WEEK 2 : LECTURE 2 - PROCESSOR & PROCESS MANAGEMENT

Bachelor's Degree in Information Technology
BITOS4111, OPERATING SYSTEMS
Trimester: 05
From **July 2024** to **October 2024**

OBJECTIVES/TOPICS

- Process Concepts.
- Operations on Processes.
- Threads.
- Inter Process Communication.
- Process Scheduling.
- Scheduling Algorithms.
- Multiple -Processor Scheduling.
- Thread Scheduling.

PROCESS CONCEPTS.

- a process is a program in execution

- The status of the current activity of a process is represented by the value of the program counter & the contents of the processor's registers.

- The memory layout of a process is typically divided into multiple sections as shown in the next diagram

PROCESS CONCEPTS.

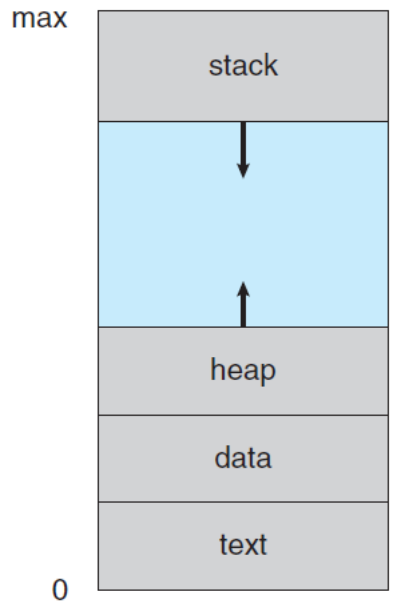


Figure 3.1 Layout of a process in memory.

- **Text section** —the executable code
- **Data section**—global variables
- **Heap section** —memory that is dynamically allocated during program run time
- **Stack section** —temporary data storage when invoking functions (such as function parameters, return addresses, & local variables)

PROCESS CONCEPTS.

- The sizes of the text and data sections are fixed, as their sizes do not change during program run time.

- However, the stack and heap sections can shrink and grow dynamically during program execution

- Each time a function is called, an activation record containing function parameters, local variables, & the return address is pushed onto the stack
    - when control is returned from the function, the activation record is popped from the stack.

PROCESS CONCEPTS.

- Similarly, the heap will grow as memory is dynamically allocated, and will shrink when memory is returned to the system.

- Although the stack and heap sections grow toward one another, the operating system must ensure they do not overlap one another.

PROCESS CONCEPTS.

- a program by itself is not a process
  - A program is a passive entity, such as a file containing a list of instructions stored on disk (often called an executable file).

- In contrast, a process is an active entity, with a program counter specifying the next instruction to execute & a set of associated resources

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

PROCESS CONCEPTS.

- Although two processes may be associated with the same program, they are nevertheless considered two separate execution sequences
  - E.g. several users may be running different copies of the mail program, or the same user may invoke many copies of the web browser program.

  - Each of these is a separate process; and although the text sections are equivalent, the data, heap, and stack sections vary

PROCESS CONCEPTS : Process State

- As a process executes, it changes state.
- A process state 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 (e.g. 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.

PROCESS CONCEPTS : Process State

- The states are illustrated in the diagram below:

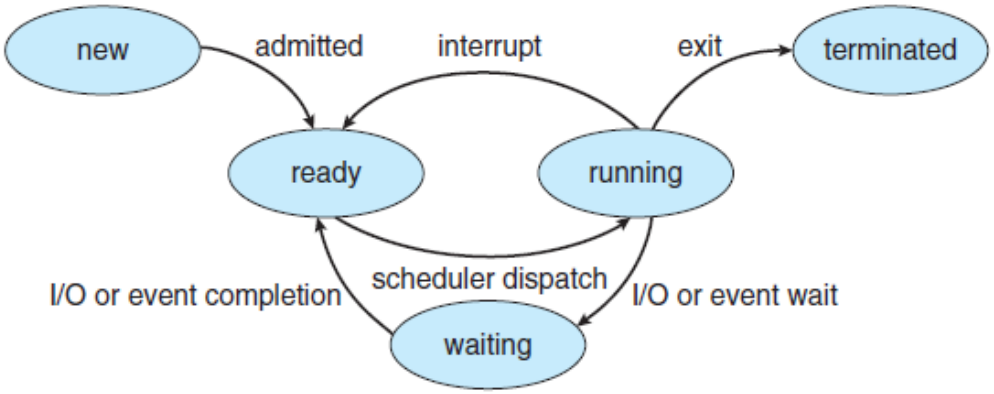


Figure 3.2 Diagram of process state.

- It is important to realize that only one process can be running on any processor core at any instant.
- Many processes may be ready and waiting, however.

