Process Concept

An operating system executes a variety of programs:

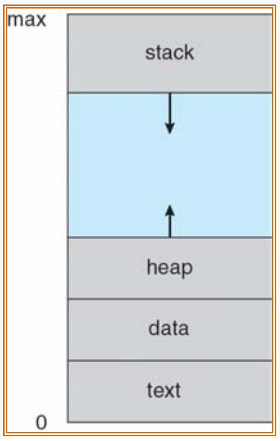
- Batch system – jobs
- Time-shared systems – user programs or tasks

Textbook uses the terms *job* and *process* almost interchangeably

Process – a program in execution; process execution must progress in sequential fashion

A process includes:

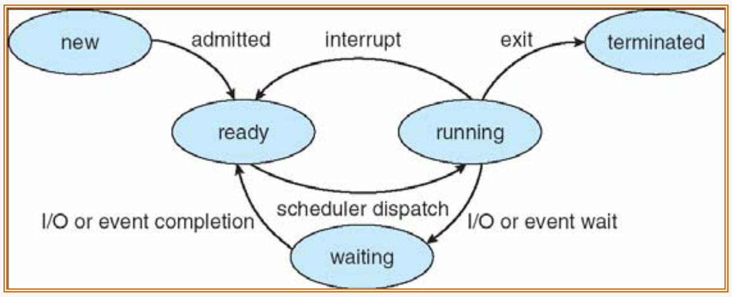
- program counter
- stack
- data section



Process State

As a process executes, it changes *state*

- **new**: The process is being created
- **running**: Instructions are being executed
- **waiting**: The process is waiting for some event to occur
- **ready**: The process is waiting to be assigned to a process
- **terminated**: The process has finished execution

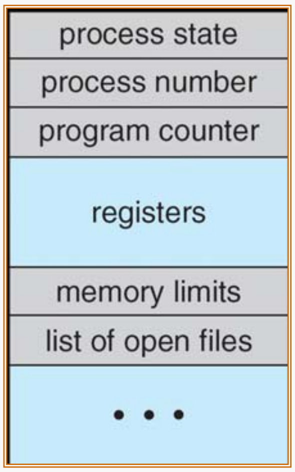


Process Control Block (PCB)

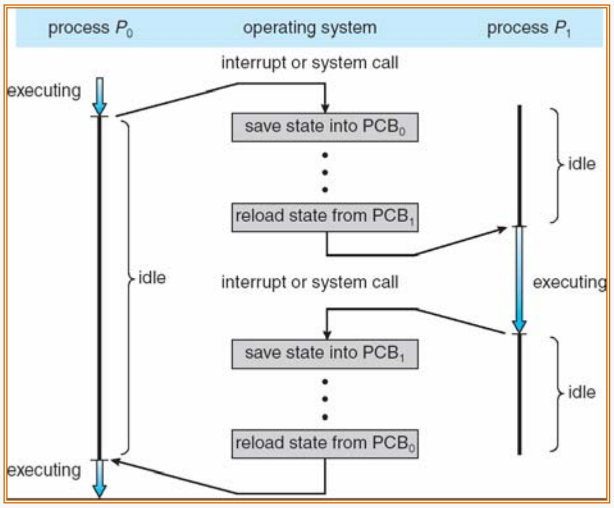
Information associated with each process

