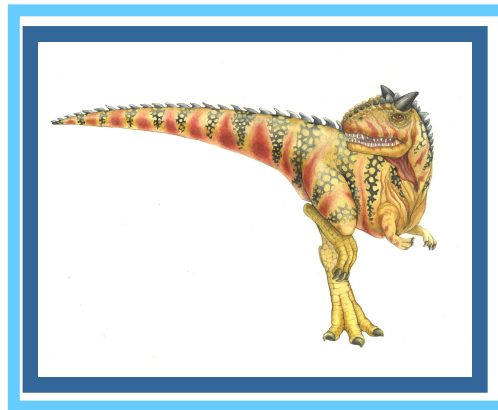


Chapter 3: Processes





Administrivia

- Friday: lab day.
- For Monday: Read Chapter 4.
- Written assignment due Wednesday, Feb. 25 --- see web site.





Outline

- What is a process?
- How is a process represented?
- Process states.
- Scheduling and context switches.
- Process creation/termination.
- Interprocess communication, synchronization.





Objectives

- To introduce the notion of a process -- a program in execution, which forms the basis of all computation
- To describe the various features of processes, including scheduling, creation and termination, and communication





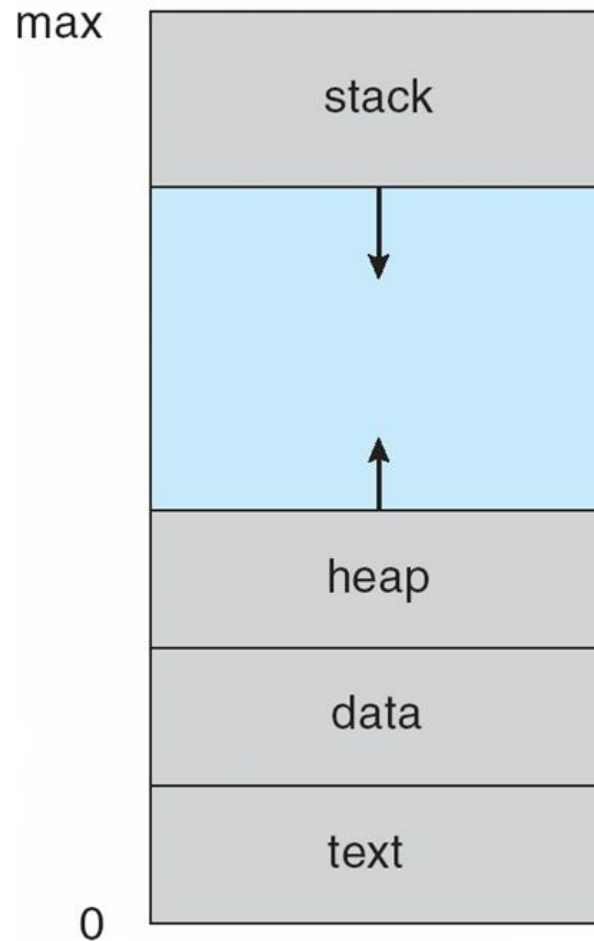
Process Concept

- 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 in Memory





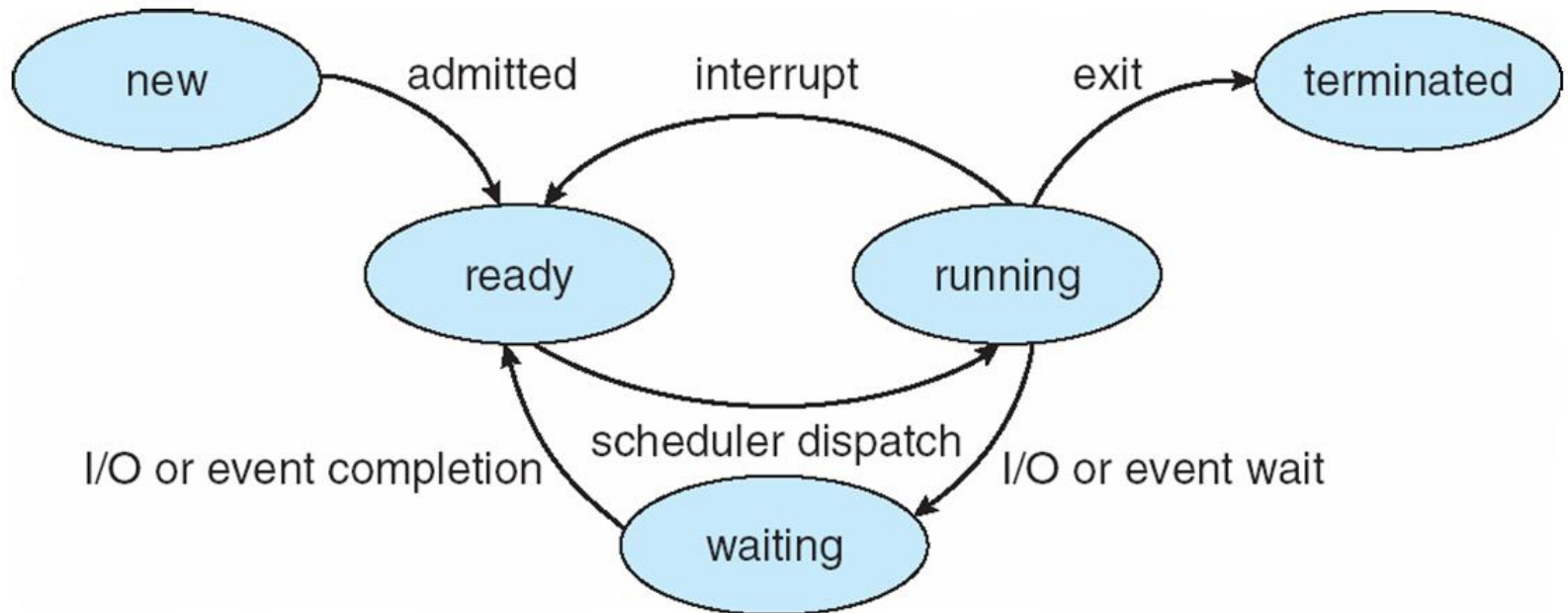
Process State

- 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

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



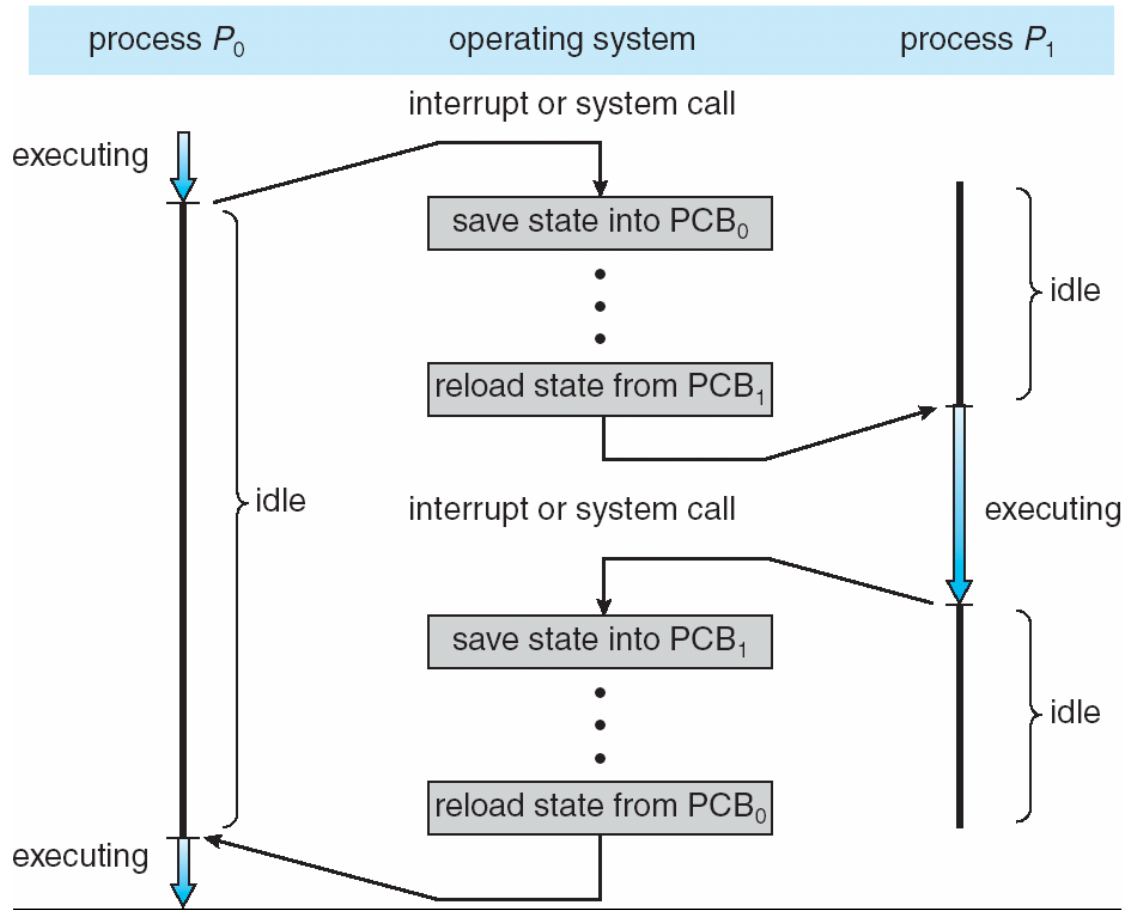


Process Control Block (PCB)





CPU Switch From Process to Process





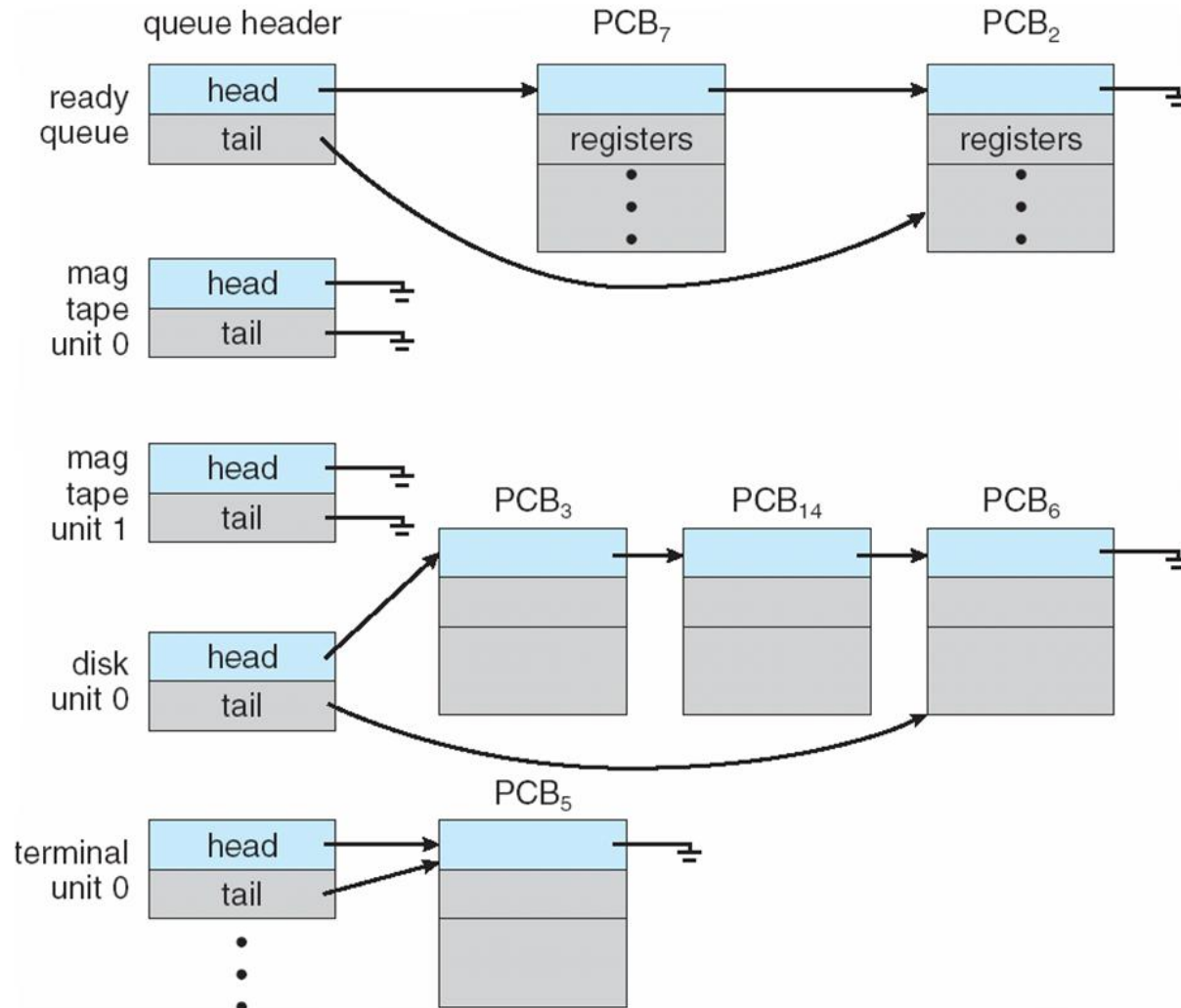
Process Scheduling Queues

- **Job queue** – set of all processes in the system (**PCB table**)
- **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



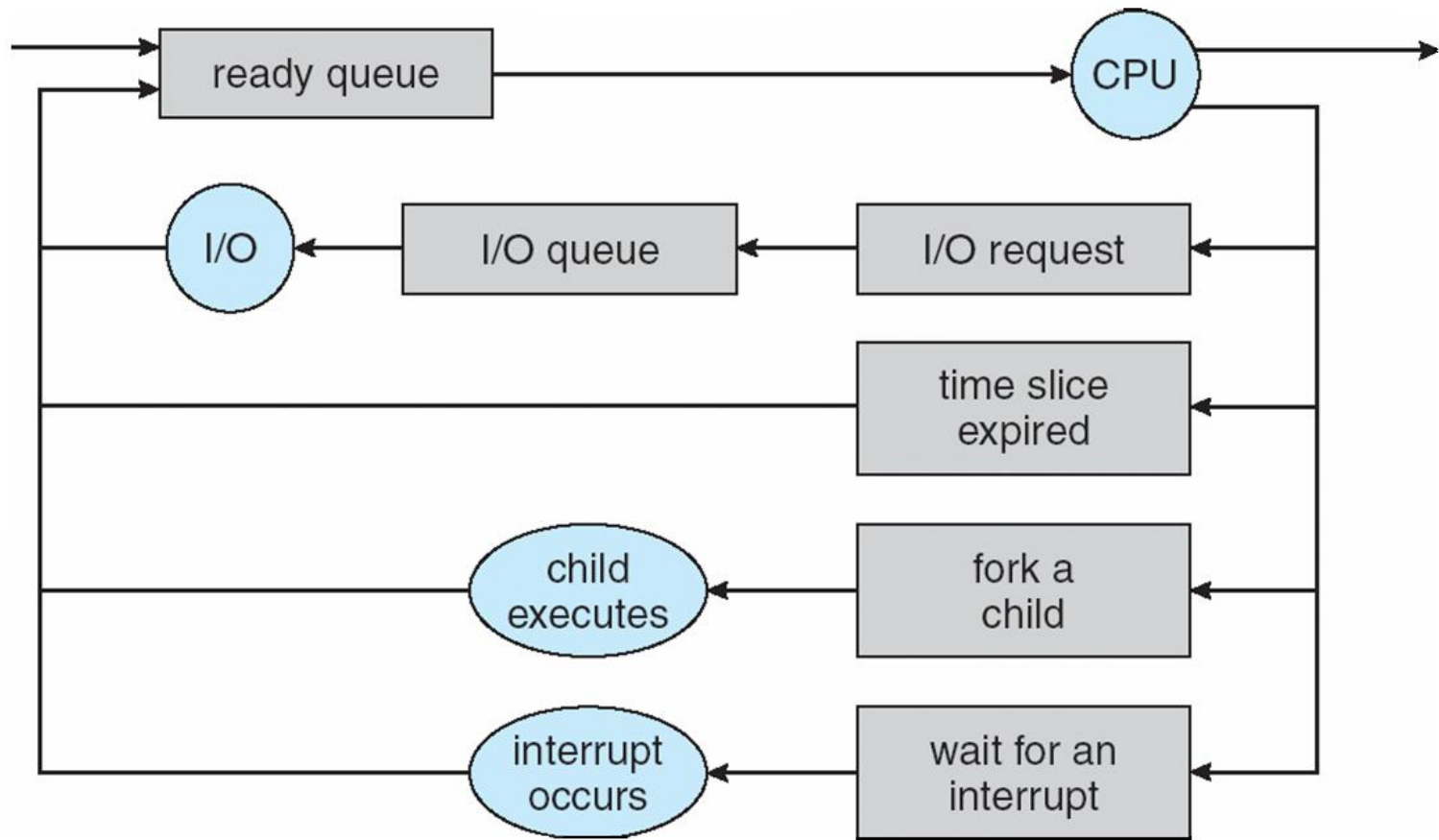


Ready Queue And Various I/O Device Queues





Representation of Process Scheduling





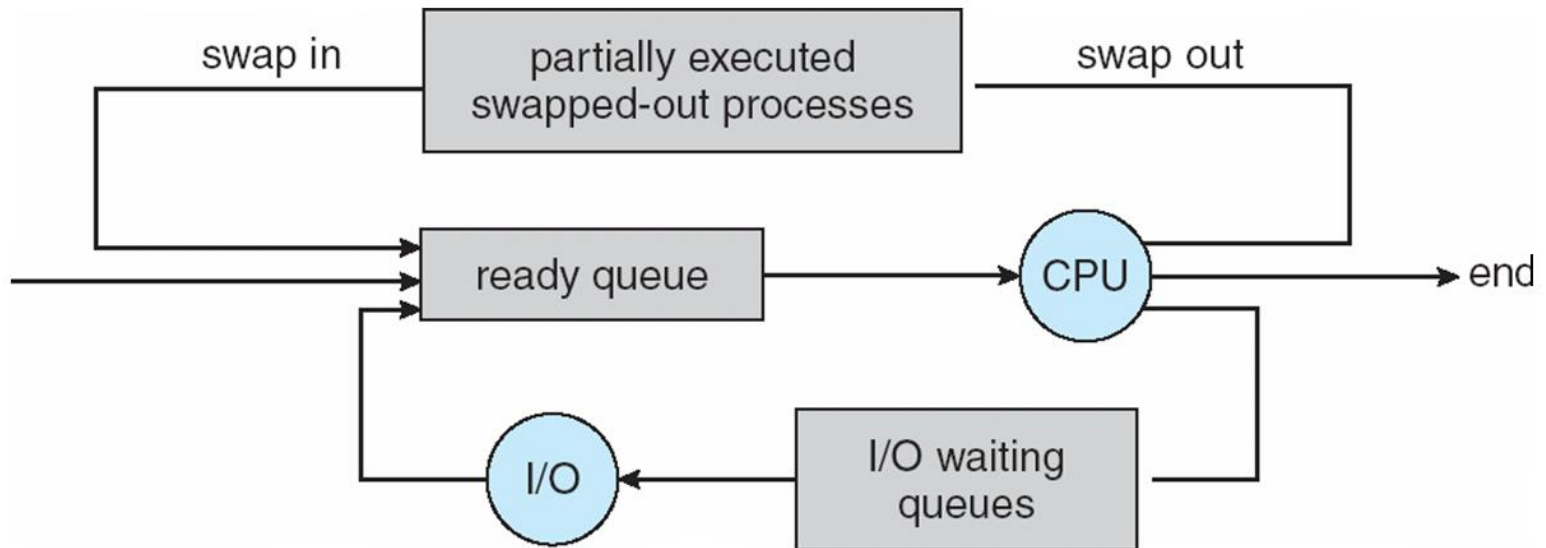
Schedulers

- **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





Addition of Medium Term Scheduling





Schedulers (Cont)

- Short-term scheduler is invoked very frequently (milliseconds) \Rightarrow (must be fast)
- Long-term scheduler is invoked very 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





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
- Time dependent on hardware support





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
 - 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





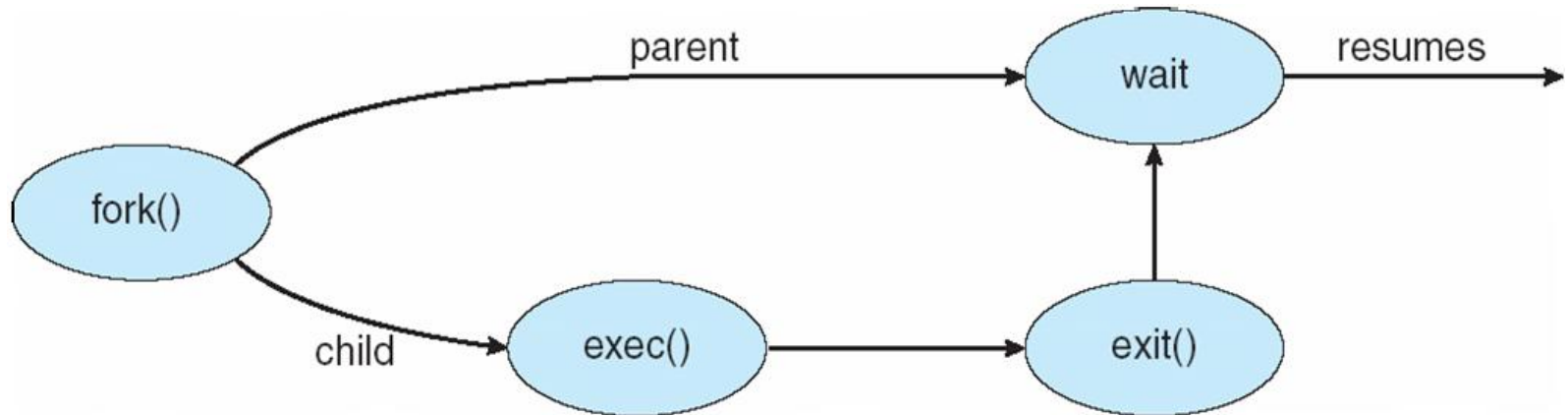
Process Creation (Cont)

- 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





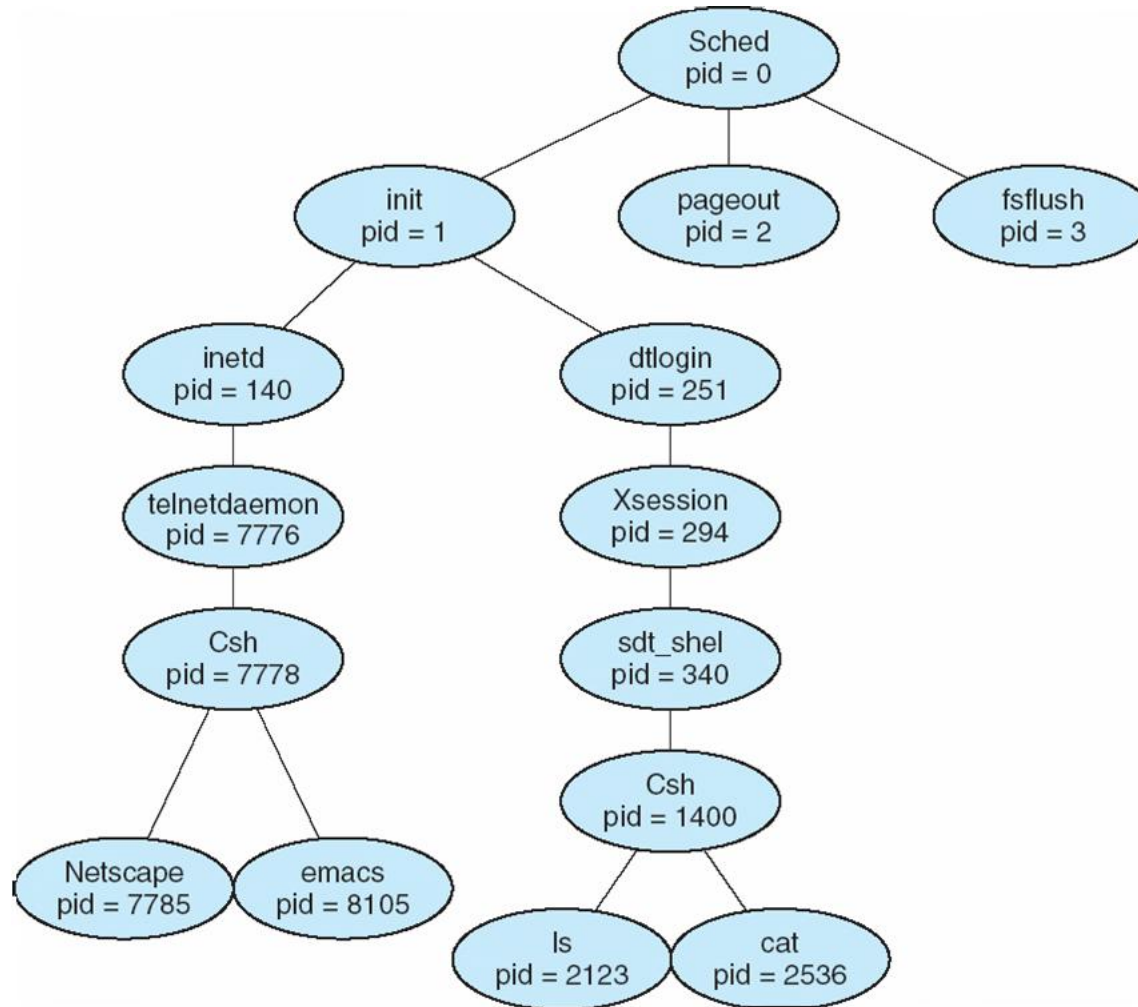
C Program Forking Separate Process

```
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**





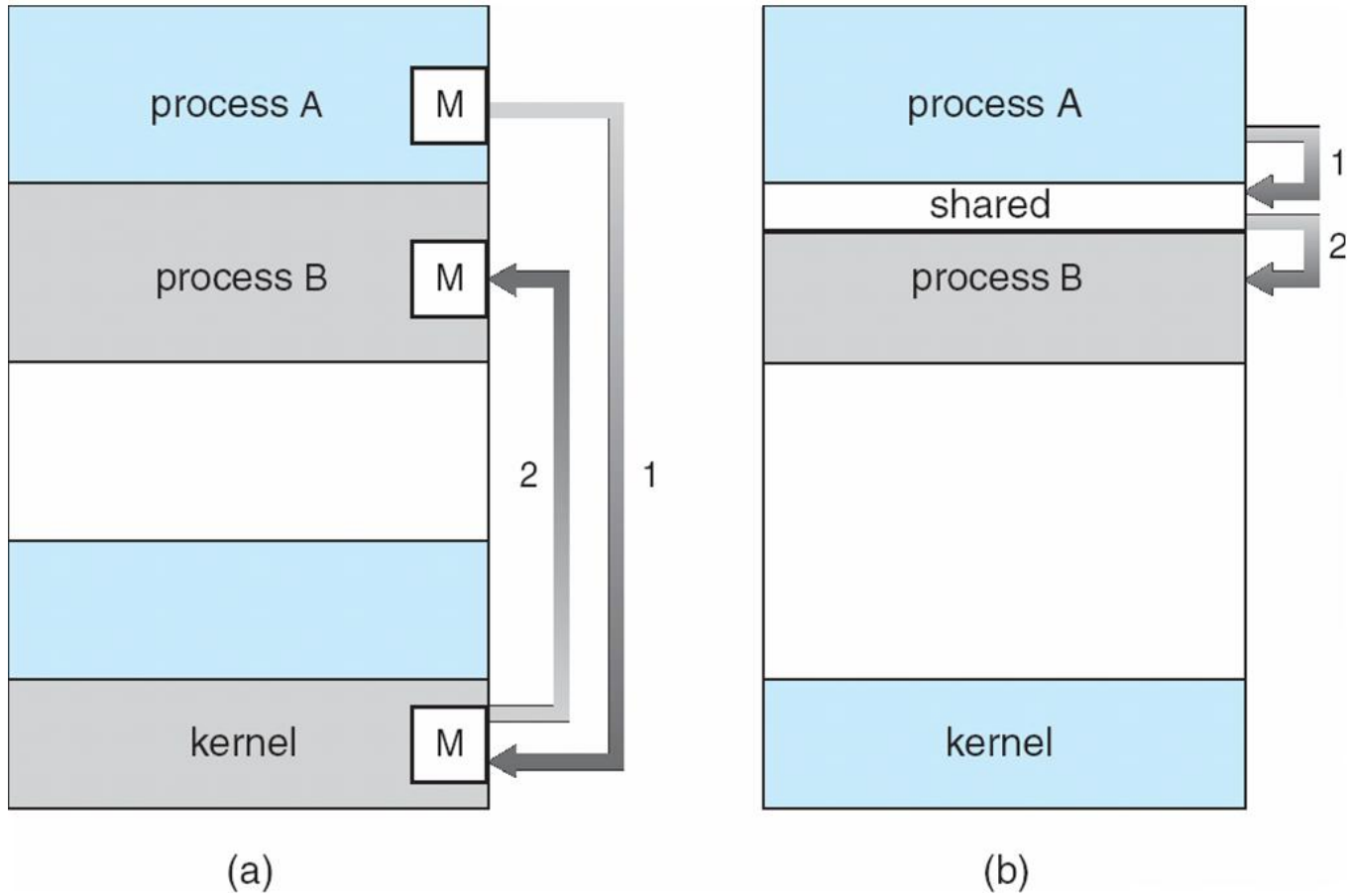
Interprocess Communication

- Processes within a system may be **independent** or **cooperating**
- Cooperating process can affect or be affected by other processes, including sharing data
- Reasons for cooperating processes:
 - Information sharing
 - Computation speedup
 - Modularity
 - Convenience
- Cooperating processes need **interprocess communication (IPC)**
- Two models of IPC
 - Shared memory
 - Message passing





Communications Models





Synchronization

- Message passing may be either blocking or non-blocking
- **Blocking** is considered **synchronous**
 - **Blocking send** has the sender block until the message is received
 - **Blocking receive** has the receiver block until a message is available
- **Non-blocking** is considered **asynchronous**
 - **Non-blocking send** has the sender send the message and continue
 - **Non-blocking receive** has the receiver receive a valid message or null





Buffering

- Queue of messages attached to the link; implemented in one of three ways
 1. Zero capacity – 0 messages
Sender must wait for receiver (rendezvous)
 2. Bounded capacity – finite length of n messages
Sender must wait if link full
 3. Unbounded capacity – infinite length
Sender never waits



End of Chapter 3