PROCESS CONCEPTS : Process Control Block

- Each process is represented in the operating system by a process control block (PCB)—also called a task control block, as shown below:

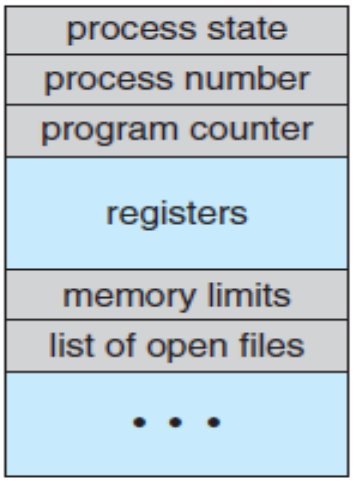


Figure 3.3 Process control block (PCB).

PROCESS CONCEPTS : Process Control Block

- It contains many pieces of information associated with a specific process, including these:
  - **1. Process state** : The state may be new, ready, running, waiting, halted, and so on.
  - **2. Program counter :** The counter indicates the address of the next instruction to be executed for this process.
  - **3. CPU registers :** The registers vary in number and type, depending on the computer architecture

PROCESS CONCEPTS : Process Control Block

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

- **5. 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

PROCESS CONCEPTS : Process Control Block

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

- **7. 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.

- In brief, the PCB simply serves as the repository for all the data needed to start, or restart, a process, along with some accounting data.

OPERATIONS ON PROCESSES

1. Process Creation

- During the course of execution, a process may create several new processes

- 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

OPERATIONS ON PROCESSES

1. Process Creation

- Most operating systems (including UNIX, Linux, and Windows) identify processes according to a unique process identifier (or pid), which is typically an integer number.

- The pid provides a unique value for each process in the system, and it can be used as an index to access various attributes of a process within the kernel.

OPERATIONS ON PROCESSES

1. Process Creation

- An example of a tree of processes on a typical Linux system is show in the diagram below.

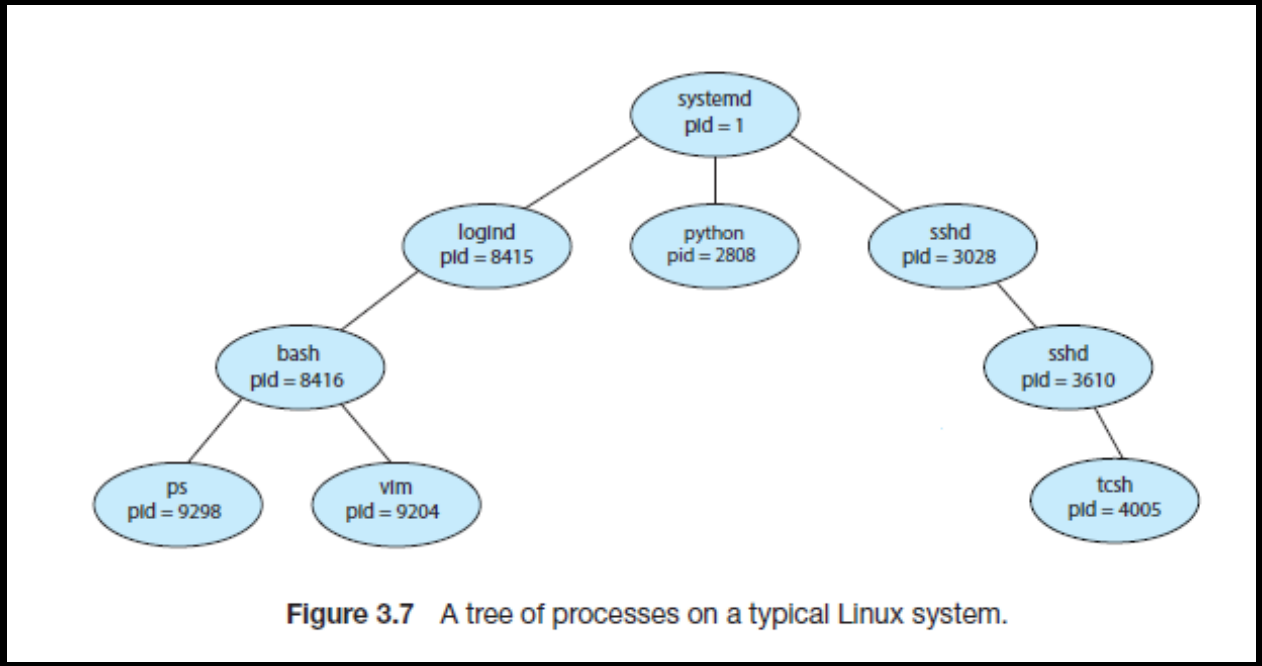


Figure 3.7 A tree of processes on a typical Linux system.

OPERATIONS ON PROCESSES

1. Process Creation

- The system process (which always has a pid of 1) serves as the root parent process for all user processes, and is the first user process created when the system boots.

- Once the system has booted, the system process creates processes which provide additional services such as a web or print server, an ssh server etc.

- And so on.

