TP 1

Création de processus et signaux

1.1 But de ce TP

Ce TP doit vous permettre de manipuler les signaux pour réaliser la synchronisation entre 2 processus s'échangeant des données à travers un tube. Vous commencerez par créer un processus père et son fils. Ensuite, il manipuleront des signaux. Ils s'échangeront ensuite un jeton via le mécanisme de tube. Pensez à observer ce qu'il se passe à partir de commandes lancées dans le SHELL.

1.2 Création de processus

Objectif: *manipulation de fork, kill et fonctions de mise en place et de traitement des signaux, ps, grep*
Créez un programme qui lance un fils. Les deux programmes seront sensibles au signal SIGUSR1. Ils afficheront un message avant de se tuer à la réception du signal SIGUSR1. Le fils fera une boucle infini. Le père aprés une attente de 10 seconde (sleep) enverra un signal à son fils et fera ensuite une boucle infini. Pour arrêter le père envoyez à partir du SHELL un signal SIGUSR1. Il est conseillé de mettre des "printf" avec comme champ le pid du processus afin de comprendre ce qu'il se passe. Pensez à mettre en place la gestion de la mort du fils chez le père en utilisant le handler associé au signal SIGCHLD.

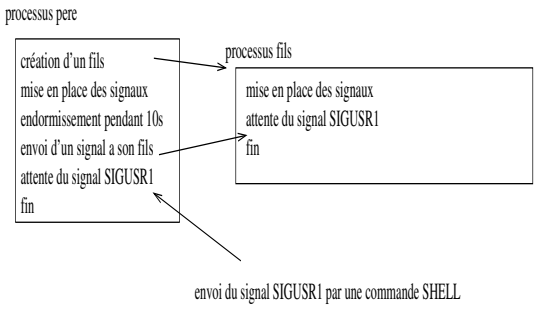

Figure 1.1: création d'un processus fils et synchronisation

1.3 Echange d'information entre les 2 processus

Objectif: *manipulation des tubes ordinaires et nommés*

Vous compléterez le programme ci-dessus afin de réaliser la communication de données par utilisation de tube entre le processsus père et le processus fils. Les deux processus réalisent un certain nombre de ping pong. Il faudra faire attention au problème de synchronisation initiale. Chaque processus renvoie le débit calculé.

Le père commence par écrire des données ensuite son fils lit les données et écrit ses propre données dans un tube que le père viendra ensuite lire.Le jeton échangé sera un entier incrément'e à chaque écriture.Faîtes une boucle qui réalise 1000 ping pong. Mesurez le débit (d en Mb/s) en mesurant la date avant la première écriture du père (t_d) et après la dernière lecture du père (t_f). La taille des données est la taille d'un entier soit 4 octets. La partie entière du débit est le code retourné par le programme.
$d = (1000 * 2 * 4 * 8) * 10^{-6}/(t_f - t_d)$

remarque: le code de retour d'un programme est une valeur comprise entre 0 et 255. Si le débit dépasse 255 Mb/s, passez en Mo/s.
Remarque: Vous utiliserez 2 tubes un pour envoyer et un pour recevoir. Afin de ne pas être tenté d'utiliser un tube dans les 2 sens de communication, vous pouvez fermer le descripteur ne vous servant pas.
Remarque: Pour la prise du temps utiliser la fonction gettimeofday et pensez à la précision de vos mesures en ayant un temps d'exécution suffisant.

1.3.1 Communication par tube ordinaire

Objectif: *manipulation du tube ordinaire*
Reprenez le principe du ping pong en utilisant comme méthode de communication et de synchronisation uniquement le tube non nommé. Ceci est possible grâce à la parenté entre les deux processus.

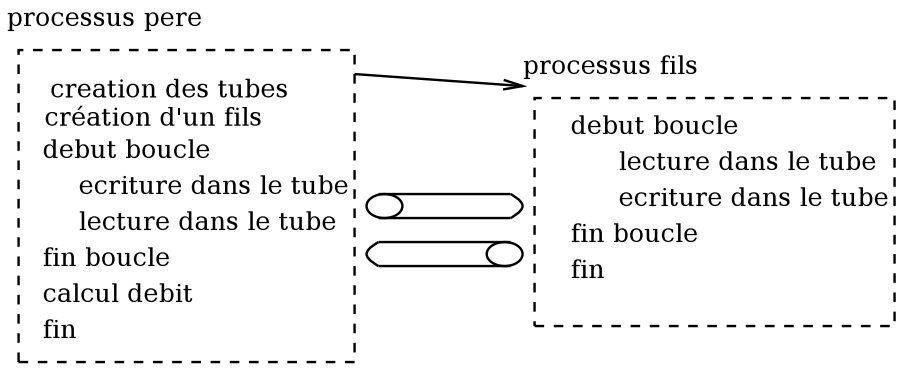

Figure 1.2: communication par tubes ordinaires

Communiquez un entier qui s'incrémente à chaque accès au tube. Vérifiez le débit.

1.3.2 Communication par tube nommé

Objectif: *manipulation du tube nommé, visualisation dans le système de fichiers, ls*
On désire faire communiquer deux processus indépendants. On utilise ici des tubes nommés qui permettent cela. Il faudra donc écrire soit deux programmes ou bien ce qui est mieux un programme acceptant des paramètres et qui se comportera en fonction de la valeur des paramètres soit comme le processus P1 soit comme le processus P2.

