Chapter Three

3.1. Process Management

A process can be thought of as a program in execution. A process will need certain resources such as CPU time, memory, files, and I/O devices to accomplish its task. These resources are allocated to the process either when it is created or while it is executing.

Early C/S allowed only one program to be executed at a time. This program had complete control of the system and had access to all of the system resources. Today C/S allows multiple programs to be loaded into memory and to be executed concurrently. The more complex the O.S the more it is expected to do on behalf of its users. A system therefore consists of a collection of processes:

O.S processes executing system code and user processes executing user code. By switching the CPU between processes the O.S can make the C/S more productive.

3.2. Process Concept

A question that arises in discussing operating systems involves what to call all the CPU activities. A batch system executes jobs, whereas a time-shared system has user programs, or tasks. Even on a single-user system, a user may be able to run several programs at one time: a word processor, a Web browser, and an e-mail package. And even if a user can execute only one program at a time, such as on an embedded device that does not support multitasking, the operating system may need to support its own internal programmed activities, such as memory management. In many respects, all these activities are similar, so we call all of them processes. The terms job and process are used almost interchangeably in this text.

Although we personally prefer the term process, much of operating-system theory and terminology was developed during a time when the major activity of operating systems was job processing. It would be misleading to avoid the use of commonly accepted terms that include the word job (such as job scheduling) simply because process has superseded job. The execution of a process must progress in a sequential fashion. A process is more than the program code: sometimes known as the Text section, the value of program counter and the contents of the processor's registers.

3.3. Process State

As a process executes, it changes **state**. The state of a process is defined in part by the current activity of that process. A process may be in one of the following states:

• **New**. The process is being created.

• **Running**. Instructions are being executed.

• **Waiting**. The process is waiting for some event to occur (such as an I/O completion or reception of a signal).

• **Ready**. The process is waiting to be assigned to a processor.

• **Terminated**. The process has finished execution.

These names are arbitrary, and they vary across operating systems. The states that they represent are found on all systems, however. Certain operating systems also more finely delineate process states. It is important to realize that only one process can be *running* on any processor at any instant. Many processes may be *ready* and *waiting*, however. The state diagram corresponding to these states is presented in Figure 3.1.

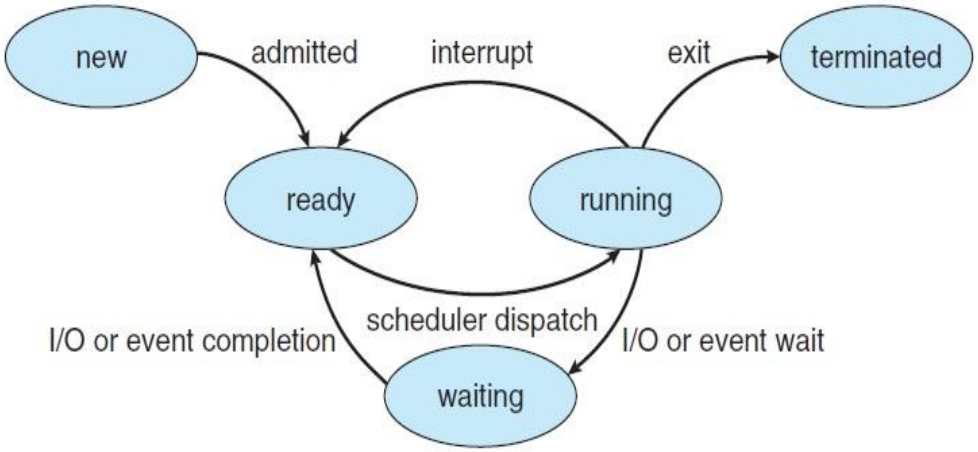

Figure 3-1 Diagram of process state

3.4. Process Control Block

A Process Control Block (PCB) also called a task control block represents each process in the O.S. A PCB is shown in the following figure 3.2. It contains many pieces of information associated with a specific process including these:

- Process state: It may be new, ready, running, waiting, halted and so on.

