UNIT-IV

UNIT-IV:

Operating System Support: Introduction, The Operating System Layer, Protection, Processes and Threads –Address Space, Creation of a New Process, Threads.

OPERATING SYSTEM SUPPORT

The Operating System layer:

Networking operating systems:-

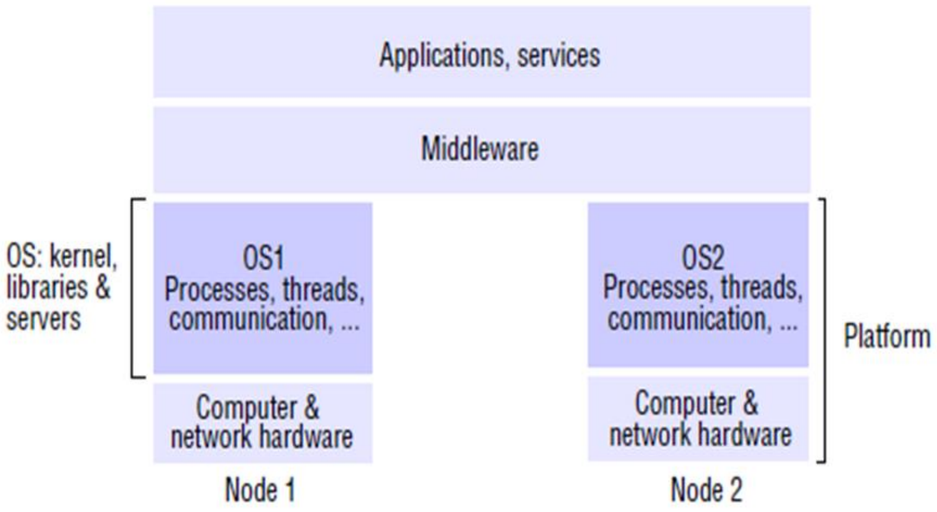

Applications, services
Middleware
OS: kernel, libraries & servers
OS1 Processes, threads, communication, ...
OS2 Processes, threads, communication, ...
Platform
Computer & network hardware
Computer & network hardware
Node 1
Node 2

Figure shows the operating system layer at each of two nodes supporting a common middleware layer and providing distributed infrastructure for applications and services.

The task of any operating system is to provide abstractions of the underlying physical resources – the processors, memory, networks, and storage media.

UNIX and Windows are examples of *network operating systems*. They have a networking capability (TCP/IP) built into them and so can be used to access remote resources.

Access is network-transparent for some – not all – types of resource. For example, through a distributed file system such as NFS, users have network-transparent access to files. But FTP/TELNET (or SSH) do not provide such network transparent access.

The defining characteristic is that the nodes running a network operating system retain autonomy in managing their own processing resources. In other words, there are multiple system images, one per node.

With a network operating system, a user can remotely log into another computer, using SSH, for example, and run processes there.

However, while the operating system manages the processes running at its own node, it does not manage processes across the nodes.

Distributed Operating System:-

On the other hand, we can envisage an operating system in which users are never concerned with where their programs run, or the location of any resources. There is a single system image.

The operating system has control over all the nodes in the system, and it transparently locates new processes at whatever node suits its scheduling policies.

For example, it could create a new process at the least-loaded node in the system, to prevent individual nodes becoming unfairly overloaded.

An operating system that produces a single system image like this for all the resources in a distributed system is called a *distributed operating system*.

There are no distributed operating systems in general use, only network operating systems such as UNIX, Mac OS and Windows. There are two reasons.

The first is that users have much invested in their application software, which often meets their current problem-solving needs; they will not adopt a new operating system that will not run their applications, whatever efficiency advantages it offers.

The second reason against the adoption of distributed operating systems is that users tend to prefer to have a degree of autonomy for their machines, even in a closely knit organization. This is particularly so because of performance.

The combination of middleware and network operating systems provides an acceptable balance between the requirement for autonomy on the one hand and network transparent resource access on the other.

The network operating system enables users to run their favorite word processors and other standalone applications. Middleware enables them to take advantage of services that become available in their distributed system

PROTECTION AND SECURITY:

Protection and security requires that computer resources such as CPU, softwares, memory etc. are protected. This extends to the operating system as well as the data in the system. This can be done by ensuring integrity, confidentiality and availability in the operating system. The system must be protect against unauthorized access, viruses, worms etc.

