Systèmes et réseaux

Chapitre 3

Communication inter-processus

Pablo Rauzy <pr@up8.edu>
pablo.rauzy.name/teaching/sr

Communication inter-processus

- La *communication inter-processus* (*IPC*) est l'échange d'information ente processus.
- Cet échange consiste généralement en une synchronisation ou un envoie de données.
- Cela peut techniquement se passer à travers :
  - des fichiers ou une base de données bien sûr ;
  - une zone de mémoire partagées, dans le cas des processus légers ;
  - des signaux ;
  - des tubes.

- La *communication inter-processus* (*IPC*) est l'échange d'information ente processus.
- Cet échange consiste généralement en une synchronisation ou un envoie de données.
- Cela peut techniquement se passer à travers :
  - des fichiers ou une base de données bien sûr ;
  - une zone de mémoire partagées, dans le cas des processus légers ;
  - des signaux ;
  - des tubes.

Signaux

- Un *signal* est un message envoyé de manière asynchrone à un processus.
- Il sert généralement à notifier d'un évènement pour permettre son traitement.
- Sa réception interrompt le flot d'exécution normal du processus.
  - C'est le système qui interrompt le processus pour lancer le gestionnaire du signal de ce dernier.
  - À ne pas confondre avec les *interruptions* matérielles (gérées par le CPU et le système).

Les signaux POSIX

- Quelques signaux :
  - `SIGINT` : interruption (Ctrl+C), termine le processus.
  - `SIGKILL` : tue immédiatement le processus, non-interceptable.
  - `SIGTERM` : tue immédiatement le processus.
  - `SIGSTOP` : interrompt le processus, non-interceptable.
  - `SIGTSTP` : interruption reçue du terminal (Ctrl+Z), voir aussi `SIGCONT`.
  - `SIGSEGV` : erreur de segmentation, termine le processus avec *core dump*.
  - `SIGHUP` : rupture terminal ou processus parent, termine le processus.
  - `SIGCHLD` : processus enfant terminé ou stoppé.
  - `SIGUSR1` et `SIGUSR2`.
- D'autres signaux :
  - `https://fr.wikipedia.org/wiki/Signal_(informatique)#Liste_des_signaux`

Envoyer un signal

- Pour envoyer un signal, on utilise l'appel système `kill`.
  - `int kill(pid_t pid, int sig);`
  - `#include <signal.h>`
- Se mettre un timer avec `SIGALRM` :
  - `unsigned int alarm(unsigned int seconds);`
  - `#include <unistd.h>`

Interruption et traitement

- On peut changer le gestionnaire d'un signal.
- Pour ignorer un signal, ou remettre le gestionnaire par défaut :
  - `sighandler_t signal(int signum, sighandler_t handler);`
  - `#include <signal.h>`
  - où `handler` peut être `SIG_IGN` ou `SIG_DFL`.
- Pour installer un gestionnaire alternatif :
  - ```
    int sigaction(int signum,
                  const struct sigaction *_Nullable restrict act,
                  struct sigaction *_Nullable restrict oldact);
    ```
  - `#include <signal.h>`
  - où la structure `sigaction` est définie comme :

```
struct sigaction {
  void     (*sa_handler)(int);
  void     (*sa_sigaction)(int, siginfo_t *, void *);
  sigset_t   sa_mask;
  int        sa_flags;
  void     (*sa_restorer)(void);
};
```
```

- Toujours dans la philosophie de “tout est fichier”, il est possible d’obtenir un descripteur de fichier permettant la gestion d’un signal avec l’appel système **`signalfd`** :
  - `int signalfd(int fd, const sigset_t *mask, int flags);`
  - `#include <sys/signalfd.h>`

Tubes

- Un *tube* (*pipe* en anglais) est un moyen d'envoyer des données d'un processus à un autre.
- Il existe deux types de tubes :
  - les tubes nommés,
  - les tubes anonymes.

Tubes nommés (PEPS)

- Les *tubes nommés* sont des fichiers particuliers, dans lequel on peut lire et écrire selon le modèle “premiers entrés, premiers sortis” (*first in, first out*, FIFO).
- Sur ce type de fichier, les appels de lecture et écriture vont être bloquants de sorte à synchroniser la communication.
  - Regardons ensemble ce comportement avec la commande `mkfifo`.
- On peut créer ce genre de tube avec la fonction `mkfifo` :
  - `int mkfifo(const char *pathname, mode_t mode);`
  - ```
    #include <sys/types.h>
    #include <sys/stat.h>
    ```
    ```

Tubes anonymes

- Un *tube anonyme* est une paire de descripteur de fichiers fourni par le système :
  - l'un est accessible en lecture et l'autre en écriture,
  - le système garde en mémoire tampon ce qui est écrit jusqu'à ce que ce soit lu.
- L'appel système `pipe` permet de créer un tube :
  - `int pipe(int pipefd[2]);`
  - `#include <unistd.h>`

- Les tubes anonymes permettent de créer des filtres sur la ligne de commande :
  - une ligne de commande peut être une *pipeline*,
  - elle est alors composée de plusieurs commandes qui sont chaînées à l'aide de pipes reliant la sortie standard de chaque commande à l'entrée standard de la suivante.
- Pour créer ces redirections, on recourt à l'appel système `dup2` (conjointement avec `fork` et `execve` que l'on a déjà vu) :
  - `int dup2(int oldfd, int newfd);`
  - `#include <unistd.h>`