Process Management

Process –

- A process can be thought of as a program in execution. Process is the unit of program in most systems.
- A process is unit of work in most system. Informally, 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 program counter and the contents of the registers
- In addition, a process generally includes the process stack, which contains temporary data (Such as method parameters, return address, and local variables), and a data section, which contain global variable.

Process states –

- As a process executes it changes state. The state of a process is defined in part by the current activity of that process.

Each 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.

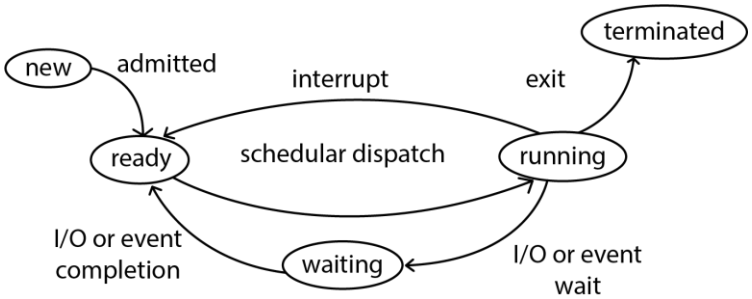


Fig. Diagram of process state

- The states names are arbitrary, and they vary across OS's. The states that they represent are found on all systems. However, only one process can be running on any processor at any instant, although many processes may be ready and waiting.

Process control block –

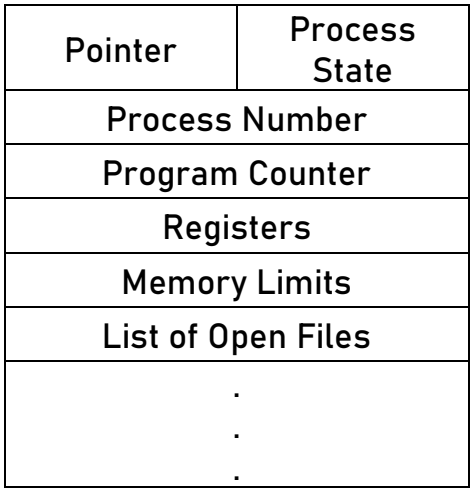


Fig. Process control block

- Each process is represented in the OS by a process control block (PCB) - also called a task control block.

It contains many pieces of information associated with a specific process, including there:

- Process state: The state may be new, ready, running, waiting, halted and so on.
- Program counter: The counter indicated the address of the next instruction to be executed for this process.
- CPU registers: The registers vary in numbers and types, depending on the computer architecture. They include accumulator, 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 afterwards.
- CPU scheduling information: This information includes a process priority pointer to scheduling queues and any other scheduling parameters.
- Memory management information: This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the OS.
- 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 states information: The information 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.

Process Scheduling –

- The objective of multi-programming is to have some process running at all time, so as to maximize CPU utilization.
- The objective of time scheduling is to switch the CPU among processes frequently that users can interact with each program while it is running.
- The process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process on the basis of a particular strategy.
- If more processes exist, the rest must wait unit the CPU is free and can be rescheduled.

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 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 tinted 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 list of process waiting for a particular I/O device is called a device queue. Each device has its own device queue.