Threats to Protection and Security

A threat is a program that is malicious in nature and leads to harmful effects for the system. Some of the common threats that occur in a system are:

Virus

Viruses are generally small snippets of code embedded in a system. They are very dangerous and can corrupt files, destroy data, crash systems etc. They can also spread further by replicating themselves as required.

Trojan Horse

A trojan horse can secretly access the login details of a system. Then a malicious user can use these to enter the system as a harmless being and wreak havoc.

Trap Door

A trap door is a security breach that may be present in a system without the knowledge of the users. It can be exploited to harm the data or files in a system by malicious people.

Worm

A worm can destroy a system by using its resources to extreme levels. It can generate multiple copies which claim all the resources and don't allow any other processes to access them. A worm can shut down a whole network in this way.

Denial of Service

These type of attacks do not allow the legitimate users to access a system. It overwhelms the system with requests so it is overwhelmed and cannot work properly for other user.

Protection and Security Methods

The different methods that may provide protect and security for different computer systems are:

Authentication

This deals with identifying each user in the system and making sure they are who they claim to be. The operating system makes sure that all the users are authenticated before they access the system. The different ways to make sure that the users are authentic are:

- Username/ Password

  Each user has a distinct username and password combination and they need to enter it correctly before they can access the system.

- User Key/ User Card

  The users need to punch a card into the card slot or use they individual key on a keypad to access the system.

- User Attribute Identification

  Different user attribute identifications that can be used are fingerprint, eye retina etc. These are unique for each user and are compared with the existing samples in the database. The user can only access the system if there is a match.

One Time Password

These passwords provide a lot of security for authentication purposes. A one time password can be generated exclusively for a login every time a user wants to enter the system. It cannot be used more than once. The various ways a one time password can be implemented are:

- **Random Numbers**

  The system can ask for numbers that correspond to alphabets that are pre arranged. This combination can be changed each time a login is required.

- **Secret Key**

  A hardware device can create a secret key related to the user id for login. This key can change each time.

Modern **operating systems** allow a **process** to be divided into multiple **threads** of execution. The **threads** within a **process** share all **process** management information except for information directly related to execution. This information is moved out of the PCB into **thread** control block (TCBs).

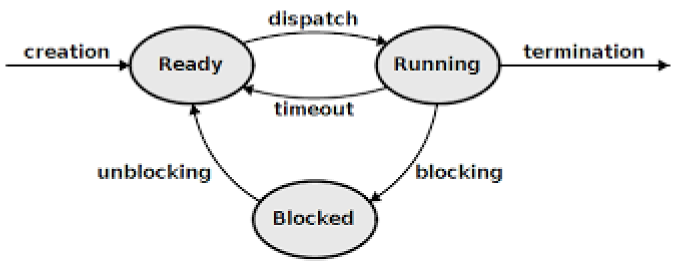
__Processes and Threads__

A *process* is an instance of program execution. This means, for example, that if you open up two browser windows then you have two processes, even though they are running the same program.

The life-cycle of a process can be described by a state diagram which has states representing the execution status of the process at various times and transitions that represent changes in execution status.

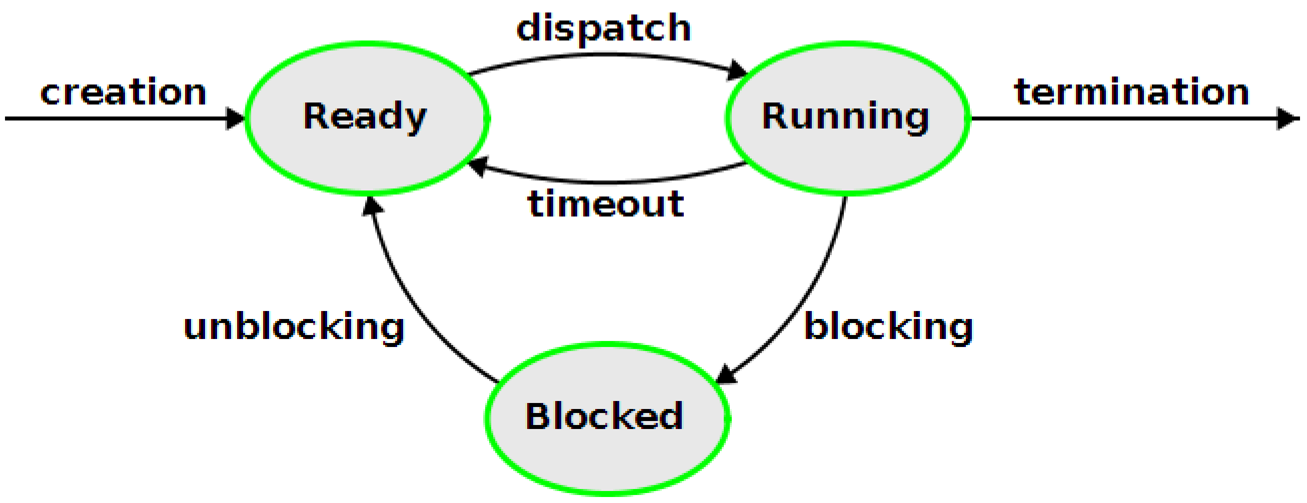
The operating system maintains management information about a process in a **process control block (PCB).**
Modern operating systems allow a process to be divided into multiple threads of execution, which share all process management information except for information directly related to execution. This information is held in a **thread control block (TCB).**

