The Process API

Sachidananda Urs

July 21, 2021

Abstract

This document briefly discusses the concepts of fork(2), wait(2), and exec(2) syscalls, how to create a process using them and the interfaces.

The *fork* system call

Best way to learn about the fork system call (apart from manual page) is to write a program to print "Hello world" in the parent and child process and And to observe their return codes along with their pids.

Note that the output is not deterministic, i.e the message printed by parent might be first or second, need not be first always.

When the parent creates the child, we have two processes. And which one gets scheduled first is determined by the scheduler. This creates a window for "non-determinism". The *non-determinism* leads to some interesting problems, especially in multi-threaded programs.

The *wait(2)* syscall

Upon creation of the child, the parent process can be made to wait till the child completes, this is achieved by the wait(2) syscall.

The *exec(2)* system call

The final and important piece of process creation API is the emphexec() call. The *fork()* call creates a new process (child) and the *exec* call in the child overlays the program on top of child. The heap and stack of child are reinitialized, it transforms the current program into a new one.

Motivation for the fork-exec API

[*Why not bundle fork and exec into a single call?*]

Keeping the *fork* and *exec* separate gives better control for the programmer by allowing him to do any housekeeping work before calling the *exec.*

One of the useful applications of *fork+exec* is the *unix shell.* This allows the shell to run code before the *exec()* call, and allows it to do a alot of intersting stuff before the actual program is called by the *exec* (for example to alter the environment).

Consider the shell redirection

$ cat hello.txt > foo.txt

Before the shell *execs "cat"* it closes the stdout and opens foo.txt. The open assigns the *fd 0* to foo.txt. Since the Unix systems assigns free file descriptors, 0 was free upon closing stdout and thus 0 was assigned to foo.txt. Thus the output was sent to foo.txt instead of stdout.

Process Control

Look up *kill* and *signal.*