[Birrell 82], cf chapitre I. Ce service était utilisé pour désigner symboliquement, de façon unique et globale, des portes, des groupes de portes, des services, des sites, des usagers, des groupes d'usagers, ..., ainsi que leurs attributs. Il répondait donc aux Objectifs 1 et 2 (cf §1) en ce qui concernait ces entités.

La vision externe offerte par le système était configurée comme l'illustre la figure 5.1.

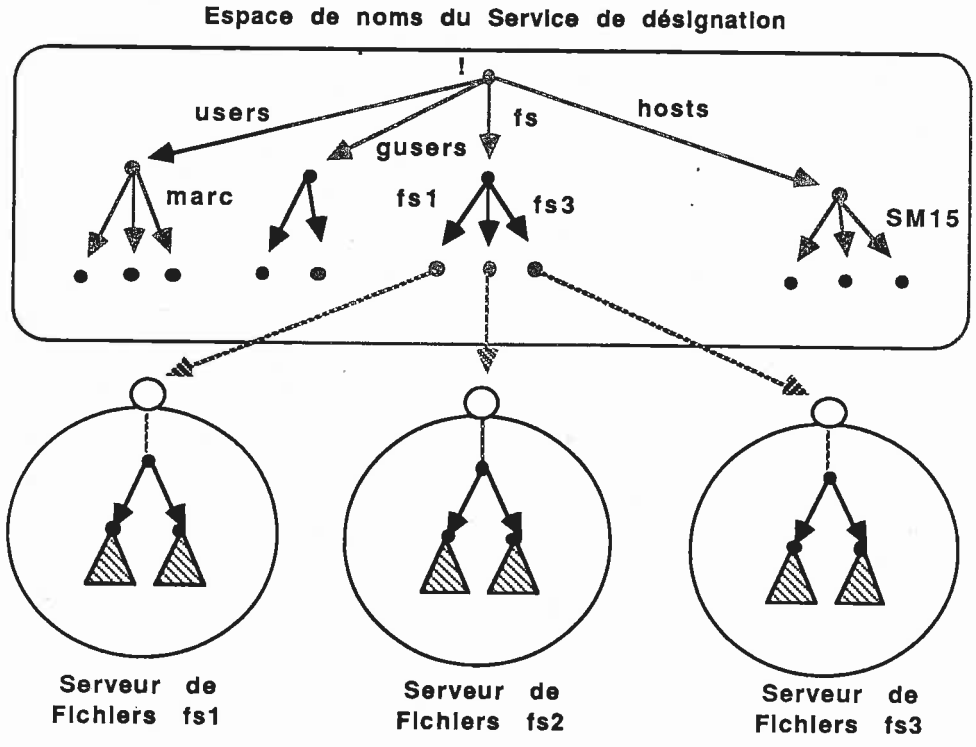

fig. 5.1 - Vision externe du système offerte à l'aide du service de désignation

Dans l'exemple, les catalogues *!users* et *!gusers* regroupent les noms globaux et uniques des usagers et des groupes d'usagers; le catalogue *!hosts* regroupe les noms des sites; le catalogue *!fs* (pour *file-servers*) regroupe les noms des serveurs de fichiers du système, etc. Des catalogues étaient prévus pour que les usagers puissent aussi enregistrer les noms et les attributs nécessaires au développement de leurs applications réparties (Objectif 2).

Associés aux noms que le service de désignation enregistrait, on trouvait des attributs qui permettaient l'accès aux ressources, en général, et aux serveurs, en particulier. Par exemple, le nom global d'un fichier [Rozz 85] était obtenu en concaténant le nom du serveur qui le gérait, avec son nom dans l'arbre du serveur. Exemple:

!fs!fs3/users/ahmad/agp/main.p

Lorsqu'un nom de fichier commencé par "!" (syntaxe d'excéption) était passé en paramètre d'une procédure d'interface, la bibliothèque d'interface demandait au service de désignation le nom de la porte du serveur de fichiers (*!fs!fs3*); en possession du nom de cette porte, elle pouvait adresser une requête au serveur de fichiers; cette requête au serveur portait sur le reste du nom (*/users/ahmad/agp.p*).

Le SDS Chorus avait, avec cette solution, plusieurs composants. D'une part, un service de désignation logiquement centralisé, et d'autre part, les SDS des serveurs matérialisant des ressources et gérant aussi leurs noms (les serveurs de fichiers par exemple). Du point de vue de la désignation globale et unique des ressources gérées par les deuxièmes, la vision externe offerte n'était pas celle qui doit caractériser un système intégré ou logiquement unique. En particulier, des noms symboliques globaux devraient être disponibles.

En effet, les conventions de désignation des fichiers obtenues étaient, pour l'essentiel, équivalentes à l'introduction d'une "racine réseau", désignée par le biais d'exceptions syntaxiques, avec une suggestion logique de la localisation d'un fichier (le nom du serveur qui le gérait) incluse dans son nom global (cf §6 chapitre I).

De plus, cette solution conduisait à l'introduction d'un service supplémentaire qui était spécifié de telle sorte qu'il introduisait également des conventions, des règles, des procédures d'interface et des commandes interactives spécifiques et différentes de celles des fichiers. En particulier, il exigeait la réadaptation de toutes les procédures administratives du système Unix et la réadaptation des utilitaires de courrier, parmi beaucoup d'autres.

En résumé, des points de vue "système offrant une vision unique" (Objectif 1 §1) et "compatibilité avec Unix" (Contrainte §1) cette solution n'était pas satisfaisante en ce qui concernait les fichiers. Du point de vue de la désignation des usagers et des groupes d'usagers, elle permettait une vision unique mais également incompatible avec celle d'Unix (Contrainte §1).

L'introduction d'un service spécifique de désignation n'étant pas capable de résoudre tous les problèmes soulevés, nous avons alors analysé la possibilité de trouver une solution selon laquelle la désignation symbolique de Chorus reposerait complètement sur les serveurs de fichiers

En effet, dans le cadre de notre scénario (Objectif 1 §1), l'accès aux fichiers est omniprésent. En plus, étant données les options prises pour la protection (cf §3 chapitre III), l'utilisation de fichiers spéciaux pour enregistrer les usagers est suffisante et convenable (Contrainte §1).

C'est cette nouvelle direction de travail que nous avons explorée.

3. La désignation symbolique dans Chorus

Chorus prend l'approche de ne pas offrir aucun service spécifiquement adressé à la désignation symbolique. Toutes les fonctionnalités concernant la gestion et l'interprétation de noms symboliques reposent sur les serveurs de fichiers. Dans le système existent plusieurs serveurs de fichiers. Chaque serveur gère un arbre de désignation de fichiers (compatible Unix). Parmi les fichiers de son arbre, un serveur gère aussi des fichiers spéciaux (*/etc/passwd* et */etc/group*, cf annexe II) qui servent à désigner les usagers et les groupes d'usagers. Par conséquence, l'espace des noms symboliques du système est formé de plusieurs arbres.

Ces options sont simples mais laissent toujours deux problèmes sans solution:

- L'absence de fonctionnalités permettant la désignation des sites, des serveurs, des services et des opérations (Objectif 2 §1);
- Offrir une vision externe caractéristique d'un SER intégré et logiquement unique. En particulier, des noms symboliques globaux doivent être disponibles.

Pour résoudre le premier de ces problèmes, un nouveau type de noeud a été introduit dans l'arbre de désignation des fichiers: les *noeuds du type porte/groupe*, déjà introduits au chapitre IV. Ces noeuds de l'arbre de désignation dénotent un descripteur capable d'enrégistrer un UID (*Unique identifier*) d'une porte ou d'un groupe. Des opérations spécifiques, permettent la mise à jour et l'obtention de la valeur de l'UID enregistrée dans le descripteur (sous un nom interne contextuel et protégé).

Il s'agit d'une fonctionnalité typique d'un serveur de noms. En effet, les serveurs de fichiers ne gèrent pas la porte (ou le groupe) qui est associée au nom, ils ne gèrent qu'un ensemble d'attributs (UID, attributs de protection, ...) enregistrés dans un descripteur. Ils permettent également la mise à jour et l'obtention de ces attributs. Cependant, la cohérence entre les attributs de l'entité et les attributs associés à son nom est de la responsabilité des clients. Ces noeuds, comme nous le verrons par la suite, permettent la désignation de sites, de services et de serveurs.

Pour résoudre le deuxième problème, i. e., offrir une vision externe unique du système, la duplication des catalogues et des fichiers est la solution qui nous semble la plus intéressante. En effet, la duplication de ces entités rend indifférent le serveur utilisé par les clients. Par conséquence, la duplication rend unique la vision externe offerte par le système. De plus elle permet de rendre le système de désignation fiable.

Cependant, les serveurs de fichiers de Chorus ne sont pas capables de dupliquer automatiquement ni les catalogues, ni les fichiers. Par conséquence, la vision externe qu'ils offerent aux usagers n'est pas celle d'un système unique mais plutôt celle d'un système où existent plusieurs arbres différents et indépendants.

La solution que nous proposons dans la suite de cet ouvrage est une solution intermédiaire. Elle est bâtie sur l'hypothèse que, à terme, la duplication automatique sera disponible. Pour le moment, Chorus n'offre que des moyens simples de réaliser la duplication à la demande, dont la mise en oeuvre est, pour l'essentiel, sous la responsabilité des usagers.

Le système de désignation de Chorus se présente comme une collection d'arbres Unix étendus et interconnectés. Outre que l'accès aux fichiers (indépendamment de leurs localisations), Chorus offre des fonctionnalités supplémentaires qui semblent bien adaptées vers l'évolution dans la direction d'un vrai espace unique de désignation.

Ces fonctionnalités supplémentaires sont les suivantes:

- les *noeuds du type porte/groupe*, permettant la désignation symbolique de sites, de services, d'opérations, de serveurs et de groupes de ces entités;
- une méthode permettant d'interconnecter entre eux les différents arbres, i. e., permettant d'établir une liaison d'un noeud d'un arbre géré par un serveur vers la racine d'un arbre géré par un autre;
- des noms symboliques globaux permettant de désigner, de manière indépendante de la localisation, n'importe quel noeud, de n'importe quel arbre;

- des liens symboliques permettant un usage plus souple du système de désignation; permettant, par exemple, qu'un identificateur d'un catalogue apartenant à un usager, dénote, en fait, le nom global d'une entité publique.

Ces fonctionnalités sont mises en oeuvre par les moyens suivants:

- par l'introduction de deux nouveaux types de noeuds dans l'arbre de désignation standard d'Unix (*type porte/groupe* et *type lien symbolique*);
- par l'introduction d'un protocole permettant l'interprétation répartie des noms symboliques;
- par la duplication automatique de quelques catalogues de référence;

Les prochains paragraphes décrivent dans les détails ces fonctionnalités, comment elles sont mises en oeuvre et comment elles peuvent être utilisées par les clients du système. Néanmoins, pour leur compréhension, il nous faut d'abord voir comment se fait, dans Chorus, l'accès aux fichiers.

4. Accès aux fichiers dans Chorus

L'accès aux fichiers dans Chorus doit être considéré selon deux niveaux d'observation du système:

1/ le client du système invoque les opérations de manipulation des fichiers en appelant des procédures d'interface (open, read, write, ...);

2/ les procédures d'interface transforment leurs appels en protocoles du type demande/réponse avec les serveurs de fichiers. Naturellement, elles font aussi les mises à jour du contexte de l'appelant en conformité avec l'opération demandée et la réponse reçue.

Comme pour tous les autres services offerts par Chorus, le client utilise une interface procédurale implantée sur l'échange de messages avec des serveurs. La bibliothèque d'interface est responsable par la traduction des appels d'un niveau dans l'autre.

Selon le modèle client/serveur, pour chaque ressource qui est en train d'être manipulée par un acteur (le client), il y une liaison entre cet acteur et le serveur qui gère la ressource. Les ressources qui un acteur peut être en train de manipuler sont, soit les catalogues (la racine et le catalogue courant, cf annexe II), soit les fichiers qu'il a ouverts. La liaison entre le client et le serveur est mise en oeuvre par deux portes. La *porte des appels système* de l'acteur, côté client, et une porte du serveur donnant accès à la ressource, côté serveur.

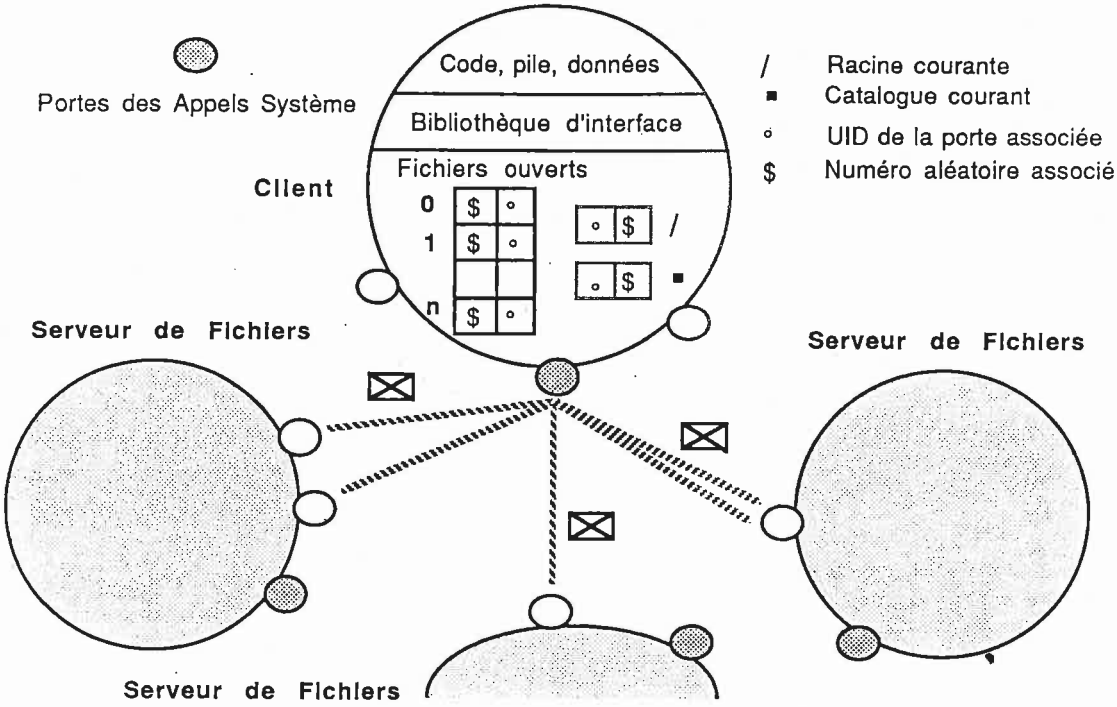

fig. 5.2 - Liaisons entre un client et les serveurs de fichiers

Cette interface entre le client et les serveurs, mise en oeuvre par l'usage des portes Chorus, permet l'accès aux fichiers indépendamment des localisations du client et des serveurs. Si un client migre, en exécutant, par exemple, un **exec** à distance, il peut conserver l'accès aux fichiers qu'il était en train de manipuler. Un acteur peut avoir sa racine courante, catalogue courant et fichiers gérés par des serveurs différents. En plus, ses fils, même créés à distance, peuvent également partager avec lui l'accès à ces ressources.

Par des raisons de performance et de facilité d'implantation, le client voit chacune des ressources qu'il est en train de manipuler comme étant un points d'accès. Un point d'accès est une paire:

(Porte du serveur, Numéro)

Ces paires, créées et détruites dynamiquement, sont attachées chacune à une ressource et sont uniques et globales parce qu'elles contiennent une porte et le serveur s'éfforce de réutiliser le plus tard possible *Numéro*. Cependant, elles ne désignent pas de façon univoque une ressource, elles ne représentent qu'un point d'attache à la ressource pour un client donné: au même instant, plusieurs points d'accès différents peuvent représenter le même fichier pour des acteurs différents et, à des moments différents, le même fichier peut être accédé, par le même acteur, par le biais de points d'accès différents. Le but principal de l'introduction des points d'accès a été celui de permettre qu'une même porte d'un serveur puisse être partagée parmi plusieurs de ses clients. A la limite, une seule porte d'un serveur peut supporter toutes les liaisons client/ressource qui

l'ont comme cible.

Quand un acteur naît, il possède au moins quelques points d'accès initiaux: la racine courante, le catalogue courant, l'entrée et la sortie standard, etc. A partir des points d'accès attachés à des catalogues, l'acteur peut acquérir d'autres points d'accès car, logiquement, un serveur de fichiers offre deux types de points d'accès:

- Point d'accès à un fichier: acceptant les requêtes de lecture/écriture sur le fichier.
- Point d'accès à un catalogue: acceptant les requêtes qui exigent l'interprétation d'un nom symbolique (l'ouverture d'un fichier par exemple); le chemin d'accès est interprété à partir de ce catalogue.

Les opérations offertes par le serveur de fichiers peuvent, du point de vue de la manipulation des points d'accès, être regroupées en trois classes.

- Classe Getacces. Regroupant les opérations qui retournent un (des) nouveau(x) point(s) d'accès dans un message. Par exemple, dans le message de réponse à l'ouverture d'un fichier, se trouve le point d'accès qui permettra de le manipuler par la suite.
- Classe Shareaccess. Quand un acteur naît, il hérite de son père un ensemble de points d'accès représentant ses fichiers ouverts, catalogue et racine courante; à l'initialisation du fils, il faut avertir les serveurs de fichiers que ces points d'accès sont partagés par un acteur de plus, afin que si le père les libère, ils ne soient pas détruits.
- Classe Releaseaccess. Quand un acteur ferme un fichier, il doit libérer le point d'accès qui le représente; le changement de catalogue courant est transformé en un *Getaccess* du nouveau catalogue, suivi d'un *Releaseaccess* sur l'ancien catalogue courant; à la mort d'un acteur, un *Releaseaccess* est adressé à tous les points d'accès qu'il possédait.

L'interface que nous venons de présenter est liée aux conventions de désignation suivantes:

Au niveau de l'interface procédurale du client, les noms symboliques des ressources ont la même syntaxe que ceux d'Unix; un nom de la forme:

/catalogue1/catalogue2/.../fichier

est interprété par rapport à la racine courante de l'acteur; un nom de la forme:

catalogue1/.../fichier

est interprété par rapport à son catalogue courant.

