NAME: Sayed Muslim shah

ID#: 14856

MODULE: Bachelors {Software Engineering}

SEMESTER: summer 2020

SUBJECT: operating system

INSTRUCTOR: Sir Daud Khan

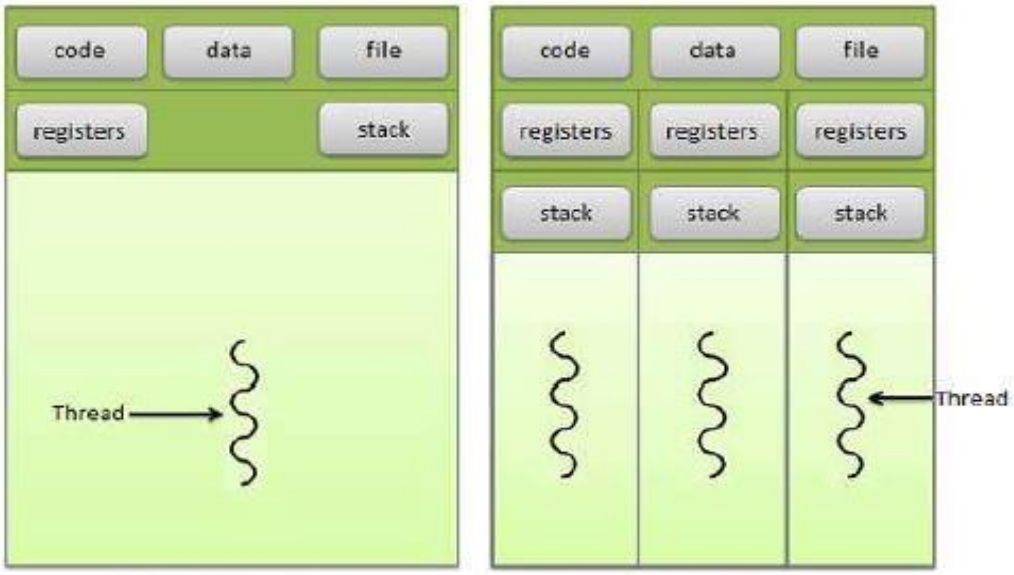
Q1. Differentiate between a process and thread with example.

ANSWER:

Thread:

A thread is a flow of execution through the process code, with its own program counter, system registers and stack. A thread is also called a light weight process. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. Following figure shows the working of the single and multithreaded processes.



Single threaded Process Multi-threaded Process

Process:

In computing, a **process** is the instance of a computer program that is being executed by one or many threads. It contains the program code and its activity. Depending on the operating system (OS), a process may be made up of multiple threads of execution that execute instructions concurrently.

Difference between Process and Thread

S.N.	Process	Thread
1	Process is heavy weight or resource intensive.	Thread is light weight taking lesser resources than a process.
1	Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
1	In multiple processing environments each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
1	If one process is blocked then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, second thread in the same task can run.
1	Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
1	In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Q2. List and discuss few types of thread.

ANSWER:

Types of Thread:

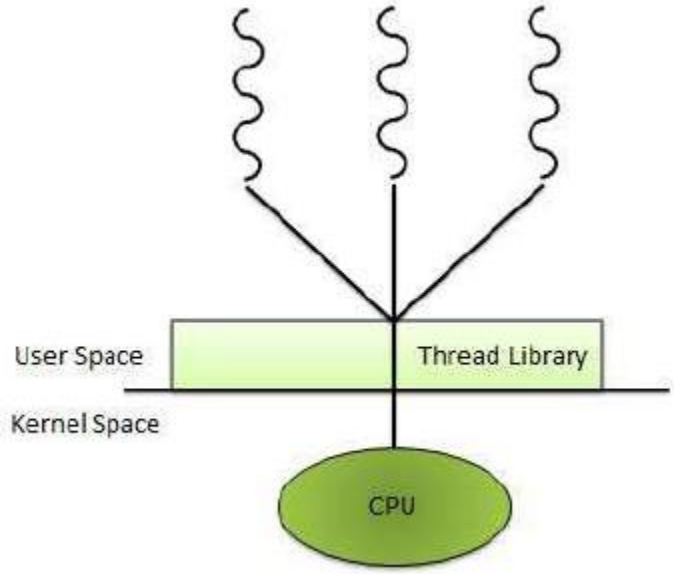
Threads are implemented in following two ways

User Level Threads -- User managed threads

Kernel Level Threads **--** Operating System managed threads acting on kernel, an operating system core.

