis les interactions entre ces modules, et la façon dont ils sont intégrés dans le noyau AIX des machines qui composent la plate–forme.

Les composants logiciels ou modules d'Arias, correspondent généralement aux différents services offerts par la plate–forme, à savoir : la gestion de la mémoire répartie, la gestion de la permanence et la gestion de la protection de cette mémoire.

L'exemple de la figure Fig. 4.2 donne une vision schématique et simplifiée de l'architecture de la plate–forme : les boîtes inférieures et grisées représentent les principaux composants d'Arias, les boîtes supérieures représentent des composants d'application qui utilisent les différents services de la plate–forme. Nous pouvons remarquer qu'une partie de ces composants se situent à l'intérieur du noyau : ce sont les parties qui permettent la personnalisation du système Arias et

son adaptation aux besoins spécifiques des applications, par exemple pour la cohérence de la mémoire ou la journalisation.

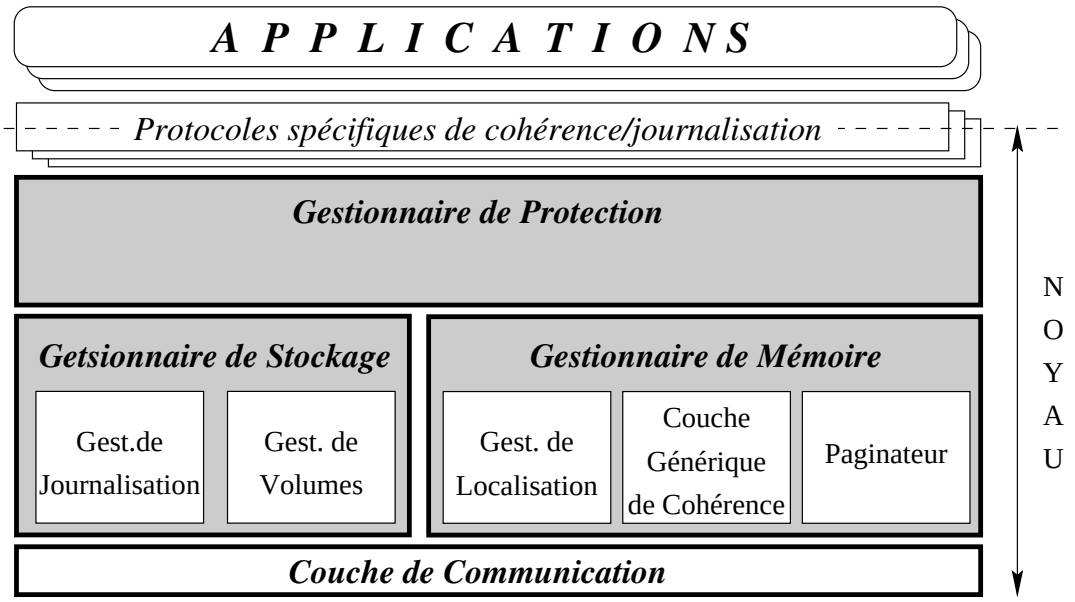


Fig. 4.2 : Architecture générale du service de cohérence et de synchronisation

Au plus bas niveau se situe la couche de communication d'Arias. Tous les autres composants utilisent cette couche pour effectuer des opérations à distance : elle permet d'échanger des messages aussi bien au niveau des composants entre eux, qu'au niveau des applications, que ce soit localement ou entre machines distantes. Cette couche fournit un service de noms permettant de désigner, soit des composants particuliers sur une machine (par exemple le gestionnaire de stockage est réparti, et utilise de tels messages pour assurer la synchronisation entre ses différents représentants locaux), soit les différentes applications qui utilisent la plate−forme Arias.

Au dessus de cette couche de communication se situent trois composants. Le plus important d'entre eux est le **Gestionnaire de mémoire**, car il constitue le cœur de la plate−forme Arias. Ce composant a la charge de permettre aux gestionnaires de mémoire virtuelle sur chacun des sites Arias d'accéder à l'espace d'adressage virtuel d'Arias. Il offre un service d'allocation de cet espace et la localisation des différentes copies des objets mémoire présents sur différents sites. Il se décompose en trois sous−modules qui coopèrent pour accomplir cette tâche :

- le **paginateur** qui agit directement avec le gestionnaire de mémoire virtuelle, et fournit une stratégie d'entrée/sortie pour traiter les requêtes de chargement/déchargement des pages de mémoire appartenant à l'espace virtuel Arias.

- le **gestionnaire de localisation** qui assure la localisation des descripteurs de segments. Il traite également l'allocation de mémoire.
- la **couche générique de cohérence** qui gère les descripteurs de segments contenant les informations de localisation des copies primaires des zones de mémoire utilisées dans ces segments.

Le **Gestionnaire de stockage** permet d'associer à un ensemble de segments une image permanente sur disque. C'est le rôle du sous–module appelé **Gestionnaire de volumes**. Ce module maintient le service d'un support de stockage réparti dont les tâches principales sont :

1. la gestion des correspondances entre les adresses en mémoire virtuelle et les adresses sur les disques
2. l'échange et le transfert de données entre le support de stockage permanent et la mémoire virtuelle.

Pour permettre à certaines applications une meilleure robustesse des segments de mémoire permanents, Arias comprend également un **Gestionnaire de journalisation**. Celui–ci permet des mises–à–jour atomiques des segments permanents, même si ces segments sont stockés et distribués sur des volumes répartis.

Enfin, Arias fournit pour sa mémoire virtuelle le service de protection réalisé pour cette étude.

IV.2.2 Mécanismes de communication sous–jacents

Le noyau de la plate–forme Arias réalise principalement un modèle de communication à base de messages. En effet, le mécanisme utilisé pour réaliser ce premier prototype d'Arias est le mécanisme des *STREAMS*. Les *STREAMS* constituent un ensemble d'outils pour le développement de services de communication, à travers une interface générale et souple. Les *STREAMS* définissent notamment une interface standard pour les opérations d'entrées–sorties sur des flots de caractères dans le noyau, et entre le noyau et le reste du système Unix. Dans la mesure où nous sommes amenés à intégrer de nouveaux services à un noyau Unix, c'est actuellement le seul mécanisme standard dont nous disposions. De plus, cette qualité de standard nous permet de faciliter l'intégration des différents composants de la plate–forme et d'espérer une facilité de portage du prototype sur des supports Unix différents d'AIX.

Pour comprendre l'intégration des différents modules qui réalisent les services de la plate–forme Arias, et leurs interactions, il est nécessaire de présenter ici l'interconnexion de ces modules et leur fonctionnement dans le contexte des *STREAMS*. Un Stream peut se définir comme un flot bidirectionnel (full–duplex) pour le transfert et le traitement de données. Ce flot de données est établi entre un processus dans

l'espace utilisateur et l'espace du noyau, comme le montre la figure Fig. 4.3. De plus, un Stream forme une pile qui est composée dans le noyau, d'une tête de Stream, d'un pilote, et éventuellement d'un ou plusieurs modules intermédiaires.

La tête de Stream est la partie la plus proche du processus utilisateur, et c'est elle qui traite tous les appels système effectués par un processus utilisateur à destination du Stream.

Le pilote de Stream peut être un pilote de périphérique ou un pilote logiciel. Il a la charge de transférer les données entre le noyau et le périphérique (réseau par exemple), sans autre traitement.

Un module de Stream incarne les fonctions de traitement des données qui transitent à travers le Stream, à la descente ou à la montée. Ces fonctions sont des routines de traitement des données qui transitent à la descente ou à la montée dans le Stream, sous la forme de messages. Chaque module fonctionne indépendamment des autres modules présents dans le Stream (à l'exception de ces deux voisins immédiats dont il reçoit des messages ou auxquels il en envoie).

La connexion des modules entre eux et la gestion des files de messages réalisant la connexion entre deux modules sont assurées par le mécanisme des *STREAMS*, et ne nécessitent aucune programmation supplémentaire dans le noyau.

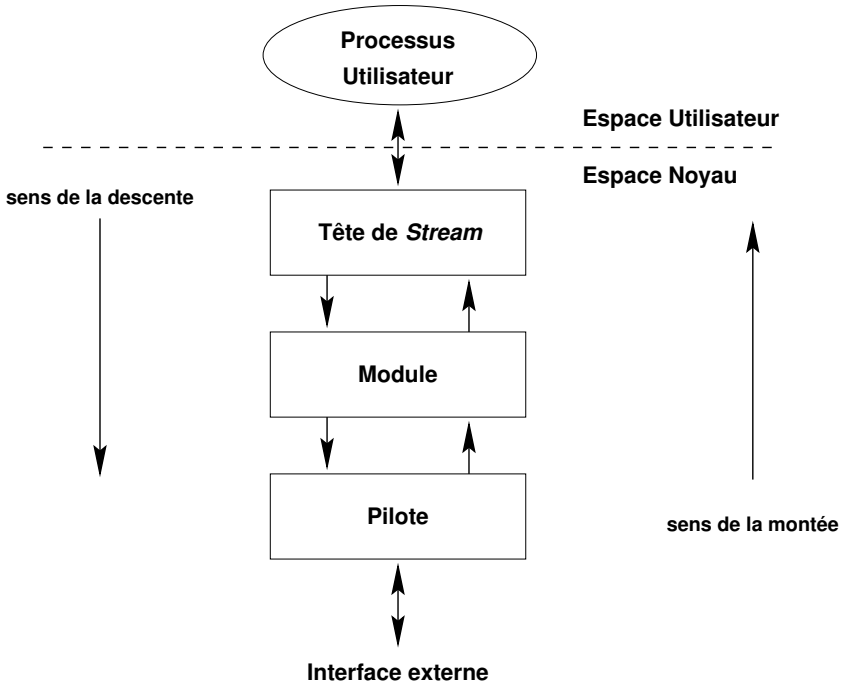


Fig. 4.3 : Organisation d'un Stream

Un processus utilisateur se connecte à un Stream par l'intermédiaire de l'appel système `open()` qui effectue le chargement dans le noyau et l'interconnexion des modules formant le Stream. Par la suite, le processus utilisateur peut s'adresser au pilote du Stream en lui envoyant des messages par l'appel système `write()` ou en recevant des messages par l'appel système `read()`, jusqu'à l'appel à `close()` qui déconnecte le Stream du processus utilisateur, démantèle les connexions entre modules et décharge ces modules.

Cette architecture modulaire est également conçue pour permettre aux applications de configurer la plate–forme en fonction de leurs besoins, et pour utiliser les services tout ou partie des services offerts. Cette caractéristique de la plate–forme impose de limiter les dépendances des modules entre eux.

La suite de ce chapitre décrit donc la mise en œuvre du service de protection proposé, et la façon dont il peut être intégré dans cette architecture.

IV.3 Réalisation du service

Le service de protection présente plusieurs aspects pour lesquels les choix de réalisation doivent tenir compte non seulement du modèle proposé au début de ce chapitre, mais également des objectifs généraux énoncés dans le précédent. Le premier aspect concerné est l'intégration des domaines de protection et des activités aux structures d'exécution Unix. Ensuite, il faut considérer les interactions entre le service de la protection et les autres services de gestion de la mémoire virtuelle partagée gérée par la plate–forme Arias. Enfin, le dernier aspect étudié aborde comment peut être réalisé le mécanisme de changement de domaine ainsi que le contrôle des droits échangés.

Par ailleurs, le choix de faire du service de protection un service «à la carte» sur la plate–forme Arias ne doit pas cacher la nécessité impérative d'intégrer ce service aux autres services de gestion des données réparties. En effet, faute d'une intégration suffisante, une utilisation non protégée des services de la plate–forme Arias pourrait aisément mettre en défaut les mécanismes de protection. De plus, l'adaptabilité impose que l'utilisation des services de la plate–forme dans une configuration non protégée reste identique dans une configuration protégée. C'est pourquoi doivent être étudiées d'une part les interactions avec les autres modules fonctionnels de la plate–forme, et décrits au début de ce chapitre, et d'autre part, les limites de l'utilisation «à la carte» du service de protection. L'aspect de l'intégration est donc présenté pour chacun des points étudiés ci–dessous où il intervient.

IV.3.1 Réalisation des domaines de protection

Comme nous l'avons vu précédemment, un domaine de protection est défini par une liste de capacités, et désigné par un identificateur unique. Dans la mesure où il constitue un espace d'adressage dans lequel peuvent s'exécuter plusieurs activités, il peut donc être représenté par un processus Unix. De plus, un domaine peut être réparti afin de profiter des possibilités de parallélisme des activités dans ce domaine. Un domaine peut donc être représenté par plusieurs processus Unix : un par machine où il est représenté.

Ainsi, le changement de site d'exécution pour une activité dans un domaine donné transfère l'activité (*thread*) dans le processus Unix représentant ce même domaine sur le site cible.

De même, le changement de domaine de protection transfère l'exécution de l'activité (*thread*) dans le processus Unix local qui représente le domaine cible sur le site courant. Ce mécanisme est détaillé plus loin (§ IV.3.5).

Dans les deux cas, le processus représentant un domaine peut être créé et initialisé s'il n'existe pas déjà. Le domaine de protection peut donc être soit dans un état *dormant*, ce qui signifie qu'aucun processus ne s'exécute pour ce domaine, soit être *actif* sur certains sites et être représenté par un processus Unix sur ces sites. Lorsqu'un changement de domaine intervient et si le domaine est dormant, le domaine est activé (création du processus) et il est initialisé avec la liste de capacités associée au domaine. Le changement de domaine peut alors être effectué comme dans le cas d'un changement vers un domaine déjà actif. Cette solution permet de gérer un nombre élevé de domaines de protection et de structures de données qui les définissent (c'est–à–dire les listes de capacités) sans pour cela devoir créer autant de processus et de structures d'exécution actives.

Le fait de permettre le partage de listes de capacités implique de gérer la cohérence de ces listes lorsqu'elles sont partagées entre des domaines sur des sites différents, ou partagées dans un domaine réparti sur plusieurs machines. Il faut donc étudier les politiques de gestion de ce partage des listes, notamment en fonction des fréquences de modification des listes. Cet aspect est décrit plus loin (§ IV.3.2).

La figure suivante donne une vision globale des domaines et des graphes de listes de capacités. Une capacité racine est associée à chaque domaine de protection et désigne un graphe de capacité. Des sous–graphes de capacités peuvent être partagés entre les domaines.

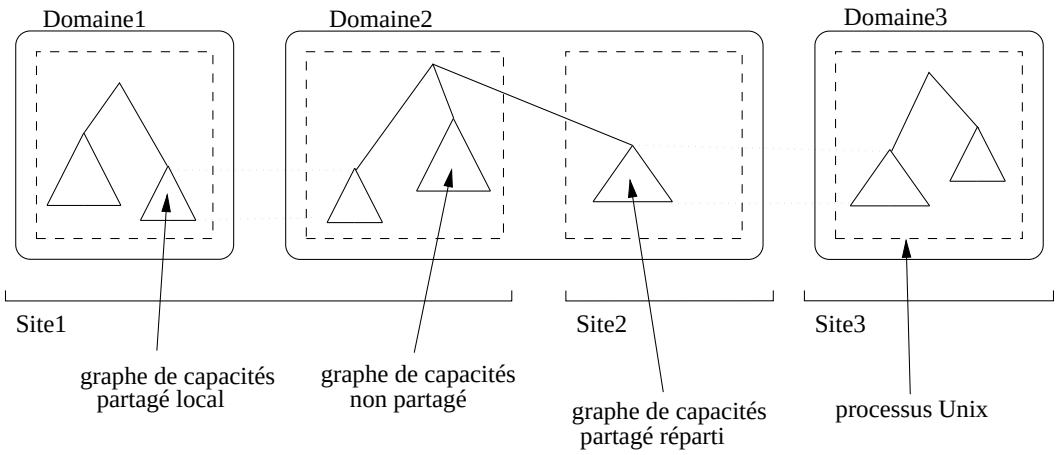


Fig. 4.4 : Réalisation des domaines de protection

Ces mécanismes de base peuvent être utilisés de différentes façons pour réaliser à la fois des serveurs ou des sous−systèmes protégés, ou des processus utilisateurs.