OPERATIONS ON PROCESSES

1. Process Creation

- When a process creates a new process, two possibilities for execution exist:
  - 1. The parent continues to execute concurrently with its children.
  - 2. The parent waits until some or all of its children have terminated.

OPERATIONS ON PROCESSES

1. Process Creation

- There are also two address-space possibilities for the new process:
  - 1. The child process is a duplicate of the parent process (it has the same program and data as the parent).
  - 2. The child process has a new program loaded into it.

OPERATIONS ON PROCESSES

2. Process Termination

- A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the exit() system call

- At that point, the process may return a status value to its waiting parent process (via the wait() system call).

- All the resources of the process —including physical and virtual memory, open files, and I/O buffers—are deallocated and reclaimed by the operating system.

OPERATIONS ON PROCESSES

2. Process Termination

- Termination can occur in other circumstances as well.
  - A process can cause the termination of another process via an appropriate system call (e.g., TerminateProcess() in Windows).
  - Usually, such a system call can be invoked only by the parent of the process that is to be terminated
  - A user or a misbehaving application could arbitrarily kill another user's processes.

OPERATIONS ON PROCESSES

2. Process Termination

- a parent needs to know the identities of its children if it to terminate them.
  - Thus, when one process creates a new process, the identity of the newly created process is passed to the parent.

OPERATIONS ON PROCESSES

2. Process Termination

- A parent may terminate the execution of one of its children for a variety of reasons, such as these:
  - The child has exceeded its usage of some of the resources that it has been allocated.
  - The task assigned to the child is no longer required.
  - The parent is exiting, and the operating system does not allow a child to continue if its parent terminates.

OPERATIONS ON PROCESSES

2. Process Termination

- Some systems do not allow a child to exist if its parent has terminated.

- In such systems, if a process terminates (either normally or abnormally), then all its children must also be terminated.

- This phenomenon, referred to as cascading termination, is normally initiated by the operating system.

OPERATIONS ON PROCESSES

2. Process Termination

- in Linux & UNIX systems, we can terminate a process by using the exit() system call, providing an exit status as a parameter. As shown below:

```
/* exit with status 1 */
exit(1);
```

- A parent process may wait for the termination of a child process by using the wait() system call

OPERATIONS ON PROCESSES

2. Process Termination

- The wait() system call is passed a parameter that allows the parent to obtain the exit status of the child

- This system call also returns the process identifier of the terminated child so that the parent can tell which of its children has terminated.

```
pid_t pid;
int status;

pid = wait(&status);
```

OPERATIONS ON PROCESSES

2. Process Termination

- When a process terminates, its resources are deallocated by the operating system.

- However, its entry in the process table must remain there until the parent calls wait(), because the process table contains the process's exit status.

- A process that has terminated, but whose parent has not yet called wait(), is known as a zombie process.

OPERATIONS ON PROCESSES

2. Process Termination

- Now consider what would happen if a parent did not invoke wait() and instead terminated, thereby leaving its child processes as orphans

OPERATIONS ON PROCESSES

2. Process Termination : Android Process Hierarchy

- Because of resource constraints such as limited memory, mobile operating systems may have to terminate existing processes to reclaim limited system resources.

- Rather than terminating an arbitrary process, Android has identified an importance hierarchy of processes, and when the system must terminate a process to make resources available for a new, or more important, process, it terminates processes in order of increasing importance.

OPERATIONS ON PROCESSES

2. Process Termination : Android Process Hierarchy

- From most to least important, the hierarchy of process classifications is as follows:
  - i. **Foreground process** —The current process visible on the screen, representing the application the user is currently interacting with
  - ii. **Visible process** —A process that is not directly visible on the foreground but that is performing an activity that the foreground process is referring to

OPERATIONS ON PROCESSES

2. Process Termination : Android Process Hierarchy

- iii. **Service process** —A process that is similar to a background process but is performing an activity that is apparent to the user (such as streaming music)

- iv. **Background process** —A process that may be performing an activity but is not apparent to the user.

- v. **Empty process** —A process that holds no active components associated with any application

THREADS

- The process model discussed so far has implied that a process is a program that performs a single thread of execution
  - For example, when a process is running a word-processor program, a single thread of instructions is being executed
  - This single thread of control allows the process to perform only one task at a time.
  - Thus, the user cannot simultaneously type in characters and run the spell checker

Threads

- The process model discussed so far has implied that a process is a program that performs a single thread of execution
  - For example, when a process is running a word-processor program, a single thread of instructions is being executed
  - This single thread of control allows the process to perform only one task at a time.
  - Thus, the user cannot simultaneously type in characters and run the spell checker

Threads

- 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.

- This feature is especially beneficial on multicore systems, where multiple threads can run in parallel.

- A multithreaded word processor could, for example, assign one thread to manage user input while another thread runs the spell checker.

