Conception et réalisation de noyaux de systèmes répartis : l'exemple du V kernel

Christine Morin

UNIVERSITE DE RENNES I
Campus de Beaulieu
Avenue du Général Leclerc
35 042 RENNES CEDEX

CONCEPTION ET REALISATION DE NOYAUX DE SYSTEMES REPARTIS : L'EXEMPLE DU V KERNEL.

Stage de D.E.A. effectué dans l'équipe Langages et Systèmes Parallèles à l'I.R.I.S.A.

Rapport de stage : Christine MORIN
Directeur de stage : Jean-Pierre BANATRE

Année universitaire 1986-1987

REMERCIEMENTS

Je remercie toutes les personnes de l'équipe Langages et Systèmes Parallèles avec lesquelles j'ai eu le plaisir de passer ces cinq mois dans une ambiance chaleureuse et sympathique.

Je tiens à remercier plus particulièrement Jean-Pierre Banâtre, qui a dirigé mon stage, pour les conseils avisés qu'il m'a donnés tout au long de mon travail.

Mes remerciements vont enfin à Christian Creveuil, Claude Le Maire et Bruno Rochat, mes amis de D.E.A., avec lesquels j'ai partagé mon bureau et qui ont contribué à créer une atmosphère de travail détendue mais néanmoins studieuse.

SOMMAIRE

1. INTRODUCTION

La structure des systèmes informatiques répartis a beaucoup évolué depuis les premiers systèmes répartis spécialisés, apparus à la fin des années 50. Ces premiers systèmes étaient entièrement dédiés à une application telle que la réservation de places d'avion à partir de guichets géographiquement répartis. Plus tard, à partir du milieu des années 70, se sont développés des systèmes répartis construits sur le modèle client/serveur. L'architecture de ces systèmes est constituée de stations de travail et de ressources interconnectées par un réseau local à grand débit. Un serveur, qui gère une ressource, est défini par son interface, qui spécifie les services qu'il fournit et leur mode d'utilisation. Par exemple, l'interface du serveur de fichiers, chargé de conserver les fichiers, comporte entre autres, des opérations de transfert de fichiers. Les clients accèdent aux ressources à partir des stations de travail en utilisant l'interface. Toutes les communications ont lieu entre un client et un serveur.

Vers la fin des années 70, apparaissent des systèmes répartis utilisant les réseaux locaux pour relier des systèmes homogènes, notamment des systèmes UNIX. Le but de ces systèmes est de permettre le partage d'un ensemble de ressources (entre des usagers répartis) sans les placer sous le contrôle d'un serveur unique. Il est même possible dans certains systèmes de ce type de créer et d'exécuter un processus à distance.

Dans ce dernier type de système, les fonctions liées à la répartition sont rajoutées à un système existant. Les premiers systèmes **conçus au départ comme répartis**, qualifiés de sytèmes "intégrés", apparaissent au début des années 80. Ce domaine fait l'objet d'une intense activité de recherche et de nombreux systèmes expérimentaux existent. Ces systèmes répartis sont construits à partir d'un noyau. Un **noyau de système réparti** fournit des fonctions de base telles que la gestion des entités du système (processus, acteurs, objets), de la mémoire, de la communication entre les entités. C'est aussi le noyau qui gère les évènements matériels (interruptions). Les couches supérieures du système réparti, qui offrent les services traditionnellement fournis aux usagers par les systèmes d'exploitation tels que le service de gestion des fichiers, sont construites à l'aide des primitives de base fournies par le noyau. Il existe un exemplaire du noyau sur chaque site. Les couches supérieures du système invoquent sur chaque site l'exemplaire local du noyau. Les noyaux communiquent entre eux et coopèrent pour réaliser les fonctions qui mettent en jeu plusieurs sites.

Les noyaux de systèmes répartis permettent d'adapter les sytèmes répartis à différents types de configurations matérielles. Ils possèdent l'avantage de permettre une conception modulaire des couches supérieures des systèmes répartis.

Le stage a pour objet l'étude des concepts sur lesquels sont fondés les noyaux de systèmes répartis. Cette étude s'appuie sur l'exemple du V Kernel. Ce noyau de système réparti a été conçu à l'université de Stanford (U.S.A.) par l'équipe de D.R. Cheriton. Il fait en quelque sorte référence dans le domaine et peut servir de base de comparaison avec d'autres noyaux. Nous

disposons à l'I.R.I.S.A. d'un exemplaire du V Kernel. La deuxième partie du stage a consisté à installer le V Kernel sur des stations de travail SUN. L'utilisation de ce noyau de système réparti permettra d'évaluer concrètement ses performances et de juger l'intérêt qu'il présente dans la conception et la mise au point d'applications distribuées.

Dans le cadre de la réalisation du système GOTHIC [BAN 86a, BAN 86c] au sein de l'équipe Langages et Systèmes Parallèles en collaboration avec le constructeur BULL, l'intérêt de cette étude est de tirer profit de l'expérience acquise avec le V Kernel, notamment pour la réalisation de la couche transport de GOTHIC.

Le plan de ce rapport est le suivant. Le chapitre 2 présente la structure des noyaux de systèmes répartis. Les mécanismes de communication des noyaux de systèmes répartis sont étudiés dans le chapitre 3. Le chapitre 4 traite les problèmes soulevés en matière de désignation dans les noyaux de systèmes répartis. L'exemple du V Kernel illustre chacun de ces chapitres. Quelques aspects de sa mise en œuvre sont présentés dans le chapitre 5. Comme application, le service de gestion de fichiers, point clé de tout système d'exploitation, est étudié dans le chapitre 6. Nous tirons quelques conclusions dans le chapitre 7.

2. STRUCTURE

Dans la conception d'un noyau de système réparti, il est fondamental de définir quelles sont les entités gérées par le noyau et l'ensemble des primitives que celui-ci offre pour la gestion de ces entités et la communication entre elles. Nous étudions dans ce chapitre les différents modèles de structure offerts par les noyaux de systèmes répartis. Dans un premier paragraphe, nous nous intéressons aux noyaux de systèmes répartis à messages. Les premiers noyaux de systèmes répartis sont de ce type. Par la suite, ce sont développés des noyaux de systèmes répartis construits autour du concept d'objet. Ces noyaux de systèmes répartis sont traités dans le paragraphe 2.2. Nous décrivons le modèle de la structure du V Kernel dans le paragraphe 2.3. Il s'avère que des outils d'abstraction de haut niveau facilitent la structuration des applications distribuées. Le paragraphe 2.4 est consacré au concept de groupement d'entités dans les noyaux de systèmes répartis. Enfin, nous terminons par une discussion. Les problèmes de communication ne sont abordés qu'à partir du chapitre 3.

2.1. Les noyaux de systèmes répartis à messages

D'une façon générale, il existe dans les noyaux de systèmes répartis à messages deux types d'entités : des entités actives et des entités passives.

Les entités actives sont connues sous le terme de **processus** dans la plupart des noyaux de systèmes répartis. Dans le système Chorus [LEG 86], développé à l'INRIA, le terme d'**acteur** est employé pour nommer les processus. Les processus exécutent un traitement séquentiel

au cours duquel ils peuvent être amenés à communiquer avec d'autres processus afin d'échanger des données ou de se synchroniser. Il y a alors échange d'un **message** soit directement entre les processus soit indirectement par l'intermédiaire de **portes**. Le noyau Accent [FIT 85, RAS 81], développé à l'université de Carnégie Mellon et Chorus utilisent le terme de porte ; le système Apollo/Domain développé à Chelmsford [LEA 83, LEV 86] emploie le terme de socket. Une porte est une file protégée par des droits d'émission et de réception destinée à contenir les messages émis par les processus qui possèdent le droit d'émission sur la porte et qui sont prélevés par le processus qui possède le droit de réception sur la porte. A tout moment, un seul processus peut posséder le droit de réception associé à la porte.

Les entités passives sont les fichiers, les segments en mémoire. Elles ne sont généralement pas gérées directement par le noyau. Dans les systèmes construits sur le modèle client/serveur, les entités passives sont gérées par des processus dits processus serveurs qui traitent les requêtes qui leur sont adressées par les autres processus dits processus clients. Dans le modèle "intégré", les entités passives sont gérées par un sous-système construit à l'aide des primitives offertes par le noyau.

En résumé, les noyaux de systèmes répartis à messages utilisent deux concepts de base : le **processus** et le **message**. Certains d'entre eux utilisent aussi le concept de porte. Pour illustrer les concepts présentés ci-dessous, nous décrivons brièvemennt la structure du noyau du système Chorus. La présentation de la structure du V Kernel fait l'objet du paragraphe 2.3.

Les éléments de base de Chorus sont les acteurs, les portes et les messages.

Un **acteur** est sensiblement l'équivalent d'un processus séquentiel. Il contient du code, des données et un contexte d'exécution. Les acteurs sont structurés en étapes de traitement. L'exécution d'une étape de traitement est déclenchée par la réception d'un message sur une porte associée à l'étape de traitement considérée. Un acteur peut définir des sélections, des aiguillages et des temporisations. Les sélections sont des liaisons représentées par des couples (porte émettrice, porte réceptrice locale). Les sélections permettent à l'acteur de sélectionner les émetteurs des messages qu'il peut recevoir. Les aiguillages sont des couples (porte, étape de traitement). Un aiguillage indique quelle étape de traitement sera effectuée lors de l'arrivée d'un message sur une porte. Une temporisation permet à un acteur de limiter le temps d'attente d'un message sur une porte. Les sélections, les aiguillages, les temporisations sont gérées par le noyau.

Les **portes** sont associées dynamiquement aux acteurs. Une porte appartient à un seul processus à la fois et peut être soit ouverte, soit fermée. Une porte ouverte est active. Le processus qui a ouvert la porte possède le droit de réception sur la porte. Les processus qui possèdent le droit d'émission sur une porte peuvent envoyer des messages sur cette porte. Une porte fermée est inactive. Elle ne peut pas être utilisée. Le noyau gère la protection des portes. Chorus offre le concept de groupe de portes. A chaque porte est associée la liste des usagers avec leurs droits respectifs. Chorus offre la notion de groupe de portes. Un groupe de portes est une liaison logique sur un ensemble de portes. Les portes d'un groupe peuvent être réparties sur plusieurs sites. Le rôle des portes de Chorus est de permettre des mécanismes d'adressage plus puissants que le point à point

déterministe offert par les portes.

La gestion des portes et groupe de portes est réalisée dans Chorus par le service réparti gestionnaire des portes et groupes de portes. Ce service est représenté par un serveur sur chaque site, qui épargne au noyau la complexité de gestion de ces entités. Le noyau ne connait que les portes ouvertes par les acteurs de son site. La création, destruction, migration, localisation des portes sont réalisées par les serveurs de portes et groupes de portes. Ces serveurs coopèrent avec le noyau pour l'ouverture et la fermeture des portes.

Le troisième concept de base dans Chorus est le message qui est un ensemble de données qui sort de l'espace d'adressage de l'émetteur et rentre dans l'espace d'adressage du récepteur.

2.2. Les noyaux de systèmes répartis à objets

Dans les noyaux de systèmes à objets, le composant élémentaire géré par le noyau est l'objet. Un objet regroupe une structure de données et les procédures d'accès à cette structure. Les objets qui ont un comportement identique font partie d'une même classe. Il est possible de créer dynamiquement un exemplaire d'un objet d'une classe donnée, c'est l'instanciation. La notion d'objet avait été préalablement introduite dans les langages Simula 67 et SmallTalk. C'est un outil de haut niveau permettant de structurer les applications de façon modulaire. La programmation par objets homogénéise les notions de données et de processus (entité active) en une seule : l'objet.

Plusieurs noyaux de systèmes répartis mettent en œuvre ces concepts et peuvent donc être qualifiés de noyaux de systèmes à objets : citons, par exemple, les noyaux de système Eden [BLA 85] et Emerald [BLA 86] développés à l'université de Washington (Seattle USA), le système SOS (Somiw Operating System) développé à l'INRIA [SHA 87], le système GUIDE (Environnement Distribué Intégré des Universités de Grenoble) [BAL 86].

Nous décrivons à titre d'exemple la structure offerte par le noyau GUIDE.

Les entités gérées par le noyau GUIDE sont les objets ordinaires, les domaines et les activités.

Tout **objet** appartient à une classe, définie par l'association d'un type abstrait et d'un type concret qui le réalise. Un objet n'est accessible qu'à travers un ensemble de procédures d'accès définies par sa classe. Une opération particulière, l'instanciation, est associée à chaque classe et permet de créer un objet de la classse (appelé instance).

Les **domaines** et les **activités** sont aussi des objets mais d'un type spécial puisqu'ils appartiennent à une classe prédéfinie. Un domaine est la réunion d'un ensemble d'objets et d'un ensemble de processus qui opèrent sur ces objets. Les processus d'un domaine sont appelés activités. L'ensemble des objets ordinaires (ie. qui ne sont ni des domaines, ni des activités) qui font partie d'un domaine à un instant donné, ou contexte, peut changer au cours du temps car les liaisons entre un domaine et un objet sont dynamiques. L'éxécution d'une activité dans un domaine consiste en appels successifs aux procédures des objets du domaine.

2.3. Le modèle du V Kernel

Nous étudions dans ce paragraphe la structure de V Kernel. Le V Kernel est un noyau de système réparti dans lequel les processus et groupes de processus communiquent par échange de messages [ZWA 85]. Il y a un exemplaire du noyau sur chaque site. Les différents exemplaires du noyau coopèrent selon le protocole internoyaux (InterKernel Protocol) (cf. figure 1). Le noyau gère les processus et groupes de processus ; il assure la communication entre les processus et entre un processus et un groupe de processus. L'étude de la désignation des entités gérées par le noyau est abordée dans le chapitre suivant. Les autres fonctions du système telles que le système de gestion de fichiers sont des applications construites au dessus du noyau.

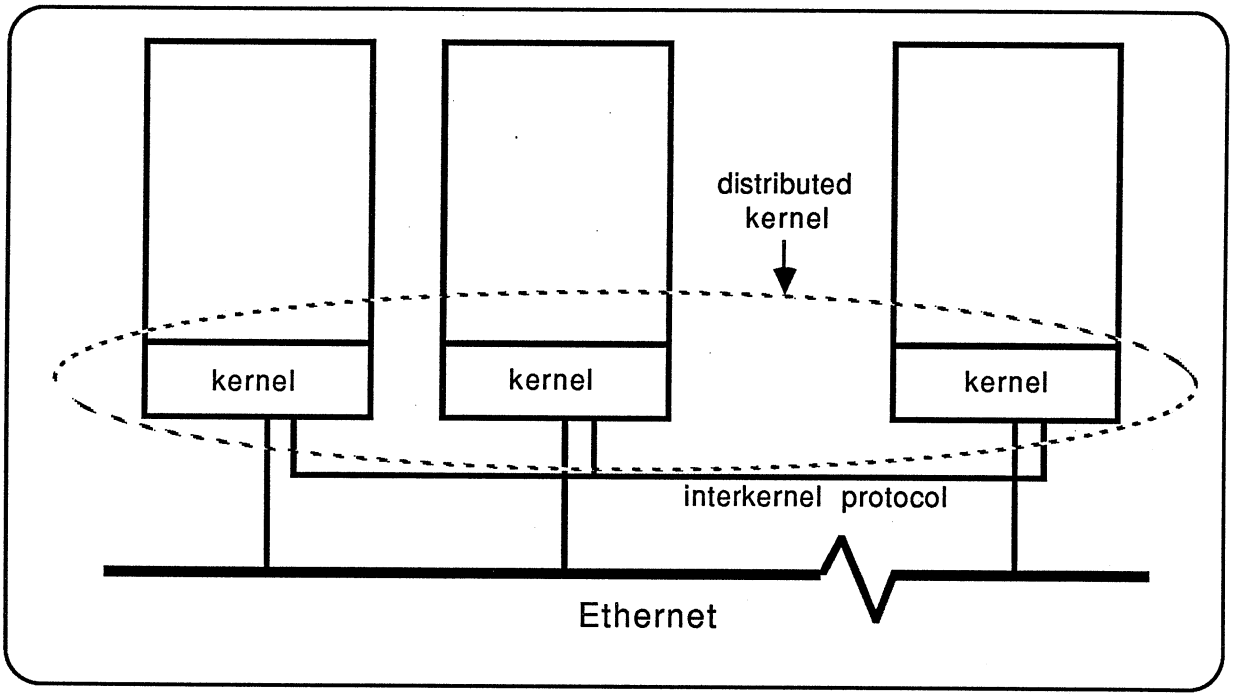

Figure 1 : Le noyau distribué.

Les entités actives du V Kernel sont les processus et groupes de processus [CHE 85a]. Un groupe de processus est un ensemble de un ou plusieurs processus qui peuvent être localisés sur des sites différents. Les membres d'un groupe ne sont pas discernables, il n'y a pas de hiérarchie interne. Les groupes de processus sont créés dynamiquement. Ils peuvent être soit à accès réglementé, soit à accès non réglementé. Tout processus peut se joindre à un groupe à accès non réglementé tandis que seuls les processus possédant une autorisation peuvent devenir membres d'un groupe à accès réglementé. L'intérêt des groupes à accès non réglementé est limité par le fait que n'importe quel processus peut se joindre à un tel groupe et perturber son fonctionnement normal en envoyant des réponses erronnées aux requêtes adressées au groupe.