Pour faire le rapprochement avec le modèle Unix sur lequel se greffe Arias, l'activation d'un domaine consiste à créer un nouveau processus et à initialiser l'espace d'adressage avec la liste des capacités définissant le domaine. Pour changer de domaine, deux situations sont possibles outre le cas où le domaine est *dormant*, et où il suffit de l'activer comme nous venons de le voir. Si le domaine est déjà actif, il convient de choisir entre :

- la réception de la requête d'exécution par un processus serveur du domaine, qui crée une nouvelle activité dans le domaine. Un domaine comporte alors un processus Unix serveur par site où il est représenté, et éventuellement d'autres processus créés à l'initiative du processus serveur, ou de l'utilisateur (car dans un souci de généralité, l'utilisation d'appels système Unix tel que `fork()` doit rester licite et garder sa sémantique).
- la création d'un nouveau processus Unix, comme dans le cas de l'activation d'un domaine, qui est initialisé avec la liste des capacités constituant le domaine. Outre sa lourdeur, cette solution pose toutefois un problème majeur, car elle restreint les modèles d'exécution possibles pour un utilisateur des domaines Arias. Par exemple un modèle d'exécution client−serveur basé sur les domaines de protection devient inaccessible, tandis que la première solution reste plus générique.

Aussi, c'est la première solution qui est retenue dans le premier prototype de la plate−forme : un domaine de protection dispose donc, sur chaque site où il est actif d'un processus serveur le représentant, et chargé de répondre aux requêtes

d'exécution dans ce domaine. Il apparaît immédiatement que la mise en place d'un serveur protégé de requêtes d'exécution nécessite la mise en place d'une vérification préalable des requêtes et de l'authentification des domaines dont elles proviennent, notamment lorsqu'une activité effectue un changement de site. Une méthode d'authentification des domaines distants (appelant et appelé) doit donc être mise en place, et il faut offrir une interface distincte pour les changements de domaine locaux (effectués ainsi par défaut) et les changements de domaine à distance. La discussion du choix d'une technique d'authentification n'entre pas dans le contexte de cette étude, mais il convient néanmoins de ne pas négliger cet aspect d'authentification qui intervient également dans la gestion des listes de capacités.

IV.3.2 Gestion des listes de capacités

Par ailleurs, le fait de permettre le partage de listes de capacités implique de gérer la cohérence de ces listes lorsqu'elles sont partagées entre des domaines sur des sites différents, ou partagées dans un domaine réparti sur plusieurs machines. Ainsi, lorsqu'une liste de capacités est modifiée, cette modification doit être répercutée sur toutes les copies de cette liste qui peuvent être réparties sur les différents sites. De cette façon, l'administration des domaines de protection et la gestion des listes de capacités constituent une application de gestion de données réparties partagées. Il est possible de mettre en œuvre des mécanismes de mémoire partagée particuliers pour cette gestion des listes partagées, mais il peut être avantageux de pouvoir disposer directement des services du gestionnaire de cohérence de la plate–forme Arias, et de réaliser ainsi cette partie du service de la protection comme une application du service de données partagées. Cependant il est très délicat de privilégier, a priori, un protocole de cohérence particulier pour la gestion des listes de capacités : en effet, le caractère dynamique de la configuration de protection des applications coopératives ne permet pas d'isoler un schéma particulier d'accès aux listes de capacités. Pour cette raison, il est donc également intéressant de pouvoir gérer la cohérence des listes de capacités comme une application du service de gestion des données partagées, car il est alors possible de changer simplement le protocole de cohérence mis en œuvre pour la gestion des listes, à la configuration de la plate–forme Arias, et sans devoir modifier les modules réalisés pour le service de la protection.

Il est important de signaler ici que la mise en cohérence des différentes copies des listes de capacités doit être consolidée :

- soit par une méthode de chiffrement des messages échangés entre sites distants et contenant des informations de protection. Cette solution doit être intégrée dans le protocole de gestion de la cohérence utilisé pour ces listes.

- soit par une méthode d'authentification des modules de gestion des capacités. Cette méthode peut être mise en œuvre indépendamment du protocole de gestion de la cohérence. Elle est donc préférable à la précédente.

IV.3.3 Recherche des capacités

La recherche d'une capacité dans un domaine peut avoir lieu en deux circonstances :

- lors d'une faute d'adressage sur un segment

  La capacité est recherchée dans le graphe de capacités du domaine. Pour le couplage d'un segment en lecture, on parcourra l'arbre jusqu'à ce que l'on trouve une capacité donnant droit de coupler en lecture. Si cette capacité donne le droit de coupler en lecture et en écriture, alors le segment sera couplé en lecture et en écriture. Pour le droit d'exécution, la situation est légèrement différente : on parcourt l'arbre jusqu'à rencontrer une capacité donnant droit d'exécution, et lors du parcours on mémorise les capacités de domaine pour l'adresse fautive. Si on ne trouve pas de capacité pour coupler avec droit d'exécution, alors il faut utiliser une capacité de domaine.
- lors du passage de paramètre sur une procédure protégée. Cet aspect est repris dans la section IV.3.6.

Outre l'efficacité de cette recherche dans le graphe des capacités, les propriétés suivantes apparaissent nécessaires :

- la recherche de capacités dans un graphe donné doit être déterministe, notamment dans le cas où un graphe comporte des capacités contradictoires pour une adresse donnée, par exemple, capacité sur le segment avec droit d'exécution et capacité d'appel. Certains cas de figures peuvent être résolus simplement. Par exemple, en présence de deux capacités de couplage dans le graphe, il suffit de considérer l'union des droits accordés par ces capacités pour résoudre le conflit. En présence d'une capacité de couplage et d'une capacité d'appel, il n'y a vraiment de conflit que dans le cas d'un accès en exécution, et le choix de favoriser le couplage et l'exécution localement dans le domaine courant semble naturel. Notons que ce choix n'est pas incompatible avec l'opération de restriction de droits dans un domaine de protection, dans la mesure où celle–ci peut être réalisée en détruisant la capacité de couplage en exécution dans la liste où elle se trouve ou bien en l'écrasant par la capacité d'appel.

Résolution des conflits potentiels	Capacité de couplage	Capacité d'appel

Résolution des conflits potentiels	Capacité de couplage	Capacité d'appel
Capacité de couplage	union des droits	couplage
Capacité d'appel	couplage	?

Ainsi, le véritable problème est de savoir s'il est possible de fournir une politique déterministe et générique pour décider du choix d'une capacité lors du traitement d'une recherche de capacités, ou bien s'il faut laisser ce choix à l'application (comme cela est possible dans le cadre du projet Mungi). L'objectif d'adaptabilité de la plate–forme Arias impose, quoi qu'il en soit, de disposer d'une politique par défaut, et satisfaisante. Comme l'indique le tableau récapitulatif ci–dessus, le seul conflit réel provient de la présence de deux capacités d'appel pour une même adresse. Dans ce cas, le conflit peut avoir deux origines. La première est l'identificateur du domaine cible, et la seconde est la spécification des règles de transfert des paramètres. L'une et l'autre sont insolubles pour le système et sont donc considérées comme des erreurs de configuration de la protection. Cet aspect est repris plus loin, à l'occasion de la discussion sur le contrôle des droits échangés (§ IV.3.6).

- enfin, la gestion et la recherche doivent également prendre en compte la répartition d'un domaine, et assurer la cohérence des graphes de capacités sur les différents sites.

En conclusion, la recherche est fortement liée à la réalisation qui en est faite ; deux solutions sont envisageables : la première solution consiste à utiliser un algorithme de recherche connu de l'utilisateur et à fournir simultanément à l'utilisateur une interface lui permettant de placer les capacités dans le graphe des listes conformément au résultat qu'il attend pour les recherches ultérieures. Cette approche nécessite donc la mise en place d'une désignation et d'une structure hiérarchique des listes de capacités, établies une fois pour toutes. Mais ce choix est incompatible avec les objectifs d'adaptabilité et de modèle de protection non hiérarchique. Il faut donc étudier une deuxième solution dans laquelle la recherche de capacités retrouve la liste exhaustive des capacités associées à une adresse, mais en effectue une synthèse en une seule et unique capacité équivalente. Par exemple, dans le cas de capacités multiples pour un segment, elle rendra une capacité de segment équivalente avec l'ensemble des droits accordés. Cette capacité équivalente doit bien évidemment être maintenue à jour, au fur et à mesure de l'évolution du graphe de listes de capacités définissant le domaine.

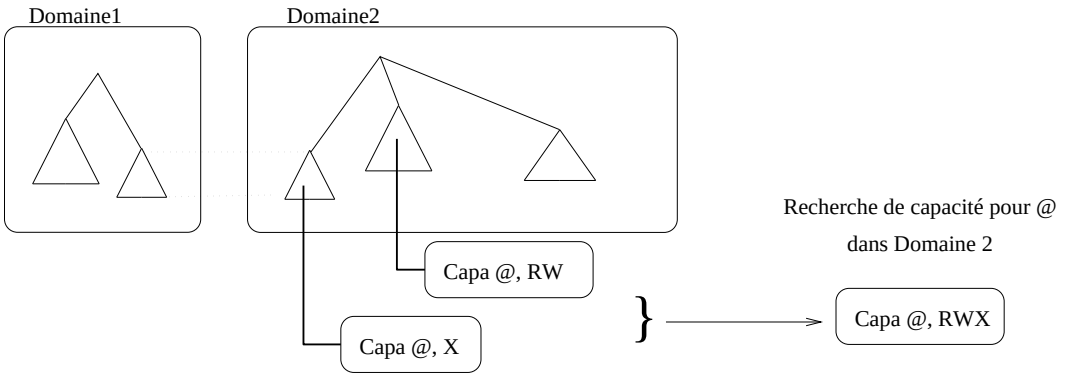


Fig. 4.5 : Synthèse de capacités

Ainsi, la gestion des capacités utilise deux structures de données, illustrées sur la figure Fig. 4.6.

La première de ces structures (représentée à gauche sur la figure) est le graphe des listes des capacités L'organisation et le contenu de ce graphe sont entièrement sous le contrôle de l'utilisateur qui le manipule à travers l'interface d'administration des capacités offerte par le service de protection. Cette structure de données est répartie, et contient des capacités de couplage, des capacités d'appel et des capacités sur des listes, dont il convient donc d'assurer la cohérence sur les différents sites de la plate−forme. La taille et la profondeur de ce graphe est illimitée, aussi, il ne serait pas raisonnable d'y effectuer une recherche.

L'autre structure de données (représentée à droite sur la figure) sert d'accélérateur pour la recherche de capacités, notamment lors d'une exception. Ainsi, cette seconde structure est réalisée par un arbre de recherche pour chaque domaine actif sur un site. Dans cet arbre, les capacités sont classées selon les adresses virtuelles auxquelles elles s'appliquent (adresse d'un segment ou point d'entrée d'un changement de domaine, car la notion de capacités sur des listes devient inutile ici).

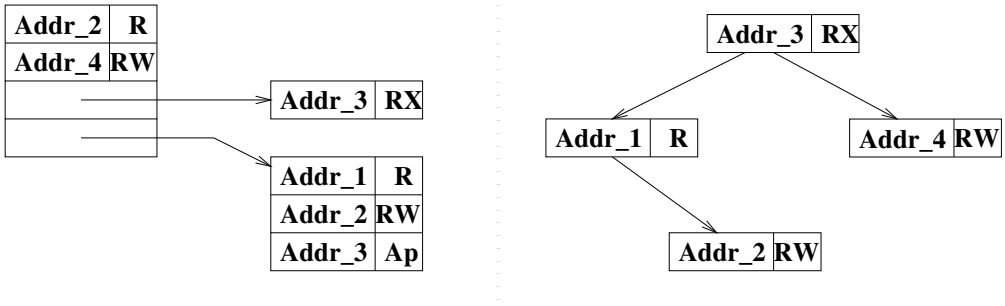


Fig. 4.6 : Graphe de listes et arbre de recherche

De plus, pour obtenir des performances acceptables lors d'une recherche, cet arbre est disponible localement sur chaque site où le domaine est actif. En contrepartie, ce choix rend plus complexe le maintien à jour de cet arbre au fur et à mesure des modifications effectuées dans le graphe de listes.

Interaction avec le gestionnaire de cohérence

Comme nous l'avons vu précédemment, les listes de capacités sont des structures de données réparties et partagées dont il convient de maintenir la cohérence des différentes copies utilisées sur les différentes sites de la plate−forme Arias. Cette mise en cohérence et synchronisation intervient donc pour les deux représentations présentées sur la figure Fig. 4.6 :

- d'abord, lorsqu'une liste de capacités est modifiée, cette modification doit être répercutée sur toutes les copies de cette liste. Cette mise à jour fait partie de la gestion des listes de capacités présentée précédemment.
- ensuite, la modification d'une liste doit également être répercutée sur tous les arbres de recherche qui synthétisent des informations de protection de cette liste. Pour cela, il convient donc d'abord de connaître, pour chaque liste de capacités, les domaines de protection qui la partagent. Et pour chacun de ces domaines, il est nécessaire de modifier en conséquence l'arbre de recherche sur chaque site où le domaine est actif.

Interaction avec le paginateur

Le paginateur est la clé de voûte du service de la protection au sein de la plate−forme. En effet, dans la mesure où les capacités doivent demeurer cachées et uniquement traitées au moment d'une exception pour défaut de segment ou faute de protection, c'est le paginateur qui devra intégrer, s'il y a lieu, la recherche de capacités dans son traitement de ces exceptions.

Outre la recherche de capacités détaillée précédemment, le traitement d'une exception, que ce soit un défaut de segment ou une violation de protection, doit effectuer les opérations suivantes et dans cet ordre :

- avant toute autre chose, le traitement de l'exception doit récupérer le contexte de l'activité qui a produit cette exception, afin de savoir s'il s'agit d'une activité Arias, ou d'un simple processus Unix. S'il s'agit d'une activité Arias, son contexte doit également permettre de savoir si le service de protection est impliqué ou non dans la configuration dans laquelle s'exécute cette activité, et si tel est le cas, d'identifier le domaine de protection dans lequel l'activité s'exécute.

- le traitement de l'exception doit également recourir au service du gestionnaire de localisation pour déterminer si l'adresse qui est à l'origine de l'exception, est une adresse de la mémoire virtuelle gérée par Arias ou non, et si c'est le cas, alors ce service déterminera, entre autres choses, l'adresse du segment de mémoire virtuelle concerné. Dans la suite, désignons par $\texttt{addr}_{\text{except}}$ l'adresse à l'origine de l'exception, et par $\texttt{addr}_{\text{seg}}$ l'adresse du segment correspondant.
- si le service de protection est impliqué, le traitement doit demander à ce service de fournir une capacité de couplage pour $\texttt{addr}_{\text{seg}}$, et vérifier si le mode d'accès est compatible avec les droits accordés par cette capacité. S'il n'y a pas de capacité adéquate, mais que l'exception correspond à une tentative d'accès en exécution, alors il convient de rechercher également une capacité d'appel pour $\texttt{addr}_{\text{except}}$ et d'effectuer le changement de domaine correspondant selon les mécanismes décrits plus loin. L'exception est alors résolue.
- le traitement doit effectuer le couplage du segment correspondant à $\texttt{addr}_{\text{seg}}$, soit sans restriction de droits de protection, si le service n'est pas sollicité, soit avec les droits indiqués par la capacité de couplage délivrée pour $\texttt{addr}_{\text{seg}}$.

