Process Management

# Process Concept (Cont.)

- Program is *passive* entity stored on disk (**executable file**), process is *active*
  - Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes
  - Consider multiple users executing the same program

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
- Multiple parts
  - The program code, also called **text section**
  - Current activity including **program counter**, processor registers
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
 - **Heap** containing memory dynamically allocated during run time

Process in Memory
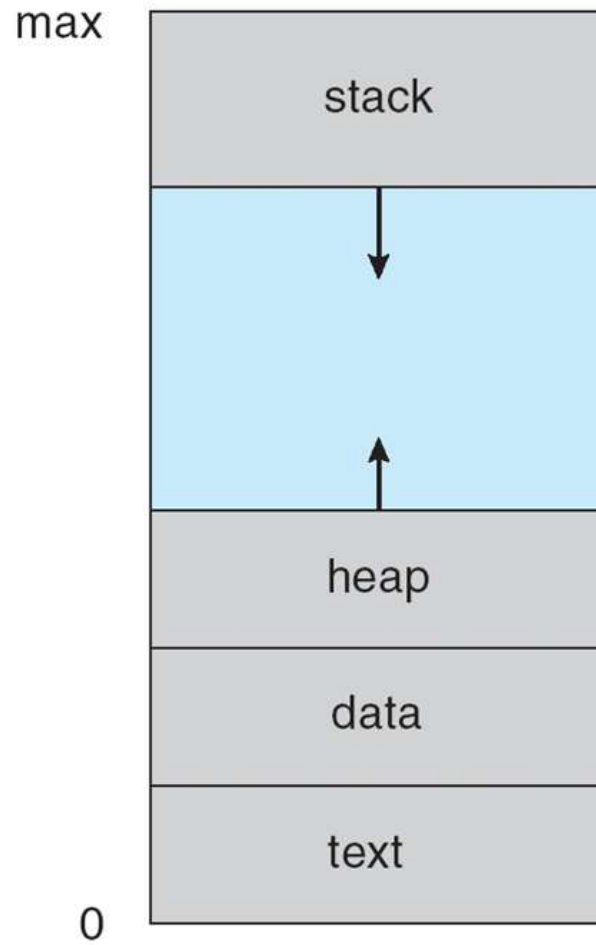
Process State

- 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 processor
 - **terminated**: The process has finished execution

Diagram of Process State
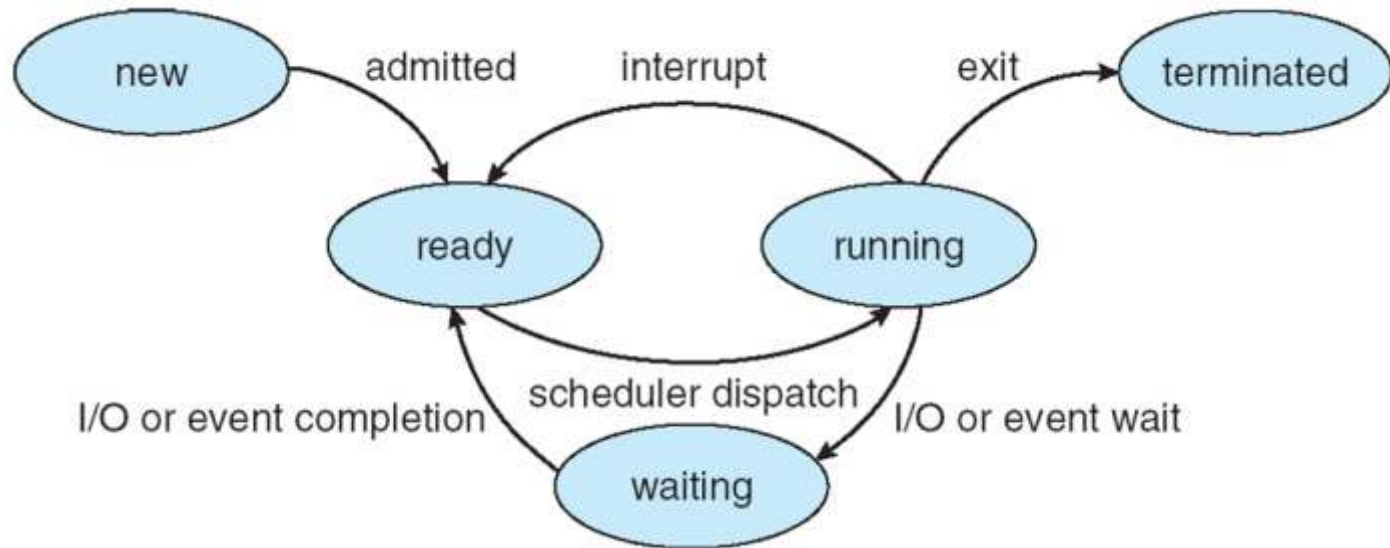
Process Control Block (PCB)