Threads in a process can execute different parts of the program code at the same time. They can also execute the same parts of the code at the same time, but with different execution state:
They have independent current instructions; that is, they have (or appear to have) independent program counters.
They are working with different data; that is, they are (or appear to be) working with independent registers.
State Diagram



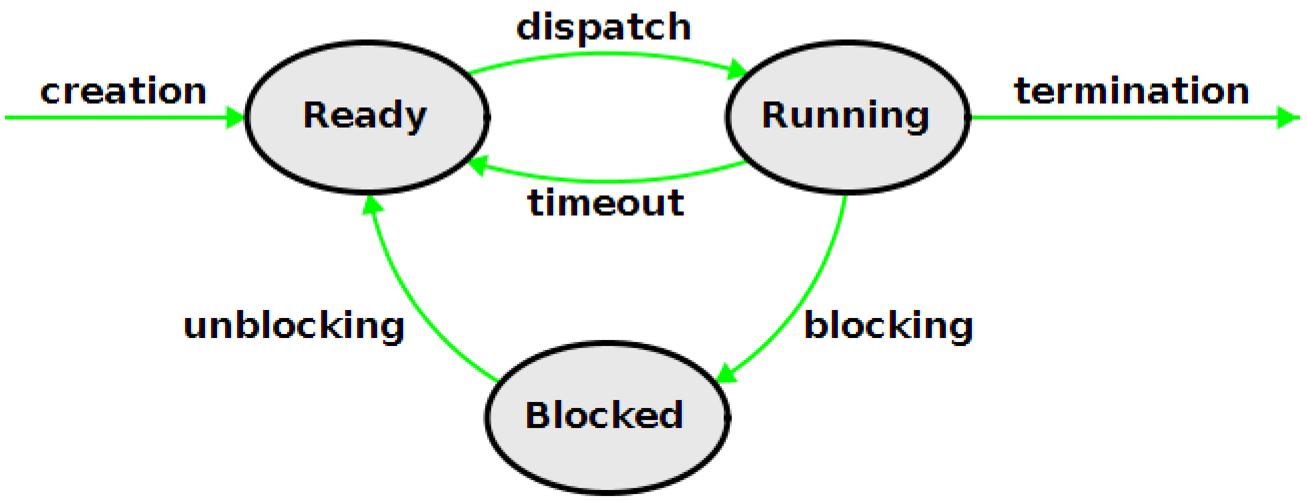
The state diagram for a process captures its life-cycle. The states represent the execution status of the process; the transitions represent changes of execution state.
Each active process has its own execution status, so there is a state diagram for each process. There are relationships between the states of various processes that are maintained by the operating system.

States



The state of a process represent its execution status.

Transitions



The transitions of a process represent changes of its execution state. The transitions can be described in terms of their causes and the resulting actions taken by the operating system.

Process Control Block

An operating system maintains a structure called a *process control block (PCB)* for each of its active processes. The PCB for a process contains or refers to the following kinds of information.

- resource management information
- administrative information
- an execution snapshot

Threads

Modern operating systems allow a process to be divided into multiple threads of execution. The threads within a process share all process management information except for information directly related to execution. This information is moved out of the PCB into thread control block (TCBs).

Threads in a process can execute different parts of the program code at the same time. They can also execute the same parts of the code at the same time, but with different execution state:

- They have independent current instructions; that is, they have (or appear to have) independent program counters.
- They are working with different data; that is, they are (or appear to be) working with independent registers.

This is accomplished by moving execution state out of the PCB into thread control blocks (TCBs). Each thread has its own TCB.