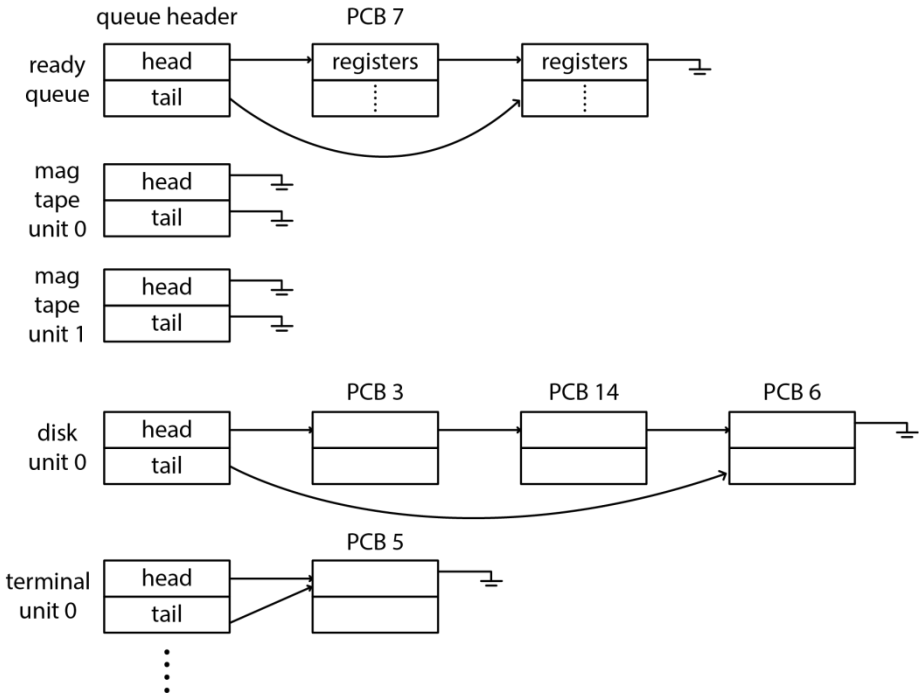


Fig. The ready queue and various I/O device queues

- A common representation of process scheduling is a queuing diagram such as that in figure below.
- Each rectangular box represents a queue.
- Two types of queues are present: the ready queue and a set of device queue
- 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 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 on I/O request, and then be placed in the I/O queue.
- The process could create a new subprocess 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.

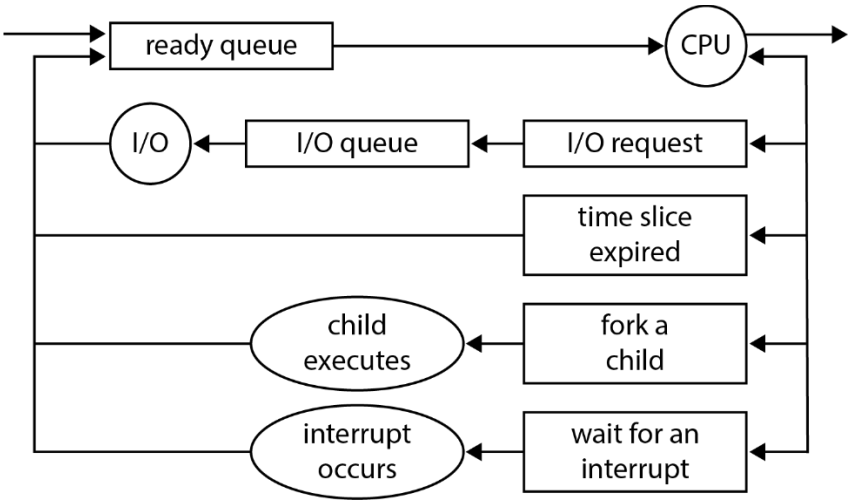


Fig. queueing – diagram representation of process scheduling

- 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 –

- Process scheduling handles the selection of a process for the processor on the basis of a scheduling algorithm and also the removal of a process from the processor.
- There are many scheduling queues that are used in process scheduling
- When the processes enter the system, they are put into the job queue. The processes that are ready to execute in the main memory are kept in the ready queue. The processes that are waiting for the I/O device are kept in the I/O device queue.

Long-term schedular –

- The job schedular or ling term scheduler selects processes from the storage pool in the secondary memory and loads them into the ready queue in the main memory for execution.
- The long-term scheduler controls the degree of multi-programming.
- It must select a careful mixture of I/O bound and CPU bound processes to yield optimum system throughput.

- If it selects too many CPU bound processes then the I/O devices are idle and if it selects too many I/O bound processes then the processor has nothing to do.
- The job of the long-term scheduler is very important and directly affects the system for a long time.

Short-term scheduler –

- The short-term scheduler selects one of the processes from the ready queue and schedules them for execution.
- A scheduling algorithm is used to decide which process will be scheduled for execution next.
- The short-term scheduler executes much more frequently than the long-term scheduler as a process may execute only for a few milliseconds.
- The choices of the short-term scheduler are very important. If it selects a process with a long burst time, then all the processes after that will have to wait for a long time in the ready queue.
- This is known as starvation and it may happen if a wrong decision is made by the short-term scheduler.

Medium-term scheduler –

- The medium-term scheduler swaps out a process from main memory. It can again swap in the process later from the point it stopped executing. This can also be called as suspending and resuming the process.
- This is helpful in reducing the degree of multiprogramming.
- The process is swapped out, and is later swapped in by the medium-term scheduler.

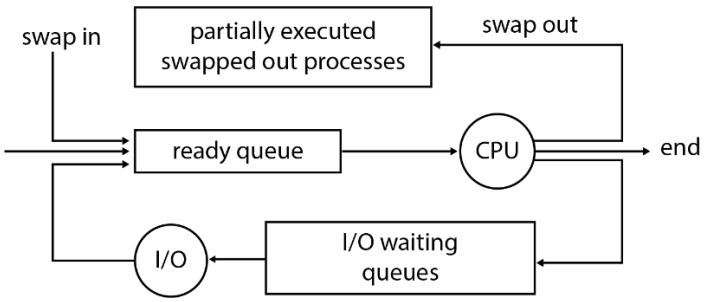


Fig. Addition of medium-term scheduling of the queuing diagram.