Information associated with each process

(also called **task control block**)

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files

| |
|---|
| process state |
| process number |
| program counter |
| registers |
| memory limits |
| list of open files |
| • • • |
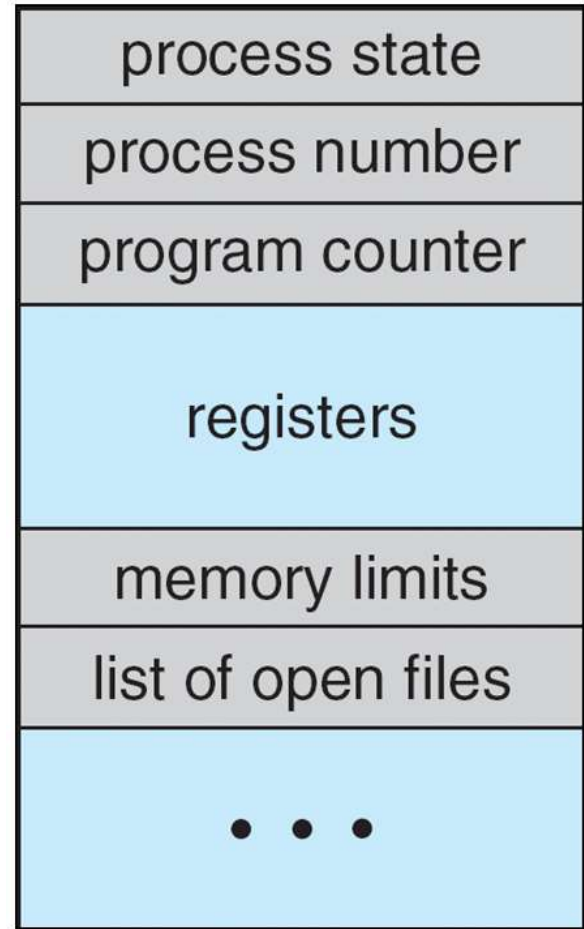
CPU Switch From Process to Process
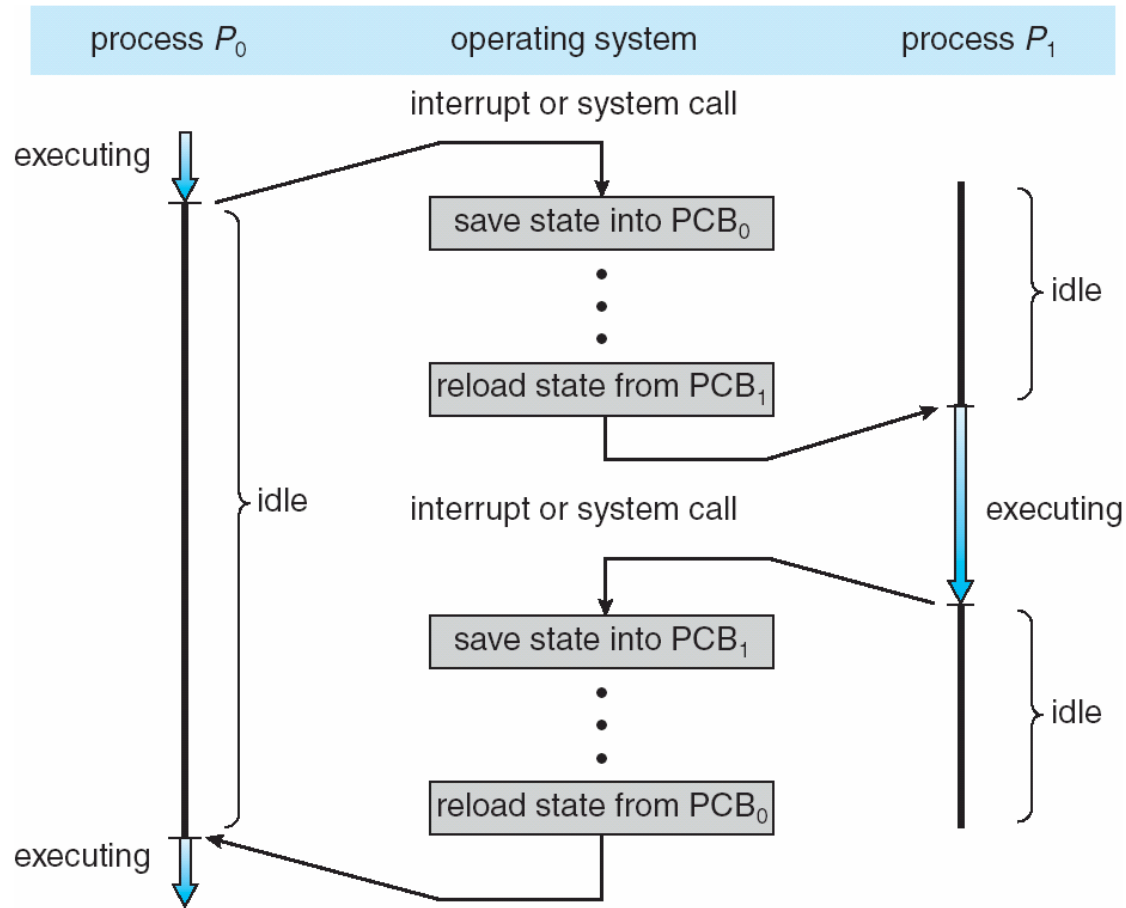
Threads

