Threads

Thread

A process is divided into a number of light weight process, each light weight process is said to be a thread. The thread has a program counter that keeps the track of which instruction to execute next. It has a register which hold its current working variable, which contains the execution history. Number of threads can share the same address space.

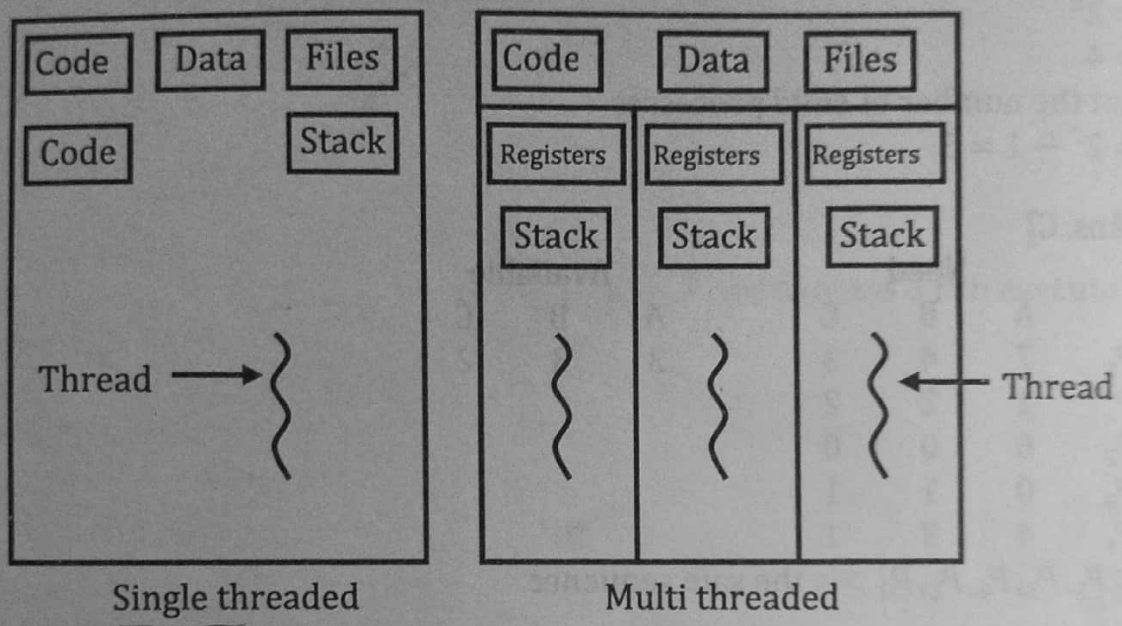


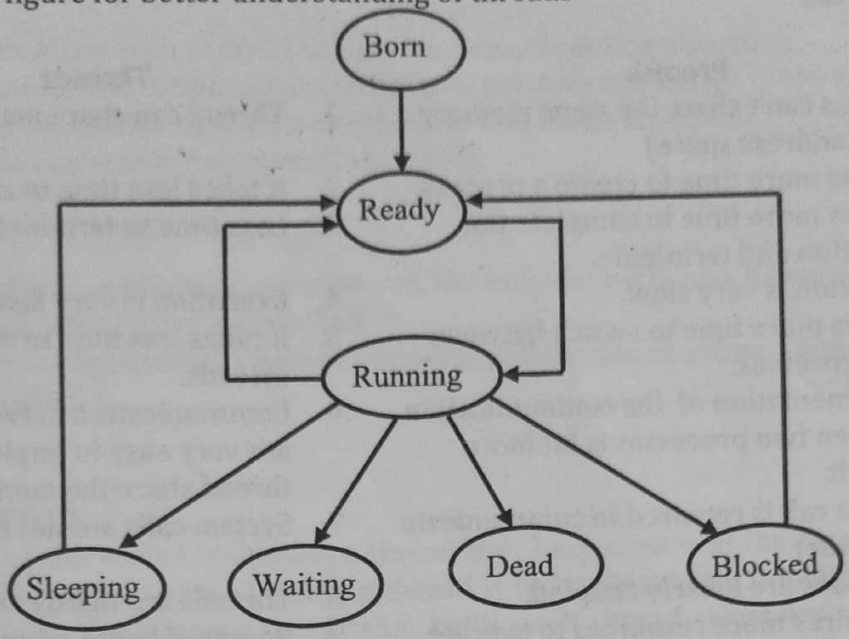
Fig. Single and Multi threaded processes

Life Cycle of Threads

A thread is said to be in one of at the following states.

