‘‘UNIX et la répartition : retour à la simplicité originelle ?’’

François Armand, Michel Gien, Frédéric Herrmann, Marc Rozier

Chorus systèmes
6, avenue Gustave Eiffel, F−78182, Saint-Quentin-en-Yvelines (France)
Tel: +33 1 30 57 00 22, Fax: +33 1 30 57 00 66, E-mail: fa@chorus.fr

1. Introduction

La nécessité d’appréhender les systèmes distribués d’une manière générale impose de structurer les systèmes d’exploitation d’une manière beaucoup plus modulaire qu’ils ne le sont actuellement. La répartition conduit à fournir des mécanismes permettant la reconfiguration dynamique pour que le système puisse s’adapter aux diverses configurations que l’on trouve en environnement réparti. Une telle "restructuration" appliquée aux noyau UNIX provoque une "révolution" semblable à celle que le système UNIX a provoqué sur les systèmes d’exploitation dans les années 70 (extraire du système toutes les fonctions qui peuvent être réalisées en dehors, telles les méthodes d’accès aux fichiers, les interpréteurs de commande ou les fonctions d’administration système).

L’architecture CHORUS® est conçue pour construire cette nouvelle génération de systèmes d’exploitation ouverts et distribués. Les systèmes CHORUS présentent les caractéristiques principales suivantes :

– un noyau minimum, intégrant au niveau le plus bas communication et exécution, fournit des services génériques utilisés par un ensemble de serveurs coopérant au sein de sous-systèmes pour fournir des interfaces standards (l’interface UNIX† est aujourd’hui disponible; des systèmes objets sont à l’étude),

– les services temps-réels fournis par le noyau et accessibles aux programmeurs systèmes sont intégrés aux différents niveaux d’interface.

– l’architecture modulaire permet une (re)configuration dynamique du système et des applications sur une large gamme de matériels et de topologies de réseaux.

CHORUS−V3 est la version courante du système développée par Chorus systèmes. Des versions précédentes ont été développées dans le cadre du projet CHORUS à l’INRIA entre 1979 et 1986. CHORUS se compare aux projets suivants : V-system pour le noyau à communication par messages[Cher88] , Mach [Rash87] et [Li86] pour la mémoire virtuelle distribuée, Topaz [McJo88] et Mach [Acce86] pour les ‘‘threads’’, Amoeba [Mull87] pour la désignation globale, et UNIX version 9 [Pres86, Wein86] des Bell Laboratories pour la désignation uniforme des fichiers.

CHORUS−V3 est écrit en C++ (et en C). Il tourne actuellement sur des machines à base de 680x0, de 88000 et de 80386, tant sur des réseaux de stations que sur des configurations multi-processeurs.

Cet article décrit l’architecture et l’implémentation des fonctions du noyau UNIX à partir des services et concepts de base du noyau CHORUS. Il décrit plus particulièrement les bénéfices qui en résultent aussi bien pour les utilisateurs que pour les concepteurs de systèmes, en faisant le parallèle avec les services fournis traditionellement par UNIX.

® CHORUS est une marque déposée de Chorus systèmes

† UNIX est une marque déposée d’AT&T

Article publié dans les actes de la convention UNIX’90

Le chapitre suivant résume les abstractions et services de base offerts par le noyau CHORUS ; le lecteur en trouvera une description plus complète dans [Rozi88]. Le chapitre 3 décrit la structure du sous-système UNIX de CHORUS, constitué de serveurs indépendants qui coopèrent, illustrant comment manipuler les services offerts par le noyau CHORUS. Les extensions apportées à l’interface traditionnelle d’UNIX sont aussi décrites. Enfin quelques exemples d’utilisation de CHORUS sont donnés, en faisant le parallèle avec des utilisations d’UNIX.

2. L’Architecture CHORUS

2.1 Description macroscopique

Un système CHORUS est composé d’un **Noyau** de petite taille et d’un ensemble de **Serveurs Systèmes**. Ces serveurs coopèrent au sein de **Sous-Systèmes** (par exemple le sous-systèmes UNIX), pour fournir un ensemble cohérent de services et d’interfaces aux utilisateurs (Figure 1).

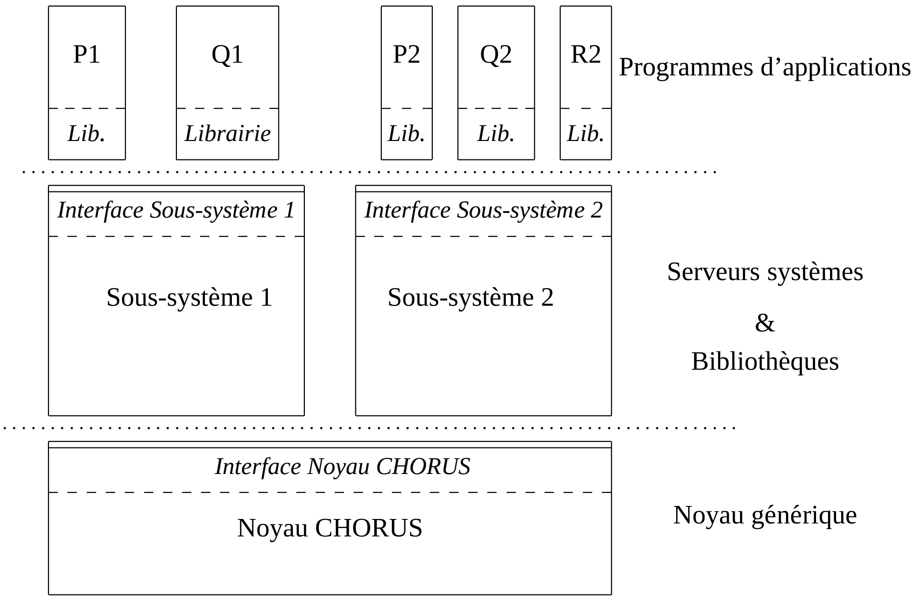

Figure 1. – L’ Architecture CHORUS

Le noyau CHORUS (Figure 2) joue un double rôle :

1. Services locaux :
   La gestion des ressources physiques d’un ‘‘processeur’’ (en fait d’un *site*, voir la définition plus loin), est assurée par trois composants distincts:
   - *L’Exécutif temps-réel multi-tâches* gère l’accès au processeur : il fournit les primitives de synchronisation de bas niveau et offre un ordonnancement préemptif basé sur des priorités fixes, modifiables dynamiquement.
   - *Le Gestionnaire de Mémoire (Virtuelle)* gère la mémoire locale,
   - *Le Superviseur* permet aux composants des sous-systèmes implantés hors du noyau de contrôler les évènements matériels : interruptions, déroutements, exceptions.
2. Services globaux :
   Le Gestionnaire d’IPC (Inter Process Communication) fournit les services de communication, en assurant la transmission de *messages* de manière uniforme et transparente à la répartition, c’est à dire indépendamment de la localisation des correspondants. Il s’appuie sur un serveur externe (Gestionnaire de réseau) pour la réalisation des protocoles utilisés.

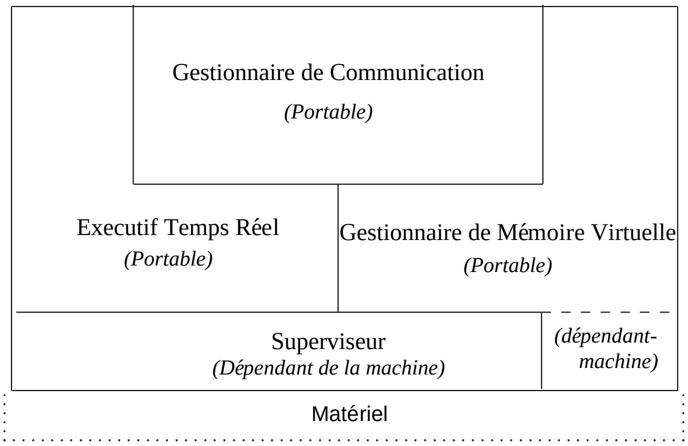

Figure 2. – Le Noyau CHORUS

2.2 Abstractions de base offertes par le noyau CHORUS

Un système CHORUS s’exécute sur un ensemble de *sites* (machines ou cartes), interconnectés par un *système de communication* (réseau ou bus). Un **site** est constitué de ressources physiques fortement couplées : un ou plusieurs processeurs, de la mémoire, et éventuellement des périphériques. Il y a un Noyau CHORUS par site.