User Level Threads:

In this case, application manages thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application begins with a single thread and begins running in that thread.



Advantages

- Thread switching does not require Kernel mode privileges.
- User level thread can run on any operating system.

- Scheduling can be application specific in the user level thread.
- User level threads are fast to create and manage.

Disadvantages

- In a typical operating system, most system calls are blocking.
- Multithreaded application cannot take advantage of multiprocessing.

Kernel Level Threads:

- In this case, thread management done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.
- The Kernel maintains context information for the process as a whole and for individuals' threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.

Advantages

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can multithreaded.

Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within same process requires a mode switch to the Kernel.

Multithreading Models

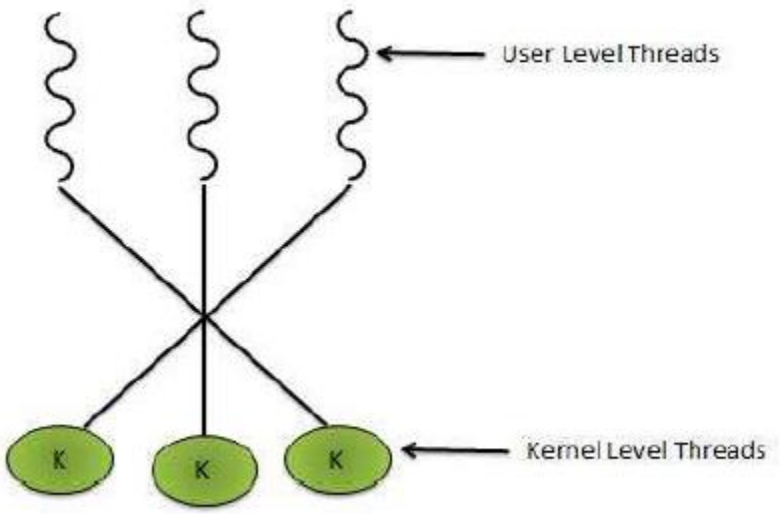
Some operating system provides a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

- ✓ Many to many relationships.
- ✓ Many to one relationship.
- ✓ One to one relationship.

Many to Many Model:

In this model, many user level threads multiplexes to the Kernel thread of smaller or equal numbers. The number of Kernel threads may be specific to either a particular application or a particular machine.

Following diagram shows the many to many model. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallels on a multiprocessor.

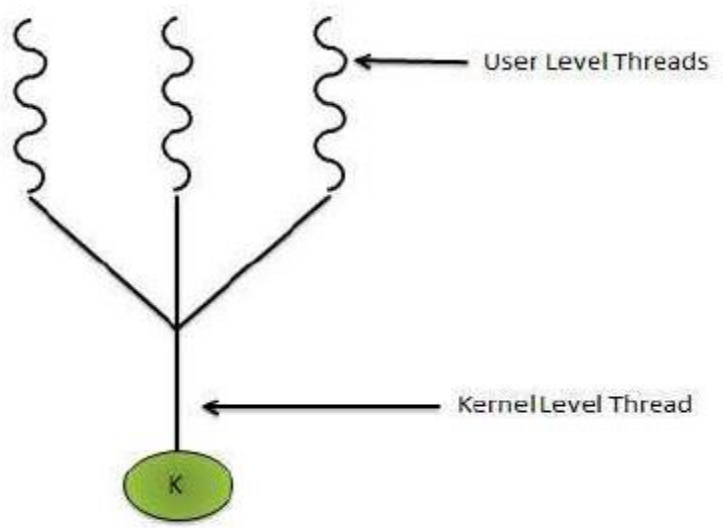


Many to One Model

Many to one model maps many user level threads to one Kernel level thread. Thread management is done in user space. When thread makes a blocking system call, the

entire process will be blocks. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

