Etat de l'art des communications basées sur Mach

Julien Simon *
Julien.Simon@freenix.fr

29 mars 1995

1 Objectifs

Notre objectif est d'offrir des services de communication susceptibles de satisfaire les besoins de toutes les composantes du système Masix, c'est à dire :

- les serveurs du système d'accueil ;
- les serveurs qui composent les différents environnements ;
- les applications qui s'excécutent au-dessus de ces environnements ;

Il va sans dire que chacune d'entre elles imposent des contraintes bien spécifiques au serveur réseau. Cependant, il est possible d'en dégager des besoins communs, que nous allons détailler un par un.

1.1 Transparence des communications inter-processus

Afin de concevoir le système Masix de manière cohérente et d'en faciliter le développement, il est absolument indispensable que les communications inter-processus distantes s'effectuent selon les mêmes sémantiques que les communications locales.

Deux tâches qui souhaitent communiquer ne doivent en aucun cas préjuger de leur localisation respective. En effet, certains mécanismes, comme la migration, peuvent provoquer le déplacement d'une tâche d'un noeud du réseau vers un autre noeud. Par ailleurs, il est tout à fait envisageable que plusieurs occurences d'un même serveur coexistent sur le réseau, afin d'assurer une meilleure tolérance aux fautes. Dans ce cas, une tâche ne peut pas savoir à priori laquelle d'entre elles va traiter sa requête.

* This work has not been financed by the MASI Laboratory

Ces deux exemples illustrent le premier principe fondamental qui guidera la conception du serveur réseau, à savoir qu'il est absolument impératif que toutes les entités communicantes aient l'illusion de s'exécuter sur la même machine.

1.2 Performances maximales

Les performances des IPC ont-elles une importance dans les systèmes basés sur un micronoyau ? La question parait surprenante, mais [Bershad 1992] affirme qu'elles ne sont ne constituent pas un goulot d'étranglement, et qu'il est par conséquent inutile de les optimiser. Par contre, [Hsieh *et al.* 1993, Condict *et al.* 1993, Condict *et al.* 1994] affirment que les IPC sont au contraire le problème majeur de ces systèmes.

Pour notre part, notre objectif en termes de performances est simple : atteindre, voire dépasser, les performances d'un système monolithique. Cet objectif est certes ambitieux, mais en aucun cas irréalisable dans un système basé sur un micro-noyau.

De nombreuses optimisations ont déjà été proposées et montrent que cet objectif peut être atteint. Nous les examinerons en détail lors de l'état de l'art.

1.3 Support multi-protocoles

Le système Masix est un système multi-environnements. Par conséquent, le serveur réseau doit être capable de gérer les protocoles de communication propres à chaque environnement. De plus, il doit être possible d'ajouter ou de retirer dynamiquement un protocole dans le serveur réseau. En effet, il serait inconcevable de le redémarrer à chaque fois qu'un environnement est lancé ou se termine.

Afin d'atteindre ces objectifs, mais aussi afin d'en faciliter le développement, le serveur réseau devra posséder une structure modulaire, tout en conservant des performances élevées.

1.4 Sécurité des communications

La mise en place d'un système réparti comme Masix aggrave les problèmes de sécurité traditionnellement posés par les communications. En effet, les serveurs qui le composent peuvent échanger des informations par le réseau. Si ces informations étaient écoutées ou altérés, la sécurité et l'intégrité du système pourraient s'en trouver gravement compromises.

Nous veillerons donc à garantir la confidentialité des données qui circulent sur le réseau, grâce à une authentification des entités communicantes et un cryptage des données.

2 Etat de l'art des communications dans Mach

2.1 Communications en mode noyau

2.1.1 Communications inter-processus de Mach 3.0

Mach offre des mécanismes puissants de communication entre tâches [Draves 1990]. Les communications sont effectuées par des ports. Un port est simplement une file dans laquelle des messages peuvent être ajoutés et retirés. Les opérations sur les ports sont effectuées via des droits sur ces ports qui sont accordés aux tâches. Trois types de droits existent :

1. droit de réception, accordé à une seule tâche ;
2. droits d'émission, accordés à plusieurs tâches ;
3. droits d'émission unique, permettant à des tâches d'envoyer un — et un seul — message sur ce port.

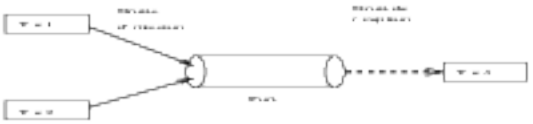

Figure 1: Communication par ports dans Mach

Chaque port est géré par une seule tâche qui possède le droit de réception sur ce port. Certains ports privilégiés sont gérés directement par le noyau Mach lui-même.

Les tâches accèdent aux ports via des noms de ports (identificateurs numériques) qui sont convertis de manière interne en droits par le noyau.

Une tâche peut envoyer ou recevoir des messages sur des ports. Un message est simplement une structure contenant des données. Un message est composé :

- d'une en-tête décrivant le message : taille du message, nom du port d'émission, nom du port de réception, type du message, code opération ;
- d'un ensemble de données typées : type de données, nombre de données, valeurs.