Un **acteur** représente l’unité de répartition et d’allocation de ressources. Un *acteur* définit un espace d’adressage protégé. Une ou plusieurs **activités** (*lightweight processes* ou *threads*) peuvent s’exécuter au sein d’un même acteur (donc du même espace d’adressage). Un espace d’adressage est composé d’un espace *utilisateur* et d’un espace *système*. Sur un site donné, l’espace système est commun à tous les acteurs de ce site, mais son accès est réservé au mode d’exécution privilégié. Chaque acteur ayant son propre espace d’adressage utilisateur, un acteur définit une machine virtuelle protégée.

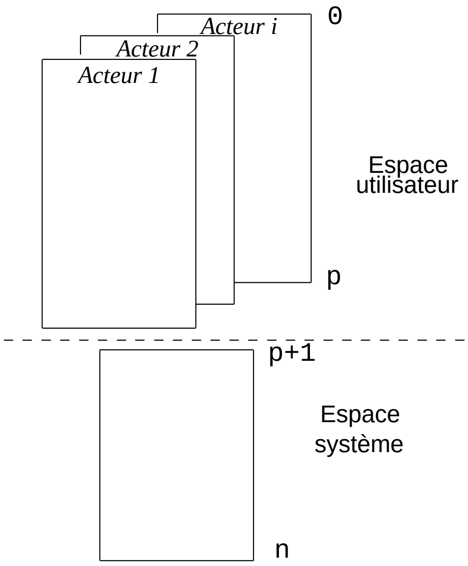

Figure 3. – Espace d’adressage d’un acteur

Un acteur est lié à un site, et ses activités s’exécutent sur ce site. Plusieurs acteurs peuvent s’exécuter simultanément sur un site.

L’activité est l’unité d’exécution dans un système CHORUS. Elle est caractérisée par un *contexte* correspondant à l’état du processeur (registres, compteur ordinal, pointeur de pile, etc...). Une activité est liée à un acteur et un seul. Toutes les activités d’un acteur partagent les ressources de cet acteur et de celui-là seulement. Les activités sont ordonnancées par le noyau comme des entités indépendantes. En particulier, les activités d’un acteur peuvent s’exécuter en parallèle sur les différents processeurs d’un site multiprocesseur. L’ordonnancement des activités est préemptif, et basé sur leur priorité (fixe).

Les activités d’un même acteur peuvent communiquer en utilisant la mémoire de cet acteur comme mémoire partagée. CHORUS offre également un mécanisme de communication par message (appelé IPC dans la suite de ce document) qui permet à toute activité de communiquer et de se synchroniser avec n’importe quelle autre activité s’exécutant sur n’importe quel autre site. Les échanges de messages peuvent être asynchrones ou du type demande/réponse (appelé Remote Procedure Call ou RPC dans la suite de ce document). La caractéristique principale de l’IPC CHORUS est sa transparence vis à vis de la localisation des activités : l’interface de communication est identique que les messages soient échangés entre activités d’un même acteur, entre activités d’acteurs différents du même site ou entre activités d’acteurs différents résidant sur des sites différents.

Un **message** est composé d’un *corps* optionnel et d’une *annexe* également optionnelle. L’annexe et le corps sont tous deux une suite d’octets continue et non typée. Le couplage étroit entre Gestionnaire de Communication et Gestionnaire de Mémoire Virtuelle permet d’éviter au maximum les recopies physiques d’information lors des échanges de messages.

Les messages ne sont pas adressés directement aux activités, mais à des entités intermédiaires appelées **portes**.

Une **porte** est une adresse logique à laquelle des messages peuvent être envoyés, et une file d’attente sur laquelle ils seront reçus. Une porte est attachée à un acteur, et non à une activité. Une porte est donc une ressource partagée par toutes les activités d’un même acteur. Une porte ne peut être attachée qu’à un seul acteur à un instant donné, mais peut être attachée sucessivement à plusieurs acteurs. Une porte peut ainsi *migrer* d’un acteur à un autre. Toutes les activités d’un acteur (et elles seules) peuvent consommer les messages reçus sur les portes de cet acteur. Par contre, il suffit à une activité de connaître le nom d’une porte pour pouvoir lui adresser un message.

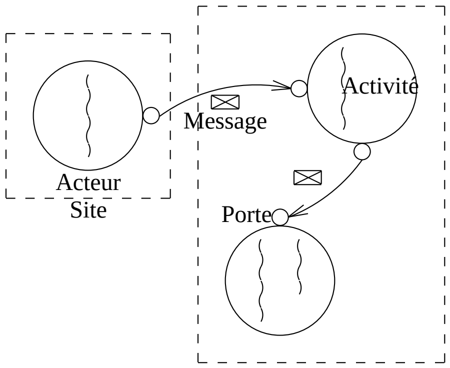

Figure 4. − Abstractions de base du Noyau CHORUS

La notion de porte fournit une mécanique élémentaire pour la reconfiguration dynamique : une activité "cliente", adressant une requête à un acteur "serveur" au travers d’une porte, n’est pas affectée par la migration de cette porte vers un autre acteur, qui peut ainsi "reprendre" le service d’une manière transparent pour le client.

Les portes peuvent être dynamiquement regroupées en ensembles logiques appelés **groupes** de portes. Les *groupes* enrichissent les mécanismes d’adressage pour l’émission de message. Ils offrent des mécanismes de *diffusion* permettant à une activité de communiquer directement avec un groupe d’activités au sein d’acteurs différents. Ils offrent aussi un mode d’adressage dit *fonctionnel* permettant à une activité de communiquer avec une autre choisie parmi un groupe d’activités (souvent fournissant des services équivalents).

Les portes sont désignées par des identificateurs uniques globaux (appelés UI par la suite). Un **UI** est unique dans un système CHORUS. Le Noyau CHORUS implémente un mécanisme de localisation, permettant ainsi aux activités d’utiliser ces noms sans se soucier de la localisation réelle des entités ainsi désignées. Les *UI* peuvent être librement échangés entre acteurs.

Les noms globaux pour les autres types d’objets sont basés sur les *UI*, mais contiennent d’autres informations, comme par exemple des attributs de protection. Ces noms sont appelés **capacités**. Une *capacité* est composée d’un *UI* et d’une structure additionnelle : la *clé*. Quand les objets sont des objets du noyau (par exemple un acteur), l’UI est alors le nom global de l’objet et la clé est seulemnt une clé de protection. Quand un objet est géré par un serveur externe au noyau CHORUS (par exemple un fichier), l’UI est alors le nom global d’une porte du serveur et la signification de la clé est définie par le serveur lui-même. Généralement, la clé identifie l’objet dans le serveur et contient un attribut de protection. Les capacités comme les UI peuvent être librement échangées entre acteurs.

2.3 Mémoire Virtuelle

Le Gestionnaire de mémoire CHORUS gère des espaces d’adressages séparés (si le matériel le permet) associés aux acteurs, appelés **contextes**. Il offre des services efficaces et complets pour le transfert de données entre *contexte* et mémoire secondaire. Les mécanismes sont adaptés à des besoins divers tels que l’IPC, l’accès aux fichiers (par read/write aussi bien que par ‘‘mapping’’), le partage de mémoire entre contextes ou la duplication de contexte.

Un *contexte* est composé d’un ensemble de **régions** disjointes, qui composent les parties valides du *contexte*.

Les régions permettent (généralement) d’accéder (par mapping) à des objets stockés en mémoire secondaire appelés **segments**. Les segments sont gérés à l’extérieur du noyau CHORUS par des serveurs appelés **Mappeurs**. Les mappeurs gèrent l’implantation des segments ainsi que leur protection et leur désignation.

2.4 Le Superviseur

Pour permettre aux acteurs systèmes de gérer les évènements matériels comme les interruptions ou les traps, le noyau CHORUS offre les services suivants :

Les activités s’exécutant en mode système peuvent attacher des ‘‘handlers’’ (procédures situées dans l’espace d’adressage de leur acteur) à des interruptions matérielles. Quand une interruption se produit, ces handlers sont exécutés. Plusieurs handlers peuvent être simultanément connectés à une même interruption, des mécanismes de contrôle permettent d’ordonner ou d’arrêter leur exécution. Ces handlers d’interruption peuvent communiquer avec d’autres activités en utilisant les primitives de synchronisation offertes par le noyau ou l’IPC asynchrone.