- Program counter: The address of the next instruction to be executed for this process.
- CPU registers: They include accumulators, index registers, stack pointer, and any general-purpose registers plus and condition-code information.
- CPU scheduling Information: It Includes a process priority, pointers to scheduling queues.
- Memory information management: It may include the value of the base and limit registers, the page tables... etc.
- Accounting information: It includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.
- I/O status information: It includes the list of I/O devices allocated to this process, a list of open files, and so on.

The PCB simply serves as the repository for any information that may vary from process to process.

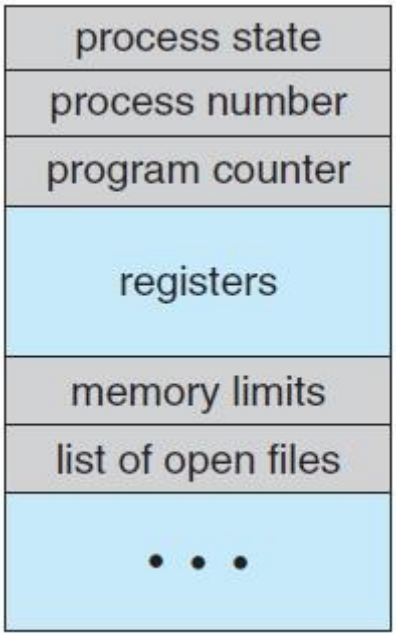

Figure 3-2 Process control block (PCB)

3.5. Process Scheduling

The objective of multiprogramming is to have some process running at all times, to maximize CPU utilization. The objective of time sharing is to switch the CPU among processes so frequently that users can interact with each program while it is running. To meet these objectives, the **process scheduler** selects an available process (possibly from a set of several available processes) for program execution on the CPU. For a single-processor 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.

