جامعة ألأنبار

كلية علوم الحاسوب وتكنولوجيا المعلومات

قسم أنظمة شبكات الحاسوب

المرحلة الرابعه

Operating System

التدريسي: أ.م.د عمر منذر حسين

- 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.
- Most modern operating systems now support processes that have multiple threads.
- The operating system is responsible for several important aspects of process and thread management: the creation and deletion of both user and system processes; the scheduling of processes; and the provision of mechanisms for synchronization, communication, and deadlock handling for processes.

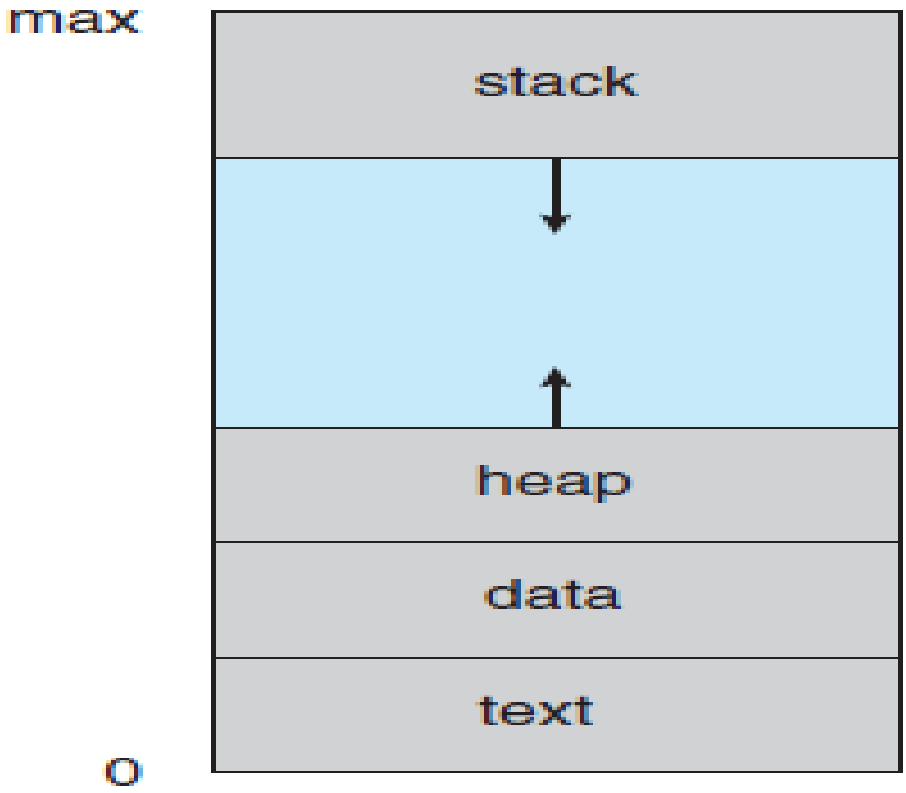
The Process

- A process is more than the program code, which is sometimes known as the **text section.** It also includes:

1. the **current activity**, as represented by the value of the program counter and the contents of the processor's registers.
2. the **process stack**, which contains temporary data (such as function parameters, return addresses, and local variables).
3. a **data section**, which contains global variables.
4. A process may also include a **heap**, which is memory that is dynamically allocated during process run time.

   The structure of a process in memory is shown in the next Figure.

- A program becomes a process when an executable file is loaded into memory.



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.
- 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.