Mach offre de nombreuses options pour l'envoi et la réception de messages. Quand un message est envoyé sur un port, la file correspondante peut être pleine. Si elle n'est pas pleine, le message est copié dans la file. Si la file est pleine, la tâche émettrice a le choix entre quatre options :

1. attendre indéfiniment qu'un message soit lu depuis la file : la tâche émettrice est suspendue jusqu'à ce qu'une autre lise un message depuis le port et libère ainsi la place nécessaire au stockage du message dans la file ;
2. attendre en spécifiant un délai : la tâche émettrice est alors suspendue jusqu'à ce qu'une autre lise un message depuis le port et libère ainsi la place nécessaire au stockage du message dans la file ou jusqu'à ce qu'un délai spécifié soit écoulé ;
3. ne pas attendre, le message n'est pas ajouté à la file si elle est pleine et la tâche émettrice est préverme par un code d'erreur ;
4. transmettre le message à Mach, le noyau Mach stocke le message et se charge de l'ajouter dans la file quand une place est libérée ; un seul message peut ainsi être transmis à Mach pour une file pleine par tâche.

Mach offre de nombreux appels système permettant :

- d'allouer ou désallouer des ports ;
- de transmettre des droits sur ports en :
  - copiant un droit d'émission pour une autre tâche, les deux tâches possèdent alors ce droit ;
  - offrant un droit d'émission ou de réception à une autre tâche, seule la deuxième tâche possède alors ce droit ;
  - transmettant un droit de réception à une autre tâche en le convertissant en droit d'émission, la première tâche possède alors un droit de réception et la deuxième un droit d'émission sur le même port ;
- d'associer des noms aux ports ;
- d'envoyer et recevoir des messages sur des ports.

Mach offre la possibilité de regrouper plusieurs droits de réception sur des ports en un ensemble de ports. Une tâche peut regrouper plusieurs ports en un tel ensemble et recevoir des messages sur cet ensemble. Dans ce cas, le premier message reçu sur l'un des ports faisant partie de l'ensemble est transmis à la tâche.

Cette possibilité est particulièrement utile dans le cas d'un serveur chargé de gérer plusieurs objets représentés par des ports : un ou plusieurs *threads* peuvent se placer en attente sur un ensemble de ports et être réveillé lors de la réception de la première requête.

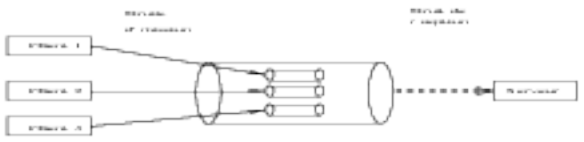

Figure 2: Ensemble de ports

Les IPC locales de Mach comportent quatre phases :

1. copyin : le message est copié de l'espace d'adressage de la tâche vers une mémoire tampon du noyau. Les données out-of-line et les ports contenus dans le message sont convertis en représentations internes au noyau.
2. queuing : le message est placé dans la file d'attente du port de destination ;
3. dequeuing : le message est lu et converti.
4. copyout : le message est recopié dans l'espace d'adressage de son destinataire ;

La structure d'un message Mach, illustrée par la figure 3, est la suivante :

- une entête de taille fixe, qui contient :
  - le port de destination ;
  - le port de réponse ;
  - la taille du message ;
  - le numéro de séquence ;
  - des flags, indiquant certaines propriétés du message ;

- un corps de taille variable, composé d'une ou plusieurs données. Chaque donnée est précédée d'un descripteur qui précise son type. Dans le cas où la donnée est OOL, seule l'adresse de la donnée est placée dans le message.

Figure 3: Structure d'un message Mach

2.1.2 Extension des IPC à un réseau de processeurs

[Barrera 1991] décrit un mécanisme d'extension des IPC locales de Mach 3.0 à l'échelle d'un réseau de processeurs. Il peut s'agir de stations de travail reliées par un réseau local ou d'une machine multiprocesseurs sans mémoire partagée.

Les objectifs de ce mécanisme sont :

- latence minimale ;
- intégration avec les IPC existantes [Draves 1990] ;
- gestion des ports globaux : capacités, décompte des références, ...

Latence minimale : plusieurs optimisations permettent de réduire le temps nécessaire à la réception ou l'envoi d'un message :

- éviter des changements de contexte, notamment en modifiant le thread d'interruption pour qu'il effectue le plus de travail possible ;
- éviter des copies multiples des données :

- en partageant une mémoire tampon entre les applications et le pilote de périphérique. Cependant, cette solution pose des problèmes de sécurité car plusieurs applications se partagent une mémoire tampon unique. Le découpage de cette mémoire en plusieurs parties résoud ce problème, mais limite la taille maximale des messages.
- en mappant les données dans l'espace d'adressage du noyau. Cependant, cette méthode n'est pas adaptée aux données out-of-line et pose également des problèmes de sécurité, liées au fonctionnement de la mémoire virtuelle de Mach. En effet, les pages mémoire mappées par le pilote risquent d'être modifiées par l'application avant d'être envoyées. Un certain nombre de solutions ont été étudiées pour résoudre ce problème.