Au niveau de l'interface "messages échangés avec les serveurs", un nom symbolique de ressource est de la forme:

(Porte du serveur, Numéro, Chemin d'accès)

où la porte du serveur est implicitement indiquée en lui envoyant le message de requête. Avec cette interface, tous les noms symboliques sont relatifs à un point d'accès, i. e., rélatifs à une porte d'un serveur de fichiers.

Nous avons montré comment dans Chorus les acteurs dialoguent avec les serveurs de fichiers. Nous allons maintenant décrire les différentes extensions introduites dans l'arbre de désignation standard d'Unix que ces serveurs gèrent.

5. Fonctionnalités supplémentaires des serveurs de fichiers

Avec les hypothèses de travail présentées au §3, nous avons décidé d'introduire quelques fonctionnalités supplémentaires dans les serveurs de fichiers. Ces extensions leur permettent de rendre également les services caractéristiques d'un serveur de noms, permettent d'interconnecter les arbres de noms gérés par les différents serveurs et permettent, finalement, de disposer de noms globaux pour les ressources, comme décrit au §6. Ces fonctionnalités supplémentaires sont décrites ci-dessous.

5.1. Noeuds du type porte/groupe

L'Objectif 2, cf §1, exige que l'on puisse désigner symboliquement des services. Chorus offre les moyens de le réaliser par le biais de noeuds spéciaux de l'arbre des fichiers, les *noeuds du type porte/groupe*, déjà introduits au chapitre précédent.

Un noeud de ce type permet l'association d'une porte (ou d'un groupe de portes) avec un nom symbolique. L'opération:

mkport (path-name, pgd)

crée un *noeud du type porte/groupe* sous le nom *path-name* et lui associe l'UID de l'entité (porte ou groupe de portes) désignée par le nom interne contextuel *pgd*. Un acteur peut, par le biais de l'opération **symbbind** (cf chapitre IV), redéfinir la porte ou le groupe associé au noeud, ou acquérir un nom interne contextuel désignant la porte ou le groupe qui lui ont été préalablement associés. Ainsi, l'acquisition, par un client, du nom de la porte d'un serveur, ou d'un groupe de serveurs, est possible.

Par exemple, un usager développe une application répartie, l'application "jeu", constituée par trois serveurs: "gaston", "spirou" et "marsupilami". Chacun de ces serveurs a besoin de connaître les portes de ses partenaires et d'inscrire une de ses portes dans un groupe, le groupe de portes "jeu/joueurs".

A l'initialisation, chacun des serveurs crée et ouvre deux portes, avec les noms internes contextuels *p1* et *p2*; inscrit *p1* sous son nom symbolique et insère *p2* dans le groupe de l'application. Exemple de l'initialisation de "gaston":

```
p1 := creatport;
p2 := creatport;
diagnostic := openport ( p1, 10 );
diagnostic := openport ( p2, 0 );
diagnostic := mkport ( "jeu/gaston", p1 );
pgd_groupe := symbbind ( GETNAME, "jeu/joueurs" );
diagnostic := groupctl ( INSERTPORT, p2, pgd_groupe );
```

5.2. Désignation de sites à l'aide de noeuds du type porte/groupe

Les *noeuds du type porte/groupe* sont également utilisés pour désigner symboliquement les sites. Dans Chorus, la désignation symbolique des sites sert, pour l'essentiel, à désigner le nom du site de chargement d'un acteur. Quand il s'agit d'une création à distance, le client doit spécifier le nom symbolique du site de création. Ce nom dénote un *noeud du type porte/groupe* qui désigne symboliquement une porte du serveur d'acteurs du site de chargement.

Le catalogue "/hosts" regroupe les noms symboliques des SM90 du réseau; chacun

de ces noms est un catalogue qui regroupe, à son tour, le nom symbolique des portes des serveurs d'acteurs de chaque site de la machine. Par exemple, le nom "/hosts/sm3/mt1" est le nom d'un *noeud du type porte/groupe* qui enregistre le nom de la porte du serveur d'acteurs du site du Module de Traitement 1 de la SM90 numéro 3 du système.

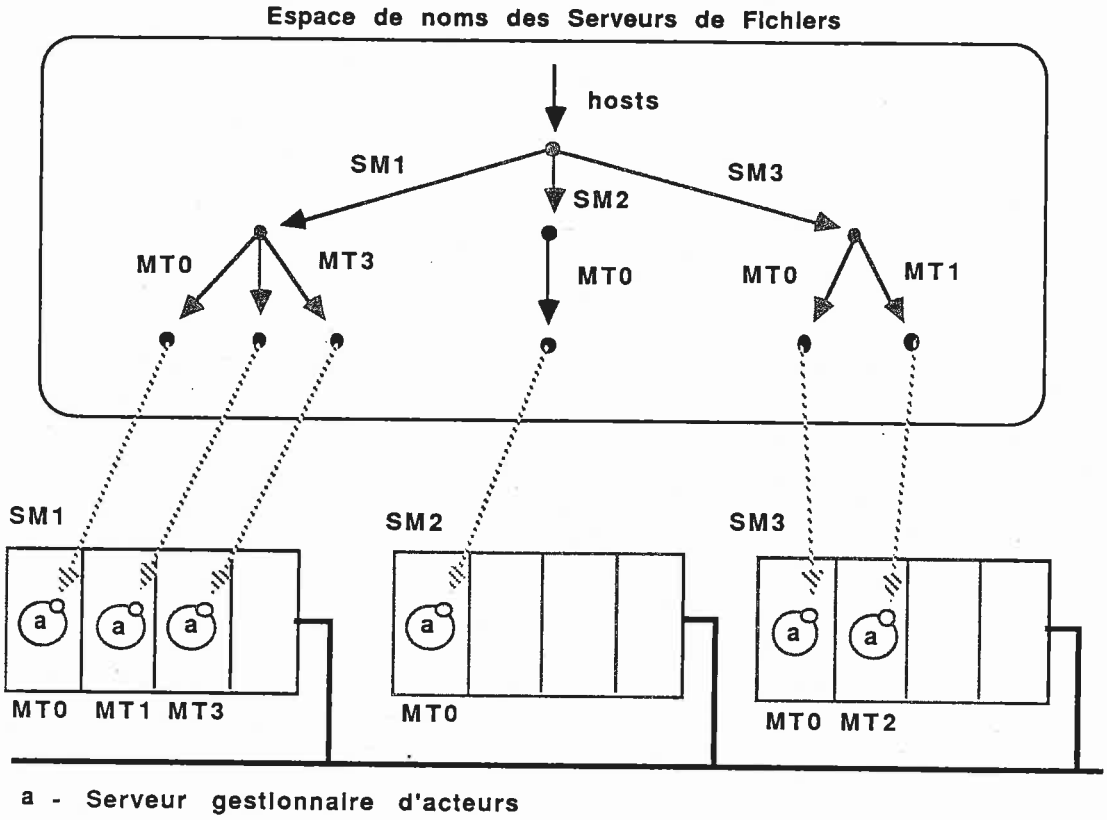

fig. 5.3 - Désignation des sites dans Chorus

Quand le serveur d'acteurs du site du créateur reçoit sa requête, il reçoit aussi le nom du site de chargement et le point d'accès à partir duquel ce nom doit être interprété. Il peut ainsi obtenir le nom de son partenaire distant, car le nom symbolique qui lui est passé en paramètre désigne une de ses portes, et lui redirige la requête du client.

Comme nous verrons par la suite, la redirection de requêtes par le biais de noms symboliques est un mécanisme très intéressant de Chorus. En réalité, le serveur d'acteurs n'acquiert pas le nom de la porte de son partenaire, il demande au serveur de fichiers de rediriger la requête vers son partenaire, en utilisant le nom symbolique. Il s'agit d'un protocole d'adressage symbolique qui a été introduit dans les serveurs de fichiers de Chorus. Ce protocole est la deuxième fonctionnalité supplémentaire qui a été introduite dans les serveurs de fichiers. Il va permettre qu'un *noeud du type porte/groupe* d'un arbre d'un serveur, puisse être interprété comme une "sorte de lien" vers un noeud d'un autre arbre. Cela peut être assimilé à une sorte de "montage distant", réalisé par le

biais d'une interprétation répartie de noms symboliques.

5.3. Interprétation répartie de noms symboliques

Toutes les requêtes adressées à un serveur de fichiers, concernant l'interprétation d'un nom symbolique, ont l'entête standard suivante:

Code d'opération	Diagnostic	Numéro	Chemin d'accès	Paramètres

fig. 5.4 - Entête standard des requêtes qui exigent l'interprétation d'un nom

Du point de vue de l'interface en terme de messages échangés, un nom Chorus est entièrement spécifié par le couple (Point d'accès, Chemin d'accès); le nom concerné par la requête de la figure 5.4 est le suivant:

((Porte réceptrice, Numéro), Chemin d'accès)

Avec cette information, le serveur de fichiers commence l'interprétation du paramètre *Chemin d'accès* à partir du catalogue associé à *Point d'accès,* tout en ignorant *Code d'opération.*

Si pendant l'interprétation il rencontre un *noeud du type porte/groupe*, il remplace le paramètre *Chemin d'accès* par sa partie non encore analysée, il met 0 à la place du paramètre *Numéro* (en cas de redirection, le numéro unique associé au point d'accès n'est pas significatif) et redirige le message vers la porte associée au noeud, en utilisant l'appel **Putfwd** (cf chapitre IV). Si le noeud désigne un groupe de portes, le message est redirigé sur le groupe avec le mode d'adressage *fonctionnel* (cf chapitre IV).

Il s'agit d'un protocole de redirection symbolique de requêtes similaire à celui qui a été introduit dans le V-system [Cheriton 84a] (cf §7.2 chapitre I). Il présente exactement les mêmes avantages que celui-là. L'interprétation d'un nom peut commencer dans un serveur et continuer dans un autre. Les serveurs matérialisant la désignation symbolique sont capables d'acheminer symboliquement des requêtes concernant des opérations qu'ils ne connaissent pas et dont la syntaxe peut comporter des parties terminales qu'ils ne sauraient pas interpréter.

C'est ce protocole que les seveurs d'acteurs utilisent pour rediriger symboliquement les requêtes de création d'acteurs à distance. Les noms concernés par cette redirection sont les *noeuds du type porte/groupe* qui désignent symboliquement leurs portes. Ces noeuds, comme nous l'avons vu ci-dessus, désignent les sites Chorus.

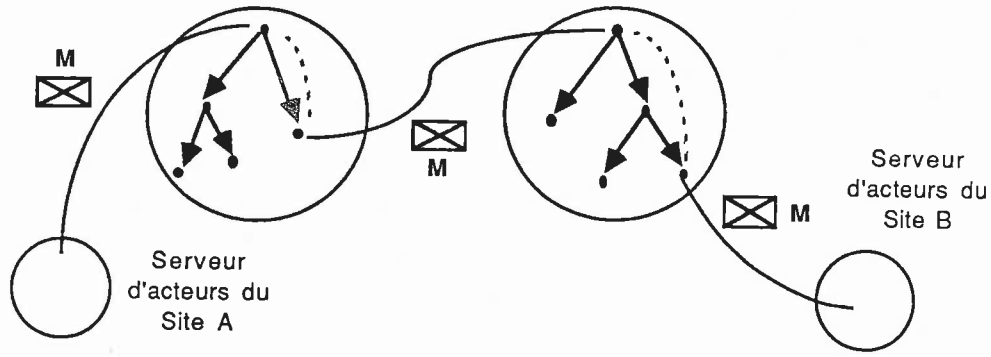

fig. 5.5 - Redirection symbolique de requêtes par l'intermédiaire des serveurs de fichiers

Ce protocole est aussi à la base des conventions de désignation de Chorus dans la mesure où il permet l'interconnexion des différents arbres des différents serveurs. Les différents serveurs concernés par la désignation symbolique l'utilisent pour coopérer. Cet aspect sera repris aux paragraphes suivants.

5.4. Liens symboliques

Une troisième fonctionnalité a été introduite dans les serveurs de fichiers de Chorus. Il s'agit de la notion de *noeud du type lien symbolique*. Ces noeuds vont permettre de contourner les limitations des liens d'Unix (qui ne peuvent pas traverser les frontières d'un volume physique de fichiers). Ils sont aussi à la base des fonctionnalités qui permettent aux usagers de percevoir le système comme un système presque unique. Fondamentalement, il s'agit d'une fonctionnalité qui permet une plus grande souplesse d'utilisation du système de désignation symbolique.

Un *noeud du type lien symbolique* est un noeud de l'arbre de désignation qui enregistre un *chemin d'accès*.

Quand, pendant l'interprétation d'un nom N, un serveur de fichiers traverse un *lien symbolique* dénotant le chemin d'accès L, il concatène L avec la partie non encore analysée du chemin de N, disons N', et recommence l'analyse du chemin N'' = L.N'†. L'analyse de N'' commence au catalogue racine de l'arbre du serveur, si L commence par "/", ou au catalogue qui contient L dans le cas contraire.

Il s'agit d'un mécanisme bien connu qui est présent dans beaucoup de systèmes parmi lesquels la version d'Unix réalisée par l'Université de Berkeley aux Etats Unis.

† Où c1.c2 dénote la concaténation des chaînes c1 et c2.

5.5. Détection de boucles dans l'interprétation d'un nom

L'interprétation de noms traversant des *liens symboliques* peut donner lieu à des boucles sans fin. Ce problème n'existe pas dans l'arbre conventionnel d'Unix.

Dans Chorus, chaque message concernant un nom symbolique contient une liste de points d'accès (UID d'une porte plus un numéro unique par rapport à cette porte). Cette liste enregistre les points d'accès aux catalogues ou où il y a eu une modification du contenu de son chemin d'accès, dû à la traversée d'un *lien symbolique*. Cette liste permet la détection de boucles dans l'interprétation d'un nom.

5.6. Resumé

Par rapport à la sémantique conventionnelle qui caractérise une collection d'arbres de désignation, compatibles avec ceux du système Unix, les serveurs de fichiers de Chorus offrent trois fonctionnalités supplémentaires:

- Les *noeuds du type porte/groupe* permettant l'association d'une porte (ou d'un groupe de portes) avec un nom symbolique. Ces noeuds servent à la désignation de serveurs, de services, etc. Ils sont également utilisés pour la désignation des sites.
- La redirection symbolique de requêtes par le biais des *noeuds du type porte/groupe*. Cette fonctionnalité permet l'interprétation de noms, répartie par différents serveurs ainsi que l'interconnexion des arbres des différents serveurs.
- Les *noeuds du type lien symbolique*. Ces noeuds augmentent la souplesse d'utilisation du SDS (système de désignation symbolique) de Chorus.

Ces fonctionnalités nouvelles, comme nous le verrons au prochain paragraphe, permettent l'interconnexion des différents arbres de désignation des différents serveurs, de telle sorte que les usagers disposent également de noms globaux, i. e., indépendants de l'arbre (et donc du serveur) qui les interprète.

6. Conventions de désignation symbolique

Comme nous avons introduit au §3, Chorus n'offre qu'une solution intermédiaire vis-à-vis de l'exigence "système à vision unique". Cette solution est bâtie sur l'hypothèse que, à terme, la duplication automatique sera disponible. Nous montrons dans ce paragraphe comment se matérialise cette solution.

Dans Chorus, tous les noms symboliques sont rélatifs au serveur qui commence leur interprétation. Même les noms qui commencent par "/" sont aussi contextuels à un serveur de fichiers, i. e., ils ne sont pas globaux.

Appelons nom "s-global" (global à un serveur) un nom qui commence par le caractère "/" et dont l'interprétation commence dans la racine de l'arbre de désignation d'un serveur F.

Deux noms s-globaux identiques ne désignent pas nécessairement la même entité. Ils ne le font que s'ils sont interprétés par le même serveur, ou que si les entités ainsi désignées, dans deux serveurs différents, sont identiques du point de vue de l'usager, c.-à.-d., si elles sont dupliquées. En particulier, le nom s-global d'une entité automatiquement dupliquée dans tous les serveurs est un nom global et unique de cette entité.

Par exemple, le nom s-global "/users/jose/exemple.p", interprété par deux serveurs différents, dénote deux entités différentes. Du point de vue de l'usager, cela peut lui être indifférent si les deux entités sont identiques. En l'absence de mécanismes de duplication automatique, ce problème est, néanmoins, laissé au client.

Ainsi, un nom s-global ne dénote pas une entité mais une classe d'entités dont, éventuellement, plusieurs instances différentes existent, chacune dans un serveur différent. L'équivalence des différentes instances est un problème que Chorus délègue, pour le moment, vers le client. Sa discussion sera reprise aux paragraphes suivants.

Cependant, pour ce faire, le client doit pouvoir distinguer une instance parmi l'ensemble. Pour résoudre ce problème Chorus introduit le mécanisme suivant: chaque serveur de fichiers contient un catalogue de nom "/fs" (abréviation de "file-servers") qui regroupe les identificateurs de tous les serveurs de fichiers du système. Ce catalogue est rigoureusement identique dans les différents serveurs (cf §8). Les noeuds présents dans ce catalogue sont des *noeuds du type porte/groupe.* Chacun d'eux dénote intrinsèquement la porte associée à la racine de l'arbre gérée par le serveur qu'il désigne.

Ainsi, tout nom s-global commencé par "/fs" est un nom global au système. En effet, l'interprétation de ce nom commence dans la racine du serveur dans lequel l'appelant a sa racine, mais, sa requête sera aussitôt redirigée par le protocole standard de redirection de requêtes (cf §5.3) vers le serveur désigné par la partie initiale du chemin d'accès restant. Eventuellement, l'interprétation peut se poursuivre dans le même serveur mais, quel que soit le serveur où commence la traduction, elle aboutira toujours sur le même descripteur d'entité.

Par exemple, dans un système avec trois serveurs de fichiers, le fichier "/etc/passwd" est présent dans tous les serveurs du système. Les catalogues "/fs" de tous les serveurs sont identiques et contiennent les noms des trois serveurs de fichiers: "/fs/fs1", "/fs/fs2" et "/fs/fs3". Le nom s-global "/etc/passwd" désigne la classe "fichier de définition des usagers du système". Cette classe regroupe trois instances du fichier; ces instances sont désignées respectivement par "/fs/fs1/etc/passwd", "/fs/fs2/etc/passwd" et "/fs/fs3/etc/passwd". En particulier, le nom "/fs/fs1/etc/passwd", par exemple, désigne toujours le même fichier quel que soit le serveur qui commence son interprétation.

