MIAG
formation
continue

Promo
2003-04

Sujet de TP - Systèmes d'exploitation

Communication et synchronisation des processus

Le but de ce tp est de vous faire mettre en œuvre rapidement quelqu'uns des moyens de communication et de synchronisation pouvant être utilisés à partir de programmes écrits en C. Quelques sources de programmes vous sont proposés afin d'accelerer votre familiarisation et éviter une grande consommation de votre temps dans la consultation de pages man. Chaque fichier source (*.c) est à survoler, à compiler et exécuter afin d'en comprendre le mécanisme.

Ouvrez votre session, placez vous dans ~dclot et récuperez les fichiers correspondant au patron MIAG_*.c. Il vous est conseillé de créer un répertoire cbin dans votre home et de l'ajouter à la variable PATH de votre environnement :

```
> mkdir ~/cbin
> export PATH=$PATH:~/cbin
> echo export PATH=\$PATH:\~/cbin >> ~/.profile
> cp ~dclot/pub/OSs/cours/tp/code/MIAG_*.c ~/cbin
```

Familiarisation avec `fork`, `pipe`, et `mkfifo`

- Prenez connaissance du contenu de MIAG_fork.c, compilez-le et exécutez le. Pour la compilation, utilisez la commande **cc**. Par exemple `cc -o truc.o source.c` permet de compiler le fichier source.c et de placer le résultat de la compilation dans truc.o qui est executable.
- Idem pour MIAG_pipe.c. Peut-être serait-il interessant de se reposer sur le script suivant pour faciliter les compilations :

  ```
  #!/bin/ksh
  cc -o ${1%c}o $1
  ```

  Placé dans le fichier comp avec les droits d'exécution, ce script vous permet d'utiliser la commande `comp MIAG_fork.c` par exemple.

- MIAG_pipe.c illustre les limites de l'utilisation d'un seul pipe pour la communication entre processus. Si l'on souhaitait faire répondre le père au fils une fois de plus, ce ne serait possible : supposons que le fils continue sa lecture du pipe, attendant un réponse de son père, les chances qu'il lise sa dernière réponse ne sont pas nulles. De même, l'inversion de ce dialogue n'est pas possible. La fonction pipe de notre système crée ce qui est appelé un pipe et plus précisément un "half duplex pipe". Certains systèmes permettent de créer un "full duplex pipe", i.e. un pipe permettant une communication dans les deux sens : chaque fichier correspondant au couple de descripteurs est ouvert en lecture/écriture. Un tel pipe peut être substitué par une paire paire de "half duplex pipes".
- Construire un programme qui, après avoir créé un couple de pipes, fork[1] afin que le père et son fils énumère en alternance les entiers de un à dix dans l'ordre croissant.
- Une des limites du pipe ou tube simple est qu'il rend possible la communication entre processus de filiation directe. Les tubes nommés ou FIFOs permettent à des processus à priori indépendants de communiquer selon le même mode que celui des pipes.
- Observez les mécanismes de gestion d'un couple de FIFOs dans MIAG_{FI,FO}.c. Ces programmes vont ensembles. Remarquez que le lancement de MIAG_FO.o seul conduit à une sortie rapide alors que le lancement de MIAG_FI.o suivi du lancement de MIAG_FO.o produit l'output attendu. Modifiez MIAG_FO.c de façon à observer à quel moment la sortie se produit puis arrangez son code de façon à s'assurer l'attente de l'ouverture des fifos. Enfin, inversez l'ordre des ouvertures dans l'un des codes et observez les conséquences engendrées.

[1] du verbe fork : je fork, tu fork, ...désolé pour ces anglicismes

★ Construisez un premier programme qui utilisera "/tmp/fifo.server" comme FIFO pour effectuer ses lectures après l'avoir créée. Les lignes qui seront lues seront constituées du PID du processus ayant écrit dans le tube, et d'une chaîne de caractères, les deux étant séparés par une virgule. Une fois la lecture faite, le programme affichera la chaîne de caractères et tentera l'ouverture de "/tmp/fifo.client.PID" afin d'y écrire un caractère, puis attendra la prochaine entrée dans son tube de lecture. Un second programme voué à jouer le rôle du client, procédera à l'ouverture de "/tmp/fifo.server" afin d'y écrire la chaîne "PID, PID was here!" (où PID est le PID du processus), créera "/tmp/fifo.client.PID" comme FIFO pour effectuer ses lectures et attendra l'arrivée d'une entrée dans ce tube pour le fermer, le supprimer et prendre fin.

ϟ☻ Pour finir, notez que FIFO et NFS ne font pas bon ménage. Seul le système de fichiers local permet la création d'une FIFO.

Familiarisation avec `kill` et `signal`

- Prenez connaissance du contenu de MIAG_{signal,signal2}.c, compilez-le et exécutez le.

Familiarisation avec les files de messages IPC

- Prenez connaissance des contenus de MIAG_msg{"",snd}.c et compilez-les. Observez la création de la file. La commande **ipcs** permet de lister les objets IPC actifs sur le système. Lancez MIAG_msg.o en le plaçant en arrière plan suivi de **ipcs**. Une fois que MIAG_msg.o a fini, vérifiez que la file créée a bien été supprimée avec la commande **ipcs**. Exécutez les deux programmes en parallèles pour observer leur échange de message.
- Contruisez deux programmes. Le premier initialise une file de messages, gènère un fils puis scrute les messages de type 1 et les imprime. Son fils scrute les messages de type 2 et envoie un signal SIGINT à l'expéditeur de chacun des messages qu'il reçoit. Le pid de l'expediteur est le contenu d'un message de type 2. Le second programme envoie le message de type 1 ayant pour contenu : "le processus de pid PID a laisse un msg", puis envoie un message de type 2 renfermant son pid et entre dans une boucle infinie. Vous lancerez plusieurs instances du second programme et une seule du premier.

Sémaphores

- Prenez connaissance du contenu de MIAG_sem_manip.c. Commentez les lignes concernant la suppression du tableau de sémaphores et compilez. Exécutez et observez le tableau créé avec **ipcs**. Supprimez vos commentaires du code pour retrouver le code initial et recompilez. Exécutez et vérifiez qu'à présent le tableau de sémaphores n'existe plus.
- Prenez connaissance des contenus des fichiers MIAG_sem_{1,2}_2.c et exécutez les codes objet associés.
- Construisez un programme créant un tableau d'un sémaphore et comportant une boucle infinie renfermant une section critique encadrée par l'écriture de message : "PID est hors de sa section critique". La section critique renferme un message signalant son entrée (resp. sortie) en section critique de la forme : "PID entre (resp. sort) dans (resp. de) sa region critique".

★ Proposez un programme qui simule le problème du producteur et du consommateur sur un nombre maximal de dix objets. Plus précisément, après la création du tableau de sémaphores nécessaires, une opération fork permettra d'une part au père de mettre en œuvre le comportement du producteur et d'autre part au fils de mettre en œuvre le comportement du consommateur. Pour un problème concret, il faudrait prévoir un partage des objets entre les deux processus (soit par le biais de segments de mémoire partagée, soit en faisant intervenir des threads en remplacant l'appel fork). Pour l'instant, nous souhaitons simplement travailler sur la production et la consommation d'objets virtuels : la production d'un objet consistera simplement à afficher le message : "Je produis le N-e objet." ou N est le nombre d'objets, et la consommation sera également limité à l'affichage du message : "Je consomme le N-e objet.". Il est également souhaité la production des messages suivants aux moments appropriés : "Production saturée", "Production relancée", "Consommation imposible" et "Consommation relancée".