Intégration avec les IPC existantes : Comme nous l'avons vu au 2.1.1, les IPC locales ont lieu en quatre étapes. L'intégration des IPC distantes intervient lors de la seconde étape : lorsque le message est sur le point d'être placé dans la file d'attente, le noyau détermine s'il est destiné à une tâche distante. Si c'est le cas, il est converti, puis transmis sur le réseau. La conversion est différente : les données out-of-line sont recopiées dans le message et les ports traduits en ports globaux. Le message est traité de manière symétrique lors de sa réception.

Gestion des ports globaux :

- nommage global : chaque port est nommé de manière unique, ce qui permet de déterminer la destination d'un message et de transmettre des droits sur un port entre deux tâches. L'identificateur d'un port contient des informations locales, qui facilitent les envois de message. Cependant, ces informations deviennent obsolètes lorsque le port migre. Par conséquent, une tâche qui migre doit changer de ports globaux.
- utilisation d'un mandataire (ou *proxy*) ;
- détection de l'absence d'émetteurs (ou *no-senders*) ;
- mort des ports ;
- migration des ports ;

Fiabilité des IPC : l'utilisation d'un réseau de communication impose certains mécanismes permettant de garantir la fiabilité des communications, c'est à dire l'arrivée à bon port de tout message émis.

Ces mécanismes dépendent de la fiabilité du réseau :

- réseau fiable : acquittement négatif, envoyé lorsque le destinataire ne dispose pas de suffisamment de mémoire pour recevoir le message ;
- réseau non fiable :
  - acquittement positif, envoyé à chaque réception d'un message ;

– si l'acquittement n'est pas reccu, le message est retransmis aprés expiration d'un temporisateur ;

2.1.3 IPC non typées

OSF a modifié les IPC de Mach 3.0 [Reynolds *et al.* 1993] :

- remplacement des messages auto-descriptifs (cf. 2.1.1) par desmessages contenant des données non typées ;
- nouvelles sémantiques pour l'envoi de données OOL ;
- ajout d'un jeton de sécurité ;
- ajout d'un *trailer* extensible dans chaque message ;

La structure d'un message non typé est décrite par la figure 4.

Ils sont composés :

- d'une entête ;
- de descripteurs optionnels gérés par le noyau ;
- du corps du message ;
- d'un trailer ;

Figure 4: Message non typé

La structure des messages non typés est quasiment identique à celles des messages typés, ce qui permet de réutiliser une très grande partie du code existant.

Les différences sont les suivantes :

- le numéro de séquence figure désormais dans le *trailer* ;
- les droits sur les ports et les données OOL figurent dans les descripteurs ;

Le noyau ne cherche pas à interpréter la structure des données : les données ne sont donc pas encodées lors de la transmission du message . C'est MIG qui se charge de l'encodage, selon le protocole NDR (Network Data Representation), déjà utilisé par DCE (Distributed Computing Environment). Ceci permet d'éviter le surcoût introduit par l'encodage et le décodage systématique des données, qui est inutile lorsque les machines sont homogènes ;

Les modifications des sémantiques des données OOL ont pour objectif d'améliorer les performances des serveurs, afin d'améliorer les capacités temps réel de Mach, et d'accroître la sécurité.

- une option permettant de copier physiquement les données OOL lors de l'envoi d'un message a été ajoutée. Elle résoud les problèmes de partage des données OOL entre la tâche et le noyau. De plus, la transmission de données OOL de petite taille entre deux noeuds devient plus rapide ;
- les IPC non typées améliorent également les performances des listes *gather/scatter* de données OOL. Il est désormais possible de définir une liste de mémoires tampons *déjà allouées* dans lesquelles seront copiées les données OOL lors de leur réception ;
- seules les données OOL sont transmises : il n'y a plus d'alignement sur les pages, ce qui posait un sérieux problème de sécurité. En effet, dans le cas d'une donnée OOL de quelques octets, toute la page était transmise. Un client pouvait ainsi avoir accès à des données privées du serveur ;

2.1.4 NORMA

NORMA, développé par OSF [Bryant *et al.* 1993, Foundation 1994] est une extension de Mach qui permet à des tâches distantes de communiquer en utilisant les sémantiques des IPC standards de Mach. Ainsi, les communications entre tâches distantes sont totalement transparentes.

Lorsqu'une tâche envoie un droit d'émission sur un des ses ports à une tâche distante, ce port est pris en charge par NORMA : il devient ainsi un port NORMA. Chaque port NORMA possède un identificateur unique, appelé uid. Chaque noeud gère une table de correspondance, contenant les ports NORMA dont il connait l'existence. Un port NORMA figure dans la table de tout noeud sur lequel s'exécute une tâche possédant un droit d'émission sur ce port. Sur ces noeuds, le port est appelé port mandataire. Sur le noeud d'origine du port, celui-ci est appelé port principal.