Les acteurs sytèmes peuvent aussi connecter des procédures aux déroutements matériels (traps). On peut connecter une procédure ou un tableau de procédures (dans ce cas, la procédure réellement invoquée est spécifiée par un ‘‘numéro’’ de service stocké dans un registre).

Enfin, une porte d’exception ou une procédure d’exception peuvent être associées à un acteur, permettant à un sous-système de réagir aux erreurs survenant dans d’autres acteurs.

3. Le Sous-Système UNIX

3.1 Structure

Les services UNIX peuvent être logiquement partitionés en plusieurs classes suivant le type des ressources gérées : processus, fichiers, périphériques, tubes, sockets. L’architecture du sous-système UNIX de CHORUS, basée sur une définition rigoureuse des interactions entre ces différentes classes de services, offre une structure réellement modulaire.

Le sous-système UNIX est composé d’un ensemble de serveurs systèmes qui s’exécutent sur le Noyau CHORUS. Chaque type de ressource système (processus, fichier...) est géré par un serveur système dédié. Les interactions entre ces serveurs sont basées sur l’IPC CHORUS qui force la définition d’interfaces claires et précises.

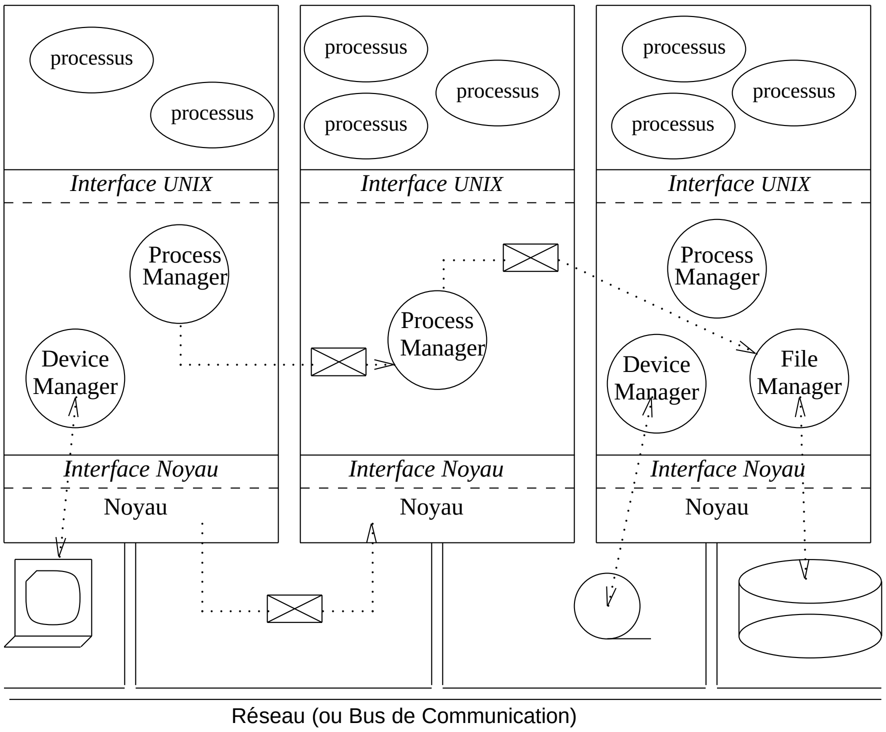

Figure 5. – UNIX Modulaire

Les différents serveurs qui composent un sous-système UNIX sont :

– **Le Gestionnaire de Processus (Process Manager ou PM)** offre les services relatifs à la manipulation des processus (création et destruction de processus, signaux, etc.). Les services UNIX ont été enrichis pour offrir l’accès transparent aux ressources réparties. Les PM des différents sites d’un réseau CHORUS coopèrent pour fournir des services répartis (tels que des signaux distants, ou des exécutions distantes).
Le PM gère le contexte système des processus. Quand le PM ne peut pas fournir un service UNIX (invoqué par un appel système UNIX) par lui-même, il fait appel au(x) serveur(s) approprié(s).

– **Le Gestionnaire de Fichiers (File Manager ou FM)** offre les services de gestion de fichiers. La version courante est compatible avec System V Rel 3.2 tant pour les services offerts que pour la structure du disque. Une version compatible avec UNIX BSD 4.3 est en cours de développement.
Le FM joue aussi le rôle d’un *mappeur externe* CHORUS, participant à la gestion de la mémoire virtuelle répartie, en répondant aux requêtes de chargement/déchargement de pages émises par la gestion de mémoire virtuelle du Noyau CHORUS.

– **Le Gestionnaire de Périphériques (Device Manager Manager ou DM)** gère les lignes asynchrones, les écrans bitmap, les pseudo-terminaux (pseudo-ttys), etc... Il implante les ‘‘*line disciplines*’’ UNIX. Plusieurs DM peuvent s’exécuter en parallèle sur un site où sont connectés plusieurs types de périphériques.

– **Le Gestionnaire de Tubes (Pipe Manager ou PiM)** implante la gestion et la synchronisation des tubes UNIX. Les requêtes d’ouvertures des tubes nommés, reçues par les Gestionnaires de Fichiers sont transmises aux Gestionnaires de Tubes. Il peut y avoir un Gestionnaire de Tubes par site, Cela permet notamment de réduire le trafic réseau quand les tubes sont utilisés sur des stations sans disque.

– **Le Gestionnaire de Sockets (Socket Manager ou SM)** offre les services sockets compatibles avec BSD 4.3 fournissant l’accès aux protocoles TCP/IP.

Les serveurs systèmes peuvent s’exécuter en *espace utilisateur* ou en *espace système*. Ceux qui doivent attacher leurs procédures aux exceptions (comme le PM) ou exécuter des instructions privilégiées (comme des instructions d’entrée/sortie) s’exécutent en espace système. Charger un serveur en espace système, permet des gains de performance en évitant des changements de contexte mémoire quand le serveur est invoqué.

3.2 Extensions Fonctionnelles

Afin d’assurer une portabilité complète des applications utilisateur, l’interface offerte par le sous-système UNIX sur une machine donnée, peut être rendue compatible (au niveau binaire exécutable) avec un système UNIX classique pris comme référence (par exemple avec System V Rel 3.2 sur un AT/386). De plus, les pilotes de périphériques UNIX peuvent être intégrés dans un serveur UNIX moyennant un effort minimum.

Des extensions aux services UNIX traditionnels ont été apportées, essentiellement par extension transparente de ces services à la répartition. Elles résultent directement des services fournis par le noyau CHORUS et de l’architecture modulaire du sous-système UNIX.

3.2.1 Extensions du Système de Fichiers

Les services de désignation offerts par UNIX ont été enrichis pour permettre la désignation de services accessibles par des Portes.

Des nœuds de type *Porte Symbolique* (nouveau type de fichier UNIX) peuvent être crées dans une arborescence UNIX. Ils associent un nom de nœud à un identificateur unique (UI) de porte. Ce mécanisme est similaire à celui utilisé pour la désignation des périphériques UNIX. Quand le nom d’un nœud de type "Porte" est trouvé au cours de l’analyse d’un chemin d’accès, la requête correspondante est transmise à la porte associée. La requête est préalablement marquée avec l’état courant de l’analyse du chemin d’accès.

Des serveurs utilisateur comme des serveurs système peuvent être désignés par de tels noms de portes symboliques, permettant ainsi d’étendre dynamiquement le système. Ce mécanisme est en particulier utilisé pour interconnecter les systèmes de fichiers d’un réseau de machines CHORUS et ainsi fournir un espace global de désignation. Par exemple, dans la Figure 6, ‘‘pipo’’ et ‘‘piano’’ sont des noms de portes symboliques.