1. Born State: A thread that has just created
2. Ready State: A thread is waiting for processor
3. Running: The system assigns the processor to the thread (i.e. the thread is being executing).
4. Blocked State: The thread is waiting for an event to occur or waiting for an I/O device.
5. Sleep: A sleeping thread becomes ready after the designated sleeping time expires.
6. Dead: The execution of the thread finished.

Consider the below figure for better understanding of threads



Life Cycle of a Thread

Examples of Threads

Word Processor: A programmer wishes to type a text in a word processor. Then the programmer open a file in the word processor and typing the text (it is one thread) the text is automatically formatting (it is another thread), the text is automatically specifies spelling mistake, it is another thread. Typing in a file is a process. But typing, formatting, spell checking, saving these are threads.

Multithreading

A process is divided into number of smaller tasks, each task is called threads. Number of thread with in a process execute at a time is called multithreading. Based on the functionality threads are divided into 4 categories. These are

1. **One Process and one thread:** In this approach the process maintain only one thread, so it is called signal thread approach. Examples MS DOS
2. **One Process and multiple threads:** In this a process is divided into a number of threads. Example java runtime environment
3. **Multiple threads and one process per thread:** An operating system support multiple user processes but only support one thread per process. Example UNIX operating system
4. **Multiple processes and multiple threads per process:** In which each process divided into number of threads. Example Linux, Windows 2000, Solaris.

Benefits of Thread

- A thread takes less time to terminate than process.
- It takes less time to switch between two threads with in the same process.
- It takes less time to be created within the processes
- Efficient communication between threads.

Process vs Thread

Process	Threads
1. Process can't share the same memory area (address space)	1. Thread can share memory and files.
2. It takes more time to create a process.	2. It takes less time to create a thread.
3. It takes more time to complete the execution and terminate.	3. Less time to terminate
4. Execution is very slow.	4. Execution is very fast.
5. It takes more time to switch between two processes.	5. It takes less time to switch between two threads.
6. Implementation of the communication between two processes is bit more difficult	6. Communication between two threads are very easy to implement. Because thread share the memory
7. System call is required to communicate each other	7. System calls are not required.
8. Processes are loosely coupled.	8. Threads are tightly coupled
9. It requires more resources to execute.	9. Required fewer resources, so it is also called lightweight process.
10. Processes are not suitable for parallel activities.	10. Suitable for parallel activities

Thread Synchronization

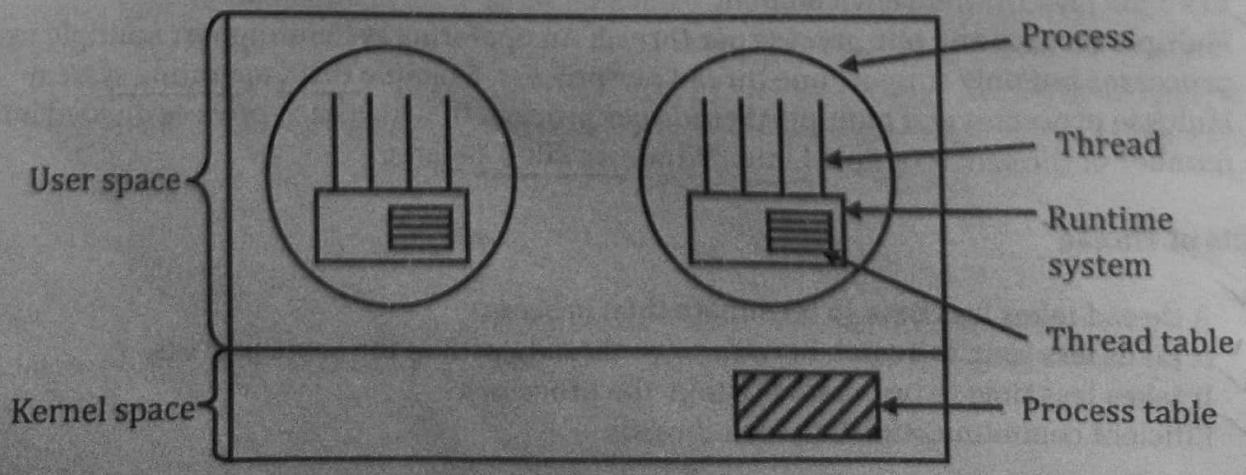
All threads within a process share the memory and files. One thread is trying to modify one record in a file, it will effect with another threads, so need to try synchronizing the activities of threads.