Cette méthode revient donc à l'introduction de noms globaux dans le système.

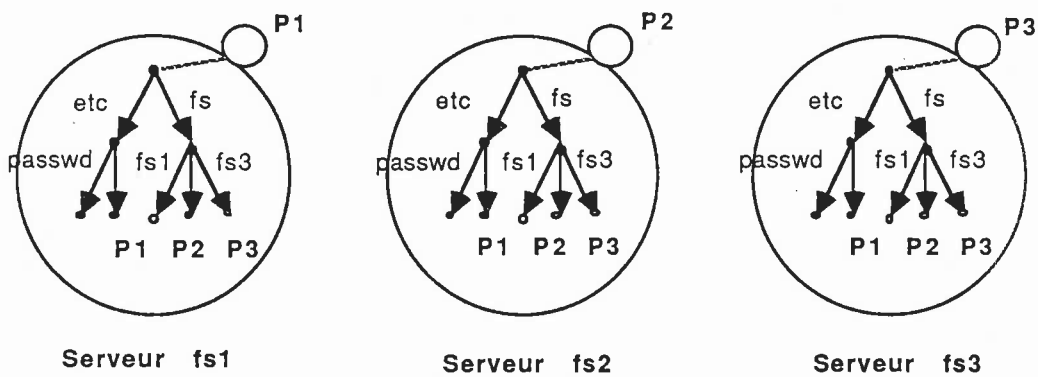

fig. 5.6 - Configuration de la foret de désignation Chorus

Les conventions de désignation introduites peuvent être ainsi caractérisées:

- Tout nom commencé par "/fs" est un nom global.
- Tout nom commencé par "/" mais sans le préfixe "/fs" (nom s-global) est contextuel à un serveur.
- Tout nom s-global d'une entité dupliquée est un nom global et unique de cette entité. En particulier, tous les noms de la forme "/fs/serveur" sont globaux et uniques.

Il y a deux interprétations duales de ces conventions:

1/ les différents arbres de désignation sont interconnectés par une sorte de "racine réseau" (cf §6.1 chapitre I), la racine "/fs" (introduite sans exceptions syntaxiques);

2/ le catalogue "/fs" de chaque arbre regroupe les catalogues sur lesquels sont "montés" les racines des autres arbres (cf §6.2 chapitre I); cependant, dans Chorus, les montages se font automatiquement et reflètent le caractère intégré de son environnement.

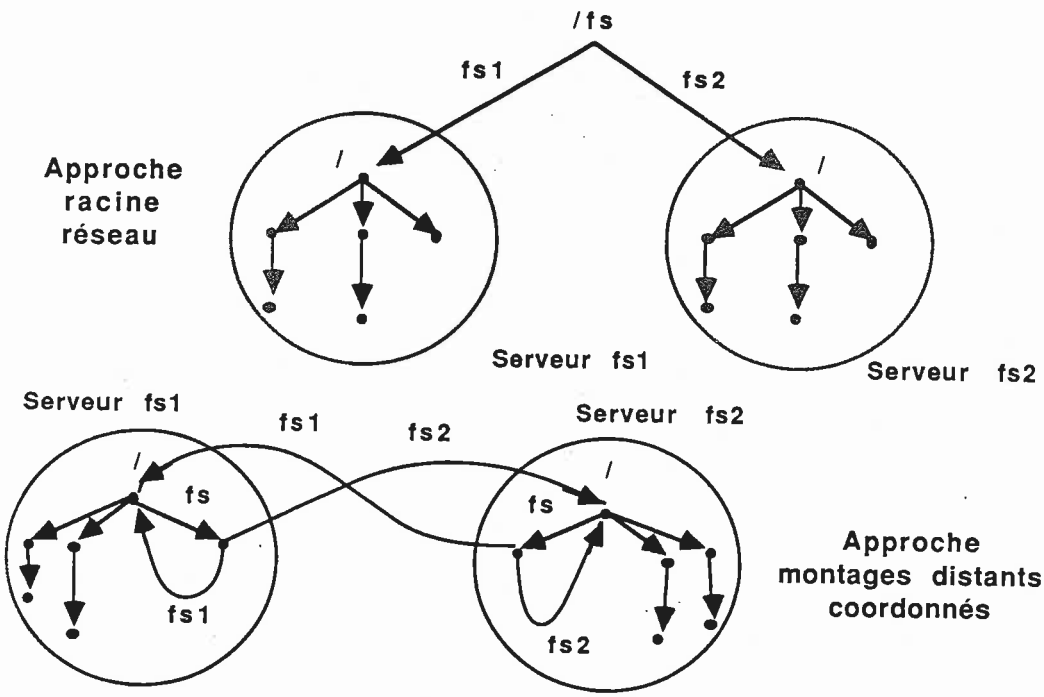

fig. 5.7 - Dualité des deux interprétations du SDS Chorus

Remarque

A première vue, cette configuration de l'arbre pourrait être optimisée en transformant l'entrée "/fs/serveur-x" du "serveur-x", en un lien direct vers la racine de ce serveur. Cependant, une telle méthode exigerait la modification de l'arbre géré par le serveur au cas où il serait transporté vers un autre serveur, avec un nom différent. L'utilisation de portes dans tous les cas, évite cette modification manuelle. La mise à jour se fera dynamiquement. **Fin de remarque.**

Un autre catalogue est aussi rigoureusement identique dans tous les serveurs, c'est

le catalogue "/hosts". Les catalogues "/fs" et "/hosts" sont disjoints parce qu'il peut y avoir un nombre différent de serveurs de fichiers et de sites; en particulier, il peut y avoir des machines sans serveur de fichiers. Les noms de la forme "/fs/..." et "/hosts/..." sont donc des noms globaux et uniques dans le système.

Il y a aussi une autre situation dans laquelle l'usager du système a besoin de savoir quelle instance d'une classe il manipule. Il s'agit de connaître explicitement les noms globaux de son catalogue courant et de sa racine courante. La commande standard Unix qui affiche le nom du catalogue courant, *pwd,* n'est capable, dans Chorus, que d'afficher le nom s-global de ce catalogue, mais pas son nom global. Par exemple, en invoquant cette commande, un usager obtient le nom "/etc". De quelle instance de ce catalogue s'agit-il?

Chaque serveur de fichiers contient un fichier standard où est enregistré son nom global et unique par rapport au catalogue "/fs", par exemple, le nom "/fs/fs1". Ce nom est le nom global et unique de la racine du serveur, sa concaténation avec le résultat de *pwd* constitue le nom global du catalogue courant, par exemple "/fs/fs1/etc". En utilisant ces fonctionnalités de base, il est possible de construire des commandes qui affichent les noms globaux du catalogue et de la racine courante quel que soient les serveurs qui les gèrent.

Revenons maintenant au problème de l'équivalence de toutes les instances d'entités appartenant à la même classe, i. e., désignées par le même nom s-global. Comme nous l'avons vu, les noms s-globaux non commencés par "/fs" ou par "/hosts" ne sont pas globaux. Cependant, ce sont ces noms que les usagers en général, et les utilitaires d'Unix en particulier, sont bien préparés pour utiliser.

7. Utilisation pratique du SDS Chorus

Nous n'avons pas encore une expérience d'utilisation pratique des conventions qui ont été introduites ci-dessus, mais nous pensons qu'elles permettent une utilisation du système qui ne met pas en cause son aspect "environnement intégré", même en absence de mécanismes de duplication automatique.

Par la suite nous présentons quelques scénarios qui l'illustrent.

Indépendance du site de login

Le sous-système de *login* d'Unix initialise une session pour le compte de l'usager en chargeant un interpréteur de commandes dont le catalogue courant est le catalogue initial de l'usager. Le nom de ce catalogue initial est un attribut de l'usager enregistré dans un fichier spécial ("/etc/passwd"). Si ce fichier contient un nom global du catalogue initial de l'usager, il se trouvera toujours avec le même catalogue initial, sur le même serveur, quel que soit son site d'attache au système.

Boîtes à lettres des usagers

Les utilitaires de courrier d'Unix présupposent que la boîte à lettres de l'usager *user* se trouve dans le fichier "/usr/spool/mail/user".

Dans le système, un serveur de fichiers contrôlant une importante mémoire de masse, le serveur "ref", par exemple, est choisi comme serveur gérant les boîtes à lettres. Dans tous les serveurs de fichiers, à l'exception de "ref" (pour éviter des boucles dans l'interprétation), les noeuds de nom "/usr/spool/mail" sont des *liens symboliques* sur le noeud de nom "/fs/ref/usr/spool/mail".

Quel que soit le site de *login* d'un usager, il peut lire ou envoyer du courrier comme si le système était unique. Naturellement, si le site du serveur "/fs/ref" est en panne, le courrier est momentanément indisponible tant que ce serveur n'est pas rechargé.

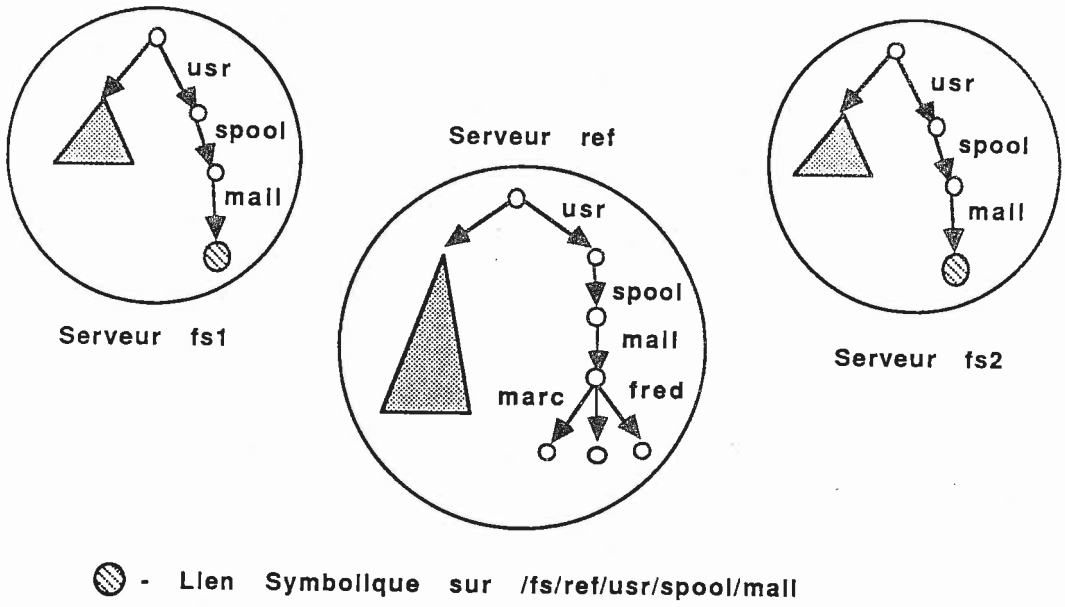

fig. 5.8 - Indépendance des boîtes à lettres vis-à-vis du site de *login*

Répartition de la charge entre différents serveurs

L'exemple précédent suggère une méthode simple d'utilisation des *liens symboliques* pour répartir, sur les différents serveurs de fichiers, la charge de gérer les différents sous-arbres du système. Ce scénario est illustré par la figure 5.9.

Comme la figure l'illustre, chaque usager possède un sous-arbre de fichiers primaire et un sous-arbre de fichiers secondaire (de *backup*). A l'aide de *liens symboliques*, les usagers ont une vision de ces sous-arbres indépendante de leur répartition par les différents serveurs et indépendante de leurs sites de *login*. Périodiquement, chaque usager peut dupliquer son sous-arbre primaire sur son sous-arbre secondaire, en utilisant les utilitaires standards du système Unix pour la copie incrémentalle de sous-arbres.

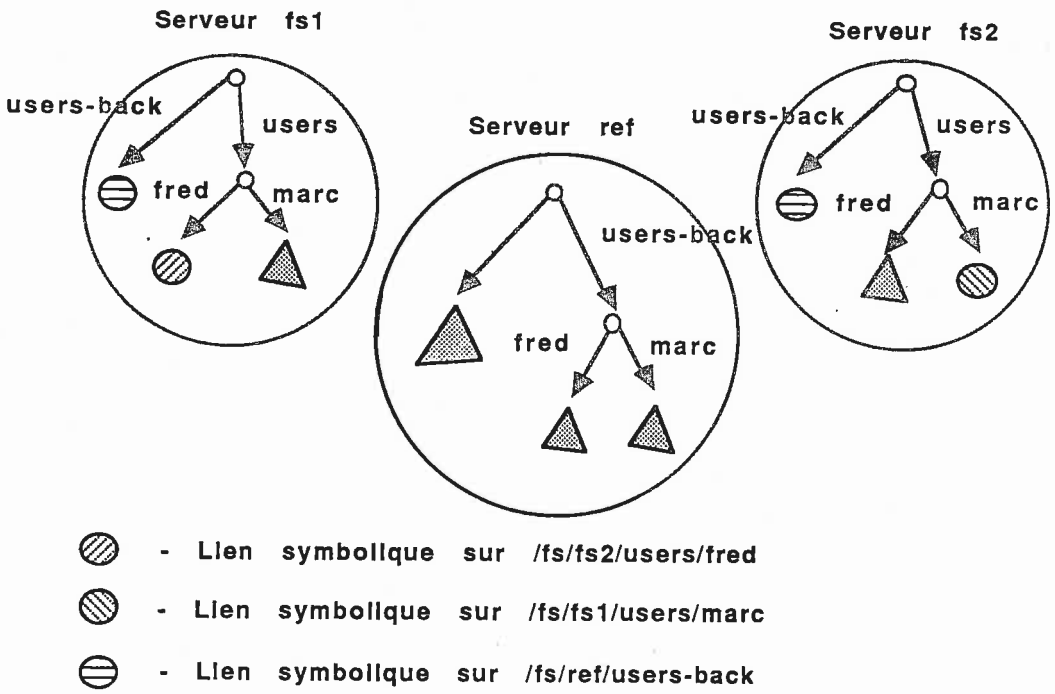

fig. 5.9 - Répartition transparente de sous-arbres par différents serveurs

Désignation des usagers et groupes d'usagers

Dans Unix, les noms des usagers, ainsi que leurs attributs, et les noms des groupes d'usagers, sont enregistrés dans des fichiers spéciaux ("/etc/passwd" et "/etc/group"). Ces fichiers sont critiques pour le système et leur absence empêche pratiquement son fonctionnement. Le schéma proposé pour les boîtes à lettres ne leur est pas applicable.

Un autre scénario peut être considéré, étant donné qu'ils sont souvent consultés, mais très rarement mis à jour. Dans tous les serveurs de fichiers il y a une copie de ces fichiers; ces copies, à l'exception de celles qui sont dans le serveur de référence, ne peuvent être que consultées, et pas modifiées; seules les copies qui sont sur le serveur de référence peuvent être modifiées.

A l'initialisation de chaque machine ayant un serveur de fichiers, une copie des versions de référence est exécutée automatiquement; naturellement, les administrateurs du système peuvent déclencher manuellement une copie des versions de référence sur les autres serveurs indépendamment de leurs sites de *login*. Dans chaque machine supportant un serveur de fichiers, une commande peut être déclenchée périodiquement de façon automatique pour exécuter une copie de la version de référence de ces fichiers.

Conclusions

Comme nous venons de montrer, le SDS Chorus offre, du point de vue des conventions de désignation, au moins les mêmes fonctionnalités que les systèmes batîs sur la notion de "racine réseau" ou de "montage distant".

Cependant, le fait que tous les noms de la forme "/fs/serveur" soient globaux, permet un confort d'utilisation supérieur à celui des systèmes orientés "montage distant", quand le but est d'offrir un environnement intégré, ce qui est notre cas.

D'autre part, il ne présente pas les exceptions syntaxiques qui caractérisent la plupart des systèmes orientés "racine réseau". En particulier, son SDS peut être étendu par intégration de fonctionnalités de "montage distant" d'autres systèmes. En plus, sa racine "/fs" peut être aussi la cible d'un montage d'un système non Chorus offrant des montages distants. Cet SDS résiste également sans problèmes à l'introduction de serveurs de fichiers qui dupliquent automatiquement les ressources.

8. Duplication des catalogues /fs et /hosts

Les conventions de désignation du SDS de Chorus sont bâties sur l'hypothèse de la duplication des catalogues "/fs" et "/host". Nous avons deux solutions pour ce problème. La première est triviale, mais pas totalement satisfaisante. La deuxième soulève quelques problèmes supplémentaires, mais peut s'avérer, potentiellement, plus intéressante.

Solution 1

La configuration d'un système avec quelques dizaines de sites et assez intégré (cf §1) est en général stable et facilement administrable. Les noeuds des sous-arbres "/fs" et "/hosts" sont souvent consultés, mais très rarement mis à jour. Ils ne sont modifiés que quand il y une reconfiguration du système.

Les serveurs d'acteurs et les serveurs de fichiers sont des serveurs système. Ils peuvent par conséquence créer et ouvrir des portes statiques (cf chapitre IV), i. e., des portes spéciales, réservées au système, dont les UID sont prédéfinis; ces UID peuvent donc se connaître à l'avance.

Nous pouvons donc inscrire dans un fichier le nom des noeuds de ces deux sous-arbres et les UID qui doivent leurs être associés. Un utilitaire spécial pourra, à partir de ce fichier, faire la mise à jour de l'arbre d'un serveur de fichiers.

La cohérence de ces sous-arbres dans les différents serveurs est donc de la même nature que la cohérence des fichiers "/etc/passwd" et "/etc/group" (voir le paragraphe précédent), la méthode utilisée pour maintenir leur cohérence peut être fondamentalement la même.

Solution 2

La solution précédente présente l'inconvénient de rendre l'administration du système complexe, ou même impossible, si le nombre de sites et serveurs est important. D'autre part, elle n'est pas du tout souple face aux pannes, ou face au besoin de reconfigurer rapidement le système (pour des raisons administratives ou de répartition de charge). En plus, elle est mise en oeuvre par l'utilisation des portes statiques, une fonctionnalité système exceptionnelle dont l'usage doit être limité.