Interaction avec le gestionnaire de localisation

Lors d'une recherche de capacités, l'interaction avec le gestionnaire de protection consiste, comme nous l'avons vu ci–dessus, à identifier et à localiser le segment de la mémoire répartie qui englobe une adresse virtuelle donnée (sous réserve d'existence d'un tel segment). Ainsi, cette fonction du gestionnaire de localisation permet, lors du traitement d'une exception, d'abord de connaître si la mémoire répartie la plate–forme Arias est impliquée ou non, ensuite de retrouver l'adresse du segment correspondant. Cette localisation est réalisée systématiquement à chaque traitement d'exception. Il est donc suffisant de disposer de l'adresse des segments comme critère de recherche, et il n'est pas nécessaire de conserver la taille d'un segment dans les capacités de couplage.

IV.3.4 Création et destruction des segments

Si le service de protection fait partie de la configuration de la plate–forme Arias, alors il doit intervenir lors de la création et de la destruction d'un segment de la mémoire répartie partagée, pour respectivement créer ou détruire la (ou les) capacités correspondant à ce segment dans le domaine de protection courant.

De plus, pour respecter nos objectifs d'invisibilité des capacités, l'interface de création et de destruction d'un segment doit rester identique dans une configuration de la plate–forme avec le service de la protection, et dans une configuration non protégée. Aussi, lorsqu'un utilisateur ou une application effectue une requête de création de segment, il n'est pas concevable qu'il doive fournir les droits de protection désirés sur ce nouveau segment. Il est donc impératif de créer une capacité de couplage, accordant tous les droits possibles sur un segment, pour en permettre l'utilisation dans le domaine de protection où il est créé. Ces droits peuvent être cependant affinés et restreints ultérieurement lors d'une opération d'administration des capacités.

Ainsi le traitement que le service de la protection doit effectuer lors de la création et la destruction d'un segment, doit s'intégrer complètement avec le gestionnaire de localisation qui fournit l'interface de cette création ou destruction, quelle que ce soit la configuration de la plate–forme Arias.

Interaction avec le gestionnaire de localisation

L'architecture modulaire de la plate–forme et l'utilisation des messages STREAMS permet au module réalisant la gestion des capacités d'intercepter les messages de requête de création et de destruction de segments, d'effectuer le traitement correspondant, et de faire suivre ces messages jusqu'au gestionnaire de localisation. La figure Fig. 4.7 illustre cette interposition du module de gestion des capacités entre l'activité qui crée (ou détruit) un segment, et le gestionnaire de localisation qui répond à cette requête de création (ou de destruction).

Lors de la création d'un segment, l'interception du message de requête à sa descente permet de préparer l'ajout d'une capacité accordant les droits maximum pour ce segment dans la liste de capacités du domaine de l'activité, puis de faire suivre la requête au gestionnaire de localisation. Enfin, lorsque le gestionnaire a traité la requête, l'interception du message de réponse à la montée permet de valider (ou d'invalider) l'ajout de la nouvelle capacité.

Lors d'une destruction de segment, le gestionnaire de localisation assure que le segment est détruit immédiatement si aucune application ne le couple, ou dès que les applications qui couplent ce segment le libèrent. Et dans les deux cas, toute requête de localisation qui arrive après la requête de

destruction est rejetée, puisque l'on ne réutilise pas les adresses virtuelles pour l'allocation des segments. Quant au gestionnaire de protection, la destruction des capacités correspondant à ce segment nécessite le parcours intégral du graphe de listes du domaine courant, et de tous les autres domaines (il faut précisément parcourir toutes les listes de capacités). C'est une opération excessivement lourde. Toutefois l'algorithme utilisé pour le traitement d'une exception fait d'abord appel au gestionnaire de localisation pour connaître si un segment est géré par le service de données partagées de la plate–forme Arias, ou pas. Dans le cas d'un segment détruit, cette requête d'identification auprès du gestionnaire de localisation échoue, et aucune consultation du service de protection n'est en fait nécessaire. Il est donc possible ici de tirer avantage de l'indépendance des deux mécanismes, et de repousser la destruction des capacités sur des segments détruits, jusqu'à la réalisation d'un glanage des capacités obsolètes dans l'ensemble des listes.

Cette opération de glanage fait coopérer le service de localisation et le service de protection pour permettre de ramasser les capacités obsolètes désignant des segments détruits d'une part, et les segments qui ne sont plus désignés par aucune capacité, d'autre part.

Hormis l'opération de glanage, le module de gestion des capacités est donc transparent vis–à–vis des requêtes de création ou de destruction de segments.

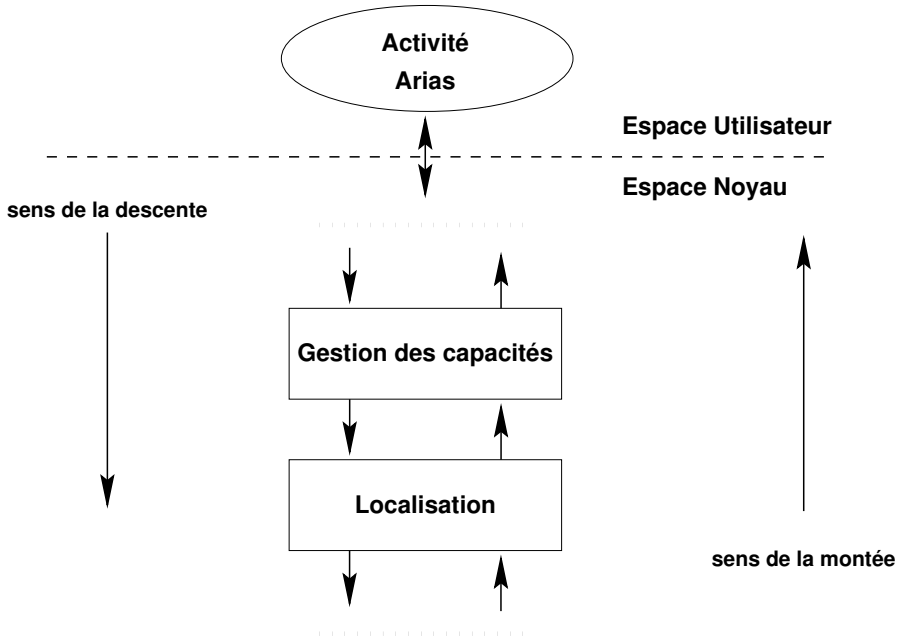


Fig. 4.7 : Intégration de la localisation et de la protection dans l'architecture STREAMS d'Arias

Interaction avec le gestionnaire de stockage

L'interaction du service de la protection avec le gestionnaire de stockage intervient lorsque le service de permanence de ce gestionnaire de stockage est mis en œuvre pour des segments partagés et protégés. En effet, il convient alors de garantir également la permanence de toutes les capacités qui désignent un segment permanent. De même, il faut assurer la permanence des listes qui contiennent ces capacités, et des domaines qui partagent ces listes. Toutefois, même si notre objectif est d'éviter qu'une application ne paie le coût d'un service qu'elle n'utilise pas, l'identification des capacités, des listes et des domaines permanents, et de ceux qui ne le sont pas, pose un problème de choix qui n'appartient pas au service de la protection. De plus la complexité de ce problème croît de manière exponentielle avec le degré de partage ou de recopie des capacités, et le degré de partage des listes. Dès lors que le gestionnaire de stockage est sollicité dans une configuration de la plate–forme Arias, il semble donc plus raisonnable, voire indispensable, d'assurer la permanence systématique des informations de protection. Il faut toutefois distinguer ici les graphes de listes qui définissent les domaines de protection et qui sont donc potentiellement permanents, des arbres de recherche de capacités dont la permanence n'est pas requise dans la mesure où ces arbres peuvent être régénérés à tout moment à partir des graphes de listes.

Enfin, un glanage coopératif des miettes, tel que celui−ci que nous avons vu avec le gestionnaire de localisation, permet d'éliminer les capacités et les listes obsolètes du support de stockage permanent. Cette opération constitue une première application répartie coopérative interne ; elle fait intervenir l'ensemble des services de la plate−forme, mais dont sa description sort du contexte de cette étude.

IV.3.5 Changement de domaine : réalisation de l'appel

Les objectifs de capacités implicites et d'efficacité s'appliquent aux capacités d'appel et à l'ensemble des mécanismes de changements de domaine de la même façon que pour les capacités de couplage.

La réalisation des domaines de protection proposée ne fait intervenir que des changements de domaines sur des appels de procédure. Les mécanismes de changement de domaine peuvent donc être réalisés suivant les principes des appels de procédure à distance introduits par Birrel et Nelson [Birrel 84]. De tels mécanismes sont mis en œuvre dans le projet Mungi par exemple. De plus, le contexte de la plate−forme Arias permet de profiter de plusieurs optimisations pour ce mécanisme. En effet, puisque la répartition des domaines et la séparation des domaines sont présentées comme deux services distincts et indépendants, le changement de domaine s'effectue ici toujours localement sur un site d'exécution. De plus, il est possible de bénéficier du service de la mémoire partagée pour accélérer les communications et les échanges de données entre les deux domaines locaux. Plusieurs travaux de recherche ont déjà été menés en ce sens. Les plus connus de ces travaux ont été publiés sous le nom de *Lightweight RPC* (LRPC [Bershad 90]), et ce mécanisme a été adopté par d'autres systèmes comme le système Spring [Kougiouris 94] où ce mécanisme assure le service d'appels de méthode d'un espace d'adressage vers un autre. L'étude menée par Bryce et Muller [Bryce 95] dans le cadre du système Mach offre également une approche intéressante et détaillée pour la réalisation des domaines de protection et des changements de domaine.

Rappelons ici brièvement le principe de ces mécanismes appliqués au service de protection. En suivant la définition du LRPC sur l'exemple de la figure Fig. 4.8, une procédure est représentée par son talon dans le domaine du client et son talon dans le domaine du serveur. Lorsqu'une activité (représentée par un flot) appelle une procédure (étape 1), le talon client attribue à cet appel une pile en mémoire partagée parmi un ensemble de piles pré−allouées (étape 2) pour transférer les paramètres du domaine client vers le domaine serveur, et effectue l'appel système correspondant au LRPC. Cet appel système est responsable notamment de vérifier et de valider d'une part l'identité du domaine appelant et grâce à la capacité d'appel associée à la procédure, celle du domaine appelé d'autre part. Connaissant le domaine appelé, le

traitement du changement de domaine consiste alors à sélectionner une pile d'exécution dans le domaine serveur parmi un ensemble de piles pré–allouées (étape 3). Rappelons qu'ici le processus représentant local du domaine serveur est créé s'il n'existe pas déjà. Ensuite, le changement de domaine est effectué par une simple commutation de la pile du flot représentant l'activité, et du processus représentant le domaine dans lequel l'activité s'exécute (étape 4).

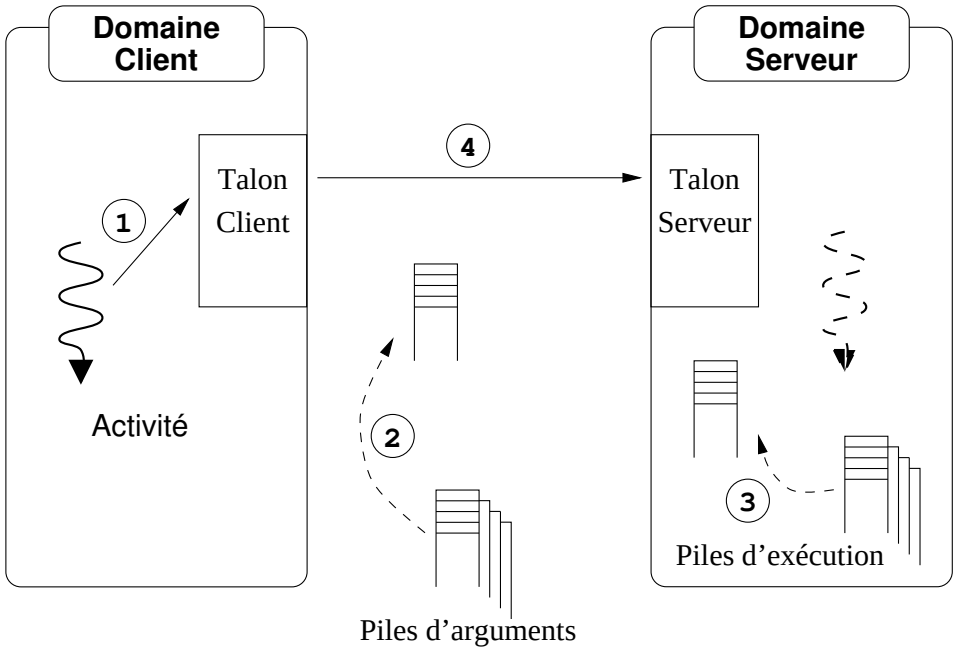


Fig. 4.8 : Transposition du LRPC au changement de domaine

Mais faute de disposer d'un accès au noyau d'AIX pour manipuler les structures d'un flot d'exécution, et notamment changer son contexte d'exécution, il est hélas impossible ici de bénéficier complètement des optimisations apportées par ces mécanismes. Nous devons nous contenter ici d'une mise en œuvre qui au lieu de commuter l'exécution du flot de l'activité appelante dans le domaine serveur sur une nouvelle pile, bloque le flot appelant et commute l'exécution sur un nouveau flot sélectionné parmi un ensemble de flots pré–alloués dans le domaine serveur.

Par ailleurs, le changement de domaine est mis en œuvre à la suite d'un défaut de segment et lors du traitement de cette exception. Le flot de l'activité est donc dérouté avant d'avoir empilé les paramètres et procédé à l'étape (1). Mais avant de pouvoir évaluer les conséquences de ce déroutement prématuré, il convient d'étudier la façon dont est réalisé le contrôle des droits échangés avec les paramètres de l'appel.

IV.3.6 Changement de domaine : contrôle des droits échangés

Le mécanisme de contrôle des droits échangés est réalisé selon les spécifications de la PPI associée à la capacité d'appel d'une procédure protégée.

Cette spécification d'interface permet de générer d'une part un talon pour le domaine client où une activité fait appel à la procédure protégée, et un talon pour le domaine serveur d'autre part.

Le rôle du talon de la procédure protégée, côté domaine client, est bien sûr de transférer les paramètres de la procédure, mais aussi de transférer les définitions des capacités nécessaires pour les paramètres à l'aller. Pour chaque capacité à transférer, cette définition comprend l'adresse virtuelle concernée, le type de capacité nécessaire et les droits que cette capacité doit accorder. Éventuellement cette définition désigne également la liste dans laquelle cette capacité doit être recherchée dans le domaine client, ou la liste dans laquelle cette capacité doit être copiée dans le domaine serveur.

