**Process Management in OS**

**<https://www.youtube.com/playlist?list=PLxCzCOWd7aiGz9donHRrE9I3Mwn6XdP8p> - Complete OS**

A Program does nothing unless its instructions are executed by a CPU. A program in execution is called a process. In order to accomplish its task, process needs the computer resources.

There may exist more than one process in the system which may require the same resource at the same time. Therefore, the operating system has to manage all the processes and the resources in a convenient and efficient way.

Some resources may need to be executed by one process at one time to maintain the consistency otherwise the system can become inconsistent and deadlock may occur.

The operating system is responsible for the following activities in connection with Process Management

1. Scheduling processes and threads on the CPU.
2. Creating and deleting both user and system processes.
3. Suspending and resuming processes.
4. Providing mechanisms for process synchronization.
5. Providing mechanisms for process communication.

# ◼ process is comprised of:

# 1. Program code (possibly shared)

# 2. A set of data

# 3. A number of attributes describing the state of the process

# Attributes of a process

The Attributes of the process are used by the Operating System to create the process control block (PCB) for each of them. This is also called context of the process. Attributes which are stored in the PCB are described below.

### 1. Process ID

When a process is created, a unique id is assigned to the process which is used for unique identification of the process in the system.

### 2. Program counter

A program counter stores the address of the last instruction of the process on which the process was suspended. The CPU uses this address when the execution of this process is resumed.

### 3. Process State

The Process, from its creation to the completion, goes through various states which are new, ready, running and waiting. We will discuss about them later in detail.

### 4. Priority

Every process has its own priority. The process with the highest priority among the processes gets the CPU first. This is also stored on the process control block.

### 5. General Purpose Registers

Every process has its own set of registers which are used to hold the data which is generated during the execution of the process.

### 6. List of open files

During the Execution, Every process uses some files which need to be present in the main memory. OS also maintains a list of open files in the PCB.

### 7. List of open devices

OS also maintain the list of all open devices which are used during the execution of the process.

**Process Control Block**

▪ Contains the process elements

▪ Created and manage by the operating system

▪ The PCB is constructed at process creation.

▪ PCB includes a pointer to be used in lists (queues) of PCBs. Allows support for multiple processes

▪ The PCB is used to save information about a process

when switched out of CPU.

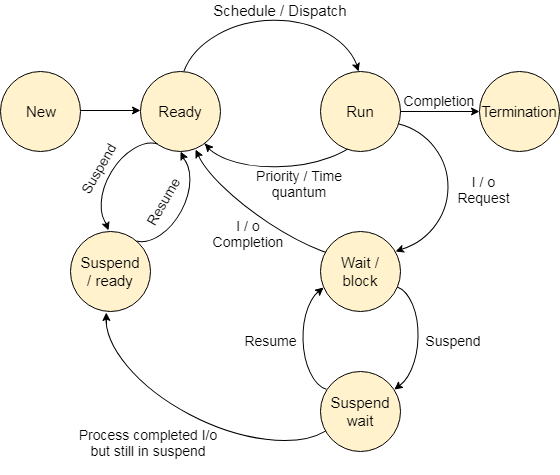
Dispatcher is a small program

which switches the processor from

one process to another.

# Process States

****State Diagram****



The process, from its creation to completion, passes through various states. The minimum number of states is five. The names of the states are not standardized although the process may be in one of the following states during execution.

### 1. New

A program which is going to be picked up by the OS into the main memory is called a new process.

### 2. Ready

Whenever a process is created, it directly enters in the ready state, in which, it waits for the CPU to be assigned. The OS picks the new processes from the secondary memory and put all of them in the main memory.

The processes which are ready for the execution and reside in the main memory are called ready state processes. There can be many processes present in the ready state.

### 3. Running

One of the processes from the ready state will be chosen by the OS depending upon the scheduling algorithm. Hence, if we have only one CPU in our system, the number of running processes for a particular time will always be one. If we have n processors in the system then we can have n processes running simultaneously.

### 4. Block or wait

From the Running state, a process can make the transition to the block or wait state depending upon the scheduling algorithm or the intrinsic behavior of the process.

When a process waits for a certain resource to be assigned or for the input from the user then the OS move this process to the block or wait state and assigns the CPU to the other processes.

### 5. Completion or termination

When a process finishes its execution, it comes in the termination state. All the context of the process (Process Control Block) will also be deleted the process will be terminated by the Operating system.

### 6. Suspend ready

A process in the ready state, which is moved to secondary memory from the main memory due to lack of the resources (mainly primary memory) is called in the suspend ready state.

If the main memory is full and a higher priority process comes for the execution then the OS have to make the room for the process in the main memory by throwing the lower priority process out into the secondary memory. The suspend ready processes remain in the secondary memory until the main memory gets available.