- Light weight process
- Single thread of instruction is being executed
- This single thread of control allows the process to perform only one task at a time
- Word processors – could not do both typing and spell check simultaneously
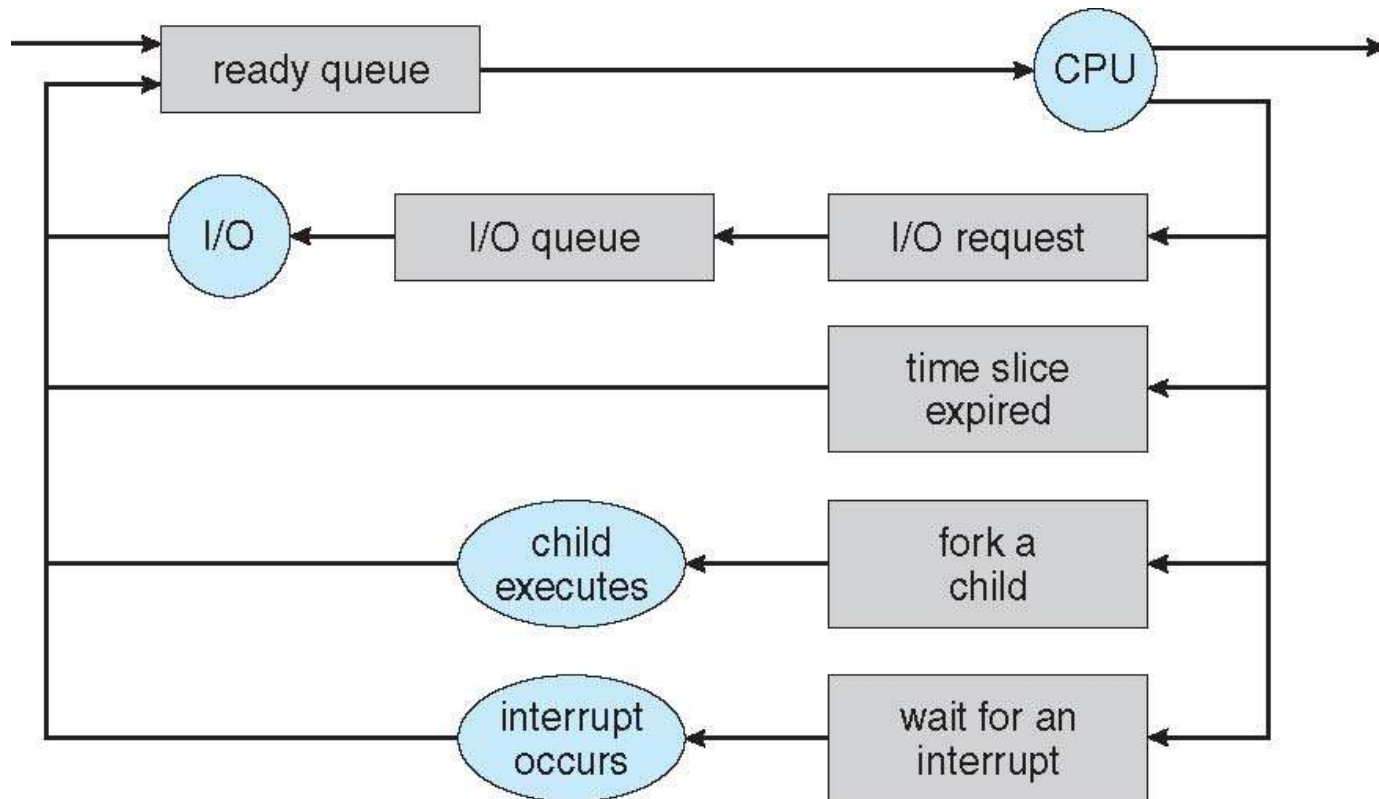
Process Scheduling

