% Configuration des listings

% Configuration des tcolorbox

% Protection des underscores dans le texte

% Configuration des marges

osj

April 9, 2025

Contents

0.1 Introduction

% Cours : osj Process creation & termination • One process (the “parent”) can create another (the “child”) – A new PCB is allocated and initialized – Homework: run ‘ps auxwww’ in the shell; PPID is the parent’s PID • In POSIX, child process inherits most of parent’s attributes – UID, open files (should be closed if unneeded; why?), cwd, etc. • While executing, PCB moves between different queues – According to state change graph – Queues: runnable, sleep/wait for event i (i=1,2,3…) • After a process dies (exit()s / interrupted), it becomes a zombie – Parent uses wait* syscall to clear zombie from the system (why?) – Wait syscall family: wait, waitpid, waitid, wait3, wait4; example: – pid_t wait4(pid_t, int *wstatus, int options, struct rusage *rusage); • Parent can sleep/wait for its child to finish or run in parallel – wait*() will block unless WNOHANG given in ‘options’ – Homework: read ‘man 2 wait’ OS (234123) - processes & signals 7

Vulgarisation simple

Imagine the fork() function as a factory duplicator. It takes a model (the parent process) and creates a new, identical product (the child process). Both the model and the product continue their work from the point where the duplication happened. The only way to tell them apart is by the ID (pid) the factory assigns to them. The model (parent) gets the ID of the product (child), and the product (child) gets the ID 0. If the duplication fails, the factory (fork function) signals an error.

0.2 OS (234123) - Processes & Signals

0.2.1 fork() - Spawn a Child Process

The function fork() is used to create a new process, which becomes the child of the caller. After a new child process is created, both processes will execute the next instruction following the fork() system call. Therefore, we have to distinguish the parent from the child. This can be done using the following code:

- The function fork() initializes a new Process Control Block (PCB) based on the parent’s value. The PCB is then added to the runnable queue.
- Now, there are two processes at the same execution point.
- The child’s new address space is a complete copy of the parent’s space, with one difference: fork() returns twice. In the parent process, fork() returns the child’s pid, and in the child process, fork() returns 0.
- The order of printing is not determined. It depends on the order in which the processes are scheduled.
- ’errno’ is a global variable that holds the error number of the last system call.

eginlstlisting

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