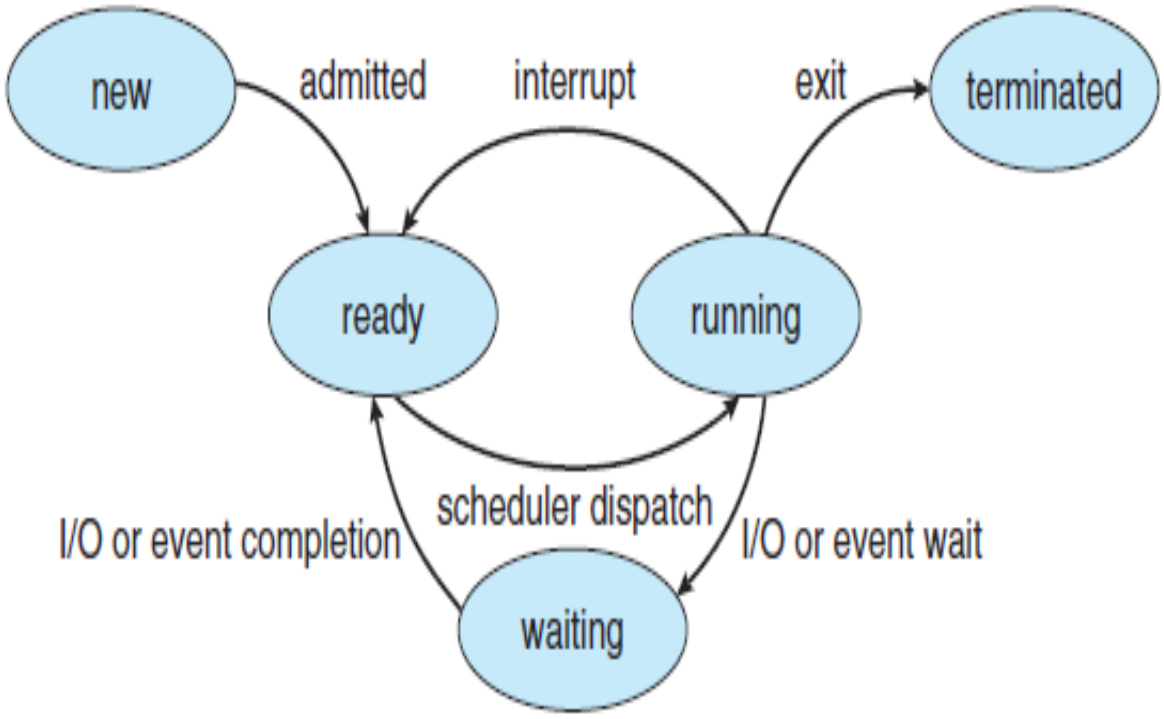


Figure 3.2 Diagram of process state.

Process Control Block

- Each process is represented in the operating system by a **process control block (PCB)**—also called a **task control block**.
- It contains many pieces of information associated with a specific process, including:
  - Process state. The state may be new, ready, running, waiting, halted, and so on.
  - Program counter. The counter indicates the address of the next instruction to be executed for this process.
  - CPU registers. They include accumulators, index registers, stack pointers, and general-purpose registers, plus any condition-code

information. Along with the program counter, this state information must be saved when an interrupt occurs, to allow the process to be continued correctly afterward (Figure 3.4).

– CPU-scheduling information. This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

– Memory-management information. This information may include such items as the value of the base and limit registers and the page tables, or the segment tables, depending on the memory system used by the operating system.

– Accounting information. This information includes the amount of CPU and real time used, time limits, account numbers, job or process numbers, and so on.

– I/O status information. This information includes the list of I/O devices allocated to the process, a list of open files, and so on.

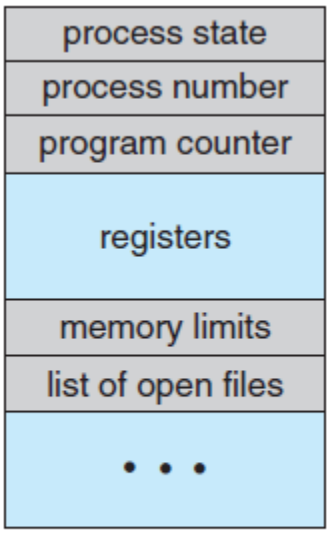


Figure 3.3 Process control block (PCB).