Les groupes du V Kernel sont des groupes ouverts. Un groupe ouvert est un groupe

auquel toute entité communicante, qu'elle soit membre du groupe ou pas peut envoyer un message. Cette notion s'oppose à la notion de groupe fermé. Un groupe fermé est un groupe auquel seuls les membres du groupe peuvent envoyer des messages.

Les opérations sur les groupes ne sont pas fiables. La fiabilité des applications qui mettent en jeu des groupes de processus doit être assurée par des protocoles adaptés à l'application.

Les primitives ainsi que leur mise en œuvre sont présentées dans le chapitre 5.

2.4. Le groupement d'entités

Nous venons de voir que dans la structure des noyaux de systèmes répartis à messages est introduit le concept de groupe : groupe de processus dans le V Kernel, groupe de portes dans Chorus. Dans ce paragraphe, nous nous intéressons plus particulièrement à ce concept. Dans un premier temps, nous dégageons l'intérêt des groupes de processus et de portes tels qu'ils apparaissent dans les noyaux de systèmes répartis et donnons des exemples de leur utilisation possible. Nous abordons ensuite des outils de structuration de plus haut niveau fondés également sur le concept de groupe.

1. Groupes de processus et groupes de portes

Nous mettons en évidence l'intérêt que présentent les groupes de processus et groupes de portes. Des exemples appuient notre propos.

Les applications, dans les systèmes distribués, mettent souvent en jeu un ensemble de processus coopérants. Il est donc intéressant de disposer d'outils tels que les groupes de processus qui permettent de structurer les applications en tenant compte des interactions entre plusieurs processus.

Les membres d'un groupe peuvent être situés sur des sites distincts. De plus, l'ensemble des membres d'un groupe peut évoluer dynamiquement, c'est à dire qu'à tout moment, un membre peut quitter le groupe, un nouveau membre peut arriver dans le groupe ou un membre du groupe peut migrer. Le caractère dynamique des groupes est intéressant pour construire des services ou des applications résistants aux pannes. Prenons l'exemple du service d'impression. Supposons que ce service est fournit par plusieurs serveurs répartis sur plusieurs sites différents. Si une des imprimantes est arrêtée pour une opération de maintenance, le serveur qui la gère est retiré du groupe. Il est réintégré lorsque l'imprimante est remise en service.

Les groupes offrent essentiellement deux modes d'adressage : la diffusion et le mode d'adressage fonctionnel.

Lors d'une diffusion, le message est envoyé à tous les membres du groupe. Ce mode d'adressage est utile pour appliquer, par exemple, une même opération à tous les membres du groupe. D'autre part, il est intéressant pour implanter des services répartis qui nécessitent la diffusion de messages tels que le courrier électronique. Par exemple, le système Grapevine [BIR 82] possède la

notion de groupe symbolique. Un groupe symbolique est un ensemble de noms symboliques. Pour le service du courrier électronique, les usagers font partie de groupes (groupe des professeurs de licence, groupe des professeurs de maîtrise ...). Les messages sont diffusés sélectivement aux groupes d'usagers. Dans les services répartis tels que le service gestionnaire des portes dans Chorus, qui ont à gérer des entités qui peuvent migrer, la diffusion est utile dans les algorithmes de localisation. La diffusion est également un mode d'adressage intéressant pour la mise en œuvre de protocoles gérant des entités répliquées (bases de données réparties).

Les groupes associés à la diffusion permettent de mettre en øuvre des protocoles palliant à l'absence de mémoire commune dans les systèmes répartis. Les groupes de processus du V Kernel permettent un adressage 1-N avec M réponses (M<N) que ne permettent pas les groupes de portes de Chorus. Les groupes de processus du V Kernel semblent donc plus intéressants pour mettre en øuvre les protocoles de diffusion fiable que les groupes de portes de Chorus.

Dans le mode d'adressage fonctionnel, un message adressé à un groupe n'est envoyé qu'à un seul membre du groupe choisi arbitrairement. Ce mode d'adressge est utile pour construire des services composés de serveurs ayant tous le même comportement (service d'impression) ou gérant le même objet (base de données localisée sur un site avec plusieurs serveurs permettant d'y accéder).

Les groupes permettent de dissimuler aux clients la structure interne d'un service. En effet, pour adresser un message à un groupe, il n'est pas nécessaire de connaître le nombre de membres du groupe ni la liste des membres du groupe (ie. le nom de chacun des membres).

2. Autres propositions

Plusieurs propositions d'outils de haut niveau pour la structuration d'applications distribuées sont en cours de validation. Des noyaux de systèmes répartis fournissant la base nécessaire à la construction de ces outils sont ou vont être expérimentés. Les noyaux de systèmes à objets fournissent déjà un outil de structuration de haut niveau : l'objet. Nous décrivons trois de ces propositions : la notion de "guardian" du système Argus [LI84], la notion de "team" du système Raddle [FOR 86] et enfin la notion de multifonction du système Gothic [BAN 86b].

1. Les "guardians"

Dans le système Argus [LIS 84], fondé sur la notion de Guardians et d'action atomique, une application est décrite comme un ensemble de guardians. Les guardians sont des groupes de processus qui partagent des données communes. Les guardians communiquent entre eux par appel de procédure à distance. Les paramètres sont transmis par valeur, ce qui assure la protection des données des guardians. Ce mécanisme de communication est décrit dans le chapitre suivant. Pour faire face à la panne d'un site, le système Argus introduit les actions atomiques qui sont des actions qui se déroulent normalement ou n'ont pas d'effet de bord dans le cas contraire.

2. Les "teams"

Dans le système Raddle [FOR 86], il est possible de faire de la programmation parallèle à l'aide d'objets appelés "teams". Une application est décrite par des teams. Une team est composée d'un ensemble de données et de roles. Les roles sont de deux types : les roles marqués et les roles non marqués. Une team peut être vue comme un objet. Dès l'instanciation de la team, les roles marqués ou processus commencent leur exécution. Les roles non marqués sont en fait des procédures réentrantes qui sont disponibles pour tout role (même à l'extérieur de la team où le role non marqué est défini) et permettent d'accéder aux données de la team. Le comportement des processus est décrit par des règles qui sont des sortes de commandes gardées dont la forme générale est : condition & interaction -> liste de commandes. Les commandes ne sont effectuées que lorsque la condition est vérifiée et que tous les processus intervenant dans l'interaction sont prêts. Une interaction est une généralisation du rendez-vous à N processus. Les interactions sont atomiques.

3. Les multifonctions

Les multifonctions [BAN 86b] sont également des outils pour structurer les applications qui font intervenir des processus coopérants. Le concept de multifonction orthogonalise la notion d'activité introduite dans le système Enchère [BAN 84]. Une activité est un ensemble de processus coopérants dont les interactions caractérisent la structure dynamique de la tâche considérée. Les applications sont décrites par un arbre d'activités. Deux activités mère et fille ne communiquent qu'à la création et à la terminaison de l'activité fille par appel de procédure et retour. Les activités sont atomiques. Elles sont indivisibles, les états intermédiaires ne sont pas observables. En cas de défaiilance, les objets modifiés peuvent être restaurés dans leur état initial. De la même façon qu'une activité est l'abstraction du bloc, la multifonction est l'abstraction de la proposition parallèle. Le corps d'une multifonction [BAN 86a] peut être constitué de plusieurs composants qui s'éxécutent en parallèle. L'exécution d'une multifonction est atomique, les multifonctions peuvent être imbriquées.

2.5. Discussion

Le concept de groupe fourni par un noyau de système réparti à messages ou le concept d'objet fourni par un noyau de système réparti à objets sont intéressants pour structurer les applications distribuées. La gestion des objets et des groupes en environnement réparti est complexe. La fiabilité des applications et la résistance aux pannes doivent être assurées. Le problème est de savoir à quel niveau ces propriétés doivent être traitées : dans le noyau ou dans les sous-systèmes construits au dessus du noyau. Lorsque plusieurs processus utilisent et mettent à jour des données partagées, il faut que celles-ci restent cohérentes même en cas de panne. Le concept d'action atomique est un outil de contrôle des accès concurrents à des données partagées. Une action atomique est indivisible c'est à dire que ses états intermédiaires ne sont pas observables et elle est effectuée en totalité ou pas du tout (c'est à dire qu'il n'y a pas d'effet de bord dans le cas où son exécution est abandonnée ; cela suppose que les modifications effectuées doivent pouvoir être annulées). Une

transaction est un ensemble d'actions atomiques. Pour assurer la propriété de recouvrement, il faut disposer d'une mémoire stable. Les transactions sont en général mises en œuvre par un protocole à deux phases. Une fois que la première phase est validée (commitment), le protocole assure la terminaison de la transaction. Le V Kernel ne propose aucun mécanisme pour l'atomicité. A mon avis, il serait intéressant de fournir au niveau du noyau des opérations de base telles qu'une primitive de recopie d'un objet en mémoire stable ou une primitive de validation qui permettraient de faciliter la mise en œuvre, à un niveau supérieur, des protocoles assurant l'atomicité. Les concepteurs du système Argus semblent aller dans cette direction pour la définition du noyau Argus.

Dans le V Kernel, la notion de groupe de processus cohabite avec la notion de processus. Les noyaux de systèmes répartis à objet vont plus loin en encapsulant les processus dans le concept d'objet.

3. COMMUNICATION

Nous nous intéressons à la communication dans les noyaux de systèmes répartis. La communication permet de partager des données et d'établir des relations de coopération entre les entités (processus, acteurs, activités) qui participent à une même application. Il existe plusieurs modes de communication dans les noyaux de systèmes répartis ; ils correspondent à un niveau d'abstraction croissant : communication par messages, par appel de procédure.

Nous étudions la communication par messages dans le paragraphe 3.1. Le paragraphe 3.2. est consacré à la communication par appel de procédure. La communication dans le V Kernel fait l'objet du paragraphe 3.3. Nous terminons par une discussion.

3.1. La communication par messages

Les premiers systèmes répartis ont adopté le modèle de la communication par messages. La communication entre deux processus (nous utilisons ce terme dans un sens général) est définie par deux opérations primitives d'échange de messages qui sont :

envoyer (message, destinataire)
recevoir (message, origine).

La désignation des processus correspondants peut être directe ou indirecte.

Si la désignation est directe, le nom des processus destinataire et origine est indiquée dans les primitives. Les primitives d'échange de messages dans le V Kernel sont de ce type.

Si la désignation est indirecte, les processus utilisent le nom d'un objet intermédiaire

dans les primitives. Dans Chorus [LEG 86] et dans le noyau Accent [RAS 81], l'objet intermédiaire est la porte ; dans le système Apollo/Domain c'est le "socket".

Dans certains noyaux de système, la primitive recevoir peut être de la forme recevoir (message). Le récepteur ne précise pas l'origine des messages qu'il attend. Le V Kernel possède une primitive recevoir de chaque forme.

Dans les noyaux qui offrent le concept de groupe, le destinataire et l'origine d'un message peuvent être le nom d'un groupe.

Un message possède une structure imposée par l'émetteur qui doit être connue du récepteur.

Plusieurs modes de synchronisation peuvent être définis pour les primitives de communication. L'opération de réception est par nature bloquante. Par contre, l'opération d'émission peut être bloquante (échange synchrone) ou non bloquante (échange asynchrone). Nous traitons les deux modes de synchronisation.

1. échange synchrone

La primitive envoyer bloquante ne rend le contrôle au processus qui l'a invoquée que lorsque le message a été envoyé (communication non fiable) ou lorsque le message a été envoyé et acquitté. Dans ce dernier cas, il y a rendez-vous.

2. échange asynchrone

Dans ce cas, l'échange se fait à travers un tampon. La primitive envoyer rend le contrôle au processus dès que le message a été recopié dans un tampon ou mis dans une file d'attente.

Les échanges de messages dans le noyau Accent (implanté sur une seule machine mais mettant en œuvre les concepts liés à la répartition) sont asynchrones [FIT 85] et s'effectuent par l'intermédiaire de portes. Le noyau fournit des capacités, droit de réception, droit d'émission aux processus pour leur permettre respectivement de lire ou écrire un message sur une porte. Le droit de réception n'est pas partagé. Les échanges sont asynchrones. Lors de l'envoi d'un message sur une porte par un processus A, le message qui est dans l'espace virtuel du processus A est mis dans un tampon de l'espace virtuel du noyau. Il n'y a pas de recopie physique du message. Lorsqu'un processus B invoque la primitive de réception, le message est transféré par le noyau dans l'espace virtuel du processus B (toujours sans recopie physique). Après la réception du message, le message est partagé entre le processus A et le processus B. Le noyau Accent possède un mécanisme original qui assure que le message délivré au processus B est bien celui émis initialement par le processus A malgré l'asynchronisme de l'échange. Pour cela, le message est mis dans l'état "copier sur écriture". Ainsi, si entre l'émission et la réception, le processus A modifie le message, celui-ci (tout au moins la partie modifiée de celui-ci) sera recopié par le noyau dans une autre zone physique de façon à laisser le message dans son état initial pour le récepteur. Le lecteur intéressé trouvera une analyse plus détaillée de ce mécanisme et une discussion sur son utilisation possible pour la gestion de la mémoire virtuelle dans un système distribué dans [ROC 87].

3.2. L'appel de procédure

Dans les noyaux de systèmes répartis orientés objet, les processus utilisent l'appel de procédure pour manipuler les structures de données à l'aide des procédures d'accès. Prenons quelques exemples. Dans le noyau GUIDE, les activités d'un domaine appellent les procédures des objets du domaine. Les guardians d'Argus communiquent entre eux par appel de procédure. Deux activités mère et fille du système Enchère communiquent par appel de procédure. Dans les teams de Raddle, les roles marqués sont des processus qui peuvent faire appel aux roles non marqués qui sont des procédures réentrantes. Il s'agit d'une forme d'appel de procédure. Dans le système Eden, un objet est une entité active constituée d'un ensemble d'activités qui communiquent à l'aide de variables partagées et se synchronisent à l'aide de moniteurs.Les objets de EDEN communiquent par des messages qui sont des requêtes d'exécution d'un traitement sur l'objet destinataire (forme d'appel de procédure à distance).

Dans chacun de ces exemples, lorsque l'entité appelante et la procédure sont sur le même site, il s'agit d'appel de procédure simple. Lorsque l'entité appelante et la procédure sont sur des sites distincts, il s'agit d'appel de procédure à distance.

L'appel de procédure à distance est l'appel de procédure simple habituel dont la mise en œuvre est différente. L'appel de procédure simple est exécuté sur un site unique. Par contre, pour l'appel de procédure à distance, la procédure appelante et la procédure appelée s'exécutent sur des sites distincts. De la même façon qu'un appel simple donne le contrôle et les données à l'intérieur d'un programme, l'appel de procédure à distance transfère le contrôle et les données à travers le réseau de communication [BIR 84]. Le processus qui exécute un appel de procédure à distance transmet au site appelé le nom de la procédure et ses paramètres et reste bloqué jusqu'au retour. Sur le site appelé, un processus exécute l'appel et renvoie ses résultats à l'appelant (cf. figure 2).

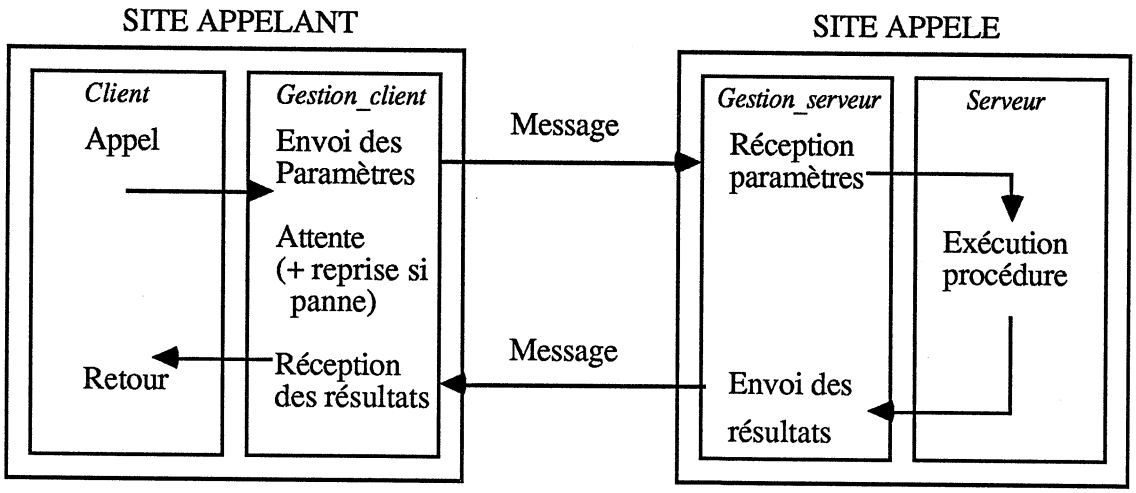

Figure 2 : Appel de procédure à distance.

