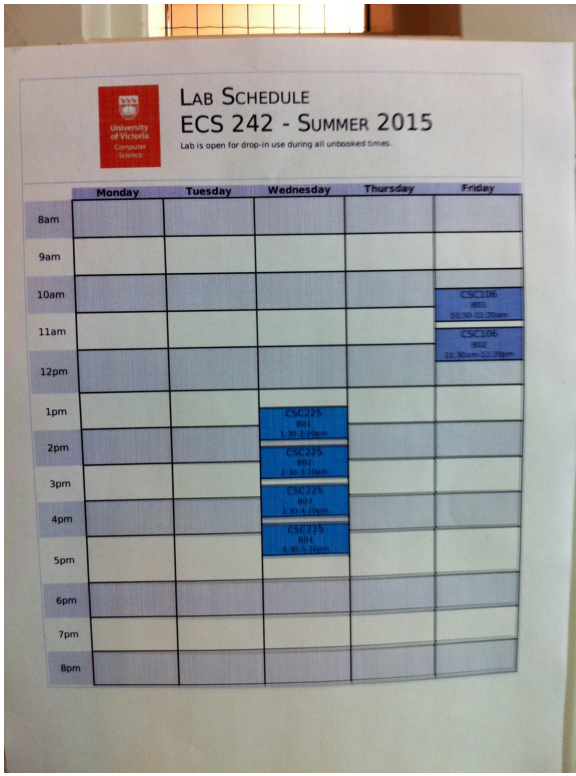
CSc 360
Operating Systems
Process Operations

Jianping Pan

Summer 2015

Linux computers in ECS 242?!

- No longer remote accessible
  - due to the “flexilab” w/ hypervisors
  - both Linux and Win images
- Drop-in possible
  - see schedule
- Other alternatives
  - linux.csc.uvic.ca
  - auto-load-balanced to multiple machines



Review: process

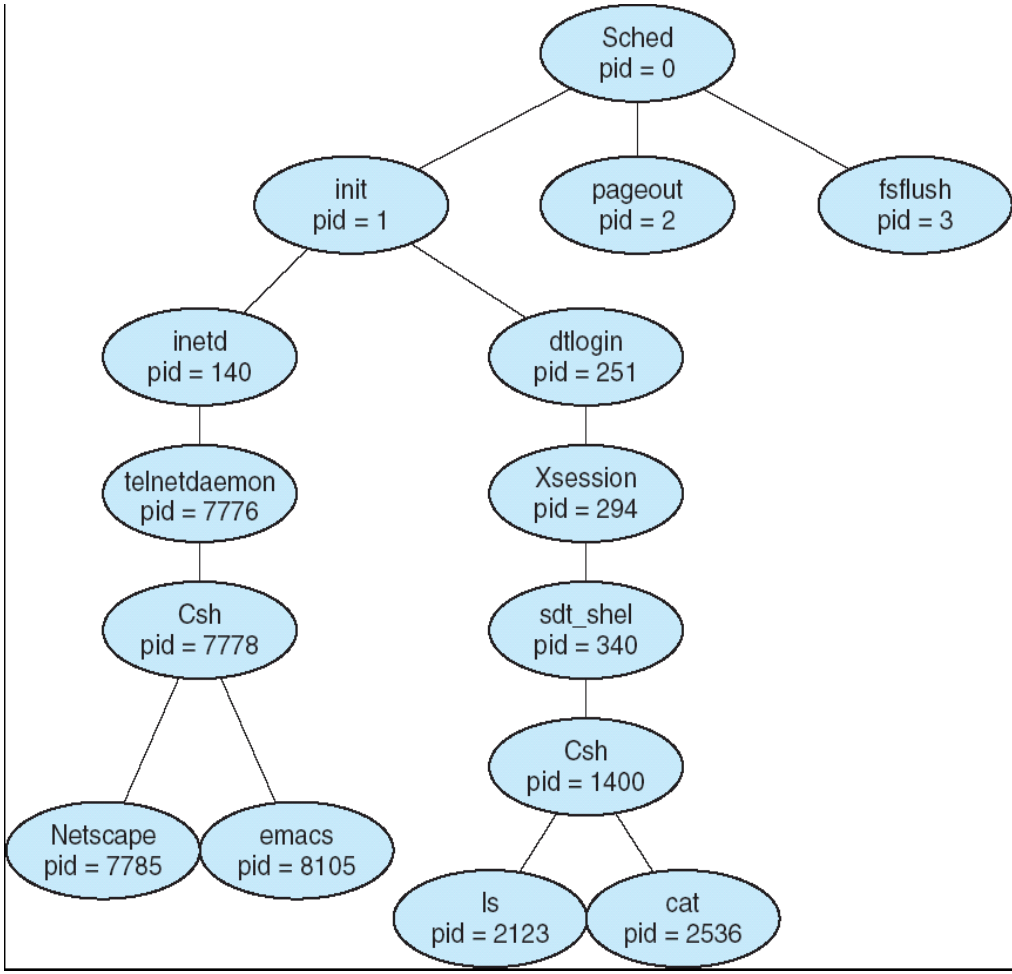
- Process vs program
  - active vs passive entity
- Process control block
  - context switching
- Process scheduling
  - long-term scheduling
  - short-term scheduling
  - medium-term scheduling

