Programmation système
Multi processing

Florent Devin

EISTI

3 février 2009

Plan

Programme / Processus

- Programme : fichier binaire inerte
- Processus : programme en cours d'exécution dans un environnement
- PID : Numéro unique d'un processus
- PPID : Numéro du processus père
- État du processus : géré par l'ordonnanceur
- Processus : utilise des ressources allouées par le SE
- Gestion arborescente des processus

Format d'un exécutable

- Un en-tête : décrit l'ensemble du fichier, carte, et section
- Taille à allouer
- Section text : code en langage machine
- Section data : données initialisées
- D'autres sections : symbole de débuggage, images, ...

Action du SE

- Création/suppression de processus
- Suspension/reprise d'un processus (ordonnancement)
- Synchronisation de processus
- Communication inter-processus
- Traitement des interblocages

Élément constitutif d'un processus

- Section de texte : code du programme
- Process Block Control (PCB)
  - Compteur d'instructions
  - Contenu des registres
  - Pile d'exécution
  - PID
  - Fichiers ouverts
  - État du processus
  - ...
- Section de données : variables du programme

État d'un processus

- Nouveau (en cours de création)
- En exécution (par le processeur)
- En attente (attente "active" d'un événement)
- Prêt (potentiellement exécutable par le processeur)
- Terminé

Intérêt

- Multiplicité des exécutions
- Protection des exécutions (impossibilité d'exécuter un autre code que le sien)
- Communication possible grâce aux appels systèmes
- Gain de temps de traitement
- Meilleure utilisation des machines mutliprocesseurs
- Problème d'accès concurrents

Création d'un nouveau processus

- Création d'un nouveau processus impossible
- Duplication d'un processus possible
- Appel système :

```
#include <sys/types.h>
#include <unistd.h>

pid_t fork (void);
```

Qui est qui ?

- Père : processus qui exécute `fork`
- Fils : processus résultant de la duplication
- Un seul père pour un processus
- Plusieurs fils possibles (notion d'arborescence)
- Valeur de `fork`
  - 0 : processus fils
  - -1 : erreur dans la duplication
  - nb : processus père (permet de connaître le numéro du fils)

Attributs hérités

- Duplication exacte ? Pas tout à fait...
- Pas d'héritage de
  - PID : différent de celui du père
  - PPID : PID du père
  - verrouillage mémoire (`mlock`, `mlockall`)
  - statistiques d'utilisation des ressources
  - signaux en attente
  - des ajustements de sémaphores (`semop`)
  - `man fork` pour plus de détails
- Hérite d'une copie des descripteurs de fichiers, état, positionnement

Cas spécifique des descripteurs de fichiers

- Après un `fork` : entrée de la table de `fd` pointe sur la *même* table des fichiers
- Nombre de descripteurs de la table des fichiers : incrémenté
- Offset commun aux deux processus

Exemple

```
pid_proc = fork ();
switch (pid_proc) {
   case -1 : // Cas de l'erreur
      .../... break;
   case 0 :  // Code du fils
      .../... break;
   default :  // Code du père
      .../... break;
}
```

Combien de processus

```
int main (int argc, char** argv)
    {
  fork ();
  fork ();
  fork ();

  return (0);
}
```

Terminaison d'un processus

- Fin d'un processus : fin de la fonction `main`
- Deux primitives pour arrêter l'exécution :

```
#include <unistd.h>

void _exit (int status);
void exit (int status);
```

- `status` : code de retour (entre 0 et 255) transmis au père
- Par convention : terminaison normale ⇒ `status = 0`

Identité d'un processus

- Processus : PID
- Possibilité d'accéder à son PID, mais aussi à celui de son père

```
#include <sys/types.h>
#include <unistd.h>

pid_t getpid(void);
pid_t getppid(void);
```

Attente de terminaison

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait (int *status) ;
```

- `wait` : suspend l'exécution du processus appelant, jusqu'à la terminaison d'un de ses fils.
- Si fils déjà fini, `wait` retourne immédiatement
- Retourne le PID du fils, ainsi que la valeur de retour du fils (si `status != NULL`)

Information sur la terminaison

- `WIFEXITED (status)` : vrai si terminaison normale du fils
- `WEXITSTATUS (status)` : renvoie la valeur du fils, si `WIFEXITED (status)`
- `WIFSIGNALED (status)` : vrai si terminaison par un signal
- `WTERMSIG (status)` : valeur du signal
- Plus d'info : `man 2 wait`

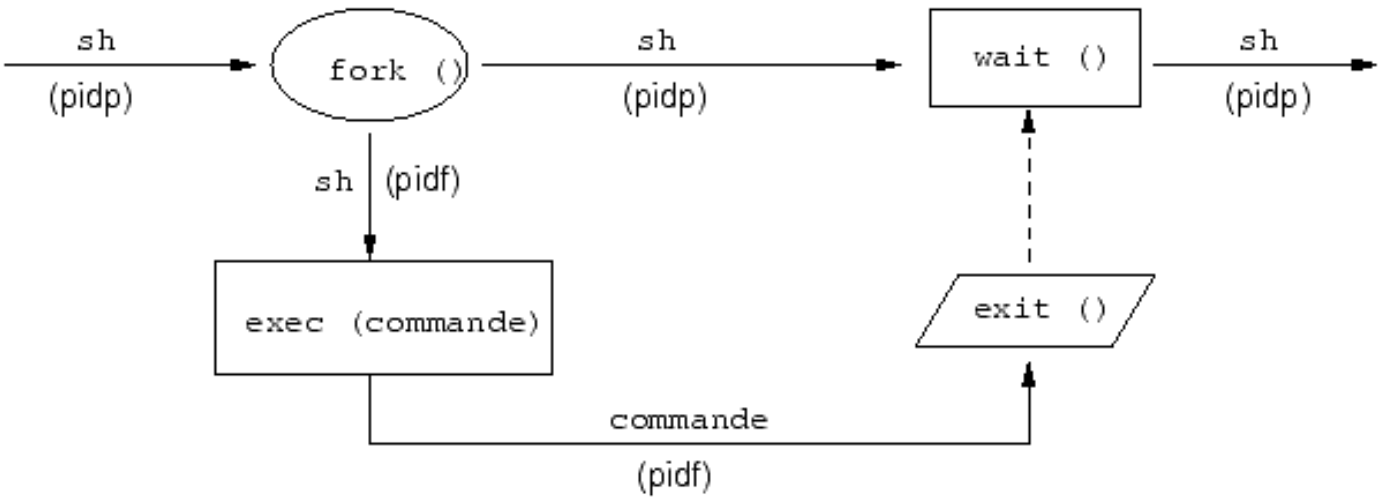
Changement de code

- Famille des `exec[lv][ ep]`
- Permet de changer le code d'un processus par un autre (et donc l'exécution)
- Desallocation des sections, et allocation des nouvelles : géré par le SE
- Peu d'intérêt d'utiliser `fork` sans `exec`
- Très peu d'intérêt d'utiliser `exec` sans `fork`

Utilisation classique

- Fabrication d'un shell
- Attente de commandes par le processus père
- Dès réception d'une commande : création d'un processus fils
- Changement du code par la commande demandée
- Dès terminaison du fils, récupération du résultat
- Affichage du prompt, et attente

Exemple


sh
(pidp)
fork ()
sh
(pidp)
wait ()
sh
(pidp)
sh
(pidf)
exec (commande)
exit ()
commande
(pidf)