L'appel de procédure à distance est synchrone c'est à dire que le contrôle est rendu au

processus appelant uniquement au retour de la procédure appelée. L'appel synchrone garantit une structure d'exécution simple et uniforme. L'appel de procédure à distance possède l'avantage d'avoir une sémantique simple en l'absence de défaillance, celle de l'appel de procédure simple par valeur. Toutefois, la prise en compte du traitement des défaillances pose des problèmes délicats de réalisation. Il y a plusieurs réalisations possibles : en cas de terminaison normale de l'appel de procédure à distance, l'appelant est assuré suivant le cas que soit la procédure a été exécutée au moins une fois, soit elle a été exécutée une fois et une seule, soit elle a été exécutée au plus une fois avec, dans le cas où la procédure n'a pas été exécutée, restauration de l'état avant l'appel. Dans les deux premiers cas, le comportement en cas de défaillance n'est pas bien défini. Dans le cas d'une défaillance du processus appelant, il faut être capable de détecter cette situation et alors de détruire les processus appelés qui exécutent l'appel, nommés "orphelins", et de restaurer les données modifiées dans leur état avant l'appel. Dans Argus, un protocole de détection (construit au dessus du noyau) des orphelins est proposé. Il est fondé sur des informations complémentaires véhiculés dans les messages ou conservées dans chaque guardian.

Dans la réalisation de l'appel de procédure à distance se pose également le problème du passage des paramètres par référence, plus complexe que l'appel par valeur.

L'appel de procédure à distance peut se généraliser à l'appel de multifonctions.

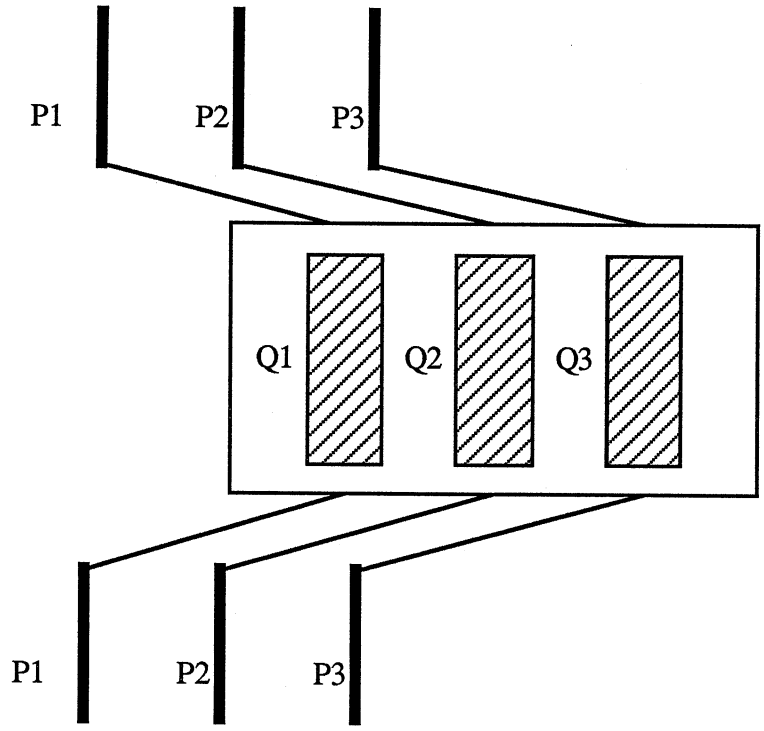

Figure 3 : Appel coordonné de multifonction.

La forme la plus générale étant l'imbrication de deux multifonctions, les processus P_i s'exécutent en parallèle et se synchronisent pour la transmission des paramètres aux processus Q_j. Les processus Q_j s'exécutent en parallèle. Ils se synchronisent pour la construction du résultat et la

transmission. Le résultat est distribué aux composants P_i de l'appelant qui finalement reprennent leur exécution (cf. figure 3).

3.3. La communication dans le V Kernel

Nous étudions dans ce paragraphe la communication dans le V Kernel. Les processus et groupes de processus communiquent par échange de messages.

La communication entre processus, du type requête/réponse, fait intervenir trois primitives : Send, Receive, Reply. Les primitives Send et Receive sont bloquantes, la primitive Reply est non bloquante. Un échange de messages entre deux processus se déroule de la manière suivante : l'émetteur envoie à l'aide de la primitive Send un message au récepteur et se bloque en attente d'une réponse. Le récepteur, qui était bloqué, en attente d'un message par exécution de la primitive Receive reçoit le message : il y a Rendez-Vous entre l'émetteur et le récepteur du message. Le récepteur envoie alors un message de réponse à l'aide de la primitive Reply immédiatement ou après un traitement (cas d'un message de requête reçu par un serveur) et continue son exécution (il n'y a pas Rendez-Vous). La réception du message de réponse a pour effet de débloquer l'expéditeur (cf. figure 4).

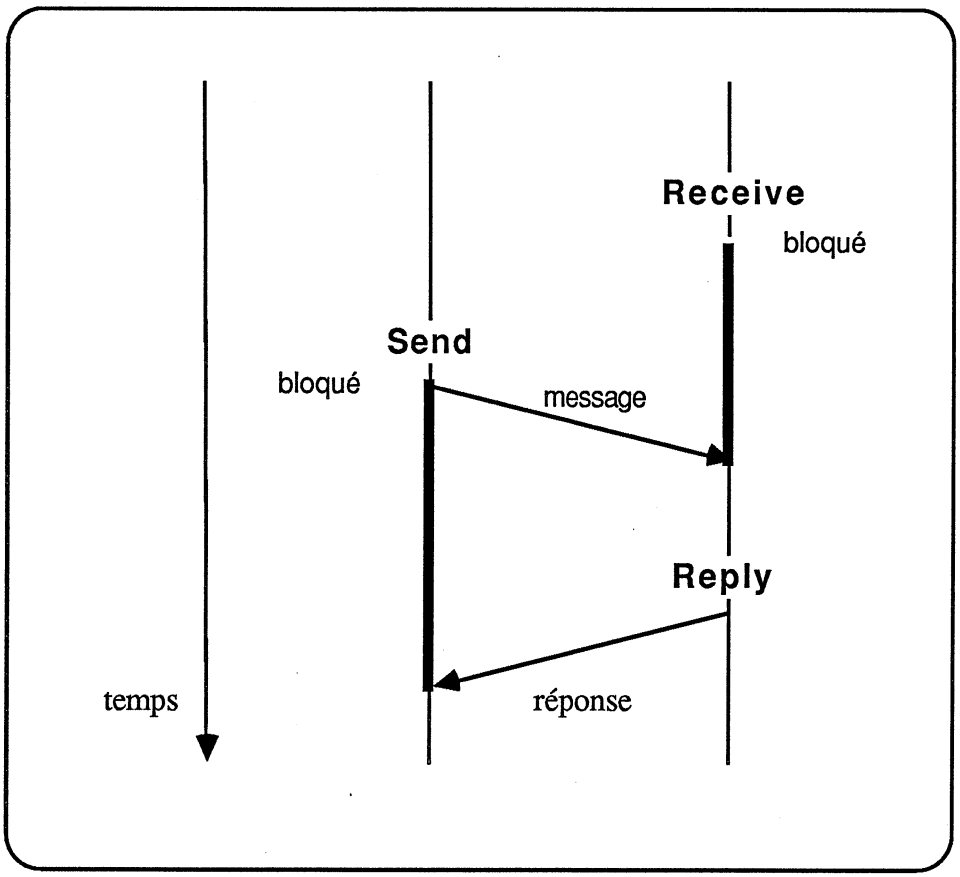

Figure 4 : Un échange de messages.

La synchronisation des processus n'est pas globalement de type Rendez-Vous. En effet, si le processus récepteur "plante" entre le moment où il a reçu le message et avant qu'il invoque la primitive Reply, l'émetteur du message recevra un message d'erreur. Le principe d'atomicité du Rendez-Vous n'est pas respecté puisqu'aucun mécanisme n'est prévu pour annuler les modifications effectuées par le processus récepteur entre l'arrivée du message et le moment de la défaillance.

L'envoi d'un message à un groupe est similaire à l'envoi d'un message à un processus. Le nom du groupe de processus est donné comme paramètre des primitives de communication à la place du nom du processus. La primitive Send a pour effet d'envoyer le message à tous les membres du groupe. Si l'expéditeur est membre du groupe, il ne reçoit pas de copie de son propre message afin qu'il ne se bloque pas en attente d'une réponse de lui-même. L'émetteur indique dans le message le nombre de réponses souhaitées. L'arrivée du premier message de réponse débloque l'expéditeur, les réponses suivantes sont mises dans une file, de laquelle l'émetteur du message les extrait en invoquant la primitive GetReply (cf. figure 5).

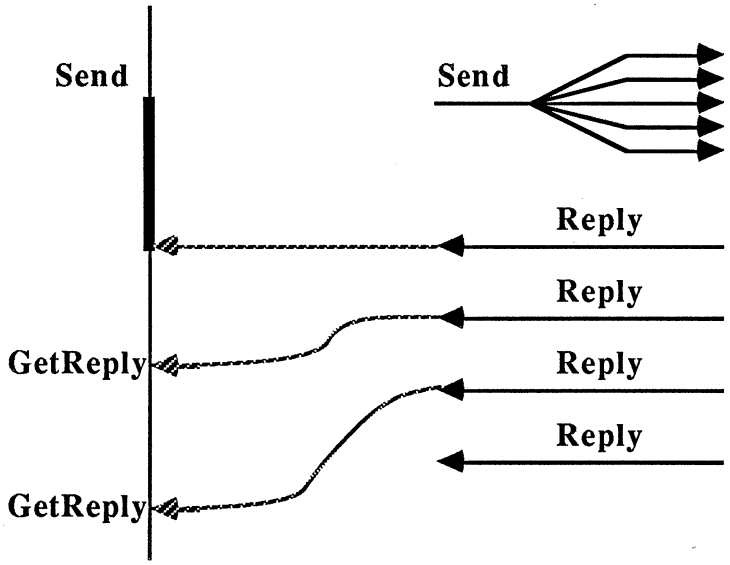

Figure 5 : Echange de messages entre un processus et un groupe de processus

Nous donnons dans le chapitre 4 une description plus précise des primitives de communication ainsi que quelques aspects de leur mise en œuvre.

3.4. Discussion

La diffusion 1-N avec M réponses est le mode de communication le plus général entre un processus et un groupe de processus. Le noyau de Chorus offre le mode d'adressage 1-N sans réponse et le mode d'adressage fonctionnel qui le cas particulier de la diffusion 1-N avec M réponses quand M vaut 1. Le V Kernel offre la diffusion 1-N avec M réponses non fiables. Ce mode d'adressage permet de mettre en œuvre des protocoles demande/réponse.

Les noyaux de systèmes répartis offrent en général des mécanisme de communication non fiales de type datagrammes. C'est le cas par exemple du système Apollo/Domain, du V Kernel, de Chorus. Les applications qui nécessitent un communication fiable mettent en œuvre des protocoles de communication fiable. Cela permet de limiter la taille et la complexité du noyau. Le rôle d'un noyau est seulement de fournir les outils de base nécessaires à la construction d'applications distribuées.

Le système Raddle propose comme mécanisme de communication le rendez-vous généralisé à N participants avec échange multidirectionnel de données. Pour mettre en œuvre ce mécanisme de manière performante, il semble que des primitives de base aussi rudimentaires que celles du V Kernel ne soient pas suffisantes. Un noyau plus complexe, fournissant en particulier des mécanismes permettant d'assurer la propriété d'atomicité du rendz-vous, serait certainement plus adapté. Il en ressort que les noyaux de systèmes répartis doivent être conçus en même temps que le langage permettant d'écrire les applications distribuées des niveaux supérieurs pour assurer une facilité d'écriture des sous-systèmes et de bonnes performances.

4. LA DESIGNATION DANS LES NOYAUX DE SYSTEMES REPARTIS

Un noyau de sytème réparti doit permettre l'attribution de noms aux ressources disponibles dans le système pour permettre leur désignation. L'ensemble des noms doit être aussi structuré, de telle sorte que l'on puisse établir des relations entre les entités qu'il désigne. La désignation des entités d'un système utilise en général un schéma à deux niveaux : les utilisateurs connaissent un nom symbolique, le système utilise un nom interne qui désigne soit l'objet lui-même, soit un objet relais (descripteur, capacité) [TAN 85].

Dans un premier paragraphe, nous montrons l'intérêt de distinguer plusieurs couches de désignation puis nous voyons, dans le paragraphe 4.2, les différentes manières de constituer des noms internes. La désignation symbolique et la traduction des noms symboliques en noms internes sont étudiées dans le paragraphe 4.3. Enfin, comme illustration, nous verrons la désignation dans le V Kernel.

4.1. Les couches de désignation

Les usagers désignent les objets qu'ils utilisent par des chaînes de caractères plutôt que par des identificateurs numériques plus difficiles à mémoriser pour eux. Il n'est pas souhaitable, pour des raisons d'efficacité et de performance que ces noms symboliques soient utilisés par le noyau. C'est pourquoi, on distingue une couche de désignation interne servant au noyau à désigner les entités qu'il manipule. Les adresses de site sur le réseau ne peuvent pas servir de couche de désignation interne dans un système réparti. En effet, un système de désignation réparti doit permettre de désigner les ressources indépendamment de leur localisation ; les ressources doivent pouvoir

migrer sans changer de nom et enfin les pannes ne doivent pas modifier la relation entre les noms et les entités désignées. Les adresses sur le réseau ne respectent pas ces exigences car elles ne sont pas transparentes vis à vis de la localisation. C'est ce qui explique l'existence de la couche de désignation interne entre la couche de désignation symbolique et les adresses sur le réseau (cf. figure 6). Nous allons maintenant étudier chacune des deux couches de désignation.

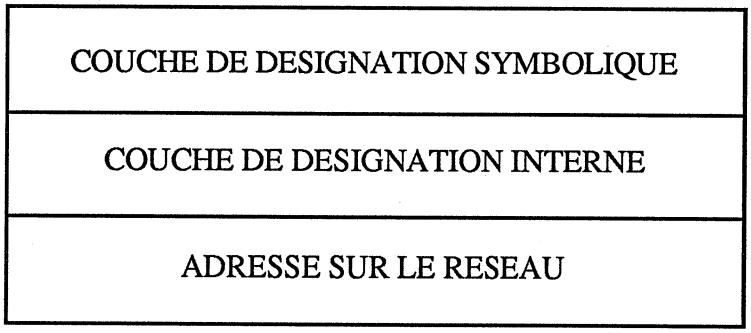

Figure 6 : Les couches de désignation.

4.2. Désignation interne

La désignation interne des entités d'un noyau de système réparti (processus, portes, groupes) peut se faire de deux manières : désignation soit par des noms internes uniques et globaux, soit par des noms internes locaux ou contextuels [LEG 86]. Nous nous intéressons, dans la suite, à chacun de ces deux modes de désignation interne.

Les identificateurs uniques (uid) sont des noms uniques dans l'espace et dans le temps. Ces identificateurs sont produits par le noyau de système à la création de l'entité. En général, un nom interne est constitué de l'identificateur interne du créateur de l'entité et d'un identificateur unique par rapport au créateur. Il existe deux méthodes de construction d'un nom interne unique et global : la méthode de désignation physique et la méthode de désignation logique.

1. La désignation physique

Un nom interne est de la forme suivante :

<nom interne> ::= <adresse de site> <nom local au site>.

La correspondance entre le nom interne de l'entité et son adresse est immédiate à moins que l'entité ait migré. Dans ce cas, une indirection est nécessaire. L'adresse de site ne donne alors qu'une indication sur la localisation de l'objet. L'indirection peut se faire grâce à des informations conservées par le site de création.

Par exemple, dans le système Apollo/Domain [LEA 83], les objets sont identifiés par des noms internes uniques constitués de la façon suivante :

<site de création> <heure de création> <type>.

Le système Chorus utilise également des uid pour désigner des portes et des groupes de portes [LEG 86]. Leur forme est la suivante :

<nom du site de création> <type> <estampille unique au site>.

Les noms de site sont des numéros uniques alloués statiquement. L'estampille est calculée à partir de l'heure logique de création de l'entité.

2. Désignation logique

Un nom interne est construit de la façon suivante :

<nom interne> ::= <nom de serveur> <nom local au serveur>.

Dans les systèmes utilisant cette méthode de désignation, les noms sont gérés par des serveurs. L'avantage de cette méthode est de permettre implicitement la migration des objets puisque le nom interne n'est pas directement lié à la localisation de l'objet. En outre, elle autorise les reconfigurations dynamiques. Par exemple, dans un système où un processus est associé à une porte, en cas de défaillance d'un processus, celui-ci peut être remplacé par un autre processus sans incidence pour les utilisateurs du service.

Les principaux avantages de l'utilisation des uid comme nom interne sont les suivants : ils sont indépendants de la localisation physique et permettent donc la migration des entités ; ils sont non réutilisables ce qui permet aux processus de mémoriser le nom des entités puisque celles-ci sont désignées sans ambiguité ; enfin, les noms peuvent être échangés par les processus sans traduction.

Cependant, les uid présentent un inconvénient : dans les systèmes où la connaissance de l'uid est une condition suffisante pour envoyer un message, l'espace d'adressage des processus est incontrôlé ; tout processus peut envoyer un message à tout autre processus. C'est pourquoi, certains systèmes utilisent la désignation interne contextuelle.