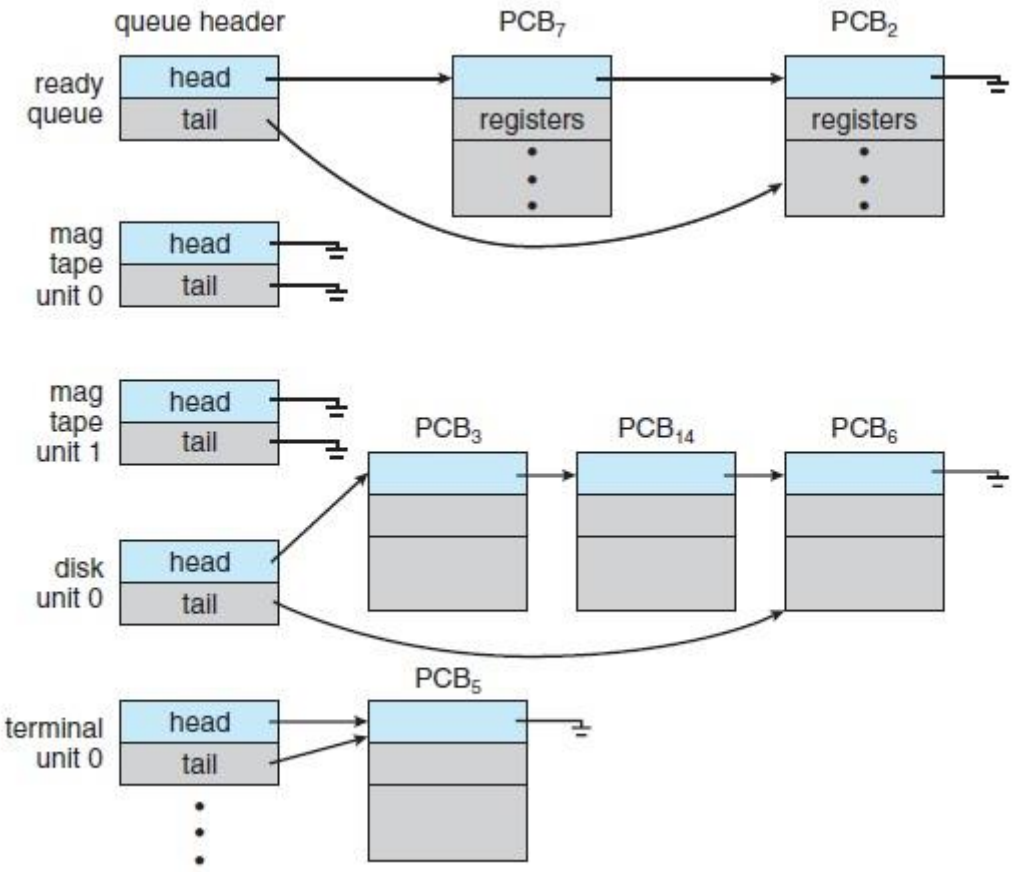

Figure 3-3 The ready queue and various I/O device queues

3.6. Scheduling Queues

- As processes enter the system, they are put into a job queue. This queue consists of all processes in the system.

- The processes that are residing in memory and are ready and waiting to execute are kept on a list called the Ready queue.
- The queue is generally stored as a linked list the queue header contains pointers to the first and last PCB's in that list.
- Each PCB has a pointer field that points to the next process in the queue. Each device has its own device queue (Figure 3.4).

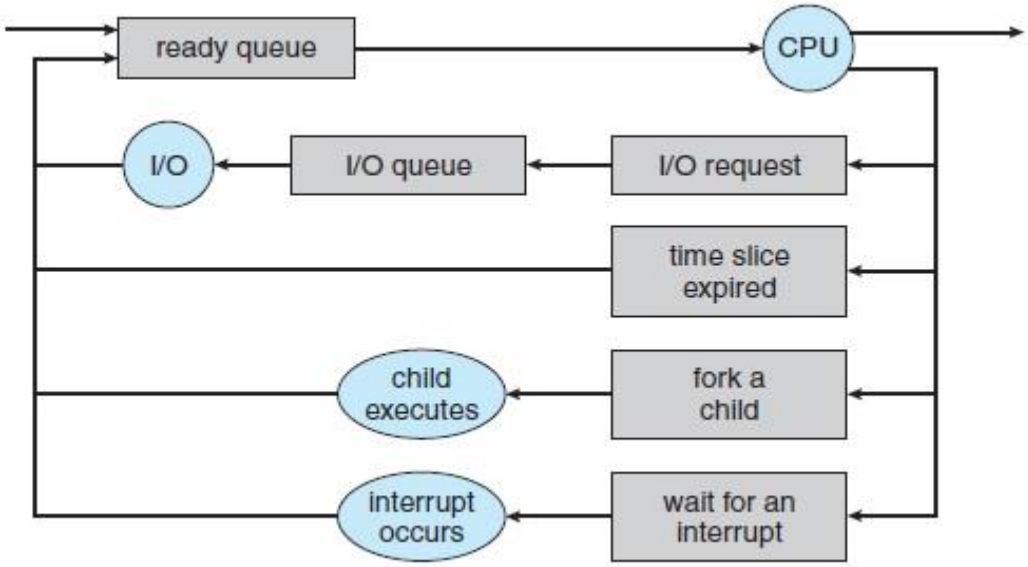

Figure 3-4 Queuing-diagram representation of process scheduling

- There are also other queues in the system; such as a list of processes waiting for a particular I/O device is called a device queue.
- A new process is initially put in the ready queue waits until it is selected for execution (or dispatched) and is given the CPU.
- Once the process is allocated the CPU and is executing one of several events could occur:

a. The process could issue an I/O request and then be placed in an I/O queue.

b. The process could create a new sub-process and wait for its termination.

c. The process could be removed forcibly from the CPU as a result of an interrupt and be put back in the ready queue.

In the first two cases, the process eventually switches from the waiting state to the ready state and is then put back in the ready queue. A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.

3.7. Scheduling levels

A process migrates between the various scheduling queues throughout its lifetime. The O.S must select processes from these queues in some fashion. The appropriate scheduler carries out the selection process. There are three levels (terms) of scheduling:

3.7.1. Long-term scheduler

The long-term scheduler or (Job scheduler) selects processes from the job pool on the disk and loads them into memory for execution. The long-term scheduler execute much less frequently there may be minutes between the creation of new processes in the system. The L.T.S control the degree of multi programming (The number of processes in memory). If the degree of multi programming is stable than the average rate of processes creation must be equal to the average departure rate of processes leaving the system.

