osj

April 9, 2025

Contents

0.1 Introduction

0.2 Process Creation and Termination

One process, referred to as the parent, can create another process, known as the child. This is achieved by allocating and initializing a new Process Control Block (PCB). As an exercise, you can run the command 'ps auxwww' in the shell; PPID is the parent's Process ID (PID).

0.2.1 Inheritance in POSIX

In POSIX, a child process inherits most of the parent's attributes. These include the User ID (UID), open files, current working directory (cwd), and so on. It is important to close any unneeded open files. Can you think of why this might be necessary?

0.2.2 Process Control Block Movement

While a process is executing, its PCB moves between different queues. The movement is determined by the state change graph. The queues can be runnable, sleep/wait for event i (where i can be any integer).

0.2.3 Process Termination

After a process dies, either by calling exit() or being interrupted, it becomes a zombie. The parent process uses the wait* system call to clear the zombie from the system. Can you think of why this is necessary? The wait system call family includes wait, waitpid, waitid, wait3, and wait4. Here is an example of how to use wait4:

0.2.4 Parent Process Behavior

The parent process can either sleep/wait for its child to finish or run in parallel. The wait*() function will block unless WNOHANG is given in 'options'. As an exercise, try reading 'man 2 wait'.

```

  \begin{tcolorbox}[    colback=yellow!10,    colframe=yellow,    title={\
    fontfamily{lmr}\selectfont \faBookmark À retenir},    fonttitle=\bfseries,
    fontupper=\fontfamily{lmr}\selectfont,    boxrule=1pt,    sharp corners,  ]  A
     parent process can create a child process by allocating and initializing a
    new PCB. In POSIX, the child process inherits most of the parent's attributes.
     The PCB of a process moves between different queues during its execution.
    When a process dies, it becomes a zombie and the parent process uses the wait*
     system call to clear it from the system.  \end{tcolorbox}

\section{Création d'un processus enfant avec fork()}
 La fonction fork() est utilisée pour créer un nouveau processus, appelé processus
     enfant, à partir du processus appelant, appelé processus parent. Cette
```

```
fonction initialise un nouveau Bloc de Contrôle de Processus (PCB) basé sur la valeur du processus parent et l'ajoute à la file d'attente exécutable.
\subsection{Fonctionnement de fork()}
Lorsqu'un processus appelle fork(), deux processus sont maintenant présents au même point d'exécution. L'espace d'adresse du nouveau processus enfant est une copie complète de l'espace du processus parent, avec une différence : la fonction fork() retourne deux fois. Dans le processus parent, elle retourne l'identifiant du processus enfant (pid > 0), tandis que dans le processus enfant, elle retourne 0.
\subsection{Gestion des erreurs}
Si la fonction fork() échoue, elle retourne une valeur négative. Dans ce cas, une chaîne associée à la variable globale 'errno' est imprimée. 'errno' contient le numéro d'erreur du dernier appel système.
\subsection{Ordre d'impression}
L'ordre d'impression n'est pas déterminé et dépend de l'ordonnanceur du système d'exploitation. Il peut varier d'une exécution à l'autre.
\begin{lstlisting}

int main(int argc, char *argv[])
{
  int pid = fork();
  if( pid==0 ) {
   //
   // child
      //
      printf('parent=%d son=%d\n',
             getppid(), getpid());
  }
  else if( pid > 0 ) {
      //
      // parent
      //
      printf('parent=%d son=%d\n',
             getpid(), pid);
  }
  else { // print string associated
         // with errno
      perror('fork() failed');
  }
  return 0;
}
```

À retenir

La fonction fork() crée un nouveau processus en initialisant un nouveau Bloc de Contrôle de Processus (PCB) basé sur la valeur du processus parent. Elle retourne deux fois : dans le processus parent avec l'identifiant du processus enfant (pid > 0) et dans le processus enfant avec 0. Si fork() échoue, elle retourne une valeur négative et imprime une chaîne associée à 'errno', qui contient le numéro d'erreur du dernier appel système.