ECE 353: Systems Software
Lecture 4

Process Creation

1.0.0

Jon Eyolfson
January 16, 2023

We Could Create Processes from Scratch

We load the program into memory and create the process control block

This is what Windows does

Unix decomposes process creation into more flexible abstractions

Instead of Creating a New Process, We Could Clone It

Pause the currently running process, and copy it's PCB into a new one

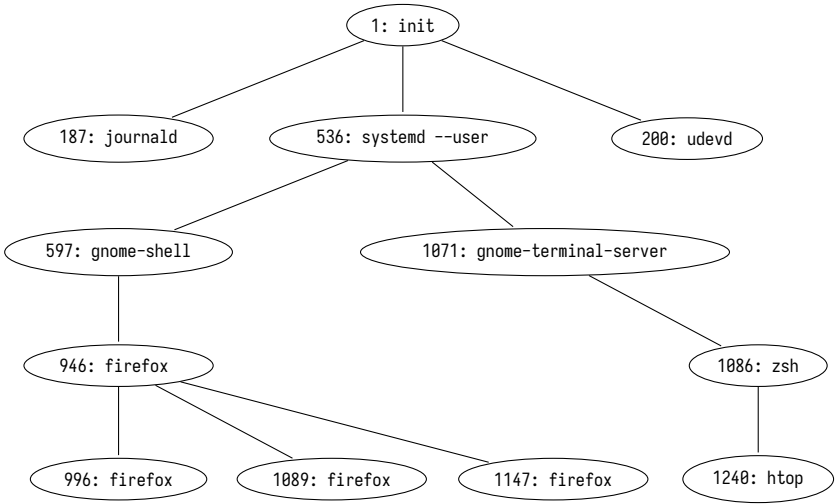
- This will reuse all of the information from the process, including variables!

Distinguish between the two processes with a parent and child relationship

- They could both execute different parts of the program together

We could then allow either process to load a new program and setup a new PCB

A Typical Process Tree on the Virtual Machine



How You Can See Your Process Tree

Use `htop`

You can press F5 to switch between tree and list view

Linux Terminology Is Slightly Different

You can look at a process' state by reading `/proc/<PID>/status | grep State`

Replace `<PID>` with the process ID (or `self`)

R: Running and runnable [Running and Waiting]
S: Interruptible sleep [Blocked]
D: Uninterruptible sleep [Blocked]
T: Stopped
Z: Zombie

The kernel lets you explicitly stop a process to prevent it from running

You or another process must explicitly continue it

On Unix, the Kernel Launches A Single User Process

After the kernel initializes, it creates a single process from a program

This process is called `init`, and it looks for it in `/sbin/init`

- Responsible for executing every other process on the machine
- Must always be active, if it exits the kernel thinks you're shutting down

For Linux, `init` will probably be `systemd` but there's other options

Aside: some operating systems create an "idle" process that the scheduler can run

Standard File Descriptors for Unix

All command line executables use the following standard for file descriptors:

- `0` — `stdin` (Standard input)
- `1` — `stdout` (Standard output)
- `2` — `stderr` (Standard error)

The terminal emulators job is to:

- Translate key presses to bytes and write to `stdin`
- Display bytes read from `stdout` and `stderr`
- May redirect file descriptors between processes

Checking Open File Descriptors on Linux

`/proc/<PID>/fd` is a directory containing all open file descriptors for a process

`ps x` command shows a list of processes matching your user (lots of other flags)

A terminal emulator may give the output:

```
> ls -l /proc/21151/fd
0 -> /dev/tty1
1 -> /dev/tty1
2 -> /dev/tty1
```

`lsof <FILE>` shows you what processes have the file open

For example, processes using C: `lsof /lib/libc.so.6`

On POSIX Systems, You Can Find Documentation Using `man`

We'll be using the following APIs:

- `execve` (last lecture)
- `fork` (today)
- `wait` (next lecture)

You can use `man <function>` to look up documentation,
or `man <number> <function>`

2: System calls
3: Library calls

`fork` Creates a New Process, A Copy of the Current One

`fork` as the following API:

- Returns the process ID of the newly created child process
  - -1: on failure
  - 0: in the child process
  - >0: in the parent process

There are now 2 processes running

Note: they can access the same variables, but they're separate

Operating system does "copy on write" to maximize sharing

fork-example.c Has One Process Execute Each Branch

```c
int main(int argc, char *argv[]) {
  pid_t pid = fork();
  if (pid == -1) {
    int err = errno;
    perror("fork failed");
    return err;
  }
  if (pid == 0) {
    printf("Child returned pid: %d\n", pid);
    printf("Child pid: %d\n", getpid());
    printf("Child parent pid: %d\n", getppid());
  }
  else {
    printf("Parent returned pid: %d\n", pid);
    printf("Parent pid: %d\n", getpid());
    printf("Parent parent pid: %d\n", getppid());
  }
  return 0;
}
```

Unix Systems Clone Processes with a Parent/Child Relationship

- You can only create new processes with `fork`
- After a `fork` both processes are exactly the same
  - except for the value of `pid` (the child is always 0)
- The scheduler decides when to run either process