

Chapter 4: Processes

- Process Concept
- Process Scheduling
- Operation on Processes
- Cooperating Processes
- Interprocess Communication
- Buffering

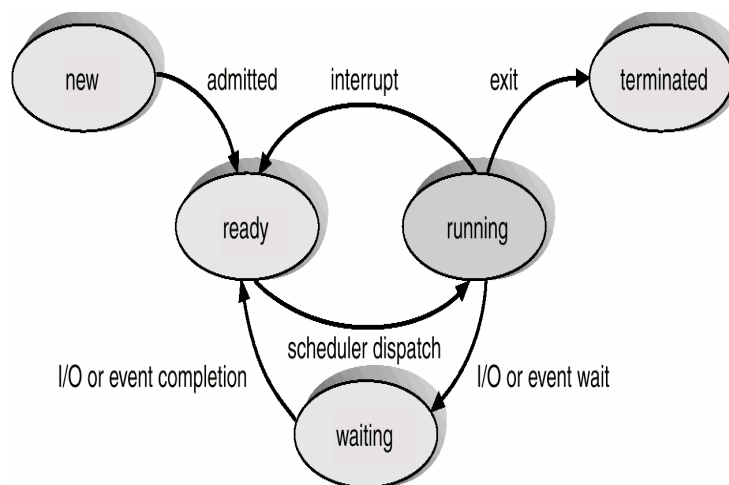
Process Concept

- 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 – specifying next instruction to be executed.
 - Stack – containing temporary data such as return address.
 - data section – containing global variables.

Process State

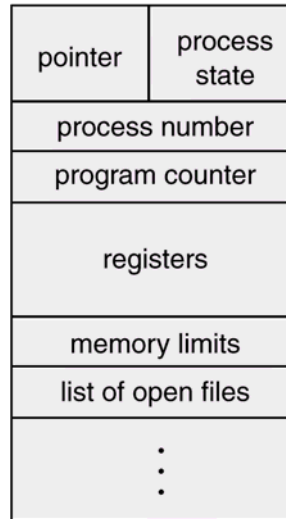
- As a process executes, it changes *state*
 - new: The process is being created.
 - running: Instructions are being executed.
 - waiting: The process is waiting for some event to occur such as I/O completion.
 - ready: The process is waiting to be assigned to a processor.
 - terminated: The process has finished execution.
- Only one process can be running on any processor at any instant.
- Many processes may be ready and waiting.

Diagram of Process State



Process Control Block (PCB)

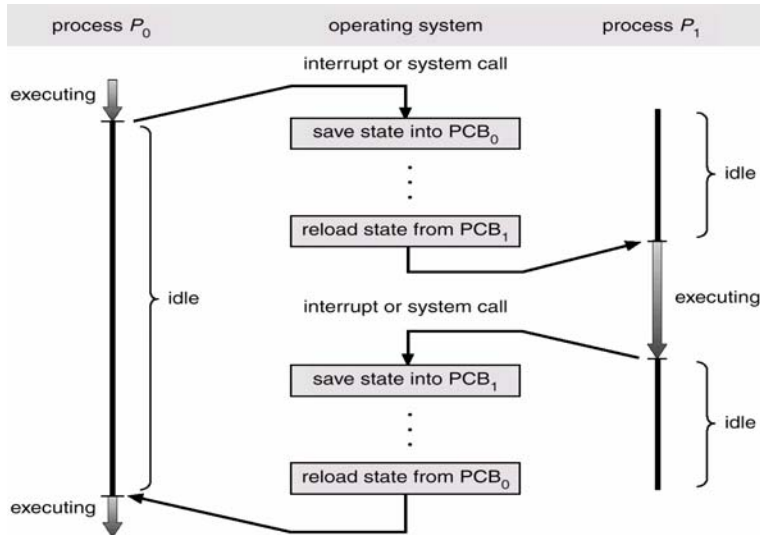
- Each process is represented in the O.S. by a Process Control Block.



Process Control Block (PCB)

- A PCB contains the following Information:
 - *Process state*: new, ready, ...
 - *Program counter*: indicates the address of the next instruction to be executed for this program.
 - *CPU registers*: includes accumulators, stack pointers, ...
 - *CPU scheduling information*: includes process priority, pointers to scheduling queues.
 - *Memory-management information*: includes the value of base and limit registers (protection) ...
 - *Accounting information*: includes amount of CPU and real time used, account numbers, process numbers, ...
 - *I/O status information*: includes list of I/O devices allocated to this process, a list of open files, ...

CPU Switch From Process to Process



Operating System Concepts

4.7

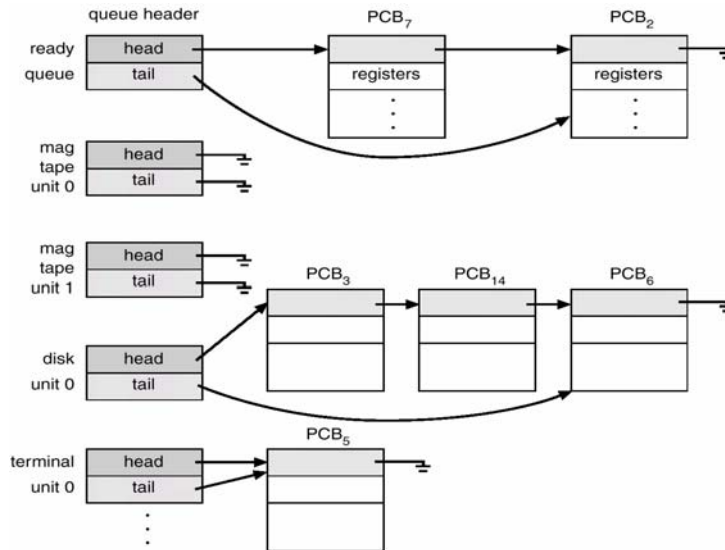
Process Scheduling Queues

- Objective of multiprogramming is to have some process running at all time to maximize CPU utilization.
- Objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running.
- For a uniprocessor system, there will never be more than one running process. If there are more processes, the rest will have to wait until the CPU is free and can be rescheduled.
- *Job queue* – set of all processes in the system.
- *Ready queue* – set of all processes residing in main memory, ready and waiting to execute. *Ready queue* is stored as linked list. A *Ready Queue Header* will contain pointers to the first and last PCBs in the list. Each PCB has a pointer field that points to the next process in the Ready Queue.
- *Device queues* – set of processes waiting for an I/O device. Each device has its own device queue.

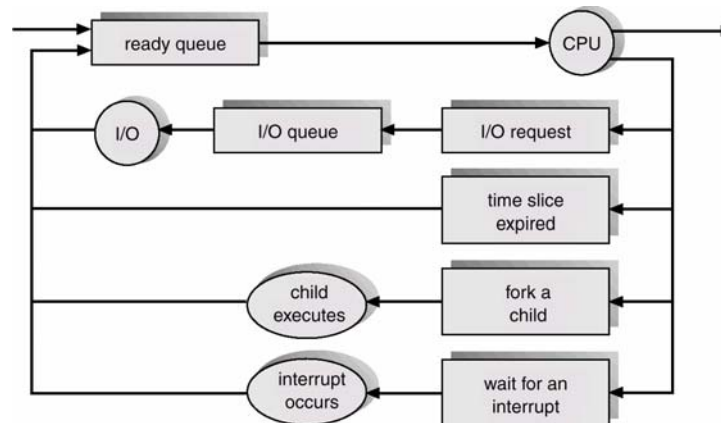
Operating System Concepts

4.8

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling



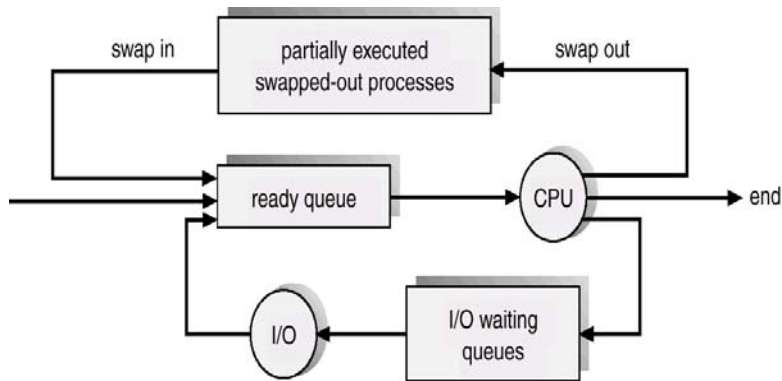
Schedulers

- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue (i.e, selects processes from pool (disk) and loads them into memory for execution).
- Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU (i.e, selects from among the processes that are ready to execute, and allocates the CPU to one of them) .

