IN4I13 – Systèmes d'exploitation

Communication inter-processus

Damien MASSON
http://www.esiee.fr/~massond/

Dernière modification: 13 mars 2018

- Damien Masson – Bureau 4206 – `http://www.esiee.fr/~massond`
- les transparents ne contiennent pas tout :
  - prendre des notes (relevées aléatoirement pour compilation et diffusion)
  - poser des questions

Pourquoi les communications inter-processus ?

- Processus indépendants :
  - Ils ne partagent pas de ressources avec d'autres processus, ils ne communiquent pas, ils n'affectent ni ne sont affectés par aucun autre processus
- Processus coopératifs :
  - Ils s'exécutent en parallèle sur un même ordinateur (monoprocesseur ou multiprocesseurs) ou sur des ordinateurs différents pour réaliser une tâche commune
  - Ils doivent s'échanger des informations
  - Ils partagent des ressources et communiquent entre eux

Mécanismes de communication

- Signaux
- Messages
- Mémoire partagée
- Fichiers partagés

Dans les deux derniers cas, on a le problème de données incohérentes qui peuvent résulter d'un accès simultané aux ressources

1. Signaux
   - Qu'est-ce qu'un signal ?
   - Appels système pour manipuler des signaux
   - Exemples

2. Tubes
   - Qu'est-ce qu'un tube ?
   - Tubes sans nom
   - Tubes nommés

Qu'est qu'un signal ?

- Interruptions logicielles
- Mécanisme de communication
- Un signal permet à un processus de réagir à un événement sans être obligé d'en tester en permanence l'arrivée
- Plusieurs signaux différents peuvent être reçus mais on n'empile pas le même signal (sauf pour les signaux temps réel)

Quand recoit-on un signal ?

- Situation d'erreur :
  - Exécution d'une instruction non autorisée (division par 0)
  - Des adresses non valides, etc.
- Interruption envoyée par l'usager
- Un autre processus veut annoncer l'occurrence d'un événement ou désire un comportement (le processus émetteur appartient au même usager ou au super-utilisateur)

Quoi faire à l'arrivée d'un signal ?

Les processus peuvent indiquer au système ce qui doit se passer à la réception d'un signal :

- Ignorer le signal
  - certains signaux ne peuvent être ignorés
- Le prendre en compte
  - en lui associant une routine de traitement
- Laisser le comportement par défaut
  - souvent, par défaut, on tue le processus

Quoi faire à l'arrivée d'un signal ?

- Pour chaque signal qu'un processus choisit de ne pas ignorer, on doit associer une procédure de gestion de signal
- Quand un signal arrive, la procédure associée est exécutée
- À la fin de l'exécution de la procédure, le processus reprend l'exécution à l'endroit où il a été interrompu

1. Signaux
   - Qu'est-ce qu'un signal ?
   - Appels système pour manipuler des signaux
   - Exemples

2. Tubes
   - Qu'est-ce qu'un tube ?
   - Tubes sans nom
   - Tubes nommés

Comment envoie-t-on un signal ?

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

- Le second argument est une constante symbolique qui représente le signal que l'on veut envoyer
  - Exemples : SIGKILL, SIGSTOP, SIGUSR1, SIGUSR2
  - Si on envoie 0, le système ne fait que vérifier si le processus existe
- La transmission du signal est effectuée par le Système d'Exploitation

Comment capturer un signal ?

```
sighandler_t signal(int signum, sighandler_t handler);
```

- On installe un gestionnaire en appelant la routine `signal`
  - signum est le signal que l'on veut capter
  - handler indique la fonction gestionnaire qui sera appelée lorsque le processus recevra ce signal
- La fonction gestionnaire recevra en paramètre le numéro du signal qui aura interrompu le processus
  - On peut ainsi associer le même gestionnaire à plusieurs signaux différents

Comportement à la réception d'un signal

- Chaque signal a un comportement par défaut
  - Exemple : SIGUSR1 et SIGUSR2 tuent le processus, SIGCHLD est ignoré
- Si un gestionnaire est installé, la routine de gestion est exécutée
  - Pour ignorer un signal, on passe SIG_IGN en second argument de `signal`
  - Pour rétablir le comportement par défaut, on passe SIG_DFL en second argument
- On peut également bloquer/débloquer la réception de signaux avec

```
int sigprocmask(int how, const sigset_t *set, sigset_t *oldset);
```

  - Un signal bloqué est mis en attente et traité lorsqu'on ne le bloque plus
- SIGKILL et SIGSTOP ne peuvent être ni ignorés, ni bloqués, ni captés par un gestionnaire

1. Signaux
   - Qu'est-ce qu'un signal ?
   - Appels système pour manipuler des signaux
   - Exemples

2. Tubes
   - Qu'est-ce qu'un tube ?
   - Tubes sans nom
   - Tubes nommés

Exemple 1

cf signal1.c

- ./signal1
- ps
- kill -SIGTERM ...
- ps
- kill -SIGUSR2 ...
- kill -SIGTERM ...
- ps

Exemple 2

cf signal2.c

- Conclusion : Les signaux ne permettent pas de synchroniser (finement) deux processus

1. Signaux
   - Qu'est-ce qu'un signal ?
   - Appels système pour manipuler des signaux
   - Exemples

2. Tubes
   - Qu'est-ce qu'un tube ?
   - Tubes sans nom
   - Tubes nommés

1. Signaux
   - Qu'est-ce qu'un signal ?
   - Appels système pour manipuler des signaux
   - Exemples

2. Tubes
   - Qu'est-ce qu'un tube ?
   - Tubes sans nom
   - Tubes nommés

Les tubes de communication

- Objectif : échanger des messages entre plusieurs processus
- Chaque message véhicule des données
- On peut se les imaginer comme des fichiers spéciaux
- 2 types de tubes :
  - Les tubes sans nom (unamed pipe)
  - Les tubes nommés (named pipe)