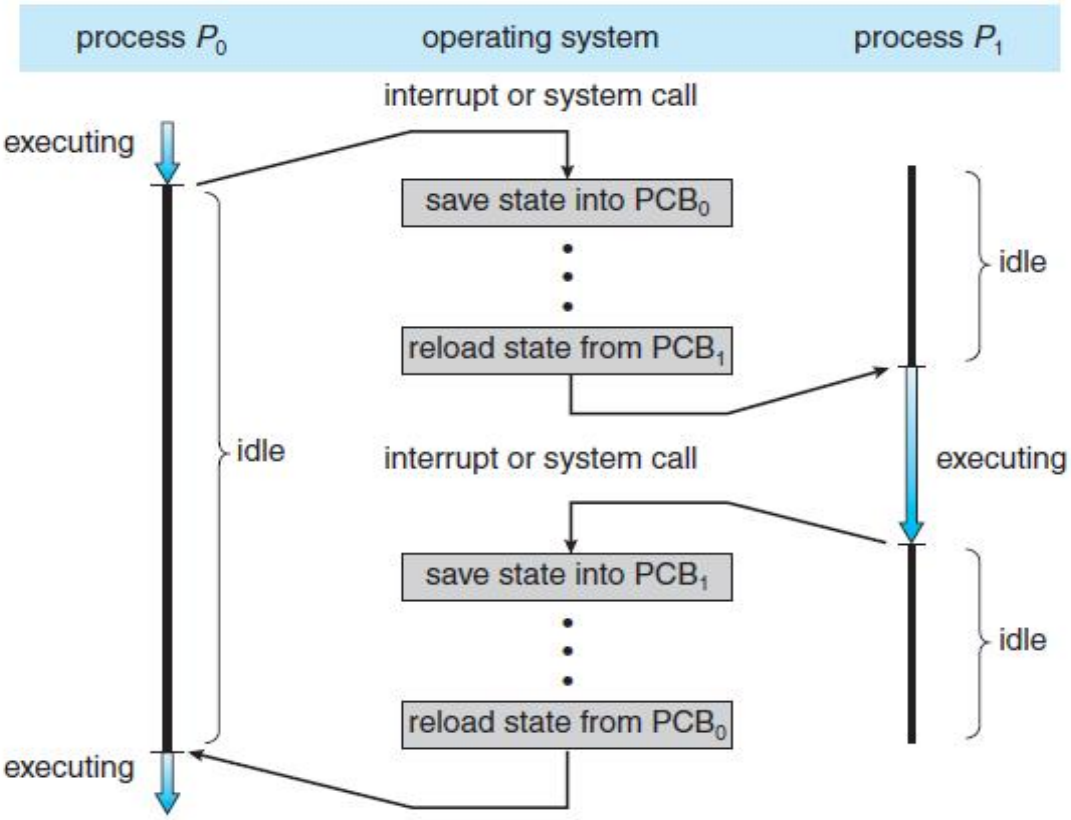


Figure 3.4 Diagram showing CPU switch from process to process.

Threads

- This single thread of control allows the process to perform only one task at a time.
- The user cannot simultaneously type in characters and run the spell checker within the same process, for example.
- Most modern operating systems have extended the process concept to allow a process to have multiple **threads** of execution and thus to perform more than one task at a time.
- On a system that supports threads, the PCB is expanded to include information for each thread. Other changes throughout the system are also needed to support threads.