Processes start out with a single main thread. The main thread can create new threads using a thread fork system call. The new threads can also use this system call to create more threads. Consequently, a thread not only belongs to a process; it also has a parent thread - the thread that created it.

Threads

Modern operating systems allow a process to be divided into multiple threads of execution. The threads within a process share all process management information except for information directly related to execution. This information is moved out of the PCB into thread control block (TCBs).

Threads in a process can execute different parts of the program code at the same time. They can also execute the same parts of the code at the same time, but with different execution state:

- They have independent current instructions; that is, they have (or appear to have) independent program counters.
- They are working with different data; that is, they are (or appear to be) working with independent registers.

This is accomplished by moving execution state out of the PCB into thread control blocks (TCBs). Each thread has its own TCB.

Processes start out with a single main thread. The main thread can create new threads using a thread fork system call. The new threads can also use this system call to create more threads. Consequently, a thread not only belongs to a process; it also has a parent thread - the thread that created it.

Thread Control Block

A *thread control block (TCB)* contains a thread identifier, a reference to the PCB for the process to which it belongs, a reference to the TCB for the thread that created it, and an execution snapshot.

A thread has indirect access to non-execution related resources through the PCB of the process to which it belongs. Thus threads within a process share memory, open files, and I/O streams.

Thread Fork

A *thread fork* operation behaves like a function call. It creates a new child thread whose execution snapshot is identical to the parent thread's snapshot except for the register that contains the function return value. For the child thread the return value is 0; for the parent the return value is the child thread's thread identifier.

Normally, the return value is used in the condition of an if-else statement. If the return value is 0 then child code is executed; otherwise, parent code is executed.

At the system call level, a thread fork operation involves

- saving a new execution snapshot for the parent thread in its TCB
- creating a new TCB whose execution snapshot is identical to the parent's execution snapshot
- changing the return value register in the two snapshots

The operating system then has the option of

- dispatching the child, putting the parent in the ready queue, or
- dispatching the parent, putting the child in the ready queue, or
- putting both the child and the parent in the ready queue, scheduling and dispatching a different thread.

Process Address Space

The process address space is the set of logical addresses that a process references in its code. For example, when 32-bit addressing is in use, addresses can range from 0 to 0x7fffffff; that is, 2^31 possible numbers, for a total theoretical size of 2 gigabytes.

The operating system takes care of mapping the logical addresses to physical addresses at the time of memory allocation to the program. There are three types of addresses used in a program before and after memory is allocated –

S.N.	Memory Addresses & Description
1	**Symbolic addresses** The addresses used in a source code. The variable names, constants, and instruction labels are the basic elements of the symbolic address space.
2	**Relative addresses** At the time of compilation, a compiler converts symbolic addresses into relative addresses.
3	**Physical addresses** The loader generates these addresses at the time when a program is loaded into main memory.

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 runtime 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.

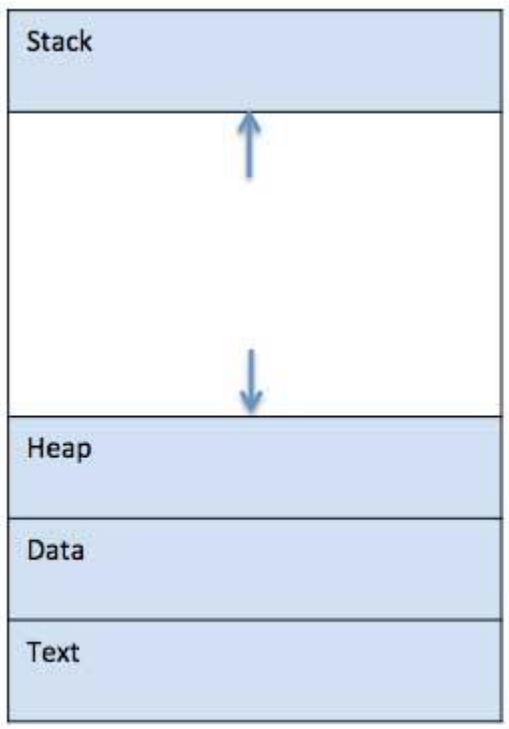
Process

A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

A process is defined as an entity which represents the basic unit of work to be implemented in the system.

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections ─ stack, heap, text and data. The following image shows a simplified layout of a process inside main memory –



S.N.	Component & Description
1	**Stack** The process Stack contains the temporary data such as method/function parameters, return address and local variables.
2	**Heap** This is dynamically allocated memory to a process during its run time.
3	**Text**

	This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
4	**Data** This section contains the global and static variables.