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* (the number of processes in memory).
- Medium-term scheduler – to remove processes from memory and reduce the degree of multiprogramming (the process is swapped out and swapped in by the medium-term scheduler).

Addition of Medium Term Scheduling



Schedulers (Cont.)

- 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.
- If all processes are I/O bound, the ready queue will almost always be empty and the short-scheduler will have little to do.
- If all processes are CPU bound, the I/O waiting queue will almost always be empty, devices will go unused, and the system will be unbalanced.
- To get best performance the system should have a combination of CPU and I/O bound processes.

Context Switch

- 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.
- Context-Switch Time depends on hardware support.
- Context-Switch Speed varies from machine to machine depending on memory speed, number of registers copied. The speed ranges from 1 to 1000 microsecond.

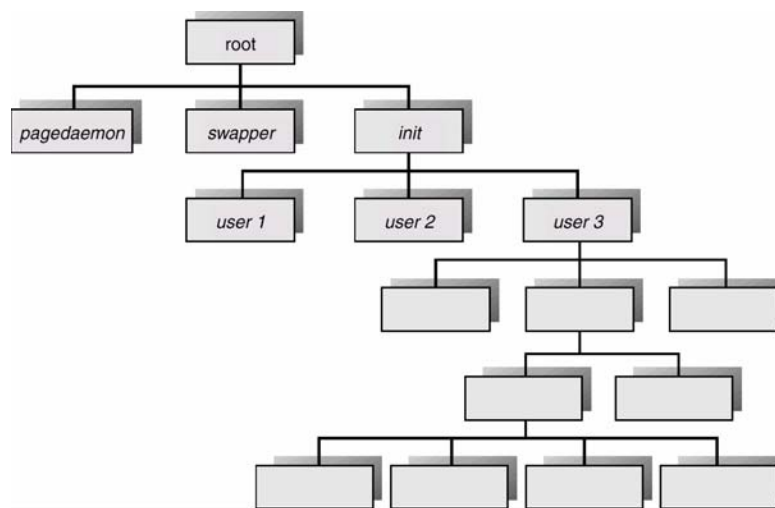
Process Creation

- A process may create several new processes, via a create-process system call, during execution.
- Parent process creates children processes, which, in turn create other processes, forming a tree of processes.
- Resource sharing, such as CPU time, memory, files, I/O devices ...
 - Parent and children share all resources.
 - Children share subset of parent's resources.
 - Parent and child share no resources.

Process Creation (Cont.)

- When a process creates a new process, two possibilities exist in terms of execution:
 - Parent and children execute concurrently.
 - Parent waits until children terminate.
- There are also two possibilities in terms of the address space of the new process:
 - Child duplicate of parent.
 - Child has a program loaded into it.
- UNIX examples
 - **fork** system call creates new process
 - **execve** system call used after a **fork** to replace the process' memory space with a new program.

A Tree of Processes On A Typical UNIX System



Process Termination

- Process executes last statement and asks the operating system to delete it by using the **exit** system call.
 - Output data from child to parent via **wait** system call.
 - Process' resources are deallocated by operating system.
- Parent may terminate execution of children processes via **abort** system call for a variety of reasons, such as:
 - Child has exceeded allocated resources.
 - Task assigned to child is no longer required.
 - Parent is exiting, and the operating system does not allow a child to continue if its parent terminates.

Cooperating Processes

- *Independent* process cannot affect or be affected by the execution of another process.
- *Cooperating* process can affect or be affected by the execution of another process.
- Any process that shares data with other processes is a cooperating process.

Cooperating Processes (Cont.)