User Level and Kernel Level Threads

There are two main ways to implement threads: in user space and in the kernel.

User Level Thread

This type of threads loaded entirely in the user's space, the kernel knows nothing about them. Each process maintains its own private threads table. The thread table consist the information of program counter, stack pointer, registers, state etc. The thread table managed by run time system



Advantages

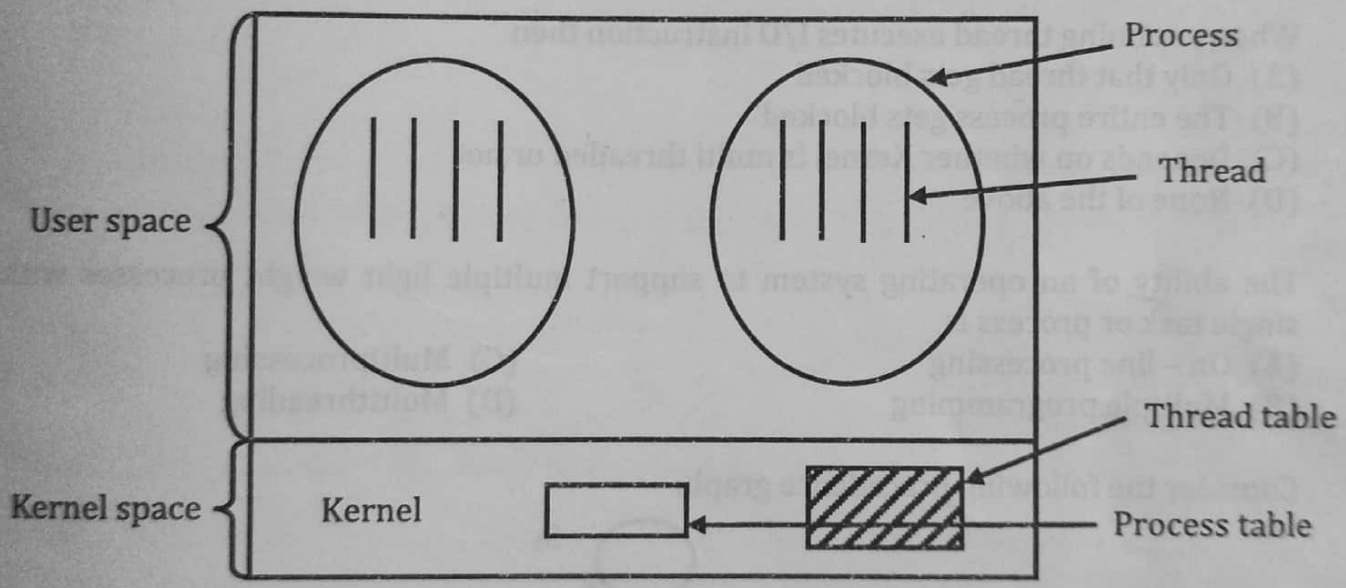
1. User level threads allow each process to have its own scheduling algorithm.
2. The entire process loaded in user space, so the process does not switch to the kernel mode to do thread management. This saves the overhead of two mode switches.
3. User level threads can run on any operating system.

Disadvantages

1. When user level threads executes a system call, not only the particular threads blocked all of the threads within the process are blocked
2. In user level, only a single threads within a process can execute at a time. So multiprocessing can't be implemented.

Kernel Level Thread

In kernel level thread the kernel maintains a thread that keeps track all the threads in system. When a new thread is to be create or existing thread is to be destroyed, it makes a kernel call, which takes the action. The kernel thread table holds each thread registers, state and other information



Advantages

1. The kernel can simultaneously schedule a multiple threads from the same process on multiple processors. It means it supports multiprocessing where as user level thread cannot support this
2. A thread in a process is blocked; the kernel can schedule another thread of the same process
3. Kernel routine themselves can be multi threaded.

Disadvantages

1. More cost for creating and destroying threads in the kernel.
2. The transfer of control from one thread to another with in the same process requires a mode switch to the kernel.