- The objective of multiprogramming is to have some process running at all times, so that we can maximize CPU utilization
- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
  - **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

Ready Queue And Various I/O Device Queues
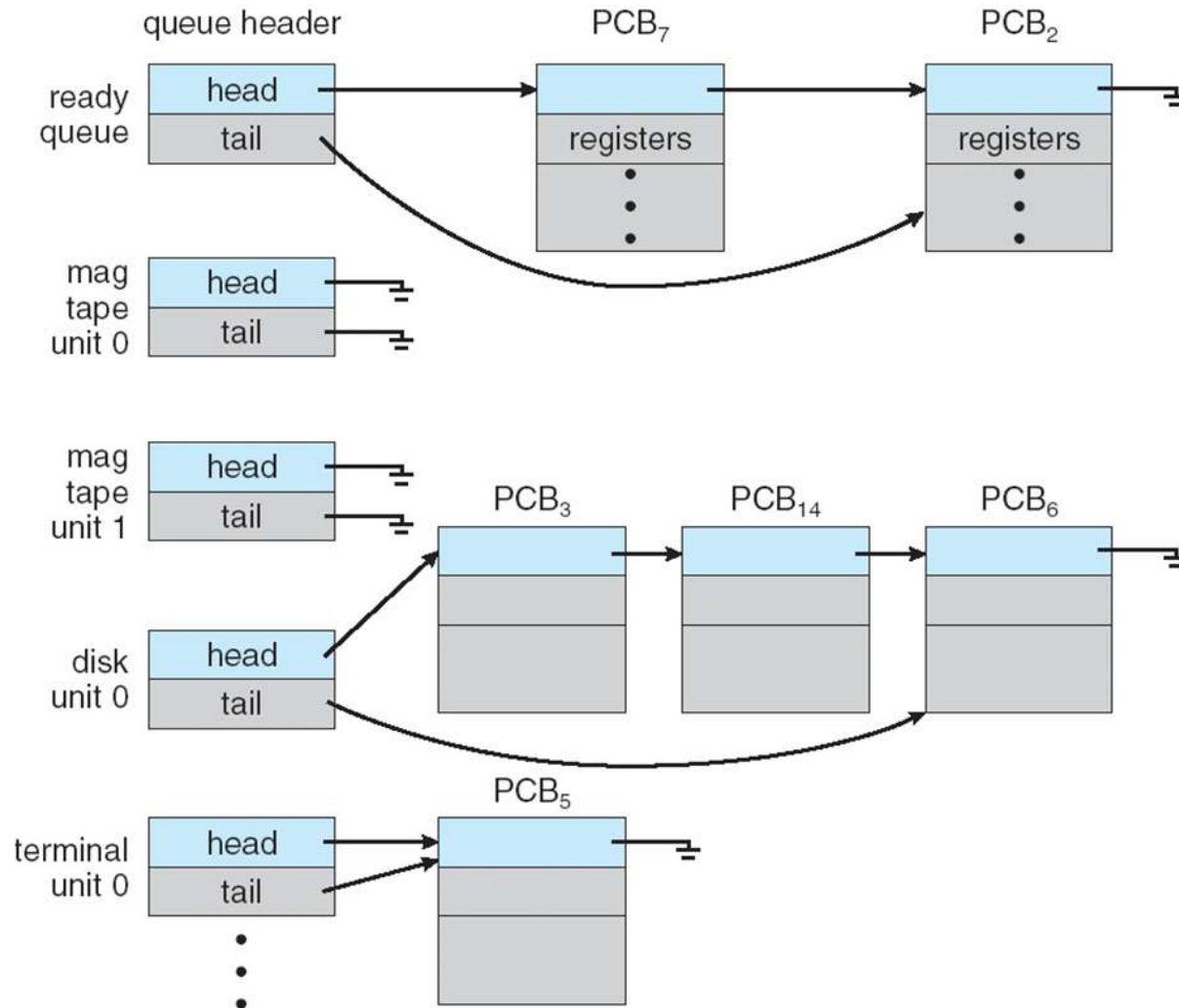
Representation of Process Scheduling

□ **Queueing diagram** represents queues, resources, flows

- processes scheduling queue
- New process always is put in ready queue. It unit in the ready queue until it is selected for execution. Once it is allocated to CPU is starts executing. The several events may occur.
  1. A process may issue I/O request then be placed in I/O queue.
  2. A process may create a sub process and wait for its termination.
  3. A process may be interrupted in between and put back in ready queue. The process continues this cycle until if terminates.

Schedulers

- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) $\Rightarrow$ (must be fast)
- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked infrequently (seconds, minutes) $\Rightarrow$ (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
- Long-term scheduler strives for good *process mix*