It is important that the L.T.S .make a careful selection. In general most processes can be described as either I/O bound or CPU bound.

- An I/O bound process is one that spends more of its time doing I/O than it spends doing computations.
- A CPU-bound process is one that generates I/O requests infrequently, using more of its time doing computation than an I/O-bound process.

The L.T.S select a good process mix of I/O-bound and CPU-bound processes.

3.7.2. The short-term scheduler (or CPU Scheduler)

It is selects from among the processes that are ready to execute and allocates the CPU to one of them. The S.T.S must select a new process for the CPU quite frequently. Often the S.T.S must be very fast. If it takes 10 milliseconds to decide to executes a process for 100 milliseconds then 10/(100+10) = 9% of the CPU used (wasted) for scheduling the work. If all processes are I/O bound the ready queue will almost be empty and the S.T.S will have little to do. If all processes are CPU-bound the waiting queue will almost be empty. The system with the best performance will have a combination of CPU bound and I/0-bound processes.

3.7.3. The medium-term scheduler

Some O.S such as time-sharing systems may introduce an additional intermediate level of scheduling. The key behind the M.T.S is that sometimes it can be advantageous to remove processes from memory and thus to reduce the degree of multiprogramming. The process can be swapped out and swapped in later by the M.T.S swapping may be necessary to improve the process mix. The figure 3.5 shows the M.T.S.

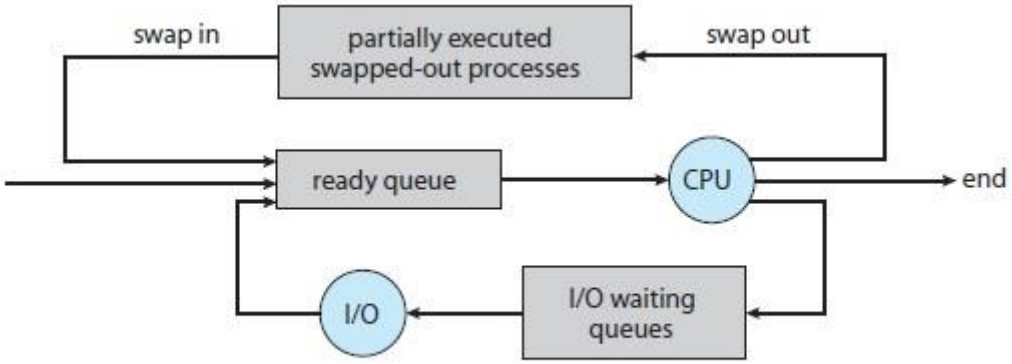

Figure 3-5 Addition of medium-term scheduling to the queuing diagram

3.8. Context Switch

- Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as a context switch.
- Context-switch time is pure overhead because the system does no useful work while switching.
- The more complex the O.S the more work must be done during a context switch.

3.9. Operations on Processes

- O.S that mange processes must be able to perform certain operation on and with processes. These include: create, destroy, suspend, resume, change a

process priority, block a process, wake up a process dispatch a process, enable a process to communicate with another process.

- Creating a process involves many operations including: name a process, insert it in the ready queue, determine the process initial priority, create the PCB and allocate the process's initial resources.
- A process may create a new process. If it does the creating process is called the parent process and the created process is called the child process.

When a process creates a new process two possibilities exist in terms of execution:

a. The parent continues to execute concurrently with its children.

b. The parent waits until some or all of its children have terminated.

- A process terminates when it finishes executing its last statement and asks the O.S to delete it by using the exist system call.
- A parent may terminate the execution of one of its children for a variety at reasons such as:

a. The child has exceeded its usage of some of the resources it has been allocated.

b. The task assigned to the child is no longer required.

c. The parent is finished and the O.S does not allow a child to continue if its parent terminated.

