osj

April 9, 2025

Contents

0.1 Introduction

0.2 Process Creation and Termination

In an operating system, one process, which we can refer to as the parent, can create another process, known as the child. This is done by allocating and initializing a new Process Control Block (PCB).

Simple Explanation

Think of the PCB as a sort of ID card for the process. It contains all the information the operating system needs to manage the process.

0.2.1 Parent and Child Processes

In POSIX, a child process inherits most of the parent's attributes such as the User ID (UID), open files, current working directory (cwd), etc. However, if these inherited files are not needed, they should be closed.

0.2.2 Process Control Block Movement

While a process is executing, its PCB moves between different queues according to the state change graph. These queues can be runnable, sleep/wait for event i (where i can be any number).

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.

0.3 Parent-Child Process Interaction

The parent can either wait for its child to finish or run in parallel. The wait*() function will block unless WNOHANG is given in 'options'.

0.3.1 Homework

Run 'ps auxwww' in the shell to see the PPID, which is the parent's PID. Also, read 'man 2 wait' to understand more about the wait system call family: wait, waitpid, waitid, wait3, wait4.

```

\begin{tcolorbox}[ colback=blue!10, colframe=blue, title={\fontfamily{lmr}\
    selectfont \faComment\ Vulgarisation simple}, fonttitle=\bfseries, fontupper
    =\fontfamily{lmr}\selectfont, boxrule=1pt, sharp corners,]
```

```
Imaginez que vous êtes en train de cuisiner et que vous décidez de faire une
    recette qui nécessite de préparer deux choses en même temps. Vous pourriez
    faire appel à un assistant pour vous aider. Dans ce cas, vous (le processus
    parent) donneriez à votre assistant (le processus enfant) une copie de la
    recette (l'espace d'adresse) et vous continueriez à travailler en parallèle.
    Si votre assistant rencontre un problème (fork() échoue), il vous le fait
    savoir (la variable 'errno' contient le numéro d'erreur).
\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 actuel, 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 des processus prêts à être
     exécutés.

\subsection{Fonctionnement de fork()}

Lorsque fork() est appelé, il crée un nouvel espace d'adresse pour le processus
    enfant qui est une copie complète de l'espace d'adresse du processus parent, à
     une différence près : la valeur de retour de fork(). En effet, fork()
    retourne deux fois : une fois dans le processus parent avec une valeur
    supérieure à zéro (le PID du processus enfant) et une fois dans le processus
    enfant avec une valeur de zéro.

\subsection{Gestion des erreurs avec fork()}

Si fork() échoue, il retourne une valeur négative. Dans ce cas, la variable
    globale 'errno' contient le numéro d'erreur du dernier appel système. Cette
    variable peut être utilisée pour afficher un message d'erreur approprié avec
    la fonction perror().
\begin{lstlisting}[language=C]
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;
}
```