Objectives

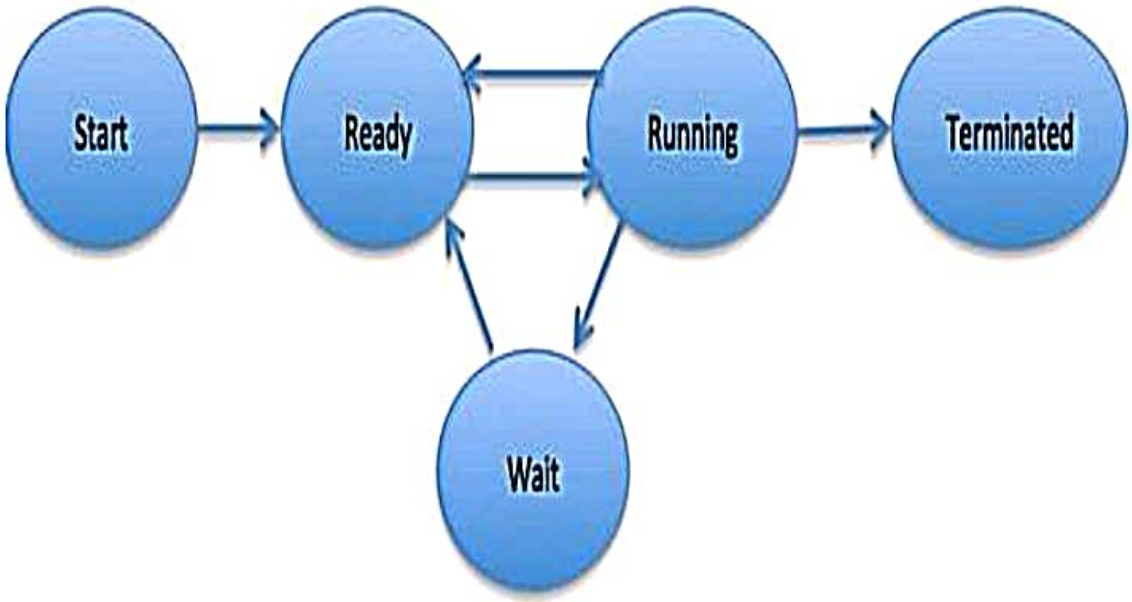
- **Convenience:** An OS makes a computer more convenient to use.
- **Efficiency:** An OS allows the computer system resources to be used in an efficient manner.
- **Ability to evolve:** An OS should be constructed in such a way as to permit the effective development, testing, and introduction of new system functions without interfering with service.

WHAT IS AN OPERATING SYSTEM ?

- Collection of software that manages the computer hardware resources.
- An interface between user and computer hardware.
- Without OS computer becomes useless.

- **New**: the new process is created
- **Running**: Instructions are executed
- **Waiting**: the process is waiting for some event to occur.
- **Ready**: the process is waiting to be assigned to a processor
- **Terminated**: the process has finished execution.