Communication unidirectionnelle avec tubes

schéma

Communication bidirectionnelle avec tubes

schéma

Tube

- Liaison unidirectionnelle de communication

Fonctionnement FIFO

- L'écriture dans un tube positionne les nouvelles données à la suite de celles qui s'y trouvent déjà
- Lorsqu'un processus lit dans un tube, il extrait les données positionnées au début

1. Signaux
   - Qu'est-ce qu'un signal ?
   - Appels système pour manipuler des signaux
   - Exemples

2. Tubes
   - Qu'est-ce qu'un tube ?
   - Tubes sans nom
   - Tubes nommés

Tube sans nom

- Les tubes sans nom peuvent être créés par
  - le shell
  - l'appel système

```
int pipe(int pipefd[2]);
```

- Leur taille varie d'une version à l'autre d'UNIX, mais vaut approximativement 4Ko

Tube sans nom du shell

- Les tubes sans nom du shell sont créés par l'opérateur binaire |
- Il dirige la sortie standard d'un processus vers l'entrée standard d'un autre processus
- Les tubes de communication du shell sont supportés par toutes les versions d'UNIX

Exemple : ls | wc -w

- Deux processus s'exécutent en parallèle :
  - Processus 1 : 'ls'
  - Processus 2 : 'wc -w'
  - Les résultats récupérés sur la sortie standard du premier processus sont dirigés vers l'entrée standard du second processus via le tube qui les relie
- Si le tube devient plein, le processus écrivain est suspendu jusqu'à ce que l'espace nécessaire soit libéré
- Si le tube devient vide, le processus lecteur est suspendu jusqu'à ce qu'il y ait suffisamment de données

Tubes POSIX

```
int pipe(int pipefd[2]);
```

- Les tubes sans nom peuvent être utilisés uniquement entre un processus parent et ses fils
- Un descripteur de fichier est un entier qui désigne l'endroit où on peut accéder à un fichier ouvert
- Identification : deux descripteurs de fichiers
  - Pour les lectures du tube : pipefd[0]
  - Pour les écritures dans le tube : pipefd[1]

Exemple de tube sans nom

schéma

Lire dans un tube POSIX

```
ssize_t read(int fd, void *buf, size_t count);
```

- Comportement
  - essaye de lire n octets dans le tube fd
  - copie les octets lus à l'adresse pointée par buffer
- Si le tube n'est pas vide et contient `taille` caractères : renvoie `min(taille, count)`
- Si le tube est vide
  - si le nombre d'ecrivains est nul, renvoie 0
  - si le nombre d'ecrivains est non nul, attente blocante
    - on peut aussi creer des tubes non bloquants avec `pipe2`, dans ce cas renvoie -1

Ecrire dans un tube POSIX

```
ssize_t write(int fd, const void *buf, size_t count);
```

- Atomique si `count < PIPE BUF`, la taille du tube sur le système (cf `<limits.h>`).
  - Garantie que les n caractères contenus dans buffer seront stockés contiguement dans le tube
- Si le nombre de lecteurs est nul envoi du signal SIGPIPE à l'écrivain.
- Sinon
  - Si l'écriture est bloquante, il n'y a retour que quand les n caractères ont été écrits dans le tube.
  - Si écriture non bloquante
    - Si $n > PIPE\_BUF$, retour avec un nombre inférieur à n (éventuellement -1) !
    - Si $n \leq PIPE\_BUF$ et si n emplacements libres, écrit et renvoie n (sinon retour -1 ou 0).

Echange de données par tube POSIX

- Flux de données unidirectionnel
- Pour une communication bidirectionnelle, il faut deux tubes
- Mécanisme avec capacité de stockage
- Lectures et écritures : attention avec les grosses tailles

Execution de la commande : ls | wc -w

Schema detaillé

Exemple 2

- Un processus crée un tube
- Ensuite, il crée deux fils, qui héritent automatiquement du tube
- Les deux fils s'en servent pour communiquer entre eux

cf fichier tube1.c

1. Signaux
   - Qu'est-ce qu'un signal ?
   - Appels système pour manipuler des signaux
   - Exemples

2. Tubes
   - Qu'est-ce qu'un tube ?
   - Tubes sans nom
   - Tubes nommés

Tubes de communication nommés

- Ils ont chacun un nom qui existe dans le système de fichiers (table des fichiers)
- Ils sont considérés comme des fichiers spéciaux
- Ils peuvent être utilisés par des processus indépendants, à condition qu'ils s'exécutent sur une même machine
- Ils existeront jusqu'à ce qu'ils soient supprimés explicitement
- Leur capacité maximale est d'environ 40K
- Ils sont créés par l'appel système mkfifo()

Remarques

- Lorsqu'on fait ll, p indique que c'est un tube
- Une fois le tube créé, il peut être utilisé pour réaliser la communication entre deux processus
- Pour ce faire, chacun des deux processus ouvre le tube :
  - L'un en mode écriture
  - L'autre en mode lecture

Exemple lecteur écrivain

cf codes prod.c cons.c

- Après avoir compilé séparément les deux programmes, il est possible de lancer leurs exécutions en arrière plan
- Les processus ainsi créés communiquent via le tube de communication mypipe