3.10. Cooperating processes

The concurrent processes executing in the O.S may be either independent processes or cooperating processes. A process is independent if it cannot affect or be not affected by the other processes executing in the system. Any process that does not share or any data (temporary or persistent) with any other process is independent. A process is cooperating if it can affect or be affected by the other processes executing in the system or any process that share data with other processes is a cooperating process. There are several reasons for providing an environment that allows process cooperation:

1. Information sharing. 2. Computation speedup.

3. Modularity: Dividing the system functions into separate processes.

4. Convenience, Many tasks to work on at one time, a user may be editing, printing and compiling in parallel.

To illustrate the concept of cooperating processes let us consider the producer-consumer problem as an example of cooperating processes. A produce process produces information that is consumed by a consumer process. For example a print program produces characters that are consumed by the printer driver. To allow producer and consumer to run concurrently we must have a buffer of item that can be filled by the producer and emptied by the consumer. A producer can produce one item while the consumer is consuming another item. The producer and consumer must be synchronized. The consumer must wait until an item is produced (the buffer is empty) and the producer must wait if the buffer is full.

In the bounded-buffer and be one solution for the producer and consumer processes share the following variables:

var n;

type item = ...;

var buffer: array [0 .. n-l] of item;

in out: 0 .. n-1

With in, out initialized to the value 0. The shared buffer is implemented as a circular array with two logical pointers: in and out.

in points to the next free position in the buffer; out points to the first full position in the buffer.

The buffer is empty when in=out; the buffer is full when in+1 mod n=out.

The producer process has a local variable nextp in which the new item to be produced is stored.

Repeat

……

produce an item in nextp

……

while in+1 mod n=out do no-op;

buffer [in] :=nextp;

in:=in+l mod n;

until false;

The consumer process has a local variable nextc in which the item to be consumed is

stored;

repeat do-nothing instruction

while in=out do no-op;

nextc:= buffer [out],

out:=out+1 mod n;

……

consume the item in nextc; until false;

3.11. Thread structure

A thread sometimes called light weight process (LWP) is a basic unit of CPU utilization and consists of a program counter, a register set, and a stack space. It shares with peer threads its code section, data section and O.S resources such as open files and signals collectively known as a task. A traditional or heavy weight process is equal to a task with one thread.

Threads can be in one of several states ready, blocked, running, or terminated. Threads can create child threads if one thread is blocked another thread can run. Unlike processes threads are not independent of one another, because all threads can access in the task.

3.12. Interrupt Processing

An interrupt is an event that alters the sequence in which a processor executes instructions. The H/W of C/S generates the interrupt. When an interrupt occurs the following actions will be taken:

a. The O.S gains control.

b. The O.S saves the state of interrupted process in its PCB.

c. The O.S analyzes the interrupt and passes control to the appropriate routine to handle the interrupt.

d. The interrupt handler routine processes the interrupt.(IHR)

e. The state of the interrupted process (or some other next process) is restored.

f. The interrupted process (or some other next process) executes.

An interrupt may be specifically initiated by a running process (in which case it is often called a trap and said to be synchronous with the operation of the process). Or it may be caused by some event that may or may not be related to the running process. It is said to be asynchronous with the operation of the process.

3.13. Interrupt Classes (types)

There are six interrupt classes. These are:

3.13.1. SVC (Supervisor call) interrupts

A running process that executes the SVC instruction such as initiates these:

- I/O request. - Obtaining more storage. - Communicating with user operator.

3.13.2. I/O interrupts:

There are initiated by the I/O H/W. such as:

- An I/O operation completes.

- An I/O error occurs.

- When a device is made ready.

3.13.3. External interrupts:

These are caused by various events including: the expiration of a quantum on an interrupting clock.

- Pressing of the console's interrupt key by the operator.

- Receipt of a signal from another processor.

3.13.4. Restart interrupts:

These occur when the operator:

- Presses the console's restart bottom.

- When a restart signal processor instruction arrives from another processor on a multi-processor system.

3.13.5. Program checks interrupt:

These are caused by many problems such as:

- Divide by zero.

- Arithmetic overflow.

- Data is in the wrong format.

- Attempt to execute invalid operation code.

- Attempt to reference a memory location beyond the limits of main memory.

- Attempt to execute a privileged instruction.

- Attempt to reference a protected resource.