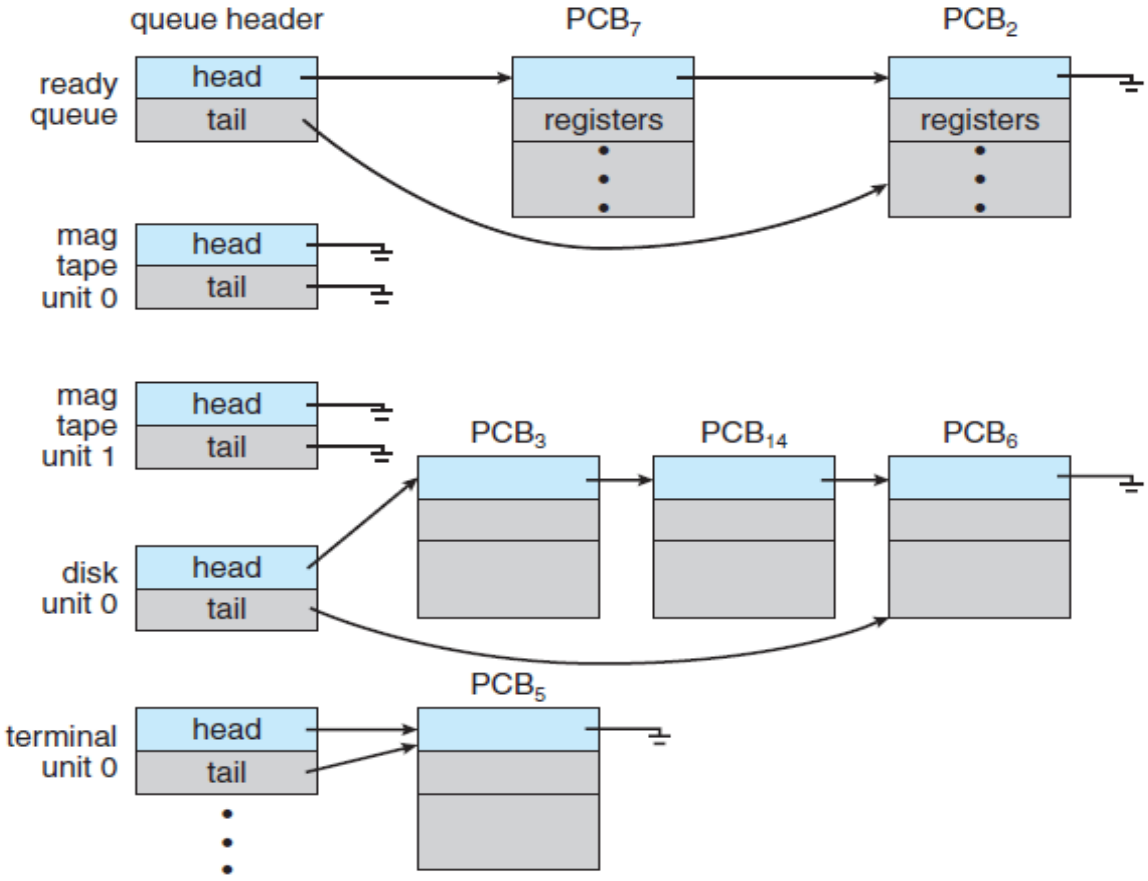
Process Scheduling

- 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.

Scheduling Queues

- As processes enter the system, they are put into a **job queue**, which consists of all processes in the system.
- The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the **ready queue**.
- This queue is generally stored as a linked list.
- A ready-queue header contains **pointers** to the first and final PCBs in the list.
- Each PCB includes a pointer field that points to the next PCB in the ready queue.
- The system also includes other queues.
- When a process is allocated the CPU, it executes for a while and eventually quits, is interrupted, or waits for the occurrence of a particular event, such as the completion of an I/O request.
- Suppose the process makes an I/O request to a shared device, such as a disk.
- Since there are many processes in the system, the disk may be busy with the I/O request of some other process. The process therefore may have to wait for the disk.

- The list of processes waiting for a particular I/O device is called a **device queue**. Each device has its own device queue (Figure).



- A common representation of process scheduling is a **queueing diagram**, such as that in Figure 3.6.
- Each rectangular box represents a queue.
- Two types of queues are present: the ready queue and a set of device queues.
- The circles represent the resources that serve the queues, and the arrows indicate the flow of processes in the system.
- A new process is initially put in the ready queue. It waits there until it is selected for execution, or **dispatched**. Once the process is allocated the CPU and is executing, one of several events could occur:
- The process could issue an I/O request and then be placed in an I/O queue.
- The process could create a new child process and wait for the child's termination.

- 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.