### 7. Suspend wait

Instead of removing the process from the ready queue, it's better to remove the blocked process which is waiting for some resources in the main memory. Since it is already waiting for some resource to get available hence it is better if it waits in the secondary memory and make room for the higher priority process. These processes complete their execution once the main memory gets available and their wait is finished.

**Operating system control tables**

•OS has to manage processes and resources, it must have

information about the current status of each process and

resource.

•Tables are constructed for each entity, the operating

system manages.

**1. Memory tables:** are used to keep track of both main memory and

secondary memory.

- Part of main memory is reserved for use by the operating

system;

The memory tables generally include the following information:

– the allocation of main memory to processes

– the allocation of secondary memory to processes

– protection attributes of blocks of main or virtual memory

**2. I/O tables:** are used by the operating system to manage I/O

devices. They should record:

– the availability of each particular device

– the status of I/O operations relating to each device

- the location in main memory being used as the source or

destination of the I/O transfer

**3. File tables:** provides information about

– Existence of files

-Location on secondary memory

-Current Status

-Attributes

-Sometimes this information is maintained by a file management system

**4. Process table :**

- To manage processes the OS needs to know details of the processes Current state

– Process ID

– Location in memory

-- Process control block

**Change process state:**

• If the currently running process is to be moved to another state (Ready,

Blocked, etc.), then the OS must make substantial changes in its

environment.

◼Steps involved in process switching:

1.Save the context of the processor including program counter and other

register.

2.Update the PCB of the process that is currently in the Running state

3.Move PCB to appropriate queue – ready; blocked; ready/suspend

4.Select another process for execution

5.Update PCB of selected process

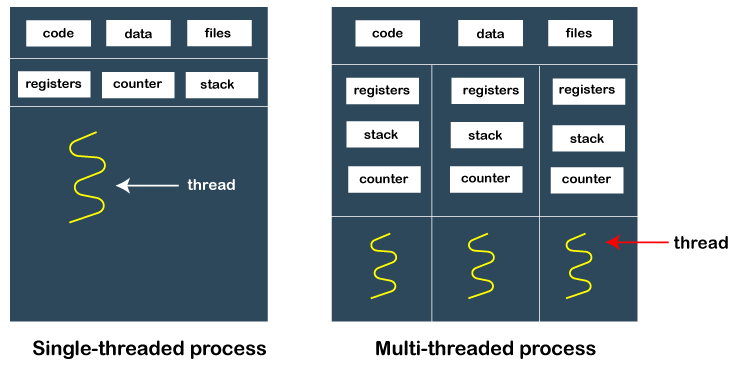
6.Update memory management data structure

7.Restore context of the processor

Processes and Threads

A thread is a unit of execution within a process or it’s a smallest unit of a process. A process may have from one thread to many.

All the threads within one process are interrelated to each other. Threads have some common information, such as ****data segment, code segment, files, etc.,**** that is shared to their peer threads. But contains its own registers, stack, and counter.



### How does thread work?

As we have discussed that a thread is a subprocess or an execution unit within a process. A process can contain a single thread to multiple threads. A thread works as follows:

* When a process starts, OS assigns the memory and resources to it. Each thread within a process shares the memory and resources of that process only.
* Threads are mainly used to improve the processing of an application. In reality, only a single thread is executed at a time, but due to fast context switching between threads gives an illusion that threads are running parallelly.
* If a single thread executes in a process, it is known as ****a single-threaded**** And if multiple threads execute simultaneously, then it is known as ****multithreading.****

### Types of Threads

There are two types of threads, which are:

****1. User Level Thread****

As the name suggests, the user-level threads are only managed by users, and the kernel does not have its information. These are faster, easy to create and manage. The kernel takes all these threads as a single process and handles them as one process only. The user-level threads are implemented by user-level libraries, not by the system calls.

****2. Kernel-Level Thread****

The kernel-level threads are handled by the Operating system and managed by its kernel. These threads are slower than user-level threads because context information is managed by the kernel. To create and implement a kernel-level thread, we need to make a system call.

**Benefits of threads**

1. Since threads within the same process share memory and files, they

can communicate with each other without invoking the kernel.

2. It takes less time to create new thread in an existing process than to

create brand new process.

3. It takes less time to terminate a thread than a process.

4. It takes less time to switch between two thread with in the same

process.

5. Threads enhance efficiency in communication between different

executing programs.

6. Utilization of multiple processor architecture, where each thread may

be running in parallel on a different processor.

**Disadvantages of Threads**

◼ Suspending a process involves suspending all threads of

the process since all threads share the same address

space.

◼ Termination of a process, terminates all threads within

the process.