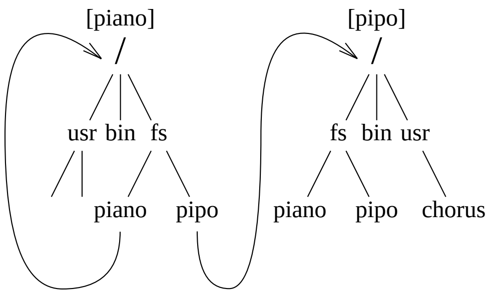

Figure 6. – Interconnexion d’arborescences de fichiers

3.2.2 Extensions de la Gestion des Processus

Les extensions suivantes ont été apportées à la gestion des processus:

- La création (`fork(2)`) ou l’exécution (`exec(2)`) de processus peut être invoquée sur des sites distants. Un attribut ‘‘site de création’’ a été ajouté au contexte système des processus UNIX. Cet attribut est hérité lors des appels systèmes `fork(2)` et `exec(2)`. Il peut être positionné au moyen d’un nouvel appel système : `csite (SiteId)`. Lors d’un `fork(2)`, le processus fils sera créé sur le site désigné par `SiteId`. Lors d’un `exec(2)`, le processus commencera l’exécution du nouveau programme sur le site désigné par `SiteId`.

Créer ou migrer (lors de l’appel `exec(2)`) des processus sur n’importe quel site nécessite que les identificateurs de processus soient uniques dans tout le système distribué. Les identificateurs de processus (PID) utilisés par les serveurs UNIX sont codés sur 32 bits. Les processus UNIX peuvent manipuler soit des PID de 16 bits (pour des raisons de compatibilité binaire), soit des PID de 32 bits. Dans le dernier cas, il peuvent émettre des signaux vers des processus distants. Un nouvel appel système : `pcntl (LONGPID)` positionne un indicateur du contexte système d’un processus, qui lui permet de manipuler des identificateurs de processus sur 32 bits.

- Lors d’un `exec(2)`, un processus peut dynamiquement être chargé en espace système, à condition que les régions dont il a besoin pour son code et ses données soient libres. Un tel processus s’exécutera alors en mode privilégié lui permettant ainsi d’exécuter des instructions d’entrée/sortie.
- Les processus peuvent augmenter ou diminuer leur priorité, permettant ainsi à des applications temps-réel de s’exécuter dans un environnement UNIX. (cf 4.4)

3.2.3 Autres Extensions

Les services offerts aux processus UNIX sont tout naturellement complétés par quelques uns des services offerts par le Noyau CHORUS comme l’IPC, la Mémoire Virtuelle et les Activités. Ces services ne sont pas fournis directement par le noyau CHORUS, mais plutôt par le Gestionnaire de Processus UNIX (PM), de manière à rendre homogène l’ensemble des services fournis et à éliminer les incohérences potentielles avec la "sémantique" UNIX. Par exemple, si un processus UNIX pouvait créer une *activité* en invoquant directement le Noyau CHORUS en ‘‘contournant’’ le Gestionnaire de Processus, cette activité ne pourrait pas invoquer d’appels systèmes UNIX correctement.

En conséquence, quelques services offerts par le Noyau CHORUS ne sont pas accessibles aux processus UNIX (par exemple les services de création ou de destruction d’acteurs), et quelques restrictions et contrôles sont effectués : par exemple, un processus ne peut pas créer d’activités dans un autre processus ; dans l’appel système `portMigrate` l’identificateur de processus (pid) doit être utilisé et non l’Identificateur Unique (UI) d’un acteur CHORUS.

Pour distinguer clairement les deux niveaux d’interface, les primitives UNIX permettant d’accéder aux services CHORUS ont été préfixées par ‘‘u_’’ (par exemple : `u_portCreate` au lieu de `portCreate`).

3.2.3.1 Services Liés à la Mémoire Virtuelle

Les processus UNIX peuvent utiliser les services du Noyau CHORUS pour créer des régions, ‘‘mapper’’ des segments dans des régions, partager des régions, etc. Ils peuvent ainsi avoir accès à la mémoire physique (par exemple : accès à la mémoire d’un écran graphique bitmap).

3.2.3.2 Inter Process Communication (IPC)

Les processus UNIX peuvent créer des portes, insérer des portes dans des groupes et émettre et recevoir des messages. Ils peuvent faire migrer des portes d’un processus à un autre. Les mécanismes de l’IPC CHORUS leur permettent de communiquer de manière transparente à travers le réseau. Les applications peuvent ainsi être testées sur une seule machine puis réparties sur le réseau sans qu’aucune modification soit nécessaire pour les adapter à cette nouvelle configuration. La possibilité de faire migrer des portes ou d’utiliser les groupes de portes procure une base saine pour développer des applications qui se reconfigurent dynamiquement et/ou résistent aux pannes.

3.2.3.3 Processus UNIX Multi-Activités

Il est possible d’écrire des processus UNIX multi-activités, ou multi-**u_threads**. Une *u_thread* peut être considérée comme un processus léger s’exécutant dans un processus UNIX classique. Elle partage toutes les ressources du processus et en particulier son espace d’adressage et sa table de fichiers ouverts. Chaque *u_thread* représente un "fil" de contrôle différent.

Quand un processus est créé par `fork(2)`, il commence son exécution avec une seule activité (u_thread), de même après `exec(2)` ; quand un processus arrête son exécution par `exit(2)`, toutes les u_threads de ce processus se terminent avec lui.

Les traitements sont associés aux signaux sur la base des u_thread et non seulement du processus : chaque u_thread possède son propre contexte de traitement des signaux. Un signal émis suite à une exception (division par zéro) est délivré à l’activité ayant provoqué l’erreur ; de même les signaux d’alarme sont délivrés à l’activité ayant positionné l’alarme. Tous les autres signaux sont diffusés à l’ensemble des u_threads du processus. Les handlers de signaux sont exécutés dans la pile de l’activité l’ayant positionné. Ainsi, la cohérence est assurée avec les handlers de signaux existants. Les u_threads peuvent invoquer des appels système UNIX, pour des raisons de simplicité (assurer efficacement la cohérence du contexte système d’un processus), les appels systèmes sont sérialisés à l’exception des appels bloquants comme `read(2)`, `write(2)`, `pause(2)`, `wait(2)`, `u_ipcReceive(2)` et `u_ipcCall(2)` (c.a.d. ceux interruptibles par signaux).

3.3 Implantation

3.3.1 Structure d’un processus UNIX

Un processus UNIX classique peut être perçu comme un "fil" d’exécution unique dans un espace d’adressage. En conséquence, chaque processus UNIX est implanté comme un acteur CHORUS déroulant une seule activité (u_thread). Son contexte système UNIX est géré par le Gestionnaire de Processus. L’espace d’adressage d’un processus est structuré en régions de mémoire pour le code, les données et la pile.

De plus, le Gestionnaire de Processus crée une porte de contrôle et une activité de contrôle à chaque acteur implantant un processus UNIX. Cette porte et cette activité de contrôle ne sont pas visibles du programmeur du processus.

L’activité de contrôle qui s’exécute dans le contexte du processus partage son espace d’adressage et peut facilement accéder et modifier son image mémoire (modifications de la pile sur réception de signaux, accès au code et aux données pour le débogueur, etc). Elle gère aussi les évènements asynchrones reçus par le processus (principalement des signaux). Ces évènements sont implantés comme des messages CHORUS reçus sur la porte de contrôle du processus.

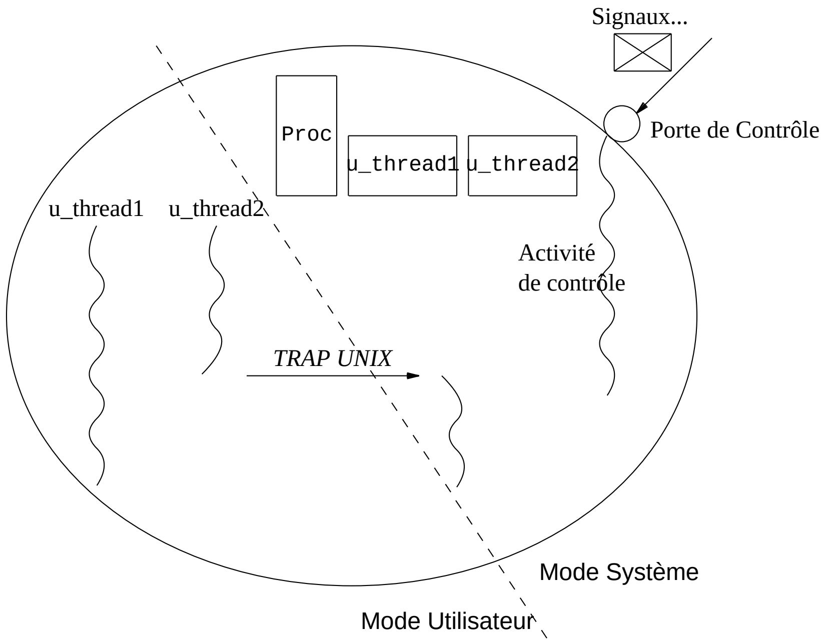

Figure 7. – Processus UNIX

Comme un processus peut être multi-activités, le contexte système d’un processus a été séparé en deux : un contexte associé au processus (`Proc`) et un contexte d’activité (`u_thread`).