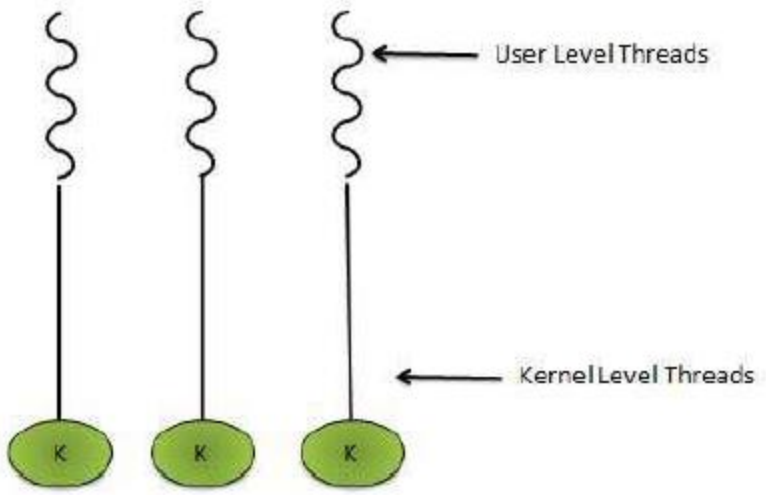
If the user level thread libraries are implemented in the operating system in such a way that system does not support them then Kernel threads use the many to one relationship modes.



One to One Model

There is one to one relationship of user level thread to the kernel level thread. This model provides more concurrency than the many to one model. It also another thread to run when a thread makes a blocking system call. It supports multiple thread to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, Windows NT and windows 2000 use one to one relationship model.



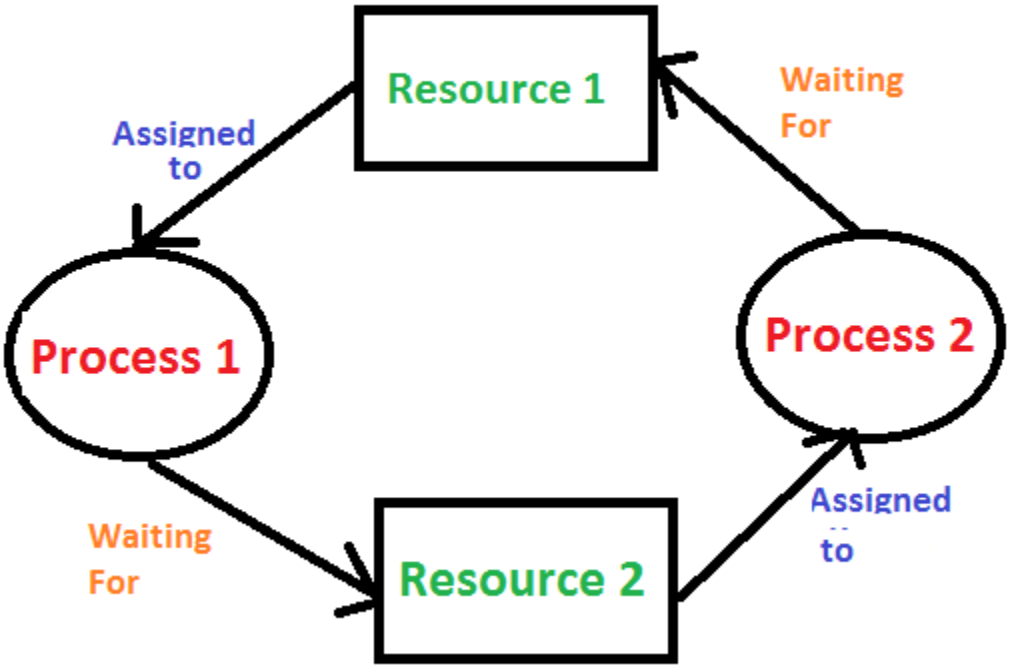
Q3. What is a deadlock? In what situations it occurs in an OS.