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

aka: *process descriptor*



CPU Switch From Process to Process



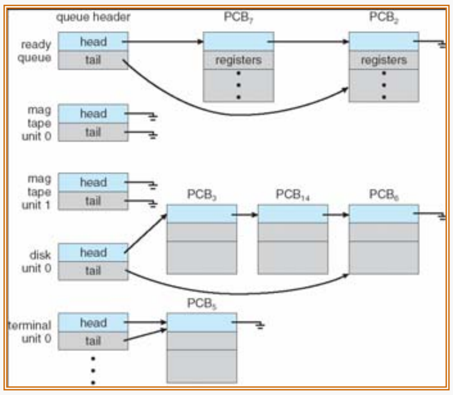
Process Scheduling Queues

Job queue – set of all processes in the system

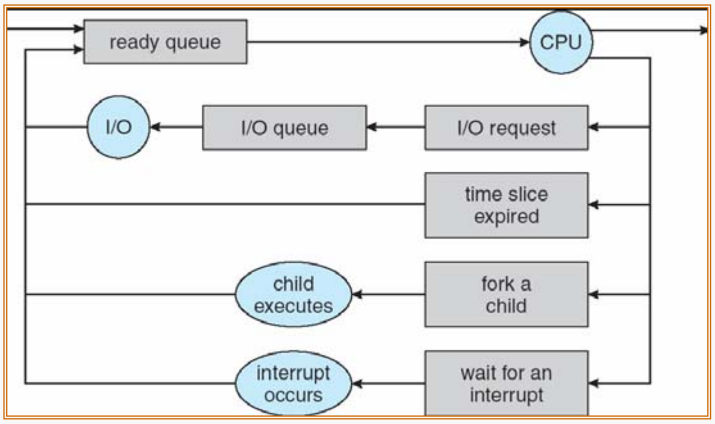
Ready queue – set of all processes residing in main memory, ready and waiting to execute

Device queues – set of processes waiting for an I/O device

Processes migrate among the various queues



Representation of Process Scheduling



Schedulers

Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue

Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU

Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast)

Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow)

The long-term scheduler controls the *degree of multiprogramming*

Processes can be described as either:

- **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
- **CPU-bound process** – spends more time doing computations; few very long CPU bursts

Context Switch

When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process

Context-switch time is overhead; the system does no useful work while switching

Time dependent on hardware support

Process Creation

Parent process create children processes, which, in turn create other processes, forming a tree of processes

Resource sharing

- Parent and children share all resources
- Children share subset of parent's resources
- Parent and child share no resources

Execution

- Parent and children execute concurrently
- Parent waits until children terminate

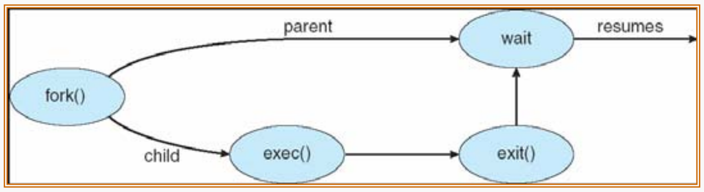
Address space

- Child duplicate of parent
- Child has a program loaded into it

UNIX examples

- **fork** system call creates new process
- **exec** system call used after a **fork** to replace the process' memory space with a new program

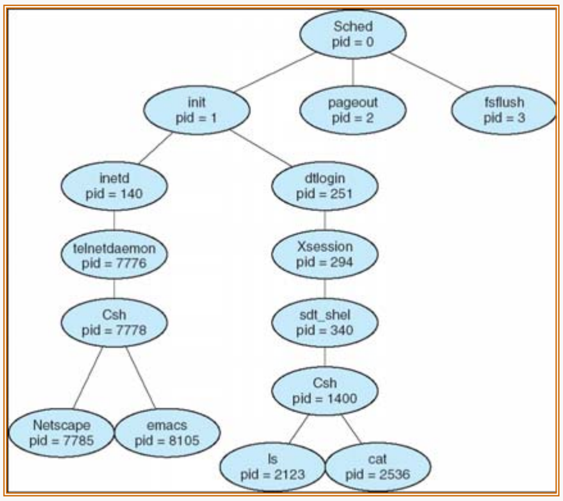
Process Creation



C Program Forking Separate Process

```
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);
   }
}
```

A tree of processes on a typical Solaris



Process Termination

Process executes last statement and asks the operating system to delete it (**exit**)
- Output data from child to parent (via **wait**)
- Process' resources are deallocated by operating system

Parent may terminate execution of children processes (**abort**)
- Child has exceeded allocated resources
- Task assigned to child is no longer required
- If parent is exiting
  - Some operating system do not allow child to continue if its parent terminates
    - All children terminated - *cascading termination*