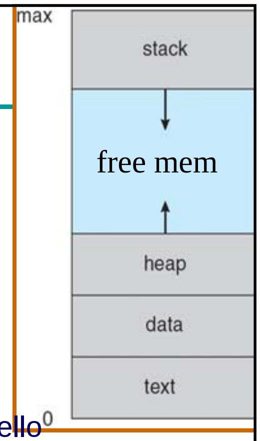
Processes

Chapter 3.1, 3.2, 3.3, 3.4

Agenda

1. What is a process?
2. Process states
3. PCB
4. Context switching
5. Process scheduling
6. Process creation
7. Process termination
8. Process communication

1. What is a Process?



Process: a program in execution

Program: passive entity

- static binary file on storage
  - – e.g., gcc -o hello hello.c; ls -l hello
  - -rwxrwxr-x 1 *user group size date/time* hello

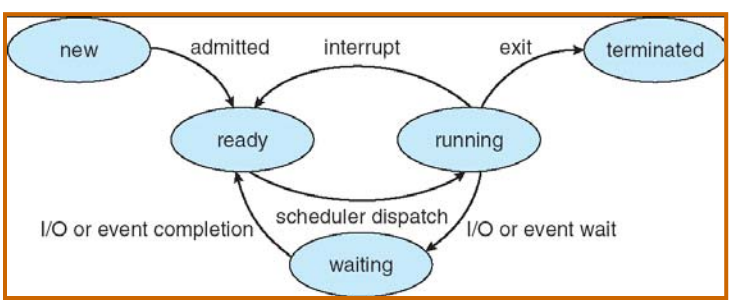
Process: active entity; resource allocated!

- ./hello
  - – text (code); data (static), stack, heap
  - – process control block

2. Process states

E.g., one CPU

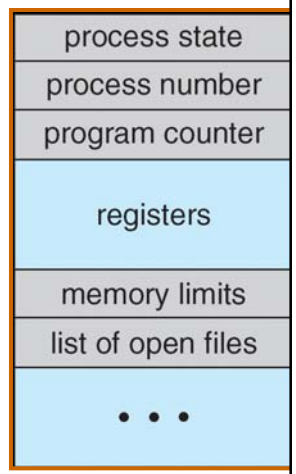
- one running process at any time
- maybe many ready/waiting processes



3. Process control blocks (PCB)

PCB: keep track processes

- state: ready/running, etc
- CPU
  - PC, registers, priority, etc
- memory
  - memory control information
- I/O
  - e.g., opened files
- accounting



