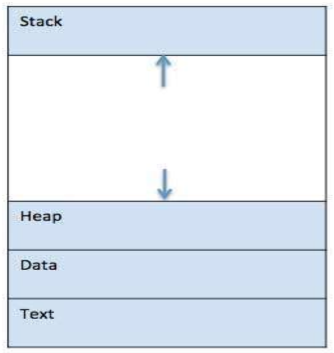
CHAPTER 2

PROCESS MANAGEMENT

A process is a program in execution. A process is more than the program code, which is sometimes known as the text section. It also includes the current activity, as represented by the value of the program counter and the contents of the processor's registers. In addition, a process generally includes the process stack, which contains temporary data (such as method parameters, return addresses, and local variables), and a data section, which contains global variables. We emphasize that a program by itself is not a process; a program is a passive entity, such as the contents of a file stored on disk, whereas a process is an active entity, with a program counter specifying the next instruction to execute and a set of associated resources.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The following image shows a simplified layout of a process inside main memory

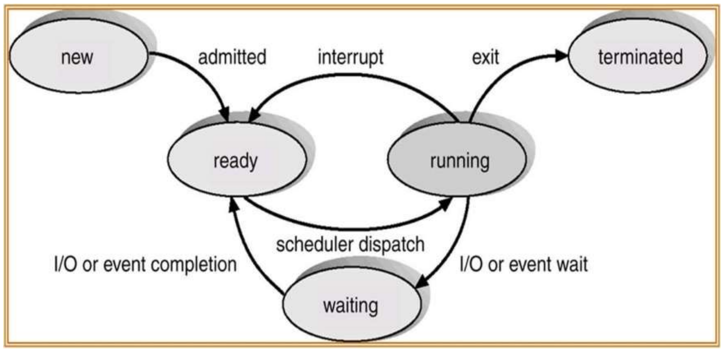


- **Stack**-The process Stack contains the temporary data such as method/function parameters, return address and local variables.
- **Heap-**This is dynamically allocated memory to a process during its run time.
- **Text-**This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
- **Data-**This section contains the global and static variables.

PROCESS STATES

For a program to be executed, a process, or task, is created for that program. From the processor's point of view, it executes instructions from its repertoire in some sequence dictated by the changing values in the program counter register. Over time, the program counter may refer to code in different programs that are part of different processes. From the point of view of an individual program, its execution involves a sequence of instructions within that program.

Some processes in the Not Running state are ready to execute, while others are blocked, waiting for an I/O operation to complete. Thus, using a single queue, the dispatcher could not just select the process at the oldest end of the queue. Rather, the dispatcher would have to scan the list looking for the process that is not blocked and that has been in the queue the longest. A more natural way to handle this situation is to split the Not Running state into two states: Ready and Blocked. We have added two additional states that will prove useful.



The five states in this new diagram are:

- **Running:** The process that is currently being executed. For this chapter, we will assume a computer with a single processor, so at most one process at a time can be in this state.
- **Ready:** A process that is prepared to execute when given the opportunity.
- **Blocked/Waiting:** A process that cannot execute until some event occurs, such as the completion of an I/O operation.
- **New:** A process that has just been created but has not yet been admitted to the pool of executable processes by the OS. Typically, a new process has not yet been loaded into main memory, although its process control block has been created.
- **Terminate:** A process that has ben released from the pool of executable processes by the OS, either because it halted or because it aborted for some reason.

Above figure indicates the types of events that lead to each state transition for a process; the possible transitions are as follows:

- **Null - New:** A new process is created to execute a program.
- **New - Ready:** The OS will move a process from the new state to the Ready state when it is prepared to take on an additional process. Most systems set some limit

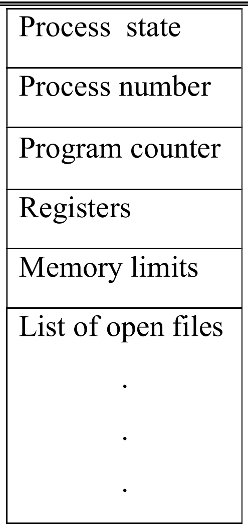
based on the number of existing processes or the amount of virtual memory committed to existing processes. This limit assures that there are not so many active processes as to degrade performance.