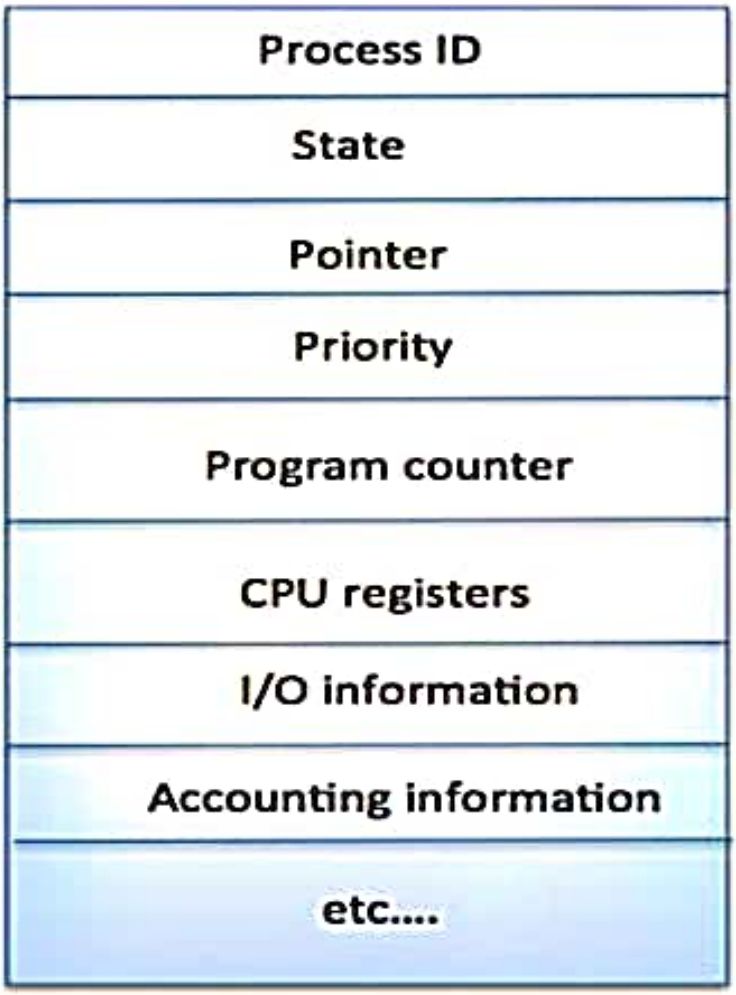
Process states



throughout its

ess terminates.

PROCESS CONTROL BLOCK



Yes, a multithreaded solution using multiple user-level threads can achieve better performance on a multiprocessor system than on a single-processor system.

On a single-processor system, only one thread can be executing at any given tim so multithreading may not necessarily provide a performance boost. However, a multiprocessor system with multiple CPUs or cores, multiple threads can

Dual mode operation

- For proper execution of the OS we must be able to distinguish between the execution of OS code and user defined code.
- **Dual-mode** operation allows OS to protect itself and other system components
- **User mode** and **kernel mode** (or supervisor mode, system mode, or privileged mode).
- A bit called **Mode bit** provided by hardware
- Provides ability to distinguish when system is running user code (1) or kernel code (0)
- Some instructions designated as **privileged**, only executable in kernel mode
- System call changes mode to kernel, return from call resets it to user
- The dual mode of operation provides us with the means for protecting the OS from users and users from one another.

Benefits of Multithreads

- **Enhanced throughput of the system:** When the process is split into many threads, and each thread is treated as a job, the number of jobs done in the unit time increases. That is why the throughput of the system also increases.
- **Effective Utilization of Multiprocessor system:** When you have more than one thread in one process, you can schedule more than one thread in more than one processor.
- **Faster context switch:** The context switching period between threads is less than the process context switching. The process context switch means more overhead for the CPU.
- **Responsiveness:** When the process is split into several threads, and when a thread completes its execution, that process can be responded to as soon as possible.
- **Communication:** Multiple-thread communication is simple because the threads share the same address space, while in process, we adopt just a few exclusive communication strategies for communication between two processes.
- **Resource sharing:** Resources can be shared between all threads within a process, such as code, data, and files. Note: The stack and register cannot be shared between threads. There is a stack and register for each thread.

Need of Thread:

- It takes less time to create a new thread in an existing process than to create a new process.
- Threads can share the common data, they do not need to use Inter-Process communication.
- Context switching is faster when working with threads.
- It takes less time to terminate a thread than a process.

System Boot

- When power initialized on system, execution starts at a fixed memory location
  - Firmware ROM used to hold initial boot code
- Operating system must be made available to hardware so hardware can start it
  - Small piece of code – **bootstrap loader**, stored in **ROM** or **EEPROM** locates the kernel, loads it into memory, and starts it
  - Sometimes two-step process where **boot block** at fixed location loaded by ROM code, which loads bootstrap loader from disk
- Common bootstrap loader, **GRUB**, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then **running**

Advantages:

- Multicore and FPGA processing helps to increase the performance of an embedded system.
- Also helps to achieve scalability, so the system can take advantage of increasing numbers of cores and FPGA processing power over time.
- Concurrent systems that we create using multicore programming have multiple tasks executing in parallel. This is known as concurrent execution. When multiple parallel tasks are executed by a processor, it is known as multitasking.