Program

A program is a piece of code which may be a single line or millions of lines. A computer program is usually written by a computer programmer in a programming language. For example, here is a simple program written in C programming language –

```
#include <stdio.h>

int main() {
  printf("Hello, World! \n");
  return 0;
}
```

A computer program is a collection of instructions that performs a specific task when executed by a computer. When we compare a program with a process, we can conclude that a process is a dynamic instance of a computer program.

A part of a computer program that performs a well-defined task is known as an **algorithm**. A collection of computer programs, libraries and related data are referred to as a **software**.

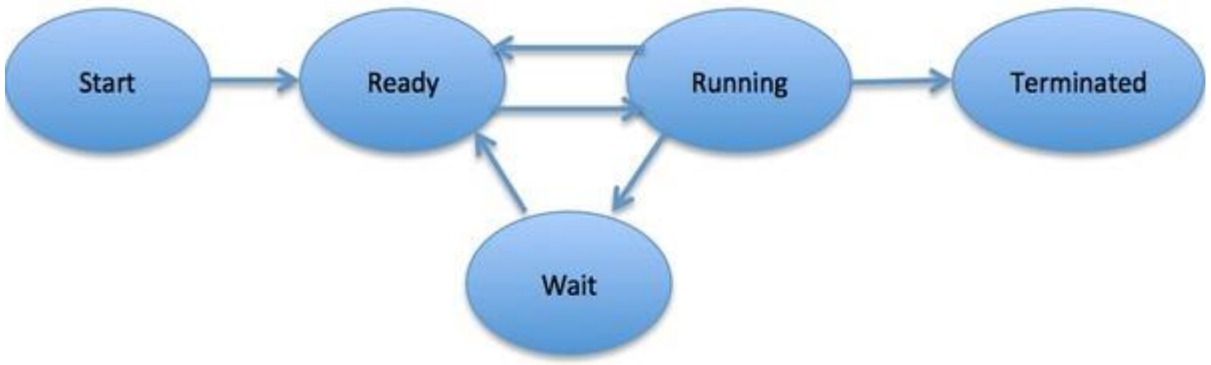
Process Life Cycle

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.

In general, a process can have one of the following five states at a time.

S.N.	State & Description
1	**Start** This is the initial state when a process is first started/created.
2	**Ready** The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process.
3	**Running** Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.

4	**Waiting** Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.
5	**Terminated or Exit** Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.



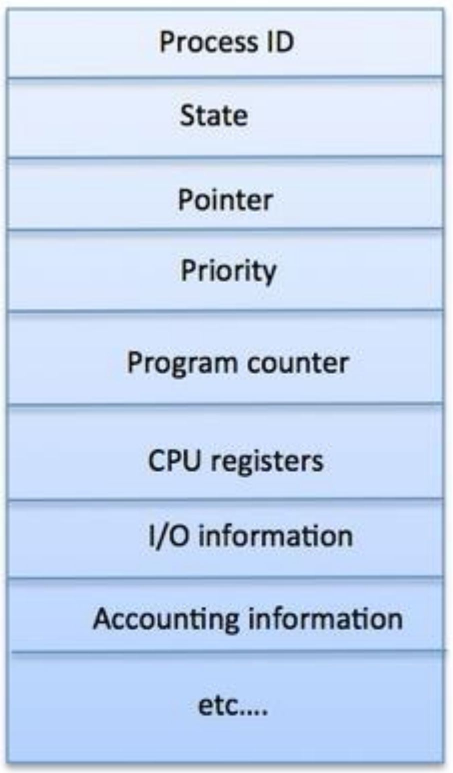
Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table –

S.N.	Information & Description
1	**Process State** The current state of the process i.e., whether it is ready, running, waiting, or whatever.
2	**Process privileges** This is required to allow/disallow access to system resources.
3	**Process ID** Unique identification for each of the process in the operating system.

4	**Pointer** A pointer to parent process.
5	**Program Counter** Program Counter is a pointer to the address of the next instruction to be executed for this process.
6	**CPU registers** Various CPU registers where process need to be stored for execution for running state.
7	**CPU Scheduling Information** Process priority and other scheduling information which is required to schedule the process.
8	**Memory management information** This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.
9	**Accounting information** This includes the amount of CPU used for process execution, time limits, execution ID etc.
10	**IO status information** This includes a list of I/O devices allocated to the process.

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB −



The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.