Process creation

- Creating processes
  - parent process: create child processes
  - child process: created by its parent process
- Process tree
  - recursive parent-child relationship; why tree?
  - /usr/bin/pstree
- Process ID (PID) and Parent PID (PPID)
  - usually nonnegative integer

Process tree

- sched (0)
  - init (1)
    - all user processes
  - pageout
    - memory
  - fsflush
    - file system




```
├─snmpd
├─sshd─┬─5*[sshd───sshd───sftp-server]
│      ├─sshd───sshd───bash───ssi
│      ├─sshd───sshd───bash─┬─more
│      │                    └─pstree
│      ├─sshd───sshd───csh───sftp-server
│      ├─sshd───sshd───bash
│      ├─sshd───sshd───bash───mutt
│      ├─sshd───sshd───tcsh───nano
│      └─sshd───sshd───tcsh
├─syslogd
```

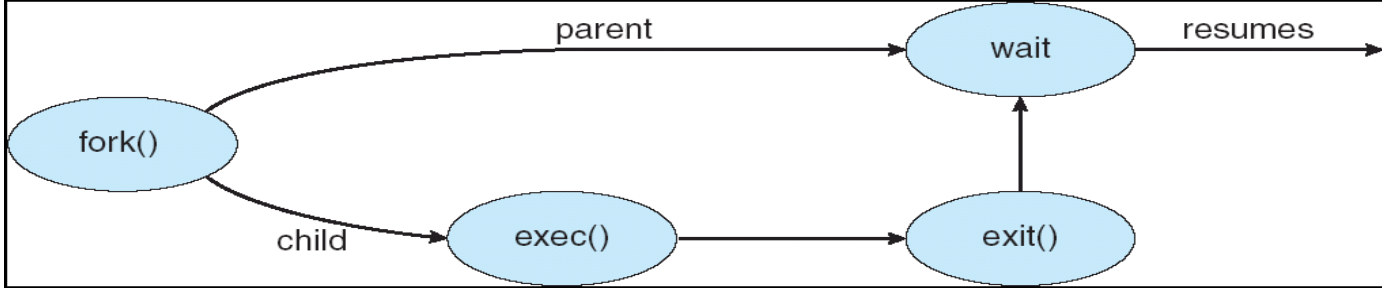
pstree on linux.csc.uvic.ca

Parent vs child processes

- Process: running program + resources
- Resource sharing: possible approaches
  - all shared
  - some shared (e.g., read-only code)
  - nothing shared*
- Process execution: possible approaches
  - parent waits until child finishes
  - parent and child run concurrently*

fork(), exec*(), wait()

- Create a child process: fork()
  - return code < 0: error (in “parent” process)
  - return code = 0: you’re in child process
  - return code > 0: you’re in parent process
    - return code = child’s PID
- Child process: load a new program
  - exec*(): front-end for execve(file, arg, environ)
- Parent process: wait() and waitpid()



Example

```
int main()
{
    Pid_t  pid;
  /* fork another process */
  pid = fork();
  if (pid < 0) { /* error occurred */
     fprintf(stderr, "Fork Failed");
     exit(-1);
  }
  else if (pid == 0) { /* child process */
     execlp("/bin/ls", "ls", NULL);
  }
  else { /* parent process */
  /*parent will wait for the child to complete*/
     wait (NULL);
     printf ("Child Complete");
     exit(0);
  }
}
```