- **CPU utilization:** the CPU should kept busy as possible.
- **Throughput:** it is the number of processes completed per unit time.
- **Turnaround time:** the interval from the time of submission of a process to the time of completion is the turn around time

Sections of a Program in OS

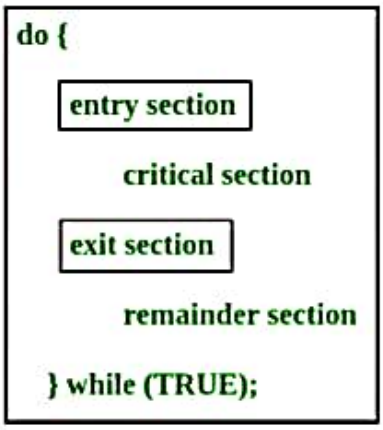
Following are the four essential sections of a program:

1. Entry Section: This decides the entry of any process.

2. Critical Section: This allows a process to enter and modify the shared variable.

3. Exit Section: This allows the process makes sure that the process is removed through this section once it's done executing.

4. Remainder Section: Parts of the Code, not present in the above three sections are collectively called Remainder Section.



Lock Variable

- This is the simplest synchronization mechanism. This is a Software Mechanism implemented in User mode. This is a busy waiting solution which can be used for more than two processes.
- In this mechanism, a Lock variable **lock** is used. Two values of lock can be possible, either 0 or 1. Lock value 0 means that the critical section is vacant while the lock value 1 means that it is occupied.
- A process which wants to get into the critical section first checks the value of the lock variable. If it is 0 then it sets the value of lock as 1 and enters into the critical section, otherwise it waits.

```
Entry Section
While (Lock! = 0);
Lock = 1;
// Critical Section
Exit Section
Lock = 0:
```

A Semaphore is an integer variable, which can be accessed only through two atomic operations for process synchronization

- *wait() operation (P- to test)*-The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.
- *signal()* operation (V- to increment)-The signal operation increments the value of its argument S.

Rules/conditions for Critical Section

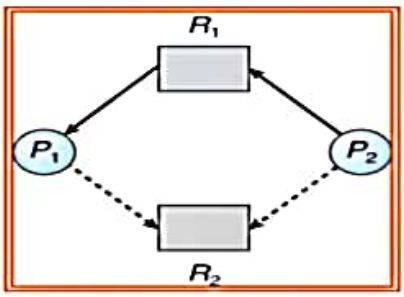
- There are three rules that need to be enforced in the critical section. They are:

- **Mutual Exclusion-** Mutual exclusion implies that only one process can be inside the critical section at any time. If any other processes require the critical section, they must wait until it is free.
- **Progress-** Progress means that if a process is not using the critical section, then it should not stop any other process from accessing it. In other words, any process can enter a critical section if it is free.
- **Bounded Waiting-** Bounded waiting means that each process must have a limited waiting time. It should not wait endlessly to access the critical section.

- If the necessary condition of deadlock is in place it is still possible to avoid feedback by allocating resources carefully.
- **A deadlock avoidance algorithm** dynamically examines the resources allocation state to ensure that a circular wait condition case never exists.
- Where the resources allocation state is defined by the available and allocated resources and the maximum demand of the process.

1) Resource allocation graph

- In resource allocation graph for deadlock avoidance we introduce a third kind of edge called the **claim edg** which is a dotted line from a process towards a resource meaning that the resource can be requested by th process in future.
- Whenever a process requests for a resource the claim edge is changed to request edge and if the resourc can be granted the request edge is changed to assignment edge. After this change look for a cycle in th graph. If no cycle exists, then the system is in safe state and the deadlock will not occur else the system is unsafe state and deadlock may or may not occur.

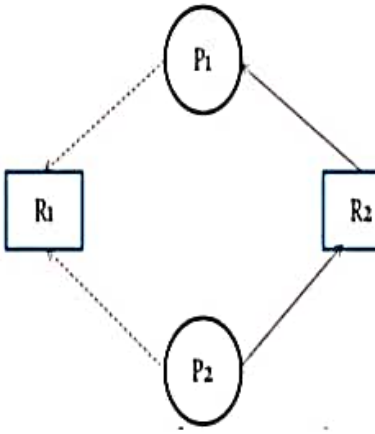


Important to remember

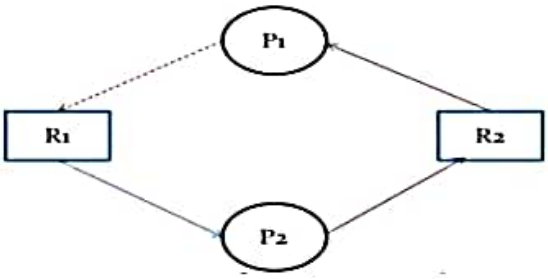
- Applicable if each resource has only one instance.
- Claim edge : ——-> from process to resource
- Process may request resource in future
- When process requests resource change claim edge to request edge.
- Request edge is converted into assignment edge only if the conversion does not lead to the formation of cycle in the graph.