Cette autre solution est introduite par le biais d'un service spécial, le service d'analyse des préfixes de configuration. Un serveur de préfixes de configuration (serveur de préfixes) est un serveur qui gère les sous-arbres "/fs" et "/hosts" tout en calculant dynamiquement leur contenu. Ce serveur est aussi capable d'interpréter un nom symbolique et sait participer au protocole d'acheminement de requêtes, comme les serveurs de fichiers. Dans le cadre de la solution 2, les serveurs de fichiers ne connaissent plus les catalogues "/fs" et "/hosts".

Quand un serveur de fichiers reçoit une requête concernant un nom dont l'interprétation commence à sa racine et dont il ne connaît pas son identificateur initial, il redirige la requête vers le groupe des serveurs de préfixes en mode fonctionnel.

Les serveurs de préfixes appartiennent tous à un groupe bien connu des serveurs de fichiers; leur adressage en mode fonctionnel permet une souplesse importante dans le choix de la configuration exacte de ce service. En effet, n'importe lequel des serveurs de préfixes peut traiter la requête redirigée par un serveur de fichiers. Grossièrement, tous les serveurs de préfixes connaissent la même version des catalogues "/fs" et "/hosts".

Si le serveur de préfixes qui reçoit la requête redirigée par l'un des serveurs de fichiers connaît le noeud concerné par la partie initiale du nom, il redirige la requête vers le serveur concerné. Sinon, il répond tout de suite au client initial lui signalant que ce nom n'existe pas.

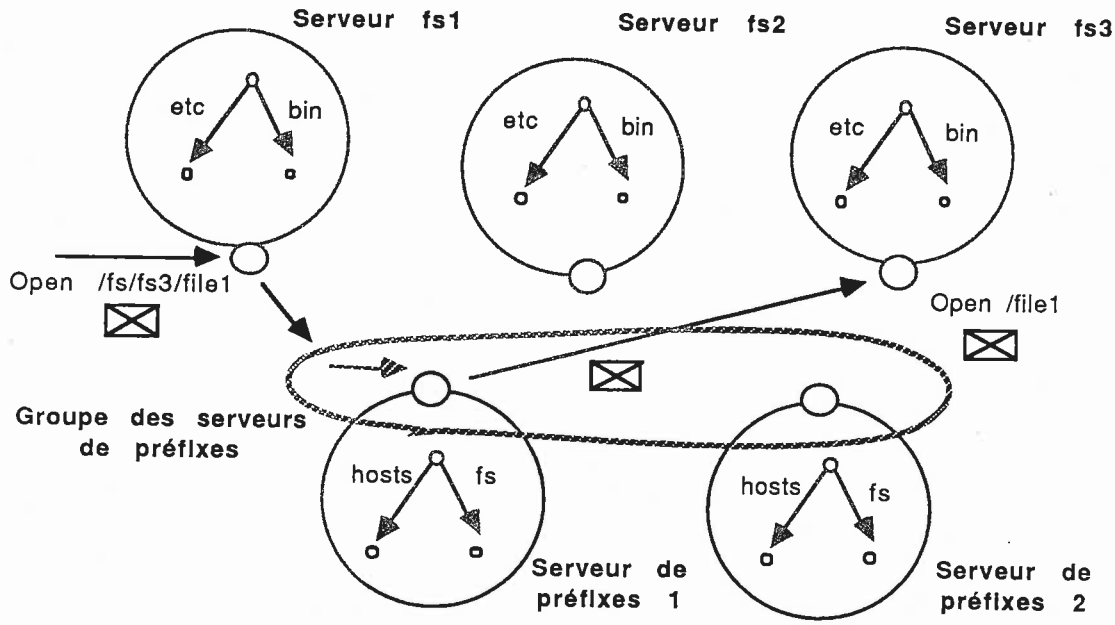

fig. 5.10 - Redirection des requêtes des clients au travers des serveurs de préfixes

Pour que les serveurs de préfixes puissent calculer dynamiquement les sous-arbres qu'ils sont censés gérer, il faut que les serveur de fichiers et les serveurs d'acteurs connaissent leurs noms et sachent répondre à une requête de *demande d'identification*. Quand un de ces serveurs reçoit une telle requête, il répond en indiquant son nom et le nom de la porte qui lui donne accès. Les serveurs de préfixes calculent dynamiquement le contenu de leurs arbres en diffusant périodiquement des demandes d'identification au groupe des serveur de fichiers et au groupe des serveurs d'acteurs.

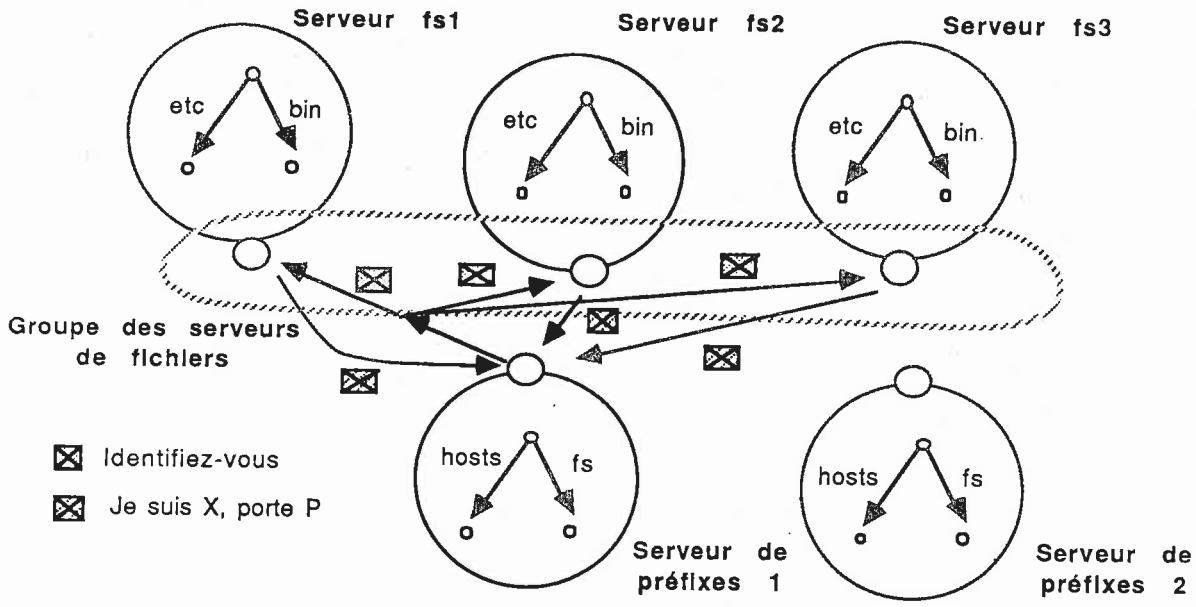

fig. 5.11 - Mise à jour dynamique du catalogue "/fs" par un serveur de préfixes

Nous pensons que cette méthode est plus souple que la précédente. Elle présente néanmoins l'inconvénient d'introduire un nouveau service de désignation et de faire un usage assez important de la diffusion (un mécanisme cher). Ce dernier défaut pourrait être tempéré en modifiant l'algorithme de calcul dynamique des sous-arbres "/fs" et "/hosts", par exemple, en ne diffusant automatiquement qu'avec une période très large, mais tout en diffusant à la demande explicite d'un usager.

Cependant, ces serveurs peuvent avoir d'autres utilisations. En particulier, ils peuvent, par exemple, construire un cache des sous-arbres connus par les différents serveurs de fichiers. Ainsi, un serveur de fichiers, ne connaîssant pas un sous-arbre A, il redirige toujours une requête le concernant vers les serveurs de préfixes. Ceux-ci peuvent quand même réussir à l'acheminer vers un serveur de fichiers qui connaît A. Il s'agirait d'un mécanisme de la même nature que celui qui est proposé dans [Welch 86].

En conclusion, les serveurs de préfixes implantent une sorte de catalogues de désignation "sans mémoire", dont le contenu est mis à jour dynamiquement, en fonction des besoins.

9. Récapitulatif

Chorus est un environnement réparti et intégré de développement et exécution d'applications réparties. Chaque machine du réseau est gérée par le même sytème d'exploitation. Cependant, par paramétrisation administrative de leurs configurations, les différentes machines qui constituent le système peuvent être adaptées à des fonctions spécifiques. Par exemple, une machine dépourvue de moyens de mémoire de masse peut utiliser des serveurs de fichiers des autres machines; une machine particulièrement

robuste et puissante peut être paramétrée en serveur de référence; les périphériques chers peuvent être partagés, etc. L'interface de cet environnement est un sur-ensemble de celle du système Unix.

Bien qu'offrant des moyens importants pour aider au développement d'applications réparties, Chorus prend l'approche de n'offrir aucun service spécialisé dans la désignation symbolique. Son sous-système de désignation symbolique (SDS) est obtenu par une combinaison judicieuse de plusieurs mécanismes. Par exemple:

- Les services, les serveurs, leurs portes d'accès et leurs groupes, sont désignés symboliquement à l'aide de noeuds nouveaux de l'arbre de désignation de fichiers. Ces noeuds, les ***noeuds du type porte/groupe***, sont capables d'enregistrer une sorte de capacité logiciel vers les portes et les groupes de portes. Ils permettent ainsi la mise en oeuvre de fonctionnalités d'édition de liens caractéristiques des serveurs de noms.
- La vision externe offerte par le système est celle d'une collection d'arbres de fichiers compatibles Unix, mais étendus. Les noms des racines de ces arbres sont regroupés dans le catalogue "/fs" et sont aussi globaux. Les entrées de ce catalogue sont des ***noeuds du type porte/groupe*** enregistrant le nom des portes des serveurs de fichiers.
- Les sites sont également désignés à l'aide des ***noeuds du type porte/groupe***. Leurs noms sont regroupés dans le catalogue "/hosts" et sont globaux, i. e., indépendants de la localisation, comme ceux du catalogue "/fs".
- Les serveurs de fichiers de Chorus ne dupliquent pas automatiquement les fichiers; cependant, le système offre les moyens qui permettent, de manière semi-automatique, d'avoir une vision externe assez indépendante du site de *login*. Parmi ces moyens, les ***liens symboliques*** jouent un rôle très important.
- Les usagers et les groupes d'usagers sont désignés par des identificateurs symboliques et caractérisés par des attributs, tous les deux enregistrés dans des fichiers spéciaux (/etc/passwd et /etc/group) semi-automatiquement dupliqués.
- Dans Chorus, tous les noms commençant par "/" (i. e., globaux à un serveur) d'entités dupliquées sur les différents serveurs, sont des noms globaux de ces entités, dans la mesure où le serveur utilisé est indifférent.

La mise en oeuvre de cette solution intermédiaire, dans la voie vers un vrai système unique et intégré, est réalisée par le biais des mécanismes suivants:

- Par une méthode d'accès aux serveurs de fichiers basée sur l'usage des portes. Cette méthode permet l'accès aux fichiers de n'importe quel serveur, par n'importe quel client, indépendamment de la localisation du client et du serveur.
- Par l'introduction de deux nouveaux types de noeuds dans l'arbre de désignation standard d'Unix (***type porte/groupe*** et ***type lien symbolique***);
- Par l'introduction d'un protocole spécial, le protocole de redirection symbolique de requêtes sur les ***noeuds du type porte/groupe***, mis en oeuvre par les serveurs de fichiers, en particulier, et par tous les serveurs participant à la désignation symbolique, en général. Ce protocole permet l'interprétation répartie de noms symboliques ainsi que l'interconnexion des arbres des différents serveurs.
- Par la duplication automatique des catalogues spéciaux "/fs" et "/hosts". Cette duplication garantit que tous les noms commencés par "/fs" ou "/hosts" sont globaux. Ces catalogues, regroupant des ***noeuds du type porte/groupe***, sont dynamiquement mis à jour de telle sorte qu'ils reflètent à tout moment la configuration exacte du système. Ils constituent une sorte de catalogues "sans mémoire" dont le contenu est calculé dynamiquement.

10. Travaux similaires et conclusions

Quelques-uns des mécanismes de désignation symbolique, intégrés dans Chorus, sont aussi utilisés par d'autres systèmes.

Le système Spice [Thompson 85] par exemple, implanté sur le noyau Accent, offre aussi la notion de noeud de désignation symbolique, permettant de désigner symboliquement, d'enregistrer et d'obtenir une capacité logiciel vers une porte.

Le système V-System [Cheriton 84a] par exemple, utilise un protocole d'interprétation répartie de noms symboliques, bâti sur la redirection de requêtes entre serveurs.

Les conventions du SDS Chorus ont deux interprétations duales:

1/ Le catalogue "/fs" représente une sorte de "racine réseau", comme celles des systèmes Newcastle Connection [Brownbridge 82], Apollo/DOMAIN [Leach 83], ou Ibis [Tichy 84] par exemple, mais elle est introduite sans exceptions syntaxiques.

2/ Chorus s'apparente aux systèmes bâtis sur la notion de "montage distant" tels que NFS [Sandberg 85] ou Unix V8 [Weinberger 84], mais les montages des arbres de fichiers se font de façon automatique, cachée aux clients et reflétant l'état du système ainsi que son caractère intégré.

Voilà quelques conclusions qui ont été mises en évidence par nos travaux.

Si un SER dispose, (a) d'un système de désignation interne bâti sur des noms globaux et uniques et (b) d'algorithmes de localisation des entités indépendants de la localisation, de la migration et de la duplication, il est alors possible de construire un sous-système de désignation symbolique (SDS) offrant une vision externe unique et indépendante de la localisation et capable de traduire les noms symboliques dans les noms internes des entités qu'ils désignent. En particulier, nous avons fait l'étude complet de la faisabilité d'un tel service pour la désignation des portes et des groupes de portes de Chorus (cf §2).

Cependant, dans la pratique, pour des raisons de performance et de facilité de mise en oeuvre, beaucoup de systèmes répartis intègrent la désignation symbolique et interne des entités aux serveurs qui les gèrent. Dans tels systèmes, les entités, autres que les serveurs, ne sont pas connues par des noms uniques et globaux. Elles ne sont caractérisables, de façon unique, que par rapport aux serveurs qui les gèrent. Ces serveurs peuvent ainsi être plus facilement mis en oeuvre parce qu'ils se trouvent, vis-à-vis la désignation et la gestion des ressources qu'ils matérialisent, dans un "environnement centralisé". C'est le cas de Chorus vis-à-vis de la gestion des fichiers.

Dans ces systèmes, le nom d'une entité est contextuel au serveur qui la gère. Si l'entité change de serveur elle doit aussi changer de nom. Par conséquence, seule la duplication des entités par les différents serveurs permet au système d'offrir une vision externe unique ainsi qu'indépendante des serveurs et de la localisation.

En absence de la duplication automatique, la vision externe offerte par le système est nécessairement celle d'un ensemble de sous-graphes interconnectés (un sous-graphe par serveur). Dans ces conditions, les problèmes devant lesquels se trouve le concepteur du système sont:

a) Comment désigner les serveurs?

b) Comment interconnecter les différents sous-graphes?

c) Quelle vision externe cette interconnexion offre?

Chorus résoud ces problèmes avec, respectivement, les *noeuds du type porte/groupe*, le protocole de redirection de requêtes et l'introduction de noms globaux au travers de la duplication de certains catalogues de référence. Ces catalogues permettent la désignation globale et la localisation des serveurs. En résumé, Chorus résoud ces problèmes en intégrant complètement le service de désignation et le service de gestion de fichiers.

Cette solution est ouverte à deux évolutions: la duplication automatique et l'extension du type de noeuds offerts par les arbres de désignation. Ces deux évolutions sont, simultanément, à la base de sa justification.

Notre étude met également en évidence que la généralisation de la notion de lien entre noeuds de désignation, par le biais de la redirection de requêtes entre serveurs "qui parlent la même langue", permet une grande souplesse de la configuration du graphe de désignation d'un SER. L'introduction de noms globaux, ou de n'importe quel autre modèle de désignation, revient alors à un choix de paramétrisation de ce graphe, en fonction du style de l'environnement que le système a choisi d'offrir. De telles fonctionnalités permettent de trouver des solutions intermédiaires assez simples et qui s'annoncent correctes vis-à-vis des contraintes et du scénario envisagé.

Grossièrement, dans Chorus, un serveur de fichiers, sans les extensions décrites dans cet ouvrage, offre fondamentalement les mêmes fonctionnalités que le noyau du système Unix pour la gestion des fichiers.

Etant donnée l'adéquation de l'architecture Chorus à la répartition, la mise en oeuvre des fonctionnalités supplémentaires des serveurs de fichiers, que nous venons de présenter, s'est traduite dans l'introduction de deux nouveaux types de noeuds; de moins d'une dizaine de nouveaux appels système; et dans la modification de la procédure d'interprétation des chemins d'accès. Par rapport aux résultats obtenus, nous pensons que cette complexité supplémentaire est négligeable.

CONCLUSIONS

CONCLUSIONS

Fondamentalement, le rôle des mécanismes de désignation et d'édition de liens caractéristiques des SER (systèmes d'exploitation répartis) est le même que ceux des mêmes mécanismes en environnement centralisé.

Cependant, l'environnement réparti introduit de nouveaux problèmes: les noms des ressources doivent permettre leur accès et partage de manière indépendante de la localisation; une entité ou une ressource doit pouvoir changer de localisation (migrer) ou être dupliquée, sans changer de nom; les pannes, les arrêts/redémarrages ou les reconfigurations des différents sites du système, ne doivent non plus modifier la relation entre les noms et les entités qu'ils désignent.

Cette thèse met en évidence aux chapitres I et II quelles sont les principales couches de désignation présentes dans un SER: adresses réseau, UID (noms uniques et globaux), noms internes contextuels et noms symboliques. Elle analyse le rôle de chacune de ces couches et met en évidence dans quelles conditions et en face de quels cahiers des charges elles doivent être utilisées. Cet analyse est, à nôtre avis, un résultat à retenir.

Nos travaux se sont déroulés au sein du projet Chorus de l'INRIA. Chorus est un système d'exploitation réparti qui a évolué d'une architecture de système réparti vers un système réparti, ouvert, complet et compatible avec Unix (System-V). Il met en oeuvre, aussi bien la construction d'applications réparties, style contrôle de processus, qu'un environnement réparti de développement et d'exécution d'applications réparties. Cette thèse décrit les mécanismes de désignation et d'édition de liens que nous avons été amenés à introduire dans le système pour permettre cette évolution.