Structure de NORMA : NORMA est composé de plusieurs modules :

- module de sortie : il convertit les messages au format NORMA, puis les passe au module de transport ;
- module de contrôle de flux : il gère le protocole de comuunication, qui fonctionne selon le modèle *stop-and-wait* ;
- module de transport : il constitue l'interface entre NORMA et le pilote de périphériques;
- module d'entrée : il réassemble les fragments et les place dans la file d'attente du port de destination ;

Les interactions entre ces modules sont décrites sur la figure 5.

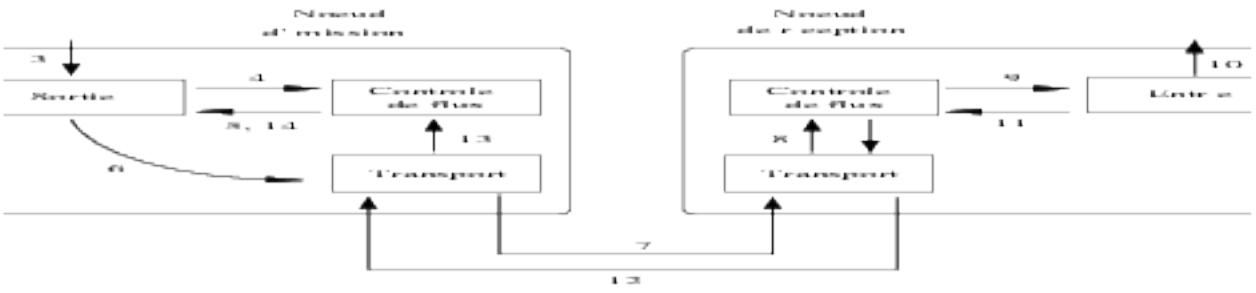

Figure 5: Interactions entre les modules de NORMA

Nous pouvons maintenant détailler les traitements effectués par NORMA. Supposons qu'une tâche A souhaite envoyer un message à une tâche B distante :

1. A compose le message et l'envoie par **mach_msg**() ;
2. cet appel provoque une trappe dans le noyau, qui copie le message de l'espace d'adressage de A vers celui du noyau ;
3. le noyau détermine que le message est destiné à une tâche distante et appelle NORMA, invoquant ainsi le module de sortie ;
4. le module de sortie convertit le message au format NORMA :

- les noms de ports contenus dans l'entête et le corps du message en uid NORMA ;
- si le kmsg ne contient que des données inline et si sa taille est inférieure ou égale à une page, le module de sortie le transmet au module de contrôle de flux ;
- sinon, le module de sortie crée un ou plusieurs *page-list copy object* .

5. lorsque le module de contrôle de flux décide qu'il est temps d'envoyer le premier paquet, il le transmet au module de sortie ;
6. le module de sortie lui ajoute une en-tête, puis le passe au module de transport ;
7. le module de transport transmet le message, en utilisant les **page-list copy object** et un mécanisme de continuation. Si sa taille dépasse la taille maximale d'une unité de transmission, celui-ci est fragmenté ;
8. sur le noeud distant, le gestionnaire d'interruptions du pilote de périphérique réseau invoque le module de transport, qui copie les fragments du kmsg dans une mémoire tampon avant de les transmettre au module de contrôle de flux ;
9. le module de contrôle de flux décide d'accepter les fragments et les transmet au module d'entrée ;
10. le module d'entrée réassemble les fragments, place le message dans la file d'attente de la tâche destinataire puis acquitte la réception ;
11. le message d'acquittement est transmis au module de transport ;
12. le module de transport transmet le message d'acquittement sur le réseau ;
13. sur le noeud de départ, le module de transport transmet le message d'acquittement au module de contrôle de flux ;
14. le module de contrôle de flux décide que le paquet a bien été reçu par son destinataire, et que le paquet suivant peut être envoyé.
15. Lorsque B est prête à recevoir le message, elle appelle **mach_msg()**. NORMA convertit le message au format standard, puis le recopie de l'espace d'adressage du noyau vers celui de B.

2.1.5 Mach Packet Filter

[Yuhara *et al.* 1994] décrit un nouveau mécanisme de filtrage des paquets. Les filtres de CMU/Stanford [Mogul *et al.* 1987] et de Berkeley [McCanne et Jacobson 1993] souffrent de deux défauts majeurs :

1. le temps de traitement des paquets est proportionnel au nombre d'entités communicantes , car chacune possède son propre filtre. Ceci est illusté par la figure 6 ;
2. ils ne gèrent pas les messages fragmentés, qui doivent donc être traités par une couche supérieure. En effet, seul le premier fragment contient l'adresse du destinataire. De plus, les fragments peuvent arriver dans le désordre, voire ne pas arriver du tout.

