Lecture Page 1-7

Operating System Concepts

An operating system acts as an intermediary between the user of a computer and the computer hardware. The purpose of an operating system is to provide an environment in which a user can execute

Programs in a convenient and efficient manner. An operating system is software that manages the computer hardware. The hardware must provide appropriate mechanisms to ensure the correct operation of the computer system and to prevent user programs from interfering with the proper operation of the system.

An **operating system** is a program that manages a computer’s hardware. It also provides a basis for application programs and acts as an intermediary between the computer user and the computer hardware. An amazing aspect of operating systems is how they vary in accomplishing these tasks

What Operating Systems Do?

The **hardware**—the **central processing unit (CPU)**, the **memory**, and the **input/output (I/O) devices**—provides the basic computing resources for the system. The **application programs**—such as word processors, spreadsheets, compilers, and Web browsers—define the ways in which these resources are used to solve users’ computing problems. The operating system controls the hardware and coordinates its use among the various application programs for the various users.

1-User View

2-System View

3-Defining Operating Systems

Defining Operating Systems?

the low-level software that supports a computer's basic functions, such as scheduling tasks and controlling Input Devices

Computer-System Organization

1.2.1

Modern general-purpose computer system consists of one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory.

Each device controller is in charge of a specific type of device (for example, disk drives, audio devices, or video displays). The CPU and the device controllers can execute in parallel, competing for memory cycles. To ensure orderly access to the shared memory, a memory controller synchronizes access to the memory

***STORAGE DEFINITIONS AND NOTATION***

The basic unit of computer storage is the **bit**. A bit can contain one of two values, 0 and 1. All other storage in a computer is based on collections of bits Given enough bits, it is amazing how many things a computer can represent: numbers, letters, images, movies, sounds, documents, and programs, to name a few. A **byte** is 8 bits, and on most computers it is the smallest convenient chunk of storage. For example, most computers don’t have an instruction to move a bit but do have one to move a byte. A less common term is **word**, which is a given computer architecture’s native unit of data. A word is made up of one or more bytes. For example, a computer that has 64-bit registers and 64-bit memory addressing typically has 64-bit (8-byte) words. A computer executes many operations in its native word size rather than a byte at a time. Computer storage, along with most computer throughput, is generally measured and manipulated in bytes and collections of bytes. A **kilobyte**, or **KB**, is 1,024 bytes; a **megabyte**, or **MB**, is 1,0242 bytes; a **gigabyte**, or **GB**, is 1,0243 bytes; a **terabyte**, or **TB**, is 1,0244 bytes; and a **petabyte**, or **PB**, is 1,0245 bytes. Computer manufacturers often round off these numbers and say that a megabyte is 1 million bytes and a gigabyte is 1 billion bytes. Networking measurements are an exception to this general rule; they are given in bits (because networks move data a bit at a time).

**1.2.2 Storage Structure**.

The CPU can load instructions only from memory, so any programs to run must be stored there. General-purpose computers run most of their programs from rewritable memory, called main memory (also called **random-access memory**, or **RAM**). Main memory commonly is implemented in a semiconductor technology called **dynamic random-access memory (DRAM)**.

This arrangement usually is not possible for the following two reasons:

**1.** Main memory is usually too small to store all needed programs and data permanently.

**2.** Main memory is a **volatile** storage device that loses its contents when power is turned off or otherwise lost.

**1.2.3 I/O Structure**

Storage is only one of many types of I/O devices within a computer. A large portion of operating system code is dedicated to managing I/O, both because of its importance to the reliability and performance of a system and because of the varying nature of the devices. Next, we provide an overview of I/O. A general-purpose computer systemconsists of CPUs andmultiple device controllers that are connected through a common bus. Each device controller is in charge of a specific type of device. Depending on the controller, more than one device may be attached. For instance, seven or more devices can be attached to the **small computer-systems interface (SCSI)** controller. A device

controller maintains some local buffer storage and a set of special-purpose registers. The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage. Typically, operating systems have a **device driver** for each device controller. This device driver understands the device controller and provides the rest of the operating

system with a uniform interface to the device.

Computer-System Architecture

**1.3.1 Single-Processor Systems**

Until recently, most computer systems used a single processor. On a singleprocessor system, there is one main CPUcapable of executing a general-purpose instruction set, including instructions from user processes. Almost all single processor systems have other special-purpose processors as well. They may come in the form of device-specific processors, such as disk, keyboard, and graphics controllers; or, on mainframes, they may come in the form of more general-purpose processors, such as I/O processors that move data rapidly among the components of the system.

**1.3.2 Multiprocessor Systems**

Within the past several years, **multiprocessor systems** (also known as **parallel systems** or **multicore systems**) have begun to dominate the landscape ofcomputing. Such systems have two or more processors in close communication,sharing the computer bus and sometimes the clock, memory, and peripheraldevices. Multiprocessor systems first appeared prominently appeared inservers and have since migrated to desktop and laptop systems. Recently,multiple processors have appeared on mobile devices such as smartphonesand tablet computers.