**Thread Execution States**

1. Spawn (another thread)

2. Block

- Issue: will blocking a thread block other, or all,

threads

3. Unblock

4. Finish (thread)

• Deallocate register context and stacks

Multithreading Models

◼ Many-to-One

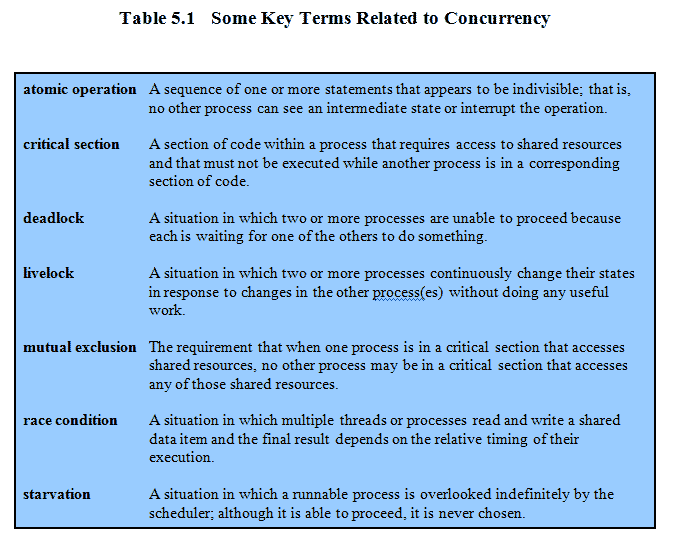
◼ One-to-One

◼ Many-to-Many

## What is Concurrency?

It refers to the execution of multiple instruction sequences at the same time. It occurs in an operating system when multiple process threads are executing concurrently. These threads can interact with one another via shared memory or message passing. Concurrency results in resource sharing, which causes issues like deadlocks and resource scarcity. It aids with techniques such as process coordination, memory allocation, and execution schedule to maximize throughput.

## Issues of Concurrency



# Mutual Exclusion In OS

During concurrent execution of processes, processes need to enter the [critical section](https://www.geeksforgeeks.org/g-fact-70/) (or the section of the program shared across processes) at times for execution. It might so happen that because of the execution of multiple processes at once, the values stored in the critical section become inconsistent. In other words, the values depend on the sequence of execution of instructions – also known as a [race condition](https://practice.geeksforgeeks.org/problems/what-is-race-condition). The primary task of process synchronization is to get rid of race conditions while executing the critical section.   
This is primarily achieved through [mutual exclusion](https://practice.geeksforgeeks.org/problems/what-is-mutual-exclusion).

Mutual exclusion is a property of [process synchronization](https://www.geeksforgeeks.org/introduction-of-process-synchronization/) which states that “no two processes can exist in the critical section at any given point of time”. The term was first coined by Dijkstra. Any process synchronization technique being used must satisfy the property of mutual exclusion, without which it would not be possible to get rid of a race condition.

## Necessary Conditions for Mutual Exclusion

****There are four conditions applied to mutual exclusion, which are mentioned below :****

* Mutual exclusion should be ensured in the middle of different processes when accessing shared resources. There must not be two processes within their critical sections at any time.
* Assumptions should not be made as to the respective speed of the unstable processes.
* The process that is outside the critical section must not interfere with another for access to the critical section.
* When multiple processes access its critical section, they must be allowed access in a finite time, i.e. they should never be kept waiting in a loop that has no limits.

**Semaphores in Process Synchronization :**

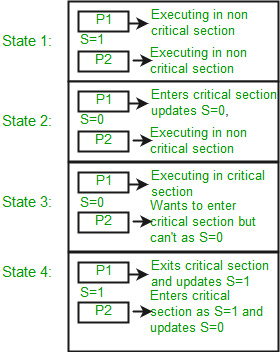
Semaphore was proposed by Dijkstra in 1965 which is a very significant technique to manage concurrent processes by using a simple integer value, which is known as a semaphore. A semaphore is simply an integer variable that is shared between threads. This variable is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment.   
Semaphores are of two types:

**Binary Semaphore –**  
This is also known as mutex lock. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problems with multiple processes.

**Counting Semaphore –**  
Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

Now, let us see how it implements mutual exclusion. Let there be two processes P1 and P2 and a semaphore s is initialized as 1. Now if suppose P1 enters in its critical section then the value of semaphore s becomes 0. Now if P2 wants to enter its critical section then it will wait until s > 0, this can only happen when P1 finishes its critical section and calls V operation on semaphore s.

This way mutual exclusion is achieved. Look at the below image for details which is Binary semaphore.



# Mutex

The word "mutex" stands for an object providing MUTual EXclusion between threads. Mutex ensures that only one thread has access to a critical section or data by using operations like a lock and unlock. A thread having the lock of mutex can use the critical section while other threads must wait till the lock is released.

**Misconception:**

There is an ambiguity between binary semaphore and mutex. We might have come across that a mutex is a binary semaphore. But it is not! The purpose of mutex and semaphore are different. Maybe, due to similarity in their implementation a mutex would be referred to as a binary semaphore.