Process termination

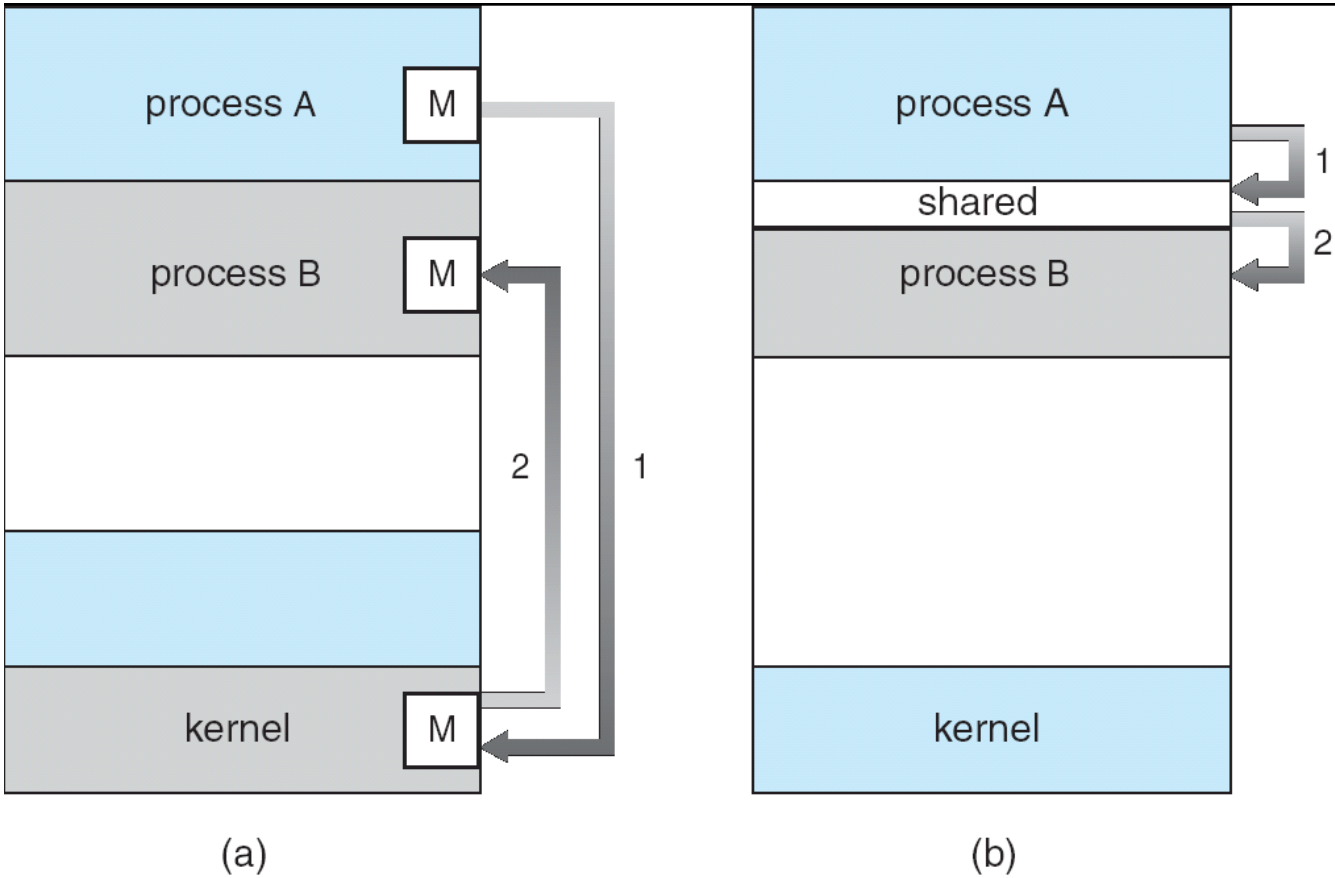
- Terminate itself: exit()
  - report status to parent process
  - release allocated resources
- Terminate child processes: kill(pid, signal)
  - actually send a signal to the child
    - child resource exceeded, child process no long needed, and so on
  - parent is exiting
    - cascading termination, or find another parent

Process communication

- Independent process
  - standalone process
- Cooperating process
  - affected by or affecting other processes
    - sharing, parallel, modularity, convenience
- Process communication
  - shared memory
  - message passing

Message passing vs shared memory

- Overhead
- Protection



This lecture

- Process operations
  - process creation
    - process tree
  - process termination
  - the need for inter-process communication
- Explore further
  - /bin/ps, /usr/bin/top, /usr/bin/pstree
  - how does a child process find its parent's PID?

Next lecture

- Inter-process communication
  - read OSC7 Chapter 3 (or OSC6 Chapter 4)