- **Ready - Running:** When it is time to select a process to run, the OS chooses one of the processes in the Ready state. This is the job of the scheduler or dispatcher.
- **Running - Exit:** The OS terminate the currently running process if the process indicates that it has completed, or if it aborts.
- **Running - Ready:** The most common reason for this transition is that the running process has reached the maximum allowable time for uninterrupted execution; virtually all multiprogramming operating systems impose this type of time discipline. There are several other alternative causes for this transition, which are not implemented in all operating systems. Of particular importance is the case in which the OS assigns different levels of priority to different processes.
- **Running - Blocked:** A process is put in the Blocked state if it requests something for which it must wait. A request to the OS is usually in the form of a system service call;

  that is, a call from the running program to a procedure that is part of the operating system code.
- **Blocked - Ready:** A process in the Blocked state is moved to the Ready state when the event for which it has been waiting occurs.
- **Ready - Exit:** For clarity, this transition is not shown on the state diagram. In some systems, a parent may terminate a child' process at any time. Also, if a parent terminates, all child processes associated with that parent may be terminated.
- **Blocked - Exit:** The comments under the preceding item apply.

PROCESS CONTROL BLOCK (PCB)

Two essential elements of a process are program code (which may be shared with other processes that are executing the same program) and a set of data associated with that code. Let us suppose that the processor begins to execute this program code, and we refer to this executing entity as a process. At any given point in time, while the program is executing, this process can be uniquely characterized by a number of elements, including the following:



- **Process State:** State may enter into new, ready, running, waiting, halted.
- **Process Number (PID):** A unique identification number for each process in the operating system (also known as Process ID).
- **Program Counter (PC):** A pointer to the address of the next instruction to be executed for this process.
- **CPU Registers:** Indicates various register set of CPU where process need to be stored for execution for running state.
- **CPU Scheduling Information:** Indicates the information of a process like process priority, pointers to scheduling queue and any other scheduling parameters.
- **Memory Limit:** It specifies the memory limits
- **List of Open File:** Provides information about files that are open.

The information in the preceding list is stored in a data structure, typically called a process control block that is created and managed by the OS. The significant point about the process control block is that it contains sufficient information so that it is possible to interrupt a running process and later resume execution as if the interruption had not occurred. The process control block is the key tool that enables the OS to support multiple processes and to provide for multiprocessing. When a process is interrupted, the current values of the program counter and the processor registers (context data) are saved in the appropriate fields of the corresponding process control block, and the state of the process is changed to some other value, such as blocked or ready (described subsequently). The OS is now free to put some other process in the running state. The program counter and context data for this process are loaded into the processor registers and this process now begins to execute. Thus, we can say that a process consists of program code and associated data plus a process control block. For a single-processor computer, at any given time, at most one process is executing and that process is in the running state.

PROCESS SHEDULING

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues −

- **Job queue** − This queue keeps all the processes in the system.
- **Ready queue** − This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** − The processes which are blocked due to unavailability of an I/O device constitute this queue.

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 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. We extend each PCB to include a pointer field that points to the next PCB in the ready queue.

The operating system also has 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. In the case of an I/O request, such a request may be to a dedicated tape drive, or to a shared device, such as a disk. Since the system has many processes, 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.

A common representation of process scheduling is a queuing diagram. 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.

