UNIX FORK Summary

kl4227 Lin, Kuan-You

In an unix-like environment, the fork function provides the users with the ability to proactively generate a child process. When the fork function is being called, a new process is created by duplicating the original process resources. The newly created process will be referred to as child process while the original process will be its parent process. The two processes will then run independently in their own memory spaces, namely the rewrite of the sequential variables in one process does not affect the other process. However, the child process resets the CPU time and resources utilization, and does not inherit parent's memory locks, semaphore adjustment, timer, outstanding asynchronous I/O operations, process-associated record locks, pending signal, and port access permission bits set by ioperm() function.

The child process receives no signal when its corresponding parent process terminates, but gives off a termination signal SIGCHLD. The child process has its own unique ID; the getpid() function will return the current process id.

The fork() function in C / C++ will also returns a value typed pid_t, indicating the status of such a calls. The value can be any of the following:

1. -1 : when the fork() function fails
2. 0 : indicates that the current process is a child process
3. >0 : indicates that the current process is a parent process

The following snippet prints out the result of different process

```cpp
int main() {
    pid_t id = fork();
    switch (id) {
        case -1:
            cout << "The fork fails" << endl;
            exit(-1);
        case 0:
            cout << "Currently running a child process, ID: " << getpid() << endl;
            sleep(3);
            break;
        default:
            cout << "Currently running a parent process, ID: " << getpid() << endl;
            break;
    }
    return 0;
}
```

Also, I think it is noteworthy to mention zombie process. If I add sleep(3) to the default section of the above snippet. The parent process will return 0 and close before child process finish its work. In the following message log, we can see a zombie child process which does nothing but take up the resources. We can cope with it by adding signal() or wait() function, but I think it is not in the scope of this assignment. In a nutshell, we either put the parent process in wait or we transfer the termination to the signal.

```
Currently running a parent process, ID: 52439
Currently running a child process, ID: 52440
lin-kuan-you@linguanyous-MacBook-Pro ~ % ps
  PID TTY           TIME CMD
52389 ttys000    0:00.08 -zsh
52440 ttys000    0:00.00 /Users/lin-kuan-you/Documents/School/NYU Bridge/Home W
lin-kuan-you@linguanyous-MacBook-Pro ~ %
```