La plupart des services implantés hors du Gestionnaire de Processus sont relatifs aux fichiers.. Cependant, le ‘‘contexte fichier’’ d’un processus (c.a.d. catalogues courant et racine, table des fichiers ouverts, attributs `umask` et `ulimit`) sont stockés dans la structure `Proc` gérée par le Gestionnaire de

Processus. Pour ce faire, un protocole spécifique entre le Gestionnaire de Processus et les autres serveurs a été conçu.

Les deux contextes système `Proc` et `u_thread` sont gérés par le gestionnaire de Processus du site courant d’exécution du processus. Ces contextes ne sont accédés ni par le Noyau CHORUS, ni par les autres serveurs système. Par ailleurs, le sous-système UNIX ne peut accéder les structures internes du Noyau associées aux acteurs et aux activités. Le seul moyen de les accéder est d’utiliser les services offerts par le Noyau CHORUS (ceci est essentiel, pour autoriser plusieurs sous-système à cohabiter au dessus d’un même Noyau CHORUS).

3.3.2 Environnement d’un processus identifié par un ensemble de portes

Les propriétés des portes CHORUS (noms globaux uniques, adressage indépendant de la localisation) les rend extrêmement intéressantes pour désigner des entités système. Leur intérêt principal réside dans l’indirection qu’elles induisent entre un processus et son environnement, et dans leur robustesse vis à vis des évolutions de configuration. Les noms de portes stockés dans le contexte d’un processus restent valides si le processus migre vers un autre site (c.a.d. `exec` sur un site distant) ou si une entité ainsi désignée migre.

Les portes constituent l’essentiel du contexte d’un processus, qu’elles soient utilisées directement ou incluses dans une *capacité*. Incluses dans des capacités, les portes sont utilisées pour désigner les ressources du processus : fichiers ouverts, segments mappés dans l’espace d’adressage du processus (code, données). Mais, elles peuvent aussi être utilisées pour adresser directement des processus (processus père).

Ressources et Capacités

Chaque ressource (gérée par un serveur) utilisée par un processus est désignée de manière interne par une capacité : fichier ouvert, tube ouvert, périphérique ouvert, répertoires courant et racine, segments de codes et de données, etc. De telles capacités peuvent être utilisées pour créer des régions de mémoire virtuelle, leur structure est donc celle exportée par le Noyau CHORUS.

Par exemple, l’ouverture d’un fichier associe la capacité retournée par le serveur approprié à un descripteur de fichier ouvert. Cette capacité se compose de la porte du serveur qui gère le fichier d’une part et de la référence du fichier ouvert dans le serveur d’autre part. Par la suite, toutes les requêtes sur le fichier ouvert (par exemple : `lseek(2)`, `read(2)`) sont transcrites et envoyées directement au serveur concerné sous forme de message.

Les serveurs étant désignés par des portes, localisées par l’IPC CHORUS, le sous-système UNIX n’a pas besoin de localiser lui-même ses serveurs.

Les capacités sont construites et retournées par les serveurs. Un serveur peut ainsi déléguer la réalisation d’un service à un autre serveur, sans que les clients sachent quel serveur répond réellement à leurs requêtes.

3.3.3 Le Gestionnaire de Processus

Le code du sous-système UNIX relatif à la gestion de processus, la gestion des signaux et les interfaces systèmes accessibles depuis un processus sont inclus dans le Gestionnaire de Processus (PM).

Cet acteur est chargé à l’initialisation du système. Son code et ses données résident dans l’espace système. La présence du PM dans l’espace système lui permet d’implanter l’invocation des appels système par `traps` comme pour un noyau UNIX traditionnel (du code PM est exécuté par les activités lors de chaque appel système). Il est ainsi possible d’assurer une compatibilité au niveau binaire avec d’autres systèmes UNIX.

3.3.4 Le Gestionnaire de Fichiers

Le Gestionnaire Fichiers est un acteur système qui possède deux portes sur lesquelles il reçoit des messages de requêtes. L’une d’elles sert exclusivement pour les messages de demande de pagination émis par le Gestionnaire de Mémoire Virtuelle du Noyau CHORUS. L’autre porte reçoit toutes les autres requêtes : services UNIX, messages de coopération entre les PM et les FM, etc.

Une fois l’initialisation terminée, plusieurs activités s’exécutent en parallèle dans le FM. Elles traitent les messages de l’une ou de l’autre porte.

Comme le contexte du processus pour lequel le FM traite une requête ne lui est pas directement accessible, l’information contextuelle nécessaire pour rendre un service est incluse dans le message de requête avec les paramètres de l’appel système. En retour, le serveur inclut dans le message de réponse les informations nécessaires pour la mise à jour du contexte fichier du processus. Les autres serveurs comme les gestionnaires de tubes, de périphériques ou de sockets opèrent suivant un mécanisme similaire.

4. Retour à la simplicité originelle

4.1 Des caractéristiques ‘‘UNIX’’

Le système UNIX outre son interface est aussi devenu populaire grâce à un ensemble de caractéristiques, et à une philosophie appliquée au système lui-même, mais aussi et surtout au niveau de l’interface utilisateur :

- UNIX *était* un petit noyau portable et facilement maitrisable. Si le fait d’être écrit dans un langage de ‘‘haut’’ niveau continue à assurer partiellement sa portabilité, et favorise son apprentissage, le noyau UNIX a cessé d’être petit et facilement maitrisable.
- Faire des programmes qui ne font qu’une chose simple, mais qui la font bien. Ce principe induit la notion de modularité ; son respect entraine en effet la réalisation dans des entités indépendantes des fonctions système, interpréteurs de commandes, compilateurs, éditeurs de liens...
- Faire que toute sortie d’un programme puisse servir d’entrée à un autre programme. Ce principe trouve toute son application grâce aux tubes et aux programmes filtres tels `sed`, `grep`, etc.
- Construire des applications (ou tout au moins leurs prototypes) à partir de briques existantes en les combinant pour offrir des services plus élaborés, plutôt que de tout écrire. Cette combinaison d’outils existants prend toute sa valeur grâce au principe précédent : on considère alors l’existant comme une boite à outils réutilisables dans d’autres configurations pour satisfaire d’autres besoins.
- Enfin, des services rendus par UNIX, même s’ils ne font pas vraiment partie de "la philosophie", lui donnent une touche de convivialité pour le programmeur : par exemple le contrôle d’exécution (job control).

Si certaines de ces caractéristiques sont toujours vraies, notamment ce qui caractérise l’interface utilisateur, le noyau UNIX a été peu à peu dénaturé par l’ajout de fonctionnalités résultant du travail de milliers de hackers×années sur un noyau monolithique.
CHORUS/MIX, qui identifie le système UNIX construit à l’aide du Noyau CHORUS et de ses "serveurs" UNIX applique les principes énoncés ci-dessus à la structure intime du système sans en changer l’interface. Le système ainsi construit est composé d’entités

— petites et facilement maitrisables (écrites dans un langage de haut niveau),

— jouant un seul rôle, bien défini, plutôt que plusieurs, c’est à dire responsables de la gestion d’un seul type de ressource plutôt que de plusieurs,

— combinables, remplaçables et extensibles pour réaliser des systèmes plus ou moins complexes suivant les besoins des applications, fournissant ainsi une boite à outils système extensible,

— offrant des mécanismes permettant de contrôler et d’adapter le comportement et l’exécution au mieux des besoins de l’utilisateur.

On retrouve là les principales caractéristiques UNIX présentées plus haut. La suite de ce chapitre décrit les aspects de CHORUS/MIX qui illustrent ce parallèle entre les mécanismes UNIX bien connus et leur application à la structure du système.