Les noms contextuels ou locaux cachent les uid aux processus. Chaque processus possède en effet un contexte de désignation interne. Cela introduit une nouvelle couche de désignation, les noms contextuels étant traduits en uid. Le noyau de système distribué Accent, dans lequel les processus communiquent par messages à travers des portes, utilise la méthode de désignation contextuelle pour la désignation interne des portes. Pour pouvoir émettre un message ou recevoir un message sur une porte, un processus doit possèder une capacité avec les droits correspondants : droit d'émission, droit de réception. La capacité est le nom contextuel de la porte. Cette méthode de désignation facilite les traitements d'exceptions liées aux portes mais impose une double traduction nom local - nom global à chaque accès [RAS 81].

4.3. La désignation symbolique

Les usagers d'un système ont la possibilité de désigner les ressources et les services par des chaînes de caractères ou noms symboliques. Généralement, l'espace des noms symboliques est structuré hiérarchiquement. Les noms sont constitués de plusieurs composants de la même façon que dans les systèmes centralisés. Dans le système centralisé Multics, on a un arbre de désignation. En partant de la racine vers les nœuds terminaux et en nommant tous les nœuds intermédiaires, on obtient le nom absolu de l'entité. Tous les nœuds non terminaux représentent des catalogues. Les entrées de ces catalogues font la correspondance entre un nom symbolique et soit le nom interne d'un autre catalogue, soit le nom interne d'une entité. En environnement réparti, le problème de la traduction des noms symboliques en noms internes est plus complexe. Cette traduction dépend en effet de la structure de l'espace des noms. Nous considérons trois possibilités. Une première façon d'interconnecter les arbres de désignation de chaque site est d'avoir une super-racine ou catalogue-réseau regroupant la racine de chaque site. Les entités ont un nom global par rapport à la super-racine. Généralement une syntaxe particulière indique qu'un nom est global sinon il s'agit d'un nom contextuel par rapport à la racine de l'arbre de désignation du site. Dans d'autres systèmes, les espaces des noms de chaque site sont interconnectés arbitrairement et à la demande en établissant des références. C'est l'analogue du "montage" dans le système Unix. Les noms sont des noms contextuels. Enfin, il existe des systèmes où l'espace des noms est logiquement centralisé. Il y a dans ces systèmes un serveur des noms central, éventuellement implanté de manière répartie, qui gère les noms des entités du système. Ces noms sont des noms globaux. Dans le système Locus par exemple, les usagers ne voient qu'un seul espace des noms de fichiers [WAL 83]. Locus comporte un système de gestion de fichiers logiquement centralisé mais mis en œuvre de façon répartie.

Il y a deux approches pour établir la correspondance entre un nom symbolique et un nom interne : d'une part l'approche serveur des noms qui peut être mise en œuvre de façon centralisée ou répartie et d'autre part l'approche serveur des noms et des ressources. Le système Grapevine [BIR 82] illustre la première approche, il fournit un service de désignation réparti. Les noms symboliques dans Grapevine sont des noms globaux. L'espace des noms est structuré à deux niveaux. Un catalogue racine contient les identificateurs des sous-catalogues qui eux contiennent les identificateurs des entitées désignées. On trouvera des détails supplémentaires dans [LEG 86]. Le V Kernel comme le système Chorus suit l'approche des serveurs des ressources et des noms. Dans le V Kernel [CHE 84], la désignation et l'interprétation des noms symboliques sont explicitement réparties ; chaque serveur qui met en œuvre des entités permanentes doit aussi mettre en œuvre leur désignation symbolique. Cela est abordé plus en détail dans le paragraphe suivant consacré à la désignation dans le V Kernel.

4.4. La désignation dans le V Kernel

1. Désignation interne dans le V Kernel

Un V Domain est un ensemble de machine reliées par un réseau local supportant le V Kernel. Les processus et groupes de processus sont désignés par des identificateurs uniques et globaux au sein d'un V Domain. Ces identificateurs sont structurés en deux champs de seize bits chacun (cf. figure 7). Le premier champ est un identificateur de site logique (Logical host identifier), unique dans un V Domain, qui correspond au site de création du processus ou groupe de processus. Le deuxième champ est un identificateur de processus local (Local process identifier) dans le cas d'un identificateur de processus ou un identificateur spécifique de groupe (Specific group identifier) dans le cas d'un identificateur de groupe [CHE 85a].

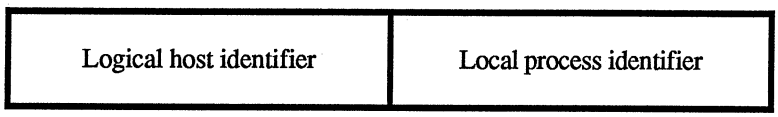

Figure 7 : Identificateur de processus.

Un bit, le "GroupId bit", permet de distinguer les deux types d'identificateurs : zéro pour un identificateur de processus, un pour un identificateur de groupe (cf. figure 8). Cette convention assure que les deux espaces des noms sont disjoints et permet au noyau de distinguer aisément les deux types d'entités.

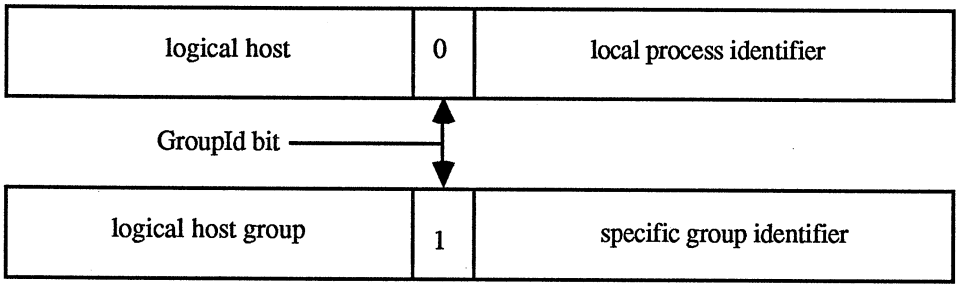

Figure 8 : Le group-id bit.

2. La désignation symbolique dans le V Kernel

D'abord, nous décrivons la désignation symbolique des objets puis brièvement la désignation symbolique des services.

On distingue deux types d'objets : les objets permanents et les objets temporaires. Un objet temporaire n'existe que pendant la durée du programme qui a demandé sa création. Il est nommé

par un identificateur généré par le serveur à la création de l'objet. Cet identificateur est utilisé par le programme et le serveur qui met en œuvre l'objet. C'est un identificateur numérique de seize bits (Object Instance Identifier) purement interne qui n'est donc pas choisi par l'usager. Un objet permanent survit à la fin de l'exécution du processus qui l' a créé. Il est désigné par un identificateur de type chaîne de caractères appelé "Character String Name". Ces identificateurs et leur interprétation sont étudiés dans la suite.

Un nom de type chaîne de caractères - ou CSname - est composé d'une chaîne de un ou plusieurs octets de longueur spécifiée. L'espace des noms est hiérarchique, c'est pourquoi la chaîne de caractères peut être constituée de plusieurs composants. Plusieurs serveurs de désignation peuvent participer à l'interprétation d'un nom symbolique. On appelle serveur de désignation (Character String Name Handling Server ou CSNHserver) tout serveur qui fait la liaison entre un nom et un objet. Les serveurs de désignation peuvent faire autre chose en plus de leur fonction de désignation. L'interprétation d'un CSname dépend du contexte dans lequel il est utilisé. D'une façon générale, un contexte est un ensemble de couples (nom, objet). Dans le V Kernel, un contexte est un ensemble de couples (identificateur de serveur, identificateur de contexte). L'identificateur de contexte désigne un contexte particulier du serveur. En effet, un serveur peut avoir plusieurs contextes s'il gère plusieurs types d'objets (si un serveur ne gère qu'un seul contexte, l'identificateur de contexte vaut zéro par défaut). Les identificateurs de contextes sont des identificateurs numériques qui sont liés au serveur et existent tant que celui-ci existe. Pour les contextes usuels tels que le HomeDirectory, il existe des identificateurs de contexte fixés.

Un nom symbolique est constitué d'un identificateur de processus serveur, d'un identificateur de contexte et d'une chaîne d'octets. La chaîne d'octets représente le chemin d'accès à l'objet. La paire (identificateur de serveur, identificateur de contexte) indique le contexte dans lequel doit être interprété le premier composant du nom. Les messages de requête contenant un CSname comportent les champs suivants :

- la chaîne de caractères,
- la longueur de la chaîne de caractères,
- l'index dans la chaîne de caractères indiquant où en est l'interprétation,
- l'identificateur du contexte dans lequel le nom doit être interprété.

Le nom du serveur est indiqué implicitement puisque c'est le destinataire du message. Tout message de requête doit être conforme à ce format standard. Les champs standards sont toujours au même endroit dans un message ; cela permet aux serveurs de désignation d'interpréter un nom sans comprendre le code opération de la requête. L'algorithme exécuté par les serveurs de désignation est décrit ci-dessous [CHE 84]. On se place, pour cet algorithme, dans le cas général d'une désignation hiérarchique. Le nom représente le chemin d'accès à l'objet, il comporte éventuellement plusieurs composants.

Lors de la réception d'une requête

(codeopération,nom,longueur,index,contexte)

par un serveur de désignation S :

```
contexte_courant := contexte ;
si (nom[index],S') ∈ contexte_courant et type (nom[index]) = identificateur de
                                                              contexte
alors
   contexte_courant := nom[index] ;
   si S' ≠ S alors
        index := index +1 ;
        contexte := contexte_courant ;
        envoyer la requête (codeopération, nom, longueur, index,contexte);
   sinon
        interprétation de nom[index +1] par S
   fsi
sinon
   si nom[index]∈ contexte_courant et type (nom[index]) = identificateur d'objet
   alors
        exécuter l'opération de code codeopération
   sinon
        erreur
   fsi
fsi
```

L'espace des noms du V Kernel peut être vu comme une forêt dont les arbres sont les espaces des noms des serveurs de désignation (cf. figure 9). Chaque station de travail exécute pour le compte de son usager, en plus des autres serveurs, un pur serveur de désignation appelé serveur de préfixe qui permet d'une part d'associer des identificateurs à des noms internes de catalogues et d'autre part d'associer automatiquement des identificateurs prédéfinis aux catalogues racines des serveurs de désignation du réseau. Les serveurs de préfixe permettent à tous les usagers d'avoir la même vision du monde.

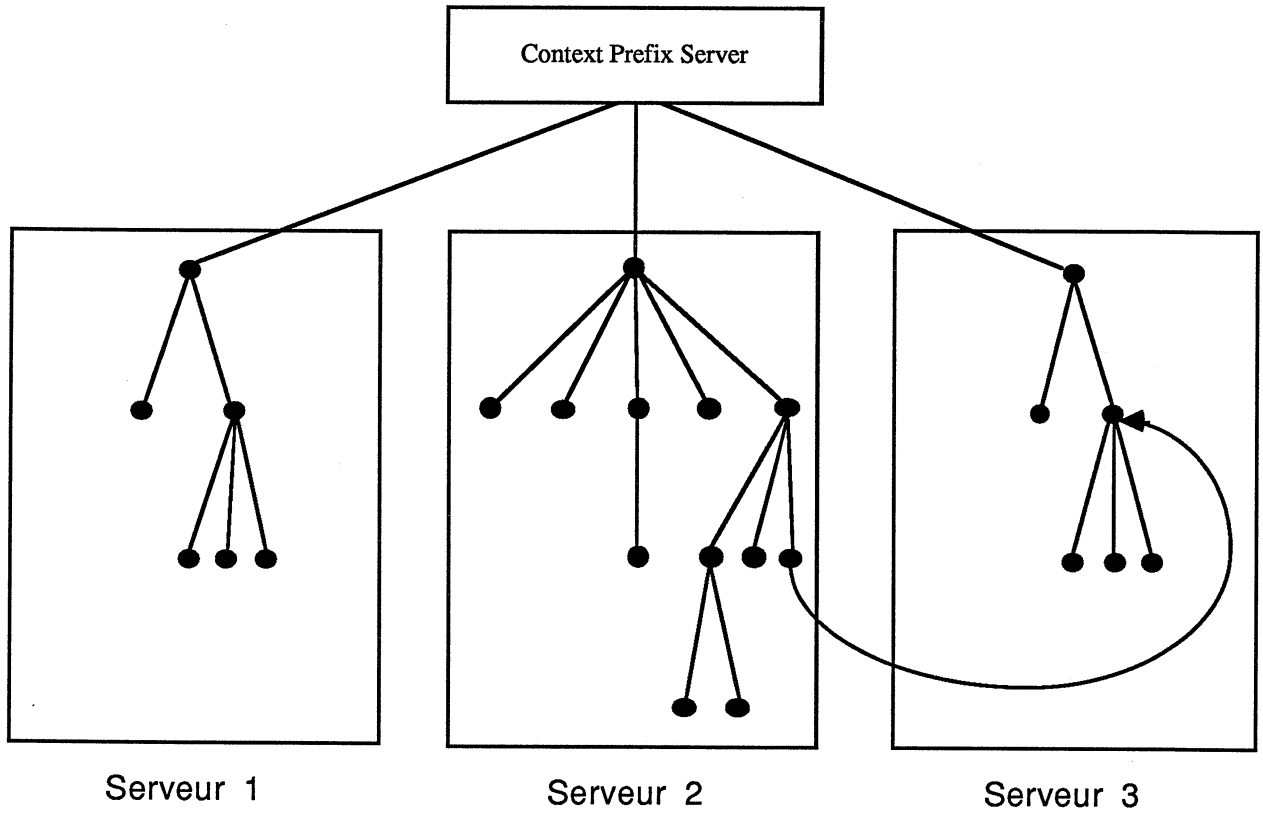

Figure 9 : L'espace des noms du V-KERNEL.

La syntaxe d'une requête servie par le serveur de contexte est la suivante :

[préfixe] identificateur .

Une telle requête est envoyée au serveur de préfixe qui analyse le préfixe et redirige la requête (avec le nom privé du préfixe) vers le serveur associé au préfixe.

exemple :

open [préfixe] fich

Comme le nom comporte un préfixe, la requête est envoyée au serveur de contextes de la station qui analyse le préfixe et envoie la requête *open fich* au serveur de fichier mettant en œuvre le contexte spécifié.

open fich

Comme le nom ne comporte pas de préfixe, le contexte est par défaut le contexte courant. La requête est donc envoyée directement au serveur qui le met en œuvre sans passer par le serveur de contextes.

Chaque serveur de contexte définit les contextes utilisés par l'utilisateur de la station :

- préfixes de contextes standards
- préfixes correspondant à des serveurs de fichiers
- préfixes de contextes spéciaux tel que le home directory.

Les préfixes doivent être de l'une des deux formes suivantes :

- (identificateur de processus, identificateur de contexte)

- (identificateur logique, identificateur de contexte prédéfini).

Pour la deuxième forme, le serveur exécute un GetPid pour transformer l'identificateur logique en identificateur de processus.

Dans le V Kernel, les services peuvent également être désignés par des noms symboliques ; "impression" peut, par exemple, désigner le service d'impression. Les principaux services sont le stockage des données, l'impression, le temps la gestion des contextes, la gestion des terminaux graphiques, le traitement des exceptions survenant à l'exécution (par exemple, débordement d'un pointeur de pile). La primitive *SetPid* permet aux processus d'être enregistrés comme fournissant un service particulier. Par exemple, le service "impression" est fourni par le processus P1. Les clients disposent de la primitive *GetPid* leur permettant d'obtenir l'identificateur d'un processus qui fournit le service demandé.

SetPid (LogicalId,ProcessId,Scope)

- LogicalId : nom logique du service
- ProcessId : identificateur d'un processus qui va rendre le service désigné par LogicalId
- Scope: portée

ProcessId = GetPid (LogicalId, Scope)

- LogicalId : nom logique du service demandé
- ProcessId : identificateur d'un processus rendant le service demandé
- Scope : portée

Scope peut prendre l'une des deux valeurs suivantes :

- "local"
- "local et distant".

Le noyau comporte une table locale qu'il consulte lors du *GetPid* . S'il ne trouve pas le service demandé et que la portée n'est pas égale à "local" alors il diffuse la requête aux noyaux des autres sites.

Les programmes sont écrits en terme de services. La liaison entre service et serveur a lieu à l'exécution. Ainsi, l'ensemble des serveurs rendant un service donné peut varier au cours du temps. Selon le type de serveur, la liaison est établie lors de chaque requête ou reste établie pour plusieurs requêtes.

5. QUELQUES ASPECTS DE LA MISE EN ŒUVRE DU V KERNEL

Le V Kernel est mis en œuvre sur un ensemble de stations de travail SUN interconnectées par un réseau local Ethernet à débit élevé.

La machine SUN est conçue autour du processeur MC 68010 (cf. figure 10). Son système d'exploitation est Unix 4.2 BSD, qui en plus des services du système Unix standard, comporte des extensions permettant la communication entre processus.