L'ensemble de ces informations sont ensuite transmises au service de protection pour qu'il puisse traiter la spécification d'interface de la procédure protégée. Pour chaque paramètre (adresse virtuelle) nécessitant le passage d'une capacité, le service de protection doit alors :

- effectuer la recherche des capacités correspondant à ces adresses virtuelles, suivant le même procédé utilisé lors des défauts de segments (§ IV.3.3), c'est–à–dire : localiser le segment concerné par une adresse virtuelle, et rechercher soit dans l'arbre de recherche du domaine client, soit dans la liste de capacités si elle est spécifiée, les capacités de couplage pour ce segment et les capacités d'appel pour cette adresse virtuelle,
- vérifier que ces capacités sont compatibles avec le type de capacités et les droits spécifiés par l'interface. La spécification d'une capacité de couplage nécessite de trouver une capacité de couplage comprenant au moins les droits spécifiés. Quant à la spécification d'une capacité d'appel, elle requiert soit une capacité d'appel, soit une capacité de couplage. Ces vérifications sont absolument identiques à celles qui doivent être effectuées lors d'une opération de copie de capacités,
- transmettre et recopier les capacités spécifiées par l'interface dans le domaine appelé, soit dans une liste spécifique à cet appel qui sera détruite au retour, soit dans les listes éventuellement spécifiées pour chaque capacité,
- déclencher le changement effectif de domaine, et relancer l'exécution de l'activité en activant le flot sélectionné dans le nouveau domaine pour effectuer le traitement du talon de ce côté.

Le rôle du talon de la procédure protégée, côté domaine serveur, consiste donc à initialiser le flot d'exécution qui représente l'activité dans le processus représentant le domaine serveur, et à appeler la procédure protégée effective en lui passant les paramètres de l'appel. Au retour de cette procédure, le talon doit appliquer le même traitement que le talon client à l'appel, pour les paramètres de retour.

IV.4 Conclusion

La mise en œuvre du service de protection a donc permis de mettre en évidence les aspects sensibles dans la conception et la réalisation de ce service pour respecter les objectifs de généralité et d'adaptabilité de la plate–forme ainsi que les principes du service de protection proposé.

La seule difficulté de conception réside dans l'intégration du service de la protection avec les autres services offerts par la plate–forme pour la gestion des données partagées, tout en respectant l'indépendance de ces services pour que la plate–forme soit configurable.

Comme nous l'avons vu pour chaque aspect, l'architecture globale choisie pour la plate–forme, basée sur le modèle d'un flot de messages et de modules de traitement des STREAMS, réalise ces objectifs d'elle–même.

Par ailleurs, le seul obstacle rencontré dans cette mise en œuvre est celui de la réalisation du LRPC, non pour des raisons techniques, mais seulement pour des raisons d'interface et de visibilité des structures internes du noyau d'exécution des machines d'expérimentation.

Chapitre V

Évaluation

Ce chapitre présente une évaluation du service de la protection réalisé sur la plate–forme Arias, dans son état à la fin du premier semestre 1996.

Le prototype de cette plate–forme permet au terme de ce semestre :

- d'utiliser un service de localisation pour les segments de la mémoire virtuelle répartie,
- d'utiliser éventuellement une couche générique de cohérence pour mettre en œuvre des protocoles spécifiques pour assurer la cohérence et la synchronisation de zones de mémoire partagées à l'intérieur des segments,
- d'utiliser éventuellement un service de protection pour garantir un contrôle d'accès sélectif aux segments, suivant le domaine de protection courant.

Grâce à cette réalisation, l'évaluation qui suit présente d'abord une étude qualitative générale du service de protection proposé, puis une étude qualitative de chaque aspect de la mise en œuvre de ce service. De plus, les mesures effectuées sur cette réalisation nous permettent une étude quantitative pour évaluer la viabilité du service de protection.

V.1 Modèle de protection

Le modèle de protection basé sur des capacités confinées peut être jugé satisfaisant dans le contexte de la plate–forme Arias, dans la mesure où :

- la désignation reste uniforme sur l'ensemble des sites qui mettent en œuvre une instance de la mémoire virtuelle partagée d'Arias,
- la désignation reste indépendante des mécanismes de protection, ce qui permet aux applications d'utiliser de la même façon la plate–forme quelle que soit sa configuration, avec ou sans protection,
- l'authentification est clairement séparée du contrôle d'accès à la mémoire partagée, ce qui permet, d'une part, de conserver ce contrôle d'accès indépendant des autres services lorsqu'il s'agit d'authentifier les utilisateurs de

l'ensemble des services de la plate–forme, et de configurer indépendamment les droits d'accès et les méthodes d'authentification d'autre part.

V.2 Réalisation du service

V.2.1 Manipulation des segments

Les interfaces de manipulation des segments, de gestion de la cohérence et de la synchronisation et l'adressage de la mémoire virtuelle répartie sont identiques dans une configuration de la plate–forme sans protection, et dans une configuration avec protection. Ainsi la gestion du contrôle d'accès aux segments de la mémoire virtuelle est complètement invisible pour les applications. L'objectif de garder une désignation implicite par capacités est donc réalisé.

Il n'y a pas eu d'application réelle développée sur le prototype, mais le développement d'applications de tests a permis de vérifier cette utilisation implicite de capacités. Ces applications de tests permettent notamment :

- de créer une configuration initiale de segments persistants dans la mémoire virtuelle répartie lorsque la plate–forme n'est pas configurée pour offrir le service de permanence,
- de tester les mécanismes de pagination par le parcours de la mémoire virtuelle contenue dans ces segments.

L'exécution de ces applications de tests peut se faire indifféremment dans une configuration où la plate–forme Arias met en œuvre le service de la protection, ou dans une configuration où la plate–forme ignore ce service, sans nécessiter aucune nouvelle compilation.

V.2.2 Changement de domaine

Le modèle proposé pour le changement de domaine comporte deux aspects. Le premier est le mécanisme sous–jacent qui met en œuvre ce modèle, c'est–à–dire essentiellement un mécanisme d'appel de procédure à distance. Le deuxième aspect qui nous intéresse ici, est le support pour un contrôle intégré des droits transférés lors d'un changement de domaine.

Le mécanisme de changement de domaine, géré sur une exception pour défaut de segment, permet d'effectuer une liaison implicite de la procédure appelée dans un autre domaine de protection. Ainsi l'objectif de cacher les capacités à l'utilisateur est également atteint pour les capacités d'appel.

Par contre, dans la mesure où ce mécanisme de changement de domaine s'avère très coûteux, il est nécessaire de s'attacher à en améliorer les performances. Le coût d'un appel de procédure suivant le modèle du RPC repose essentiellement sur deux origines :

- les allers–retours entre l'espace utilisateur et l'espace noyau pour assurer la gestion des paramètres et leur transmission d'un espace d'adressage (domaine) à l'autre,
- la commutation de contexte pour effectuer le transfert effectif de l'exécution d'un espace d'adressage (domaine) à l'autre.

C'est pourquoi nous nous intéressons à faire bénéficier le mécanisme de changement de domaine des optimisations proposées pour le LRPC. En effet, celles–ci s'appliquent précisément dans le cas où :

- un appel de procédure à distance est effectué localement,
- les communications peuvent être effectuées par mémoire partagée.

Mais la réalisation de ce mécanisme sur un système propriétaire tel qu'AIX ne permet pas d'appliquer totalement ces optimisations qui nécessitent l'accès à un interface interne du noyau. La réalisation peut effectivement bénéficier du gain apporté par une communication par mémoire partagée plutôt que par messages, pour échanger les paramètres, et les capacités nécessaires à l'appel. Par contre, faute de disposer d'un accès aux structures internes du noyau, la réalisation souffre du coût important – de 10 à 15 ms en moyenne – d'une commutation de contexte entre l'appelant et l'appelé.

Par ailleurs, le mode dans lequel les talons des appels doivent s'exécuter pose un problème de compromis entre performances et souplesse d'utilisation : en effet, dans le mécanisme original du LRPC, le talon s'exécute dans l'espace utilisateur, puis l'appel de procédure effectif est réalisé par un appel système et la suite du mécanisme se déroule donc dans l'espace du noyau. Ici, l'exécution d'une activité est d'abord déroutée sur une exception pour défaut de segment, et le traitement de cette exception doit permettre de retrouver une capacité d'appel et à partir de cette capacité, de retrouver les talons de la procédure protégée. Pour réaliser le LRPC, l'exception est résolue et l'activité reprend son exécution pour effectuer le traitement du talon, puis l'appel de procédure effectif par un appel système. L'exécution du talon dans l'espace utilisateur présente l'avantage de ne pas contraindre le traitement qui peut y être effectué. L'appel système qui met en œuvre l'appel de procédure effectif est chargé de retrouver les capacités nécessaires, de les transférer, de

démarrer un nouveau flot dans un autre domaine et de bloquer le flot appelant. Cet appel réalise une traversée supplémentaire de la frontière entre l'espace utilisateur et l'espace noyau. Pour éviter le surcoût, on pourrait envisager d'effectuer le traitement réalisé par le client dans l'espace noyau à condition bien sûr que la compilation de ce talon assure d'une part l'adressage des paramètres rangés en espace utilisateur depuis l'espace noyau, et la certification du code de ce talon importé dans le noyau. Il convient toutefois de noter que le surcoût d'un appel système supplémentaire (2 µs) est vraiment négligeable dans le coût du traitement de l'exception de défaut de segment, comme l'indiquent les résultats des mesures (§ V.3.2).

L'optimisation la plus importante et donc la plus urgente dont le changement de domaine puisse bénéficier est d'abord celle qui consisterait à ne pas suspendre le flot appelant, mais à changer directement son contexte d'exécution, évitant ainsi le surcoût d'une commutation de contexte inutile.

V.2.3 Administration de la protection

L'administration de la protection est réalisée comme une application directe de la mémoire virtuelle répartie d'Arias. Cette application assure deux tâches principales qu'il convient d'évaluer : il s'agit de la gestion de la cohérence des structures de données réparties correspondant aux graphes de listes de capacités, et la maintenance des structures de données correspondant aux arbres de recherche pour les domaines de protection.

Tout d'abord, l'interface offerte à l'utilisateur, basée sur un adressage plat des listes de capacités est satisfaisante dans la mesure où elle est suffisamment générale pour permettre de construire n'importe quel type de désignation.

Par ailleurs, le développement des applications de test révèle que la mise en place d'une désignation symbolique pour les listes de capacités est indispensable. En effet, les identificateurs de listes sont alloués dans un espace de noms plat : ils offrent ainsi une interface ouverte, mais dont l'utilisation telle quelle n'est pas très conviviale.

Ensuite, l'approche qui consiste à considérer l'administration de la protection comme une application répartie à part entière permet de bénéficier du service de la couche générique de cohérence et du service de stockage et de journalisation permettant de résister aux pannes. Ainsi, nous pouvons espérer adapter le protocole utilisé pour la gestion de la cohérence aux besoins de l'administration de la

protection. Mais la première version du prototype ne propose d'une part qu'un seul protocole de cohérence (cohérence stricte), et aucune application administrant la protection d'autre part. Il n'est donc pas encore possible d'évaluer l'adéquation et l'influence de protocoles de cohérence sur l'administration de la protection dans une situation réelle de partage des listes de capacités.

Cependant, la gestion de la cohérence des listes de capacités doit aussi s'attacher à maintenir à jour les arbres de capacités. Cette mise à jour est provoquée par la modification d'une liste de capacités, et elle doit être déclenchée localement pour l'ensemble des domaines qui partagent cette liste, et sur l'ensemble des sites où ces domaines sont représentés. La mise en œuvre de cette cascade de mises à jour s'avère une opération complexe que la répartition rend de plus fragile vis à vis de la tolérance aux défaillances et coûteuse. Or l'interface proposée pour l'administration des listes de capacités provoque cette mise à jour de façon systématique lors de la modification d'une liste de capacités. Une interface de type transactionnelle serait peut−être plus adaptée à la gestion des listes de capacités : elle permettrait toujours de bénéficier du service de la couche générique de cohérence pour les graphes de listes, mais également de factoriser les mises à jour correspondantes des arbres de capacités lors de la validation de la transaction.

Enfin, il convient de noter ici que la gestion des listes de capacités constitue une application particulière et privilégiée du système, dans la mesure où l'accès aux listes de capacités ainsi qu'aux capacités qu'elles contiennent, doit être lui−même rigoureusement contrôlé : sans cette précaution, les listes de capacités pourraient être modifiées (par accident ou par malveillance) et le contrôle d'accès qu'elles mettent en œuvre serait compromis. Les capacités demeurant confinées au sein du noyau AIX, elles disposent de la protection intrinsèque garantie par l'enceinte du noyau. Cependant il peut subsister des possibilités d'accès non contrôlé aux listes de capacités :

- un utilisateur ou une application peut parvenir à modifier les structures de données confinées localement, dès lors qu'il acquiert légitimement ou non, les privilèges superviseur sur une machine de la plate−forme Arias. L'administration de la protection dans Arias est ici soumise exactement aux mêmes risques d'effraction que l'administration globale des systèmes Unix composant la plate−forme.
- un utilisateur ou une application peut intercepter, modifier ou forger les données et les messages qui transitent d'un site à un autre, notamment lors d'un requête de propagation d'une modification dans une liste partagée de

capacités, ou lors d'une requête de changement de site d'exécution. L'exploitation d'une telle faille peut permettre à un utilisateur ou à une application de s'accorder à distance des privilèges sur une liste de capacités ou un domaine de protection. Pour éviter ce type d'intrusion dans une situation de méfiance mutuelle des sites de la plate–forme Arias, il est nécessaire de mettre en place un service d'authentification permettant de vérifier les identités respectives des domaines ou des gestionnaires de protection émetteur et récepteur de tout message de requête.

De plus, ce service d'authentification ne concerne pas seulement la sécurité de l'administration des listes de capacités, mais il intéresse aussi la sécurité de chaque service participant à la gestion des données réparties, partagées et persistantes d'Arias. Cet aspect d'authentification n'a pas été traité dans la plate–forme Arias, et il convient d'en tenir compte pour l'évaluation du service de la protection :

a) d'un point de vue qualitatif, la réalisation actuelle du prototype de la plate–forme ne permet donc pas de réaliser une configuration de domaines de protection mutuellement méfiants,

b) les mesures qui suivent et les performances que l'on peut en déduire ne reflètent pas l'intégralité d'un service de protection garantissant la sécurité. Les résultats de cette étude permettent d'évaluer les mécanismes strictement liés au contrôle d'accès aux données partagées. Ils ne concernent en aucun cas les aspects liés à l'authentification et le surcoût que cette opération représente.

V.3 Mesures et performances

V.3.1 Bilan des développements

Avant de présenter les résultats des mesures relevées sur le fonctionnement de la plate–forme, il peut être intéressant d'évaluer la complexité relative des différents mécanismes mis en œuvre pour le service de la protection. Le tableau décrit sur la figure Fig. 5.1 présente la répartition des différents modules identifiés dans le service de la protection. Cette répartition est exprimé en termes de lignes d'instructions C écrites.

La complexité du code écrit n'est pas liée a priori au nombre de lignes d'instruction qui le constituent. Cependant, pour la gestion des domaines, de celle des

listes de capacité, et de celle des arbres de recherche et de synthèse des domaines, la complexité et le temps consacré à la conception, à la mise en œuvre et aux tests correspondent à la taille relative de ces modules. La difficulté rencontrée dans la conception de la gestion des listes de capacités était due :

- à la complexité de la gestion de la cohérence de ces structures de données partagées,
- aux développements nécessaires pour la mise en œuvre de ce module dans l'architecture *STREAMS* adoptée pour Arias.

Mais cette difficulté d'initialisation des développements s'est révélée intéressante dans la suite, car après trois mois de programmation et de tests, l'intégration du service de la protection dans le prototype n'a nécessité qu'une semaine de travail supplémentaire.