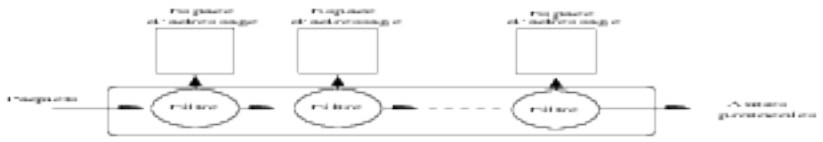

Figure 6: Ancien filtre de paquets

Pour résoudre le premier problème, le nouveau filtre tente d'envoyer tous les paquets d'un protocole déterminé en une seule étape : il n'y a donc plus qu'un seul filtre par protocole. Ceci est illustré par la figure 7.

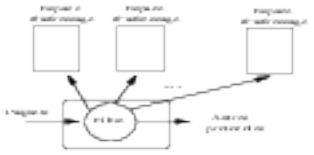

Figure 7: Nouveau filtre de paquets

Pour résoudre le second, il utilise une mémoire pour chaque filtre qui lui permet de faire le lien entre les informartions qui ne figurent que dans le premier fragment (destinataire) et les informations communes à tous les fragments (identificateur du message). Ainsi, le filtre peut suspendre le traitement des fragments du message tant que le premier fragment n'est pas arrivé.

Ce nouveau filtre est particulièrement intéressant lorsqu'il est combiné avec l'implémentation

réseau décrite dans [Maeda et Bershad 1993]. La structure du système qui en résulte est décrite par la figure 8.

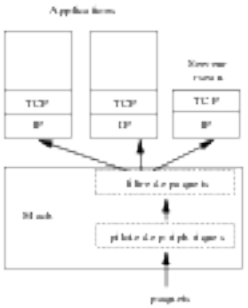

Figure 8: Combinaison du Mach Packet Filter et des librairies de protocoles

Les performances du nouveau filtre sont intéressantes, puisqu'il est 7,8 fois plus rapide que celui de CMU/Stanford et 4,3 fois plus rapide que celui de Berkeley.

2.2 Communications en mode utilisateur

2.2.1 Le netmsg server

Le netmsg server [Group 1989], développé par CMU, constitue la première tentative d'extension des IPC locales de Mach 3.0 à l'échelle d'un réseau local.

Il est composé d'une tâche Mach multithreadée, qui s'exécute en mode utilisateur sur chaque noeud du réseau. Les serveurs réseau communiquent entre eux afin d'avoir chacun une vue cohérente de l'ensemble des tâches qui s'exécutent sur tous les noeuds.

2.2.2 Structure du serveur

Les principaux services assurés sont :

- conversion des ports, grâce à une base de données contenant les informations suivantes :
  - un port local, représentant la tâche locale ;
  - un port réseau, repéré par un identificateur unique, auquel sont associées certaines informations permettant de préserver la sécurité des communications ;
- gestion des ports : vérification de la validité des informations contenues dans la base de données et mise à jour si nécessaire ;
- gestion des protocoles de transport : segmentation et réassemblage des messages, contrôle de flux, gestion des erreurs de transmission ;
- gestion des messages : certaines informations contenues dans les messages Mach n'ont pas de sens à l'échelle du réseau. C'est le cas des données out-of-line ou des droits sur les ports. Il est donc impératif de les convertir une première fois avant de transmettre le message sur le réseau, puis à nouveau avant de livrer le message à son destinataire. Une conversion est également nécessaire lorsque l'émetteur et le destinataire n'utilisent pas la même représentation interne des données (little endian ou big endian).
- nommage ;
- cryptage des messages ;

La figure 9 illustre la communication entre deux tâches Mach.

Figure 9: Communication grâce au netmsg server

Cette approche souffre d'un certain nombre de problèmes qui dégradent les performances de manière importante. En effet, l'envoi et le réception d'un message nécessitent de nombreux changements de contexte, ainsi que de multiples copies des données.

Ces surcoûts sont particulièrement pénalisants lorsque les messages sont de petite taille.

2.2.3 Mémoire partagée

[Reynolds et Heller 1991] propose une solution à ces problèmes de performances. La figure 10 illustre ce mécanisme.

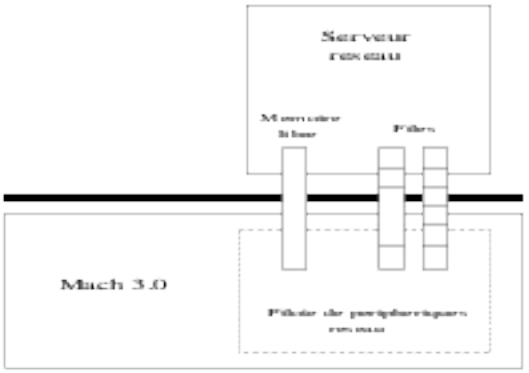

Figure 10: Mémoire partagée entre le serveur réseau et le pilote de périphériques

Il repose sur deux extensions du noyau :

- la possibilité de partager de la mémoire entre un pilote de périphériques du noyau et une tâche en mode utilisateur ;
- la possibilité pour un gestionnaire d'interruptions du noyau de débloquer un thread en mode utilisateur ;

Le serveur réseau et le pilote de périphériques réseau partagent en lecture-écriture une région mémoire. Elle est composée de files de messages et de mémoires tampons.