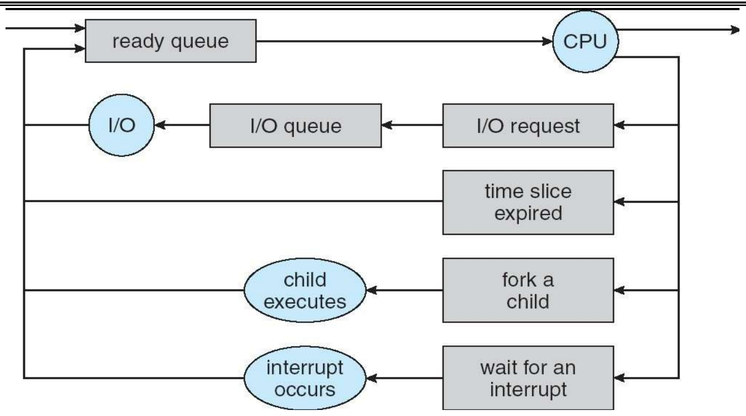


Figure: Queuing-diagram representation of process scheduling

A new process is initially put in the ready queue. It waits in the ready queue until it is selected for execution (or dispatched). Once the process is assigned to 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 sub process and wait for its 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.

TYPES OF SCHEDULERS

Schedulers are special system software which handles process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types −

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler

It is also called a job scheduler. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling. The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system. On some systems, the long-term scheduler may not be available or minimal. Timesharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler

It is also called as CPU scheduler. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them. Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

Medium-term scheduling is a part of swapping. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes. A running process may become suspended if it makes an I/O request. A suspended process cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called swapping, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

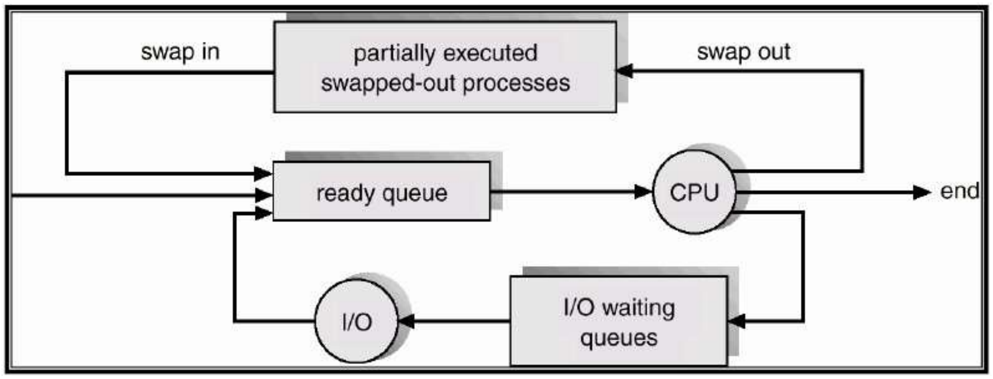


Figure: Addition of medium term scheduling to the queuing diagram

Comparison among Scheduler

Long-Term Scheduler	**Short-Term Scheduler**	**Medium-Term Scheduler**
It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

CO-OPERATING PROCESSES

The concurrent processes executing in the operating system may be either independent processes or cooperating processes. A process is independent if it cannot affect or be affected by the other processes executing in the system. Clearly, any process that does not share any data (temporary or persistent) with any other process is independent. On the other hand, 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. We may want to provide an environment that allows process cooperation for several reasons:

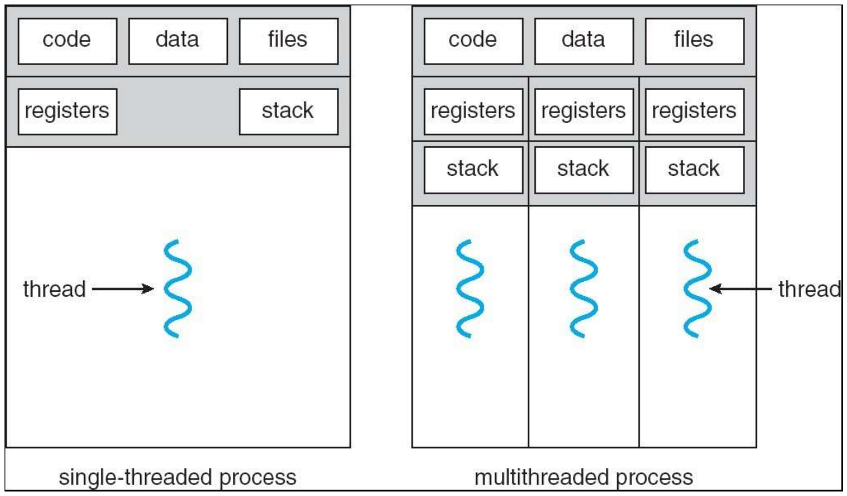
- **Information sharing:** Since several users may be interested in the same piece of information. We must provide an environment to allow concurrent access to these types of resources.
- **Computation speedup:** If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others. Such a speedup can be achieved only if the computer has multiple processing elements.

