Processes

Chapter 3

Process

Also called job or task

- When a program or executable file is loaded from disk and started running, it is called a process
  - identified by a unique process ID (PID) number
  - has an owner
  - private data
- A program can be loaded more than once
  - creates multiple processes
  - each process has a different PID
  - each process may have a different owner
  - does not share data with other instances
- PIDs are unique, nonnegative integers

Process State (1)

Process memory layout

command line args & environment vars

parameters & auto variables

stack

heap

uninitialized data

initialized data

text

Process control block (pcb)

process state

process id (pid)

program counter

registers

memory bounds

open files

low memory

high memory

dynamic variables

global and static data

machine instructions

Stack Frames

Allocating variables on the stack

void forward(int number)
{
    if (number != 0)
    {
        forward(number / 10);
        cout << number % 10;
    }
}

void reverse(int number)
{
    if (number != 0)
    {
        cout << number % 10;
        reverse(number / 10);
    }
}

Zombies, Orphans, and Daemons

Special process states

- A zombie is a process that has exited but whose parent hasn’t yet checked its exit status
  - Waiting for a parent to wait on it
  - Retains PID, termination status, and CPU time
  - A zombie in BSD days: <defunct> in SVR4
- An orphan is a process whose parent dies before it does
  - Orphans are adopted by init
  - Some systems kill background processes at logout; the nohup utility sees that they are adopted by init instead
- A daemon is a process that runs in the background
  - Parent exits, adopted by init
  - Disassociated from any tty

Unix System Processes

Processes created during system boot

- 0 System kernel
  - "hand crafted" at boot
  - called swap in older versions (swaps the CPU between processes)
  - called sched in newer versions (schedules processes)
  - creates process 1
- 1 init (the parent of all processes except process 0)
  - general process spawner
  - begins building locale-related environment
  - sets or changes the system run-level
- 2 page daemon (pageout on most systems)
- 3 file system scanner (fsflush)
Unix ps Example

BSD and SVR4

```
\$ ps -ax
19  0  0  0  0  96  0  0  0  T  T  0:00 bash
19  0  0  0  0  55  20  466  136  wc -l
16  0  0  0  0  98  0  0  0  S  S  0:05 ps
19  0  0  0  0  60  0  0  0  S  T  10:41 fgetc
18  0  0  0  0  0  0  0  0  S  T  0:00 <defunct>
18  0  29550  515  0  54  2772  1790  8243  S  T  0:00 /usr/bin/ps
16  0  29556  57  0  57  20  3044  1188  psវនខ្លះ  S  T  0:07 ACION.exe -p
16  0  4923  4921  0  28  20  1256  1264  akeakm_s  S  T  0:00 login -d
16  0  1288  4924  4923  0  58  20  1506  893  akeakm_s  S  T  0:00 <defunct>
16  0  5023  4924  1  28  20  1440  860  C  T  0:00 ps -ax
```

```
\$ /bin/ps -ax
16  0  0  0  0  0  0  0  0  T  T  0:00 bash
16  0  0  0  0  0  0  0  0  T  T  0:00 <defunct>
16  0  0  0  0  0  0  0  0  T  T  0:00 login
16  0  0  0  0  0  0  0  0  T  T  0:00 <defunct>
16  0  0  0  0  0  0  0  0  T  T  0:00 <defunct>
16  0  0  0  0  0  0  0  0  T  T  0:00 <defunct>
16  0  0  0  0  0  0  0  0  T  T  0:00 <defunct>
16  0  0  0  0  0  0  0  0  T  T  0:00 <defunct>
16  0  0  0  0  0  0  0  0  T  T  0:00 <defunct>
```

Windows Example

Task Manager

Context Switch

Stopping and starting processes

- Running process
- Blocks
- Is interrupted
- Spawns/wait
- Terminates
- State of suspending process saved in PCB
- Does not return from switch function call
- State of dispatched process loaded
- Appears as if returning from switch function call

Switch Function Call

Unix Process Life Cycle

Overview of creating new processes

fork creates two identical processes (parent and child)
exec
- replaces the process’s instructions or program with the instructions for another program
- because it is the same process, it does not change the PID
- The system maintains completed processes in a quiescent state until their exit status is examined by their parent.
**Unix vs Windows**

- Process creation
  - Unix
    - `fork`
    - `child`
      - replica of parent – most data duplicated (copy on write – cow)
      - `exec(command)`
    - `new command`
  - Windows
    - `spawn(command)`
    - `child`
      - new command, does not inherit parent’s data

**Scheduling Queues**

- Figure 3.6
  - Ready queue
    - Head
    - Tail
    - PCBs
    - Process
    - Process
    - Process
    - Process
    - Process

**Process Scheduling**

- Systems will have two or three
  - Kinds of processes and achieving a balance
    - I/O bound
    - CPU bound
  - Long-term/job/admission scheduler
    - Choose job/process to load from disk
    - Balances I/O and CPU bound processes – How???
    - Not used on desktop operating systems (Windows, Unix, Linux, etc.)
  - Medium-term scheduler
    - Swaps jobs/processes in and out of memory
    - Balances I/O and CPU bound processes
    - Required by systems with virtual memory
  - Short-term/CPU scheduler or dispatcher
    - Selects a ready process – does context switch

**CPU Scheduling**

- Short-term or CPU (Figure 3.7)
  - $\Delta t = 100$ mSec
  - Ready queue
  - CPU
  - I/O
  - I/O queue
  - I/O request
  - Time slice expired
  - Fork a child
  - Interrupt occurs
  - Wait for an interrupt

**CPU Scheduling**

- Medium-term (Figure 3.8)
  - Swap in
  - Partially executed swapped-out processes
    - (on disk)
  - Ready queue
    - (in memory)
  - CPU
    - (in memory)
  - I/O
    - I/O waiting queues

**Interprocess Communication (IPC)**

- Figure 3.13
  - (a) Pipes
  - (b) Shared memory
  - Process A
    - Shared
    - Process B
    - Kernel
    - Kernel
    - 2
    - 1