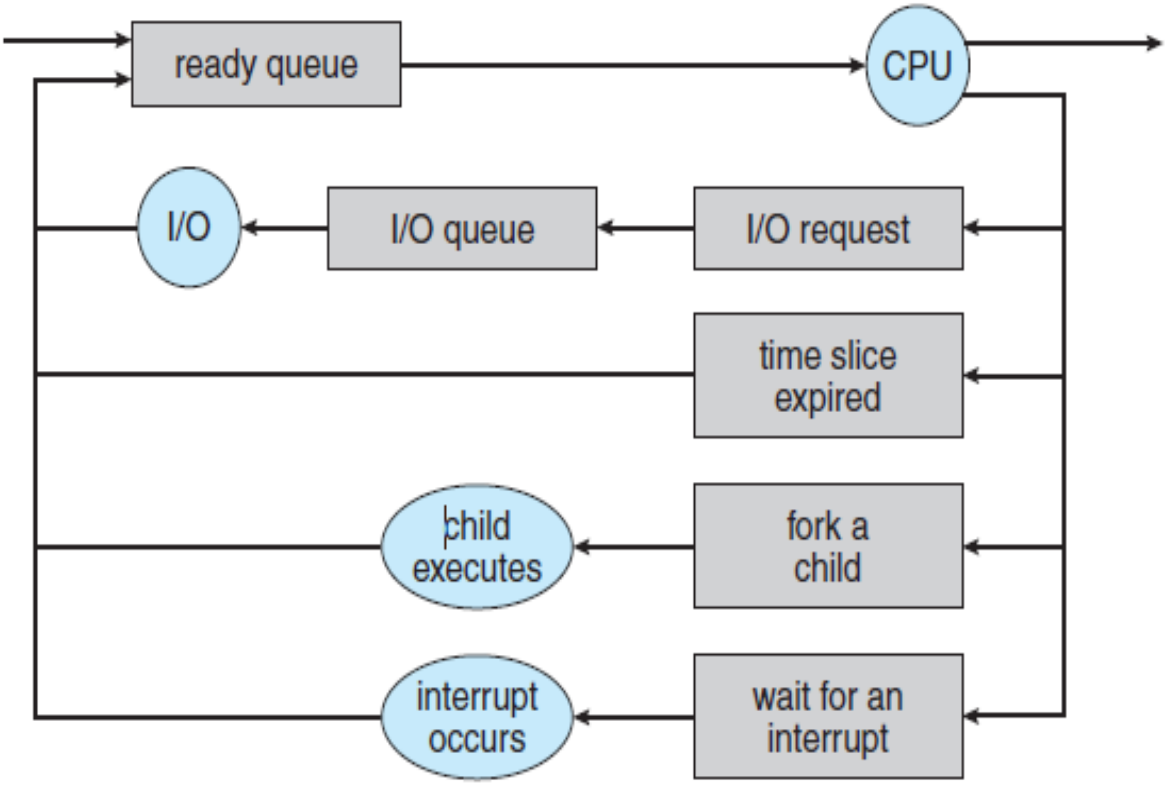


Figure 3.6 Queueing-diagram representation of process scheduling.

Schedulers

- A process migrates among the various scheduling queues throughout its lifetime. The operating system must select, for scheduling purposes, processes from these queues in some fashion.
- The selection process is carried out by the appropriate **scheduler**.

- The long-term scheduler, or job scheduler, selects processes from the pool (i.e. disk) and loads them into memory for execution.
- The **short-term scheduler**, or **CPU scheduler**, selects from among the processes that are ready to execute and allocates the CPU to one of them.
- The primary distinction between these two schedulers lies in frequency of execution.
- The short-term scheduler must select a new process for the CPU frequently.
- A process may execute for only a few milliseconds before waiting for an I/O request.
- Often, the short-term scheduler executes at least once every 100 milliseconds. Because of the short time between executions, the short-term scheduler must be fast.
- If it takes 10 milliseconds to decide to execute a process for 100 milliseconds, then 10/(100 + 10) = 9 percent of the CPU is being used (wasted) simply for scheduling the work.
- The long-term scheduler executes much less frequently; minutes may separate the creation of one new process and the next.
- It is important that the long-term scheduler 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, in contrast, generates I/O requests infrequently, using more of its time doing computations.
- It is important that the long-term scheduler select a good process mix of I/O-bound and CPU-bound processes.
- If all processes are I/O bound, the ready queue will almost always be empty, and the short-term 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 again the system will be unbalanced.
- The system with the best performance will thus have a combination of CPU-bound and I/O-bound processes.