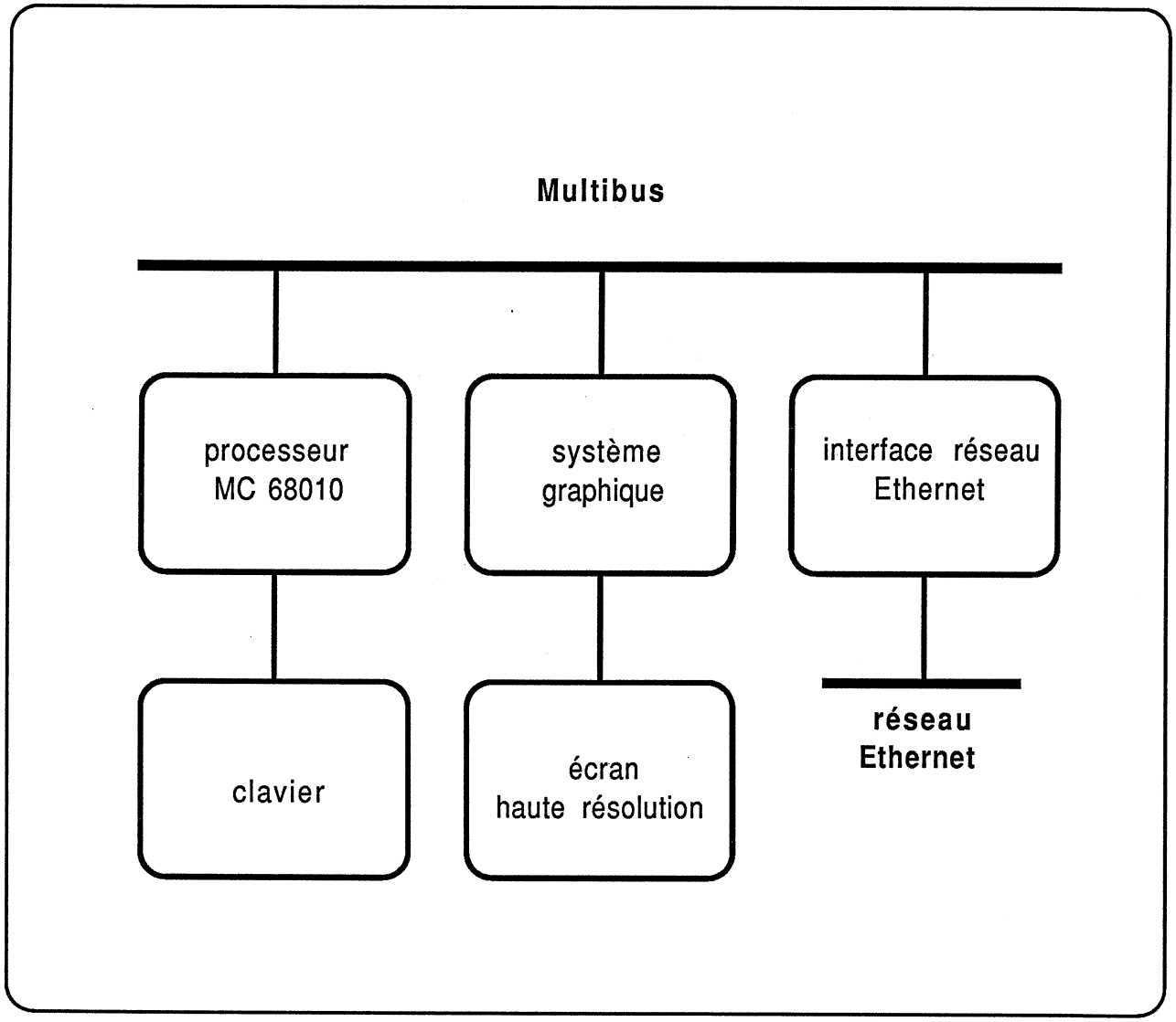

Figure 10 : Station de travail SUN.

Dans ce chapitre, nous nous intéressons à quelques aspects de la mise en œuvre. Le paragraphe 5.1. est consacré au problème de la localisation des entités dans le système. La gestion des échanges de messages et des transferts de données est étudiée dans le paragraphe 5.2. Les primitives de manipulation des groupes sont décrites dans le paragraphe 5.3. Enfin, le paragraphe

5.4. donne une brève analyse des performances du noyau.

5.1. Localisation des processus et groupes de processus

1. Traduction d'un nom interne de processus en une adresse sur le réseau

Conceptuellement, la correspondance entre les identificateurs de processus et leur adresse peut être faite à l'aide d'une table dont chaque entrée associe une adresse à un identificateur interne de processus. Cette solution n'est pas mise en œuvre. En effet, il n'est pas nécessaire d'avoir une copie de la table complète sur tous les sites. Chaque noyau conserve une table partielle. L'entrée de la table associée à un processus ne donne qu'une indication de sa localisation de façon à permettre la migration des processus de manière transparente. Si le processus n'est pas trouvé à l'adresse indiquée alors le noyau diffuse le message à tous les sites. Pour établir qu'un identificateur est invalide, le noyau donc faire une diffusion. Par contre, pour établir la validité d'un identificateur de processus, il suffit de déduire de la table l'adresse du processus et de vérifier que l'identificateur est bien valide sur le site indiqué. Lors de l'envoi d'un message à un processus non présent dans la table, le message est diffusé à tous les sites du réseau (l'envoi à une adresse physique particulière du réseau signifie une diffusion à tous les sites du réseau). En fait, il s'agit d'une table de hachage. Les processus qui sont sur un même site possèdent la même entrée dans la table. Cette entrée est obtenue d'après le champ adresse logique de site de l'identificateur de processus. Les entrées de la table sont créées d'après les paquets reçus par le site (ceux-ci contiennent l'adresse d'origine et l'identité du processus origine). Les entrées sont éliminées selon un algorithme LRU [ZWA 85].

Le système Apollo/Domain permet également la migration des objets hors de leur site de création. Dans ce système, l'algorithme de localisation d'un objet à partir de son nom interne (constitué, rappelons le, du numéro du site de création concaténé à l'heure de création) utilise le numéro de site création. Si l'objet ne se trouve pas sur son site de création, il utilise alors les indications conservées par un gérant d'indications (Hint Manager). Chaque fois qu'un objet migre, le gérant d'indication en est informé. Pour améliorer les performances de l'algorithme de localisation, sur chaque site, les associations noms internes - localisations les plus récentes sont conservées dans un cache [LEA 83].

2. Localisation des membres d'un groupe à partir du nom interne de groupe

Un groupe est localisé grâce à son identificateur interne. Il est caractérisé par les seize bits de poids faible de son nom interne :

- le "Static Group Bit" indique si l'identificateur de groupe a été alloué statiquement.
- le "Local Group Bit" indique si le groupe est local. Un groupe est local si tous ses membres sont situés sur le site de création. Cette indication permet d'optimiser les échanges de

messages entre les processus et les groupes.

- le "Unrestricted Group Bit" indique si le groupe est ouvert à tout processus.

Les seize bits de poids fort servent à coder l'ensemble des sites sur lesquels les membres du groupe résident. Dans le cas d'un groupe local, l'interprétation est la même que pour un identificateur de processus (cf. figure 11).

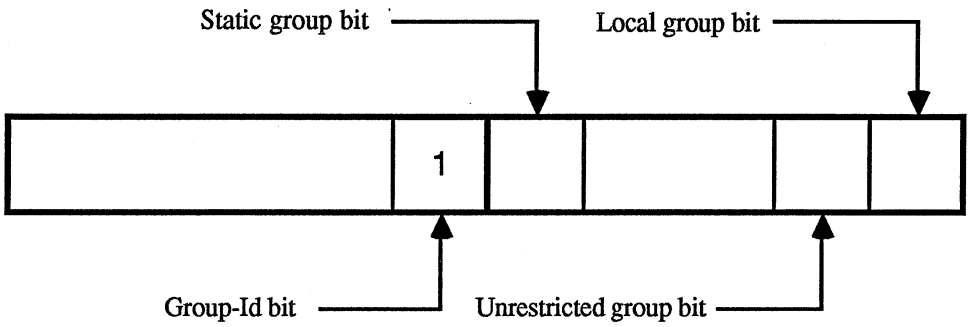

Figure 11 : Identificateur de groupe.

Chaque exemplaire du noyau conserve une table de hachage accèdée par l'identificateur de groupe. Une entrée de la table donne les membres du groupe qui sont situés sur le site considéré. Pour chaque processus appartenant au groupe, il existe un couple (identificateur de groupe, identificateur de processus). L'information sur les membres d'un groupe est donc répartie sur tous les sites où il y a au moins un des membres. Chaque noyau ne conserve des informations que sur les processus de son site. Cette mise en œuvre nécessite moins d'espace mémoire et engendre moins de traffic sur le réseau qu'une solution qui consisterait à dupliquer les informations concernant les membres du groupe sur tous les sites. En outre, comme il n'y a pas de duplication d'information, il n'y a pas de problème de consistance. Tout cela suppose de traduire efficacement l'identificateur de groupe en "Logical Host Group" et le "Logical Host Group" en l'ensemble des sites où se trouvent les membres du groupe. On passe très simplement de l'identificateur de groupe au "Logical Host Group" puisque le "Logical Host Group" est un des champs de l'identificateur de groupe (cf. figure 11).

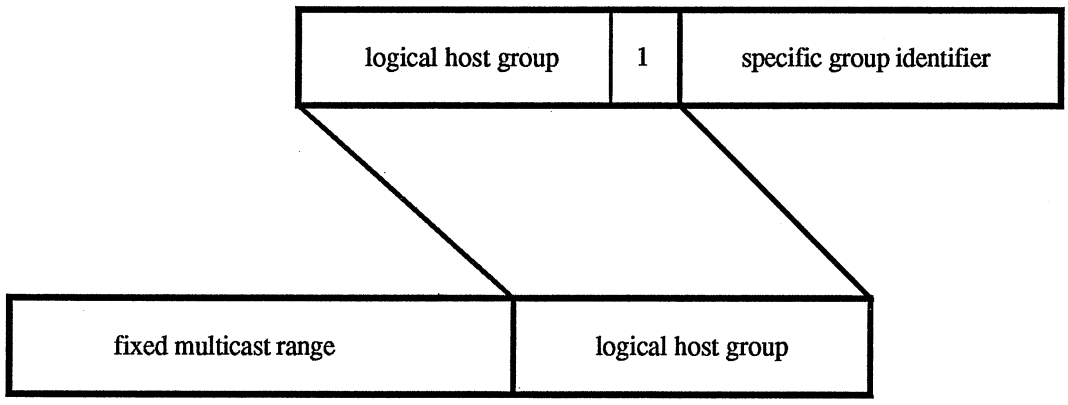

Figure 12 : Passage d'un identificateur de groupe à une adresse réseau.

Le passage du "Logical Host Group" aux sites dépend du réseau. Il se fait de la manière suivante : le V Kernel utilise au niveau transport des paquets dont le format est défini par le protocole internoyaux indépendant du réseau. Lors de la transmission par le réseau, ces paquets sont mis dans un datagramme. Le noyau transmet son paquet internoyaux au gérant du réseau, n'appartenant pas au noyau, qui est chargé de convertir les adresses utilisées par les processus en adresse de sites et de préparer l'entête du paquet réseau (cf. figure 13).

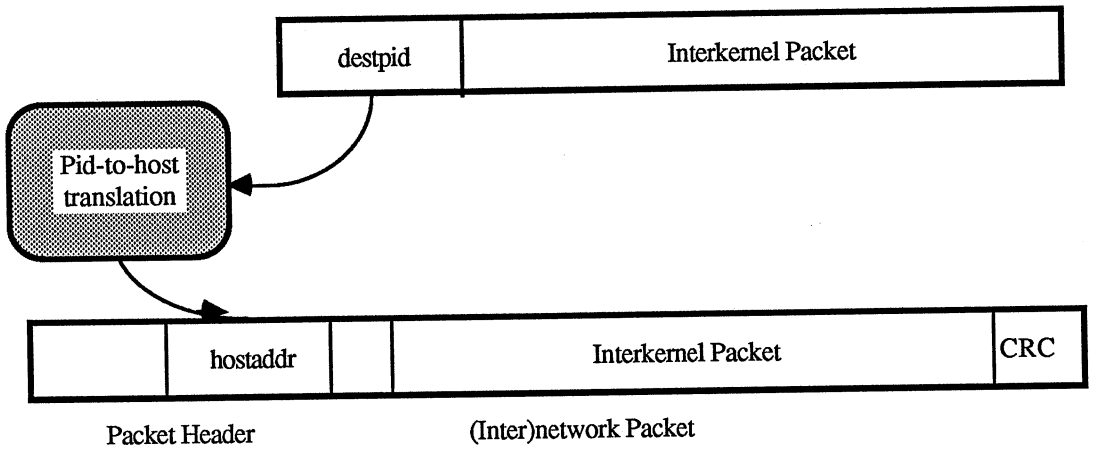

Figure 13 : Un paquet internoyaux dans un paquet du réseau.

Le noyau conserve un cache contenant les correspondances entre adresse logique et adresse sur le réseau les plus récemment utilisées. Si une correspondance est absente du cache, le noyau utilise une adresse qui représente l'ensemble des sites du V Kernel. Cela revient à diffuser le paquet à tous les sites du V Domain sinon le noyau extrait le "Logical Host Group" de l'identificateur de groupe et envoie le paquet à l'adresse correspondant au groupe de sites.

5.2. Les échanges de messages et les transferts de données

Après avoir décrit le format standard des paquets véhiculant les messages et les données défini par le protocole internoyaux, nous décrivons les primitives d'échange de messages pour un échange entre deux processus dans un premier temps puis pour un échange entre un processus et un groupe de processus dans une seconde partie. Enfin, nous décrivons brièvement les mécanismes de transfert de données.

1. Le format des paquets

Un paquet comporte deux parties : l'entête et la zone de données. L'entête est de taille fixe et contient de l'information de service [ZWA 85]. La zone de données est de taille variable (de zéro au maximum permis par le format des paquets du réseau local) ; elle permet de joindre des données à un message de type *Send*, *Reply*, *MoveTo* ou *MoveFrom*. La figure 14 donne les différents champs d'un paquet.

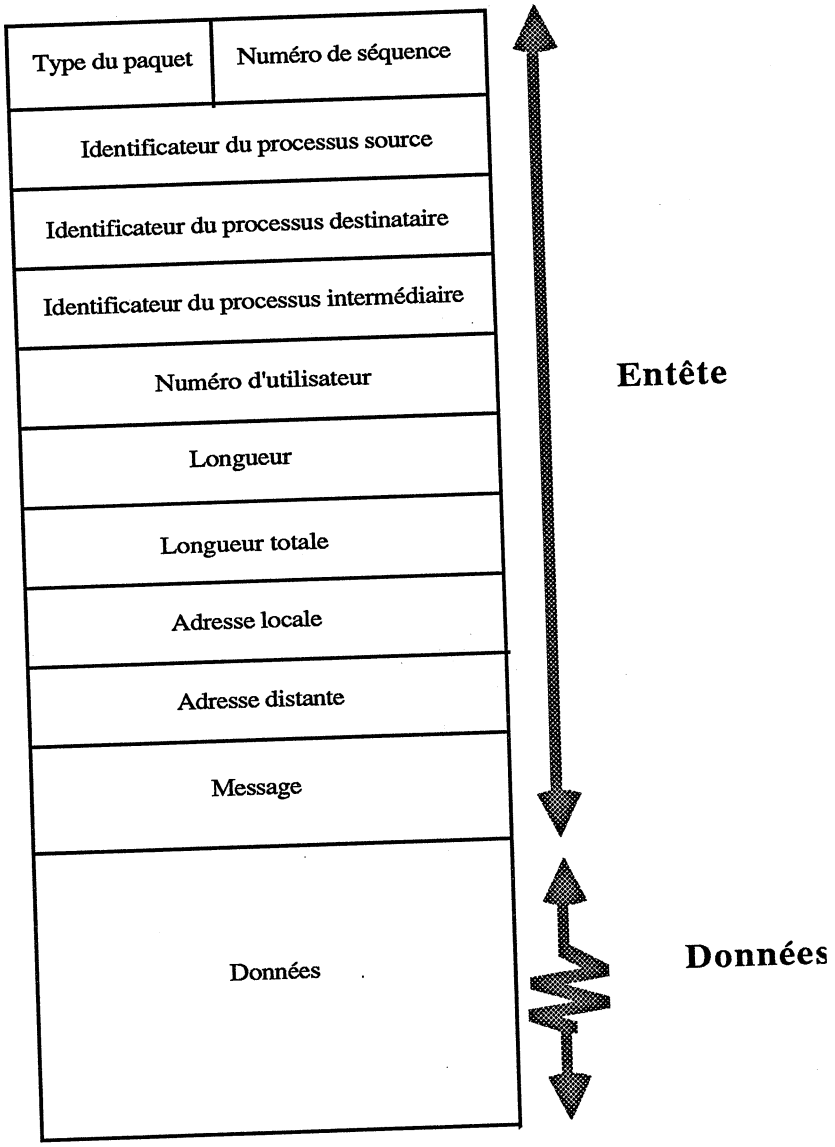

Figure 14 : Format des paquets.

Décrivons les différents champs :

-*Type du paquet* : indique le type de l'opération (RemoteSend, RemoteReply,RemoteForward, ReplyPending ...)

- *Numéro de séquence* : numéro de séquence de l'échange de messages (unique pour le

processus ayant fait le Send)

- *Identificateur du processus source* : identité du processus qui exécute l'opération.
- *Identificateur du processus destinataire* : identité du processus destinataire du paquet.
- *Identificateur du processus intermédiaire* : identité du processus ayant transmis le paquet.
- *Numéro d'utilisateur* : numéro d'utilisateur du processus source. Ce champ permet de faire des contrôles de sécurité mais il n'est pas utilisé dans la version actuelle du V-KERNEL.
- *longueur* : longueur en nombre d'octets du champ données du paquet.
- *Longueur totale* : longueur totale du segment dont les données du paquet font partie.
- *Adresse locale* : adresse d'origine du segment de données.
- *Adresse distante* : adresse de destination du segment de données.

Tous les champs ne sont pas toujours utilisés, cela dépend du type du paquet. Les paquets du V-KERNEL sont mis dans des paquets du réseau. Un champ de ces derniers permet de distinguer les paquets du V-KERNEL des autres.

2. Les échanges de messages entre processus

Send (Message, ProcessId)

Le message contenu dans la variable Message est envoyé au processus désigné par l'identificateur de processus ProcessId. Le processus qui a invoqué la primitive se bloque en attente d'une réponse de la part du récepteur.

Le format des messages est défini par le V Kernel. Un processus doit indiquer dans le message, dans le cas d'une lecture ou d'une écriture, le segment de son espace d'adresses qui pourra être accédé par le récepteur. Un segment est spécifié par les deux derniers mots du message qui sont respectivement son adresse de début et sa longueur. Des bits de contrôle en début de message indiquent si un segment est spécifié en fin de message et le cas échéant les droits d'accès.

(ProcessId, Bytecount) = Receive(Message, SegmentPointer, SegmentSize)

Le processus ayant invoqué la primitive Receive se bloque en attente de message. A la réception d'un message, ProcessId est l'identificateur de l'expéditeur, Message contient le message reçu. Le segment joint au message est rangé dans l'espace mémoire du récepteur en commençant à l'adresse SegmentPointer, les Segmentsize premiers octets du segment sont copiés. ByteCount repésente le nombre d'octets du segment qui ont été effectivement reçus.

Les messages sont mis dans une file d'attente fonctionnant suivant le principe premier arrivé premier servi.

Reply (Message, ProcessId, DestinationPointer, SegmentPointer, SegmentSize)

Le message de réponse contenu dans la variable Message , ainsi que le segment commençant à SegmentPointer et de longueur SegmentSize sont envoyés au processus ProcessId. Le message de réponse est copié à la place du message original chez l'initiateur de l'échange de messages. Si un segment est joint au message, il est copié à partir de l'adresse DestinationPointer.

L'exemplaire du noyau de la machine sur laquelle est situé le processus qui a invoqué la primitive Send envoie un paquet de type RemoteSend sur le réseau, soit directement à la machine où se trouve le processus destinataire, soit au moyen d'un paquet de diffusion. Les champs du paquet qui sont utilisés sont :

- numéro de séquence
- identificateur du processus source
- identificateur du processus destinataire
- identificateur du processus intermédiaire (égal à l'identificateur du processus de destination)
- message.

Si le noyau du site de destination a un tampon disponible, il accepte le message et le met dans une file pour le processus destinataire. Quand le processus destinataire répond, un paquet du type RemoteReply est envoyé au processus source avec le champ numéro de séquence identique à celui du paquet RemoteSend. Les valeurs des champs processus source, processus destinataire et message sont significatives.

3. Traitement des anomalies dans les échanges de messages entre processus

Les anomalies possibles sont la perte de packets sur le réseau et la mort de l'un ou l'autre des processus intervenant dans l'échange de messages.

Nous examinons les moyens mis en œuvre pour traiter ces erreurs du côté de l'expéditeur et du côté du destinataire.

Lors de l'envoi d'un paquet du type RemoteSend, deux réveils sont armés, l'un pour la retransmission avec un intervalle de temps T_r, l'autre pour le timeout avec un intervalle de temps T_t. Lorsque le délai de retransmission expire, un paquet de type RemoteSend identique au paquet original est envoyé et le réveil de retransmission est réarmé.

A l'expiration du timeout, le processus de destination est considéré mort ou inatteignable et l'échange de message est terminé par l'envoi d'un code d'erreur au processus qui a invoqué la primitive Send.

A la réception d'un paquet de type RemoteSend, le noyau compare le couple (identificateur de processus source, numéro de séquence) du paquet à une liste de couple de même format correspondant aux échanges de messages en cours. S'il s'agit d'une retransmission le paquet est écarté et le noyau envoie un paquet de type ReplyPending. Un message de ce type est également envoyé lorsque le noyau est obligé d'écarter le paquet par manque de tampons.

A la réception d'un paquet de type ReplyPending, le noyau réarme les deux réveils (retransmission et timeout).

On peut classer les requêtes en deux groupes selon qu'elles possèdent la propriété d'idempotence ou pas. Une requête est idempotente si l'effet produit dans le cas où elle est exécutée plusieurs fois est le même que dans le cas où elle n'est exécutée qu'une seule fois. La lecture d'une donnée dans un fichier peut être considérée comme une requête idempotente. Par contre, l'incrémentation d'un compteur n'est évidemment pas une opération idempotente. En fait, la propriété d'idempotence dépend du niveau auquel on se place : niveau usager, système, noyau. Une lecture est une requête idempotente pour une application mais pas forcément pour le système qui peut modifier ses informations de contrôle lors des accès aux fichiers. Dans le V Kernel, les messages peuvent être marqués pour indiquer si la requête est idempotente ou pas. Pour une requête idempotente, le noyau du récepteur perd la trace de l'échange de message dès l'envoi du paquet de type Reply. Si le paquet de type Reply se perd ou si un paquet de type RemoteSend est retransmis parallèlement à l'envoi du Reply, le message sera délivré deux fois au destinataire.

Si le message est marqué non-idempotent, le noyau garde une copie du message de réponse pendant une durée au moins égale à T_r . Si un paquet de retransmission d'un RemoteSend arrive, le message n'est pas délivré au processus et le noyau réenvoie un paquet Reply. Lorsque le message suivant du même processus arrive, le noyau détruit sa copie du Reply. L'arrivée du message suivant du même processus indique que l'échange de message précédent est terminé, le noyau détruit donc sa copie du Reply (l'échange se termine soit par expiration du timeout, soit par réception du Reply par l'émetteur du Send).

Quand un segment est joint à un Send ou à un Reply, le champ longueur du paquet correspondant indique la taille du segment de données joint au message. Dans le cas du Reply, le champ adresse distante indique l'origine de segment dans l'espace d'adresse de l'expéditeur où les données doivent être rangées.

Les paquets de type n-Ack servent à optimiser les performances. Lorsqu'un Send est envoyé à un processus qui n'existe pas, le noyau de la machine sur laquelle le processus aurait dû se trouver répond par un paquet de type n-Ack. Il faut rappeler que chaque noyau conserve une table locale qui donne une indication de la localisation des processus. Cela évite les retransmissions en attendant l'expiration du timeout.

4. les échanges de messages entre un processus et un groupe de processus

L'envoi d'un message à un groupe de processus est similaire à l'envoi d'un message à un processus. Le noyau reconnaît qu'un message est destiné à un groupe de processus en testant le "GroupId Bit". Si ce bit est à un, le noyau délivre le message à tous les membres locaux au site puis transmet un paquet contenant le message à l'adresse du groupe des sites où sont situés les autres membres du groupe. Le paquet est éventuellement retransmis plusieurs fois jusqu'à ce que la

première réponse soit reçue. Celle-ci débloque l'expéditeur du message. Les autres réponses, pourvu qu'elles arrivent avant le début du Send suivant, sont mises dans une file. Pour sortir les réponses de la file, le processus exécute la primitive GetReply. Nous donnons ci-dessous la description des primitives de communication lorsqu'elles sont utilisées avec un groupe de processus.

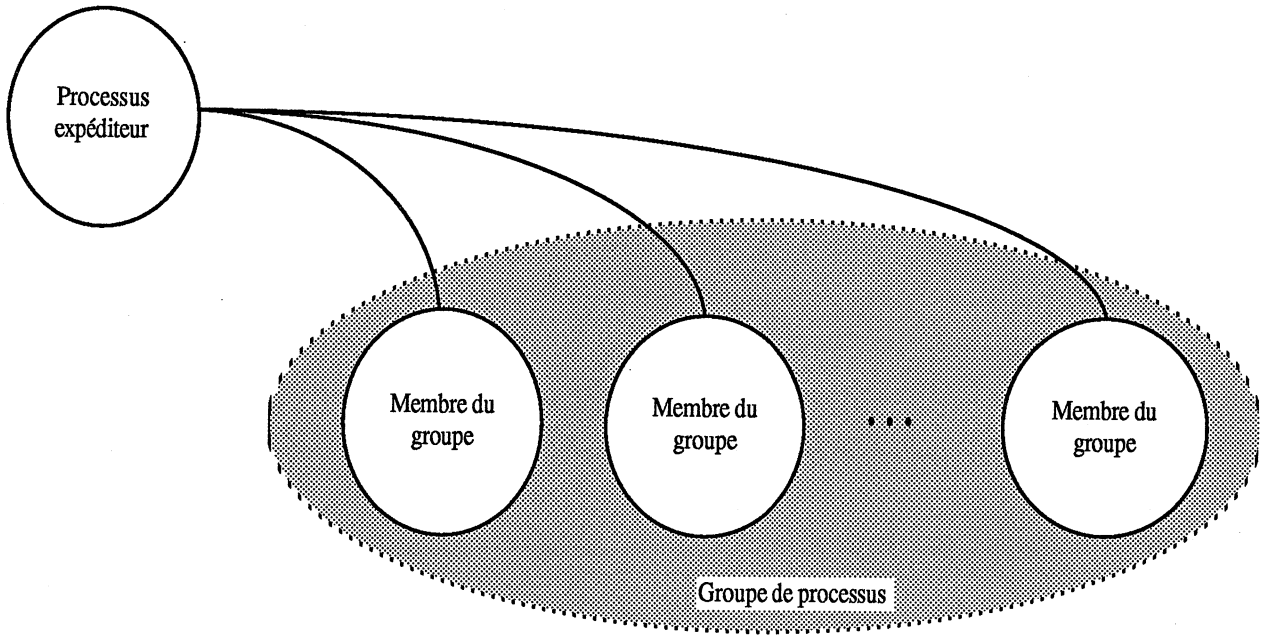

Figure 15 : Envoi d'un message à un groupe.

Pid = Send (Message,GroupId)

Le message contenu dans la variable Message est envoyé à tous les membres du groupe GroupId (cf figure 15). Deux bits du message indiquent si l'expéditeur souhaite zéro, une ou plusieurs réponses :

- aucune réponse : la primitive est non bloquante et il s'agit d'une diffusion non fiable.
- une réponse unique : l'émetteur reste bloqué jusqu'à réception du premier message de réponse. Il sait alors qu'un processus au moins a reçu le message. Les réponses suivantes ne sont pas prises en compte. Ce schéma de communication est équivalent à une communication fiable point à point avec le processus qui a envoyé le premier message de réponse et à une communication de type datagramme avec les autres processus du groupe.
- réponses multiples : ce schéma de communication est identique au précédent sauf que les messages de réponse qui arrivent après le premier sont mis dans une file du noyau pour l'expéditeur jusqu'au début du Send suivant. C'est au processus de décider combien de réponses il désire recevoir et pendant combien de temps il attend les messages de réponse. Une fois que la première réponse est arrivée, les réponses suivantes sont reçues avec la primitive GetReply non bloquante qui délivre le message de réponse placé en tête de la file des messages de réponses qui sont arrivés et rend l'identificateur du processus ayant envoyé la réponse. Cette primitive est invoquée autant de fois que le processus veut obtenir une réponse. L'invocation de la primitive Receive par les

membres d'un groupe se déroule exactement comme si le message avait été envoyé directement au processus récepteur, indépendamment du fait que c'est l'identité du groupe auquel il appartient qui a été indiquée.

Pid = GetReply (ReplyMessage,TimeOut)

Le message de réponse placé en tête de la file des messages de réponse reçus à la suite d'un Send adressé à un groupe est mis dans ReplyMessage et Pid est alors l'identité du processus qui a envoyé ce message de réponse. Pour avoir les réponses suivantes, il faut invoquer de nouveau la primitive GetReply . C'est à l'expéditeur de déterminer la valeur du timeout. Si aucun message n'est disponible à la fin du timeout, la primitive GetReply rend "0" dans Pid . Toutes les réponses qui arrivent après l'appel suivant de la primitive Send sont écartées.

Pid = Receive (Message)

On suppose que le processus qui a appelé la primitive Receive fait partie du groupe auquel le processus Pid a envoyé un message. La primitive Receive se déroule exactement comme si le message avait été directement envoyé au récepteur.

4. Les transferts de données

Les primitives MoveTo et MoveFrom sont utilisées pour effectuer des transferts de données de taille importante entre les processus. Elles permettent un transfert direct de données entre les espaces virtuels de deux processus. Ce transfert se fait explicitement en indiquant dans les paramètres des primitives l'origine et la destination des données. Par exemple, le récepteur d'un message peut utiliser les primitives MoveTo et MoveFrom pour respectivement écrire ou lire un certain nombre d'octets dans l'espace mémoire de l'expéditeur pendant que ce dernier est bloqué en attente d'un message de réponse. Une fois le transfert effectué, le récepteur envoie une réponse en utilisant la primitive Reply. Le chargement d'un fichier à partir d'une station de travail sans disque est un exemple d'utilisation de la primitive MoveTo. Nous donnons ci-dessous une description de ces deux primitives.

MoveTo (DestinationPid,DestinationPointer,SourcePointer, ByteCount)

ByteCount octets du segment d'adresse de début SourcePointer dans l'espace d'adresses du processus actif sont copiés dans un segment d'adresse de début DestinationPointer dans l'espace d'adresses du processus DestinationPid. Le processus DestinationPid doit être en attente d'une réponse du processus actif et doit lui avoir donné des droits d'accès en écriture sur le segment de destination lors de l'envoi du Send.

Lors de l'exécution de la primitive MoveTo, le noyau décompose le segment à envoyer en un certain nombre de paquets de taille maximum. Il les transmet dans une séquence de paquets de type

RemoteMoveToRequest (le noyau n'attend pas d'acquittement). Les champs significatifs de ces paquets sont :

- identificateur du processus source
- identificateur du processus destinataire
- numéro de séquence
- longueur
- longueur totale
- adresse locale
- adresse distante.

Le champ longueur contient la longueur du segment de données contenu dans le paquet. Le champ longueur totale indique la longueur totale du segment de données sur lequel porte le MoveTo.

Quand la séquence de paquets est arrivée sur la machine de destination, le noyau de cette machine envoie un acquittement par un paquet de type RemoteMoveToReply.

MoveFrom (SourcePid,DestinationPointer,SourcePointer,ByteCount)

ByteCount octets du segment d'adresse de début SourcePointer dans l'espace d'adresses du processus SourcePid sont copiés dans un segment d'adresse de début DestinationPointer dans l'espace d'adresses du processus ayant invoqué la primitive. Le processus SourcePid doit être en attente d'une réponse du processus exécutant le MoveFrom. Il doit avoir donné des droits d'accès en lecture sur le segment lors de l'invocation de la primitive Send.

La mise en œuvre de la primitive MoveFrom est similaire sauf qu'il y a un seul paquet du type "Remote MoveFrom Request" et un plusieurs paquets du type " Remove MoveFrom Reply".

En cas d'erreur, il y a retransmission complète de la séquence de données. Cela pose problème dans le cas où l'erreur est dûe à un processus défaillant.

5.3. Les primitives de manipulation des groupes de processus

Nous donnons ci-dessous les principales primitives de manipulation des groupes de processus.

Groupid = CreateGroup (InitialPid,Type)

La primitive CreateGroup permet de créer dynamiquement un groupe de processus. Le nom alloué à ce nouveau groupe est mis dans GroupId. InitialPid est le premier membre du groupe. Le paramètre Type indique si le groupe est global ou local et s'il est à accès réglementé ou pas (restricted ou unrestricted).

Un groupe est global si ses membres peuvent être répartis sur tous les sites du *V-Domain*. Un groupe local est un groupe dont tous les membres sont situés sur le même site (le site où est situé le premier membre). La distinction entre groupe local et global permet d'optimiser les échanges de messages dans le cas où tous les membres d'un groupe sont situés sur un même site.

Un groupe à accès réglementé sélectionne ses membres. Les processus qui veulent devenir membres du groupe doivent posséder une autorisation. Par contre, n'importe quel processus peut se joindre à un groupe de type non réglementé.

Un certain nombre d'identificateurs de processus sont réservés statiquement par le système. Un identificateur de groupe est désalloué lorsque le dernier membre quitte le groupe.

JoinGroup (GroupId,Pid)

La primitive JoinGroup permet à un processus de devenir membre d'un groupe. Le processus Pid devient membre du groupe GroupId. Un processus peut appartenir à plusieurs groupes.

La primitive provoque l'allocation d'un descripteur de groupe sur le site de résidence du processus. Ce descripteur permet de faire la correspondance entre le groupe et le processus qui y entre.

LeaveGroup (GroupId,Pid)

La primitive LeaveGroup permet à un processus de quitter un groupe. Le processus Pid est retiré du groupe GroupId. Lorsque le dernier processus d'un groupe le quitte, le groupe disparaît.

La primitive LeaveGroup provoque la désallocation du descripteur de groupe associé au processus qui quitte le groupe.