Lors de l'arrivée d'un paquet, le gestionnaire d'interruptions de l'interface réseau le place dans la file appropriée et débloque un thread du serveur réseau qui traite le paquet.

Le partage des files entre le gestionnaire d'interruption et le serveur posent des problèmes complexes :

- synchronisation des accès ;
- allocation et désallocation des tampons ;

De plus, une seule tâche utilisateur peut bénéficier de ce mécanisme.

Cette implémentation augmente les performances de manière non négligeable, mais elles restent très inférieures à celles d'un système monolithique.

2.2.4 Mapping du driver dans l'espace d'adressage du serveur

[Forin *et al.* 1991] présente une autre solution pour améliorer les performances du netmsg server. Il s'agit ici de mapper le pilote de périphériques dans l'espace d'adressage du serveur, où figurent déjà les protocoles réseaux, ce qui permet d'éviter la recopie des messages entre l'espace d'adressage du micronoyau et celui du serveur, ainsi que de nombreux changements de contexte.

En terme de performances, cette optimisation permet de doubler le taux de transfert du serveur réseau, ce qui reste toutefois encore loin des performances d'un système monolithique.

La figure 11 illustre ce mécanisme.

Mach 3.0 contient les appels systèmes permettant à une tâche utilisateur de mapper un pilote de périphérique dans son propre espace d'adressage, puis de communiquer directement avec ce périphérique :

- **device_map()** ;
- **device_open()** ;
- **device_close()** ;
- **device_get_status()** ;
- **device_set_status()** ;
- **device_read()** ;
- **device_write()** ;

2.2.5 Adaptation du code réseau au micronoyau Mach

Les mesures de performances montrent systématiquement que les systèmes basés sur un micronoyau sont plus lents que les systèmes monolithiques. On pourrait donc en conclure que les micronoyaux sont ne sont pas adaptés aux communications réseau. Sans doute s'agirait-il là d'une conclusion un peu hâtive.

En effet, [Maeda et Bershad 1992] montre qu'il est tout à fait possible d'obtenir de bonnes performances, à condition de tenir compte des spécificités du micronoyau. Cet article montre en substance que les performances réseau des micronoyaux sont mauvaises car le code réseau qu'ils

Figure 11: Driver mappé dans l'espace d'adressage du serveur

utilisent est inadapté : il provient le plus souvent d'un système monolithique. C'est le cas d'Unix Server [Golub *et al.* 1990], qui utilise le code réseau de 4.3BSD.

En modifiant les interactions entre le code réseau de Unix Server et le micronoyau, les auteurs ont nettement amélioré ses performances. Leurs optimisations sont les suivantes :

- pas d'utilisation des données out-of-line, qui obligent le noyau à mapper ces données dans son espace d'adressage. Lorsque les messages sont petits, il est moins coûteux de les lui envoyer directement ;
- envoyer les messages directement au pilote de périphériques, plutôt que de les envoyer au micronoyau ;

Unix Server ainsi modifié gagne en rapidité, mais ses performances restent très inférieures à celles de Mach 2.5. Elles sont par contre comparables à celles du serveur utilisant la mémoire partagée, décrit dans [Forin *et al.* 1991].

D'autres optimisations s'imposent donc pour espérer combler le décalage :

- réecriture des clients, qui appellent le serveur en utilisant des appels systèmes Mach, et non pas des appels Unix, afin d' éviter le passage dans l'émulateur ;
- déverrouillage des C-Threads du serveur, afin de réduire le nombre de changements de contexte .

Les auteurs ont développé un serveur UDP afin de tester l'efficacité de ces optimisations. Leurs

mesures montrent que ces modifications permettent d'obtenir des performances supérieures à celles de Mach 2.5.

Le tableau suivant regroupe les performances des différentes implémentations de UDP décrites jusqu'ici. Les tests ont été effectués sur deux DECstation 2100 reliées par un réseau Ethernet. Le temps indiqué est la durée d'un aller-retour d'un message UDP.

Implémentation de UDP	Temps (ms)
Mach 2.5	4,8
Unix Server (IPC, VM)	19,5
Unix Server (IPC, pas de VM)	14,3
Unix Server (ni IPC, ni VM)	13,6
Unix Server (driver mappé)	13,1
Serveur UDP	4,2

La conclusion de cette expérience est que les communications réseau peuvent avoir lieu dans l'espace d'adressage du serveur sans dégrader les performances, à condition d'éviter au maximum les interactions avec le noyau.

2.2.6 Librairie de protocoles

[Maeda et Bershad 1993] prolonge l'approche précédente en plaçant le maximum de code réseau dans l'espace d'adressage des applications. Le code réseau est donc scindé en trois parties :

- une partie située dans le serveur, chargée des opérations qui nécessitent l'accès à des structures de données globales, et qui influent peu sur les performances des communications (établissement et terminaison d'une communication, routage, ...) ;
- une librairie multithreadée, mappée dans l'espace d'adressage de chaque application, qui implémente les sémantiques de la couche socket de 4.3BSD ;
- une couche d'interface au-dessus de l'adaptateur réseau, chargée d'envoyer et de recevoir les paquets.