**UNIT - 3 : Uniprocessor Scheduling:**

The 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 Multi programming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

There are three types of process scheduler. 

**Long Term or job scheduler :**   
It brings the new process to the ‘Ready State’. It controls Degree of Multi-programming, i.e., number of process present in ready state at any point of time. It is important that the long-term scheduler make a careful selection of both I/O and CPU-bound processes. I/O bound tasks are which use much of their time in input and output operations while CPU bound processes are which spend their time on CPU. The job scheduler increases efficiency by maintaining a balance between the two.

**Short term or CPU scheduler :**  
It is responsible for selecting one process from ready state for scheduling it on the running state. Note: Short-term scheduler only selects the process to schedule it doesn’t load the process on running.  Here is when all the scheduling algorithms are used. The CPU scheduler is responsible for ensuring there is no starvation owing to high burst time processes.  
Dispatcher is responsible for loading the process selected by Short-term scheduler on the CPU (Ready to Running State) Context switching is done by dispatcher only. A dispatcher does the following:

Switching context.

Switching to user mode.

Jumping to the proper location in the newly loaded program.

**Medium-term scheduler :**  
It is responsible for suspending and resuming the process. It mainly does swapping (moving processes from main memory to disk and vice versa). Swapping may be necessary to improve the process mix or because a change in memory requirements has over committed available memory, requiring memory to be freed up. It is helpful in maintaining a perfect balance between the I/O bound and the CPU bound. It reduces the degree of multi programming.

Scheduling Algorithm Optimization Criteria

◼ Max CPU utilization

◼ Max throughput

◼ Min turnaround time

◼ Min waiting time

◼ Min response time

# UNIT 4 : Deadlock in Operating System

A deadlock in OS is a situation in which more than one process is blocked because it is holding a resource and also requires some resource that is acquired by some other process. The four necessary conditions for a deadlock situation to occur are mutual exclusion, hold and wait, no preemption and circular set. We can prevent a deadlock by preventing any one of these conditions. There are different ways to detect and recover a system from deadlock.

## Neccessary Conditions for Deadlock

The four necessary conditions for a deadlock to arise are as follows.

* ****Mutual Exclusion****: Only one process can use a resource at any given time i.e. the resources are non-sharable.
* ****Hold and wait****: A process is holding at least one resource at a time and is waiting to acquire other resources held by some other process.
* ****No preemption****: The resource can be released by a process voluntarily i.e. after execution of the process.
* ****Circular Wait****: A set of processes are waiting for each other in a circular fashion. For example, lets say there are a set of processes {P\_0*P*0​,P\_1*P*1​,P\_2*P*2​,P\_3*P*3​} such that P\_0*P*0​ depends on P\_1*P*1​, P\_1*P*1​ depends on P\_2*P*2​, P\_2*P*2​ depends on P\_3*P*3​ and P\_3*P*3​ depends on P\_0*P*0​. This creates a circular relation between all these processes and they have to wait forever to be executed.

**Dealing with deadlocks**

1. Deadlock prevention – by adopting a policy that

eliminates one of the four conditions

2.Deadlock avoidance – by making the appropriate

dynamic choices based on the current state of

resource allocation

3.Deadlock detection and recovery – attempt to detect

presence of deadlock and take actions to recover

4.Ignore the problem and pretend that deadlocks never

occur in the system.

Deadlock prevention

Design a system in such a way that the possibility of

deadlock is excluded by ensuring that at least one of

the necessary conditions cannot hold

• Two main methods

1. Indirect – prevent all three of the necessary conditions

occurring at once (Mutual exclusion, Hold-wait,No

preemption)

1. Direct – prevent Circular wait

**Deny Mutual Exclusion**

◼ **Mutual Exclusion** – not required for sharable resources;

➢ must hold for non-sharable resources.

➢A process never needs to wait for a sharable resource

➢But in reality, some resources are intrinsically non

sharable and hence we cannot prevent deadlock by

denying mutual exclusion

➢If access to a resource requires mutual exclusion, then

mutual exclusion must be supported by the OS.

➢ Some resources, such as files, may allow multiple

accesses for reads but only exclusive access for writes.

➢Even in this case, deadlock can occur if more than one

process requires write permission.

**Disable hold & wait**

• Must guarantee that whenever a process requests a resource, it

does not hold any other resources

– Requires each process to request and be allocated all its

resources before it begins execution

– Allow process to request resources only when the process has

none

Disadvantages:

– Low resource utilization

– Starvation is possible

– Process may not know in advance all of the resources that it will

require.

**Enable Pre-emption**

• If a process that is holding some resources and requests

another resource that cannot be immediately allocated to it,

then all resources currently being held are released.

• Preempted resources are added to the list of resources for

which the process is waiting.

• Process will be restarted only when it can regain its old

resources, as well as the new ones that it is requesting

**Disable Circular Wait**

◼ Impose a total ordering of all resource types,.

