NFA003 : L'appel système fork

Amélie Lambert

Sources : cours de NSY103 de Joëlle Delacroix

2013-2014

Création de processus

Primitive de création de processus (1/6)

```
#include <unistd.h>
pid_t fork(void)
```

- La primitive `fork()` permet la création dynamique d'un nouveau processus qui s'exécute de manière concurrente avec le processus qui l'a créé.
- Tout processus Unix/Linux hormis le processus 0 est crée à l'aide de cette primitive.
- Le processus créateur (le père) par un appel à la primitive `fork()` crée un processus fils qui est une copie exacte de lui-même (code et données) (Exemple d'utilisation : programmation d'une application parallèle).
- Il est également possible de remplacer le code et les données du processus créé par ceux d'un autre programme, via les primitives de recouvrement.

Primitive de création de processus (2/6)

MODE UTILISATEUR

PERE (pid :12222)

```
main () {
   pid_t ret ;
   int i, j=0;
   for(i=0; i<8; i++)
      i = i + j;
   ret = fork();
}
```

MODE SYSTEME

Exécution de l'appel système `fork()`. Si les ressources noyau sont disponibles :

1. allouer une entrée de la table des processus au nouveau processus
2. allouer un pid unique au nouveau processus
3. dupliquer le contexte du processus parent (code, données, pile)
4. retourner le pid du processus crée à son père et 0 au processus fils

Rappel : un processus est identifié par un numéro unique appelé `pid`

Primitive de création de processus (3/6)

MODE UTILISATEUR

PERE (pid :12222)

```
main () {
   pid_t ret ;
   int i, j=0;
   for(i=0; i<8; i++)
      i = i + j;
   ret = fork();
   // ret = 12223
}
```

FILS (pid :12223)

```
main () {
   pid_t ret ;
   int i, j=0;
   for(i=0; i<8; i++)
      i = i + j;
   ret = fork();
   // ret = 0
}
```

MODE SYSTEME

Lors de l'exécution de l'appel système `fork()`, si les ressources noyau sont disponibles, il est retourné :

- le pid du processus crée au processus père (ici 12223) ;
- la valeur 0 au processus fils.

Primitive de création de processus (4/6)

Le mécanisme du `copy on write` (copie sur écriture) :

Lors de sa création, le fils partage avec son père :

- le segment de code ;
- le segment de données et la pile sont également partagés et mis en accès lecture seule.

Lors d'un accès en écriture par l'un des deux processus, le segment de données ou la pile sont effectivement dupliqués.

Primitive de création de processus (5/6)

PERE (pid :12222)

```
main () {
  pid_t ret ;
  int i, j=0;
  for(i=0; i<8; i++)
    i = i + j;
  ret = fork();
  // ret = 12223
}
```

FILS (pid :12223)

```
main () {
  pid_t ret ;
  int i, j=0;
  for(i=0; i<8; i++)
    i = i + j;
  ret = fork();
  //ret = 0
}
```

- Chaque processus père et fils reprend son exécution après le `fork()`.
- Le code et les données étant strictement identiques, il est nécessaire de disposer d'un mécanisme pour différencier le comportement des deux processus après le `fork()`
- On utilise pour cela le code retour du `fork()` qui est différent chez le fils (toujours 0) et le père (pid du fils crée)

Primitive de création de processus (6/6)

PERE (pid :12222)

```
main () {
  pid_t ret ;
  int i, j=0;
  for(i=0; i<8; i++)
    i = i + j;
  ret= fork();
  if (ret == 0)
    printf("␣je␣suis␣le␣fils␣")
  else {
    printf ("je␣suis␣le␣père");
    printf ("pid␣de␣mon␣fils␣%d", ret)
  }
}
```

Pid du fils : 12223
getpid : 12222
getppid :shell

FILS (pid :12223)

```
main () {
  pid_t ret ;
  int i, j=0;
  for(i=0; i<8; i++)
    i = i + j;
  ret= fork();
  if (ret == 0)
    printf("␣je␣suis␣le␣fils␣")
  else {
    printf ("je␣suis␣le␣père");
    printf ("pid␣de␣mon␣fils,␣%d" , ret)
  }
}
```

getpid : 12223
getppid : 12222

Le processus fils hérite de tous les attributs de son père sauf son pid.
Les exécutions des processus père et fils peuvent s'enchevêtrer en fonction de la politique d'ordonnancement.

Synchronisation père / fils et notion de primitive de recouvrement

Synchronisation père / fils (1/3)

```
#include <stdlib.h>
void exit (int valeur);
pid_t wait (int *status);
```

- un appel à la primitive `exit()` provoque la terminaison du processus effectuant l'appel avec un code retour valeur. (Par défaut, le franchissement de la dernière `}` d'un programme C tient lieu d'`exit()`)
- un processus qui se termine passe dans l'état Zombie et reste dans cet état tant que son père n'a pas pris en compte sa terminaison.
- Le processus père "récupère" la terminaison de ses fils par un appel à la primitive `wait ()`

Synchronisation père / fils (2/3)

PERE (pid :12222)

```
main () {
   pid_t ret ;
   int i, j=0;
   for(i=0; i<8; i++)
      i = i + j;
   ret= fork();
   if (ret == 0) {
      printf("␣je␣suis␣le␣fils␣");
      exit();
   }
   else {
      printf ("je␣suis␣le␣père");
      printf ("pid␣de␣mon␣fils,%d",ret);
      wait();
   }
}
```

FILS (pid :12223)

```
main () {
   pid_t ret;
   int i, j=0;
   for(i=0; i<8; i++)
      i = i + j;
   ret= fork();
   if (ret== 0) {
      printf("␣je␣suis␣le␣fils␣");
      exit ();
   }
   else {
      printf ("je␣suis␣le␣père");
      printf ("pid␣de␣mon␣fils,%d",ret);
      wait();
   }
}
```

Synchronisation père / fils (3/3)

- Lorsqu'un processus se termine (`exit()`), le système démantèle tout son contexte, sauf l'entrée de la table des processus le concernant.
- Le processus père, par un `wait()`, "récupère" la mort de son fils, cumule les statistiques de celui-ci avec les siennes et détruit l'entrée de la table des processus concernant son fils défunt. Le processus fils disparaît complètement.
- Un processus fils défunt reste zombie jusqu'à ce que son père ait pris connaissance de sa mort
- Un processus fils orphelin, suite au décès de son père (le processus père s'est terminé avant son fils) est toujours adopté par le processus 1 (init).

Primitive de recouvrement

Il s'agit d'un ensemble de primitives (famille `exec()` permettant à un processus de charger en mémoire, un nouveau code exécutable (`execl`, `execlp`, `execle`, `execv`, `execvp`, `execve`).

```
main () {
  pid_t pid ;
  int i, j=0;
  for(i=0; i<8; i++)
    i = i + j;
  pid= fork();
  if (pid == 0) {
    printf("␣je␣suis␣le␣fils␣");
    execl("/home/calcul","calcul","3","4", NULL);
    // execution du programme calcul avec les
    // pramètres 3 et 4
  }
  else {
    printf ("je␣suis␣le␣père");
    printf ("pid␣de␣mon␣fils,␣%d" , pid);
    wait();
  }
}
```

1. Création d'un processus fils par duplication du code et données du père ;
2. Le processus fils recouvre le code et les données hérités du père par ceux du programme calcul. Le père transmet des données de son environnement vers son fils par les paramètres de l'`exec(...)`
3. Le père attend son fils