La structure de ce système est décrite sur la figure 12.

Plusieurs versions de la librairie ont été développées :

- la première version utilise le Mach Packet Filter [Yuhara *et al.* 1994] et les IPC pour communiquer avec le noyau. Les paquets sont envoyés un par un.

Figure 12: Librarie réseau mappée

- la seconde utilise le Mach Packet Filter (MPF) et une mémoire partagée entre entre le noyau et l'application. Ceci n'améliore pas le temps de latence de la transmission : lors de l'arrivée d'un paquet, celui-ci est d'abord copié dans un tampon du noyau avant d'être lu par le MPF.

- la troisième utilise un MPF mieux intégré au noyau : seule l'entête du paquet est transmise du noyau vers le MPF. Une fois que celui-ci a déterminé la destination du paquet, il le copie directement du noyau vers l'espace d'adressage du destinataire.

Afin d'améliorer encore les performances de la librairie, le fonctionnement de la couche socket a été légèrement modifié : l'application et la librairie partagent un tampon afin d'éviter une copie des données lors de l'envoi ou de la réception d'un paquet.

Le tableau suivant indique les performances des différentes versions. Les mesures ont été effectuées entre deux DECstation 5000/200 reliées par un réseau Ethernet. Le débit et la latence de TCP ont été mesurés pour des paquets de 1 Ko. Les performances de Mach 2.5 et de Unix Server sont indiquées à titre de comparaison.

	Débit (Ko/s)	Latence (ms)
Mach 2.5	1070	4,56
Unix Server	740	7,82
Librairie MPF+IPC	910	5,09
Librairie MPF+SHM	1076	5,32
Librairie IMPF+SHM	1088	5,09
Librairie NEWAPI+MPF+IPC	959	4,96
Librairie NEWAPI+MPF+SHM	1083	4,94
Librairie NEWAPI+IMPF+SHM	1099	4,80

En conclusion, il est tout à fait possible d'obtenir des performances équivalentes, voire même supérieures, à celles d'un système monolithique, à condition :

- de bien décomposer les services et d'en placer le plus possible dans l'espace d'adressage des applications ;
- de tirer profit des spécificités du micronoyau pour optimiser les performances (pilote de périphériques mappés, mémoire partagée, ...) ;
- d'améliorer le fonctionnement interne du protocole, tout en préservant ses sémantiques de départ ;
- d'éviter si possible le passage dans un émulateur Unix en réecrivant les clients pour qu'ils utilisent directement les appels Mach. Ceci est tout à fait envisageable pour les applications courantes comme **ftp** ou **telnet**.

2.3 Co-location

[Condict *et al.* 1993]

[Condict *et al.* 1994]

3 Autres systèmes

3.1 V kernel

VMTP : [Cheriton 1986]

3.2 Amoeba

FLIP : [Kaashoek *et al.* 1993b]

Communication de groupe : [Kaashoek et Tanenbaum 1992, Kaashoek *et al.* 1993a]

Comparaison entre communications en mode noyau et en mode utilisateur: [Oey *et al.* 1995]

3.3 Sprite

3.4 Firefly

[Schroeder et Burrows 1990]

3.5 Chorus

3.6 x-kernel

Présentation de x-kernel : [Hutchinson et Peterson 1991]

Implémentation de l'Université d'Arizona : [Orman *et al.* 1993]

Evaluation de x-kernel par OSF : [Travostino et Reynolds 1993]

Implémentation d'OSF : [Sakaruba et Travostino 1994]

Références

[Barrera 1991] J. Barrera. A Fast Mach Network IPC Implementation. In *Proceedings of the Usenix Mach Symposium*. Usenix Association, November 1991.

[Bershad 1992] B. Bershad. The Increasing Irrelevance of IPC Performance for Microkernel-Based Operating Systems. In *Proceedings of the 1992 Usenix Workshop on Microkernels*, 1992.

[Bryant *et al.* 1993] B. Bryant, A. Langerman, S. Sears, et D. Black. *NORMA IPC: A Task-to-Task Communication System for Multicomputer Systems*. Technical report, Open Software Foundation, October 1993.

[Cheriton 1986] D. Cheriton. VMTP : A Transport Protocol for the Next Generation of Communication Systems. *Computer Communications Review*, 16(3):406–414, 1986.

[Condict *et al.* 1993] M. Condict, D. Mitchell, et F. Reynolds. *Optimizing Performance of Mach-based Systems by Server Co-Location: A Detailed Design*. Technical report, Open Software Foundation, August 1993.

[Condict *et al.* 1994] M. Condict, D. Bolinger, D. Mitchell, et E. McManus. *Microkernel Modularity with Integrated Kernel Performance*. Technical report, Open Software Foundation, June 1994.

[Draves 1990] R. Draves. A Revised IPC Interface. In *Proceedings of the Usenix Mach Workshop*, pages 101–121. Usenix Association, October 1990.

[Forin *et al.* 1991] A. Forin, D. Golub, et B. Bershad. An I/O System for Mach 3.0. In *Proceedings of the 1st Usenix Mach Symposium*, pages 163–176. Usenix Association, November 1991.

[Foundation 1994] Open Software Foundation. *NORMA IPC Version Two: Architecture and Design*. Technical report, Open Software Foundation, October 1994.

[Golub *et al.* 1990] D. Golub, R. Dean, A. Forin, et R. Rashid. Unix as an Application Program. In *Proceedings of the Summer 1990 USENIX Conference*, pages 87–96, June 1990.

[Group 1989] Mach Networking Group. *Network Server Design*. Technical report, Open Software Foundation, August 1989.

[Hsieh *et al.* 1993] W. Hsieh, F. Kaashoek, et W. Weihl. The Persistent Relevance of IPC Performance : New Techniques for Reducing the IPC Penalty. In *Proceedings of the 4th Workshop on Workstation Operating Systems*, pages 186–190, October 1993.

[Hutchinson et Peterson 1991] N. Hutchinson et L. Peterson. The x-kernel : an Architecture for Implementing Network Protocols. *IEEE Transactions on Software Engineering*, 17(1), January 1991.

[Kaashoek *et al.* 1993a] F. Kaashoek, A. Tanenbaum, et K. Verstoep. Using Group Communication to Implement a Fault-Tolerant Directory Service. In *Proceedings of the 13th International Conference on Distributed Computing Systems*, pages 130–139, 1993.

[Kaashoek *et al.* 1993b] F. Kaashoek, R. van Renesse, H. van Staveren, et A. Tanenbaum. FLIP : an Internetwork Protocol for Supporting Distributed Systems. *ACM Transactions on Computer Systems*, pages 73–106, February 1993.

[Kaashoek et Tanenbaum 1992] F. Kaashoek et A. Tanenbaum. *Efficient Reliable Group Communication For Distributed Systems*. Technical Report IR-295, Vrije Universiteit, Amsterdam, July 1992.

[Maeda et Bershad 1992] C. Maeda et B. Bershad. Networking Performance for Microkernels. In *Proceedings of the Third Workshop on Workstation Operating Systems*, April 1992.

[Maeda et Bershad 1993] C. Maeda et B. Bershad. Protocol Service Decomposition for High Performance Networking. In *Proceedings of the Fourteenth ACM Symposium on Operating Systems Principles*, pages 244–255, December 1993.

[McCanne et Jacobson 1993] S. McCanne et V. Jacobson. The BSD Packet Filter : A New Architecture for User-level Packet Capture. In *Proceedings of the Winter 1993 Usenix Conference*, pages 259–269. Usenix Association, January 1993.

[Mogul *et al.* 1987] J. Mogul, R. Rashid, et M. Accetta. The Packet Filter : An Efficient Mechanism for User-level Network Code. In *Proceedings of the 11th ACM Symposium on Operating Systems Principles*, pages 39–51, 1987.

[Oey *et al.* 1995] M. Oey, K. Langendoen, et H. Bal. Comparing Kernel-Space and User-Space Communication Protocols on Amoeba. In *Proceedings of the 15th International Conference on Distributed Computing Systems*, May 1995.

[Orman *et al.* 1993] H. Orman, E. Menze, S. O'Malley, et L. Peterson. A Fast and General Iimplementation on Mach IPC in a Network. In *Proceedings of the 3rd Usenix Mach Conference*, pages 75–88. Usenix Association, April 1993.

[Reynolds *et al.* 1993] F. Reynolds, F. Travostino, R. MacDonald, D. Ellistion, et K. Loepere. *Design for a New Untyped IPC for Mach.* Technical report, Open Software Foundation, September 1993.

[Reynolds et Heller 1991] F. Reynolds et J. Heller. Kernel Support For Network Protocol Servers. In *Proceedings of the Usenix Mach Symposium.* Usenix Association, November 1991.

[Sakaruba et Travostino 1994] T. Sakaruba et F. Travostino. *Using a Networked Mach IPC implemented in user-space with x-kernel.* Technical report, Open Software Foundation, April 1994.

[Schroeder et Burrows 1990] M. Schroeder et M. Burrows. Performance of Firefly RPC. *ACM Transactions on Computer Systems*, 8(1):1–17, February 1990. dispo bib.

[Travostino et Reynolds 1993] F. Travostino et F. Reynolds. *x-kernel Evaluation at the OSF RI.* Technical report, Open Software Foundation, September 1993.

[Yuhara *et al.* 1994] M. Yuhara, B. Bershad ans C. Maeda, J. Eliot, et B. Moss. Efficient Packet Demultiplexing fot Multiple Endpoints and Large Messages. In *Proceedings of the Winter Usenix Conference.* Usenix Association, January 1994.