osj

April 9, 2025

Contents

0.1 Introduction 2
0.2 Process Creation and Termination 2
0.2.1 Inheritance in POSIX 2
0.2.2 PCB Movement 2
0.2.3 Process Termination 2

0.1 Introduction

0.2 Process Creation and Termination

One process, referred to as the parent, can create another process, referred to as the child. This is achieved through the allocation and initialization of a new Process Control Block (PCB). As a homework assignment, you can run the command `'ps auxwww'` in the shell to see this in action. The PPID displayed is the parent's 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, and the current working directory (cwd), among others. It is important to close any unneeded open files. Can you think of why this might be necessary?

0.2.2 PCB Movement

While a process is executing, its PCB moves between different queues. This movement is according to the state change graph. The queues include the runnable queue and the sleep/wait for event i queue, where i can be any integer.

0.2.3 Process Termination

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

```

\begin{tcolorbox}[
  colback=green!10,
  colframe=green,
  title={\fontfamily{lmr}\selectfont \faLightbulb Intuition},
  fonttitle=\bfseries,
  fontupper=\fontfamily{lmr}\selectfont,
  boxrule=1pt,
  sharp corners,
]
 La fonction fork() crée un nouveau processus en dupliquant le processus existant.
     Le processus enfant obtient une copie exacte de tous les segments de mémoire
    du processus parent. Cependant, le processus enfant a son propre espace d'
    adressage et ne partage pas la mémoire avec le processus parent.
\end{tcolorbox}
\section{Création d'un processus enfant avec fork()}
 La fonction fork() est utilisée pour créer un nouveau processus, appelé processus
     enfant, en initialisant un nouveau Bloc de Contrôle de Processus (PCB). Le
```

PCB est basé sur la valeur du processus parent et est ajouté à la file d'attente exécutable.

Après l'appel de fork(), il y a maintenant deux processus au même point d'exécution. Le nouvel espace d'adresse de l'enfant est une copie complète de l'espace du parent, avec une différence : fork() retourne deux fois. Dans le processus parent, il retourne avec pid>0 et dans le processus enfant, il retourne avec pid=0.

'errno' est une variable globale qui contient le numéro d'erreur du dernier appel système. Si fork() échoue, une chaîne associée à 'errno' est imprimée.

Un point d'interrogation demeure : quel est l'ordre d'impression ?

```
\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;
 }
```