- **Modularity:** We may want to construct the system in a modular fashion, dividing the system functions into separate processes or threads.
- **Convenience:** Even an individual user may have many tasks on which to work at one time. For instance, a user may be editing, printing, and compiling in parallel.

Concurrent execution of cooperating processes requires mechanisms that allow processes to communicate with one another and to synchronize their actions.

THREADS

A thread, sometimes called a lightweight process (LWP), is a basic unit of CPU utilization; A traditional (or heavyweight) process has a single thread of control. If the process has multiple threads of control, it can do more than one task at a time. A web browser might have one thread display images or text while another thread retrieves data from the network. A word processor may have a thread for displaying graphics, another thread for reading keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.



SINGLE AND MULTIPLE THREADS BENEFITS

- **Responsiveness:** Multithreading an interactive application may allow a program to continue running even if part of it is blocked or is performing a lengthy operation thereby increasing responsiveness to the user.

- **Resource sharing:** by default threads share the memory and the resources of the process to which they belong, The benefit of code sharing is that it allows an application to have several different of activities all within the same address space.
- **Economy:** Allocating memory and resources for process creation is costly alternatively, because threads share resources of the process to which they belong it is more economical to create and context switch threads.
- **Utilization of multiprocessor architectures:** the benefits of multithreading can be greatly increased in a multithreading architecture, where each thread may be running in parallel on a processor.

Comparison between Process and Thread:

	Process	**Thread**
Definition	An executing instance of a program is called a process.	A thread is a subset of the process.
Process	It has its own copy of the data segment of the parent process.	It has direct access to the data segment of its process.
Communication	Processes must use inter-process communication to communicate with sibling processes.	Threads can directly communicate with other threads of its process.
Overheads	Processes have considerable overhead.	Threads have almost no overhead.
Creation	New processes require duplication of the parent process.	New threads are easily created.
Control	Processes can only exercise control over child processes.	Threads can exercise considerable control over threads of the same process.
Changes	Any change in the parent process does not affect child processes.	Any change in the main thread may affect the behaviour of the other threads of the process.
Memory	Run in separate memory spaces.	Run in shared memory spaces.
File descriptors	Most file descriptors are not shared.	It shares file descriptors.

File system	There is no sharing of file system context.	It shares file system context.
Signal	It does not share signal handling.	It shares signal handling.
Controlled by	Process is controlled by the operating system.	Threads are controlled by programmer in a program.
Dependence	Processes are independent.	Threads are dependent.

Comparison between Single Thread and Multi Thread:

Single Thread	Multi Thread
Single threaded processes contain the execution of instructions in a single sequence. In other words, one command is processes at a time.	These processes allow the execution of multiple parts of a program at the same time. These are lightweight processes available within the process.
Refers to executing an entire process from beginning to end without interruption by a thread.	Refers to allowing multiple threads within a process such that they execute independently but share their resources
When one thread is paused, the system waits until this thread is resumed.	When one thread is paused due to some reason, other threads run as normal.
It results in less efficient programs.	It results in more efficient programs
Idle time is more.	Idle time is minimum.
CPU time is wasted.	CPU time is never wasted.