Mécanismes	Taille du code	% Total
Gestion des domaines	1.000	14%
Gestion des listes de capacités	2.500	36%
Gestion des arbres de recherche	800	11%
Gestion des PPIs(1)	2.000	–
Interface applicative	700	10%
Tests unitaires	2.000	28%
TOTAL	7.000 lignes	22% du code intégré au prototype(2)

Fig. 5.1 : Répartition du code pour le service de la protection

(1) la gestion des interfaces protégées comprend essentiellement un compilateur de PPIs qui a été développé parallèlement par une autre personne de l'équipe. Ce compilateur n'a pas encore été intégré au prototype de la plate–forme et n'est pas comptabilisé dans les totaux présentés dans ce tableau. Par ailleurs son interface avec les services réalisés et analysés ici est décrite en annexe.

(2) rappelons que le prototype à la fin du premier semestre 1996 intègre le service de localisation des segments répartis, ainsi que le service de la couche générique supportant des protocoles de cohérence spécifiques. Le code intégré totalise alors 31.000 lignes de code C.

Les mesures présentées ici ont été réalisées sur des machines Bull Escala, en version monoprocesseur PowerPC 601, et s'exécutant sous le système AIX 4.1.4. L'application–test que nous avons utilisée est un outil qui nous sert, en l'absence d'un service de permanence, à reproduire une configuration de la mémoire persistante d'après des descriptions de la composition d'un domaine de protection en termes de segments, de listes de capacités et de capacités de tout type. Les mesures ont été effectuées sur l'utilisation de cet outil pour reproduire dans la mémoire répartie d'Arias le système de fichiers d'une station de travail. Ce système de fichiers est alors représenté par :

- 41252 segments, totalisant un espace de 824 méga–octets,
- 10440 listes de capacités,
- 16 utilisateurs.

Un essai consiste alors à reproduire tout ou partie de ce système de fichiers en sélectionnant un sous–ensemble d'utilisateurs propriétaires des fichiers, et en représentant chaque utilisateur par un domaine de protection. Pendant cet essai, les mesures sont cumulées en mémoire dans les différents modules du service de la protection. En effet, comme seuls des outils de mesures temps réel sont disponibles à l'intérieur du noyau AIX, il est nécessaire de contourner autant que possible toute perturbation due à l'ordonnancement des processus, des *threads* ou des *STREAMS*. Puis à la fin de l'essai ce cumul est extrait pour calculer le coût moyen de chaque opération.

Une série d'essais consiste à recréer le système de fichiers pour plusieurs sous–ensembles d'utilisateurs pour obtenir une série de mesures réparties dans l'intervalle de 100 à 41252 segments ; pour confirmer les résultats obtenus par une série d'essais, elle est reproduite sur plusieurs nuits afin d'éviter les perturbations générées par des pics de charge réseau. De plus, une série d'essais est très longue (deux heures et demie environ) car il est nécessaire de réinitialiser les machines de test : la réalisation des *STREAMS* dans AIX souffre de fuites de mémoire qui pénalisent fortement les performances obtenues dans la gestion des messages à l'intérieur d'un flux *STREAM* à la suite d'installations/désinstallations répétitives de ce flux, et par conséquent d'Arias. La durée effective totale des essais d'une série ne représente cependant que trente à quarante minutes.

Les résultats de ces mesures donnent finalement un nuage de points dont on peut alors tracer la médiane.

V.3.2 Mécanismes de pagination

Après avoir reproduit la configuration de tout ou partie de la mémoire répartie décrite ci−dessus, les performances des mécanismes de pagination peuvent être mesurées par un simple parcours de cette mémoire. Nous avons ainsi mesuré le coût de traitement d'une exception pour défaut de segment dans deux configurations de la plate−forme Arias, avec et sans le service de la protection.

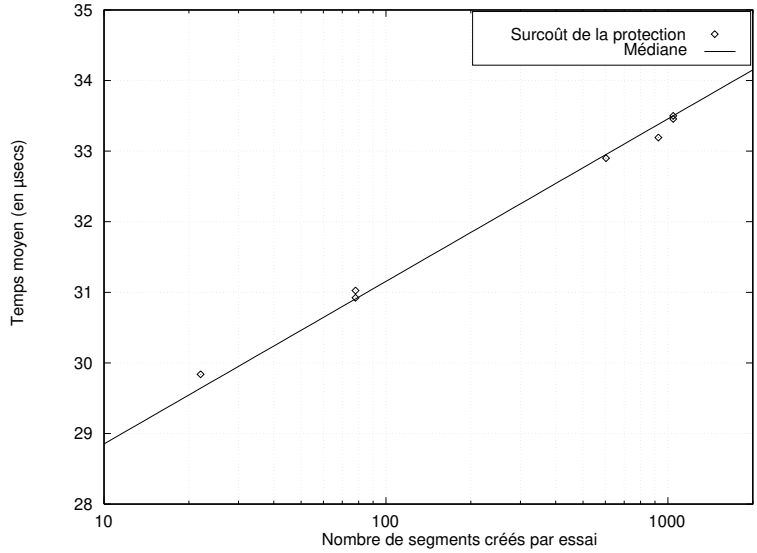


Fig. 5.2 : Surcoût de la protection dans la pagination

Lors d'une exception pour défaut de segment, le surcoût de la protection provient de deux traitements supplémentaires :

- l'identification du domaine courant dont le coût est fixe puisqu'il s'agit d'une consultation dans une table de hachage,
- la recherche des capacités dans l'arbre de recherche du domaine courant.

Les algorithmes utilisés pour cette recherche de capacités sont basés sur des arbres équilibrés [Knuth 73]. La complexité d'une recherche dans un tel arbre est donc de l'ordre de la hauteur de l'arbre. Or la hauteur h d'un arbre équilibré de n nœuds est caractérisée par :

$log_2(n+1) \leq h+1 < 1{,}44 log_2(n+2)$

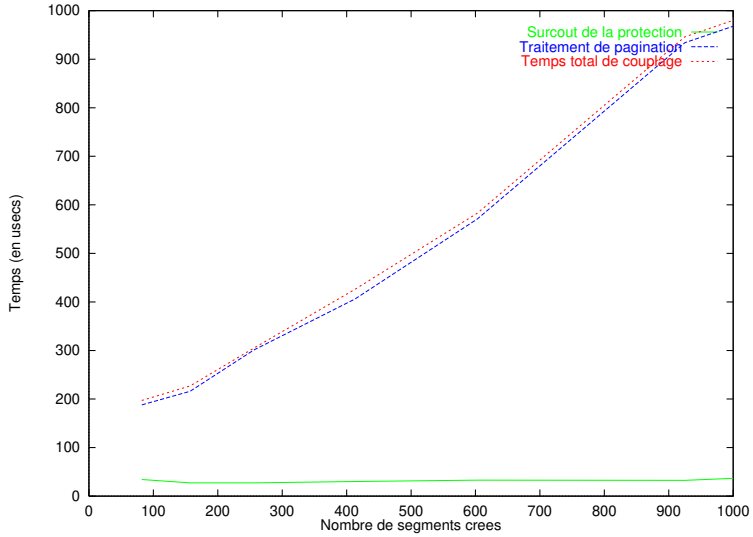


Fig. 5.3 : Comparaison des temps de traitement de défaut de segment

Les résultats apparaissent sur les figures Fig. 5.2 et Fig. 5.3. Ils confirment que le surcoût de la protection est de l'ordre de log(n), où n est le nombre de capacités dans le domaine courant, et surtout que ce surcoût est relativement faible dans les mécanismes de pagination : il varie de 10% lorsque le nombre de segments est très faible, à 3% lorsque le nombre de segments croît et devient plus réaliste.

V.3.3 Gestion des informations de protection confinées

Outre les mécanismes de pagination, le coût induit par le contrôle d'accès à la mémoire répartie intervient également dans la gestion des informations de protection confinées, en particulier lors de la création ou la destruction d'un segment.

La création (resp. la destruction) d'un segment réclame la création (resp. la destruction) implicite d'une capacité, comme nous l'avons vu au chapitre précédent (§ IV.3.4). Le surcoût de la protection lors de la création d'un segment est donc essentiellement dû à :

- la création de cette capacité et à son insertion dans le graphe de listes dont les coûts sont fixes (exception faite de la gestion de la cohérence des listes),
- l'insertion de cette capacité dans l'arbre de recherche du domaine courant notamment.

En ce qui concerne l'insertion d'une capacité dans un arbre de recherche, la complexité de l'algorithme d'insertion dans un arbre équilibré est de l'ordre de log(n), n représentant le nombre total de capacités dans l'arbre. Cette complexité logarithmique reste valable même dans le pire des cas, c'est–à–dire celui où l'opération nécessite un rééquilibrage.

Les figures Fig. 5.4 et Fig. 5.5 décrivent les mesures réalisées pour respectivement la création de segments et pour l'entretien des arbres de recherche, et les résultats obtenus reproduisent cette complexité d'ordre log(n). L'analyse du coût de gestion des graphes de listes n'inclut pas le maintien de la cohérence des listes dans la mesure où :

- ce coût fait intervenir le degré de partage et de répartition des listes, c'est–à–dire un facteur dépendant de l'application,
- cette analyse doit prendre en compte l'adéquation du protocole de cohérence et de synchronisation au schéma d'accès aux listes de capacités construit par l'application.

Pour évaluer ces deux aspects, nous ne disposons ni de véritables applications, ni d'une caractérisation de ces applications qui permettrait une simulation des accès, ni de plusieurs protocoles de cohérence pour gérer les graphes de listes réparties. Afin d'approcher le surcoût de la protection lors de l'exécution d'une application, les mesures ci–dessous sont donc prises au cours de l'exécution d'activités pendant laquelle elles effectuent des créations de segments et des copies de capacités (pour éventuellement partager les segments) dans des listes partagées.

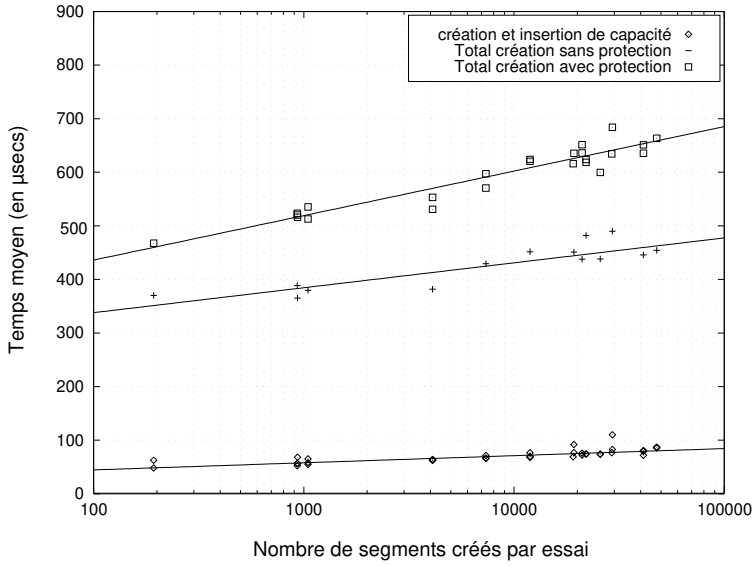


Fig. 5.4 : Temps de création de segments

Sur l'ensemble de l'expérimentation, les résultats indiqués sur la figure Fig. 5.4, nous montrent que le surcoût inhérent à la protection (courbe inférieure) représente 10% environ du coût total de la création d'un segment protégé.

Il convient de noter ici que l'écart entre le temps total de création avec protection et le temps total de création sans protection est supérieur au coût propre de la

protection : cette différence est de l'ordre de 50 µs, et elle provient des traitements effectués par le mécanisme des STREAMS, pour acheminer les messages entre les modules, en présence du module supplémentaire de la protection.

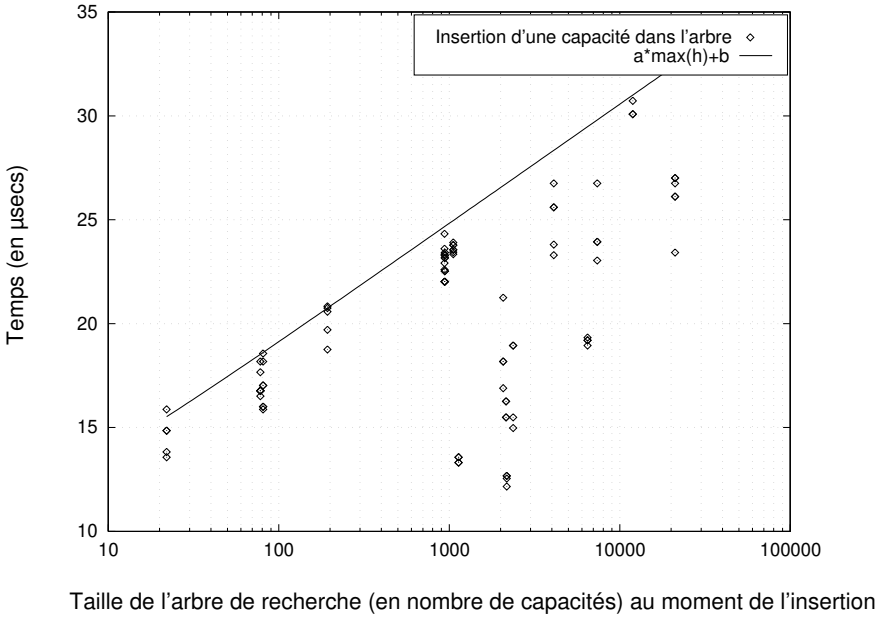


Fig. 5.5 : Coût d'entretien d'un arbre de recherche

V.4 Conclusion et perspectives

Les objectifs particuliers du service de protection sont donc de fournir aux applications un mécanisme générique de contrôle d'accès aux données réparties partagées. Outre la souplesse et le dynamisme du contrôle d'accès, rappelons les propriétés qui sont d'abord recherchées, et qui ont guidé les choix de conception de ce service de protection vers la proposition présentée ici. Il s'agit des propriétés suivantes :

a) une application doit pouvoir s'exécuter de la même façon qu'elle ait besoin du service de la protection ou non,

b) en corollaire, une application qui n'utilise pas le service de la protection ne doit payer le coût de ce service,

c) enfin, une application doit pouvoir utiliser le service de la protection dans différentes configurations afin de permettre la mise en œuvre de différentes politiques dans les mécanismes sous–jacents sans modifier l'application.

La réalisation du service de protection proposé doit s'accorder avec les choix d'une architecture globale orientée vers la portabilité du prototype avec l'utilisation du modèle de communication des STREAMS. La réalisation doit aussi s'intégrer et coopérer avec d'autres services spécifiques afin d'offrir une plate–forme totalement configurable.

La proposition et la réalisation que j'ai faites dans ce cadre et présentées ici abordent deux aspects du contrôle d'accès, à savoir :

1. le contrôle d'accès aux données réparties partagées à l'intérieur d'un domaine de protection, réalisé à travers les mécanismes de pagination et de manipulation de segments,
2. le contrôle d'accès à ces domaines de protection, réalisé à travers le mécanisme de changement de domaine et d'interface de procédure protégée.

Le travail effectué permet de fournir un service qui satisfait aux propriétés a) et b) sur ces deux aspects. La mise en œuvre du service démontre que les objectifs souhaités sont accessibles par la réunion de mécanismes et de techniques parfaitement connus aujourd'hui tels que les capacités logicielles confinées, les arbres équilibrés, le LRPC et les langages définition d'interface. De plus, les mesures effectuées montrent que cette proposition est également viable et raisonnable en matière de performances puisque le surcoût du contrôle d'accès à des données partagées est minime par rapport au temps total d'un accès non contrôlé. Toutefois, ces mesures ne permettent qu'une validation partielle dans la mesure où les contraintes d'une plate–forme propriétaire ne nous ont pas permis de réaliser un mécanisme de changement de domaine efficace.