Notre travail confirme que la désignation à l'aide d'UID est la méthode la plus simple générale et fiable de désignation en environnement réparti. En effet, les UID sont, dans Chorus, à la base du fonctionnement du système. Le noyau et les serveurs système s'en servent pour mettre en oeuvre et désigner les portes, les groupes, la communication et l'accès aux fichiers. Cette méthode de désignation s'est montré particulièrement bien adaptée à la transparence de la localisation et à la migration d'entités. Le système met en oeuvre, également, des algorithmes répartis de localisation dynamique d'entités nomades.

Nous avons introduit dans Chorus une nouvelle couche de désignation interne offrant des noms locaux ou contextuels aux processus. Ces noms constituent un contexte d'adressage protégé et qui peut être hérité de père en fils. Ce contexte cache aux clients du système les UID des entités, joue un rôle très important vis-à-vis de la protection et supporte aussi des mécanismes d'édition de liens au chargement. Des mécanismes simples, comme la transmission automatique du nom de l'émetteur, l'héritage de noms et les *noeuds symboliques du type porte/groupe* (introduits dans le service de gestion de fichiers) permettent l'acquisition dynamique de noms internes contextuels et l'édition de liens dynamique.

Au contraire d'autres systèmes qui utilisent aussi des noms internes contextuels, la solution de Chorus est assez simple. Elle consiste en séparer l'espace de noms vus par les clients, de l'espace des noms vus par le système (serveurs système et noyau). Ainsi, comme les composants système reposent directement sur la visibilité directe des UID, tous les avantages de cette visibilité directe peuvent être utilisés à leur profit. Le résultat final est un système simple et peu pénalisé par les problèmes sous-jacents à

l'extension à la répartition des méthodes d'adressage contextuels.

Le SDS (Sytème de Désignation Symbolique) de Chorus répond à deux contraintes essentielles: fournir des fonctionnalités du style "serveur de noms", requises par le développement d'applications réparties, et supporter un système de gestion de fichiers répartis qui est à la base de l'environnement de développement de logiciel offert par le système. Ces contraintes sont satisfaites par une méthode qui consiste à intégrer les fonctionnalités d'un service réparti de désignation aux serveurs de fichiers. Ces fonctionnalités mettent en ouevre également des noms symboliques globaux, i. e., indépendants de la localisation.

Nous avons aussi réalisé des expériences d'introduction de fonctionnalités de diffusion, accessibles aux clients du système. Chorus rejoint les rares systèmes répartis qui offrent les concepts de groupe de portes et de groupe de processus. Ces groupes mettent en oeuvre le *contrôle d'exécution* des applications réparties et la notion de service réparti avec plusieurs points d'accès équivalents. En particulier, nous avons introduit la notion de *mode d'adressage*, dont un cas particulier permet la communication de 1 à (1 parmi N). Cette forme de communication est intéressante du point de vue de l'édition de liens, comme nous l'avons mis en évidence.

Les mécanismes de désignation et d'édition de liens introduits dans un système ne peuvent être apréciés que dans le contexte du système qui les utilise. Ceux de Chorus se caractérisent de la même forme que le système lui-même, par un souci de recherche de généralité avec de la simplicité. Cette simplicité est obtenue sans sacrifier le niveau des services offerts aux clients.

CONCLUSIONS

BIBLIOGRAPHIE

BIBLIOGRAPHIE

[Almes 85] G.T. Almes, A.P. Black, E.D. Lazowska, J.D. Noe
The Eden System: A Technival Review
IEEE Transactions on Software Engineering, vol. SE-11, 1, (January 1985), pp. 43, 58

[Armand 85] F. Armand, B. Deslandes, M. Gien, M. Guillemont, P. Léonard
Intégration des systèmes CHORUS et UNIX
Journées SM90, (Décembre 1985). pp. 10

[Armand 86] F. Armand, M. Gien, M. Guillemont, P. Léonard
Towards a Distributed UNIX System - The Chorus Approch
EUUG, Autumn'86. Manchester, (Septembre 1986), pp. 413, 431

[Banino 82] J.S. Banino, A. Caristan, C. Gailliard, M. Guillemont, G. Morisset, H. Zimmermann
Architecture of the CHORUS Distributed System
INRIA/PCL International Seminar "Synchronization, Control and Communication in Distributed Computing Systems", London, England, (September 1982)

[Banino 85] J.S. Banino, J.C. Fabre, M. Guillemont, G. Morisset, M. Rozier
Experiments in Fault-Tolerant Computing with the CHORUS Distributed System
ACM/IBM Workshop on Operating Systems in Computer Networks, Zurich, Switzerland, (January 1985), pp. 11

[Birrell 82] A.D. Birrell, R. Levin, R.M. Needham, M.D. Schroeder
Grapevine : An Exercise in Distributed Computing
Communications of the ACM, vol 25, 4, (April 1982), pp. 260, 274

[Birrell 84] A.D. Birrell, B.J. Nelson
Implementing Remote Procedure Calls
ACM Transactions on Computer Systems, vol 2, 1, (February 1984), pp. 39, 59

[Brown 84] R.L. Brown, P.J. Denning, W.F. Tichy
Advanced operating systems
Computer, (October 1984), pp. 173, 190

[Brownbridge 82] D.R. Brownbridge, L.F. Marshall, B. Randell
The Newcastle Connection or UNIXes of the World Unite!
Software-Practise and Experience, vol 12, 1147-1162, (1982), pp. 16

[Cheriton 84] D.R. Cheriton
The V-Kernel: a software base for distributed systems
Research report, Computer Science Department, Standford University, (April 1984), pp. 22

[Cheriton 84a] D.R. Cheriton, T.P. Mann
Uniform access to distributed name interpretation in the V-system
Research report, Computer Science Department, Standford University, (December 1984), pp. 8

[Cheriton 85] D.R. Cheriton, W. Zwaenepoel
Distributed process groups in the V kernel
ACM Transactions on Computer Systems, Vol. 3, No. 2, (May 1985), pp. 77-107

[Comer 85] D.E. Comer, R.E. Droms
Tilde Trees in the UNIX environment
Tilde report CSD-TR-503, (January 1985), pp. 11

[Comer 86] D.E. Comer, L.L. Peterson
A Model of Name Resolution in Distributed Systems
The 6th International Conference in Distributed Systems Computing Systems
Cambridge, Massachusetts, (May 1986), pp. 523, 530

[Dion 80] J. Dion
The Cambridge File Server
Operating Systems Review, vol 14, 4, (October 1980), pp. 26, 35

[ECMA 85] ECMA
OSI Directory access service and protocol
Final draft, (July 1985), pp. 67

[Finger 81] U. Finger, G. Médigue
Architectures multi-microprocesseurs et disponibilité : la SM90
L'écho des recherches no 105, (Juillet 1981), pp. 15, 21

[Guillemont 82] M. Guillemont
The CHORUS distributed operating system : design and implementation
International Symposium on Local Computer Networks, Florence, Italy, (April 1982), pp. 207, 223

[Guillemont 84] M. Guillemont
Etude comparative de quelques systèmes répartis
TSI, vol 3, 1, (Janvier 1984), pp. 5, 21

[Guillemont 84a] M. Guillemont, H. Zimmermann, G. Morisset, J.S. Banino
CHORUS : une architecture pour les systèmes répartis
Rapport de recherche INRIA 274, (Mars 1984), pp. 78

BIBLIOGRAPHIE

[Guillemont 86] M. Guillemont, J. Legatheaux Martins
CHORUS: a next generation Unix, or CHORUS: UNIX in the communication age
Paper submited for publication. Currently available from the authors at INRIA, (December 1986), pp. 25

[Hoare 78] C. A. R. Hoare
Communicating Sequential Processs
Communications of the ACM, Vol. 21, 8, (August 1978), pp. 666, 677

[Kaiser 84] C. Kaiser
Problèmes actuels des Systèmes Répartis
Congrès "De Nouvelles Architectures pour les Communications", Paris, Eyrolles, (Septembre 1984), pp. 251, 258

[Leach 82] P.J. Leach et al
UIDS as Internal Names in Distributed Systems
ACM Symposium on Principles of Distributed Computing, Ottawa, Canada, (August 1982), pp. 8

[Leach 83] P.J. Leach et al
The Architecture of an Integrated Local Network
IEEE Journal on Selected Areas in Communications, vol 1, 5, (November 1983), pp. 842, 857

[Legatheaux 85] J. Legatheaux, A. Rozz
Arbres, montages et liens Unix dans le système réparti CHORUS (V2)
Note interne de l'équipe CHORUS, (Juillet 1985), pp. 35

[Legatheaux 85a] J. Legatheaux
Maestro - le service de désignation symbolique de CHORUS (V2)
Note interne de l'équipe CHORUS, (Juillet 1985), pp. 15

[Liskov 81] B.H. Liskov
Report on the Workshop on Fundamental Issues in Distributed Computing, Fallbrook, California, December 1980
Operating Systems Review, vol 15, 3, (July 1981), pp. 9, 38

[LOCUS 84] LOCUS
The LOCUS distributed system architecture, edition 3.1
LOCUS Computing Corporation, (June 1984), pp. 102

[Metcalfe 76] R.M. Metcalfe, D.R. Boggs
Ethernet : Distributed Packet Switching for Local Computer Networks
Communications of the ACM, vol 19, 7, (July 1976), pp. 395, 404

[Mitchell 78] J.G. Mitchell, W. Maybury, R. Sweet
Mesa Language Manual
Xerox Parc Research Report CSL-78-1, (February 1978), pp. 155

BIBLIOGRAPHIE

[Oppen 81] The Clearinghouse: A decentralized Agent for Locating Named Objects in a Distributed Environnement
Draft from Xerox Office Products Division Systems Development Departement, (July 1981), pp 53

[Pike 85] R. Pike, P.J. Weinberger
The hideous name
Usenix, Portland, (June 1985), pp. 563, 568

[Popek 81] G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin, G. Thiel
LOCUS : a Network Transparent, High Reliability Distributed System
8th ACM Symposium on Operating System Priniples, Pacific Grove, California, (December 1981), pp. 169, 177

[Powell 83] M.L. Powell, B.P. Miller
Process Migration in DEMOS/MP
9th ACM Symposium on Operating Systems Principles / OSR ACM, vol 17, 5, (October 1983), pp. 110, 119

[Radia 83] S. Radia
The Kayak Filling System
Internal Report, INRIA, Kayak Project, (May 1983), pp. 36

[Rashid 81] R.F. Rashid, G.G. Robertson
Accent : A communication oriented network operating system kernel
8th ACM Symposium on Operating System Principles, Pacific Grove, California, (December 1981), pp. 25

[Ritchie 74] D.M. Ritchie, K. Thompson
The UNIX Time-Sharing System
Communications of the ACM, vol 17, 7, (July 1974), pp. 365, 375

[Rozier 86] M. Rozier
Expression et Réalisation du Contrôle d'Exécution dans un Système Réparti
Thèse de l'Institut National Polytechnique de Grenoble (arrêté du 5 Juillet 1984), (Octobre 1986), pp. 180

[Rozz 85] A. Rozz
La gestion des fichiers dans le système réparti CHORUS
Thèse de Docteur Ingénieur, Université Paul Sabatier, Toulouse, (Octobre 1985), pp. 144

[Sandberg 85] R. Sandberg, D. Goldberg, S. Kleiman, D. Walsh, B. Lyon
Design and implementation of the Sun network filesystem
Usenix, Portland, (June 1985), pp. 119, 130
REP 4 067

BIBLIOGRAPHIE

[Senay 83] C. Senay
Un système de désignation et de gestion de portes pour l'architecture répartie CHORUS
Thèse de Docteur Ingénieur, C.N.A.M., (Décembre 1983), pp. 200

[Satyanaray 85] M. Satyanarayanan et all
The ITC Distributed File System: Principles and Design
10th ACM Symposium on Operating Systems Principles, Orcas Island, Washington, (December 1985), pp. 35, 50

[Shoch 78] J.F. Shoch
Inter-Network Naming, Addressing, and Routing
17th IEEE Computer Society International Conference - Compcon (September 1978), pp. 72, 79

[Sloman 85] M. Sloman, J. Kramer, J. Magee
The Conic Toolkit for Building Distributed Systems
6th IFAC Distributed Computer Control Systems Workshop, Monterey, California, USA, (May 1985)
To be published by IEEE Proc. on Control Theory and Applications

[Stankovic 84] J.H. Stankovic
A Perspective On Distributed Computer Systems
IEEE Transactions on Computers, vol C-33, 12, (Decembre 1984), pp. 13

[Teitelman 84] W. Teitelman
The CEDAR Programming Environment : A Midterm Report and Examination
Xerox Corporation, Palo Alto Research Center, CSL-83-11, (June 1984), pp. 120

[Terry 84] D.B. Terry, M. Painter, D.W. Riggle, S. Zhou
The Berkeley Internet Name Domain Server
USENIX Summer'84, Salt Lake City, Utah, (June 1984), pp. 23, 31

[Theimer 85] Theimer M. M., Lantz A., Cheriton D.
Preemptable Remote Execution Facilities for the V-System
The Tenth ACM Symposium on Operating Systems Principles, (December 1985), pp. 2, 12

[Thompson 78] K. Thompson
UNIX Implementation
The Bell systems Technical journal, vol 5, 6, Part 2, (July 1978), pp. 1931, 1946

[Thompson 85] M. R. Thompson, R. D. Sanson, M.B. Jones, R.F. Rashid
Sesame : The Spice File System
Carnegie-Mellon University, Technical Report CMU-CS-85-172, (December 1985), pp. 29

[Tichy 84] W.F. Tichy, Z. Ruan
Towards a Distributed File System
USENIX Summer'84, Salt Lake City, Utah, (June 1984), pp. 87, 97

[Weinberger 84] P.J. Weinberger
The Version 8 Network File System
USENIX Summer'84, Salt Lake City, Utah, (June 1984), pp. 10

[Welch 86] B. Welch, J. Ousterhout
Prefix Tables: A simple mechanism for Locating Files in a Distributed System
The 6th International Conference in Distributed Systems Computing Systems, Cambridge, Massachusetts, (May 1986), pp. 523, 530

[Zimmermann 84] H. Zimmermann, M. Guillemont, G. Morisset, J.S. Banino
CHORUS : A communication and processing architecture for distributed systems
Rapport de recherche INRIA 328, (Septembre 1984), pp. 89

BIBLIOGRAPHIE

ANNEXE I

LISTE DES ABREVIATIONS UTILISEES DANS CET OUVRAGE

ANNEXE I

ABREVIATIONS UTILISEES DANS CET OUVRAGE

Abréviation	Mnémonique de	Description
IDP	Identificateur au sens de la Protection	La paire de numéros (*uid, gid*), identificateur d'usager et de groupe d'usagers, qui caractérisent, du point de vue de la protection, un usager du système Chorus.
IPC	*Inter Process Communication facility*	Ensemble des facilités offertes par le système pour la communication entre processus.
PGD	*Port/Group Descriptor*	Nom contextuel d'une porte ou d'un groupe de portes dans Chorus.
RPC	*Remote Procedure Call*	Appel d'une procédure s'exécutant dans une autre machine
SDS	Système de Désignation Symbolique	
SER	Système d'Exploitation Réparti	
UID	*Unique IDentifier*	Nom interne unique et global

ANNEXE II

DESCRIPTION SOMMAIRE DU SYSTEME UNIX

ANNEXE II

DESCRIPTION SOMMAIRE DU SYSTEME UNIX

Unix ([Ritchie 74], [Thompson 78]) est un système centralisé universel, originellement distribué par les Laboratoires Bell, dont cet annexe rappelle quelques caractéristiques.

Unix comprend un système d'exploitation, appelé noyau Unix, autour duquel s'agencent un interpréteur de commandes (le *Shell*) et un grand nombre d'utilitaires.

Le noyau Unix est essentiellement caractérisé par:

a) un système de fichiers hiérarchisé permettant le montage et le démontage de volumes,

b) une vision uniforme des entrées/sorties par l'utilisateur, que celles-ci s'appliquent à des fichiers, des périphériques ou des supports de communication inter-processus, voir ci-dessous,

c) un mécanisme de communication entre processus, les tubes (*pipes*), selon le schéma "producteur/consommateur",

d) la possibilité pour un programme de lancer l'exécution de processus asynchrones,

e) la possibilité d'étendre l'ensemble des commandes disponibles sans modifier le noyau du système.

1. Système de gestion de fichiers

Le système de gestion de fichiers comporte un SDS avec trois types de noeuds:

- les fichiers ordinaires: une séquence quelconque d'octets enrégistrée de façon permanente,
- les noeuds périphérique: représentant des périphériques, et
- les catalogues: des catalogues de désignation symbolique.

Il est organisé en arbre. Les différents identificateurs composant un chemin d'accès sont séparés par des "/". Un nom global doit aussi commencer par le caractère "/". Chaque processus a une notion de catalogue courant; un nom de fichier relatif à ce catalogue est un nom qui ne commence pas par "/".

L'arbre des fichiers est unique. Cependant, le système supporte la notion de "sous-arbre monté", i. e., résidant dans un volume physique particulier. Cette structure logique de sous-arborescences est plaquée sur une structure physique de sous-système de fichiers.

Un sous-système physique de fichiers correspond en général à un volume disque-physique. Chaque fichier est répertorié par une entrée, appelée "i-noeud" (*i-node*). Chaque i-noeud est repéré par un numéro unique au volume, mais réutilisé, qui n'est autre que son index dans la liste des i-noeuds, la i-liste (*i-list*). Chaque i-noeud est un descripteur de fichier qui contient ses attributs. Un répertoire ou catalogue est une table

qui applique des identificateurs en des i-noeuds. Cette table ne peut contenir que des références à des fichiers du même volume que le catalogue.

2. Entrées/Sorties

Dans Unix, les entrées/sorties sont uniformes, qu'il s'agisse d'opérations sur des fichiers, des périphériques, des tubes, etc. Elles ont toujours lieu sur des "séquences d'octets", c.a.d. sur des "fichiers".