◼ Each process requests resources in an increasing order of

enumeration.

◼ Lets say tape drive is 1, disk drive is 5 and printer is 12

◼ A process can initially request any number of instances of a

resource type Ri

◼ After that process can request instances of resource type Rj

only if f(j)>f (i)

◼ Alternatively, before requesting Rj, release all Ri such that f(i)

>= f(j)

◼ When several instances of same type are needed, a single

request for all of them must be issued

◼ Re-ordering of resources requires re-programming

◼ Ordering should be as per usage pattern of resources.

**Deadlock Avoidance**

◼ Deadlock prevention leads to inefficient use of resources &

execution of processes.

◼ Requires that OS be given in advance additional

information concerning which resources a process will

request and use during its lifetime

◼ Based on this info, OS decides whether to grant the request

or delay it

◼ System must consider resources currently available,

resources currently allocated, future requests and releases

◼ With this knowledge of complete sequence of requests and

releases for each process, system can decide for each

request whether or not the process should wait in order to

avoid a future deadlock

◼ Variations in algorithms differ in amount and type of

information required

◼ Simplest: each process declares the max number of

resources of each type that it may need

◼ Dynamically examines the resource allocation state to ensure

that a circular wait condition can never exist

◼ State: number of available and allocated resources, maximum

demands of processes

Safe state is not a deadlocked

state

• Deadlocked state is an unsafe

state

• Not all unsafe states are

deadlocks

• Unsafe state may lead to a

deadlock

• Better to always be in safe

state

**Avoidance algorithms**

• When single instance of all resource types use RAG

algorithm

– Uses a variant of the RAG we saw earlier

• When multiple instances of resource types,

- Use Banker’s algorithm

**1.Resource-Allocation Graph Algorithm**

Time complexity: For detecting a cycle in a graph it requires an

order of **n2** operations, Where n in number of processes in the

system.

1. **Banker’s Algorithm**

Multiple instances of each resource type.

◼ Each process must a prior claim maximum use.

◼ Every process declares it maximum need/requirement.

◼ Maximum requirement should not exceed the total number of

resources in the system.

◼ When a process requests a resource, system determines whether

allocation will keep the system in safe state or not.

◼ If it is, resources get allocated. Otherwise need to wait until

resources get available.

◼ When a process gets all its resources it must return them in a finite

amount of time

Watch gate smashers for numericals

Deadlock Detection

• When neither prevention nor avoidance is employed, a deadlock situation may arise.

• System can provide an algorithm to detect an algorithm and an algorithm to recover from it if it has occurred.

Use RAG and Bankers algo for detection and wait -for graph.

**Deadlock Recovery**

• Once detected, several alternatives are available for recovery

• Inform the operator that a deadlock has occurred and to let the operator deal with

the deadlock manually

• Abort one or more processes to break the circular wait

• Preempt some resources from one or more of the deadlocked processes

Process Termination / Abort

• Two alternatives

• In both, system reclaims all resources held by the terminated process

• **Abort all deadlocked processes**

– Everything the processes had done so far has gone down the

drain!

• **Abort one process at a time until the deadlock cycle is**

**eliminated**

– Whose turn is next?

• Policy decision similar to scheduling decisions

– Considerable overhead of detecting deadlock after each termination!

• Aborting comes with several issues

Resource Preemption

• We successively preempt some resources

from processes and give these resources to

other processes until the deadlock cycle is

broken

• Three issues need to be addressed:

– Selecting a victim

– Rollback

– Starvation

# UNIT 5 :Memory Management

The task of subdivision is carried out dynamically by OS is known as

**Memory Management.**

◼ Memory needs to be allocated to ensure a reasonable supply of ready

processes to consume available processor time.

◼ Memory Management, involves swapping blocks of data from secondary

storage

**1. External Memory or Secondary Memory –**

Comprising of Magnetic Disk, Optical Disk, Magnetic Tape i.e. peripheral

storage devices which are accessible by the processor via I/O Module.

**2. Internal Memory or Primary Memory –**

Comprising of Main Memory, Cache Memory & CPU registers. This is

directly accessible by the processor.

Memory management requirements

1. Relocation

2. Protection

3. Sharing

4. Logical organization

5. Physical organization

Frame : ***Fixed***-length block of main memory.

Page : ***Fixed***-length block of data in secondary memory (e.g. on disk).

Segment : ***Variable-length*** block of data that resides in secondary memory.

Swapping

• Benefits:

1. Allows higher degree of multi programming

2. Allow dynamic relocation

3. Better memory utilization

4. Less wastage of CPU time on compaction

5. Can easily be applied on priority based scheduling algorithms to

improve performance

Limitations:

1. Entire program must be resident in store when it is executing

2. Processes with changing memory requirements will need to issue

system calls for requesting & releasing memory.

Principal operation of MM is to bring process from secondary memory to

main memory

❑ This typically involves Virtual Memory which in turn involves segmentation

and/or paging

❑ Prior to Virtual Memory, other simpler techniques were used

❑ Partitioning

▪ Fixed

▪ Variable / Dynamic

❑ Concept of paging

❑ Concept of segmentation

**Contiguous Memory Partitioning**

Fragmentation

◼ **External Fragmentation** – Total memory space exists to satisfy a request,

but it is not contiguous.

◼ **Internal Fragmentation** – allocated memory may be slightly larger than

requested memory; this size difference in memory is internal to a partition,

but not being used.

◼ Reduce external fragmentation by **compaction**

❑ Shuffle memory contents to place all free memory together in one large

block

❑ Compaction is possible *only* if relocation is dynamic, and is done at

execution time

❑ I/O problem

◼ Latch job in memory while it is involved in I/O

◼ Do I/O only into OS buffers

❑ Partition the available memory into regions with fixed boundaries.

**1. Fixed Partitioning:**

a) Equal-size Partitions

b) Unequal-size Partitions

**2. Variable Partitioning**

# Fixed Partitioning

The earliest and one of the simplest technique which can be used to load more than one processes into the main memory is Fixed partitioning or Contiguous memory allocation.

In this technique, the main memory is divided into partitions of equal or different sizes. The operating system always resides in the first partition while the other partitions can be used to store user processes. The memory is assigned to the processes in contiguous way.

In fixed partitioning,

1. The partitions cannot overlap.
2. A process must be contiguously present in a partition for the execution.

❑ Advantages

▪ Simple, require minimal OS and processing overhead

❑ Disadvantages

▪ Number of partitions specified at system generation time limits the

number of active processes system can support

▪ Internal fragmentation cannot be completely eliminated

• It is always possible to get small jobs which do not utilize partitions

fully

❑ Fixed partitioning is no where to be seen today

Variable Partitioning

◼ Purpose:to overcome the difficulties of fixed partitioning

◼ Partitions are of variable length and number.

◼ Process is allocated exactly as much memory as required.

◼ Eventually get holes in the memory. This is called external fragmentation

Dynamic partitioning tries to overcome the problems caused by fixed partitioning. In this technique, the partition size is not declared initially. It is declared at the time of process loading.

The first partition is reserved for the operating system. The remaining space is divided into parts. The size of each partition will be equal to the size of the process. The partition size varies according to the need of the process so that the internal fragmentation can be avoided.

# Compaction

We got to know that the dynamic partitioning suffers from external fragmentation. However, this can cause some serious problems.

To avoid compaction, we need to change the rule which says that the process can't be stored in the different places in the memory.

We can also use compaction to minimize the probability of external fragmentation. In compaction, all the free partitions are made contiguous and all the loaded partitions are brought together.

By applying this technique, we can store the bigger processes in the memory. The free partitions are merged which can now be allocated according to the needs of new processes. This technique is also called defragmentation.

Problem with compaction?

o Extra overhead in terms of resource utilization & large

response time

o Expensive

o Needs dynamic relocation of processes in memory

**Allocation Strategies**

To overcome on problem of compaction

◼ Operating system must decide which free block to allocate to a process

◼ When more than one choice available, OS must decide cleverly which hole

to fill

◼ Hole must be big enough to accommodate the to-be-loaded process

◼ Placement algorithms:

1. First fit

2. Best fit

3. Next fit

4. Worst fit

First-fit algorithm

❑ Scans memory from the beginning and chooses the first available block

that is large enough

❑ Simplest, Fastest

❑ May have many process loaded in the front end of memory that must be

searched over when trying to find a free block

Next-fit

❑ Scans memory from the location of the last placement

❑ More often allocate a block of memory at the end of memory where the

largest block is found

❑ The largest block of memory is broken up into smaller blocks

❑ Compaction is required to obtain a large block at the end of memory

Best-fit algorithm

❑ scan all holes to see which is best

❑ Chooses the block that is closest in size to the request

❑ Worst performer overall

❑ Since smallest block is found for process, the smallest amount of

fragmentation is left

❑ Memory compaction must be done more often

Worst-fit algorithm

-opposite to best-fit

**Non - Contiguous Memory Partitioning**

◼Logical :Reference to a memory location independent of the current assignment of data/instruction to memory. Generated by CPU

◼Physical or Absolute : The absolute address or actual location in main memory.

Memory Management Unit is the hardware device that maps logical address to physical address.

**Paging**

Partition memory into small equal fixed-size chunks and divide each process

into the same size chunks.

◼ The chunks of a process are called pages and chunks of memory are called

frames.

◼ Operating system maintains a page table for each process

❑ Contains the frame location for each page in the process

❑ Memory address consist of a page number and offset within the page

❑ Page number is used as an index to page table.

◼ Each process has its own page table.

◼ Each page table entry contains the frame number of the corresponding page

in main memory

◼ A bit is needed to indicate whether the page is in main memory or not.

◼ No external fragmentation

◼ all frames (physical memory) can be used by processes

◼ Possibility of Internal fragmentation

◼ The physical memory used by a process is no longer contiguous

◼ The logical memory of a process is still contiguous

◼ The logical and physical addresses are separated

◼ the process does not see the translation or the difference to having physical

memory

-The page size is defined by the hardware.

⚫ The page size is typically a power of 2.The size can be 512 bytes to 16

MB.

⚫ The selection of power of 2 as a page size makes translation of logical

address into a page number and page offset easy.

⚫ Address generated by CPU is logcal address which is divided into 2 parts:

⚫ Page number (p) – used as an index into a page table which contains

base address of each page in physical memory

⚫ Page offset (d) – combined with base address to define the physical

memory address that is sent to the memory unit

Logical to physical address translation

• Consider a logical address of n+m bits

– First n bits = page number

– Last m bits = offset

• Use the page number as an index into process page table to find

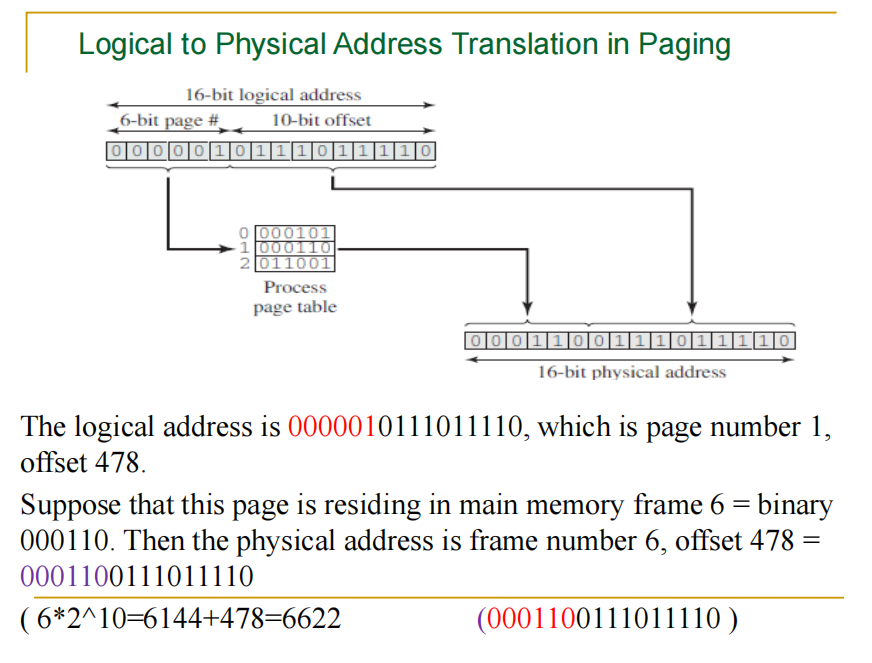
frame number, k

• Starting physical address of that frame then would be k X 2^m

• Desired physical address = this + offset, m

– Again, doesn’t need to be calculated

– Just append offset to the frame number k



**Segmentation**

User program and associated data now divided not into pages, but

segments which could be of unequal size

◼ All segments of all programs do not have to be of the same length, May be

unequal, dynamic size

◼ There is a maximum segment length, Simplifies handling of growing data

structures

◼ Logical Address consist of two parts - a segment number and an offset

◼ Similar to dynamic partitioning

A program however can now occupy more than one partitions

Partitions need not be contiguous

• Paging is invisible to the programmer, segmentation is usually visible and is

provided as a convenience for organizing programs and data

– Compiler or programmer assigns programs and data to different

segments

– One program may be further broken down into multiple segments for

purposes of modular programming

• Allows programs to be altered and recompiled independently

• Lends itself well to sharing data among processes

• Lends itself well to protection

• Simplifies handling of growing data structures

• Logical address to physical address translation is now little complicated but

similar

– Segment table

• Length of segment and starting physical address

Logical to Physical Address Translation in Segmentation