Short-Term scheduler	Long-Term scheduler	Medium-Term scheduler
Schedular which selects the jobs or processes which are ready to execute from the ready pool and allocated the CPU to one of them is called short-term scheduler or CPU scheduler.	The scheduler which picks up job from pool and loads into main memory for execution is called as long-term scheduler or job scheduler.	Medium-term scheduler removes the process from main memory and again loads afterwards when required.
Frequency of execution is high. May be in some milliseconds.	Frequency of execution is few minutes.	Frequency of execution is medium.

Short-Term scheduler work fast.	Long-Term scheduler is slow as compare to short term schedulers.	Medium-Term scheduler is called whenever required.
Deals with CPU.	Deals with main memory for loading process.	Deals with main memory for removing processes and reloading whenever required.

Context switch –

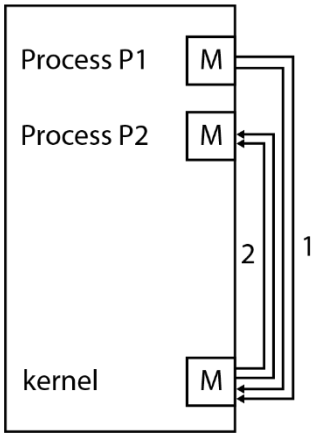
- Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as a context switch.
- The context of a process is represented in the PCB of a process, it includes the value of the CPU registers the process state and memory management information.
- 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 times are highly dependent on hardware support.
- To make context switching time to be less, registers which are the fastest access memory are used.

Inter process communication –

- Inter process communication provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space.
- Inter process is particularly useful in a distributed environment where the communicating processes many reside on different computers connected with a network. E.g., a chat program used on the world wide web.
- Inter process communication is best provided by a message passing system and message systems can be designed in many ways.

Message Passing System –

- The function of message passing system is to allow processes to communicate with one another without the need to resort to shared data.
- Communication among the user processes is accomplished through the passing of messages.
- An IPC facility provides at least the two operations send (message) and receive (message).
- Message sent by a process can be either fixed or variable size.
- If only fixed-size messages can be sent, the system-level implementation is straight forward. This restriction, however, makes the task of programming more difficult.
- On the other hand, variable-seized messages require a more complex system-level implementation, but the programming task becomes simpler.



Message Passing

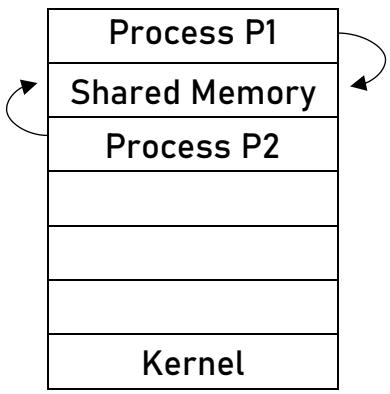
- If process P and Q want to communicate, they must send messages to and receive messages from each other, a communication link must exist between them.
- This link can be implemented in a variety of ways. We are concerned here not with the links physical implementation but rather with its logical implementation.

Here are several methods for logically implementing a link and send/receive operations:

- Direct or indirect communication
- Symmetric or asymmetric communication
- Automatic or explicit buffering
- Send by copy or send by reference
- Fixed-sized or variable sized message

Shared memory system –

- The process communication could involve a process letting another process know that some event has occurred or transferring of data from one process to another.
- One of the models from process communication is the shared memory model.
- The shared memory is the shared memory model is that memory that can be simultaneously occurred by multiple processes.
- This is done so that the processes can communicate with each other.



Shared memory model

In the above diagram, the shared memory can be accessed by process 1 and process 2.

Advantages of shared memory model –

Memory communication is faster on the shared memory model as compared to the message passing model on the same machine.

Disadvantages of shared memory model –

1. All the processes that use the shared memory model need to make sure that they are not writing to the same memory location.
2. Shared memory model may create problems such as synchronization and memory protection that need to be addressed.

Threads –

- A thread, sometimes called a lightweight process (LWP), is a basic unit of CPU utilization.
- It comprises a thread I/O, a program counter, a register set and a stack.
- It shares with other threads belonging to the same processes its code section, data section and other OS resources such as open files and signals.
- 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.

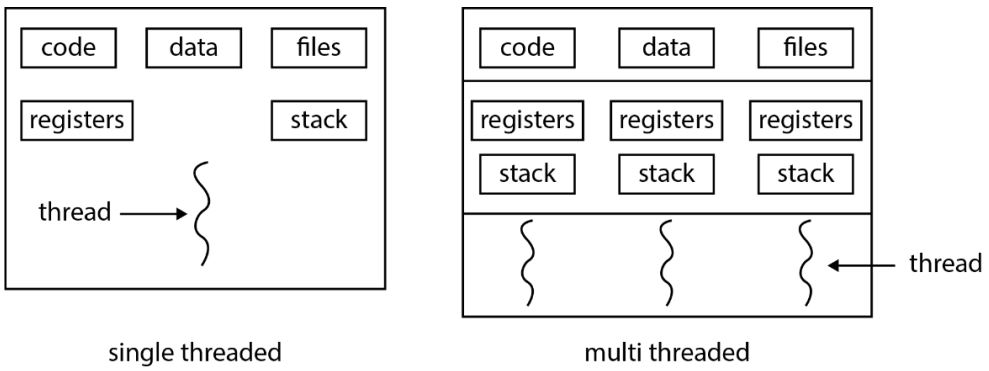


Fig. Single and Multithreaded processes

Multithreaded Benefits –

The benefits of multithreaded programming can be broken down into four major categories:

1. 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. For instance, a multithreaded web browser could still allow user interaction in one thread while an image is being loaded in another thread.

2. 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 is allows an application to have several different threads of activity all within the same address space.

3. 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.
   It can be difficult to gauge empirically the difference in overhead for creating and maintaining a process rather than a thread, but in general it is much more time consuming to create and manage processes than threads.
   Eg., In Solaris 2, creating a process is about 30 times slower than is creating a thread and context switching is about 5 times slower.

4. Utilization of multiprocessor architectures –

   The benefits of multithreading can be greatly increased in a multiprocessor architecture, where each thread may be running in parallel on a different processor. A single threaded process can only run on one GPU, no matter how many are available.
   Multithreading on a multi-CPU machine increases concurrency.
   In a single processor architecture, the CPU generally moves between each thread so quickly as to create an illusion of parallelism, but in reality, only one thread is running at a time.

   Types of threads –

   User thread –

   - User threads are supported above the kernel and are implemented by a thread library at the user level.
   - The library provides support for thread creation, scheduling and management with no support from the kernel.
   - Because the kernel is unaware of user-level threads, all thread creation and scheduling are done in user space without the need for kernel intervation.
   - Therefore, user-level threads are generally fast to create and manage, they have drawbacks, however.
   - For instance, if the kernel is single-threaded then any user-level thread performing a blocking system call will cause the entire process to block even if other threads are available to run within the application.

   Kernel thread –

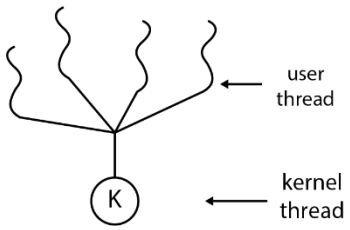
   - Kernel threads are supported directly by the OS. The kernel performs thread creation, scheduling and management in kernel space. Because thread management is done by the OS, kernel threads are generally slower to create and manage than are user threads. However, since the kernel is

managing the threads if a thread performs a blocking system call, the kernel can schedule another thread in the application for execution.
- Also in a multiprocessor environment, the kernel can schedule threads on different processors.

Multithreading models –

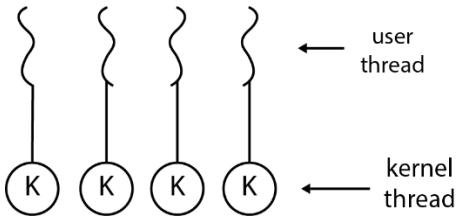
Many systems provide support for both user and kernel threads resulting in different multithreading models.

Many-to-One model –



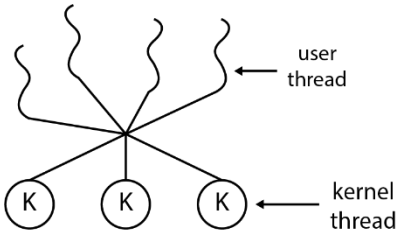
- The many-to-one model maps many user-level threads to one kernel model.
- Thread management is done in user space, so it is efficient, but the entire process will block if a thread makes a blocking system call.
- Also, because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.

One-to-One model –



- The one-to-one model maps each user thread to a kernel thread. It provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a blocking system call, it also allows multiple threads to run in parallel on multiprocessors.
- The only drawback to this model is that creating a user thread requires creating the corresponding kernel thread.
- Because the overhead of creating kernel thread can burden the performance of an application most implementations of this model restrict the number of threads supported by the system.
- Eg. Windows NT, Windows 2000 and OS / 2 implement the one-to-one model.

Many-to-Many model –



- The many-to-many model multiplexes many users level threads to a smaller or equal number of kernel threads.
- The number of kernel threads may be specific to either a particular application or a particular machine.
- Whereas the many-to-one model allows the developer to create as many user threads as she wishes, true concurrency is not gained because the kernel can schedule only one thread at a time.
- On a multiprocessor system many kernel threads can run parallel. When one thread call block system call, kernel thread calls for another system call.