Three Major Advantages:

**Increased throughput**

**Economy of scale**

**Increased reliability**

**1.3.3 Clustered Systems**

Another type of multiprocessor system is a **clustered system**, which gathers together multiple CPUs. Clustered systems differ from the multiprocessor systems described they are composed of two or more individual systems—or nodes—joined together. Such systems are considered **loosely coupled**. Each node may be a single processor system or a multicore system

Operating-System Operations

Dual-Mode and Multimode Operation

Timer : A timer can be set to interrupt the computer after a specified period. The period may be fixed (for example, 1/60 second) or variable (for example, from 1 millisecond to 1 second). A **variable timer** is generally implemented by a fixed-rate clock and a counter.

Process Management

The operating system is responsible for the following activities in connection

with process management:

• Scheduling processes and threads on the CPUs

• Creating and deleting both user and system processes

• Suspending and resuming processes

• Providing mechanisms for process synchronization

• Providing mechanisms for process communication

Memory Management

Keeping track of which parts of memory are currently being used and who

is using them

• Deciding which processes (or parts of processes) and data to move into

and out of memory

• Allocating and deallocating memory space as needed

Storage Management

File-System Management

Creating and deleting files may contains

• Creating and deleting directories to organize files

• Supporting primitives for manipulating files and directories

• Mapping files onto secondary storage

• Backing up files on stable (nonvolatile) storage media

Mass-Storage Management

* Free-space management
* Storage allocation
* Disk scheduling

**Caching** is an important principle of computer systems. Here’s how it works.

Information is normally kept in some storage system (such as main memory).

As it is used, it is copied into a faster storage system—the cache—on a

**Introduction**

temporary basis. When we need a particular piece of information, we first check whether it is in the cache. If it is, we use the information directly from the cache. If it is not, we use the information from the source, putting a copy in the cache under the assumption that we will need it again soon.

**I/O Systems**

A memory-management component that includes buffering, caching, and

spooling

• A general device-driver interface

• Drivers for specific hardware devices

Protection and Security

**Protection**, then, is any mechanism for controlling the access of processes or users to the resources defined by a computer system.

**Security** to defend a system from external and internal attacks.

**Operating System Types and Operations**

Operating-System Operations

1-**Single user system**

**Single**-**user**, **single** task - As the name implies, this **operating system** is designed to manage the computer so that one **user can** effectively **do** one thing at a time. The Palm **OS** for Palm handheld computers is a good example of a modern **single**-**user**,**single**-task **operating system**.

Example1

**Palm OS** (also known as Garnet **OS**) is a discontinued mobile**operating system** initially developed by **Palm**, Inc., for personal digital assistants (PDAs) in 1996. **Palm OS** was designed for ease of use with a touchscreen-based graphical user interface

Palm is an American tech company that developed and designed Personal Digital Assistants, mobile phones, and software. Palm devices are often remembered as "the first wildly popular handheld computers," responsible for ushering in the smartphone era.

**BATCH SYSTEM**

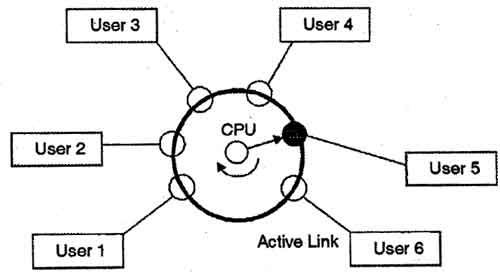
**Batch** operating **system**. The users of a **batch**operating **system** do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group.

**MULTI PROGRAMMED SYSTEM**

Multitasking has the same meaning of **multiprogramming** but in a more general sense, as it refers to having multiple (programs, processes, tasks, threads) running at the same time. This term is used in modern operating **systems** when multiple tasks share a common processing resource (e.g., CPU and Memory).

**TIME SHARING SYSTEM**

**Time**-**sharing** is a technique which enables many people, located at various terminals, to use a particular computer **system** at the same **time**. **Time**-**sharing** or multitasking is a logical extension of multiprogramming. Processor's **time** which is **shared**among multiple users simultaneously is termed as **time**-**sharing**.



**REAL TIME SYSTEM**

A **real**-**time** operating **system** (RTOS) is an operating**system** (OS) intended to serve **realtime** applications that process data as it comes in, typically without buffer delays. Processing **time** requirements (including any OS delay) are measured in tenths of seconds or shorter increments of **time**.

**INTERRUPTS**

In system programming, an interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention. An interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing

**TRAPS**

**Trap** (computing) In computing and **operating systems**, a **trap**, also known as an exception or a fault, is typically a type of synchronous interrupt caused by an exceptional condition (e.g., breakpoint, division by zero, invalid memory access).

**SOFTWARE INTERRUPTS**

A **software interrupt** often occurs when an application **software** terminates or when it requests the **operating system** for some service. This is quite unlike a hardware **interrupt**, which occurs at the hardware level. ... A **software interrupt** can also make use of some of the hardware **interrupt**routines.