4.2 Le Gestionnaire de Processus : un interpréteur d’appels système

Le Gestionnaire de Processus de CHORUS/MIX est la clé de voûte de l’implantation de l’interface UNIX. Tous les appels système UNIX passent par lui. Il répond seul à certains de ces appels systèmes (`getpid`, `setuid`, `wait`, etc). Pour d’autres, il fait appel aux autres serveurs (Gestionnaire de fichiers pour `open`, Gestionnaire de Sockets pour `socket`, etc.). Enfin, certains services sont le résultat d’une coopération entre le Gestionnaire de Processus et d’autres serveurs (`exec`, `fork`, etc).

La manière dont les services offerts aux utilisateurs par les différents serveurs apparaissent à l’interface, est complètement transparente et indifférente à ces serveurs. Seul le PM implante réellement la sémantique complète du système. Il est possible de concevoir d’autres interfaces de systèmes, tout en conservant certains des serveurs (Gestionnaires de Fichiers, de Sockets, par exemple).

On retrouve là, exactement les principes qui régissent traditionellement la réalisation des interpréteurs de commande sous UNIX. Chacun peut choisir ou développer son ‘‘`shell`’’ tout en continuant à utiliser les autres services de façon identique (par exemple les mêmes fichiers binaires exécutables).
De la même manière qu’un `shell` fournit des services ‘‘built-in’’, le PM rend certains de ses services sans concours extérieur autre que celui du système sur lequel il s’appuie. D’autre part, le PM donne accès à des services fournis par des serveurs externes (fichiers, terminaux, etc.) comme le shell permet d’accéder aux services d’édition, de compilation, etc.

4.3 CHORUS/MIX : boite à outils système

Il est possible d’utiliser ou de ne pas utiliser les serveurs de CHORUS/MIX pour réaliser tel ou tel environnement particulier, de la même manière qu’on utilise certains des utilitaires UNIX pour les assembler dans des "shell-script". La modularité de CHORUS/MIX permet de le considérer comme une boite à outils système dans laquelle l’ingénieur système choisit les briques nécessaires à telle ou telle configuration matérielle ou logicielle.

4.3.1 Configurations classiques

4.3.1.1 Machines isolées

Sur une machine isolée (standalone), qui n’a pas besoin de supporter de protocoles réseaux, le gestionnaire de réseaux n’est pas nécessaire et n’est donc pas chargé avec le système.

4.3.1.2 Stations de Travail

Sur une station de travail sans disque local, nul besoin d’un gestionnaire de fichiers. Un Noyau, un Gestionnaire de Réseau, un Gestionnaire de Processus, un Gestionnaire de Terminaux (pour l’accès à l’écran bitmap, et aux pseudo-tty) et éventuellement un Gestionnaire de Sockets sont seuls nécessaires pour offrir un environnement UNIX complet. Les appels systèmes UNIX relatifs aux fichiers sont convertis en des requêtes véhiculées via l’IPC CHORUS par le PM, permettant ainsi l’accès transparent à des Gestionnaires de Fichiers s’exécutant sur des serveurs de disques distants.

Le Gestionnaire de Tubes (PiM) est aussi présent sur les stations sans disque, bien que cela ne soit pas strictement nécessaire. S’il n’est pas présent, un autre serveur équivalent dans le système distribué répondra aux requêtes liées aux tubes émises par cette station. La présence du PiM sur la station a pour seul effet d’avoir de meilleurs temps d’accès aux tubes, en évitant d’accéder au réseau.

4.3.1.3 Multi-computers

Pour un système distribué comme CHORUS, l’architecture d’une machine multi-computer (par exemple, un hypercube) est très semblable à la structure d’un réseau de serveurs et de stations. Les mêmes choix de configuration peuvent être faits : charger des drivers seulement sur les nœuds où ils sont utiles, charger un Gestionnaire de Sockets seulement sur les nœuds offrant une connexion avec le monde extérieur, charger un Gestionnaire de Fichiers là où des disques sont connectés. Seuls un Noyau, un Gestionnaire de Communications et un Gestionnaire de Processus doivent être présents sur chaque nœud pour faire que ce nœud offre aux applications un environnement UNIX complet. Sur les nœuds où seul un processus applicatif doit s’exécuter, on peut réduire le système au seuls noyaux et Gestionnaire de Communications.

En fait, la version du Gestionnaire de Communications présente sur un nœud non connecté à un réseau externe, n’a pas besoin de fournir la totalité des protocoles réseaux, mais seulement ceux utilisés pour la communication entre les nœuds de la machine. Les Gestionnaires de Communications qui résident sur des nœuds offrant un accès au monde extérieur doivent fournir les deux familles de protocoles. Cette architecture est celle utilisée dans la machine EuroWorkStation développée dans le projet Esprit 2569

(EWS).

4.3.1.4 Multi-processeurs

Le noyau CHORUS peut s’exécuter sur des machines multi-processeurs symétriques, offrant aux acteurs un accès uniforme et simultané aux processeurs.

La modularité du sous-système UNIX permet de bénéficier immédiatement de cette possibilité: les services UNIX étant implantés comme des serveurs indépendants, aucune synchronisation n’est nécessaire entre ces serveurs. Ainsi, sur une machine quadri-processeur, chaque processeur peut exécuter une activité d’un serveur UNIX différent (par exemple, PM, FM, DM, SM). On obtient ainsi un multiplexage très simple des services UNIX sans se soucier d’ajouter des mécanismes de synchronisation dans les serveurs qui ont été directement dérivés du code UNIX (FM, DM).

De plus, certains des nouveaux serveurs, comme le Gestionnaire de Processus, ont d’ores et déjà été structurés pour qu’ils puissent eux-mêmes être multiplexés. Ainsi, plusieurs processus peuvent faire un appel système simultanément. Sur une machine à quatre processeurs, deux processus peuvent faire `fork(2)`, un autre peut faire `open(2)` et un autre encore `read(2)`, tous réellement en parallèle. La synchronisation sur les tables globales du Gestionnaire de Processus est faite avec une granularité très fine.

Vis à vis du multiplexage, les serveurs dérivés du code UNIX ont le même niveau de granularité que dans les noyaux UNIX actuels. Un parallélisme plus grand sera introduit dans les prochaines versions de CHORUS/MIX, basé sur les services de synchronisation offerts par le Noyau CHORUS.

4.3.1.5 Systèmes Embarqués

Certaines applications temps-réel embarquées n’ont pas forcément besoin de services autres que ceux offerts par le Noyau CHORUS, le Gestionnnaire de Processus et le Gestionniare de Sockets. Ces serveurs donnent accès, à de telles applications, aux services de gestion des processus et des activités, de l’IPC, de gestion mémoire et de connexion au évènements matériels. La communication avec le monde extérieur peut être basée sur les sockets.

Ces applications embarquées peuvent s’exécuter dans un environnement sans aucune autre machine gérée par le système CHORUS. De manière à offrir plus de flexibilité (accès fichiers) et de dynamisme (chargement/déchargement de programmes, mise au point à distance) à de telles applications, un Gestionnaire de Fichiers très simple peut être mis en œuvre, offrant les services UNIX d’accès aux fichiers (y compris les pseudo-ttys). Ce Gestionnaire de Fichiers transforme toutes les requêtes d’accès aux fichiers véhiculées par l’IPC CHORUS en communication sur les sockets. Les requêtes transportées au moyen d’une connexion sur les sockets sont alors exécutées par un processus UNIX jouant le rôle de serveur distant et s’exécutant sur tout système UNIX.

4.3.2 Exemples

L’adaptabilité de CHORUS/MIX est illustrée par les trois cas concrets exposés ci-après.

4.3.2.1 Serveur d’accès documentaire

Un serveur d’accès documentaire a été développé sur une carte construite autour d’un Motorola 68030 et intégrée à un MacII. L’application s’exécute sur la carte à laquelle sont connectés les disques (disques miroirs et/ou jukebox optique). Seuls les services de gestion du disque et d’accès aux fichiers sont utilisés par l’application. L’accès à cette application depuis le monde extérieur (c.a.d. le MacII, et d’autres clients connectés au MacII par un réseau), est fait via une zone de mémoire partagée par le MacII et la carte 68030.