QueryGroup (GroupId,Pid)

Cette primitive indique si le processus Pid peut être autorisé à se joindre au groupe GroupId. Si le processus Pid est déjà membre du groupe ou si le groupe n'existe pas, le processus appelant en est informé. QueryGroup retourne également le nombre de processus du groupe ou plus exactement une approximation de ce nombre puisque des processus peuvent devenir membre du groupe ou le quitter pendant l'exécution de la primitive et que, de plus, il peut y avoir perte de paquets.

DestroyGroup(GroupId)

Cette primitive a pour effet de détruire tous les processus membres du groupe.

La gestion des processus, des groupes et de la mémoire est effectuée par un processus serveur interne au noyau appelé KernelServer. Cela permet d'utiliser les facilités de la communication inter-processus standard lors de l'appel des primitives. Par exemple, la primitive LeaveGroup est en fait une routine qui envoie un message de requête au KernelServer du site. Tous les KernelServers appartiennent au KernelServer Group.

Lors d'une opération concernant un groupe dont les membres sont situés sur plusieurs sites, le message de requête est envoyé au groupe des KernelServers plutôt qu'à un KernelServer particulier. Par exemple, DestroyProcess (GroupId) provoque l'envoi d'un message au groupe des

KernelServers. Chaque KernelServer détruit les processus locaux qui font partie du groupe spécifié. De même, QueryGroup envoie un message au groupe des KernelServers dans le but d'obtenir des informations sur le groupe considéré. Ces opérations peuvent échouer partiellement. Par exemple, lors d'un DestroyGroup, il se peut que des processus ne puissent pas être détruits si l'auteur de la requête n'a pas la permission requise. En général, un code d'erreur est retourné.

Les routines doivent prendre en compte le fait que la communication est non fiable. Par exemple, lors d'un DestroyGroup, il se peut que certains KernelServers ne reçoivent pas le message de requête. Pour parer à cette éventualité, DestroyGroup utilise QueryGroup pour contrôler si tous les membres du groupe ont bien été détruits. Si ce n'est pas le cas, DestroyGroup est rappelé.

Les KernelServers utilisent l'algorithme suivant pour répondre aux requêtes mettant en jeu un groupe. S'il n'a aucune information sur le groupe alors il ne répond pas sinon il envoie une réponse standard. Par exemple, pour un QueryGroup seuls les sites sur lesquels sont situés des membres du groupe répondent. Ainsi, seuls les messages de réponse utiles sont véhiculés. Cela implique que le noyau attende l'expiration du timeout avant de signaler que le groupe considéré n'existe pas car dans ce cas aucun noyau ne répond à la requête.

La primitive JoinGroup est mis en œuvre par l'envoi d'un message au groupe des KernelServers. Si aucun KernelServer ne s'y oppose ou si au moins un l'approuve, le processus devient membre du groupe.

5.4. Analyse des performances

Les performances du V Kernel ont été évaluées par les concepteurs de ce noyau de système réparti. Elles sont difficiles à comparer avec celles des autres noyaux du fait de la diversité des méthodes de mesures utilisées et des différentes machines sur lesquelles sont implantés ces noyaux. Nous décrivons cependant, dans ce paragraphe, la méthode selon laquelle les mesures ont été effectuées et nous donnons un aperçu des résultats obtenus. Le lecteur intéressé trouvera dans [ZWA 85] une étude plus détaillée des mesures et de leur interprétation.

1. Le but des mesures

Le but des mesures effectuées par les concepteurs du V Kernel est double. Il s'agit, d'une part, de comparer le temps écoulé pour une opération à distance au temps écoulé pour la même opération exécutée localement. D'autre part, il est intéressant d'évaluer les performances du V Kernel pour les opérations à distance afin de les comparer aux performances des autres systèmes qui permettent d'effectuer des opérations à distance.

Dans les deux comparaisons, le coût des communications à travers le réseau est un facteur déterminant. La notion de pénalité du réseau a été définie pour évaluer ce coût [CHE 83].

2. La pénalité du réseau

La pénalité du réseau est le temps mis pour transférer n octets dans un datagramme d'une station de travail vers une autre sous les hypothèses suivantes : il n'y a pas d'erreur de transmission et le réseau est très peu chargé.

La pénalité du réseau dépend des performances des processeurs, du réseau, de l'interface réseau et du nombre d'octets transférés. La mesure de la pénalité du réseau donne une évaluation plus réaliste du temps minimum de transfert de données dans le réseau que l'évaluation de ce temps à partir de la vitesse du réseau car la pénalité du réseau comprend le temps de traitement des messages par le processeur et le temps de transit par l'interface réseau. Or, le temps mis par l'interface pour constituer un paquet du côté de l'expéditeur est comparable au temps de transmission du paquet.

Les deux tableaux ci-dessous (cf. figure 16) montrent les résultats obtenus ; le premier tableau consigne les résultats pour un transfert de soixante-quatre octets et le second tableau pour un transfert de mille vingt-quatre octets. Dans chaque cas, les mesures ont été effectuées avec deux microprocesseurs de type M68000 de vitesse différente (8 MHz et 10 MHz) et avec deux réseaux Ethernet de débit différent (3 Mo et 10 Mo).

64 octets

Débit du réseau / vitesse processeur	3 Mo	10 Mo
8 MHz	0.8	0.73
10 MHz	0.65	0.55

1024 octets

Débit du réseau / vitesse processeur	3 Mo	10 Mo
8 MHz	6.95	5.13
10 MHz	5.83	4.26

Figure 16 : Pénalité du réseau.

Les mesures effectuées montrent que la vitesse du processeur influe très largement sur la pénalité du réseau. Le temps de copie des données dans l'interface et l'opération inverse prennent un temps non négligeable devant le temps de transmission, au moins quand le réseau est peu chargé. C'est pourquoi nous pouvions nous attendre à ce que la vitesse du processeur soit un paramètre important dans les mesures de la pénalité du réseau.

Nous donnons maintenant les résultats obtenus pour la comparaison entre les temps écoulés pour une opération effectuée localement ou à distance. Nous n'aborderons pas les résultats obtenus pour la comparaison avec les autres systèmes. Cela nécessiterait une étude plus poussée des méthodes de mesure des performances du V Kernel et des autres noyaux de système réparti qui n'a pas sa place ici.

Les colonnes du tableau ci-dessous (cf. figure 17) donnent respectivement le temps écoulé pour une séquence Send-Receive-Reply pour deux processus locaux, pour deux processus distants, la différence entre ces deux temps, la pénalité du réseau mesurée dans le cas de l'échange à distance, le temps processeur utilisé par le processus client et par le processus serveur. Les mesures ont été effectuées dans quatre configurations de réseau et de processeur indiquées dans les lignes du tableau.

	Nature de l'échange				Temps processeur	
	local	distant	différence	pénalité	client	serveur
8 MHz 3 Mo	1.13	3.18	2.05	1.60	1.79	2.30
10 MHz 3 Mo	0.94	2.54	1.60	1.30	1.44	1.79
8 MHz 10 Mo	1.13	2.68	1.55	1.46	1.59	2.13
10 MHz 10 Mo	0.94	2.23	1.29	1.10	1.30	1.74

Figure 17 : Comparaison d'un échange de messages local avec un échange de messages à distance.

Ces mesures révèlent qu'il y a un certain degré de parallélisme dans la transmission de données. La somme des temps processeur du client et du serveur étant supérieure à la durée écoulée pour l'échange de messages, il y a forcément parallélisme. Par exemple, la transmission du paquet, le bloquage de l'expéditeur et la commutation de processus sur la station de l'expéditeur se déroulent parallèlement à la réception du paquet et à sa lecture par le récepteur sur la station de réception.

Ces mesures se rapportent au cas d'un échange de messages entre deux processus. Considérons maintenant, du point de vue des performances, le cas de l'envoi d'un message à un groupe de processus. Le temps écoulé pour l'envoi d'un message à un groupe dépend de plusieurs facteurs : le nombre de membres dans le groupe, le nombre de membres qui répondent au message, le temps pendant lequel l'expéditeur décide d'attendre les réponses, le caractère global ou local du groupe, la vitesse du microprocesseur et la vitesse de l'interface réseau. Le coût de l'envoi d'un message à un groupe augmente avec le nombre de membres dans le groupe [CHE 85a].

Calculons le temps d'utilisation du processeur pour une communication de groupe. Nous nous plaçons sous les hypothèses suivantes : il y a N membres dans le groupe (expéditeur du message non compris), tous les processus du groupe sont sur des sites différents, les sites où il n'y a aucun membre du groupe ne reçoivent rien (le temps d'utilisation du processeur sur ces sites est nul). Nous considérons un cas où il n'y a pas perte de paquet. Le temps d'utilisation du processeur dépend du nombre de réceptions et d'émission de paquets. On appelle évènement toute émission ou réception de paquet. On suppose que le temps d'utilisation du processeur pour une réception est identique à celui pour une émission. La communication de groupe nécessite 3N + 1 évènements dont :

- l'émission de la requête destinée au groupe de processus,
- N réceptions de la requête,
- N émissions de réponses à la requête,
- la réception des N réponses par l'expéditeur de la requête.

Si la communication de groupe est simulée avec le mécanisme de communication point à point, le nombre d'évènements est de 4N dont 2N évènements du côté de l'expéditeur et 2N du côté des récepteurs du message. La communication de groupe permet donc un gain sur le temps d'utilisation du processeur de l'expéditeur du message.

6. APPLICATION : LE SYSTEME DE GESTION DE FICHIERS (SGF)

Les noyaux de systèmes distribués fournissent un certain nombre de primitives de base à l'aide desquelles sont construites les applications. Les services généralement disponibles dans les systèmes d'exploitation traditionnels sont considérés comme des applications dans certains noyaux de systèmes distribués tels que le V Kernel. Il en est ainsi pour le système de gestion de fichiers, service clé des systèmes d'exploitation.

Dans une première partie, nous allons étudier les différents problèmes de conception d'un service de gestion de gestion de fichiers réparti puis nous décrirons le SGF du V Kernel.

6.1. Les caractéristiques d'un SGF réparti

Le rôle d'un SGF est de permettre aux usagers de stocker des données de façon permanente et d'accéder ces données. Il y a plusieurs motivations pour concevoir un système de gestion de fichiers distribué. Un SGF réparti doit permettre le partage des fichiers entre plusieurs usagers qui travaillent sur différentes stations de travail. En outre, un SGF réparti doit assurer la transparence du partage. Un SGF réparti doit respecter un certain nombre de propriétés sur les fichiers. Premièrement, un SGF doit assurer la consistance des données c'est à dire que tous les usagers doivent percevoir la mise à jour d'un fichier à partir de l'instant de modification. Il faut garantir l'intégrité des données ; deux usagers ne doivent pas mettre à jour en même temps les mêmes données (problème de la mise à jour d'un compte bancaire). De plus, tous les usagers doivent pouvoir accèder un fichier en utilisant un nom unique. C'est à dire que la localisation d'un fichier doit être transparente. Un fichier doit pouvoir changer de nœud de résidence sans changer de nom. Enfin, toutes les stations de travail doivent être soumises aux mêmes mécanismes de contrôle des accès aux fichiers. On peut ajouter qu'un SGF réparti doit être performant.

Les SGF répartis se sont développés avec l'utilisation fréquente de stations de travail sans disque qui ont essentiellement deux avantages : un coût moins élevé et une amélioration du confort des usagers car les unités de disques sont souvent bruyantes et dégagent de la chaleur.

Pour de nombreux SGF, le terme de serveur de fichiers est impropre. En effet, ce sont

souvent de simples serveurs de stockage des données. Ils ne remplissent pas complètement la tâche de gestion des fichiers. Le contrôle des accès concurrents, la détermination des droits d'accès, la gestion des directories sont quelquefois effectués par d'autres serveurs ou sont laissés sous la responsabilité des applications.

Les SGF offrent aux clients une interface constituée d'opérations primitives sur les fichiers : création, destruction, ouverture, fermeture d'un fichier, lecure ou écriture de données du fichier. Il faut faire la distinction entre client et usager. En général, la couche client est une couche intermédiaire entre le serveur de fichiers et les usagers. Les applications de haut niveau n'utilisent pas directement les primitives offertes par le SGF mais plutôt l'interface de la couche client. La couche client permet de s'adapter aux différentes organisations des données des clients.

Les problèmes à résoudre par les serveurs de fichiers ne sont pas tous liés au fait que la communication entre clients et serveur distants se fasse au travers d'un réseau mais cela introduit cependant des problèmes supplémentaires. Ces problèmes tiennent au fait que les lignes de communication ne sont pas fiables à 100 %, que l'un des deux sites peut être défaillant (l'autre site doit pouvoir réagir à cette situation) et que la gestion des échanges de messages est non négligeable. En outre, ces problèmes se posent non seulement entre client et serveur mais également entre serveurs lorsque le SGF est mis en œuvre de manière répartie.

Nous pouvons décomposer l'étude des fonctions d'un service de gestion de fichiers en cinq rubriques sachant que tous les serveurs de fichiers disponibles dans les noyaux de systèmes distribués ne fournissent pas directement toutes les fonctionnalités décrites ci-dessous.

1. Le système de stockage des objets

Le système de stockage des objets est responsable de la création, de la suppression, du stockage des fichiers. Il est également chargé de localiser un fichier à partir de son nom interne. Il contrôle aussi les accès des usagers.

Dans le système Apollo/Domain, à chaque objet est associé son uid, la liste de contrôle des accès, un identificateur de type et la date de dernière modification. Le système de stockage des données, mis en œuvre de façon répartie, s'adresse au "hint manager" pour localiser les objets. Si l'objet est local, l'accès a lieu sur le disque local, si l'objet est distant la requête est transmise au système de stockage distant. Afin d'accélérer les accès aux données, les pages les plus récemment utilisées sont stockées dans un cache. Les données dans le cache sont validées grâce à la date de dernière modification [LEV 86].

2. Le mécanisme d'accès aux données

On distingue plusieurs modes d'accès aux fichiers : fichier entier (par exemple, chargement d'un programme sur une station de travail sans disque à partir d'un serveur de fichiers), par pages solution efficace pour l'accès aux données depuis une station de travail sans disque, par taille variable. De plus, il y a plusieurs types de fichiers :

- les fichiers ordinaires,

- les fichiers "recouvrables", ce sont des fichiers qui peuvent être ramenés dans leur dernier état consistant en cas de terminaison anormale d'une opération ou d'une défaillance du client. Cette propriété est souvent garantie par le mécanisme des pages fantômes. Le SGF de Locus [POP 83b], par exemple, y a recours. Lors de la première modification d'une page (ou à la création d'une nouvelle page), une page fantôme sur disque est allouée. Les adresses de ces pages sont reportées dans la copie du descripteur en mémoire centrale mais non dans la copie sur disque. Les modifications ont lieu sur les pages fantômes. Si l'opération d'écriture est annulée, il suffit de désallouer le descripteur en mémoire et les pages fantômes ; le fichier est resté dans son état initial. Pour valider une opération d'écriture, le descripteur en mémoire centrale est recopié sur disque. Les anciennes pages du fichier sont alors libérées.

- fichiers "robustes", ces fichiers résistent aux défaillances du périphérique de stockage. Cette résistance est obtenue, en général, par duplication des informations dans une double unité de disque.

Les accès aux données doivent permettre de garantir la consistance des données. Cela est possible pour les fichiers de type "recouvrable". Pour garantir l'intégrité des données, les mises à jour doivent être atomiques. Une opération sur un fichier est dite atomique si dans le cas où l'opération se termine avec succès, le fichier dont l'état initial était compatible avec l'opération passe dans un nouvel état cohérent avec la sémantique de l'opération et que si l'opération échoue, le fichier reste inchangé. La mise à jour atomique des fichiers est un problème non trivial même lorsqu'elle ne porte que sur un fichier. Les opérations qui portent sur des opérations composées qui comportent plusieurs accès au même fichier ou plusieurs accès à des fichiers différents doivent parfois possèder la propriété d'atomicité. Par exemple, cela est nécessaire pour mettre à jour une base de données dont les contraintes d'intégrité doivent être respectées. De telles opérations sont appelées transactions. Le SGF de Locus comporte un mécanisme de vaklidation pour les accès en écriture qui permet de rendre atomique un ensemble de modifications sur un fichier. Le SGF utilise la méthode des pages fantômes.

De nombreux SGF utilisent des algorithmes de validation à deux phases pour mettre à jour les fichiers. Le lecteur trouvera une description de cet algorithme dans [RAY 85].

Afin d'augmenter la disponibilité, les fichiers dans Locus peuvent être répliqués [WAL 83]. Le système assure le maintien de la cohérence des copies et garantit que les accès sont faits sur la copie la plus à jour. En cas de partitionnement du réseau, le mécanisme des vecteurs de version permet de détecter l'inconsistance des copies [PAR 83]. Pour certains types de fichiers, tels que les catalogues, il est possible de "réconcilier" automatiquement les copies, c'est à dire de reconstituer une copie consistante à partir des différentes copies inconsistantes. Sinon le système fournit des outils permettant au propriétaire du fichier de "réconcilier" les copies [POP 83a].