**Operating system service**

* User Interface.
* Program Execution.
* File system manipulation.
* Input / Output Operations.
* Communication.
* Resource Allocation.
* Error Detection.
* Accounting.

**SYSTEM CALLS**

**System calls** provide an interface between user programs and **operating system**. It is a programmatic way in which a computer program requests a service from the kernel of the **operating system**

**Operating System Structures**

**Monolithic:**A **monolithic** kernel is an **operating system** architecture where the entire **operating system** is working in kernel space. The **monolithic** model differs from other **operating system** architectures (such as the microkernel architecture) in that it alone defines a high-level virtual interface over computer hardware.

**Microkernel:**n computer science, a **microkernel** (often abbreviated as μ-kernel) is the near-minimum amount of software that can provide the mechanisms needed to implement an **operating system** (**OS**). These mechanisms include low-level address space management, thread management, and inter-process communication (IPC).

**LAYERED**

The **layered** Architecture of **operating system** was developed in 60's in this approach; the **operating system** is broken up into number of **layers**. The bottom **layer** (**layer** 0) is the hardware **layer** and the highest **layer** (**layer** n) is the user interface **layer** as shown in the figure.

**VIRTUAL MACHINE**

A **virtual machine** is a software computer that, like a physical computer, runs an **operating system** and applications. The **virtual machine** is comprised of a set of specification and configuration files and is backed by the physical resources of a host. ... You do not need to touch the key files.

**Processes**

In computing, a process is an instance of a computer program that is being executed. 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.

**PROCESS STATES**

In a multitasking computer system, processes may occupy a variety of states. These distinct states may not be recognized as such by the operating system kernel. However, they are a useful abstraction for the understanding of processes.

**Process scheduler**

he **process scheduling** is the activity of the**process** manager that handles the removal of the running **process** from the CPU and the selection of another **process** on the basis of a particular strategy.**Process scheduling** is an essential part of a Multiprogramming operating systems.

**CONTEXT SWITCHING**

A **context switch** is a procedure that a computer's CPU (central processing unit) follows to change from one task (or process) to another while ensuring that the tasks do not conflict. Effective **context switching** is critical if a computer is to provide user-friendly multitasking.

**OPERATIONS ON PROCESS**

1-Creation

2-Termination

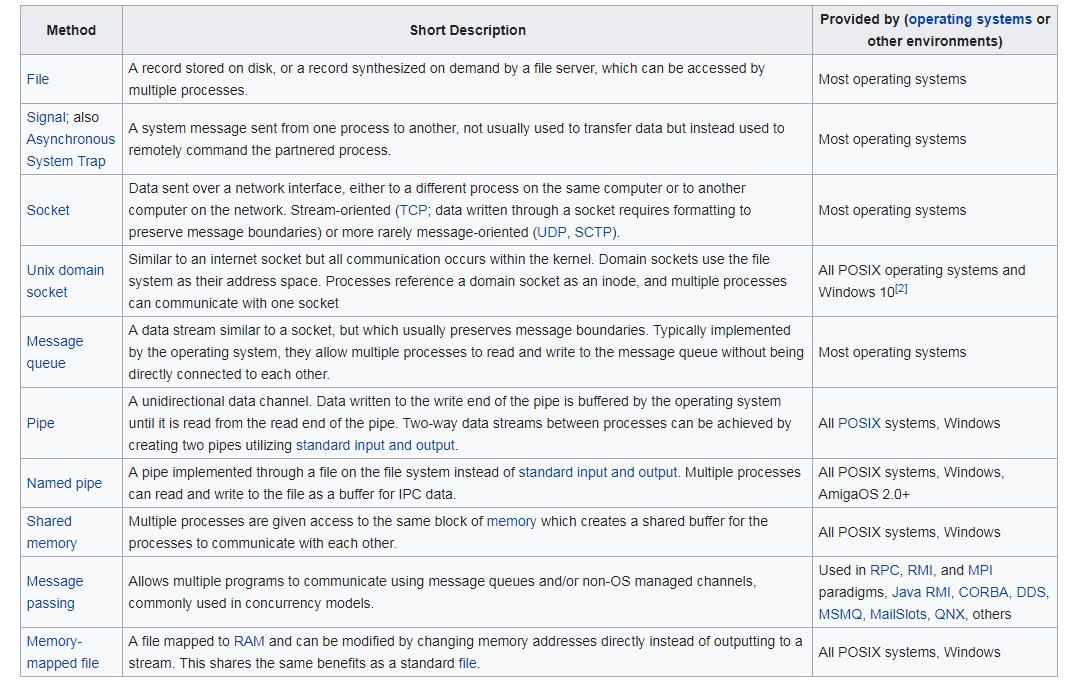
3-Signaling

**Inter Processes Communication**

In [computer science](https://en.wikipedia.org/wiki/Computer_science), **inter-process communication** or **interprocess communication** (**IPC**) refers specifically to the mechanisms an [operating system](https://en.wikipedia.org/wiki/Operating_system) provides to allow the [processes](https://en.wikipedia.org/wiki/Process_(computing)) to manage shared data. Typically, applications can use IPC, categorized as [clients and servers](https://en.wikipedia.org/wiki/Client-server_model), where the client requests data and the server responds to client requests.