Le système sur la carte comprend un Noyau CHORUS (sans Gestionnaire de Réseau) et un Gestionnaire de Fichiers. Une bibliothéque a été développée (à partir du code du Gestionnaire de Processus en un mois) qui transforme tout appel système UNIX relatif à la gestion de fichiers, en une requête IPC CHORUS. Un contexte système de fichiers émulant le contexte fichiers d’un processus UNIX est géré dans l’espace utilisateur, de la même manière que les ‘‘streams’’ d’entrée/sortie sont gérés par la bibliothèque C standard. On évite ainsi de charger la totalité du Gestionnaire de Processus, ce qui permet un gain de place et des meilleurs temps de réponse pour les accès fichiers en évitant les accès par ‘‘traps’’.

4.3.2.2 Terminal X : Le tX

CHORUS est utilisé dans le terminal X, *le tX*, développé par la société Gipsi. Le seul programme qui s’exécute sur une telle configuration est le serveur X, qui répond aux requêtes venant de clients qui s’exécutent sur d’autres machines. Il est naturellement conçu pour être utilisé dans des réseaux sans autre machine CHORUS.

Le système du terminal X est constitué d’un Noyau CHORUS, d’un Gestionnaire de Processus, d’un Gestionnaire de Sockets et d’un Gestionnaire de Périphériques (pour les drivers clavier et souris). Il n’y a aucun besoin de Gestionnaire de Tubes ou de Fichiers. Cependant, comme le serveur X invoque quelques appels systèmes `open(2)` lors de son initialisation pour accéder aux périphériques, une petite émulation très simple de ces appels systèmes a été développée pour couvrir ces besoins.

4.3.2.3 EuroWorkStation

L’EuroWorkStation développée dans le cadre du projet Esprit EWS est une station de travail scientifique haut de gamme. Elle est bâtie autour d’une carte principale basée sur un multiprocesseur SPARC symétrique. Des coprocesseurs basés sur le chip Intel 80386 peuvent être connectés au travers d’un bus MultiBus II. Des accès à Ethernet, FDDI et à un bus SCSI sont aussi fournis. L’affichage est fait à distance sur un terminal X. Parmi les coprocesseurs prévus, on trouve une machine graphique 3D, une machine Lisp et une machine de simulation.

CHORUS/MIX sur une telle configuration offrira un UNIX complet sur la carte principale. Sur les coprocesseurs, seuls un Noyau CHORUS, un Gestionnaire de Communications (adapté au MultiBus II), et un Gestionnaire de Processus UNIX seront présents. Cela permettra aux utilisateurs de charger dynamiquement des programmes sur les coprocesseurs depuis la carte principale. De plus, un acteur spécifique dérivé du Gestionnaire de Périphériques sera chargé sur chaque coprocesseur pour permettre l’accès aux ressources propres à chaque coprocesseur.

4.4 Contrôle du système

L’utilisation de l’ordonnancement temps-réel offert par le Noyau CHORUS permet le développement de différentes politiques d’ordonnancement dans les sous-systèmes. CHORUS/MIX utilise ces possibilités pour fournir un environnement d’exécution temps réel au niveau UNIX.

4.4.1 Priorité d’un Serveur

Le Noyau CHORUS cadence les activités sur la base d’une priorité fixe attachée à chaque activité. Les priorités s’échelonnent de 1 (la plus haute) à 255 (la plus basse). Les activités s’exécutant à une priorité entre 128 et 255 sont de plus soumises à un partage de temps pour un niveau de priorité donné (le mécanisme de partage de temps ne dégrade pas la priorité d’une activité). Dans une configuration CHORUS classique, les processus UNIX s’exécutent à la priorité 192, et les serveurs UNIX à des priorités comprises entre 64 et 68. Cela donne à l’utilisateur la possibilité de lancer des applications temps-réel avec une priorité plus grande que celle des processus UNIX classiques ; elles peuvent même s’exécuter avec des priorités plus grande que celle des serveurs UNIX !

Si l’intervalle des priorités utilisé par les serveurs UNIX ne convient pas pour une application donnée, les priorités des serveurs peuvent être modifiées soit statiquement par compilation des serveurs, soit dynamiquement en leur envoyant une requête leur demandant de changer leur priorité. Dans les deux cas, les priorités de ces serveurs peuvent être baissées ou augmentées. Dans tous les cas, il est cependant nécessaire de faire très attention aux nouvelles priorités affectées aux serveurs UNIX : il n’est pas forcément très sensé de faire exécuter les serveurs UNIX à la plus basse priorité pendant que les programmes standards UNIX s’exécutent avec la plus haute...

4.4.2 Gestion des interruptions

Les serveurs UNIX qui gèrent des interruptions (Gestionnaire de Fichiers, Gestionnaire de Périphériques) déroulent le code de traitement de l’interruption (‘‘handler’’ d’interruption) au niveau de l’interruption elle-même, de la même manière que dans un noyau UNIX classique, permettant ainsi d’obtenir des temps de réponse équivalents à ceux obtenus par de tels noyaux UNIX.

Cependant, le Gestionnaire de Fichiers et le Gestionnaire de Périphériques peuvent aussi s’exécuter dans un autre mode. Lors de la réception d’une interruption, ils peuvent seulement poster un évènement à destination d’une activité dédiée (nommée ‘‘*Activité d’Interruption*’’) créée à l’initialisation du système, et ensuite relacher le processeur après avoir acquitté l’interruption. Poster un évènement peut être fait au moyen des primitives de synchronisation offertes par le Noyau CHORUS. La procédure de traitement de l’interruption réelle du pilote sera alors exécutée quand l’Activité d’Interruption aura été ordonnancée par le noyau, selon sa priorité. Ce ‘‘traitement différé des interruptions’’ permet de minimiser le temps durant lequel les interruptions sont masquées, laissant ainsi plus de temps aux processus temps-réel pour s’exécuter, recevoir et gérer leurs propres interruptions. Dans ce mode, la gestion des sections critiques dans le Gestionnaire de Fichiers (et dans le Gestionnaire de Périphériques) est faite sans masquer réellement les interruptions. Evidemment, ce mécanisme implique un ordonnancement supplémentaire pour chaque interruption dont le traitement est différé. En conséquence, le temps de réponse des serveurs UNIX est dégradé ; aussi ce mode n’est-il utilisé que pendant que des applications temps-réel s’exécutent.

L’administrateur système peut changer dynamiquement le mode de traitement des interruptions du mode immédiat (équivalent aux implantations classiques d’UNIX) au mode différé, et réciproquement. De plus, la priorité des Activités d’Interruption peut être fixée dynamiquement.

Ces mécanismes permettent le développement d’applications temps-réel dans un environnement UNIX standard pendant que l’on saisit ou que l’on compile l’application. Une fois l’application écrite, avant de l’exécuter et de la tester, les priorités des serveurs UNIX peuvent être baissées, et le traitement des interruptions basculé en mode différé de manière à minimiser les périodes pendant lesquelles les interruptions sont masquées et permettre à l’application de réagir correctement. Quand l’application est terminée, les serveurs UNIX peuvent être remis dans leur état précédent : priorités plus élevées, traitement immédiat des interruptions. Cela évite d’avoir à arrêter et recharger le système pour passer du mode "développement" au mode "exécution".

4.4.3 Job Control Système

Le mécanisme décrit ci-dessus est comparable au service de ‘‘job control’’ offert par certains shells UNIX qui permettent d’arrêter des processus pour les relancer plus tard, en ‘‘background’’ ou en mode interactif.

L’exécution des serveurs de CHORUS/MIX est contrôlée en changeant leur priorité, indépendamment les uns des autres, comme si on leur appliquait une fonction `nice(2)` dynamiquement. Ce résultat n’est évidemment possible que parce que les services UNIX ont été complètement restructurés.

4.5 Redirections Système

4.5.1 Extensions des Services Systèmes

Un autre aspect de la flexibilité offerte par CHORUS est la possibilité d’adapter dynamiquement les services offerts par le système aux besoins de l’utilisateur. Il est notamment possible d’étendre les services offerts en utilisant le mécanisme des portes symboliques du Gestionnaire de Fichiers. Ceci a été utilisé, en particulier, pour dupliquer des fichiers UNIX de manière transparente à l’utilisateur, c.a.d. sans modifier l’interface ou les programmes existants, et même sans modifier le gestionnaire de Processus ou le Gestionnaire de Fichiers.