- Advantages of process cooperation:
 - Information sharing – such as shared files.
 - Computation speed-up – to run a task faster, we must break it into subtasks, each of which will be executing in parallel. This speed up can be achieved only if the computer has multiple processing elements (such as CPUs or I/O channels).
 - Modularity – construct a system in a modular function (i.e., dividing the system functions into separate processes).
 - Convenience – one user may have many tasks to work on at one time. For example, a user may be editing, printing, and compiling in parallel.

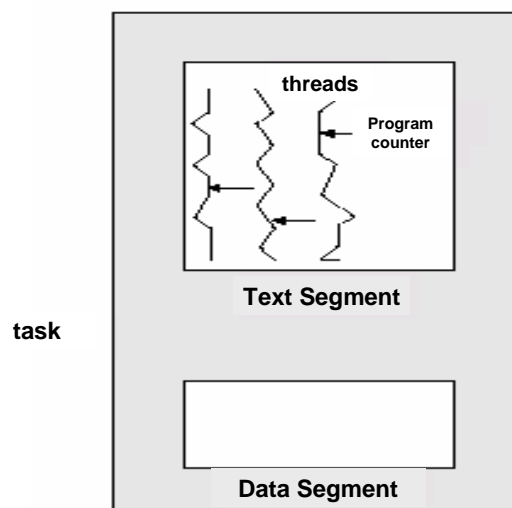
Threads

- A *thread* (or *lightweight process (LWP)*) is a basic unit of CPU utilization; it consists of:
 - program counter
 - register set
 - stack space
- A thread shares with its peer threads its:
 - code section
 - data section
 - operating-system resources collectively known as a *task*.
- A traditional or *heavyweight* process is equal to a task with one thread.

Threads (Cont.)

- In a multiple threaded task, while one server thread is blocked and waiting, a second thread in the same task can run.
 - Cooperation of multiple threads in same job confers higher throughput and improved performance.
 - Applications that require sharing a common buffer benefit from thread utilization.
- Threads provide a mechanism that allows sequential processes to make blocking system calls while also achieving parallelism.
- Kernel-supported threads (Mach and OS/2).
- User-level threads; supported above the kernel, via a set of library calls at the user level (Project Andrew from CMU).
- Hybrid approach implements both user-level and kernel-supported threads (Solaris 2).

Multiple Threads within a Task



Interprocess Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions.
- IPC is best provided by message-passing systems.
- IPC facility provides two operations:
 - **send**(*message*) – message size fixed or variable
 - **receive**(*message*)
- If *P* and *Q* wish to communicate, they need to:
 - establish a *communication link* between them
 - exchange messages via send/receive
- Processes can communicate in two ways:
 - Direct communication
 - Indirect communication

Implementation Questions

- How are links established?
- Can a link be associated with more than two processes?
- How many links can there be between every pair of communicating processes?
- What is the capacity of a link?
- Is the size of a message that the link can accommodate fixed or variable?
- Is a link unidirectional or bi-directional?

Direct Communication

- Processes must name each other explicitly:
 - **send** ($P, message$) – send a message to process P
 - **receive**($Q, message$) – receive a message from process Q
- Properties of communication link
 - Links are established automatically.
 - A link is associated with exactly one pair (two processes) of communicating processes.
 - Between each pair there exists exactly one link.
 - The link may be unidirectional, but is usually bi-directional.

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports).
 - Each mailbox has a unique id.
 - Processes can communicate only if they share a mailbox.
- Properties of communication link:
 - Link established only if processes share a common mailbox
 - A link may be associated with many processes.
 - Each pair of processes may share several communication links.
 - Link may be unidirectional or bi-directional.
- The O.S. provides a mechanism that allows a process:
 - create a new mailbox
 - send and receive messages through mailbox
 - destroy a mailbox

Indirect Communication (Continued)

- Example: Mailbox sharing
 - P_1 , P_2 , and P_3 share mailbox A.
 - P_1 sends; P_2 and P_3 receive.
 - Who gets the message?
- Solutions:
 - Allow a link to be associated with at most two processes.
 - Allow only one process at a time to execute a receive operation.
 - Allow the system to select arbitrarily the receiver. Sender is notified who the receiver was.

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.