ANSWER:

Deadlock:

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.
Consider an example when two trains are coming toward each other on same track and there is only one track, none of the trains can move once they are in front of each other. Similar situation occurs in operating systems when there are two or more processes hold some resources and wait for resources held by other(s).

For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



deadlock occurs in an OS:

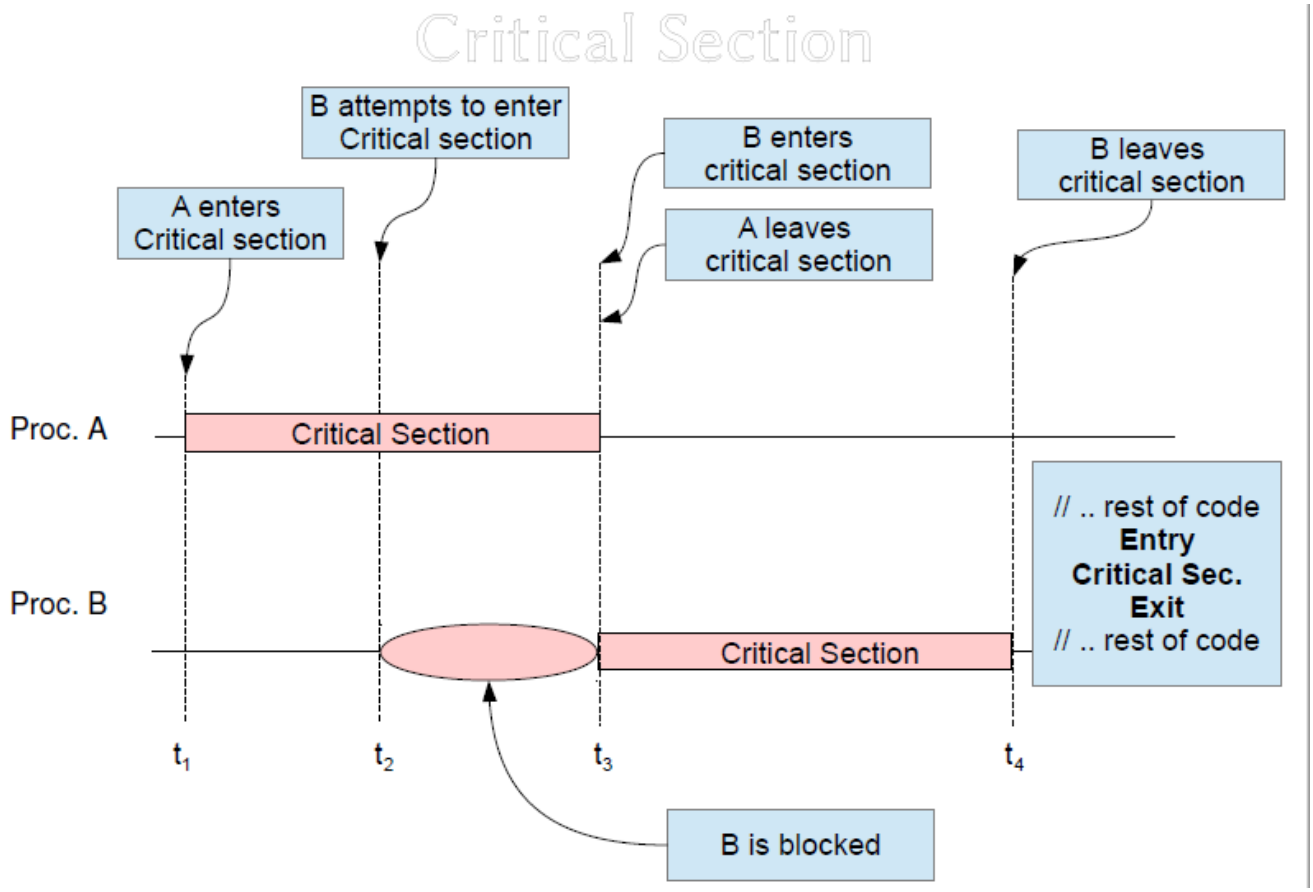
In an operating system, a deadlock occurs when a process or thread enters a waiting state because a requested system resource is held by another waiting process, which in turn is waiting for another resource held by another waiting process.