Communiquez un entier qui s'incrémente à chaque accès au tube. Vérifiez le débit. Visualisez les fichiers utilisés.

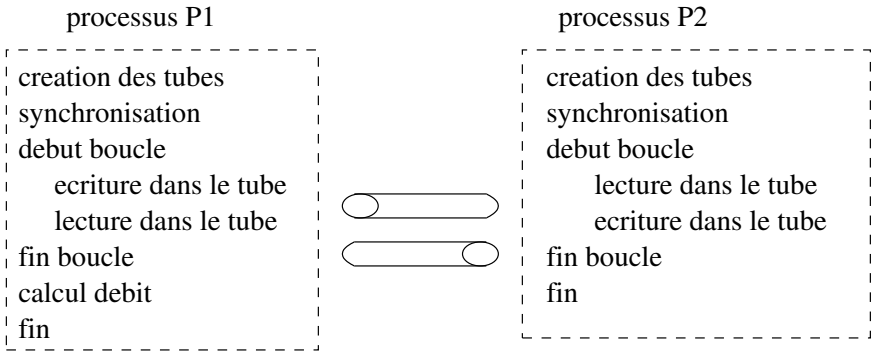

Figure 1.3: communication par tubes nommés

1.3.3 Joker

Courbe

Visualisez dans une courbe l'évolution du débit en fonction de la taille.

Petit protocole

Mettez en place un petit protocole vous permettant d'échanger des messages de tailles variables non connues à l'avance. L'objectif est d'échanger exactement la bonne taille et de ne pas sur-dimensionner.

TP 2

communication par mémoire partagée et synchronisation par sémaphore

2.1 But de ce TP

Ce TP reprend le principe des TP précédents. C'est simplement la méthode de communication qui est différente. Ici, on va utiliser de la mémoire partagée et des sémaphores pour la synchronisation. Chaque processus renvoie le débit calculé.

2.2 synchronisation par sémaphore

Objectif: *manipulation des sémaphores, ipcs, ipcrm*
Vous créerez deux processus réalisant une boucle infinie avec dedans une section critique accessible par un sémaphore. Chaque processus cherche à prendre le sémaphore, dès qu'il le possède il se met en attente d'une chaine de caractères tapée par l'utilisateur puis il libère le sémaphore et ainsi de suite. Visualisez les IPC à l'aide de commandes SHELL.

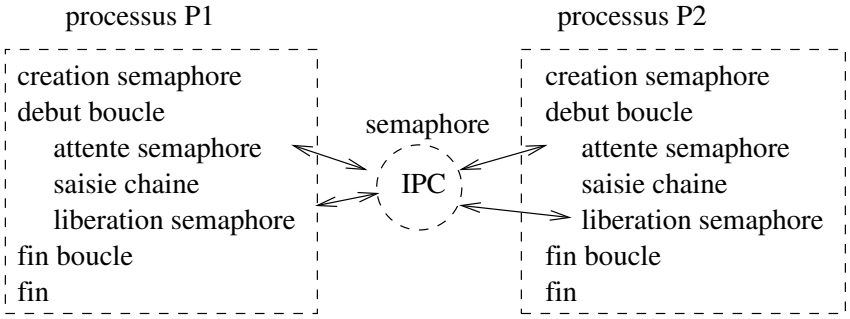

Figure 2.1: synchronisation par sémaphore

Que remarquez vous? quel est le problème ?

2.3 Communication par mémoire partagée

Objectif: *manipulation de la mémoire partagée et des sémaphores, ipcs, ipcrm*
Reprenez le principe du ping pong en utilisant comme méthode de communication entre deux processus indépendants la mémoire partagée offerte par les IPC et comme méthode de synchronisation un ou plusieurs sémaphores IPC.

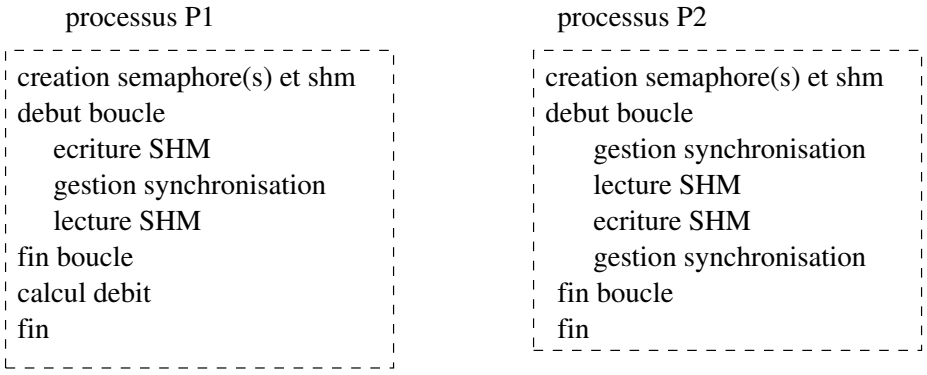

Figure 2.2: communication par mémoire partagée

2.3.1 Tampon simple

Communiquez dans un premier temps un entier qui s'incrémente à chaque accès à la mémoire partagée. Vérifiez le débit.

2.3.2 Joker

Courbe

Visualisez dans une courbe l'évolution du débit en fonction de la taille.

Petit protocole

Mettez en place un petit protocole vous permettant d'échanger des messages de tailles variables non connues à l'avance. Puis créez un système de boîtes aux lettres.