THREADS

- On systems that support threads, the PCB is expanded to include information for each thread.
- Other changes throughout the system are also needed to support threads.
- We'll look at this later.

THREADS

- On systems that support threads, the PCB is expanded to include information for each thread.
- Other changes throughout the system are also needed to support threads.
- We'll look at this later.

INTER PROCESS COMMUNICATION.

- Processes executing concurrently in the operating system may be either independent processes or cooperating processes.
  - A process is independent if it does not share data with any other processes executing in the system
  - A process is cooperating if it can affect or be affected by the other processes executing in the system.
    - Clearly, any process that shares data with other processes is a cooperating process

INTER PROCESS COMMUNICATION.

- There are several reasons for providing an environment that allows process cooperation:
  - **1. Information sharing:** Since several applications may be interested in the same piece of information (e.g. copying and pasting), we must provide an environment to allow concurrent access to such information

  - **2. Computation speedup :** If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others.

INTER PROCESS COMMUNICATION.

- 3. **Modularity** : We may want to construct the system in a modular fashion, dividing the system functions into separate processes or threads.

- Cooperating processes require an interprocess communication (IPC) mechanism that will allow them to exchange data— that is, send data to and receive data from each other.

- There are two fundamental models of interprocess communication: **shared memory** and **message passing**

INTER PROCESS COMMUNICATION: Shared-memory

- In the shared-memory model, a region of memory that is shared by the cooperating processes is established.
- Processes can then exchange information by reading and writing data to the shared region.

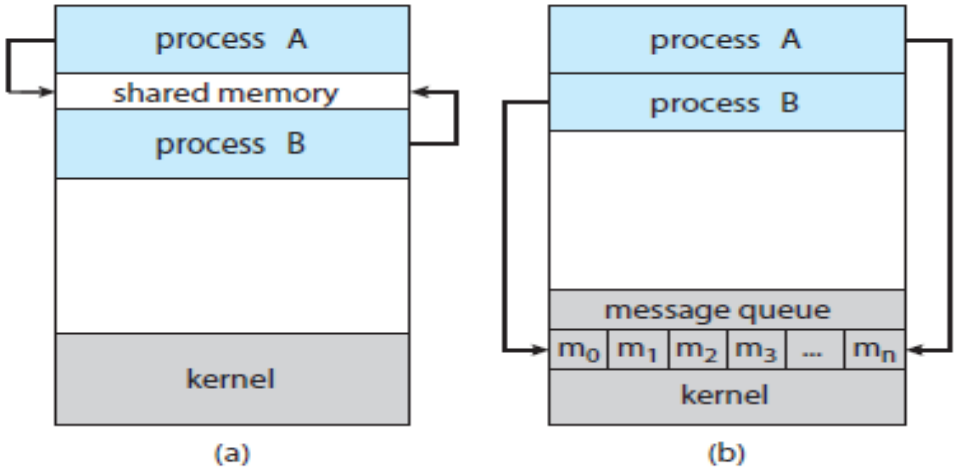


Figure 3.11 Communications models. (a) Shared memory. (b) Message passing.

INTER PROCESS COMMUNICATION: Shared-memory

- Message passing is useful for exchanging smaller amounts of data, because no conflicts need be avoided.
- Message passing is also easier to implement in a distributed system than shared memory.

- Shared memory can be faster than message passing, since message-passing systems are typically implemented using system calls and thus require the more time-consuming task of kernel intervention.

INTER PROCESS COMMUNICATION: Message-parsing

- In the message-passing model, communication takes place by means of messages exchanged between the cooperating processes

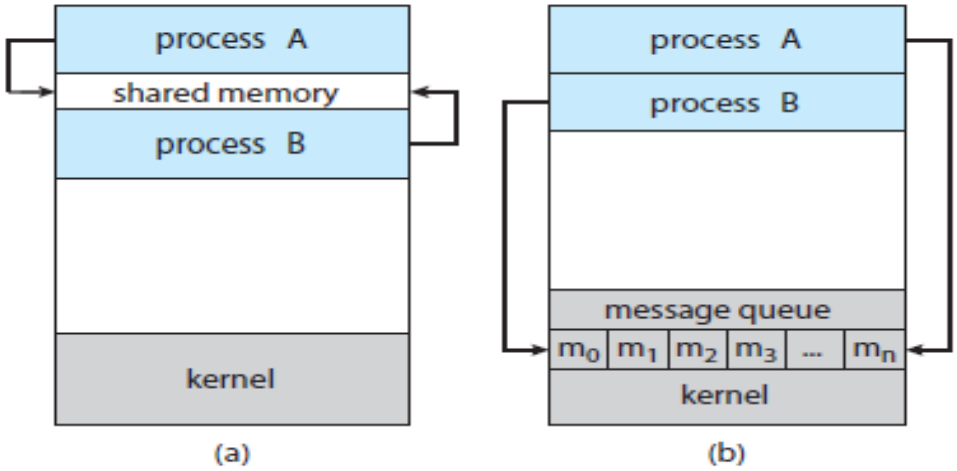


Figure 3.11 Communications models. (a) Shared memory. (b) Message passing.