Q4. Discuss a solution to the critical-section problem must satisfy the three requirements.

ANSWER:

The Critical-Section Problem:

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes.



• The important feature of the system is that, when one process is executing in its critical section, no other process is to be allowed to execute in its critical section.

• Thus, the execution of critical sections by the processes is mutually exclusive in time.

• The critical-section problem is to design a protocol that the processes can use to cooperate.
• Each process must request permission to enter its critical section.

• A solution to the critical-section problem must satisfy the following three requirements:

I. **Mutual Exclusion:** If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.

II. **Progress:** If no process is executing in its critical section, and there are some processes that want to enter into their own critical sections, then the decision of whom to enter must not be postponed indefinitely.

III. **Bounded Waiting:** There exist a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Q5. Differentiate between dynamic loading and dynamic linking with example.

ANSWER:

Dynamic Loading:

In dynamic loading, a routine of a program is not loaded until it is called by the program. All routines are kept on disk in a re-locatable load format. The main program is loaded into memory and is executed. Other routines methods or modules are loaded on request. Dynamic loading makes better memory space utilization and unused routines are never loaded.

Dynamic Linking:

Linking is the process of collecting and combining various modules of code and data into an executable file that can be loaded into memory and executed. Operating system can link system level libraries to a program. When it combines the libraries at load time, the linking is called static linking and when this linking is done at the time of execution, it is called as dynamic linking.

In static linking, libraries linked at compile time, so program code size becomes bigger whereas in dynamic linking libraries linked at execution time so program code size remains smaller.

Differentiate between Dynamic loading and Dynamic Linking:

- Dynamic loading does not require special support from Operating system, it is the responsibility of the programmer to check whether the routine that is to be loaded does not exist in main memory.
- Dynamic Loading load routine in main memory on call.
- Dynamic Linking requires special support from operating system, the routine loaded through dynamic linking can be shared across various processes.
- . Dynamic Linking load routine in main memory during execution time, if call happens before execution time it is postponed till execution time.

EXAMPLE:

- The dynamic loading for example can be created using Load Library call in C or C++.

- Dynamic linking the linker while creating the exe does minimal work. For the dynamic linker to work it actually has to load the libraries too. Hence, it's also called linking loader.

Q6. Write your understanding about logical Vs Physical address space?

ANSWER:

Definition of Logical Address

Address generated by CPU while a program is running is referred as Logical Address. The logical address is virtual as it does not exist physically. Hence, it is also called as Virtual Address. This address is used as a reference to access the physical memory location. The set of all logical addresses generated by a program's perspective is called Logical Address Space.

The logical address is mapped to its corresponding physical address by a hardware device called Memory-Management Unit. The address-binding methods used by MMU generates identical logical and physical address during compile time and load time. However, while run-time the address-binding methods generate different logical and physical address.

Definition of Physical Address

Physical Address identifies a physical location in a memory. MMU (Memory-Management Unit) computes the physical address for the corresponding logical address. MMU also uses logical address computing physical address. The user never deals with the physical address. Instead, the physical address is accessed by its corresponding logical address by the user.

The user program generates the logical address and thinks that the program is running in this logical address. But the program needs physical memory for its execution. Hence, the logical address must be mapped to the physical address before they are used.

The logical address is mapped to the physical address using a hardware called Memory-Management Unit. The set of all physical addresses

corresponding to the logical addresses in a Logical address space is called Physical Address Space.

Logical versus Physical Address Space:

An address generated by the CPU is a logical address whereas address actually available on memory unit is a physical address. Logical address is also known as Virtual address.

Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme.

The set of all logical addresses generated by a program is referred to as a logical address space. The set of all physical addresses corresponding to these logical addresses is referred to as a physical address space.

The run-time mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device. MMU uses following mechanism to convert virtual address to physical address.

- The value in the base register is added to every address generated by a user process which is treated as offset at the time it is sent to memory. For example, if the base register value is 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.

- The user program deals with virtual addresses; it never sees the real physical addresses.