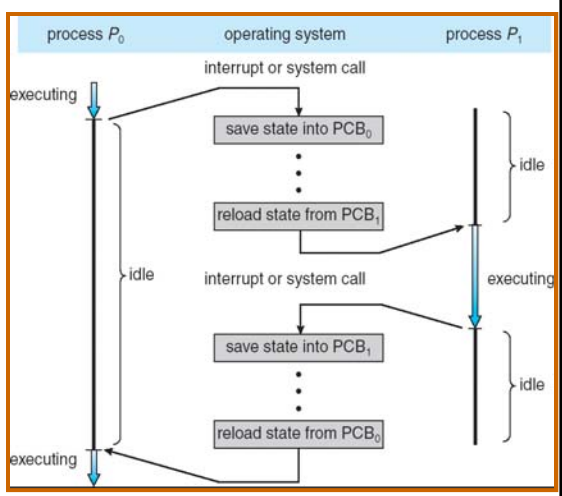
4. Context switching

Context switch

- save states
- restore states

When

- timer
- I/O, memory
- trap
- system call

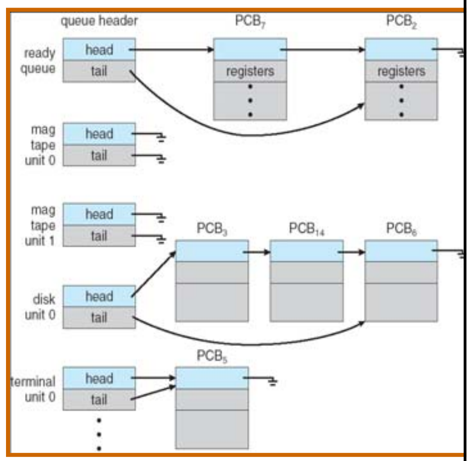


5. Process scheduling (1)

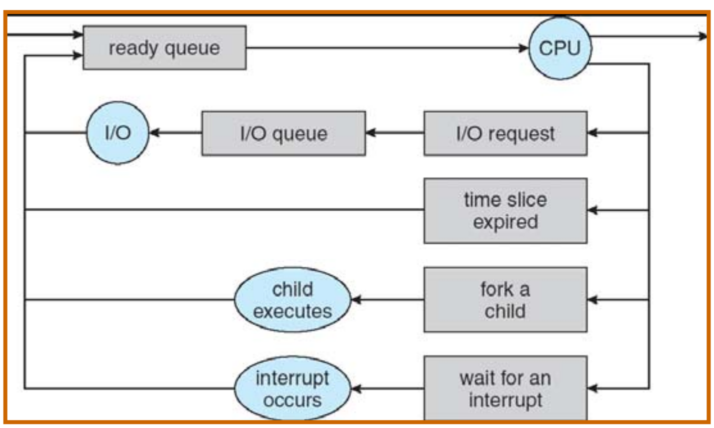
Multiprogramming
- utilization

Timesharing
- interactive

Scheduling queues
- linked list
- ready queue
- I/O queue



5. Process Scheduling (2): Queuing system



5. Process Scheduling (3): Queuing scheduler

Who's the next?

Long-term scheduler

- job scheduler (spooling)
- get to ready queue
- CPU-intensive vs I/O intensive

Short-term scheduler

- CPU scheduler
- frequency vs overhead

5. Process Scheduling (4): More on scheduling

Medium-term scheduler

- who is NOT the next
  - – reduce the degree of multiprogramming
- swap-in/out

Scheduling algorithms

- first-come-first-server, shortest-job-first, priority, round-robin, fair and weighted fair, …
- more in Chapter 5

6. Process creation (1)

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

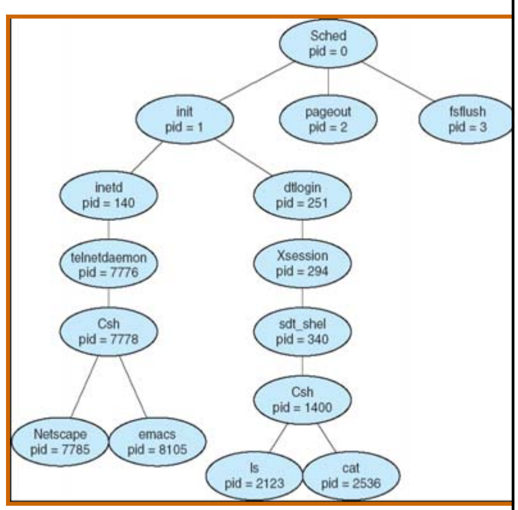
Process ID (PID) and Parent PID (PPID)

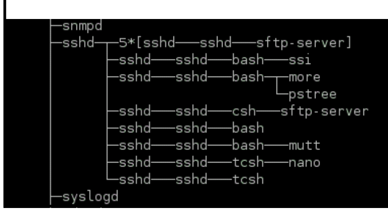
- usually nonnegative integer

6. Process creation (2): 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

6. Process creation (3): 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*

6. Process creation (4): fork(), exec*(), wait()

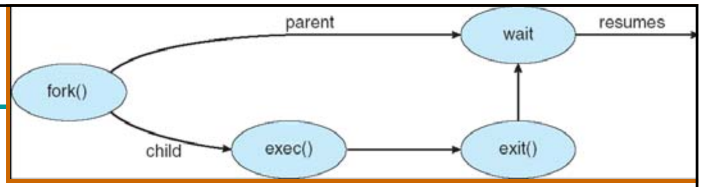
Create a child process: fork()

- 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);
   }
}
```


7. Process termination

Terminate itself: exit()

- report status to parent process
- release allocated resources

Terminate child processes: kill(pid, signal)

- child resource exceeded
- child process no long needed
- parent is exiting
  - cascading termination
  - find another parent

8. Process communication (1)

Independent process
- standalone process

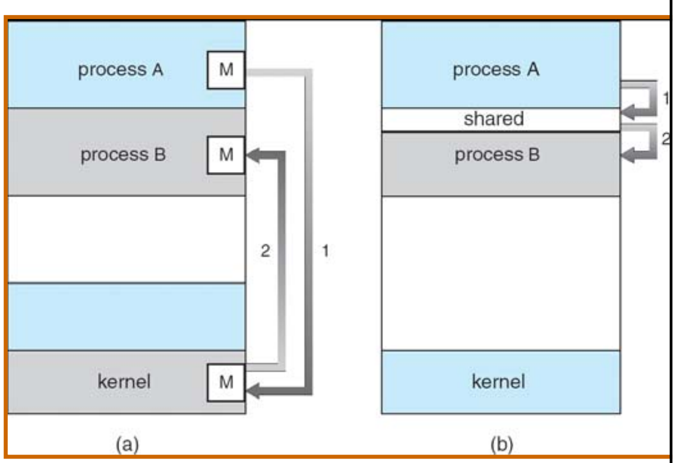
Cooperating process
- affected by or affecting other processes
  – sharing, parallel, modularity, convenience

Process communication
- shared memory
- message passing

8. Process communication (2): Message passing vs shared memory

Overhead

Protection



More Info.

Explore further

- /bin/ps, /usr/bin/top, /usr/bin/pstree
- how does a child process find its parent's PID?