Many applications are both clients and servers, as commonly seen in [distributed computing](https://en.wikipedia.org/wiki/Distributed_computing). Methods for doing IPC are divided into categories which vary based on [software requirements](https://en.wikipedia.org/wiki/Software_requirements), such as [performance](https://en.wikipedia.org/wiki/Algorithmic_efficiency) requirements, and system circumstances, such as [network bandwidth](https://en.wikipedia.org/wiki/Bandwidth_(computing)) (Maximum Rate of Data Transfer across a given path).

Approches



**Inter Processes Communication In Unix/Linux**

Processes communicate with each other and with the kernel to coordinate their activities. Linux supports a number of Inter-Process Communication (IPC) mechanisms. Signals and pipes are two of them.

**Signals**

Signals are one of the oldest inter-process communication methods used by Unix TM systems. They are used to signal asynchronous events to one or more processes. A signal could be generated by a keyboard interrupt or an error condition such as the process attempting to access a non-existent location in its virtual memory. Signals are also used by the shells to signal job control commands to their child processes.

There are a set of defined signals that the kernel can generate or that can be generated by other processes in the system, provided that they have the correct privileges. You can list a system's set of signals using the kill command (kill -l), on my Intel Linux box this gives:

1) SIGHUP 2) SIGINT 3) SIGQUIT 4) SIGILL

5) SIGTRAP 6) SIGIOT 7) SIGBUS 8) SIGFPE

9) SIGKILL 10) SIGUSR1 11) SIGSEGV 12) SIGUSR2

13) SIGPIPE 14) SIGALRM 15) SIGTERM 17) SIGCHLD

18) SIGCONT 19) SIGSTOP 20) SIGTSTP 21) SIGTTIN

22) SIGTTOU 23) SIGURG 24) SIGXCPU 25) SIGXFSZ

26) SIGVTALRM 27) SIGPROF 28) SIGWINCH 29) SIGIO

30) SIGPWR

## Pipes

In Linux, a pipe is implemented using two file data structures which both point at the same temporary.

## Sockets

A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a **port** number so that the **TCP** layer can identify the application that data is destined to be sent to. An endpoint is a combination of an IP address and a **port** number.

**FIFO**

**FIFO operating system** scheduling algorithm, which gives every process central processing unit (CPU) time in the order in which it is demanded. **FIFO's** opposite is LIFO, last-in-first-out, where the youngest entry or 'top of the stack' is processed first.

**QUEUE**

In computer science, an input **queue** is a collection of processes in storage that are waiting to be brought into memory to run a program. Input **queues** are mainly used in **Operating System** Scheduling which is a technique for distributing resources among processes.

**MKFIFO**

**mkfifo**() makes a FIFO special file with name *pathname*. *mode* specifies the FIFO's permissions. A FIFO special file is similar to a pipe, except that it is created in a different way. Instead of being an anonymous communications channel, a FIFO special file is entered into the file system by calling **mkfifo.**

**THREADS**

**THREAD :** A **thread** is a single sequence stream within in a process. Because **threads** have some of the properties of processes, they are sometimes called lightweight processes. In a process, **threads** allow multiple executions of streams. ... An **operating system** that has **thread** facility, the basic unit of CPU utilization is a **thread**.

A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack. 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.Atraditional (or ***heavyweight***) process has a single thread of control. If a process has multiple threads of control, it

can perform more than one task at a time. between a traditional **single-threaded** process and a **multithreaded** process.

**Benefits**

**1. 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. This quality is especially useful in designing user interfaces. For instance, consider what happens when a user clicks a button that results in the performance of a time-consuming operation. A single-threaded application would be unresponsive to the user until the operation had completed. In contrast, if the time-consuming operation is performed in a separate thread, the application remains responsive to the user.

**2. Resource sharing**. Processes canonly share resources through techniques such as shared memory and message passing. Such techniques must be explicitly arranged by the programmer. However, threads share the memory and the resources of the process to which they belong by default. 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.

**3. Economy**. Allocatingmemoryandresources forprocess creationis costly. Because threads share the resources of the process to which they belong, it is more economical to create and context-switch threads. Empirically gauging the difference in overhead can be difficult, but in general it is

significantly more time consuming to create and manage processes than threads. In Solaris, for example, creating a process is about thirty times

**4.Scalability.** The benefits of multithreading can be even greater in a multiprocessor architecture, where threads may be running in parallel on different processing cores. A single-threaded process can run on only one processor, regardless how many are available.

**STATES**

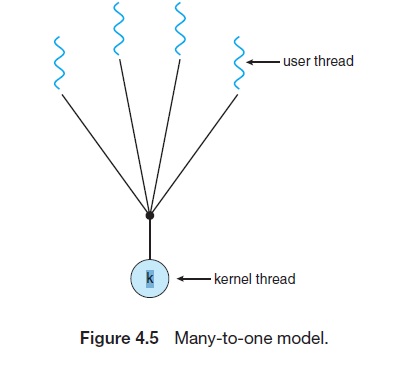
In an operating system, a process may have any number of threads that perform a particular task. A thread state diagram highlights different states of a thread: new, **runnable**, **blocked**, and **terminated**

Multithreading Models

In general, user-level threads can be implemented using one of four models.

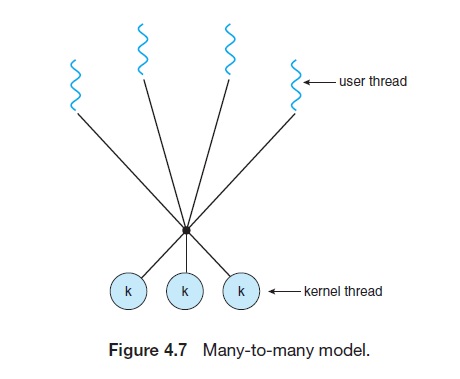
* Many-to-one
* One-to-one
* Many-to-many

All models maps user-level threads to kernel-level threads. A **kernel thread** is similar to a process in a non-threaded (single-threaded) system. The kernel thread is the unit of execution that is scheduled by the kernel to execute on the CPU. The term **virtual processor** is often used instead of kernel thread.

**Many-to-One Model** : In the many-to-one model all user level threads execute on the same kernel thread. The process can only run one user-level thread at a time because there is only one kernel-level thread associated with the process.

### **One-to-one**

In the one-to-one model every user-level thread execute on a separate kernel-level thread.

**Many-to-Many Model**In the many-to-many model the process is allocated m number of kernel-level threads to execute n number of user-level thread 

**USER LEVEL AND KERNAL LEVEL THREADS**

In general, **user**-**level threads** can be implemented using one of four models. All models maps **user**-**level threads** to **kernel**-**level threads**. A **kernel thread** is similar to a process in a non-threaded (single-threaded) system. The **kernel thread** is the unit of execution that is scheduled by the **kernel** to execute on the CPU.

## Kernel level threads

Kernel level threads are supported and managed directly by the operating system.

* The kernel knows about and manages all threads.
* One process control block (PCP) per process.
* One thread control block (TCB) per thread in the system.
* Provide system calls to create and manage threads from user space.

**Advantages**

* The kernel has full knowledge of all threads.
* Scheduler may decide to give more CPU time to a process having a large number of threads.
* Good for applications that frequently block.

**Disadvantages**

* Kernel manage and schedule all threads.
* Significant overhead and increase in kernel complexity.
* Kernel level threads are slow and inefficient compared to user level threads.
* Thread operations are hundreds of times slower compared to user-level threads.

**User level threads**

User level threads are supported above the kernel in user space and are managed without kernel support.

* Threads managed entirely by the run-time system (user-level library).
* Ideally, thread operations should be as fast as a function call.
* The kernel knows nothing about user-level threads and manage them as if they where single-threaded processes.

**Advantages**

* Can be implemented on an OS that does not suport kernel-level threads.
* Does not require modifications of the OS.
* Simple representation: PC, registers, stack and small thread control block all stored in the user-level process address space.
* Simple management: Creating, switching and synchronizing threads done in user-space without kernel intervention.
* Fast and efficient: switching threads not much more expensive than a function call.

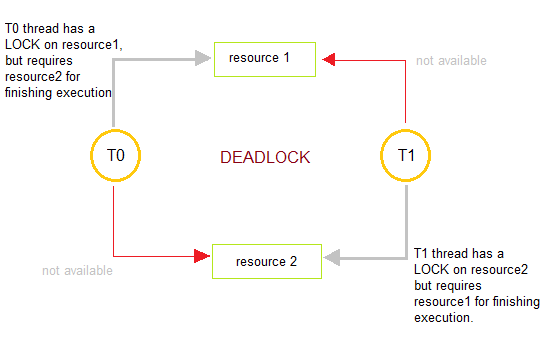
**Disadvantages**

* Not a perfect solution (a trade off).
* Lack of coordination between the user-level thread manager and the kernel.
* OS may make poor decisions like:
  + scheduling a process with idle threads
  + blocking a process due to a blocking thread even though the process has other threads that can run
  + giving a process as a whole one time slice irrespective of whether the process has 1 or 1000 threads
  + unschedule a process with a thread holding a lock.
* May require communication between the kernel and the user-level thread manager (scheduler activations) to overcome the above problems.

**Deadlocks**

In a multiprogramming environment, several processes may compete for a ﬁnite number of resources. A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock

Deadlocks are a set of blocked processes each holding a resource and waiting to acquire a resource held by another process.



**How to avoid Deadlocks**

Deadlocks can be avoided by avoiding at least one of the four conditions, because all this four conditions are required simultaneously to cause deadlock.

1. **Mutual Exclusion**

Resources shared such as read-only files do not lead to deadlocks but resources, such as printers and tape drives, requires exclusive access by a single process.

1. **Hold and Wait**

In this condition processes must be prevented from holding one or more resources while simultaneously waiting for one or more others.

1. **No Preemption (Reaction)**

Preemption of process resource allocations can avoid the condition of deadlocks, where ever possible.

1. **Circular Wait**

Circular wait can be avoided if we number all resources, and require that processes request resources only in strictly increasing (or decreasing) order.

**Under the normal mode of operation, a process may utilize a resource in only the following sequence:**

1. **Request**. The process requests the resource. If the request cannot be grantedimmediately (forexample,iftheresourceisbeingusedbyanother process), then the requesting process must wait until it can acquire the resource.

2. **Use**. Theprocesscanoperateontheirsource (forexample, iftheresource isaprinter, the processcanprintontheprinter).

3. **Release**. The processreleasesthe resource.

## What is a Livelock?

There is a variant of deadlock called livelock. This is a situation in which two or more processes continuously change their state in response to changes in the other process (es) without doing any useful work.

**A human example** of live lock would be two people who meet face-to-face in a corridor and each moves aside to let the other pass, but they end up swaying from side to side without making any progress because they always move the same way at the same time.

**DEADLOCK AVOIDENCE**

In order to avoid **deadlocks**, the process must tell **OS**, the maximum number of resources a process can request to complete its execution.

**Safe and Unsafe States**

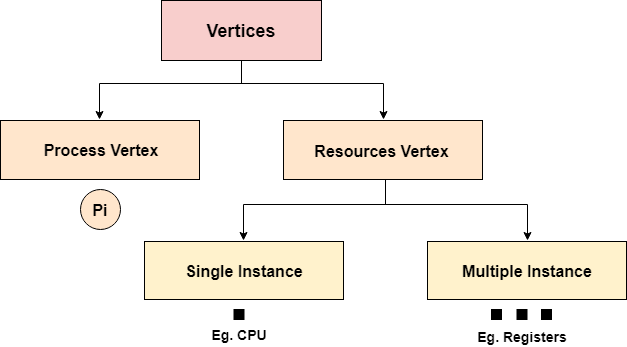
The resource allocation state of a system can be defined by the instances of available and allocated resources, and the maximum instance of the resources demanded by the processes.

# **Resource Allocation Graph**

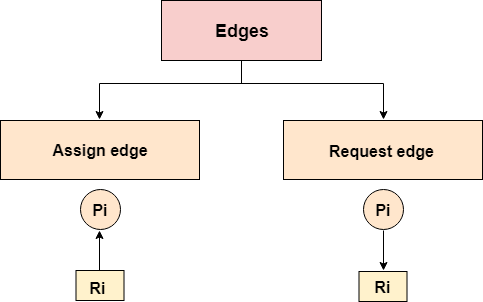
The resource allocation graph is the pictorial representation of the state of a system. As its name suggests, the resource allocation graph is the complete information about all the processes which are holding some resources or waiting for some resources.

It also contains the information about all the instances of all the resources whether they are available or being used by the processes.

In Resource allocation graph, the process is represented by a Circle while the Resource is represented by a rectangle. Let's see the types of vertices and edges in detail.

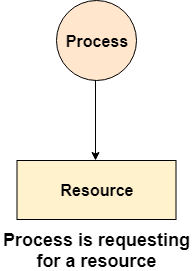
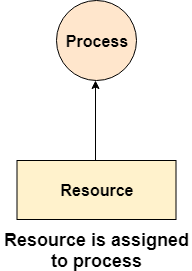


Vertices are mainly of two types, Resource and process. Each of them will be represented by a different shape. Circle represents process while rectangle represents resource.



A resource is shown as assigned to a process if the tail of the arrow is attached to an instance to the resource and the head is attached to a process.

A process is shown as waiting for a resource if the tail of an arrow is attached to the process while the head is pointing towards the resource.

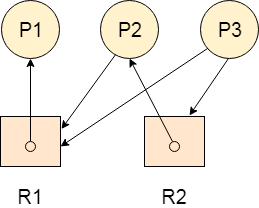
 

**Example**

Let's consider 3 processes P1, P2 and P3, and two types of resources R1 and R2. The resources are having 1 instance each.

According to the graph, R1 is being used by P1, P2 is holding R2 and waiting for R1, P3 is waiting for R1 as well as R2.

The graph is deadlock free since no cycle is being formed in the graph.



# **Banker's algorithm**

The **Banker algorithm**, sometimes referred to as the **detection algorithm**, is a [resource allocation](https://en.wikipedia.org/wiki/Resource_allocation) and [deadlock](https://en.wikipedia.org/wiki/Deadlock) avoidance [algorithm](https://en.wikipedia.org/wiki/Algorithm) developed by [Edsger Dijkstra](https://en.wikipedia.org/wiki/Edsger_Dijkstra" \o "Edsger Dijkstra) that tests for safety by simulating the allocation of predetermined maximum possible amounts of all [resources](https://en.wikipedia.org/wiki/Resource_(computer_science)), and then makes an "s-state" check to test for possible deadlock conditions for all other pending activities, before deciding whether allocation should be allowed to continue.

The algorithm was developed in the design process for the [THE](https://en.wikipedia.org/wiki/THE_(operating_system)) [operating system](https://en.wikipedia.org/wiki/Operating_system) and originally described (in [Dutch](https://en.wikipedia.org/wiki/Dutch_language)) in EWD108. When a new process enters a system, it must declare the maximum number of instances of each resource type that it may ever claim; clearly, that number may not exceed the total number of resources in the system. Also, when a process gets all its requested resources it must return them in a finite amount of time.

## Resources

For the Banker's algorithm to work, it needs to know three things:

* How much of each resource each process could possibly request[MAX]
* How much of each resource each process is currently holding[ALLOCATED]
* How much of each resource the system currently has available[AVAILABLE]

### Example

Total system resources are:

A B C D

6 5 7 6

Available system resources are:

A B C D

3 1 1 2

Processes (currently allocated resources):

A B C D

P1 1 2 2 1

P2 1 0 3 3

P3 1 2 1 0

Processes (maximum resources):

A B C D

P1 3 3 2 2

P2 1 2 3 4

P3 1 3 5 0

Need = maximum resources - currently allocated resources

Processes (possibly needed resources):

A B C D

P1 2 1 0 1

P2 0 2 0 1

P3 0 1 4 0

### Requests

Can the request be granted?

Assume that the request is granted

Is the new state safe?

## Limitations

Like the other algorithms, the Banker's algorithm has some limitations when implemented. Specifically, it needs to know how much of each resource a process could possibly request. In most systems, this information is unavailable, making it impossible to implement the Banker's algorithm

## Deadlock Avoidance

Requires that the system has some additional a priori information available  
  
ν Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need  
ν The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition  
ν Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes

## Methods for Handling Deadlocks

ν Ensure that the system will never enter a deadlock state  
ν Allow the system to enter a deadlock state and then recover  
ν Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX

# **Deadlock Detection**

# ν Allow system to enter deadlock state ν Detection algorithm ν Recovery scheme

# **Recovery from Deadlock:**

### **1. Process Termination**

ν Abort all deadlocked processes  
ν Abort one process at a time until the deadlock cycle is eliminated  
ν In which order should we choose to abort?

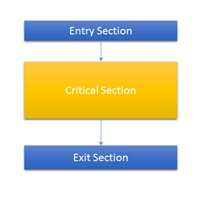
λ Priority of the process  
λ How long process has computed, and how much longer to completion  
λ Resources the process has used  
λ Resources process needs to complete  
λ How many processes will need to be terminated  
λ Is process interactive or batch?

### **2. Resource Preemption**

ν **Selecting a victim** – minimize cost  
ν **Rollback** – return to some safe state, restart process for that state  
ν **Starvation** – same process may always be picked as victim, include number of rollback in cost facto

# **3-Critical Section**

It is the part of program where shared resources are accessed by processes and the resources are used in mutually exclusive fashion.  
It is called critical because the order of execution matters tremendously here.



So as we can see that there is entry section and exit section in the above image, we have to go through first entry section and then Critical Section and then Exit section.

**MEMORY MANAGEMENT (OS)**

**Memory management**

Is the process of controlling and coordinating computer **memory**, assigning portions called blocks to various running programs to optimize overall system performance. **Memory management** resides in hardware, in the OS (operating system), and in programs and applications.

**ADRESS BINDING**

Address binding is the process of mapping from one address space to another address space. Logical address is address generated by CPU during execution whereas **Physical Address** refers to location in memory unit(the one that is loaded into memory).**Note** that user deals with only logical address(Virtual address).

**TECHNIQUES FOR MEMORY MANAGMENTS**

* Process Address Space. The process address space is the set of **logical** addresses that a process references in its code. ...
* **Static** vs **Dynamic Loading**. ...
* **Static** vs **Dynamic** Linking. ...
* Swapping. ...
* Memory Allocation. ...
* Fragmentation. ...
* Paging. ...
* **Segmentation**

**Logical to Physical Adress**

Memory consists of large array of words or arrays, each of which has address associated with it. Now the work of CPU is to fetch instructions from the memory based program counter. Now further these instruction may cause loading or storing to specific memory address.

Address binding is the process of mapping from one address space to another address space. Logical address is address generated by CPU during execution whereas Physical Address refers to location in memory unit(the one that is loaded into memory).Note that user deals with only logical address(Virtual address). The logical address undergoes translation by the MMU or address translation unit in particular. The output of this process is the appropriate physical address or the location of code/data in RAM.

An address binding can be done in three different ways:

**Compile Time –** If you know that during compile time where process will reside in memory then absolute address is generated i.e physical address is embedded to the executable of the program during compilation. Loading the executable as a process in memory is very fast. But if the generated address space is preoccupied by other process, then the program crashes and it becomes necessary to recompile the program to change the address space.

**Load time –** If it is not known at the compile time where process will reside then relocatable address will be generated. Loader translates the relocatable address to absolute address. The base address of the process in main memory is added to all logical addresses by the loader to generate absolute address. In this if the base address of the process changes then we need to reload the process again.