Par ailleurs, l'architecture modulaire globale de la plate–forme et l'intégration du service de la protection aux autres services disponibles permettent d'attendre du service de la protection qu'il soit configurable en fonction des besoins des applications qui l'utilisent. Cette latitude de configuration du service de la protection est particulièrement intéressante en ce qui concerne la définition des enceintes de protection représentés par les domaines, et l'expression du degré de confiance ou de méfiance entre deux domaines. En effet, une interface de procédure protégée permet de définir les points d'entrée dans un domaine de protection, et de spécifier les informations de protection qui doivent et peuvent être échangées pour réaliser la coopération entre deux domaines, et ces spécifications de protection sont indépendantes de l'application exécutée. Cependant, la mise en œuvre effectuée ne permet pas encore d'entériner ces résultats par une utilisation réelle car l'intégration globale de l'ensemble des services offerts sur la plate–forme réalisée n'est pas encore achevé, et les applications réelles font défaut pour une telle expérimentation.

Ce travail a permis de montrer que le service de contrôle d'accès à base de capacités mis en œuvre pour une grande mémoire virtuelle peut :

- être systématiquement réalisé de façon implicite lors des accès aux données partagées,
- être systématiquement réalisé de façon implicite lors des changements de domaines de protection (à condition que ces accès se fassent sur un appel de procédure), et intégrer un contrôle systématique et également implicite des informations de protection échangées lors de cet accès,
- être, du fait de ces contrôles d'accès implicites, totalement indépendant de toute désignation, notamment d'une désignation basée sur des capacités à mot de passe qui assimilent le contrôle d'accès aux données et l'authentification de ces accès.

Conclusion

1 Rappel des objectifs

Les objectifs du projet SIRAC consistent notamment à fournir un support système pour la gestion de données partagées persistantes et réparties et répondre aux besoins d'une large classe d'applications. Par conséquent, ce service doit donc être :

- adaptable et configurable en fonction des besoins des applications,
- portable et acceptable en termes de performances pour permettre sa diffusion et son utilisation sur un large éventail d'environnements d'exécution,
- sûr en offrant la possibilité de protéger les accès aux données partagées, et de rendre ces données résistantes aux pannes.

Le projet SIRAC a démarré en fin 1994, et dans ce cadre, une plate–forme d'expérimentation a été définie et spécifiée pour offrir un ensemble de services « à la carte » pour la gestion de données persistantes, réparties et partagées. Le premier prototype de cette plate–forme est développé sur le système AIX, et une version minimale a été achevée à la fin de l'année 1995 permettant d'offrir une mémoire virtuelle répartie, persistante et partagée sur un ensemble de machines homogènes.

Dans ce cadre, je me suis intéressé à intégrer la notion de service « à la carte » dans le service de protection d'accès à la mémoire virtuelle répartie offerte par la plate–forme, et à étudier les impacts de ces objectifs sur les mécanismes de contrôle d'accès à base de capacités. Cette approche se traduit alors essentiellement par :

1. la volonté de permettre à un utilisateur de manipuler exclusivement des adresses virtuelles,
2. la nécessité pour le système d'assurer la gestion complète des capacités dans chaque mécanisme où un contrôle d'accès est nécessaire, c'est–à–dire dans la définition d'un domaine de protection, dans son évolution au fur et à mesure des ajouts, partages ou retraits de capacités, et lorsqu'une activité évolue d'un domaine de protection à un autre.

Ces deux principaux objectifs ont alors conduit :

- à la proposition d'un service de protection d'accès à une mémoire virtuelle répartie, à base de capacités logicielles, confinées et implicites,

- à l'étude des mécanismes à mettre en œuvre dans le contexte spécifique de l'architecture retenue pour la plate−forme Arias,
- à la réalisation puis à l'intégration de premiers mécanismes dans la version disponible du prototype, ainsi qu'à leur instrumentation pour permettre le relevé de mesures,
- à l'évaluation de la proposition et des résultats obtenus par cette première expérimentation.

2 Rappel des résultats

Le service de protection proposé étudie la possibilité d'offrir un service « à la carte » pour les mécanismes de contrôle d'accès à la mémoire répartie partagée. Il est basé sur une réalisation où les capacités demeurent invisibles à l'utilisateur dans la mesure où :

- les capacités sont confinées,
- le nom d'une capacité correspond à l'adresse virtuelle de l'entité protégée, que ce soit un segment ou bien une procédure,
- le descripteur correspondant à une capacité est recherché par le système au moment du traitement d'une exception, ou au moment du transfert d'exécution dans un autre domaine.

Par ailleurs, ce service de protection offre une interface d'administration des capacités et de contrôle de leurs mouvements, totalement indépendante des mécanismes d'adressage. Cette interface permet bien sûr les opérations habituelles pour obtenir des capacités, les déplacer, les détruire ou les restreindre, mais elle permet aussi de spécifier et de contrôler quelles sont les capacités passées en paramètres et transmises à un autre domaine de protection lors d'un appel de procédure protégée. Cette interface d'administration permet donc de configurer la protection d'accès aux données et aux services partagés, et une fois que cette configuration est établie, l'utilisation de capacités est entièrement implicite et invisible pour l'application qui s'exécute dans cette configuration.

Ce service se distingue donc des projets Opal et Mungi où d'une part la désignation des capacités est explicite et d'autre part, le contrôle des droits transférés lors d'un appel de procédure protégée n'est pas intégré à l'interface de cette procédure et doit se faire manuellement et explicitement.

Le mécanisme mis en œuvre a permis de vérifier expérimentalement l'objectif d'adaptabilité car :

- le contrôle d'accès est implicitement et entièrement géré par le service de protection. Son utilisation ne nécessite donc ni modification du code ni même de recompilation des applications. Ainsi ce contrôle est totalement transparent pour tout accès aux données réparties partagées.
- l'administration de la protection offre une interface générale qui permet d'adapter le modèle de protection des listes de capacités partagées à plusieurs modèles de protection suivant les spécifications des applications. L'application développée à l'occasion des tests d'évaluation a notamment permis de reproduire le modèle de protection des fichiers Unix.
- les résultats des mesures de performances montrent que les mécanismes de protection d'accès ont un surcoût acceptable dans les mécanismes de pagination et de manipulation des segments.

D'autres résultats attendus restent encore à confirmer par l'expérimentation, aussitôt que l'intégration du prototype le permettra. Il s'agit notamment de vérifier que :

- le service de la protection peut vraiment bénéficier des possibilités de configuration de la plate−forme dans la mesure où il est conçu comme une application privilégiée qui utilise l'ensemble des services pour la gestion de données persistantes, réparties et partagées. Il faut s'assurer que la mise en place de la permanence, ou le choix de différents protocoles de cohérence pour la mémoire répartie restent compatibles avec le service proposé.
- les interfaces des procédures protégées permettent une reconfiguration facile du schéma de protection pour une application, ainsi que la mise en œuvre effective des opérations d'amplification, de restriction, de délégation et d'isolation. Il s'agit également de vérifier que l'échange implicite de capacités lors d'un changement de domaine offre des performances acceptables. Pour cela, il reste d'abord à intégrer au prototype les mécanismes chargés de retrouver les capacités nécessaires pour un changement de domaine d'après la spécification d'une PPI, et ceux chargés de la gestion des segments de code.

Par ailleurs, la réalisation a permis de révéler les contraintes imposées par l'objectif de portabilité fixé globalement dans SIRAC. En effet, cet objectif nous a d'abord conduit à développer le service de protection dans une architecture globale de STREAMS. Cette architecture globale a permis un prototypage rapide et une grande facilité de configuration de la plate−forme expérimentale. Toutefois ces avantages acquis le sont parfois au détriment des performances du service offert. Dans le cadre du service de la protection, ces inconvénients apparaissent dans le surcoût imposé par la présence du module STREAMS supplémentaire pour le

contrôle d'accès, notamment à la création d'un segment. Ensuite, l'utilisation d'interfaces standard n'a pas non plus permis de réaliser au mieux le mécanisme de changement de domaine. Il convient toutefois de noter que ces contradictions ne sont pas aberrantes dans la mesure où les performances ne constituent pas un objectif principal dans la réalisation d'une plate–forme expérimentale pour SIRAC.

3 Perspectives

Les perspectives ouvertes par cette mise en œuvre permettent donc d'envisager à court terme :

- d'offrir une souplesse plus grande de configuration de la plate–forme. Cette perspective nécessite une intégration plus avancée des différents services pour permettre au service de la protection d'utiliser plusieurs protocoles de cohérence et de synchronisation, et de bénéficier pour les informations de protection, du service de permanence offert par le gestionnaire de stockage,
- de valider l'ensemble de la plate–forme, puisque le service de la protection constitue une première application de cette plate–forme. En effet, il utilise l'ensemble des services offerts pour la gestion des données persistantes partagées et réparties,
- de développer des applications réelles complètes, notamment en faisant bénéficier du service de la protection les actions de portage d'applications réparties telles que le système de fichiers NFS.

Outre ces perspectives ouvertes pour valider l'ensemble de la plate–forme, il convient également d'envisager la réalisation d'un environnement complet pour l'administration, la configuration d'applications coopératives réparties. Cette réalisation peut se développer à plus long terme, dès que la plate–forme est totalement intégrée, et elle intervient à plusieurs niveaux :

- au niveau des services offerts pour la gestion des données réparties, il est nécessaire de compléter la protection disponible et d'y ajouter un service d'authentification indépendamment du service de contrôle d'accès. Un tel service s'avère notamment nécessaire pour assurer la sécurité de l'ensemble des services offerts par la plate–forme réalisée. De plus, cette réalisation permettrait non seulement d'offrir une sécurité complète, mais aussi d'étudier dans quelle mesure un service d'authentification peut également être conçu comme un service « à la carte », et configurable avec plusieurs politiques sans nécessiter aucune modification des applications.

- au niveau des applications, il est intéressant d'envisager l'étude et la mise en œuvre d'un modèle de protection de haut niveau pour les applications coopératives. Ce modèle doit permettre de spécifier des droits d'accès à grain fin (c'est–à–dire en termes d'opérations définies sur les données), de définir les privilèges d'un utilisateur en fonction de son rôle social dans la tâche coopérative, de contrôler le partage entre les différents utilisateurs des résultats des actions faites sur les objets partagées (rétroaction). Ce modèle doit aussi et surtout fournir des mécanismes faciles à utiliser qui permettent aux utilisateurs d'adapter dynamiquement le système en fonction de leurs besoins et de leurs rôles. Ces mécanismes sont des opérations de protection de haut niveau, telles que la délégation et la révocation sélectives et dynamiques de droits d'accès. Cette étude permettrait de caractériser l'adaptabilité des mécanismes système que nous avons proposés, et d'identifier leurs points d'accord ou leurs dissonances avec des modèles de protection de haut niveau.

Annexe A

Langage de Définition d'Interface d'Arias (PPI)

A.1 Rappels des objectifs recherchés

Avant de donner la définition retenue pour le langage de définition d'interface des PPI, il convient de rappeler brièvement les objectifs de ce langage énoncés au paragraphe IV.1.5 :

- le langage de définition d'interface doit en premier lieu permettre de préciser la signature des procédures protégées ;
- ce langage doit de plus permettre de spécifier les modalités de transfert d'éventuelles capacités lorsqu'un changement de domaine se produit sur un appel à une telle procédure.

Ces spécifications de la procédure protégée concernent potentiellement chaque paramètre qui représente une adresse virtuelle, et pour chacun elles doivent préciser si une capacité doit accompagner cette adresse virtuelle ou pas. Si oui, alors ces spécifications doivent exprimer quel type de capacité doit accompagner l'adresse virtuelle passée en paramètre (capacité de couplage ou capacité d'appel), et quels sont les droits transportés par la capacité s'il s'agit d'une capacité de couplage (lecture, écriture ou exécution). Enfin les spécifications d'interface d'une PPI doivent aussi permettre de préciser, éventuellement, dans quelle liste de capacités du domaine de protection appelant il faut essayer de trouver (ou de dériver) la capacité accompagnant le paramètre. En l'absence de cette information, la capacité demandée est recherchée dans l'arbre de recherche du domaine appelant. Et de la même façon le langage de définition d'interface d'une PPI doit permettre de préciser dans quelle liste de capacités du domaine de protection appelé, la capacité transmise avec le paramètre doit être installée.

L'étude et le choix d'un langage de définition d'interface ne constitue pas un objectif du service de la protection dans Arias. Ce langage doit avant tout être proche des langages de définition d'interface courants (RPC Sun ou IDL Corba) et ainsi

facile d'utilisation. De plus, il faut développer pour ce langage un compilateur d'interface qui puisse générer, à partir de la spécification d'une PPI, le code nécessaire pour effectuer la recherche et le transfert des capacités au sein du système.

Pour répondre à l'objectif d'un prototypage rapide, le langage présenté ci–dessous a été élaboré à partir du langage de définition d'interface du RPC Sun, et une personne a repris et modifié le générateur de talons de ce dernier pour l'adapter à la grammaire du langage de définition des PPI, ainsi qu'au principe des mécanismes de contrôle d'accès mis en œuvre dans la plate–forme Arias. Le générateur obtenu doit encore être intégré au prototype d'une part avec l'interface du gestionnaire de la protection, et avec les mécanismes de gestion des segments de code d'autre part.

A.2 Grammaire du langage

definition–list → definition '**;**'
| definition '**;**' definition–list

definition → struct–definition
| typedef–definition
| proc–definition

struct–definition → 'struct' struct–ident '**{**' arias–decl–list '**}**'

arias–decl–list → arias–declaration '**;**'
| arias–declaration '**;**' arias–decl–list

typedef–definition → '**typedef**' simple–declaration

proc–definition → '**procedure**' proc–ident '**(** ' param–list '**)** '

param–list → param
| param '**,**' param–list

param → direction arias–declaration

direction → '**in**' | '**out**'

arias–declaration → simple–declaration
| capa–declaration

simple–declaration → type–ident variable–ident
| type–ident variable–ident '**[**' value '**]**'
| type–ident '*****' variable–ident
| type–ident '**(***' function–ident '**)**'

capa–declaration → simple–capa
| simple–capa direction capa–list–ident

simple–capa → capa–type type–ident '*****' variable–ident
| '**capa_dom_call**' type–ident '**(***' function–ident '**)**'

capa–type → '**capa_seg_read**'
| '**capa_seg_write**'
| '**capa_seg_execute**'

direction → '**install**'
| '**from**'

Annexe B

Exemples détaillés d'utilisation des mécanismes de protection

Nous fournissons ici deux exemples destinés à illustrer et éventuellement clarifier le fonctionnement des mécanismes réalisés pour le service de la protection offert par la plate−forme Arias. Pour chaque exemple, nous décrivons 1− le schéma de protection désiré, 2− les étapes de sa mise en place et la construction des listes de capacités. Cette dernière étape fait plus particulièrement ressortir les interfaces de protection proposées.

B.1 Un serveur d'impression

B.1.1 L'objectif

Il s'agit de mettre en place un service public et partagé d'impression. Le service consiste en un domaine qui est le seul habilité à exécuter le code d'impression et à accéder à l'imprimante. Chaque requête d'impression se traduit par un appel au service avec en paramètre, l'adresse du ou des segment(s) où se trouvent les données à imprimer.

Segments/Domaines		Domaine du Programmeur d'Application	Domaine du Serveur d'Impression	Domaine Client
Pilote d'imprimante	Seg.	??	RW	Ø
	Liste(1)	??	R	Ø
Code d'impression	Seg.	RW	X	capacités de changement de domaine pour chaque point d'entrée
	Liste(1)	RW+Copie	R	R
Données à imprimer	Seg.	??	exactement R	au moins R
	Liste(1)	??	R, dans la liste privée de l'activité non délégable(2)	R, la capacité est placé dans une liste d'export(2)

Fig. 2.1 : Matrice de protection

B.1.2 Mise en place

Puisqu'il y a des paramètres, il convient d'une part de définir l'interface (ou PPI) du service d'impression avec les caractéristiques de protection désirées et d'autre part, de construire et de partager les capacités nécessaires à la disponibilité du service.

B.1.2.1 Définition de l'interface des procédures protégées

Cette interface peut s'écrire ainsi :

```
Status Imprimer (
    in capa_seg_read Document *doc;
    out RequestId id;
    );
```

pour indiquer :

---

(1) La ligne Liste désigne les droits donnés par une capacité de liste, sur la liste de capacités où se trouve la capacité sur le segment. Plus clairement, ces droits représentent les droits de manipulation de la capacité de couplage du segment (ou de la capacité de changement de domaine).

(2) Nous introduisons ici, un droit de délégation, c'est–à–dire le droit de copier une capacité lors d'un appel inter–domaine. Un tel droit est bien plus restrictif que le droit de copie d'une capacité. Le droit de délégation peut simplement être représenté par les listes déclarées pour l'export explicitement par 'from' ou implicitement.

1. la nécessité de passer une capacité pour le segment contenant le document à imprimer. Le client pourra « surcharger » cette définition par :

   ```
   [in capa_seg_read Document *doc] from
   LISTE_DES_IMPRIMABLES;
   ```

   où LISTE_DES_IMPRIMABLES est la désignation symbolique d'une liste de capacités délégables(2) (le droit de copie est invalidé sur les capacités contenues dans cette liste). Sinon, par défaut, la capacité nécessaire pour doc est recherchée dans la liste d'export.
2. la nécessité de disposer d'une capacité de couplage donnant au moins le droit d'accès en lecture au segment contenant le document. Lors de l'appel, si le client accepte cette PPI, et exporte une capacité satisfaisant ces contraintes, l'appel inter–domaine est réalisé en passant une capacité réalisant exactement les droits nécessaires, c'est–à–dire ici la lecture.
3. le serveur peut également préciser la destination de la capacité reçue en paramètre en « surchargeant » la définition donnée par la PPI :

   ```
   [in capa_seg_read Document *doc] install
   LISTE_A_IMPRIMER;
   ```

   où LISTE_A_IMPRIMER est la désignation symbolique d'une liste de capacités déléguées (le droit de copie est invalidé sur les capacités contenues dans cette liste, et cette liste doit être marquée pour ne pas servir de liste d'exportation par ailleurs). Sinon, par défaut, la capacité nécessaire pour doc est placée dans la liste privée de l'activité.

B.1.2.2 Définitions des listes de capacités

À l'origine, on ne dispose que d'un domaine dans lequel est disponible une capacité sur le pilote d'impression. Les étapes de mise en place du schéma de protection que nous venons de définir, sont détaillées ci–dessous, à partir de l'illustration Fig. 2.2.

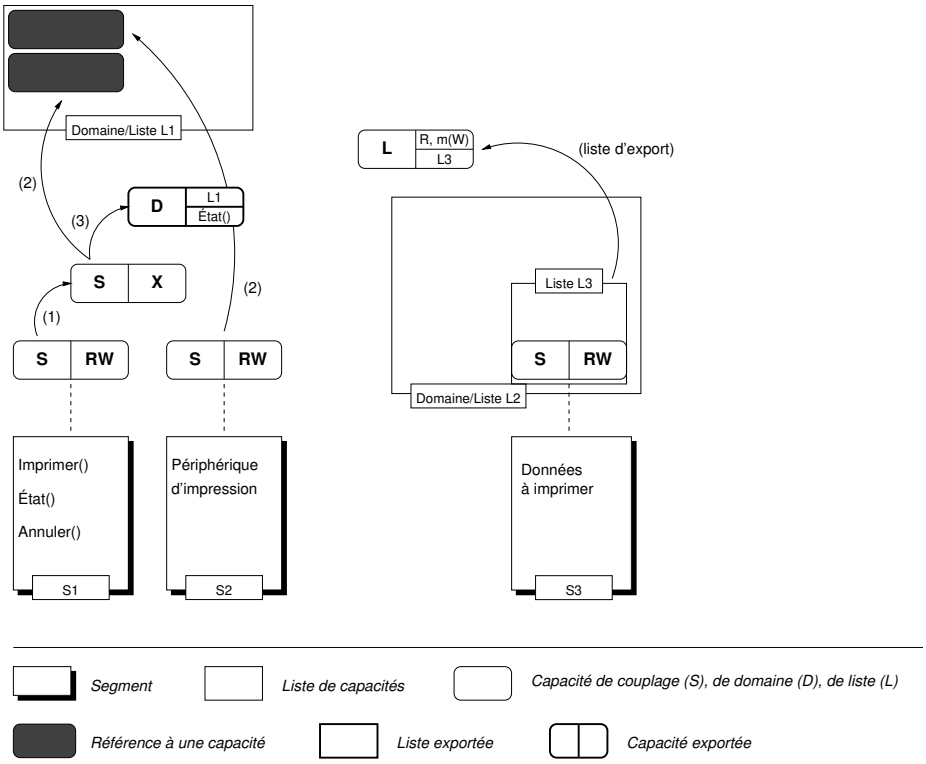


Fig. 2.2 : Construction du domaine de protection d'un serveur d'impression

La construction du domaine de protection du serveur d'impression se déroule alors selon les étapes suivantes :

1. le programmeur du serveur d'impression compile le code des fonctions du service d'impression. Rappelons que l'unité de protection est le segment. La protection souhaitée est la même pour les différentes fonctions Imprimer(), État() et Annuler() ; le code généré par le compilateur peut donc être rangé dans le même segment S1. Le compilateur doit disposer d'un droit en lecture/écriture sur ce segment. Ce droit en lecture/écriture est transformé en droit d'exécution(3) lorsque la compilation est terminée et validée.
2. le programmeur (ou une autre personne – un installateur – à laquelle le programmeur aura fourni la capacité donnant droit d'exécution sur S1) décide alors d'exporter(4) les capacités nécessaires pour créer le nouveau domaine pour le service d'impression. Pour cela, il faut d'abord créer une nouvelle liste de capacités. La création renvoie un identificateur L1. Dès lors, l'installateur peut insérer dans L1 la capacité donnant droit de lecture/écriture sur le

(3) si tant est que l'on puisse distinguer une combinaison lecture et exécution, car certains processeurs tels que le R4000 ne le permettent pas

(4) il convient d'associer la PPI à cette exportation

périphérique d'impression, et la capacité donnant droit d'exécution sur le code des fonctions Imprimer(), État(), etc.

3. le service va être rendu public ! Pour cela, l'installateur construit pour chacune des fonctions Imprimer(), État(),... une capacité de domaine pointant sur L1, et indiquant la fonction Imprimer() ou État(), .. comme point d'entrée dans L1. Les capacités de domaine ainsi créées peuvent être regroupées dans une liste de capacités, et cette liste est exportée à tous via le service de désignation/protection.

De l'autre côté, l'utilisateur récupère par le service de désignation/protection la liste des capacités d'accès au service d'impression, et l'intègre dans la liste L2 qui définit son domaine. Dans cet exemple, la seule restriction possible sur les capacités récupérées est le clonage du domaine d'impression lors de l'appel, et l'origine des capacités délégables pour ce service. L'utilisateur peut définir une liste d'export L3, dans laquelle il insérera toutes les capacités sur les segments contenant des données potentiellement imprimables. La capacité sur la liste L3 permet de restreindre le droit d'écriture pour toutes les capacités contenues par L3, et de refuser toute délégation (niveau de délégation = 1). L3 peut alors être adjointe à la liste d'export du domaine client, ou associée à ce service particulier avec un nom symbolique et une surcharge du style **from** LISTE_DES_IMPRIMABLES de la PPI.

B.2 Une application coopérative

Dans cet exemple, l'application utilisée est un « tableur » réparti, où les cellules sont des segments partagés et protégés individuellement. Pour plus de crédibilité, il conviendrait sans doute de remplacer les cellules par des hyper−documents contenant des liens sur d'autres hyper−documents (en résumé, un « LAN Wide Web »).

Dans un premier temps, nous présentons le schéma final de protection que l'on souhaite obtenir ; puis dans un deuxième nous essayons de le réaliser en utilisant l'interface proposée ci−dessus.

B.2.1 L'objectif

Comme l'indique le schéma Fig. 2.3, l'application dispose de deux types de domaine de protection : ce sont les utilisateurs d'une part, et la partie administrative et privilégiée de l'application, d'autre part. Dans cette configuration, le domaine privilégié dispose des pleins pouvoirs sur l'ensemble des données manipulables par

l'application, c'est pourquoi nous n'envisagerons pas dans ce cas, de mettre en place un IDL. Cet exemple va plutôt illustrer la gestion d'utilisateurs propres à une application.

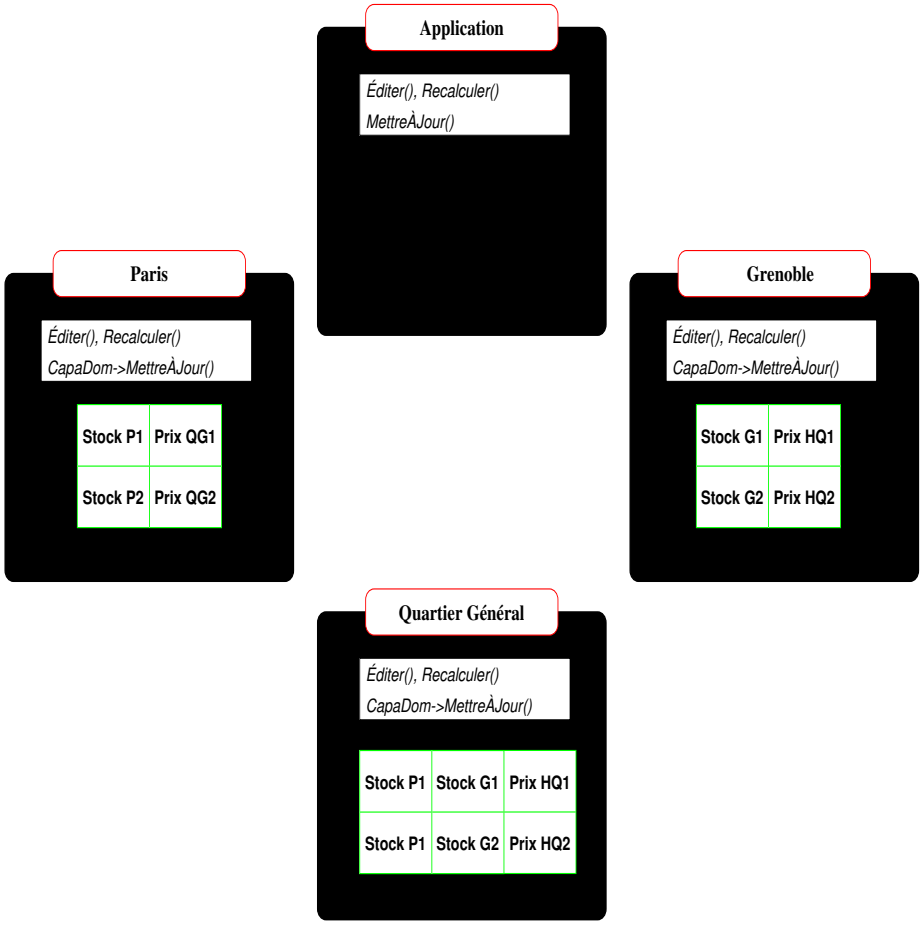


Fig. 2.3 : Schéma de protection de l'application

Domaine client générique (Cf Fig. 2.4)

Les utilisateurs désignés par Paris, Grenoble ou Quartier Général, sont issus d'un domaine client générique. Ce domaine comprend :

- une liste de capacités sur l'ensemble des méthodes offertes par l'application. Certaines méthodes comme Editer() ou Recalculer() peuvent être exécutées localement dans le domaine client.
- une liste de capacités sur les données privées du client (ses tables, et ses cellules). Le client dispose des droits de lecture et écriture sur les segments de ses propres données, et de lecture seule sur les segments contenant les données des autres utilisateurs référencées :

par exemple, Grenoble dispose des droits lecture/écriture sur les segments contenant la définition de ses tables et les valeurs de ses stocks.

Segments/Domaines		Domaine Serveur	Domaine Client Grenoble	Domaine Client Paris
Code privilégié (Mettre À Jour)	Seg.	X	changement de domaine	changement de domaine
	Liste(1)	R non délégable	R	R
Code partagé (Éditer)	Seg.	X	X	X
	Liste(1)	R	R	R
Tables de Grenoble	Seg.	RW	RW	R
	Liste(1)	RW, copie, délégable(1), exportable(R)	RW	R
Tables de Paris	Seg.	RW	R	RW
	Liste(1)	RW, copie, délégable(1), exportable(R)	R	RW

Fig. 2.4 : Domaine client générique

Domaine serveur privilégié (Cf Fig. 2.4)

La mise à jour d'une cellule peut nécessiter la mise à jour d'autres cellules dont elle dépend, et ces cellules appartiennent à un autre utilisateur : à défaut de droit direct d'écriture sur les segments d'un autre utilisateur, on doit recourir à un domaine serveur privilégié qui effectue ces mises à jour.

B.2.2 Mise en place du domaine privilégié

La construction (Fig. 2.5) du domaine de protection privilégié pour l'application se déroule selon les étapes suivantes :

1. le programmeur décide de séparer le code partagé et le code privilégié. Il compile donc le code dans deux segments distincts, S1 et S2, et obtient pour chacun d'eux une capacité avec droit d'exécution.
2. la création du domaine privilégié peut donc d'ores et déjà débuter : le programmeur crée un nouveau domaine identifié par une liste de capacités L1,

et installe dans cette liste L1, les capacités donnant droit d'exécution sur S1 et S2.

3. pour chaque point d'entrée (ici, MettreÀJour()) du domaine L1, le programmeur va rendre "publique" une capacité de domaine. Par ailleurs, le code partagé peut également être rendu accessible en "exportant" directement la capacité donnant droit d'exécution. En fait, ces capacités vont être stockées dans une liste de capacités L', non représentée sur la figure, qui constitue un domaine générique à partir duquel sont dérivés les domaines L2, L3 et L4. Cette dérivation est étudiée plus loin.