Addition of Medium Term Scheduling

- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease
  - Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**
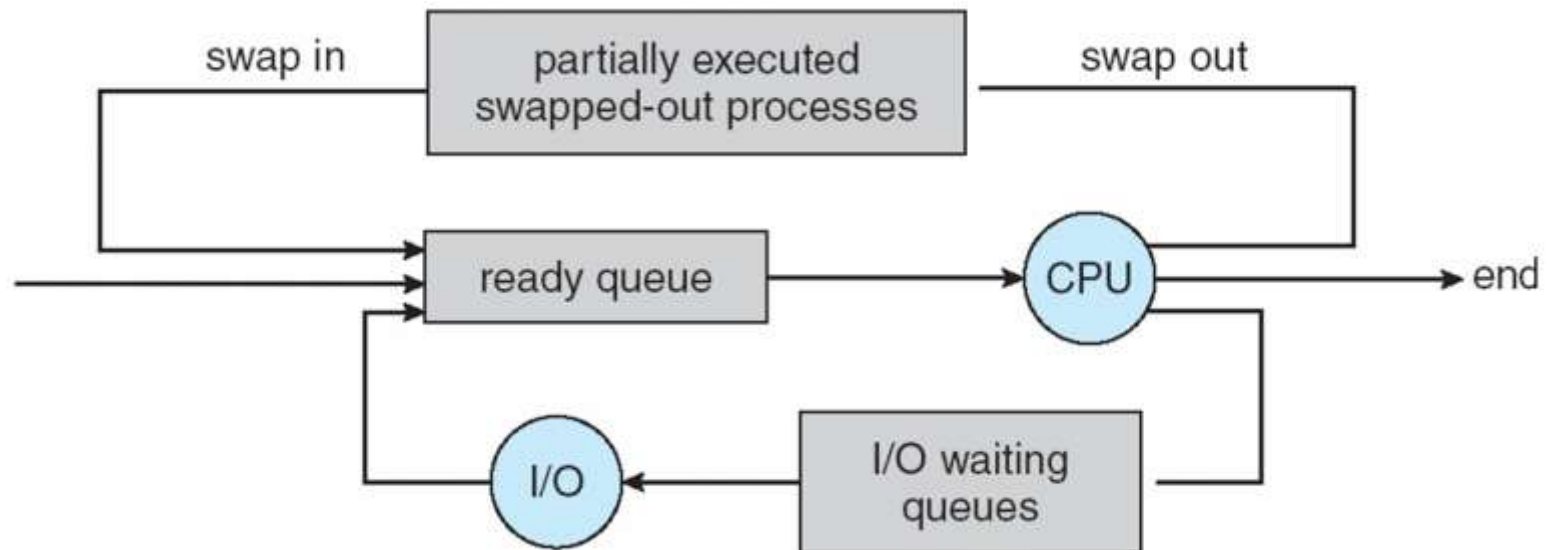
Context Switch

- 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 via a **context switch**
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB ➔ the longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU ➔ multiple contexts loaded at once

Operations on Processes

- System must provide mechanisms for:
  - process creation
  - process termination

Process creation

- A process may create several new processes, via a create-process system call, during the course of execution.

- The creating process is called a parent process, and the new processes are called the children of that process.

- Each of these new processes may in turn create other processes, forming a tree of processes.

Process Creation

- **Parent** process create **children** processes, which, in turn create other processes, forming a **tree** of processes
- Generally, process identified and managed via a **process identifier** (**pid**)
- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources the child process has a new program to load it.

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

Process Creation (Cont.)

- ## Address space
  - Child duplicate of parent.
  - Child has a program loaded into it.

- ## UNIX examples
  - **fork** system call creates new process
  - **execlp** system call used after a **fork** to replace the process' memory space with a new program.