Avant d'effectuer toute entrée/sortie, il faut ouvrir (ou créer) le fichier correspondant. Ceci est réalisé par les fonctions *Open* et *Creat*, appelées par:

```
integer fd;
...
fd := open ( nom, mode);
fd := creat ( nom, mode);
```

L'entier "fd" est un descripteur de fichier (*file descriptor*), un nom interne contextuel au processus qui le possède. A chaque fichier ouvert est associé un pointeur courant sur le prochain octet à lire ou à écrire. Les opérations de lecture et écriture (*read/write*) sont toujours relatives à cette position courante. La primitive *close* permet de fermer un fichier et libère aussi son descripteur (*fd*).

3. Les tubes

Pour assurer la communication entre processus le système Unix offre un type particulier de "séquence d'octets" appelée "tube" (*pipe*). Les processus y appliquent les mêmes opérations d'entrée/sortie que sur les fichiers "normaux". Seule l'opération de création est différente. Un processus ne peut réaliser des opérations sur un tube que s'il possède un descripteur du tube obtenu soit par création du tube, soit par héritage (voir ci-dessous). La coopération entre processus repose souvent sur la communication d'informations entre processus par l'intermédiaire de ces tubes. Les schémas de coopération les plus fréquents sont les suivants:

- le processus père crée un processus fils afin de lui faire exécuter une commande déjà existante et de:
  - soit traiter les résultats de son exécution,
  - soit lui faire traiter des données qu'il lui fournira,
- le processus père crée deux processus fils, l'un producteur et l'autre consommateur, et relie par un tube la sortie du producteur à l'entrée du consommateur.

4. Les processus

L'utilisateur a la possibilité de lancer des processus asynchrones au travers de l'appel *fork*. Cet appel est le seul moyen de créer de nouveaux processus sous Unix:

```
integer pid;
....
pid := fork;
```

où *pid* (*process-id*) est un identificateur de processus, unique au système, mais réutilisé d'un chargement à l'autre, qui désigne le processus nouvellement crée.

L'image mémoire du processus ainsi créé est la copie conforme de son père (i. e., du processus qui a appelé "fork"). Seul le résultat de la fonction distingue le père du fils. Après un *fork*, les deux processus partagent non seulement tous les fichiers ouverts, les pointeurs d'entrée/sortie sur ces fichiers mais aussi les tubes de communication. Ils possèdent également le même catalogue courant.

Un autre mécanisme est disponible pour pouvoir exécuter d'autres programmes. C'est l'appel *exec* qui remplace l'image mémoire d'un processus par celle d'un autre programme. Le processus ne change pas après un *exec*; il possède toujours le même *pid*, ou *process-id*. Seul le programme associé a changé. Dans Unix, les processus ne sont donc pas liés indissolublement à un programme. De plus, après un *exec* il y a un certain transfert d'attributs; en particulier, avant de ce métamorphoser, le processus peut décider (par le biais de l'appel *fcntl*) quels sont les fichiers ouverts et les pipes créés avant l'*exec* qui sont "transmis" au nouveau programme.

Un processus ne peut revenir d'un *exec*: ses segments de "code" et de "données" avant l'*exec* sont perdus. S'il souhaite exécuter un autre programme, il doit d'abord effectuer un *fork* et s'arranger pour que son fils fasse un *exec* de l'autre programme.

5. Synchronisation entre processus

La synchronisation entre processus père et fils s'effectue par l'appel *wait*: l'appelant suspend son exécution jusqu'à ce que l'un de ses fils ait terminé de s'exécuter. L'interpréteur de commandes fonctionne selon ce principe. Après l'analyse d'une commande, il lance, par un *fork*, un processus qui va dérouler (par un *exec*) le programme correspondant à la comande. Pendant ce temps-là, l'interpréteur (le père) attend. A la mort de son fils, i. e., à la fin de l'exécution de la commande, l'interpréteur est réveillé et est prêt à interpréter une autre commande. Pour lancer des processus en parallèle, l'interpréteur ne se met pas en attente après le *fork*.

6. Les signaux

L'autre moyen d'aide à la synchronisation entre processus est celle qui est offerte par le biais des "signaux". Un processus peut envoyer un "signal" à un autre processus par le biais de l'appel: *kill*.

```
kill ( pid, sig );
```

La réception du signal (*sig*) par le processus visé (*pid*) préempte le traitement qu'il exécutait couramment et l'action associée au signal est exécutée. L'action à exécuter peut être spécifiée par le récepteur ou est prise par défaut. Par exemple, le signal "*SIGKILL*", déclenche dans tous les cas l'appel *exit*, appel d'auto-destruction.

7. Les groupes de processus

Unix supporte plusieurs types de groupes de processus. Chaque processus appartient en permanence à quatre groupes; trois de ces groupes sont figés: le groupe de tous les processus application du système, le groupe de tous les processus qui s'exécutent au nom d'un usager U et le groupe de tous les processus dont l'entrée est attachée à une console C. Le quatrième groupe de processus auquel un processus peut appartenir est un groupe qui dépend de sa filiation.

Ainsi, si un processus appelle:

```
integer group-id;
....
group-id := setpgrp;
```

le système crée un nouveau groupe de processus, dont l'identificateur est *group-id* (*group-identifier*), auquel l'appelant appartient automatiquement. Tous ses fils appartiendront aussi à ce groupe jusqu'à que l'un d'eux décide de créer un nouveau groupe de processus. Ces groupes sont donc organisés en une hiérarchie. Cette hiérarchie s'apparente à un "arbre généalogique" des processus. Le processus créateur d'un groupe est le *process-leader* du groupe.

L'appel *signal* peut être aussi utilisé pour envoyer des signaux aux groupes de processus. Chaque processus du groupe recevra un signal comme si celui-ci lui avait été envoyé directement.

8. La protection

Le fichier "/etc/passwd" est un fichier spécial qui contient une base de données définissant les usagers du système. Chaque ligne de ce fichier est de la forme:

user : mot de passe cripté : *uid* : *gid* : ....

où *user* est le nom symbolique de l'usager, *uid* est son nom interne (un numéro) et *gid* est le nom interne de son groupe d'usagers (un numéro).

Quand un usager ouvre une session dans le système, son interpréteur de commandes reçoit son couple de numéros d'identification au sens de la protection: (*uid*, *gid*). Ce couple est, grossos modo, l'identification de l'usager au nom duquel l'interpréteur et ses fils s'exécuteront. La protection des opérations de manipulation des entités (fichiers, processus, etc.) est basée sur ce couple de numéros.

Le fichier "/etc/group" contient aussi une base de données des groupes d'usagers du système; il définit leurs noms symboliques, leurs noms internes (des numéros) et la liste des membres du groupe.

9. L'interpréteur de commandes

L'interpréteur de commandes est lui-même une commande (un processus) *sh*, ou *shell*, i. e., un fichier exécutable. Après l'ouverture d'une session, le système associe au terminal de l'utilisateur un processus *shell* qui interprète les commandes données sur ce terminal. L'interpréteur n'est donc pas un programme privilégié et il est possible de définir un nouvel interpréteur sans modifier le noyau.

Le langage de commande du *shell* d'Unix est assimilable à un langage interactif de programmation. Parmi ses plus intéressantes fonctionnalités on compte la possibilité de redéfinir les fichiers d'entrée/sortie utilisés par les commandes lancées et la possibilité de lancer des *pipe-lines* de commandes: des commandes qui coopèrent par un schéma "producteur/consommateur" pour devenir une "macro-commande". Le *shell* supporte aussi des "macro-commandes" définies par le biais de "procédures de commandes" ou *shell-scripts*.

ANNEXE III

EXTRAIT DU MANUEL DE L'INTERFACE DU SYSTEME CHORUS

ANNEXE III

EXTRAIT DU MANUEL DE L'INTERFACE CHORUS

On trouve ici la définition des appels système Chorus concernés par cet ouvrage. Ce document contitue un extrait du document qui décrit l'interface du système CHORUS (ce dernier est un document provisoire dont toute reproduction est interdite sans autorisation explicite de la part des auteurs; l'équipe Chorus se réserve le droit d'en changer le contenu à tout moment).

Cette interface est un sur-ensemble de celle du système Unix tel qu'elle est définie dans:

"X/OPEN Portability guide (July 1985) X/OPEN Members, Elsevier Science Publishers B. V., Amsterdam" ,
Part II: The X/OPEN System V Specification,
Chapter 2: System Calls.

NOM

call - envoi d'un message et attente de son retour

SYNTAXE PASCAL

```
function call (
                        VAR box: integer;
                        VAR addri: address;
                        srcport: integer;
                        destport: integer;
                        mode: integer;
                        delay: integer): integer;
```

SYNTAXE C

```
int call ( box, addri, srcport, destport, mode, delay )
int *box;
char *(*addri);
int srcport;
int destport;
int mode;
int delay;
```

PARAMETRES

box : à l'appel, la boîte contenant le message de requête.
au retour, la boîte contenant le message de réactivation.

addri : au retour, l'adresse de l'item sommet de pile.

srcport : numéro de descripteur de la porte émettrice du message de requête.

destport : numéro de descripteur de la porte réceptrice du message de requête.

mode : mode d'adressage pour l'envoi de la requête.

delay : valeur de la temporisation.

DESCRIPTION

Cette fonction provoque l'émission du message contenu dans *box* de *srcport* vers *destport* avec le mode d'adressage *mode,* et l'armement d'une temporisation de *delay*. L'acteur ne sort de cette fonction que lorsque le message de réponse ou un message de temporisation échue a été reçu. Au retour, le message de réactivation se trouve dans la boîte *box* et *addri* contient l'adresse de l'item de sommet de pile. La réception du message de réponse désarme la temporisation.

Si *destport* désigne une porte, le paramètre *mode* est ignoré. Si *destport* désigne un groupe de portes, *mode* ne peut prendre qu'une valeur correspondant à un adressage fonctionnel. Les valeurs acceptées de *mode* sont donc les suivantes:

RESTFUNC Le message sera émis vers une porte du groupe locale au sous-réseau dans lequel s'exécute l'acteur appelant.

FUNC Le message sera émis vers une porte du groupe.

Le mode PORTTOPORT n'est utilisable que si *destport* désigne une porte.

RESULTAT

0 si l'opération a été effectuée,
1 si l'acteur a été réactivé par un message de temporisation échue,
-1 si erreur.

ERREURS

EBADBOX : la boîte est inexistante ou ne contient pas de message

ENOMEM : la requête ne peut être déposée, par manque de ressources

ENOTPOPEN : *srcport* ne désigne pas une porte ouverte par l'acteur appelant.

EBADPGD : *destport* n'est pas connu de l'acteur.

EBADMODE : *mode* n'existe pas ou n'est pas autorisé.

ETIMEOUT : l'acteur a été réactivé par un message de temporisation échue.

EXEMPLES

VOIR AUSSI

putreply(2), putmsg(2), timeout(2), istimeout(2)

NOM

closeport – fermeture d'une porte

SYNTAXE PASCAL

function closeport (port : integer) : integer;

SYNTAXE C

int closeport (port)
int port;

PARAMETRES

port : numéro du descripteur de la porte à fermer.

DESCRIPTION

Cette fonction sert à fermer la porte ayant le descripteur *port*. Pour que la porte puisse être fermée, il faut qu'elle soit, au préalable, ouverte par l'appelant.

Un acteur ne peut pas fermer sa "porte des appels système" courante ni sa "porte ombilicale" courante. Pour ce faire, il doit au préalable les changer par *syscallport(2)* ou par *ubport(2)*.

Si une porte est fermée, elle ne garde aucunne trace ni de l'étape de traitement, ni des sélections, ni des temporisations qui lui étaient associées. Une porte fermée ne peut non plus recevoir des messages.

RESULTAT

0 si la porte a été fermée,
-1 s'il y a une erreur.

ERREURS

EBADPGD	: le descripteur *port* est invalide.
ENOTPOPEN	: le descripteur *port* ne désigne pas une porte ouverte de l'acteur.
EINVAL	: l'acteur a essayé de fermer sa "porte des appels système" ou sa "porte ombilicale".
ENOMEM	: la requête ne peut être déposée, par manque de ressources.

EXEMPLES

VOIR AUSSI

openport(2), creatport(2), dlport(2), syscallport(2), ubport(2).

NOM

creatgroup – création d'un groupe de portes

SYNTAXE PASCAL

function creatgroup : integer;

SYNTAXE C

int creatgroup ()

DESCRIPTION

Cette opération crée un groupe de portes et retourne le numéro de son descripteur. Après sa création le groupe est vide. Pour insérer ou rétirer des portes du groupe voir *groupctl(2)*. Un groupe de portes est, soit détruit explicitement par l'opération *dlgroup(2)*, soit détruit automatiquement par le système, voir ci-dessous.

Il existent aussi des groupes de portes qui n'ont pas à être créés, ni détruits, ce sont les groupes pré-définis, c.a.d. des groupes de portes logiquement toujours existants, voir *getpgroup(2)*.

Un groupe est une liaison logique entre en ensemble de portes. Cette liaison ne prend effet que sous le point de vue de la communication. Par exemple, la destruction d'un groupe ne détruit pas les portes qui en font couramment partie, une porte peut appartenir simultanément à plusieurs groupes (ou à aucun).

Par défaut, quand un message est envoyé à un groupe de portes, il est diffusé à l'ensemble des portes qui en font couramment partie. Cependant, l'émetteur peut limiter l'ensemble des portes appartenant au groupe qui vont effectivement recevoir une copie du message à l'aide du paramètre "mode d'adressage" des opérations d'envoi de messages, voir *putmsg(2) et call(2)*.

Attachés à un groupe de portes il y a un id-protection, l'attribut "auto-destruction" et des attributs de protection. Leurs valeurs sont données à la création et sont immuables par la suite. L'id-protection d'un groupe est l'id-protection effectif courant de son créateur au moment où la création a lieu (c.a.d. l'id-protection de la "porte des appels système" au moment de la création). L'attribut "auto-destruction" et les attributs de protection prennent des valeurs par défaut, voir *gmask(2)*.

Quand une porte sort d'un groupe, si le groupe devient vide et il a l'attribut "auto-destruction" positionné, il est automatiquement détruit par le système. Un groupe qui a cet attribut positionné ne peut pas être explicitement détruit par *dlgroup(2)*, il ne peut être qu'implicitement détruit (et vice-versa).

RESULTAT

le descripteur du groupe si *creatgroup* > *0*,
-1 s'il y a une erreur.

ERREURS

EPGDOFLOW	: trop de descripteurs connus de l'acteur.
EPOFLOW	: trop de groupes dans les tables du système.
ENOMEM	: la requête ne peut être déposée, par manque de ressources.

EXEMPLES

VOIR AUSSI

groupctl(2), dlgroup(2), gmask(2), getpgroup(2)

NOM

creatport – création d'une porte

SYNTAXE PASCAL

function creatport : integer;

SYNTAXE C

int creatport ()

DESCRIPTION

Cette opération crée une porte et retourne le numéro de son descripteur.

Une porte est créée dans l'état *fermée*. Dans cet état elle ne peut ni émettre ni recevoir des messages. Pour se servir d'une porte afin d'émettre ou de recevoir des messages, un acteur doit au préalable l'ouvrir, voir *openport(2)*. L'ouverture d'une porte revient donc à l'obtention par l'acteur du droit d'envoyer et de recevoir des messages par cette porte. Pour avoir le droit d'envoyer des messages vers une porte, ou un groupe de portes, il suffit d'avoir acquis leurs noms.

Attachés à une porte il y a un id-protection, des attributs de protection et un attribut concernant sa destruction implicite par le système. Leurs valeurs sont définis à la création et sont immuables par la suite; l'exception est l'id-protection de la "porte des appels système" courante qui peut être modifié par *setuid(2) et setgid(2)*.

L'id-protection affecté à une porte à sa création est l'id-protection effectif courant de son créateur au moment où la création a lieu (c.a.d. l'id-protection de la "porte des appels système" au moment de la création). Les autres attributs prennent des valeurs par défaut, voir *pmask(2)*.

RESULTAT

le descripteur de la porte si *creatport* > *0*,
-1 s'il y a une erreur.

ERREURS

EPGDOFLOW	: trop de descripteurs connus de l'acteur.
EPOFLOW	: trop de portes dans les tables du système.
ENOMEM	: la requête ne peut être déposée, par manque de ressources.

EXEMPLES

VOIR AUSSI

openport(2), closeport(2), dlport(2), pmask(2)

NOM

dlgroup – destruction d'un groupe de portes

SYNTAXE PASCAL

function dlgroup (group : integer) : integer;

SYNTAXE C

int dlgroup (group)
int group;

PARAMETRES

group : numéro du descripteur du groupe à détruire.

DESCRIPTION

Cette fonction sert à détruire le groupe désigné par le descripteur de numéro *group*. Pour qu'il puisse être détruit, il faut que l'id-protection effectif de l'appelant (c.a.d. celui de sa "porte des appels système" courante) possède le droit "modification" sur ce groupe et que l'attribut "auto-destruction" ne soit pas positionné, voir *gmask(2)* et *creatgroup(2)*. Un groupe de portes prédéfini, voir *getpgroup(2)*, ne peut pas non plus être détruit. La destruction d'un groupe ne concerne pas les portes qui en font couramment partie.

Si l'appel réussit, le descripteur *group* est libéré. Remarquons qu'un acteur peut toujours libérer le descripteur d'un groupe en utilisant *relpgd(2)*.

RESULTAT

0 si le groupe a été détruit,
-1 s'il y a une erreur.

ERREURS

EBADPGD	: le descripteur *group* est invalide.
EBADGROUP	: le descripteur *group* n'est pas celui d'un groupe.
EACCES	: droits insuffisants.
ENOMEM	: la requête ne peut être déposée, par manque de ressources.
ETIMEOUT	: le groupe *group* ne désigne plus un groupe connu du système ou il y a des problèmes de communication.

EXEMPLES

VOIR AUSSI

creatgroup(2), groupctl(2), gmask(2), getpgroup(2)

NOM

dlport – destruction d'une porte

SYNTAXE PASCAL

function dlport (port : integer) : integer;

SYNTAXE C

int dlport (port)
int port;

PARAMETRES

port : numéro du descripteur de la porte à détruire.