Example

- Consider the scenario in Figure 1. P1 is holding R2 and P2 is requesting R2. P1 and P2 can request for R1 in the future.



- P2 request to R1 should not be granted because granting the request will lead to the formation of cycle in the graph. P2 request for R1, since R1 is free it can be allocated to P1. The resulting graph will be as shown in Figure 2. This results in cycle formation in the resource-allocation graph. Hence, the state is an unsafe state and this request can not be granted.



2) Bankers's algorithm

- The resource allocation graph algorithms not applicable to the system with multiple instances of the type of each resource. So for this system Banker's algorithm is used.
- Here whenever a process enters into the system it must declare maximum demand

Deadlock prevention:

- This ensures that the system never enters the deadlock state.
- Deadlock prevention is a set of methods for ensuring that at least one of th necessary conditions cannot hold.
- By ensuring that at least one of the conditions cannot hold, we can prevent th occurrence of a deadlock.

1. Denying mutual exclusion:

- Mutual exclusions conditions must hold for non-sharable resources.
- Printer cannot be shared simultaneously shared by prevent processes.
- Sharable resources- example read-only files.
- If several processes attempt to open a read-only file at the same time, they ca be granted simultaneously access to the file.
- A process never needs to wait for a sharable resources.

2. Denying Hold and wait:

- Whenever a process request a resource, it does not hold any other resource.
- One technique that can be used requires each process to request and b allocated all its resources before it begins execution.
- Another technique is before it can request any additional resources, it mu release all the resources that it is currently allocated.
- These technique has two main **disadvantages**:

1. Resource utilization may be low, since many of the resources may be allocate but unused for a long time.
2. We must request all the resources at the beginning for both protocol Starvation is possible.

3. Denying No Preemption:

- If a process is holding some resources and requests another resource that cann be immediately allocated to it. (that is the process must wait), then all resourc currently being held are pre-empted.
- These resources are implicitly released.
- The process will be restarted only when it can regain its old resources.

4. Denying Circular wait:

- Impose a total ordering of all resources types and allow each process to reque for resources in an increasing order of enumeration.
- Let R={R1, R2...Rn} be the set of resource types.
- Assign to each resource type a unique integer number.
- If the set of resource types R includes tape drives, disk drives and printers.

  F(tapedrive)=1, F(diskdrive)=5, F(printer)=12.
- Each process can request resources only in an increasing order of enumeration.

Race Condition in OS

- When more than one processes execute the same code or access the same memory/shared variable, it is possible that the output or value of the shared variable is wrong.
- In this condition, all processes race ahead in order to prove that their output is correct. This situation is known as race condition.
- When multiple processes access and manipulate the same data concurrently the outcome depends on the order in which these processes accessed the shared data. When the output of multiple thread execution differs according to the order in which the threads execute, **a race condition** occurs.
- We can avoid it if we treat the **critical section** as an atomic instruction and maintain proper thread synchronization using locks or atomic variables.

Parameter	LOGICAL ADDRESS	PHYSICAL ADDRESS
Basic	generated by CPU	location in a memory unit
Address Space	Logical Address Space is set of all logical addresses generated by CPU in reference to a program.	Physical Address is set of all physical addresses mapped to the corresponding logical addresses.
Visibility	User can view the logical address of a program.	User can never view physical address of program.
Generation	generated by the CPU	Computed by MMU
Access	The user can use the logical address to access the physical address.	The user can indirectly access physical address but not directly.
Editable	Logical address can be change.	Physical address will not change.
Also called	virtual address.	real address.

Load Time Address Binding

- It will be done after loading the program into memory.
- This type of address binding will be done by the **OS memory manager i.e loader**.