CHAPITRE 4 : LA COMMUNICATION INTERPROCESSUS

4.1 INTRODUCTION :

La communication interprocessus (interprocess communication, IPC) consiste à transférer des données entre les processus. Par exemple, un navigateur Internet peut demander une page à un serveur, qui envoie alors les données HTML.

La communication interprocessus peut se faire de différentes manières : mémoire partagée, mémoire mappée, tubes, files et socket.

- La mémoire partagée permet aux processus de communiquer simplement en lisant ou écrivant dans un emplacement mémoire prédéfini.
- La mémoire mappée est similaire à la mémoire partagée, excepté qu'elle est associée à un fichier.
- Les tubes permettent une communication séquentielle d'un processus à l'autre.
- Les files FIFO sont similaires aux tubes excepté que des processus sans lien peuvent communiquer car le tube reçoit un nom dans le système de fichiers.
- Les sockets permettent la communication entre des processus sans lien, pouvant se trouver sur des machines distinctes.

Ces types d'IPC diffèrent selon les critères suivants:

- Ils restreignent ou non la communication à des processus liés (processus ayant un ancêtre commun), à des processus partageant le même système de fichiers ou à tout ordinateur connecté à un réseau.
- Un processus communiquant n'est limité qu'à la lecture ou qu'à l'écriture de données.
- Le nombre de processus pouvant communiquer.
- Les processus qui communiquent sont-ils synchronisés par l'IPC ? par exemple, un processus lecteur s'interrompt-il jusqu'à ce qu'il y ait des données à lire ?.

4.2 MEMOIRE PARTAGEE

Une des méthodes de communication interprocessus les plus simples est d'utiliser la mémoire partagée. La mémoire partagée permet à deux processus ou plus d'accéder à la même zone mémoire comme s'ils avaient leurs pointeurs dirigés vers le même espace mémoire. Lorsqu'un processus modifie la mémoire, tous les autres processus voient la modification.

La mémoire partagée est la forme de communication interprocessus la plus rapide car tous les processus partagent la même mémoire. Elle évite également les copies de données inutiles.

Pour utiliser un segment de mémoire partagée, un processus doit allouer le segment. Puis, chaque processus désirant accéder au segment doit l'attacher. Après avoir fini d'utiliser le segment, chaque processus le détache. À un moment ou à un autre, un processus doit libérer le segment.

Sous Unix, un processus alloue un segment de mémoire partagée en utilisant shmget (« SHared Memory GET », obtention de mémoire partagée). Son premier paramètre est une clé entière qui indique le segment à créer. Le second paramètre indique le nombre d'octets du segment. Le troisième paramètre est un ensemble d'indicateurs binaires décrivant les options demandées : s'agit-il d'un nouveau segment ou un segment existant qu'il faut attacher , permissions de lecture/écriture, … etc.

Inconvénient :

Les segments de mémoire partagée permettent une communication bidirectionnelle rapide entre n'importe quel nombre de processus. Chaque utilisateur peut à la fois lire et écrire. Cependant, le principe de l'exclusion mutuelle, nécessaire en cas d'accès concurrent, n'est pas garanti avec ce mode communication.

4.3 MEMOIRE MAPPEE

La mémoire mappée permet à différents processus de communiquer via un fichier partagé.

Pour mettre en correspondance un fichier ordinaire avec la mémoire d'un processus, on utilise l'appel mmap (« Memory MAPped », Mémoire mappée). Le premier paramètre est l'adresse de location du fichier . Le second paramètre est la longueur de l'espace de correspondance en octets. Le troisième paramètre définit la protection de l'intervalle d'adresses mis en correspondance.

4.3. TUBES

Un tube est un dispositif de communication qui permet une communication à sens unique. Les données écrites sur l'« extrémité d'écriture » du tube sont lues depuis l'« extrémité de lecture ». Les tubes sont des dispositifs séquentiels; les données sont toujours lues dans l'ordre où elles ont été écrites. Typiquement, un tube est utilisé pour la communication entre deux threads d'un même processus ou entre processus père et fils.