3. Le mécanisme de contrôle de la concurrence

L'aspect "tout ou rien" n'est pas le seul aspect des opérations atomiques. Le deuxième aspect des opérations atomiques est le contrôle de la concurrence qui permet de rendre les transactions

indivisibles pour les autres transactions qui s'exécutent parallèlement. Le contrôle de la concurrence est basé sur la possibilité de réserver une portion de fichier pour une transaction particulière de façon à empêcher les autres transactions d'y accéder. Les SGF possèdent des mécanisme de verrouillage des pages ou des fichiers. Le contrôle de la concurrence induit le problème de l'interblocage qui peut être réglé soit par prévention soit par détection. Des algorithmes sont décrits dans [RAY 85].

Dans le système Apollo/domain, le SGF comporte un gestionnaire de verrous distribué. Il y a dans ce noyau de système deux modes de verrouillage. Le premier mode permet d'avoir pour un fichier soit un écrivain soit un ou plusieurs lecteurs. Le deuxième mode ne donne aucune restriction sur le nombre de lecteurs ou d'écrivains mais ceux-ci doivent être tous situés sur le même site. Bien sûr, ce deuxième mode qui permet le partage de mémoire n'assure pas la consistance des données lues par les lecteurs. Pou éviter l'interblocage, le SGF du système Apollo/Domain satiusfait les requêtes sur le champ ou pas du tout.

Dans le système Locus, le contrôle de la concurrence est réglé par un mécanisme de jeton sous le contrôle d'un processus gérant unique. Il y a deux niveaux de partage dans Locus : le partage du pointeur de lecteure/écriture (1) et la partage du descripteur de fichier (2). Dans le cas (1), il y a exclusion mutuelle et un jeton unique est attribué. Dans le cas (2), un seul rédacteur ou plusieurs lecteurs sont autorisés.

4. Le mécanisme de désignation des fichiers

L'espace des noms symboliques des fichiers est en général hiérarchique. Dans plusieurs noyaux de systèmes distribués, le service de désignation symbolique, qui permet d'associer un nom symbolique à un nom interne, est rendu par le serveur de désignation (voir chapitre 3). C'est le cas dans le système Apollo/Domain par exemple.

5. Le mécanisme d'authentification des usagers

Le contrôle des droits d'accès peut se faire soit par capacité soit par liste de contrôle des accès. L'authentification peut s'effectuer soit par un serveur spécialisé comme c'est le cas dans le V Kernel, soit par le serveur de gestion des fichiers. Dans le système Apollo/Domain, le contrôle se fait par liste de contrôle des droits d'accès. Lors de la connexion, en cas de succès, le système retourne à chaque utilisateur un identificateur unique qui le représente. Cet identificateur est un quadruplet (identificateur d'usager, identificateur de projet, identificateur d'organisation, identificateur de nœud de login). Le système comporte un registre des utilisateurs répliqué. Chaque nœud connait l'adresse d'un regsitre valide. Chaque objet du système comporte comme attribut une liste de contrôle des accès qui contient les identificateurs des usagers qui peuvent accéder l'objet et les droits d'accès associés. Cette liste est consultée lors des accès.

6.2. Le SGF du V Kernel

Comme les stations de travail du V Kernel sont pour la plupart des stations de travail sans disque, le serveur de fichiers prend une place importante. Le serveur de fichiers ne fait pas partie du noyau distribué, c'est un groupe de processus qui s'exécute au dessus du noyau. Chaque serveur de fichiers est structuré en une équipe de processus. Une équipe de processus est un ensemble de processus qui résident sur un seul site et partagent le même espace d'adresses. Le processus principal, qui fait partie du groupe des serveurs de fichiers, gère les catalogues de désignation des fichiers et répond aux requêtes d'ouverture des fichiers. La désignation des fichiers est hiérarchique (cf. chapitre 3). Les autres processus de l'équipe effectuent les lectures et écritures des données dans les fichiers. Les blocs de fichiers les plus utilisés résident en mémoire principale dans un cache.

Dans le V Kernel, il existe un protocole standard pour toutes les entrées/sorties. Les clients utilisent des requêtes qui respectent ce protocole pour accéder les données gérées par les serveurs. Le serveur de fichiers respecte ce protocole pour les requêtes d'accès aux fichiers. Nous décrivons maintenant ce protocole [CHE 87].

Pour effectuer des opérations sur un fichier, il est nécessaire de créer une instance de ce fichier. Le protocole est orienté objet dans la mesure où toutes les opérations s'appliquent à un objet : l'instance de fichier. Les opérations sont possibles sont : créer une instance de fichier, demander des informations sur une instance de fichier, changer le propriétaire de l'instance de fichier, lire , écrire et relâcher l'instance de fichier. Lors de sa création, l'instance de fichier est initialisée de façon à contenir les mêmes données que le fichier permanent associé. Lorsque l'instance de fichier relâchée par le client, les informations qu'elle contient sont recopiées de façon atomique dans le fichier permanent. Les modifications de l'instance de fichier peiuvent être reflètées sur le fichier permanent à chaque accès. Les opérations fournies par l'interface sont décrites ci-dessous.

Create_Instance :

Paramètres d'entrée : - nom du fichier
- mode d'ouverture (lecture, création, ajout, modification, exécutable, directory)

Paramètres de sortie: - code
- identificateur d'instance
- identificateur de processus serveur
- taille maximale d'un bloc
- attributs de type
- index sur le dernier bloc
- nombre d'octets dans le dernier bloc
- numéro du dernier bloc à lire

Cette opération provoque l'allocation d'une instance de fichier associée au fichier dont le nom est passé en paramètre. Le fichier est ouvert dans le mode indiqué.

Query_Instance :

Paramètres d'entrée : - identificateur d'instance
Paramètres de sortie: - code
- identificateur d'instance
- identificateur de processus serveur
- taille maximale d'un bloc
- attributs de type
- index sur le dernier bloc
- nombre d'octets dans le dernier bloc
- numéro du dernier bloc à lire

Cette opération rend en paramètre de sortie les informations sur le fichier. Un code d'erreur est indiqué si par exemple le fichier ou l'instance de fichier n'existe pas.

Release_Instance :

Paramètres d'entrée : - identificateur d'instance
- numéro du bloc à lire
- mode de relâchement

Paramètres de sortie: - code

Les données sont écrites atomiquement sur l'ancienne version du fichier. Si le mode de relâchement est différent de zéro, les données sont abandonnées.

Read_Instance :

Paramètres d'entrée : - identificateur d'instance
- numéro du bloc à lire
- adresse d'un tampon où mettre les données lues (ce tampon sert uniquement le nombre d'octets lus est supérieur au nombre maximal d'octets qui peuvent être mis dans un seul message)
- nombre d'octets à lire

Paramètres de sortie: - code
- identificateur d'instance
- données lues (si leur taille n'exède pas la taille maximale des données dans un message)
- nombre d'octets lus

Cette opération permet de lire des données de taille variable dans un fichier.

Write_Instance :

Paramètres d'entrée : - identificateur d'instance
- numéro du bloc à écrire
- données à écrire si celles-ci ne sont pas trop longues
- adresse du tampon où se trouvent les données à écrire si celles-ci sont trop

longues

- nombre d'octets à écrire

Paramètres de sortie: - code

- nombre d'octets écrits

Cette opération permet d'écrire des données de taille variable dans un fichier.

Ce sont les principales opérations. Il existe d'autres opérations telles que Set_Instance_Owner qui permet de changer le propriétaire d'un fichier.

Le protocole uniforme d'entrée/sortie permet de gérer la réplication des fichiers. Les accès en lecture sont faits sur une copie quelconque, les accès en écriture sont répercutés sur toutes les copies. Il est possible également de poser un verrou sur un fichier entier ou sur un bloc de fichier. Ce mécanisme permet de mettre en œuvre des opérations atomiques. Un client qui envoie une requête incompatible avec un verrou est retardé pendant un temps fixé au bout duquel, si le verrou n'est pas levé, un code d'erreur est envoyé. Cela permet de prévenir l'interblocage. Le protocole uniforme d'entrée/sortie permet de mettre en œuvre un protocole logiquement séparé de transaction atomique. Les opérations sur un fichier peuvent être introduites dans une transaction.

Le mécanisme sous_jacent est bien sûr la communication par messages. Il est possible soit d'accéder aux pages du fichier soit d'accéder séquentiellement le fichier, soit de transférer le fichier tout entier. L'accès aux pages est mis en œuvre par un échange du type Send-Receive-Reply. Le transfert d'un fichier entier se fait en deux étapes : un échange Send-Receive-Reply permet de charger l'entête du fichier, les données sont ensuite transférées par des échanges utilisant une ou plusieurs fois l'une des primitives MoveTo ou MoveFrom.

Le temps écoulé pour une lecture locale est de 1,69 ms, pour une lecture à ditance de 4,54 ms sans compter le temps de recherche sur disque que l'on peut évaluer à environ 20 ms. On obtient le même ordre de grandeur pour une écriture. La différence entre les deux temps (2,85 ms) est faible comparé au temps total de l'accès comprenant la recherche sur disque. Le temps de transfert d'un fichier entier distant dépend du nombre d'octets transférés pour chaque appel des primitives MoveTo et MoveFrom. Pour un fichier de 64 Ko, avec une unité de transfert de 1 Ko le temps est de 376,1 ms et avec une unité de transfert de 64 Ko il est de 198 ms. Le gain de temps pour les transferts de fichiers entiers est non négligeable lorsque les octets sont rangés de manière contigue en mémoire de façon à minimiser le nombre d'invocations des primitives MoveTo ou MoveFrom pour un transfert. (Les primitives MoveTo et MoveFrom supposent en effet que les données sont contigues en mémoire.).

7. CONCLUSION

Nous avons étudié les concepts mis en œuvre dans les noyaux de systèmes répartis. Le rôle d'un noyau de sytème réparti est de fournir des primitives de base facilitant la conception des sous-systèmes constituant un système réparti. Les noyaux de systèmes répartis fournissent généralement des primitives de gestion des processus et de la communication entre les processus.

Il semble que les noyaux de systèmes répartis à objets prennent le relais des noyaux de systèmes répartis à messages. En effet, l'écriture des applications distribuées est facilitée par le concept d'objet qui permet d'encapsuler un ensemble de processus coopérants. Le V Kernel, qui est fondé sur la communication par messages, propose le groupe de processus. Cependant, le concept de groupe de processus ne remplace pas le concept de processus. Le V Kernel possède l'avantage de laisser une grande liberté pour la structuration des sous-systèmes tandis que les noyaux de systèmes répartis à objets imposent un mode de structuration.

Nous avons vu que les noyaux de systèmes répartis ne mettent généralement pas en œuvre des mécanismes assurant la fiabilité des applications ou permettant de résister aux pannes. Le V Kernel ne fait pas exception puisqu'il reporte ces problèmes aux niveaux supérieurs. Cependant, il serait bon de disposer de primitives fournies par le noyau pour mettre en œuvre simplement et de manière performante des protocoles assurant l'atomicité. En effet, il est établi que l'atomicité est un concept de base dans les applications distribuées. De plus, le noyau est la couche système la plus basse et donc à mon avis la mieux adaptée pour assurer la tolérance aux fautes qui repose sur des mécanismes matériels.

L'installation du V-KERNEL sur station de travail SUN n'est pas encore terminée à cause de problèmes apparus lors de la compilation des fichiers sources. Le compilateur fourni présente en effet une anomalie de fonctionnement puisqu'il détecte systématiquement une erreur à la première ligne des programmes. Lorsque l'installation sera effective, nous pourrons évaluer concrètement le V-KERNEL et sans doute en tirer des leçons pour la réalisation de Gothic.

REFERENCES

[BAL 86] Balter R., Krakowiak S., Meysembourg M., Roisin C., Rousset de Pina X., Scioville R., Vandôme G.
Principes de conception du système d'exploitation réparti GUIDE.
Bigre+Globule n° 52, Dec 86.

[BAN 84] Banâtre Michel.
Le système ENCHERE : une expérience dans le conception et la réalisation d'un système réparti.
Thèse d'Etat, Université de Rennes 1, Mars 84.

[BAN 86a] Banâtre J.P., Banâtre M., Ployette F.
An overview of the GOTHIC distributed operating system.
Rapport de recherche n° 504 INRIA/IRISA Mars 86.

[BAN 86b] Banâtre J.P.
New concepts for distributed system structuring.
Paper prepared for the NATO Advance Study Institute Distributed Operating Systems : Theory and Practice.
Izmir, 18-29 Août 86.

[BAN 86c] Banâtre J.P.
Gothic en quelques clichés.
Article proposé pour les 2émes journées INRIA/université de Barcelone.
St Malo 6-7 Nov 86.

[BER 84] Berglund E., Cheriton D.
Amaze : a distributed multiplayer game program using the distributed V Kernel.
IEEE May 84 pp. 248-253.

[BIR 82] Birrell A., Levin R., Needham R., Schroeder M.
Grapevine : An exercise in distributed computing.
Com. of the ACM, Vol 25, n° 4, April 82, pp. 260-274.

[BIR 84] Birell A., Nelson B.
Implementing Remote Procedure Calls.
ACM TOCS, Vol 2, N° 1, Feb. 84, pp. 39-59.

[BLA 85] Black A. P.
Supporting distributed applications : experience with EDEN.
Proc. 10th ACM Symp. on Operating Systems Principles, SIGOPS Operating Systems Review, vol 19,5 (dec. 95), pp. 181-193.

[BLA 86] Black A.P., Hutchinson N., Jul E., Levy H.
Object structure in the Emerald system,
Proc. First ACM Conf. on Object-Oriented Programming Systems, Languages and Applications, Portland, Oregon, SIGPLAN Notices, vol. 21, 11 (nov 86), pp.78-86.

[CHE 83] Cheriton D., Zwaenepoel W.
The distributed V Kernel and its performance for diskless workstations.
Proceedings of the 9th symp. on operating system principles. ACM 1983 pp. 248-253.

[CHE 84] Cheriton D., Mann T.
Uniform access to distributed name interpretation in the V system.
IEEE 1984 pp.290-297.

[CHE 85a] Cheriton D., Zwaenepoel W.
Distributed Process Group in the V Kernel.
ACM TOCS, Vol 3, n° 2, May 85, pp. 77-107.

[CHE 85b] Cheriton D.
Preliminary thoughts on problem-oriented shared memory : a decentralized approach to distributed systems.

[CHE 87] Cheriton D.
UIO : A uniform I/O system interface for distributed systems.
ACM TOCS Vol 5, n° 1, Feb 87, pp. 12-46.

[ENG 83] English R., Popek G.
Dynamic reconfiguration of a distributed operating system.

[FIT 85] Fitzgerald R., Rashid R.
The integration of virtual memory management and interprocess communication in Accent.
ACM TOCS Vol. 4, n° 2, May 86 pp. 147-177.

[FOR 86] Forman I.R.
Raddle, an informal introduction.
MCC Technical Report N° STP 182-85, FEB 86.

[LEA 83] Leach P., Levine P., Douros B., Hamilton J., Nelson D., Stumpf B.
The architecture of an integrated local network.
IEEE Journal on SAC, Vol 1, N° 5, Nov. 83, pp. 842-856.

[LEG 86] Legatheaux Martins J.
La désignation et l'édition de liens dans les systèmes d'exploitation répartis.
Thèse n° 76, Université de Rennes I, Nov. 86.

[LEV 86] Levine H. Paul.
The Apollo Domain Distributed File System.
Nato Advanced Study Institute. Distributed Operating Systems : Theory and Practice.
Izmir, Turkey 18-29 August 86.

[LIS 84] Liskov B.
The Argus Language and System.
LNCS 190, 1984, pp. 343-430.

[NOT 87] Notkin D., Hutchinson N., Sanislo J., Schwartz M.
Heterogeneous computing environments : report on the ACM SIGOPS Workshop on accommodating heterogeneity.
Com. of the ACM Vol 30, n° 2, Feb. 87, pp. 132-140.

[PAR 83] Parker D., Popek G., Rudisin G., Stoughton A., Walker B., Walton E., Chow J., Edwards D., Kiser S., Kline C.
Detection of mutual inconsistency in distributed systems.
IEEE transactions on software engineering, Vol. 9, n° 3, May 83.

[POP 83a] Popek G., Thiel G., Kline C.
Recovery of Replicated Storage in Distributed Systems.

[POP 83b] Popek G., Walker B., Chow J., Edwards D., Kline C., Rudisin G., Thiel G.
LOCUS, a network transparent, high reliability distributed system.

[RAS 81] Rashid R., Robertson G.
Accent : a communication oriented network operating system kernel.
Proc. 8th. Symp. on operating systems principles.
ACM operating systems review, Vol. 15, n° 5, Dec 1981.

[RAY 85] Raynal Michel.
Algorithmes distribués et protocoles.
Eyrolles 1985.

[ROC 87] Rochat B.
Adapter la gestion de la mémoire à la mise en oeuvre du parallélisme.
Stage de D.E.A. Juin 87.

[TAN 85] Tanenbaum A., Van Renesse R.
Distributed Operating Systems.
ACM Computing surveys, Vol. 17, n° 4, Dec. 85, pp. 419-470.

[WAL 83] Walker B.,Popek G.
The LOCUS distributed file system.
Proc. 9th. Symp. on operating systems principles.
ACM operating systems review Vol. 17, n° 5, Dec. 83.

[ZWA 85] Zwaenepoel W.
Message passing on a local network.
Stanford University, Report n° STAN-CS-85-1083-Oct 85.