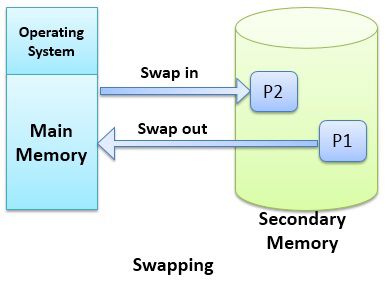
**Execution time-** The instructions are in memory and are being processed by the CPU. Additional memory may be allocated and/or deallocated at this time. This is used if process can be moved from one memory to another during execution(dynamic linking-Linking that is done during load or run time). e.g – Compaction.

**MMU(Memory Management Unit)-**  
The run time mapping between Virtual address and Physical Address is done by hardware device known as MMU.

In memory management, Operating System will handle the processes and moves the processes between disk and memory for execution . It keeps the track of available and used memory.

**Swapping**

To replace pages or segments of data in memory. **Swapping** is a useful technique that enables a computer to execute programs and manipulate data files larger than main memory. The **operating system** copies as much data as possible into main memory, and leaves the rest on the disk.



**Contiguous Memory Allocation**.

The main **memory** must accommodate both the **operating system** and the various user processes. ... When the process terminates, the partition becomes available for another process. The **operating system** keeps a table indicating which parts of **memory** are available and which are occupied.

**MFT & MVT**  
Managed file transfer (**MFT**) is a type of software used to provide secure internal, external and ad-hoc data transfers through a network. **MFT** software and products are designed to help organizations meet the increasing security, compliance and operational demands of moving information.

**MVT** (Multiprogramming with a Variable number of Tasks) is the memory management technique in which each job gets just the amount of memory it needs. That is, the partitioning of memory is dynamic and changes as jobs enter and leave the system. **MVT** is a more ``efficient'' user of resources

**Internal & External Framentation**

Internal fragmentation is the wasted space within each allocated block because of rounding up from the actual requested allocation to the allocation granularity.

External fragmentation is the various free spaced holes that are generated in either your memory or disk space.

**Pagging**

In computer **operating systems**, **paging** is a memory management scheme by which a computer stores and retrieves data from secondary storage for use in main memory. In this scheme, the **operating system** retrieves data from secondary storage in same-size blocks called pages.

## Fragmentation

Fragmentation occurs in a dynamic memory allocation system when most of the free blocks are too small to satisfy any request. It is generally termed as inability to use the available memory.

In such situation processes are loaded and removed from the memory. As a result of this, free holes exists to satisfy a request but is non contiguous i.e. the memory is fragmented into large no. Of small holes. This phenomenon is known as **External Fragmentation.**

Also, at times the physical memory is broken into fixed size blocks and memory is allocated in unit of block sizes. The memory allocated to a space may be slightly larger than the requested memory. The difference between allocated and required memory is known as **Internal fragmentation** i.e. the memory that is internal to a partition but is of no use.

## Paging

A solution to fragmentation problem is Paging. Paging is a memory management mechanism that allows the physical address space of a process to be non-contagious. Here physical memory is divided into blocks of equal size called **Pages**. The pages belonging to a certain process are loaded into available memory frames.

### Page Table

A Page Table is the data structure used by a virtual memory system in a computer operating system to store the mapping between *virtual address* and *physical addresses.*

Virtual address is also known as Logical address and is generated by the CPU. While Physical address is the address that actually exists on memory.

# What is Virtual Memory?

Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as large virtual memory is provided for user programs when a very small physical memory is there.

In real scenarios, most processes never need all their pages at once, for following reasons :

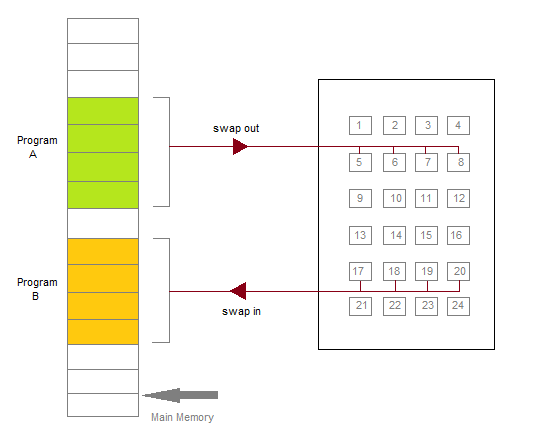
* Error handling code is not needed unless that specific error occurs, some of which are quite rare.
* Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
* Certain features of certain programs are rarely used.

### Benefits of having Virtual Memory

1. Large programs can be written, as virtual space available is huge compared to physical memory.
2. Less I/O required, leads to faster and easy swapping of processes.
3. More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.

## What is Demand Paging?

The basic idea behind demand paging is that when a process is swapped in, its pages are not swapped in all at once. Rather they are swapped in only when the process needs them(On demand). This is termed as lazy swapper, although a pager is a more accurate term.



Initially only those pages are loaded which will be required the process immediately.