DESCRIPTION

Cette fonction sert à détruire la porte désignée par le descripteur de numéro *port*. Cette porte doit necessairement être une porte fermée. Pour que la porte puisse être détruite, il faut que l'id-protection effectif de l'appelant (c.a.d. celui de sa "porte des appels système" courante) en possède le droit, voir *pmask(2)*.

Si l'appel réussit, le descripteur *port* est libéré. Remarquons qu'un acteur peut toujours libérer le descripteur d'une porte fermée en utilisant *relpgd(2)*.

RESULTAT

0 si la porte a été détruite,
-1 s'il y a une erreur.

ERREURS

EBADPGD	: le descripteur *port* est invalide.
ENOTPCLOSED	: le descripteur *port* ne désigne pas une porte fermée.
EACCES	: droit "destruction" refusé.
ENOMEM	: la requête ne peut être déposée, par manque de ressources.
ETIMEOUT	: la porte *port* n'existe plus dans le système ou il y a des problèmes de communication.

EXEMPLES

VOIR AUSSI

creatport(2), openport(2), closeport(2), pmask(2).

NOM

getchild – obtention de la "porte ombilicale" du dernier fils créé

SYNTAXE PASCAL

function getchild : integer;

SYNTAXE C

int getchild ()

DESCRIPTION

Cette fonction rend un descripteur qui représente la "porte ombilicale" du dernier fils créé telle qu'elle a été définie au moment de la naissance du fils, voir *ubport(2)*.

Remarquons que le succès de cette opération ne garantit pas que cette porte soit encore une porte ouverte du fils car, par exemple, il peut l'avoir fermée; de même, la porte peut aussi ne plus exister car, par exemple, le fils peut être détruit.

RESULTAT

le numéro du descripteur si *getchild* > *0*,
-1 s'il y a une erreur.

ERREURS

EPGDOFLOW	: trop des descripteurs connus de l'acteur.
ECHILD	: l'appelant n'a jamais créé de fils

EXEMPLES

VOIR AUSSI

getparent(2), ubport(2)

NOM

getparent – obtention de la "porte ombilicale" du père

SYNTAXE PASCAL

function getparent : integer;

SYNTAXE C

int getparent () .

DESCRIPTION

Cette fonction retourne un descripteur qui représente la "porte ombilicale" du père de l'acteur telle qu'elle était définie au moment où la création de l'appelant a eu lieu.

Remarquons que le succès de cette opération ne garantit pas que cette porte soit encore une porte ouverte du père car, par exemple, le père du processus appelant peut l'avoir fermée; de même, la porte peut aussi ne plus exister car, par exemple, le père peut être détruit.

RESULTAT

le numéro du descripteur si *getparent* > *0*,
-1 s'il y a une erreur.

ERREURS

EPGDOFLOW	: trop des descripteurs connus de l'acteur.
EPARENT	: cet acteur n'a pas de père; il s'agit d'un cas très particulier: l'appelant est un serveur système chargé en temps de chargement du système.

EXEMPLES

VOIR AUSSI

getchild(2), ubport(2)

NOM

getpgroup – rend visible le descripteur d'un groupe pré-défini

SYNTAXE PASCAL

function getpgroup (name : integer) : integer;

SYNTAXE C

int getpgroup (name)
int name;

PARAMETRES

name: nom (rélatif à l'uid effectif courant de l'appelant) du groupe de portes pré-défini

DESCRIPTION

Cette opération rend visible par l'acteur le descripteur du groupe de portes pré-défini désigné par *name*.

Un groupe pré-défini de portes est un groupe qui logiquement existe toujours. Il n'a pas a être ni créé, ni détruit. Les opérations que l'on peut appliquer à un de ces groupes sont: *getpgroup(2)*, pour obtenir son descripteur, *groupctl(2)*, pour insérer ou supprimer une porte du groupe, et les opérations d'envoi de messages tel que pour un groupe explicitament créé par *creatgroup(2)*.

Le système offre UidMax * PGroupMax groupes pré-définis. A chaque identificateur d'usager (uid) est attribuée une tranche de PGroupMax groupes pré-définis. Un acteur ne peut obtenir que le descripteur d'un groupe pré-défini de la tranche correspondante à son uid effectif courant (c.a.d à l'uid de sa "porte des appels système" courante). Le paramètre *name* désigne un groupe pré-défini rélativement à la tranche de groupes auxquels l'appelant a couramment accès. Ce paramètre doit donc prendre une valeur comprise entre 1 et PGroupMax.

L'accès au nom d'un groupe pré-défini par *getpgroup(2)* est ainsi intrinsèquement protégé; Cependant, rien n'empêche l'inscription de l'un de ces groupes sous un nom symbolique par *symbbind(2)*, ce qui permet de dépasser cette limitation.

Les attributs d'un groupe pré-défini sont aussi statiquement définis, c.a.d. indépendants des attributs par défaut positionnés par *gmask(2)*. Ces attributs sont les suivants:

Id-protection	L'id-protection d'un groupe pré-défini vaut (uid, 0, did); où "uid" est celui de la tranche du groupe et "did" est celui du domaine du système.
protections	la valeur des attributs de protection d'un groupe pré-défini vaut "0010101", i.e., le droit "auto-modification", voir *gmask(2)*, est public. L'attribut "auto-destruction" n'a pas de sens en ce qui concerne ces groupes.

RESULTAT

le numéro du descripteur si *getpgroup > 0*,
-1 s'il y a une erreur.

ERREURS

EPGDOFLOW	: trop de descripteurs connus de l'acteur.
EINVAL	: *name* contiennent une valeur incorrecte.

EXEMPLES

VOIR AUSSI

creatgroup(2), groupctl(2), dlgroup(2), gmask(2)

NOM

gmask – définition des attributs à donner à un groupe de portes

SYNTAXE PASCAL

function gmask (prot : integer) : integer;

SYNTAXE C

int gmask (prot)
int prot;

PARAMETRES

prot : attributs à donner aux groupes de portes à créer.

DESCRIPTION

Associés à un groupe il y a un id-protection, des attributs de protection et l'attribut "auto-destruction". Cette opération positionne les valeurs par défaut des attributs de protection et de "auto-destruction" à donner à un groupe de portes créé par *creatgroup(2)* au moment de sa création. Rappelons que l'id-protection d'un de ces groupes est l'id-protection effectif de son créateur. Les attributs d'un groupe pré-défini sont aussi pré-définits.

Le paramètre *prot* doit être un entier positif dont la forme binaire peut s'exprimer comme la suite de chifres binaires "duuggoo"; où:

- d spécifie l'attribut "auto-destruction",
- uu spécifie les droits "modification" et "auto-modification" pour les requêtes émises au nom de l'uid du groupe de portes,
- gg spécifie les droits "modification" et "auto-modification" pour les requêtes émises au nom du gid du groupe,
- oo respectivement les droits des autres usagers.

Le droit "modification" permet l'introduction et la suppression de n'importe qu'elle porte dans le groupe, et la destruction du groupe; le doit "auto-modification" ne permet que l'introduction et la suppression des portes ouvertes par l'acteur demandeur de la modification.

Si un groupe a l'attribut "auto-destruction" positionneé, il ne peut pas être explicitement détruit par *dlgroup(2);* il le sera automatiquement quand il deviendra vide.

Ces valeurs par défaut des attributs d'un groupe sont héritées de père en fils.

CONSTANTES SYMBOLIQUES

Les constantes symboliques suivantes sont disponibles:

GPAUTODL	= 1000000
OWMODIF	= 0100000
OWAUTOMODIF	= 0010000
GRMODIF	= 0001000
GRAUTOMODIF	= 0000100
OTMODIF	= 0000010
OTAUTOMODIF	= 0000001

RESULTAT

0 si l'opération se passe bien,
-1 s'il y a une erreur.

ERREURS

EINVAL : la valeur de *prot* est incorrecte.

EXEMPLES

0110100	enlève l'attribut "auto-destruction" du groupe et donne les droits de "modification" et de "auto-modification" au créateur du groupe de portes et le droit "auto-modification" aux autres usagers du même groupe d'utilisateurs; même vide, ce groupe ne peut être détruit que par *dlgroup*.
0110101	donne, en plus, le droit "auto-modification" à tous les usagers.
1110000	donne l'attribut "auto-destruction" au groupe et donne les droits de "modification" et de "auto-modification" au créateur du groupe. Ce groupe sera automatiquement détruit quand il deviendra vide.

VOIR AUSSI

creatgroup(2), groupctl(2), dlgroup(2), getpgroup(2)

NOM

groupctl – insertion, supression et test d'appartenance d'une porte à un groupe de portes

SYNTAXE PASCAL

```
function groupctl (
                        cmd : integer;
                        port : integer;
                        group : integer ) : integer;
```

SYNTAXE C

```
int groupctl ( cmd, port, group )
int cmd;
int port;
int group;
```

PARAMETRES

cmd : opération spécifique à réaliser.

port : numéro du descripteur de la porte visée.

group : numéro du descripteur du groupe visé.

DESCRIPTION

Cette fonction sert à insérer, à supprimer ou à tester l'appartenance de la porte *port* au groupe de portes *group* en fonction du paramètre *cmd*.

Les valeurs possibles de *cmd* sont les suivantes:

PORTISIN avec cette commande, *groupctl* rend 0 si la porte appartient au groupe et -1 si elle ne lui appartient pas, ou s'il y a erreur.

PORTINSERT insertion de la porte dans le groupe.

PORTREMOVE la porte est rétirée du groupe.

L'opération de test d'appartenance est publique; cependant l'insertion ou la supression d'une porte d'un groupe est soumise à des règles de protection, voir *gmask(2)*.

Une porte peut être insérée ou rétirée d'un groupe indépendament de son état (fermée ou ouverte). Les changements d'état d'une porte, successifs aux opérations *openport(2) et closeport(2)*, n'affectent pas sa qualité de membre d'un groupe de portes. Cependant, rappelons qu'une porte fermée ne peut en aucun cas recevoir un message.

Naturellement, une porte nouvellement créée n'appartient à aucun groupe, et quand elle est détruite, elle est rétirée de tous les groupes auxquels elle appartenait.

RESULTAT

0 si l'opération se passe bien (ou si la porte appartient au groupe pour *cmd = PORTISIN)*,
-1 s'il y a une erreur (ou si la porte n'appartient pas au groupe, pour *cmd = PORTISIN)*.

ERREURS

EBADPGD : un des descripteurs *port ou group* est invalide.

EBADPORT : le descripteur *port* ne désigne pas une porte.

EBADGROUP : le descripteur *group* ne désigne pas un groupe.

EINVAL : commande inconnue.

EACCES : insuffisance de droits pour exécuter l'opération.

ENOTIN : la porte *port* n'appartient pas au groupe de portes *group* (pour les cas de test d'appartenance ou suppression).

ENOMEM : la requête ne peut être déposée, par manque de ressources.

ETIMEOUT : la porte *port* ou le groupe *group* n'existent plus dans le système ou il y a des problèmes de communication.

EXEMPLES

VOIR AUSSI

creatgroup(2), dlgroup(2), gmask(2), getpgroup(2)

NOM

mkport – création d'un noeud de désignation symbolique d'une porte ou d'un groupe de portes

SYNTAXE PASCAL

```
function mkport (
                    var path: TcPath;
                    pgd: integer) : integer;
```

SYNTAXE C

```
int mkport (path, pgd);
char *path;
int pgd;
```

PARAMETRES

path : chaîne contenant le chemin d'accès au nouveau noeud de l'arbre des fichiers.

pgd : numéro du descripteur de porte ou groupe à associer à ce noeud de l'arbre.

DESCRIPTION

Création du noeud désigné par *path* en lui associant la porte/groupe dont le numéro de descripteur est *pgd.*

Les droits d'accès associés au noeud ainsi créé sont calculés à partir de la valeur de *umask* (voir *umask(2)*).

Les identifications du propriétaire et du groupe affectées au noeud créé sont celles d'utilisateur effectif et de groupe effectif de l'acteur demandeur.

RESULTAT

0 en cas de succès, -1 en cas d'erreur

ERREURS

EEXIST : le chemin d'accès existe déjà.

EFAULT : adresse hors mémoire.

EBADPGD : *pgd* est un numéro de descripteur invalide.

EIO : erreur d'entrée/sortie.

ENOENT : un composant du chemin d'accès n'existe pas.

ENOMEM : le message de requête ne peut être composé faute de ressources.

ENOTDIR : un composant du chemin d'accès n'est pas un catalogue.

EROFS : création demandée sur un volume en lecture seulement

ETIMEOUT : la réponse n'a pas été reçue assez tôt.

EXEMPLES

VOIR AUSSI

symbbind(2), umask(2)

NOM

msgdest – obtention du numéro du descripteur de la porte recéptrice d'un message

SYNTAXE PASCAL

function msgdest (box : integer) : integer;

SYNTAXE C

int msgdest (box)
int box;

PARAMETRES

box: boîte contenant le message.

DESCRIPTION

Cette fonction retourne le numéro du descripteur de la porte recéptrice de l'item du sommet du message contenu dans *box*. Ce descripteur est celui d'une porte ouverte de l'acteur si cette opération est exécutée à la suite de la réception du message. Cependant, il peut ne pas l'être si, par exemple, l'acteur vient d'exécuter l'opération *popitem* sur le même message.

RESULTAT

le numéro du descripteur si msgdest > 0,
-1 s'il y a une erreur.

ERREURS

ENOBOX	: la boîte est inexistante ou ne contient pas de message.
EPGDOFLOW	: trop de descripteurs connus de l'acteur.
EINVAL	: l'item courant n'a jamais été émis en sommet de pile.

EXEMPLES

VOIR AUSSI

msgsrc(2), popitem(2), pushitem(2)

NOM

msgsrc – obtention du numéro du descripteur de la porte émettrice d'un message

SYNTAXE PASCAL

function msgsrc (box : integer) : integer;

SYNTAXE C

int msgsrc (box)
int box;

PARAMETRES

box: boîte contenant le message.

DESCRIPTION

Cette fonction retourne le numéro du descripteur de la porte émettrice de l'item du sommet du message contenu dans *box*. Ce descripteur est celui de la porte de l'acteur qui a envoyé ce message si cette opération est exécutée à la suite de la réception du message. Cependant, il peut ne pas l'être si, par exemple, l'acteur vient d'exécuter l'opération *popitem* sur le même message.

RESULTAT

le numéro du descripteur si msgsrc > 0,
-1 s'il y a une erreur.

ERREURS

ENOBOX	: la boîte est inexistante ou ne contient pas de message.
EPGDOFLOW	: trop des descripteurs connus de l'acteur.
EINVAL	: l'item courant n'a jamais été émis en sommet de pile.

EXEMPLES

VOIR AUSSI

msgdest(2), popitem(2), pushitem(2)

NOM

openport – ouverture d'une porte

SYNTAXE PASCAL

```
function openport (
                        port : integer;
                        prio : integer ) : integer;
```

SYNTAXE C

```
int openport ( port, prio )
int port;
int prio;
```

PARAMETRES

port : numéro du descripteur de la porte à ouvrir.

prio : priorité de la porte pendant cette ouverture.

DESCRIPTION

Cette fonction ouvre la porte désignée par le descripteur *port*. Pour que la porte puisse être ouverte, il faut qu'elle existe au préalable dans l'état "fermée", et que l'id-protection effectif de l'acteur (c.a.d. celui de la "porte des appels système" courante) lui donne le droit "ouverture" sur cette porte, voir *pmask(2)*.

Une porte ouverte sert à émettre des messages par *putmsg(2)* et *putreply(2)* ainsi qu'à exécuter des appels externes par *call(2)*. Elle sert aussi à recevoir les messages qui lui sont envoyés, à condition qu'on l'associe à une étape de programmation, voir *switch(2)*, et qu'on lui associe des conditions de sélection, voir *enable(2)*, *enableseq(2)* et *enableto(2)*.

La valeur de *prio*, doit être >= 0. Plus elle est faible, plus la priorité est grande.

Elle est relative à l'acteur. La priorité absolue de la porte pendant cette ouverture est obtenue en additionnant *prio* avec la valeur "priorité maximale" définie au chargement de l'acteur. Elle conditionne le choix de la prochaine *étape de programmation* à exécuter sur le site : les messages parvenant aux portes de plus forte priorité absolue (de valeur la plus proche de 0) sont sélectionnés les premiers (à condition qu'ils satisfassent les conditions de sélection et que les portes soient associées à des étapes de programmation).
Dans tous les cas, *prio* doit être tel que la priorité absolue obtenue appartienne à [CcHighPri, CcLowPri].

RESULTAT

0 si la porte a été ouverte,
-1 s'il y a une erreur.

ERREURS

EBADPGD	: le descripteur *port* est invalide.
ENOTPCLOSED	: le descripteur *port* ne designe pas une porte fermée.
EINVAL	: priorité incorrecte parce que la valeur passée en paramètre est négative ou que son addition à la priorité maximale de l'acteur dépasse CcLowPri.
EACCES	: droit d'ouverture refusé.
ENOMEM	: la requête ne peut être déposée, par manque de ressources.
EPOFLOW	: trop de portes dans les tables du système; cette erreur ne peut se produire que suite à la migration de la porte de site.

EXEMPLES

VOIR AUSSI

creatport(2), closeport(2), dlport(2), pmask(2)

NOM

pgdcntl – modification/obtention de l'attribut "transmission" du descripteur d'une porte ou d'un groupe de portes

SYNTAXE PASCAL

```
function pgdcntl (
                    pgd : integer;
                    attr : integer ) : integer;
```

SYNTAXE C

```
int pgdcntl ( pgd , attr )
int pgd, attr;
```

PARAMETRES

pgd: numéro du descripteur visé.

attr: valeur de l'attribut "transmission" ou -1 si l'on ne veut que connaitre sa valeur ancienne.

DESCRIPTION

Attaché à chaque descripteur de porte ou groupe de portes il y a un attribut de transmission qui conditionne si le descripteur est ou non transmis aux fils de l'acteur dans le cas d'un *fexec(2)* et s'il est ou non transmis au nouveau code dans le cas d'un *exec(2)*. Remarquons que dans le cas d'un *fork(2)*, le descripteur est toujours transmis au fils de l'acteur et il conserve la valeur de son attribut de transmission.

L'opération *pgdcntl* sert à spécifier et/ou connaitre la valeur de cet attribut. Elle rend toujours comme résultat la valeur de cet attribut avant l'appel; si elle est appelée avec -1 comme valeur du paramètre *attr*, la valeur de cette attribut n'est pas modifiée.

Les valeurs possibles de cet attribut sont deux: "transmettre" ou "ne pas transmettre", i.e. PGDTR ou PGDNOTR; la valeur par défaut étant PGDTR, i.e. transmettre.

Outre la valeur -1, *attr* peut prendre une des valeurs suivantes:

PGDTR — Avec cet valeur, si le descripteur désigne une porte non ouverte par l'acteur, ou un groupe de portes, le descripteur est aussi visible du fils *(fexec)* et du noveau code *(exec)*. Si le descripteur désigne une porte ouverte, cette porte sera aussi une porte ouverte du nouveau code *(exec)* et le descripteur désignera une *nouvelle* porte ouverte chez le fils *(fexec)*.

PGDNOTR — Avec cet valeur le descripteur est toujours libéré suite aux deux appels *(exec ou fexec)*. La libération du descripteur d'une porte ouverte dans le cas d'un *exec(2)* exige la fermeture implicite de la porte et son éventuelle destruction selon la valeur de l'attribut "destruction automatique", voir *pmask(2)*.

La porte des "appels système" courante et la "porte ombilicale" courante sont une exception dans le mesure où elles sont toujours "transmises" quelque soit la valeur de l'attribut transmission.

RESULTAT

la valeur ancienne de l'attribut si *pgdcntl* >= *0*,
-1 s'il y a une erreur.

ERREURS

EBADPGD : *pgd* n'est pas le numéro d'un descripteur valide.

EINVAL : valeur d'attibut incorrecte.

EXEMPLES

VOIR AUSSI

exec(2), fork(2), fexec(2), exit(2), pgdtype(2), pmask(2)

NOM

pgdtype – obtention du type de l'entité désignée par un descripteur de porte ou groupe de portes

SYNTAXE PASCAL

function pgdtype (pgd : integer) : integer;

SYNTAXE C

int pgdtype (pgd)
int pgd;

PARAMETRES

pgd: le numéro du descripteur de l'entité visée

DESCRIPTION

Cette fonction retourne le type de l'entité désignée par le numéro de descripteur *pgd*.

Remarquons que mis à part les cas "porte ouverte" et "groupe pré-défini", le succès de cette opération ne garantit pas que l'entité désignée par *pgd* existe car, par exemple, depuis l'acquisition de ce nom par l'appelant, un autre acteur peut avoir détruit l'entité qu'il désigne.

RESULTAT

ISAOPPORT si le descripteur désigne une porte ouverte,
ISANOPPORT si le descripteur désigne une porte non ouverte par l'appelant,
ISAGROUP si le descripteur désigne un groupe créé par *creatgroup(2)*,
ISAPGROUP si le descripteur désigne un groupe pré-défini,
-1 s'il y a une erreur.

ERREURS

EBADPGD : *pgd* n'est pas un descripteur valide.

EXEMPLES

VOIR AUSSI

creatport(2), creatgroup(2), symbbind(2), getpgroup(2), pgdcntl(2)

NOM

pmask – modification des attributs à donner à une porte au moment de sa création

SYNTAXE PASCAL

function pmask (prot : integer) : integer;

SYNTAXE C

int pmask (prot)
int prot;

PARAMETRES

prot : attributs à donner aux portes à créer

DESCRIPTION

Chaque porte a un id-protection, des attributs de protection et l'attribut "destruction automatique". Cette opération définit la valeur des attributs de protection et l'attribut "destruction automatique" à donner à une porte au moment de sa création. Rappelons que l'id-protection d'une porte est l'id-protection effectif courant du créateur au moment où la création a lieu, voir *creatport(2)*.

Le paramètre *prot* doit être un entier positif dont la forme binaire peut s'exprimer comme la suite de chifres binaires "duuggoo"; où:

d	spécifie l'attribut "destruction automatique";
uu	spécifie les droits "ouverture" et "destruction" pour les requêtes émises au nom de l'uid de la porte;
gg	spécifie les droits "ouverture" et "destruction" pour les requêtes émises au nom du gid de la porte, et
oo	respectivement les droits des autres usagers.

Quand un acteur est détruit, toutes les portes qu'il avait ouvertes sont fermées; parmi ces portes, celles qui ont "1" comme valeur de l'attribut "destruction automatique" sont aussi détruites à ce moment là.

De même, quand un acteur exécute l'appel *exec(2)*, les portes ouvertes qui ne sont pas transmises, voir *pgdcntl(2)*, sont aussi fermées; elles seront aussi détruites, comme dans le cas de la destruction de l'acteur, si leur attribut "destruction automatique" vaut "1".

Les portes qui servent à rendre des services dynamiquement reconfigurables, c.a.d. qui peuvent être successivement ouvertes par des acteurs différents, doivent avoir "0" comme valeur de l'attribut "destruction automatique".

La valeur par défaut des attributs de création d'une porte est héritée de père en fils.

CONSTANTES SYMBOLIQUES

Les constantes symboliques suivantes sont disponibles:

PRTAUTODL	= 1000000
OWOPEN	= 0100000
OWDEL	= 0010000
GROPEN	= 0001000
GRDEL	= 0000100
OTOPEN	= 0000010
OTDEL	= 0000001

RESULTAT

0 si l'opération se passe bien,
-1 s'il y a une erreur.

ERREURS

EINVAL : la valeur de *prot* est incorrecte.

EXEMPLES

1110000

donne l'attribut "à détruire automatiquement" à la porte et les droits "d'ouverture" et de "destruction" au créateur de la porte.

0111000

donne les droits "d'ouverture" et de "destruction" au créateur de la porte et le droit "ouverture" aux autres usagers du même groupe d'usagers; la porte ne sera pas automatiquement détruite dans les situations citées au-dessus.

VOIR AUSSI

creatport(2), openport(2), closeport(2), dlport(2)

NOM

putfwd - retransmission d'un message

SYNTAXE PASCAL

```
function putfwd (
                box: integer;
                destport: integer) : integer;
```

SYNTAXE C

```
int putfwd ( box, destport )
int box;
int destport;
```

PARAMETRES

box : la boîte contenant le message.

destport : numéro du descripteur du récepteur du message (porte ou groupe).

DESCRIPTION

Cette fonction permet de retransmettre le message reçu dans *box* vers la (ou les) porte(s) désignée(s) par le descripteur *destport*. La porte émettrice initiale du message n'est pas affectée par cette opération. Le message contenu dans *box* est placé dans la liste des messages émis à la fin de l'étape de programmation. Au retour, *box* est vide.

RESULTAT

0 si la requête a été déposée,
-1 si erreur.

ERREURS

EBADBOX : la boîte est inexistante ou ne contient pas de message à retransmettre.

ENOMEM : la requête ne peut être déposée, par manque de ressources

EBADPGD : *destport* est inconnu de l'acteur

EXEMPLES

VOIR AUSSI

putmsg(2), putreply(2), call(2)

NOM

putmsg - dépôt d'un message

SYNTAXE PASCAL

```
function putmsg (
                        box: integer;
                        srcport: integer;
                        destport: integer;
                        mode: integer) : integer;
```

SYNTAXE C

```
int putmsg ( box, srcport, destport, mode )
int box;
int srcport;
int destport;
int mode;
```

PARAMETRES

box : la boîte contenant le message.

srcport : numéro du descripteur de la porte émettrice du message.

destport : numéro du descripteur du récepteur du message (porte ou groupe).

mode : mode d'adressage.

DESCRIPTION

Cette fonction place le message contenu dans *box* dans la liste des messages à envoyer à la fin de l'étape de programmation. Le message sera émis de *srcport* vers la (ou les) porte(s) désignée(s) par *destport* et *mode*. Au retour, *box* est vide.

Si *destport* désigne une porte, *mode* est ignoré. Si *destport* désigne un groupe, les valeurs de *mode* prises en considération sont les suivantes:

RESTFUNC	Le message sera émis vers une porte du groupe locale au sous-réseau dans lequel s'exécute l'acteur appelant.
FUNC	Le message sera émis vers une porte du groupe.
RESTBROAD	Une copie du message sera émise vers chaque porte du groupe locale au sous-réseau dans lequel s'exécute l'acteur appelant.
BROAD	Une copie du message sera émise vers chaque porte du groupe.

Le mode PORTTOPORT n'est utilisable que si *destport* désigne une porte.

RESULTAT

0 si la requête a été déposée,
-1 si erreur.

ERREURS

EBADBOX	: la boîte est inexistante ou ne contient pas de message
ENOMEM	: la requête ne peut être déposée, par manque de ressources
ENOTPOPEN	: *srcport* ne désigne pas une porte ouverte de l'acteur appelant
EBADPGD	: *destport* est inconnu de l'acteur
EBADMODE	: *mode* n'existe pas.

EXEMPLES

VOIR AUSSI

putfwd(2), putreply(2)

NOM

putreply - dépôt d'un message pour renvoi à son émetteur

SYNTAXE PASCAL

function putreply (box : integer) : integer;

SYNTAXE C

int putreply (box)
int box;

PARAMETRES

box : boîte contenant le message

DESCRIPTION

Cette fonction place le message contenu dans *box* dans la liste des messages à envoyer à la fin de l'étape de programmation. Le message sera retourné depuis sa porte réceptrice vers sa porte émettrice initiale. Au retour, la boîte est vide.

RESULTAT

0 si l'opération est effectuée,
-1 si erreur.

ERREURS

EBADBOX	: la boîte est inexistante ou ne contient pas de message.
ENOMEM	: la requête ne peut être déposée, par manque de ressources.
ENOTPOPEN	: la porte sur laquelle le message avait été reçue n'est plus ouverte par l'acteur.

EXEMPLES

VOIR AUSSI

putmsg(2), call(2), putfwd(2)

NOM

relpgd – libération d'un descripteur de porte ou de groupe de portes

SYNTAXE PASCAL

function relpgd (pgd : integer) : integer;

SYNTAXE C

int relpgd (pgd)
int pgd;

PARAMETRES

pgd: numéro du descripteur à liberer.

DESCRIPTION

Cette fonction sert à libérer un descripteur d'une porte ou d'un groupe de portes visible par l'acteur. C'est une erreur de l'appliquer à une porte ouverte par l'appelant.

RESULTAT

0 si l'opération se passe bien,
-1 s'il y a une erreur.

ERREURS

EBADPGD : *pgd* est un numéro de descripteur invalide.

ENOTPCLOSED : *pgd* désigne une porte ouverte par l'acteur.

EXEMPLES

VOIR AUSSI

pgdtype(2)

NOM

symbbind – obtention/association d'un descripteur de porte ou groupe avec un nom symbolique (path name)

SYNTAXE PASCAL

```
function symbbind (
                        cmd : integer;
                        VAR path : TcPath;
                        pgd : integer ) : integer;
```

SYNTAXE C

```
int symbbind ( cmd, path, pgd )
int cmd ;
char *path ;
int pgd ;
```

PARAMETRES

cmd: opération spécifique à réaliser (GETNAME, SETNAME).

path: le nom symbolique (en entrée).

pgd: le numéro du descripteur pour *cmd = SETNAME.*

DESCRIPTION

Cette fonction sert, soit à associer la porte ou le groupe de portes désigné par le descripteur *pgd* avec le nom symbolique *path,* soit à obtenir un numéro de descripteur désignant la porte ou le groupe qui a été associé au préalable à *path.*

Les valeurs possibles de *cmd* sont les suivantes:

GETNAME — avec cette commande, *symbbind* rend en sortie le descripteur de la porte ou du groupe de portes désigné symboliquement par *path.*

Cette acquisition de nom ne prend effet que si l'id-protection effectif du demandeur (c.a.d. celui de sa "porte des appels système" courante) a le droit "r" (lecture) sur le noeud *path* et qu'une association de nom à ce noeud ait été faite au préalable.

SETNAME — avec cette commande, *symbbind* associe la porte ou le groupe de portes désigné par *pgd* avec le nom symbolique *path,* en écrasant l'association (éventuellement) faite au préalable.

Cette association d'une porte ou d'un groupe de portes avec un nom symbolique ne prend effet que si l'id-protection du demandeur a le droit "w" (écriture) sur le noeud *path.*

Dans les deux cas, le noeud *path* doit exister au préalable et être du type nom symbolique de porte ou groupe de portes CHPORTE. Pour le créer on doit utiliser *mkport(2).*

La fonction *symbbind* ne fait aucun contrôle ni sur le type de l'entité désignée par *pgd,* ou enrégistrée sous *path,* ni sur l'existance de cette entité.

RESULTAT

0 si l'opération se passe bien pour SETNAME,
le numéro du descripteur pour GETNAME si symbbind > 0,
-1 s'il y a une erreur.

ERREURS

EBADPGD : le descripteur *pgd* est invalide (pour SETNAME).

EINVAL : commande inconnue.

EPGDOFLOW : trop de descripteurs connus de l'acteur (pour GETNAME).

EACCES	: droits insuffisants.
ENOTPORT	: la feuille désignée n'est pas de type porte.
ENOENT	: le fichier n'éxiste pas.

EXEMPLES

VOIR AUSSI

mkport(2)

NOM

syscallport – obtention/modification du numéro du descripteur de la "porte des appels système" courante

SYNTAXE PASCAL

function syscallport (port : integer) : integer ;

SYNTAXE C

int syscallport (port)
int port;

PARAMETRES

port : numéro du descripteur de la nouvelle "porte des appels système" ou -1 si l'ancien ne doit pas être modifié

DESCRIPTION

Cette fonction retourne toujours le numéro du descripteur de la "porte des appels système" au moment de l'appel; en plus, si *port* est <> de -1, le numéro passé en paramètre sera pris comme numéro de descripteur de la nouvelle "porte des appels système".

En d'autres termes, que la porte prise comme "porte des appels système" change ou non, la valeur retournée est toujours le numéro du descripteur de cette porte avant l'appel.

Si l'on change la "porte des appels système", cette nouvelle porte doit être une porte ouverte de l'acteur.

Un acteur ne sait pas quel a été son mode de naissance (par exec, fork ou fexec) donc, en général, il ne sait pas quelle est celle de ses portes ouvertes qui est courament la "porte des appels système". Il ne peut le savoir qu'en utilisant cette fonction.

L'id-protection de la "porte des appels système" est l'id-protection effectif courant.

RESULTAT

le numéro du descripteur de la "porte des appels système" avant l'appel si *syscallport > 0,*
-1 s'il y a une erreur.

ERREURS

EBADPGD	: le descripteur *port* est invalide.
ENOTPOPEN	: la porte *port* n'est pas une porte ouverte de l'appelant.

EXEMPLES

VOIR AUSSI

geteuid(2), getegid(2)

NOM

ubport – obtention/modification du numéro du descripteur de la "porte ombilicale" (umbilical port) courante

SYNTAXE PASCAL

```
function ubport ( port : integer ) : integer ;
```

SYNTAXE C

```
int ubport ( port )
int port;
```

PARAMETRES

port : numéro du descripteur de la nouvelle "porte ombilicale" ou -1 si l'ancien ne doit pas être modifié

DESCRIPTION

Cette opération permet de spécifier quelle est la porte affectée à la communication avec les fils, la "porte ombilicale". Elle retourne toujours le numéro du descripteur qui était affecté à cet effet avant l'appel. Si on l'appelle avec -1 en paramètre, la valeur ancienne n'est pas modifiée.

Quand un acteur est créé, le système lui transmet le nom de la "porte ombilicale" du père, c.a.d., le nom de la porte qui couramment le père avait affecté à cette fonction. D'autre part, au même moment, le père reçoit le nom de la porte du fils qui a le même numéro de descripteur que sa "porte ombilicale" (c.a.d. le nom de la "porte ombilicale" initiale du fils). Ces portes jouent donc le rôle de "liaison ombilicale", ou "cordon ombilical", entre le père et le fils.

Un acteur obtient un descripteur désignant la porte qui était la "porte ombilicale" de son père au moment de sa création, en appelant *getparent(2)*. Ainsi, le fils peut s'adresser à une porte ouverte du père.

A la naissance, la porte qui a dans le fils le même numéro de descripteur que la "porte ombilicale" du père est la "porte ombilicale" courante du fils. Le père peut aussi obtenir un descripteur de la "porte ombilicale" du dernier fils créé en appelant *getchild(2)*. Ainsi, le père peut s'adresser à une porte ouverte du fils.

RESULTAT

le numéro du descripteur de la "porte ombilicale" avant l'appel si *ubport* > *0*,
-1 s'il y a une erreur.

ERREURS

EBADPGD	: le descripteur *port* est invalide.
ENOTPOPEN	: la porte *port* n'est pas une porte ouverte de l'appelant.

EXEMPLES

VOIR AUSSI

getparent(2), getchild(2)

VU :
Le président de la Thèse :

VU :
Le Directeur de la Thèse :

VU et APPROUVE
RENNES, le
Le Directeur de l'U. E. R.

D^t UR/86/S/11 n° 58
VU pour autorisation de soutenance
RENNES, le 18 NOV. 1986
Le Président de l'Université de RENNES I,

J. P. CURTES

RESUME

Cette thèse contribue à l'étude des problèmes soulevés par la répartition en matière de désignation, traduction de noms et édition de liens. Elle propose une hiérarchie de couches de désignation des portes, groupes de portes, groupes de processus et fichiers, spécialement adaptée aux besoins d'un système réparti basé sur la communication par messages: CHORUS.

Une importance spéciale est donnée par cette proposition au problème de l'édition de liens, chargement, contrôle et reconfiguration d'applications et de services répartis construits sur des réseaux de processus communicants. En particulier, le rôle des groupes de portes et de la diffusion de messages, pour l'édition de liens et le contrôle d'applications réparties, est mis en évidence.

D'autre part, la désignation symbolique des portes et des groupes de portes est introduite par une extension du type des noeuds supportés par l'arbre de fichiers. Cette extension, et un protocole standard de redirection symbolique de requêtes, permettant l'interprétation répartie des noms symboliques, sont simultanément à la base de la construction du graphe réparti de désignation symbolique et du système de gestion de fichiers répartis de CHORUS.

Mots clés: systèmes d'exploitation, répartition, désignation, traduction de noms, édition de liens, groupes de portes et de processus, reconfiguration d'applications réparties, graphe réparti de désignation symbolique

ISBN 2 - 7261 - 0465 - 7