PROCESS SCHEDULING

- The objective of multiprogramming is to have some process running at all times so as to maximize CPU utilization.

- The objective of time sharing is to switch a CPU core 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 a core.

PROCESS SCHEDULING

- Each CPU core can run one process at a time.

- For a system with a single CPU core, there will never be more than one process running at a time, whereas a multicore system can run multiple processes at one time.

- If there are more processes than cores, excess processes will have to wait until a core is free and can be rescheduled.
- The number of processes currently in memory is known as the degree of multiprogramming.

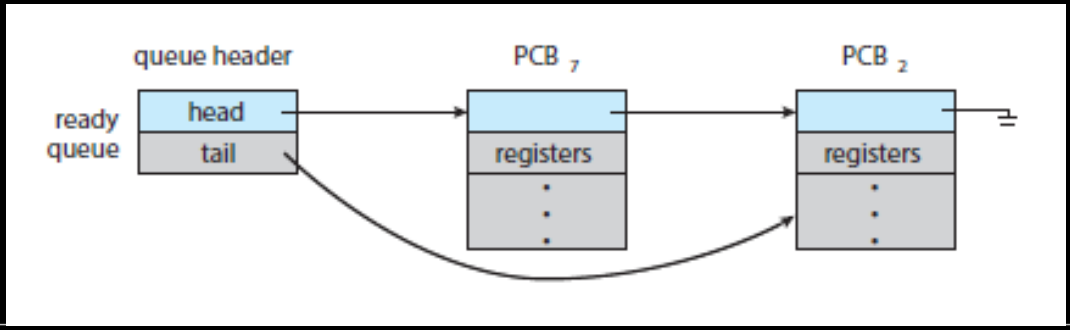
PROCESS SCHEDULING

- Balancing the objectives of multiprogramming and time sharing also requires taking the general behavior of a process into account.

- 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.

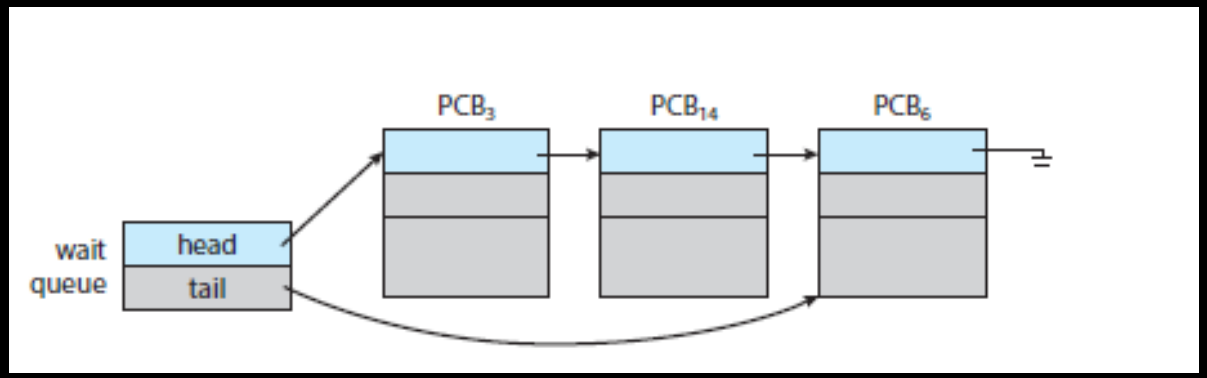
PROCESS SCHEDULING: Scheduling Queues

- As processes enter the system, they are put into a ready queue, where they are ready and waiting to execute on a CPU's core
  - This queue is generally stored as a linked list;
  - a ready-queue header contains pointers to the first PCB in the list, and each PCB includes a pointer field that points to the next PCB in the ready queue, as shown.



PROCESS SCHEDULING: Scheduling Queues

- Suppose the process makes an I/O request to a device such as a disk.
- Since devices run significantly slower than processors, the process will have to wait for the I/O to become available.
- Processes that are waiting for a certain event to occur — such as completion of I/O — are placed in a wait queue



PROCESS SCHEDULING: Scheduling Queues

- A common representation of process scheduling is a queueing diagram, as shown below