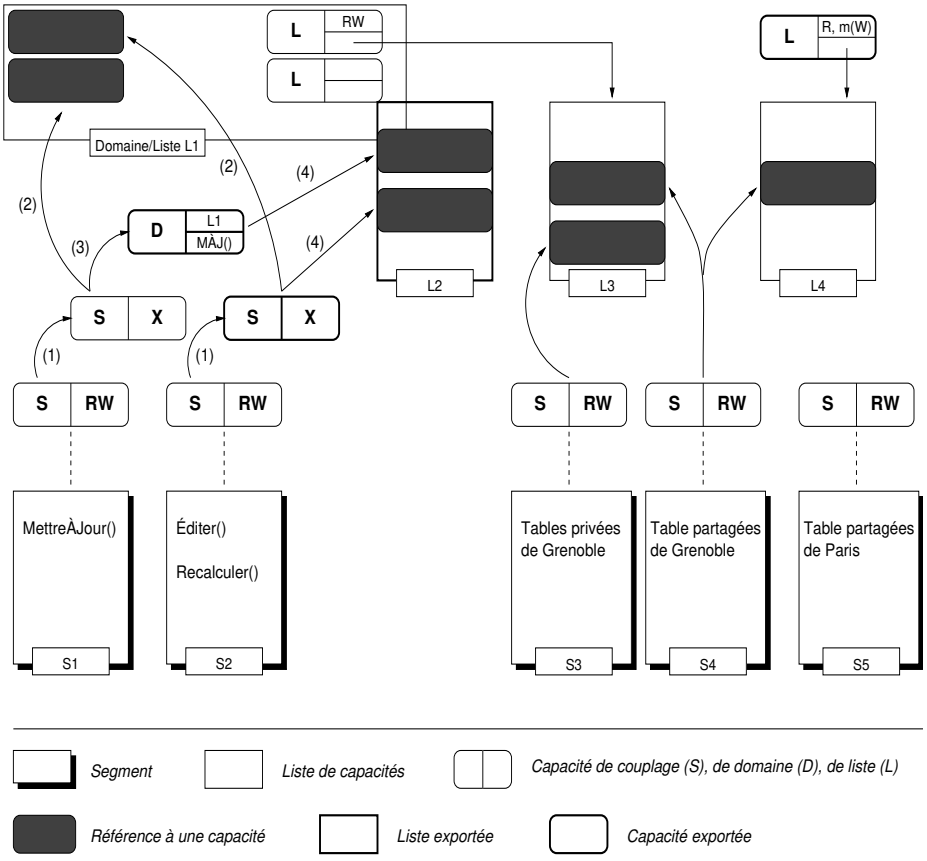


Fig. 2.5 : Structure du domaine serveur

B.2.3 Mise en place d'un domaine client générique

Un domaine client générique est une liste de capacités contenant d'abord la liste des capacités de domaine (MettreÀJour()) et les capacités de couplage pour les segments de code, puis la liste de toutes les listes des autres utilisateurs. Ces listes sont accessibles en lecture seule, sans droit de copie, ni d'insertion.

Outre cette liste générique et partagée, un domaine client doit aussi comprendre la liste de ses propres tables, avec le droit d'insertion/destruction sur cette liste. Le point délicat est la création d'un nouveau client, car il faut :

- créer un nouveau domaine (nouvelle liste L5) avec une copie du domaine « générique » L−génér. défini ci−dessus, et lui ajouter sa propre liste L3 et L4 de capacités pour ses tables.
- modifier le domaine générique, pour lui ajouter la liste L4 nouvellement créée pour ce nouvel utilisateur
- rajouter également dans la liste de tous les autres utilisateurs cette liste L. Ceci est réalisé à travers le partage de la liste L−génér.

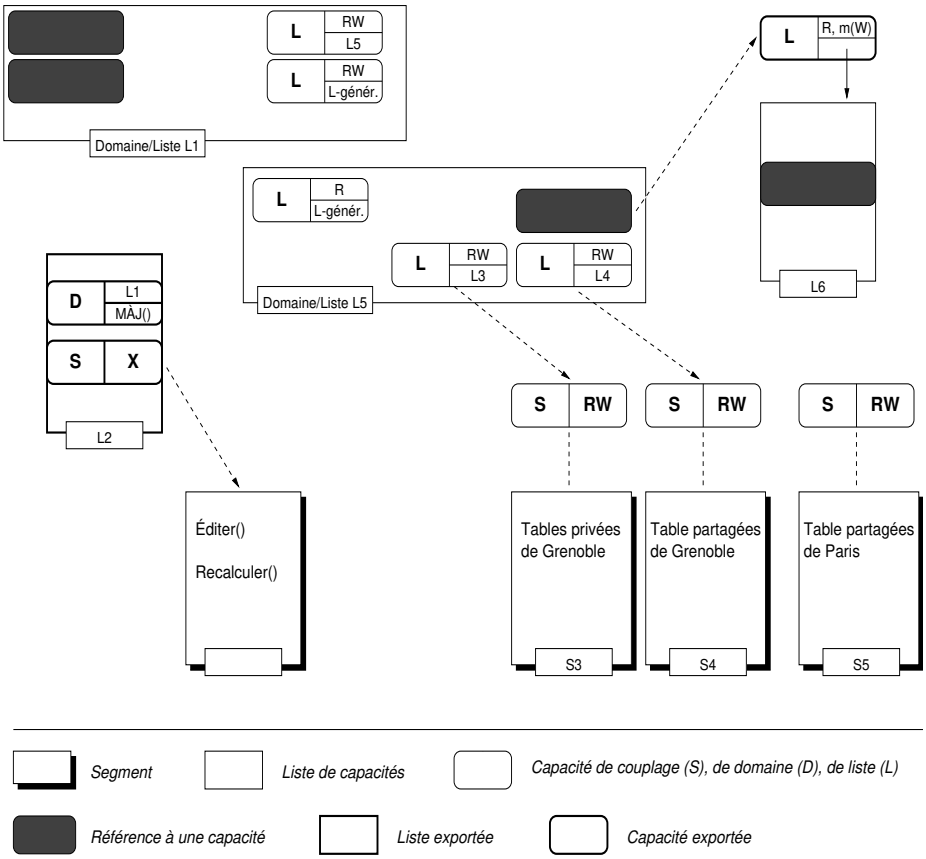


Fig. 2.6 : Structure du domaine client

L'ajout d'un utilisateur est donc un point d'entrée du domaine privilégié. Les arborescences de listes sont :

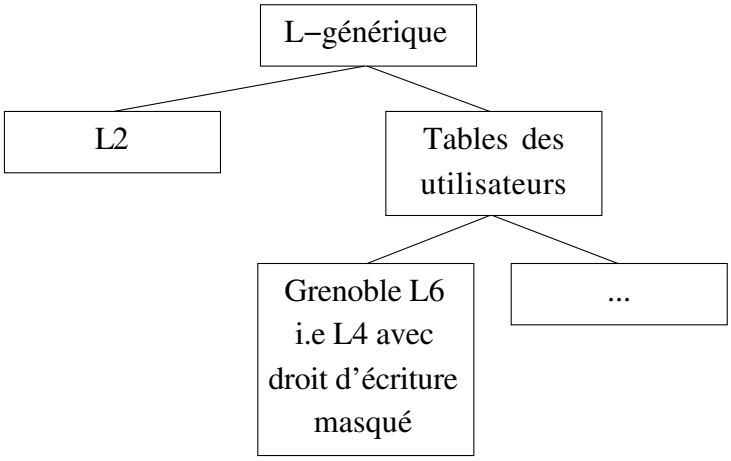


Fig. 2.7 : Vue arborescente du domaine client générique

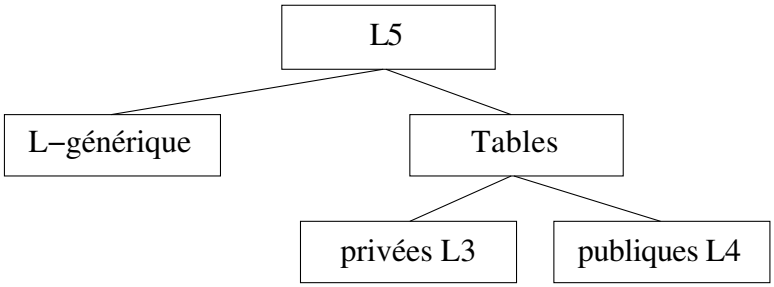


Fig. 2.8 : Vue arborescente du domaine client Grenoble

Bibliographie

[Acetta 86] M.J. Acetta, R. Baron, W. Bolowsky, D. Golub, R. Rashid, A. Tevanian et M. Young, ''Mach: a new kernel foundation for Unix Development'', *Proceedings of the USENIX 1986 Summer Conference*, pp. 93–112, Juillet 1986.

[Anderson 86] M. Anderson, R.D. Pose et C.S. Wallace, ''A Password–Capability System'', *The Computer Journal*, 29(1), pp. 1–8, Janvier 1986.

[Balter 95] R. Balter et S. Krakowiak, *Objectifs et plan de travail du projet Sirac*, (Rapport Technique Sirac 1–95), IMAG / INRIA, Grenoble, juin 1995.

[Bershad 90] B. Bershad, T. Anderson, E. Lazowska et H. Levy, ''Lightweight Remote Procedure Call'', *ACM Transactions on Computer Systems*, 8(1), pp. 37–55, Février 1990.

[Bidan 95] C. Bidan et V. Issarny, *Un aperçu des problèmes de sécurité dans les systèmes informatiques*, (Publication interne n° 959), IRISA, Rennes, Octobre 1995.

[Birrel 84] A.D. Birrel et B.J. Nelson, ''Implementing Remote Procedure Calls'', *ACM Transactions on Programming Languages and Systems*, 2(1), Février 1984.

[Bryce 95] C. Bryce et G. Muller, *Matching Micro–Kernels to Modern Applications using Fine–Grained Memory Protection*, (Broadcast Report 97), IRISA, Rennes, Septembre 1995.

[Chase 92] J.S. Chase, H.M. Levy, M. Baker–Harvey et E.D. Lazowska, *How to Use a 64–Bit Virtual Address Space*, (Rapport Technique n° 92–03–02), University of Washington, Department of Computer Science and Engineering, Seattle, Mars 1992.

[Chase 94] J.S. Chase, H.M. Levy, M.J. Feely et E.D. Lazowska, ''Sharing and Protection in a Single Address Space Operating System'', *ACM Transactions on Computer Systems*, 12(4), pp. 271–307, Novembre 1994.

[Dechamboux 95] P. Dechamboux, D. Hagimont, J. Mossière et X. Rousset de Pina, *Arias : un service de gestion des données persistantes partagées*, (Rapport Technique Sirac 2–95), IMAG / INRIA, Grenoble, octobre 1995.

[Dennis 66] J.B. Dennis et E.C. Van Horn, ‘‘Programming Semantics for Multiprogrammed Computations’’, *Communications of the ACM*, 9(3), pp. 143–155, Mars 1966.

[Fabry 74] R.S. Fabry, ‘‘Capability–Based Addressing’’, *Communications of the ACM*, 17(7), pp. 403–412, Juillet 1974.

[Ferrié 75] J. Ferrié, C. Kaiser, D. Lanciaux et B. Martin, ‘‘An extensible sttructure for protected systems’ design’’, *The Computer Journal*, 19(4), pp. 315–321, 1975.

[Hagimont 96] D. Hagimont, J. Mossière, X. Rousset de Pina et F. Saunier, ‘‘Hidden Software Capabilities’’, *Proc. 16th International Conference on Distributed Computing Systems*, pp. 282–289, IEEE Computer Society, Hong–Kong, Mai 1996.

[Heiser 93a] G. Heiser, K. Elphinstone, S. Russell et G. R. Hellenstrand, *A Distributed Single Address–Space Operating System Supporting Persistence*, (Rapport Technique n° 9302), University of New South Wales, School of Computer Science and Engineering, Kensington, NSW, Australia, 2033, Mars 1993.

[Heiser 93b] G. Heiser, K. Elphinstone, S. Russell et J. Vochteloo, *Mungi : A Distributed Single Address–Space Operating System*, (Rapport technique n° 9314), University of New South Wales, Kensington, NSW, Australia, 2033, Novembre 1993.

[Kougiouris 94] P. Kougiouris et G. Hamilton, *Support for Space Efficient Object Invocation in Spring*, SunSoft Inc., Septembre 1994.

[Koldinger 92] E.J. Koldinger, J.S. Chase et S.J. Eggers, ‘‘Architectural Support for Single Address Space Operating Systems’’, *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS–V)*, pp. 175–186, ACM, Octobre 1992.

[Knuth 73] D.E. Knuth, *Art of Computer Programming : Sorting and Searching*, vol. 3, 1973.

[Lampson 74] B.W. Lampson, ‘‘Protection’’, *Proceedings of the Fifth Princeton Symposium on Information sciences and Systems, Mars 1971, pp 437–443, reprinted in Operating Systems Review*, 8(1), pp. 18–24, Janvier 1974.

[Lampson 76] B.W. Lampson et H.E. Sturgis, ‘‘Reflections on an Operating System Design’’, *Communications of the ACM*, 19(5), pp. 251–266, Mai 1976.

[Levy 84] H. M. Levy, *Capability–based computer systems*, Digital Press, Bedford, Mass., 1984.

[Murray 93a] K. Murray, T. Stiemerling, T. Wilkinson et P. Kelly, ‘‘Angel: Resource Unification in a 64–bit Micro–Kernel’’, *Proceedings of the 27th Hawaii International Conference on Systems Science*, Juin 1993.

[Murray 93b] K. Murray, A. Saulsbury, T. Stiemerling, T. Wilkinson, P. Kelly et P.E. Osmon, ‘‘Design and Implementation of an Object–Oriented 64–bit Single Address Space Microkernel’’, *2nd USENIX Symposium on Microkernels and other Kernel Architectures*, 1993.

[Sansom 86] R.D. Sansom, D.P. Julin et R.F. Rashid, *Extending a Capability Based System into a Network Environment*, (Rapport Technique CMU–CS–86–115), Carnegie Mellon University, Pittsburgh, PA, Avril 1986.

[Saunier 95] F. Saunier, ‘‘Service de Protection d’une Mémoire Virtuelle Répartie’’, *Journées des Jeunes Chercheurs*, Réseau Doctoral en Architecture des Systèmes et des Machines Informatiques, IRISA, Rennes, Octobre 1995.

[Saunier 96] F. Saunier, ‘‘Service de Protection d’une Mémoire Virtuelle Répartie dans SIRAC’’, *Journées sur la Mémoire Partagée Répartie (MPR’96)*, Bordeaux, Mai 1996 (version courte de [Saunier 95]).

[Tanenbaum 86] A.S. Tanenbaum, S.J. Mullender et R. van Renesse, ‘‘Using Sparse Capabilities in a Distributed Operating System’’, *Proceedings of the Sixth International Conference on Distributed Computing Systems*, pp. 558–563, IEEE, 1986.

[Tanenbaum 91] A.S. Tanenbaum, M.F. Kaashoek, R. van Renesse et H. Bal, ‘‘The Amoeba Distributed Operating System – A Status Report’’, *Computing Systems*, 14( ), pp. 324–335, Juillet/Août 1991.

[Vivo 95] M. de Vivo, G. de Vivo et L. Gonzalez, ‘‘A Brief Essay on Capabilities’’, *ACM SIGPLAN Notices*, 30(7), Juillet 1995.

[Vochteloo 93] J. Vochteloo, S. Russell et G. Heiser, *Capability–Based Protection in a Persistent Global Virtual Memory System*, (Rapport Technique n° 9303), University of New South Wales, School of Computer Science and Engineering, Kensington, NSW, Australia, 2033, Mars 1993.

[Wilkinson 94a] T. Wilkinson et K. Murray, *Extensible, flexible and secure services in Angel, a single address space operating system*, (TCU/SARC/1994/4), Systems Architecture Research Centre, City University, Northampton Square, London EC1V 0HB, UK, Novembre 1994.

[Wilkinson 94b] T. Wilkinson et K. Murray, ''Extensible, flexible and secure services in Angel, a single address space operating system'', *Proceedings of the 1st International Conference on Parallel Architectures and Algorithms*, Avril 1995.

[Wulf 81] W.A. Wulf, R. Levin et S.P. Harbison, *HYDRA/C.mmp, an experimental computer system*, McGraw –Hilll, 1981.

Introduction

Chapitre I

Capacités et mémoires réparties partagées

Chapitre II

Capacités et protection de services partagés

Chapitre III

Sirac : présentation, objectifs et particularités

Chapitre IV

Mise en œuvre du service de protection pour Sirac

Chapitre V

Évaluation