Processes Tree on a UNIX System
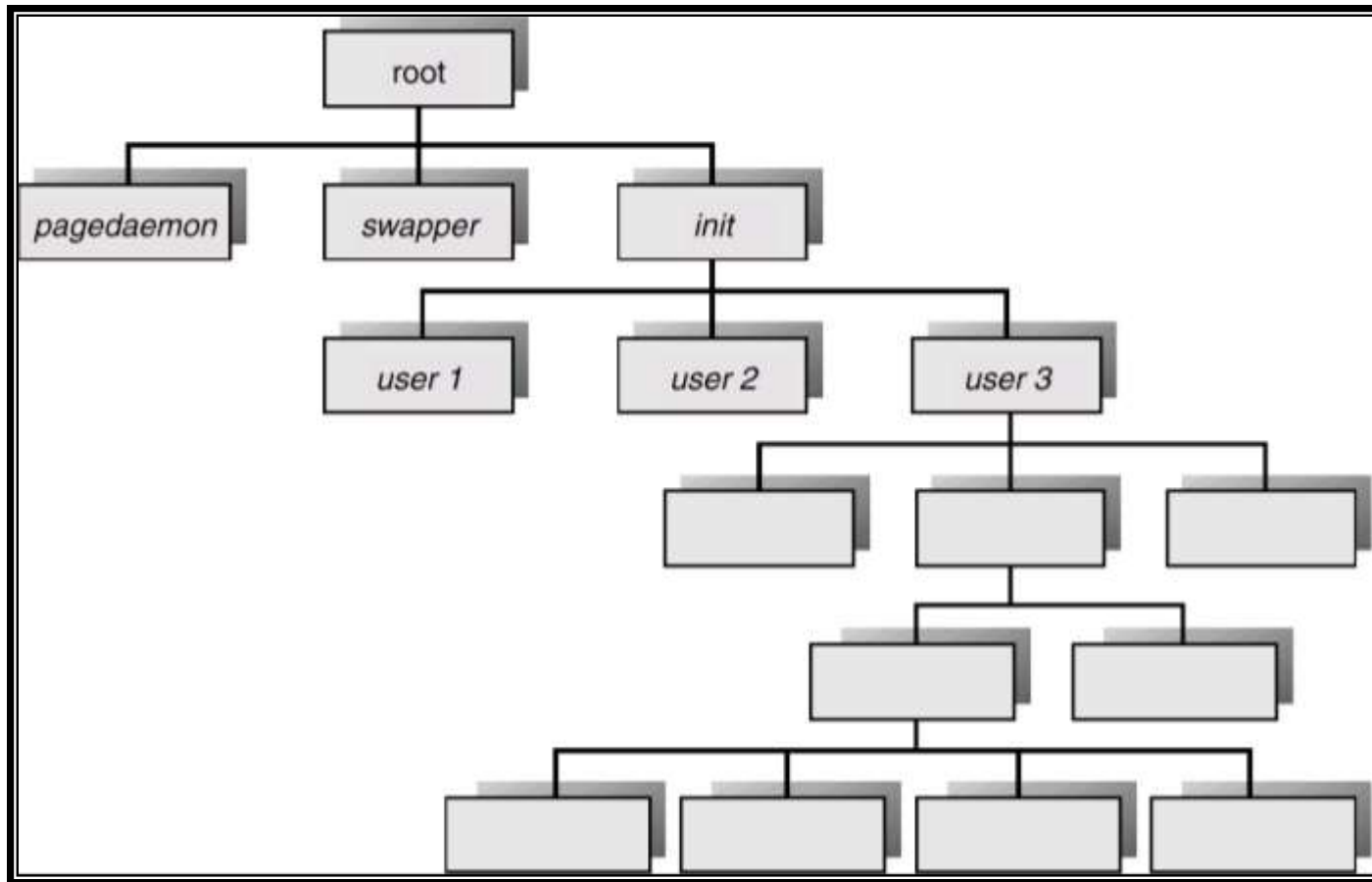
Process Termination

- Process executes last statement and then asks the operating system to delete it using the **exit()** system call.
  - Returns status data from child to parent (via **wait()**)
  - Process' resources are deallocated by operating system
- Parent may terminate the execution of children processes using the **abort()** system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates

Process Termination

- Some operating systems do not allow child to exists if its parent has terminated. If a process terminates, then all its children must also be terminated.
  - **cascading termination.** All children, grandchildren, etc. are terminated.
  - The termination is initiated by the operating system.
- The parent process may wait for termination of a child process by using the `wait()` system call.   The call returns status information and the pid of the terminated process

  ## `pid = wait(&status);`

- A process which has finished the execution but still has entry in the process table to report to its parent process is known as a zombie process. A child process always first becomes a zombie before being removed from the process table.
- If parent terminated without invoking `wait`, process is an **orphan**