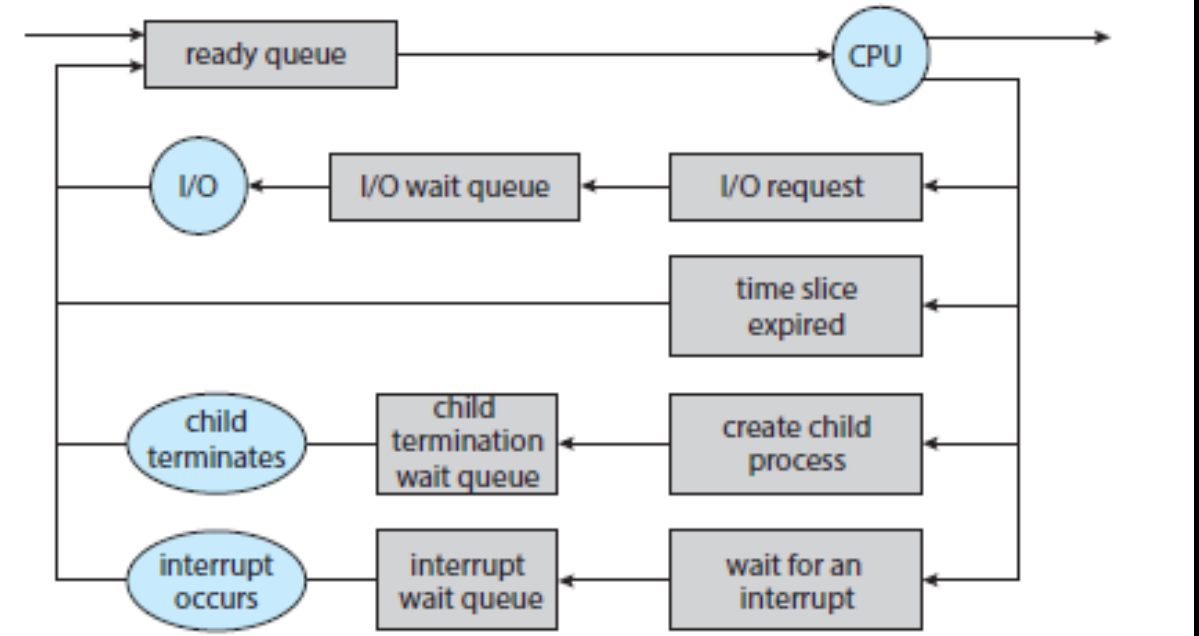


Figure 3.5 Queueing-diagram representation of process scheduling.

PROCESS SCHEDULING: Scheduling Queues

- In a queueing diagram:
  - Two types of queues are present: the ready queue and a set of wait 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.

PROCESS SCHEDULING: Scheduling Queues

- In a queueing diagram:
  - Once the process is allocated a CPU core 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 wait queue.
    - The process could create a new child process & then be placed in a wait queue while it awaits the child's termination.
    - The process could be removed forcibly from the core, as a result of an interrupt or having its time slice expire, and be put back in the ready queue.

PROCESS SCHEDULING: CPU Scheduling

- A process migrates among the ready queue and various wait queues throughout its lifetime.

- The role of the CPU scheduler is to select from among the processes that are in the ready queue and allocate a CPU core to one of them.

- The CPU scheduler must select a new process for the CPU frequently.

PROCESS SCHEDULING: CPU Scheduling

- An I/O-bound process may execute for only a few milliseconds before waiting for an I/O request.

- Although a CPU-bound process will require a CPU core for longer durations, the scheduler is unlikely to grant the core to a process for an extended period.

- Instead, it is likely designed to forcibly remove the CPU from a process and schedule another process to run.

PROCESS SCHEDULING: CPU Scheduling

- Therefore, the CPU scheduler executes at least once every 100 milliseconds, although typically much more frequently.

- Some operating systems have an intermediate form of scheduling, known as **swapping**:
  - The key idea of swapping is that sometimes it can be advantageous to remove a process from memory (and from active contention for the CPU) and thus reduce the degree of multiprogramming

PROCESS SCHEDULING: Context Switch

- Interrupts cause the operating system to change a CPU core from its current task and to run a kernel routine.

- When an interrupt occurs, the system needs to save the current context of the process running on the CPU core so that it can restore that context when its processing is done, essentially suspending the process and then resuming it

PROCESS SCHEDULING: Context Switch

- Generically, we perform a state save of the current state of the CPU core, be it in kernel or user mode, and then a state restore to resume operations.

- Switching the CPU core to another process requires performing a state save of the current process and a state restore of a different process.
  - This task is known as a context switch, and it is shown in the next diagram