Dans un shell, le symbole | crée un tube. Par exemple, cette commande provoque la création par le shell de deux processus fils, l'un pour ls et l'autre pour less:

```
ls | less
```

Le shell crée également un tube connectant la sortie standard du processus ls avec l'entrée standard de less. Les noms des fichiers listés par ls sont envoyés à less dans le même ordre que s'ils étaient envoyés directement au terminal.

La capacité d'un tube est limitée. Si le processus rédacteur écrit plus vite que la vitesse à laquelle le processus lecteur consomme les données, et si le tube ne peut pas contenir de données supplémentaires, le processus rédacteur est bloqué jusqu'à ce qu'il y ait à nouveau de la place dans le tube. Si le lecteur essaie de lire mais qu'il n'y a plus de données disponibles, il est bloqué jusqu'à ce que ce ne soit plus le cas. Ainsi, le tube synchronise automatiquement les deux processus.
La création d'un tube se fait grâce à la fonction pipe. Elle admet comme paramètre un tableau de deux entiers. L'appel à pipe stocke le descripteur de fichier en lecture à l'indice zéro et le descripteur de fichier en écriture à l'indice un.

Exemple de création d'un tube :

```
int pipe_fds[2];
int fr;
int fw;

pipe (pipe_fds);
fr = pipe_fds[0];
fw = pipe_fds[1];
```

4.4 LES FILES FIFO

Une file premier entré, premier sorti (first-in, first-out, FIFO) est un tube qui dispose d'un nom dans le système de fichiers. Tout processus peut ouvrir ou fermer la file FIFO; les processus raccordés aux extrémités du tube n'ont pas à avoir de lien de parenté. Les FIFO sont également appelés canaux nommés.

On peut créer une FIFO via la commande mkfifo.

Pour créer une FIFO par programmation, utilisez la fonction mkfifo. Le premier argument est l'emplacement où créer la FIFO; le second paramètre spécifie les permissions du propriétaire du tube, de son groupe et des autres utilisateurs.

L'accès à une FIFO se fait de la même façon que pour un fichier ordinaire

4.5 LES SOCKETS

Un socket est un dispositif de communication bidirectionnel pouvant être utilisé pour communiquer avec un autre processus sur la même machine ou avec un processus s'exécutant sur d'autres machines. Les programmes Internet comme Telnet, rlogin, FTP, talk et le World Wide Web utilisent des sockets.

Pour créer un socket, il faut indiquer trois paramètres: le style de communication, l'espace de nommage et le protocole.

Un style de communication contrôle la façon dont le socket traite les données transmises et définit le nombre d'interlocuteurs. Lorsque des données sont envoyées via le socket, elles sont découpées en morceaux appelés paquets. Le style de communication détermine comment sont gérés ces paquets et comment ils sont envoyés de l'émetteur vers le destinataire.

– Le style *connexion* garantit la remise de tous les paquets dans leur ordre d'émission. Si des paquets sont perdus ou mélangés à cause de problèmes dans le réseau, le destinataire demande automatiquement leur retransmission à l'émetteur.

– Le style *datagramme* ne garantit pas la remise ou l'ordre d'arrivée des paquets. Des paquets peuvent être perdus ou mélangés à cause de problèmes dans le réseau.

L'espace de nommage d'un socket spécifie comment les adresses de socket sont écrites. Une adresse de socket identifie l'extrémité d'une connexion par socket. Par exemple, les adresses de socket dans « l'espace de nommage local » sont des noms de fichiers ordinaires. Dans « l'espace de nommage Internet », une adresse de socket est composée de l'adresse Internet (également appelée adresse IP) d'un hôte connecté au réseau et d'un numéro de port. Le numéro de port permet de faire la distinction entre plusieurs sockets sur le même hôte.

Un protocole spécifie comment les données sont transmises. Parmi ces protocoles, on peut citer TCP/IP; les deux protocoles principaux utilisés pour Internet, le protocole réseau AppleTalk; et le protocole de communication locale d'UNIX.