CSCI 2132: Software Development

Threads, Processes, and Jobs

Norbert Zeh

Faculty of Computer Science
Dalhousie University

Winter 2019

Processes

Recall:

- A **program** is a file storing executable code
- A **process** executes a program

Process occupies memory:

- **Code** (executable machine code)
- **Data** (static data)
- **Heap** (used for dynamic memory allocation)
- **Stack** (local variables of subroutines, supports recursion)

We will learn more about these types of memory in the context of C programming.

Threads

A **thread** executes the instructions in a program one at a time (sequentially).

Threads

A **thread** executes the instructions in a program one at a time (sequentially).

Thread state:

- Program counter (memory address of next instruction to execute)
- Register contents

Threads

A **thread** executes the instructions in a program one at a time (sequentially).

Thread state:

- Program counter (memory address of next instruction to execute)
- Register contents

Traditionally, one thread per process.

Modern OSs: one process can have many threads (utilize multicore CPUs).

Different processes/threads can execute the same code in memory.

The Cost of Processes and Threads

Generally, there are many more processes/threads than CPU cores.

OS allocates “times slices” of CPU cores to threads/processes.

Processes store more admin information than threads.

Process creation and process switching is more costly than thread creation and thread switching.

Process Control Block (PCB)

Created by OS when a process starts

Includes:

- Process identifier (PID)
- Program counter
- Resources allocated to process (e.g., memory, open files)
- Process ownership (user and group)
- Process state (running, sleeping, pre-empted, created, zombie)

Process Creation

Every process is created by another process and becomes its child process.

One exception: the root process init (PID=1)

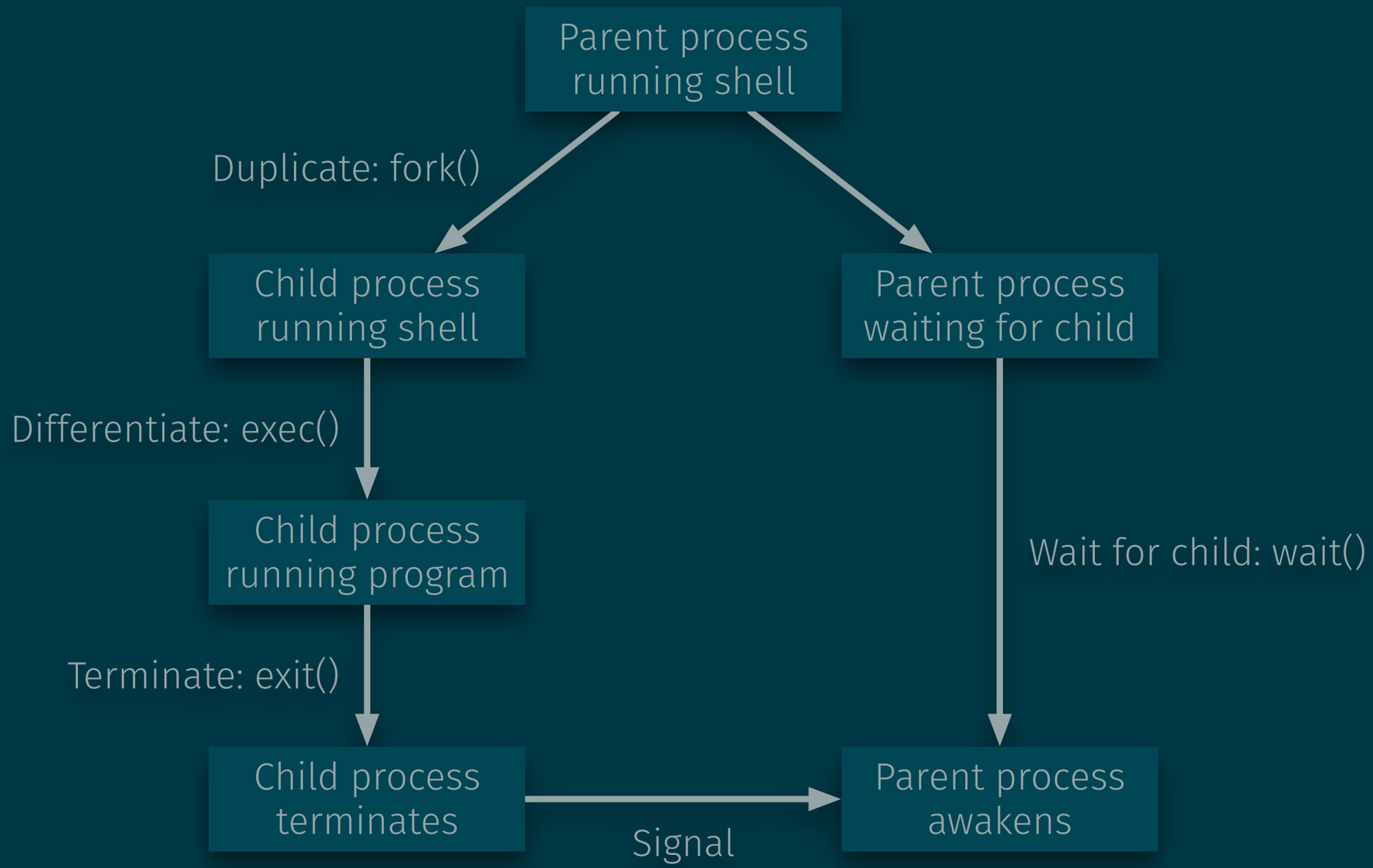
This gives us a process tree

Process Tree

```
root         1     0  0  2018 ?        Ss    15:23 /usr/lib/systemd/systemd
 ...
root      2382     1  0  2018 ?        Ss     5:51 /usr/lib/systemd/systemd-journald
root      2414     1  0  2018 ?        Ss     0:00 /usr/sbin/lvmetad
root      4575     1  0  2018 ?        S<sl   1:01 /sbin/auditd
polkitd   4598     1  0  2018 ?        Ssl    0:40 /usr/lib/polkit-1/polkitd
dbus      4600     1  0  2018 ?        Ssl    3:18 /usr/bin/dbus-daemon
root      4631     1  0  2018 ?        Ssl    3:12 /usr/sbin/NetworkManager
root      4633     1  0  2018 ?        Ss     1:45 /usr/lib/systemd/systemd-logind
chrony    4691     1  0  2018 ?        S      0:17 /usr/sbin/chronyd
root      5022     1  0  2018 ?        Ss     1:33 /usr/sbin/sshd
root      5027     1  0  2018 ?        Ss     9:54 /usr/sbin/cupsd
root      5031     1  0  2018 ?        Ssl  242:58 /usr/lib/gitlab-runner/gitlab-runner
 ...
root     15934  5022  0 22:09 ?        Ss     0:00 sshd: nzeh [priv]
 ...
nzeh     15937 15934  0 22:09 ?        S      0:00 sshd: nzeh@pts/29
 ...
nzeh     15938 15937  0 22:09 pts/29   Ss     0:00 -bash
```

Process Creation

How the Shell Starts a Program



Foreground and Background Processes

Foreground process controls the terminal

Background process cannot read from keyboard but can print to terminal

Job and Process Control

Job control = shell functionality for managing processes

Print jobs (processes started from the current shell)

```
$ jobs
```

Print processes

```
$ ps
```

Start a process in background

```
$ xterm &
```

Job and Process Control

Suspend a process

```
Ctrl-Z
```

Put suspended job in background

```
$ bg
```

```
$ bg %job
```

Resume suspended job in foreground

```
$ fg
```

```
$ fg %job
```

Terminate a job or process

```
$ kill %job
```

```
$ kill pid
```