PROCESS SCHEDULING: Context Switch

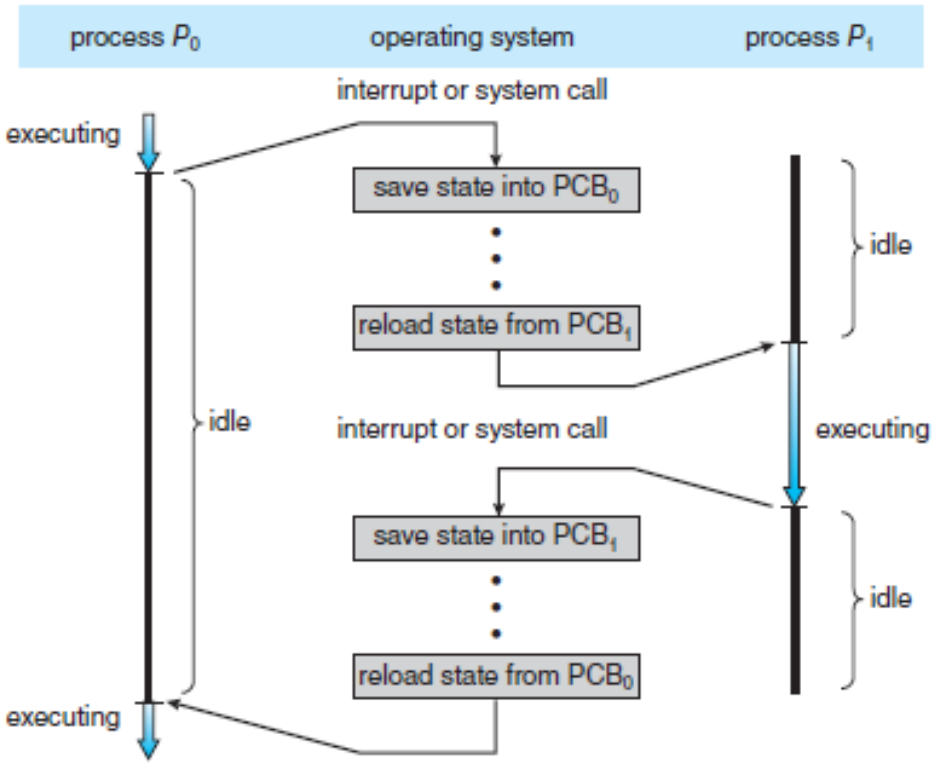


Figure 3.6 Diagram showing context switch from process to process.

PROCESS SCHEDULING: Context Switch

- When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.

- Context switch time is pure overhead, because the system does no useful work while switching.

- Switching speed varies from machine to machine depending on the memory speed, the number of registers that must be copied, and the existence of special instructions

SCHEDULING ALGORITHMS

1. First-Come, First-Served(FSFS) Scheduling Algorithm

- By far the simplest CPU-scheduling algorithm
- With this scheme, the process that requests the CPU first is allocated the CPU first.

- The implementation of the FCFS policy is easily managed with a FIFO queue
  - When a process enters the ready queue, its PCB is linked onto the tail of the queue.
  - When the CPU is free, it is allocated to the process at the head of the queue.

SCHEDULING ALGORITHMS

1. First-Come, First-Served(FSFS) Scheduling Algorithm

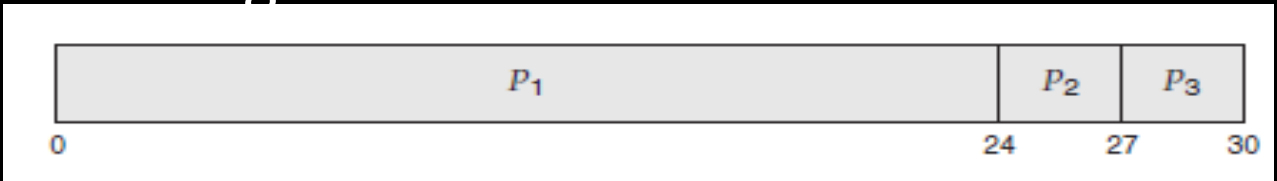
- Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst Time
P_1	24
P_2	3
P_3	3

SCHEDULING ALGORITHMS

1. First-Come, First-Served(FSFS) Scheduling Algorithm

- If the processes arrive in the order P_1, P_2, P_3, and are served in FCFS order, we get the result shown in the following Gantt chart



- The waiting time is 0 milliseconds for process P_1, 24 milliseconds for process P_2, and 27 milliseconds for process P_3.
- Thus, the average waiting time is (0 + 24 + 27)/3 = 17 milliseconds

SCHEDULING ALGORITHMS

1. First-Come, First-Served(FSFS) Scheduling Algorithm

- If the processes arrive in the order P_2, P_3, P_1, however, the results will be as shown in the following Gantt chart:

- The waiting time is 0 milliseconds for process P_2, 3 milliseconds for process P_3, and 6 milliseconds for process P_1
- The average waiting time is now (6 + 0 + 3)/3 = 3 milliseconds.
- This reduction is substantial.

SCHEDULING ALGORITHMS

1. First-Come, First-Served(FSFS) Scheduling Algorithm

- There is a convoy effect as all the other processes wait for the one big process to get off the CPU.

- This effect results in lower CPU and device utilization than might be possible if the shorter processes were allowed to go first.

SCHEDULING ALGORITHMS

2. Shortest-Job-First (SJF) Scheduling Algorithm

- This algorithm associates with each process the length of the process's next CPU burst.

- When the CPU is available, it is assigned to the process that has the smallest next CPU burst.

- If the next CPU bursts of two processes are the same, FCFS scheduling is used to break the tie.

SCHEDULING ALGORITHMS

2. Shortest-Job-First (SJF) Scheduling Algorithm

- a more appropriate term for this scheduling method would be the shortest-next-CPU-burst algorithm, because scheduling depends on the length of the next CPU burst of a process, rather than its total length

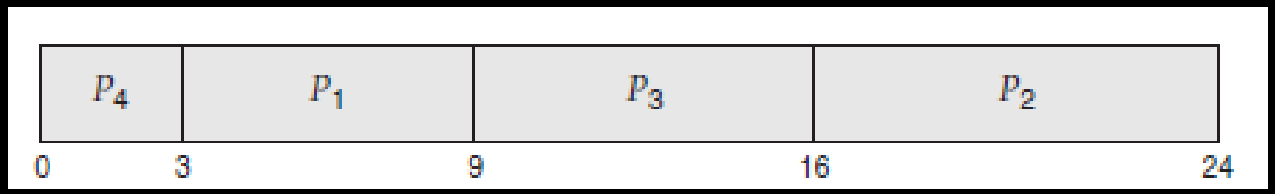
- consider the following set of processes, with the length of the CPU burst given in milliseconds:

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

SCHEDULING ALGORITHMS

3. Shortest-Job-First (SJF) Scheduling Algorithm

- Using SJF scheduling, we would schedule these processes according to the following Gantt chart:



- The waiting time is 3 milliseconds for process P_1, 16 milliseconds for process P_2, 9 milliseconds for process P_3 and 0 milliseconds for process P_4.
- Thus, the average waiting time is (3 + 16 + 9 + 0)/4 = 7 milliseconds.

SCHEDULING ALGORITHMS

3. Round-Robin (RR) Scheduling Algorithm

- similar to FCFS scheduling, but preemption is added to enable the system to switch between processes
- A small unit of time, called a time quantum or time slice, is defined.
    - A time quantum is generally from 10 to 100 milliseconds in length.

- The ready queue is treated as a circular queue.

SCHEDULING ALGORITHMS

3. Round-Robin (RR) Scheduling Algorithm

- The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum, and dispatches the process.

- Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

Process	Burst Time
P_1	24
P_2	3
P_3	3

SCHEDULING ALGORITHMS

3. Round-Robin (RR) Scheduling Algorithm

- If we use a time quantum of 4 milliseconds, then process P_1 gets the first 4 milliseconds.

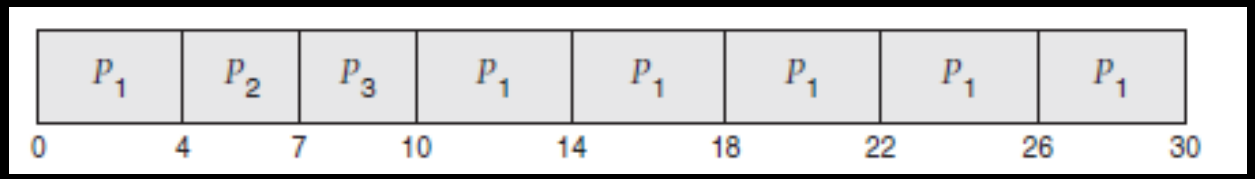
- Since it requires another 20 milliseconds, it is preempted after the first time quantum, and the CPU is given to the next process in the queue, process P_2.

- Process P_2 does not need 4 milliseconds, so it quits before its time quantum expires

SCHEDULING ALGORITHMS

3. Round-Robin (RR) Scheduling Algorithm

- The CPU is then given to the next process, process P_3
- Once each process has received 1 time quantum, the CPU is returned to process P_1 for an additional time quantum
- The resulting RR schedule is as follows:



- average waiting time is 17/3 = 5.66 milliseconds.

SCHEDULING ALGORITHMS

4. Priority Scheduling Algorithm

- The SJF algorithm is a special case of the general priority-scheduling algorithm.
- A priority is associated with each process, and the CPU is allocated to the process with the highest priority.

- Equal-priority processes are scheduled in FCFS order.

- An SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst.
  - the larger the CPU burst, the lower the priority

SCHEDULING ALGORITHMS

4. Priority Scheduling Algorithm

- Consider the following set of processes, assumed to have arrived at time 0 in the order P_1, P_2, · · ·, P_5, with the length of the CPU burst given in milliseconds:

Process	Burst Time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

SCHEDULING ALGORITHMS

Priority Scheduling Algorithm

- Using priority scheduling, we would schedule these processes according to the following Gantt chart:

- The average waiting time is 8.2 milliseconds.
- Other scheduling algorithms
  - Multilevel Queue
  - Multilevel Feedback Queue

MULTIPLE -PROCESSOR SCHEDULING.

- **Read and make notes on this**

THREAD SCHEDULING.

- **Read and make notes on this**

THE END