A l’initialisation, un Serveur de Duplication crée une porte et l’enregistre dans l’arborescence de fichiers UNIX. A chaque fois que le Gestionnaire de Fichiers rencontre cette porte symbolique au cours de l’analyse d’un chemin d’accès dans le traitement d’une requête, il redirige cette requête sur cette porte. Le Serveur de Duplication peut alors examiner la requête, la dupliquer et expédier chacune d’elles à un Gestionnaire de Fichiers différent du réseau. Après avoir reçu les deux réponses, le Serveur de Duplication peut alors répondre au client initial comme s’il était le Gestionnaire de Fichiers initial.

Ce mécanisme de redirection de requête est assez similaire au mécanisme de redirections des entrées/sorties dans UNIX. Cependant, au lieu de traiter un flot de données, on traite ici un flot de requêtes (données structurées) suivant un protocole à respecter.

5. Conclusion

CHORUS/MIX illustre parfaitement les bénéfices que l’on peut obtenir de la modularité lorsqu’elle est appliquée à un système d’exploitation. Contrairement à ce qu’on pourrait attendre ces bénéfices ne se font pas au détriment des performances. Le tableau suivant résume quelques résultats obtenus sur un

COMPAQ 386/25. CHORUS/MIX est comparé à SCO System V, 3.2.

TABLE 1. – Performances de CHORUS/MIX

Primitives	**CHORUS/MIX v3.2**	**SCO SV3.2**
`getuid`	*90 µs*	*93 µs*
`sbrk(0)`	*115 µs*	*129 µs*
`read (4Kb)`	*111 Kb/s*	*111 Kb/s*
`write (1Kb)`	*70 Kb/s*	*80 Kb/s*
`pipe (4096)`	*772 Kb/s*	*1 360Kb/s*
`creat`	*16 ms*	*17 ms*
`exec`	*15 ms*	*7.7 ms*
`fork`	*20 ms*	*8.8 ms*

CHORUS/MIX (version 3.2) supporte aujourd’hui le COMPAQ 386, et une machine basée sur le Motorola 88000. De plus, le Noyau CHORUS supporte des cartes à base de Motorola 68020, 68030, et 88000 et d’Intel 186 et 386.

Un sous-système UNIX compatible BSD est en cours de développement, ainsi qu’un sous-système compatible avec UNIX System V Rel 4.0.

La volonté de faire du Noyau CHORUS un noyau générique, a permis de ne pas y introduire des fonctionnalités ayant une sémantique contraignante. Par exemple, aucune stratégie de tolérance aux pannes n’est impalntée dans le noyau lui-même. Celui-ci fournit cependant les services élémentaires permettant la réalisation de tels services au sein de sous-systèmes.

D’autre part, CHORUS offre des solutions concrètes et performantes à quelques uns des problèmes qui posent des difficultés aux concepteurs de système, comme par exemple, la (re)configutation système (statique et dynamique), l’adaptabilité, l’extensibilité, et la mise au point qui sont facilités par la répartition des ressources au sein d’acteurs ne communiquant que par des messages selon une interface explicite et claire.

La structure modulaire de CHORUS, tout en permettant la compatibilité binaire avec UNIX, permet de garder une réalisation bien structurée, portable et efficace.

Tout ces principes sont ceux sur lesquels UNIX a été conçu il y a 20 ans. Les réseaux et les multicomputers introduisent aujourd’hui de nouvelles possibilités et contraintes qui obligent à ‘‘repenser’’ la structure interne d’UNIX de manière à ce qu’il soit toujours un système d’exploitation moderne, intégrant la répartition en son sein. De toute évidence CHORUS montre qu’UNIX peut retrouver ses vertus initiales tout en conservant une interface standard garantissant la portabilité des applications, et évoluer ainsi vers la prochaine génération de système d’exploitation.

6. Références

[Abro89] Vadim Abrossimov, Marc Rozier, and Marc Shapiro, ‘‘Generic Virtual Memory Management for Operating System Kernels,’’ *Submitted for publication*, (April 1989), p. 20.

[Abro89a] Vadim Abrossimov, Marc Rozier, and Michel Gien, ‘‘Virtual Memory Management in Chorus,’’ in *Lecture Notes in Computer Sciences, Workshop on Progress in Distributed Systems and Distributed Systems Management*, Springer-Verlag, Berlin, Germany, (18-19 April 1989), p. 20.

[Acce86] Mike Accetta, Robert Baron, William Bolosky, David Golub, Richard Rashid, Avadis Tevanian, and Michael Young, ‘‘Mach: A New Kernel Foundation for UNIX Development,’’ in *Proc. of USENIX Summer’86 Conference*, Atlanta, GA, (9-13 June 1986), pp. 93-112.

[Alve88] Jose Alves-Marques, Roland Balter, Vinny Cahill, Paulo Guedes, Neville Harris, Chris Horn, Sacha Krakowiak, Andre Kramer, John Slattery, and Gérard Vandome, ‘‘Implementing the Comandos Architecture,’’ in *Esprit’88: Putting the Technology to Use*, North-Holland Publishing Co., (November 1988), pp. 1140-1157.

[Cher88] David Cheriton, ‘‘The V Distributed System,’’ *Communications of the ACM*, vol. 31, no. 3, (March 1988), pp. 314-333.

[Herr88] Frédéric Herrmann, François Armand, Marc Rozier, Michel Gien, Vadim Abrossimov, Ivan Boule, Marc Guillemont, Pierre Léonard, Sylvain Langlois, and Will Neuhauser, ‘‘CHORUS, a New Technology for Building UNIX Systems,’’ in *Proc. of EUUG Autumn’88 Conference*, EUUG, Cascais, Portugal, (3-7 October 1988), pp. 1-18.

[Li86] Kai Li and Paul Hudak, ‘‘Memory Coherence in Shared Virtual Memory Systems,’’ in *Proc. of Principles of Distributed Computing (PODC) Symposium*, (1986), pp. 229-239.

[McJo88] Paul R. McJones and Garret F. Swart, ‘‘Evolving the UNIX System Interface to Support Multithreaded Programs,’’ Technical Report 21, DEC Systems Research Center, Palo Alto, CA, (September 1988), p. 100.

[Mino88] Régis Minot, Pierre Courcoureux, Hubert Zimmermann, Jean-Jacques Germond, Paolo Alvari, Vincenzo Ambriola, and Ted Dowling, ‘‘The Spirit of Aphrodite,’’ in *Esprit’88: Putting the Technology to Use*, North-Holland Publishing Co., (November 1988), pp. 519-539.

[Mull87] Sape J. Mullender et al., *The Amoeba Distributed Operating System: Selected Papers 1984 -1987*, CWI Tract No. 41, Amsterdam, Netherlands, (1987), p. 309.

[Pres86] David L. Presotto, ‘‘The Eight Edition UNIX Connection Service,’’ in *Proc. of EUUG Spring’86 Conference*, Florence, Italy, (21-24 April 1986), p. 10.

[Rash87] Richard Rashid, Avadis Tevanian, Michael Young, David Golub, Robert Baron, David Black, William Bolosky, and Jonathan Chew, ‘‘Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures,’’ in *ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS II)*, (October 1987), pp. 31-39.

[Rozi88] Marc Rozier, Vadim Abrossimov, François Armand, Ivan Boule, Michel Gien, Marc Guillemont, Frédéric Herrmann, Claude Kaiser, Sylvain Langlois, Pierre Léonard, and Will Neuhauser, ‘‘CHORUS Distributed Operating Systems,’’ *Computing Systems Journal*, vol. 1, no. 4, The Usenix Association, (December 1988), pp. 305-370.

[Tane86] Andrew S. Tanenbaum, Sape J. Mullender, and Robert van Renesse, ‘‘Using Sparse Capabilities in a Distributed Operating System,’’ in *Proc. of IEEE 6th. International Conference on Distributed Computing Systems*, CWI Tract No. 41, Cambridge, MA, (19-23 May 1986), pp. 558-563.

[Wein86] Peter J. Weinberger, ‘‘The Eight Edition Remote Filesystem,’’ in *EUUG Spring’86 Conference*, Florence, Italy, (21-24 April 1986), p. 1.

[Coyo89] Hugo Coyote, ‘‘Spécification et Réalisation d’un Système de Fichiers Fiables pour le Système d’Exploitation Réparti CHORUS,’’ PhD.Thesis, Université Paris VI, Paris, France, (June 1989), p. 300.

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLES