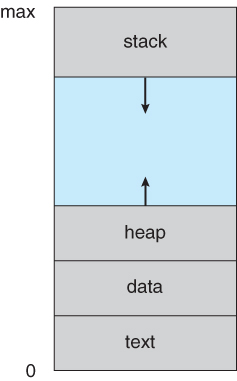
**Process Concept**

**Process: -** A process is an instance of a program in execution.(process or jobs are similar).

Program is passive entity while process is active entity.

**Process memory is divided into four sections:**

****

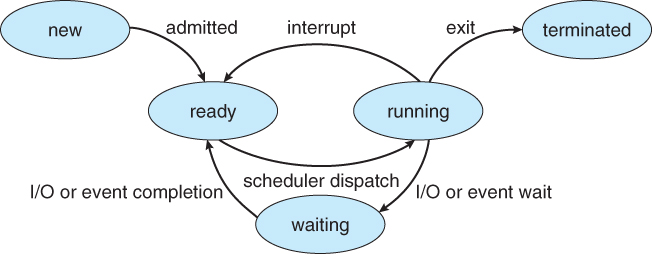
* The **text section** comprises the compiled program code, read in from non-volatile storage when the program is launched.
* The **data section** stores global and static variables, allocated and initialized prior to executing main.
* The **heap** is used for dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.
* The **stack** is used for local variables. Space on the stack is reserved for local variables when they are declared, and the space is freed up when the variables go out of scope.
* **Note** that the stack and the heap start at opposite ends of the process's free space and grow towards each other. If they should ever meet, then either a stack overflow error will occur, or else a call to new or malloc will fail due to insufficient memory available.

**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.

Processes may be in one of 5 states: -

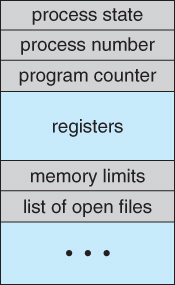


* **New** - The process is being created.
* **Ready** - The process has all the resources available that it needs to run, but it is waiting to be assigned to a processor.
* **Running** - Instructions of a process is being executed by CPU.
* **Waiting** - The process cannot run at the moment, because it is waiting for some resource to become available or for some event to occur.
* **Terminated** - The process has finished execution.

**Process Control Block**

Each process is represented in the operating system by a Process control block (PCB) also called a task control block.

For each process there is a Process Control Block, PCB, which stores the following ( types of ) process-specific information

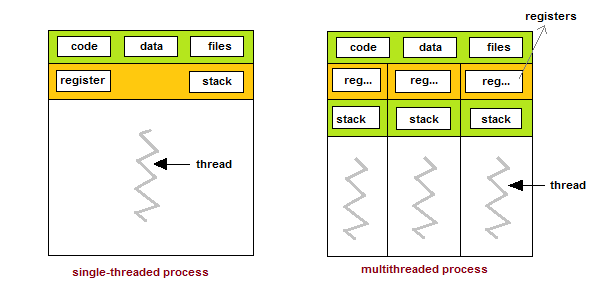


* **Process State** - The state may be new, ready, running, waiting, and so on.
* **Program counter.** The counter indicates the address of the next instruction to be executed for this process.
* **CPU registers.** The registers vary in number and type, depending on the computer architecture. They include 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.
* **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 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 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.

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

**Thread**

* Light weight process.
* It is Basic unit of CPU execution, which consisting of -
  + Program Counter,
  + thread ID,
  + Stack
  + Set of Registers.
* It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals.
* A traditional / heavyweight process has a single thread of control. If a process has multiple threads of control, it can preform more than one task at a time.



* **Benefits of Multithreaded programming can be broken down into four major categories:**
  + **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 sharing code and data is that it allows an application to have several different threads of activity within the same address space.(Efficiency)
  + **Economy**:- Allocating memory and resources for process creation is costly. Because threads share resources of the process to which they belong, it is more economical to create and context-switch threads.
  + **Utilization of multiprocessing architectures**:- The benefits of Multithreading can be greatly increased in a multiprocessor architecture, where threads may be running in parallel on different processors. A single-threaded process can only run on one CPU, no matter how many are available. Multithreading on a multi-CPU machine increases concurrency.
* Word document is process and spell checker, animation, saving are example of thread.

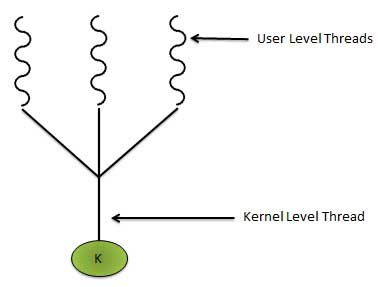
**Types of Threads:-**

* **User threads** :- Supported above the kernel and are managed without kernel support. Managed by user, example POSIX P thread
* **Kernel threads**:- Supported and managed directly by the operating system. Managed by kernel, example windows NT

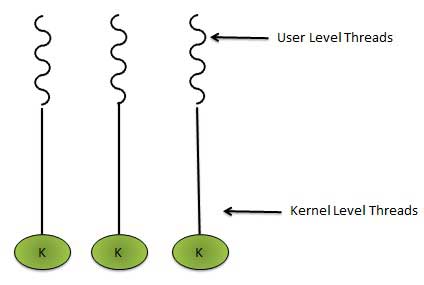
There must exist a relationship between user threads and kernel threads.

**There are three common ways of establishing this relationship:**

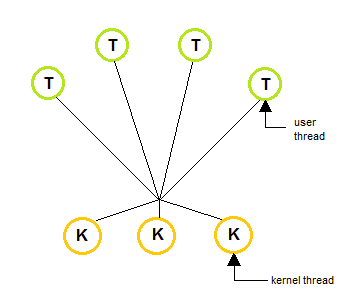
1. Many-to-one Model
2. One-to-One Model
3. Many-to-Many Model
4. Many-to-One Model:-
   1. Many user thread are associated one kernel thread.
   2. Thread management is done by the thread library in user space. So it is efficient.
   3. Disadvantages:-
      1. The entire process will block if a thread makes a blocking system call.
      2. Because only one thread can access the kernel at a time, multiple threads are unable to run in parallel on multiprocessors.



1. One-to-One Model:-
   1. Maps each user thread to one kernel thread.
   2. Provides more concurrency that the many-to-one model by allowing another thread to run when a thread makes a blocking system call.
   3. Also allows multiple threads to run in parallel on multiprocessors.
   4. Disadvantages:-
      1. Creating a user thread requires creating the corresponding kernel thread.
      2. Because the overhead of creating kernel threads can burden the performance of a application, most of the times when this model is implemented the number of kernel threads is restricted.



1. Many-to-Many Model:-
   1. Many user-level threads to a smaller or equal number of kernel threads.
   2. The number of kernel threads may be specific to either a particular application or a particular machine.
   3. Developers can create as many number of user threads as per requirement. And the corresponding kernel threads can run in parallel on a multiprocessor.
   4. Also, when a thread performs a blocking system call, the kernel can schedule another thread for execution.



**Similarities between process and Thread.**

* Only one process and thread is active at a time.
* Both execute sequentially.
* Both can create child.
* In both cases, if one thread is blocked, another thread can run.

**Difference between process and Thread.**

|  |  |
| --- | --- |
| **Process** | **Thread** |
| Heavy weight | Light weight |
| Process switching needs interaction with OS | Thread switching doesn’t need to interact with OS |
| Each process in multiprocessing environment has own memory & file resources. | All threads can share same set of open file, child processes |
| If one process is blocked, then no process will be executed until the first process get executes. | While one thread is blocked or waiting, another thread in the same task can run. |
| Each process operates independently | Not Independent |

**Process Switching: -** When a running process is interrupted and the operating system assigns another process to the running state and turns control over that process. This is called process switching.

**Process Scheduling:-**

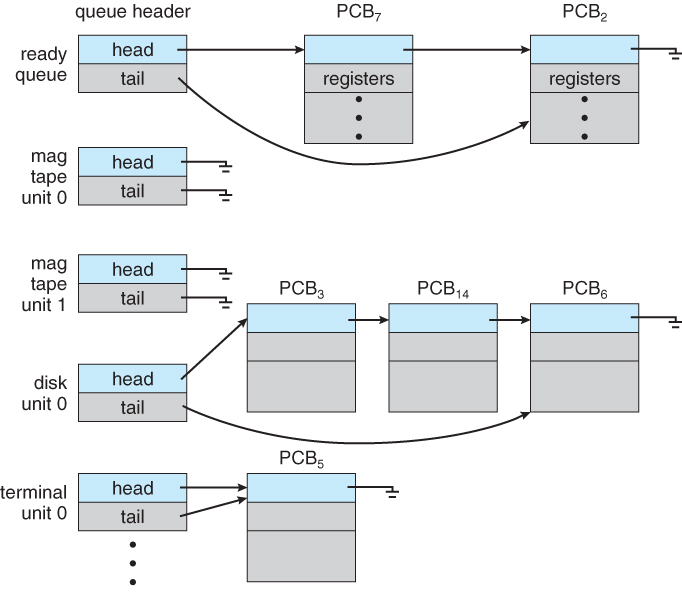
* It is a process in which process scheduler selects an available process from set of available processes for execution on the CPU.
* The two main objectives of the process scheduling system are to keep the CPU busy at all times and to deliver "acceptable" response times for all programs, particularly for interactive ones.
* 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 Queue:-**

There are two types of scheduling queue:-

1. Job Queue
2. Ready Queue

* **Job Queue:-** As processes enter the system, they are put into a job queue, which consists of all processes in the system.
* **Ready Queue:-** 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 stored as linked list.
  + Ready queue header contains the pointer to first PCB in the queue to final PCB in the queue.
  + Each PCB include a pointer field which points to the next PCB.



**Schedulers:-**

There are three types of process scheduler.

1. Long term or job scheduler
2. Short term or CPU scheduler
3. **Long term or Job Scheduler:-**
   1. It brings the new process to the ‘Ready State’.
   2. It controls Degree of **Multi-programming**, i.e., number of process present in ready state at any point of time.
   3. It is important that the long-term scheduler make a careful selection of both I/O and CPU bound process.
4. **Short term or CPU scheduler:-**
   1. It is responsible for selecting one process from ready state for scheduling it on the running state.
   2. Note: Short-term scheduler only selects the process to schedule it doesn’t load the process on running.
   3. Dispatcher is responsible for loading the process selected by Short-term scheduler on the CPU (Ready to Running State) Context switching is done by dispatcher only.
   4. A dispatcher does the following:
      1. Switching context.
      2. Switching to user mode.
      3. Jumping to the proper location in the newly loaded program.

**Context Switching:-**

A context switch occurs when the time slice for one process has expired and a new process is to be loaded from the ready queue. For this the current process's state need to be saved and the new process's state to be restored.