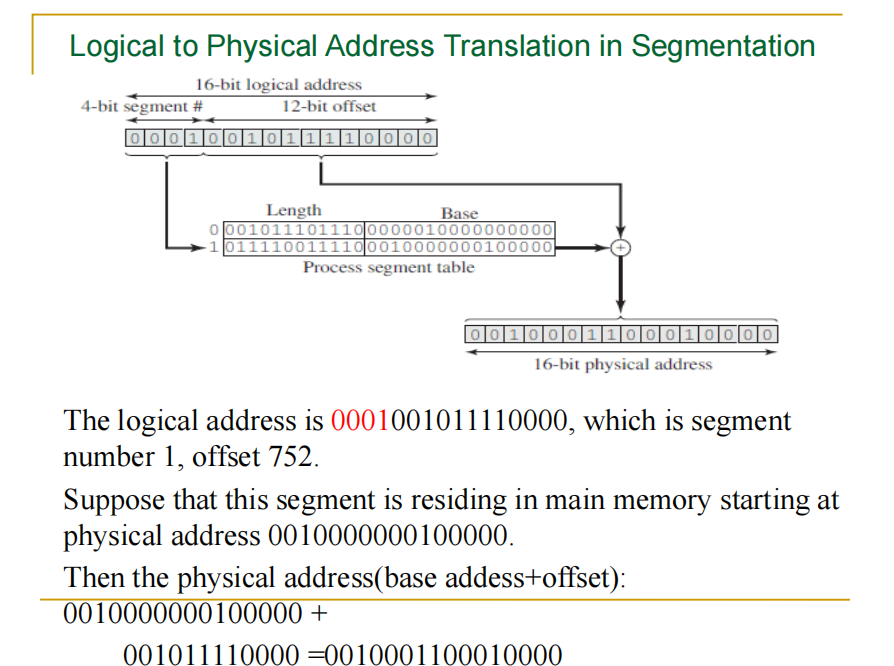
• Extract segment number from logical address (left most

n bits)

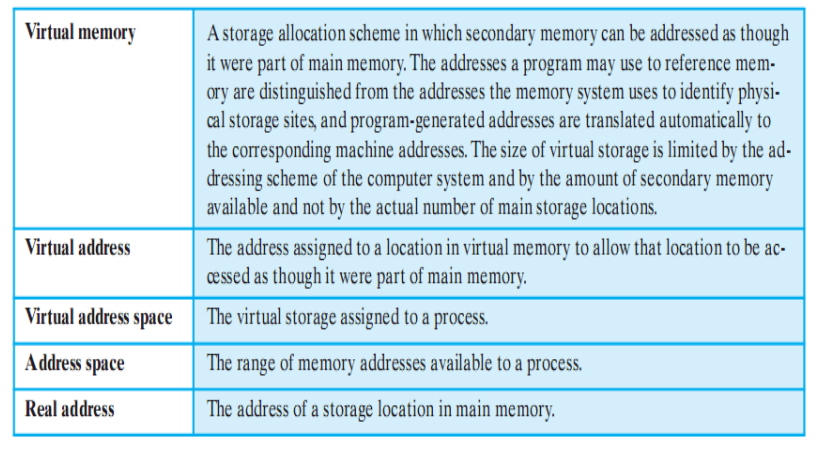
• Find base address of this segment from segment table

• Compare offset (rightmost m bits) with segment length

• If ok, desired physical address = base address + offset



Virtual Memory : Paging & Segmentation



it is not necessary that all of the pages or all of the

segments of a process be in main memory during

execution.

◼ If the next instruction, and the next data location are in

memory then execution can proceed

❑ at least for a time

Demand paging and demand segmentation mean that, when a

program is being executed, part of the program is in memory

and part is on disk. This means that, for example, a memory size of 10 MB can

execute 10 programs, each of size 3 MB, for a total of 30

MB. At any moment, 10 MB of the 10 programs are in memory

and 20 MB are on disk. There is therefore an actual memory

size of 10 MB, but a virtual memory size of 30 MB. Figure

7.11 shows the concept. **Virtual memory**, which implies

demand paging, demand segmentation or both, is used in

almost all operating systems today

• If a ‘piece’ is not in memory and is required, processor generates

interrupt indicating memory access fault

– OS takes charge, puts process in blocked state

– OS issues disk I/O to bring in the desired piece

– Schedules another process in the mean time

– Once brought in, I/O interrupt is raised, OS takes control again

and puts the process in Ready queue

Advantages of having only a portion

• More processes in memory at a given time with increase in

CPU utilization, throughput but no increase in response time

& turnaround time

• Processes can now be larger than main memory without any

tension on part of the programmer (overlaying)

• Why to waste memory with portions of program/data which

are being used only rarely

• Time is saved as unused pieces are not being swapped in/out

• Less I/O needed to load or swap user programs into memory,

so each user program would run faster.

Issues

• To bring in a piece, some other piece needs to be thrown out

– Piece is thrown out just before it is used

– Go get that piece again almost immediately

– Leads to **thrashing**

• System spends to much time swapping the pieces rather than

executing instructions

• Problem is worsened if OS mistakes it to be an indicator to increase the

level of multi programming

• Solution: OS tries to guess which pieces are least likely to be

used in the near future

Virtual Memory + Paging

• Earlier paging: when all pages of a process are loaded into memory,

its page table is created and loaded

• Page Table Entry (PTE) now needs to have

– An extra bit to indicate whether page is in memory or not (P)

– Another bit to indicate whether it is modified since it was last

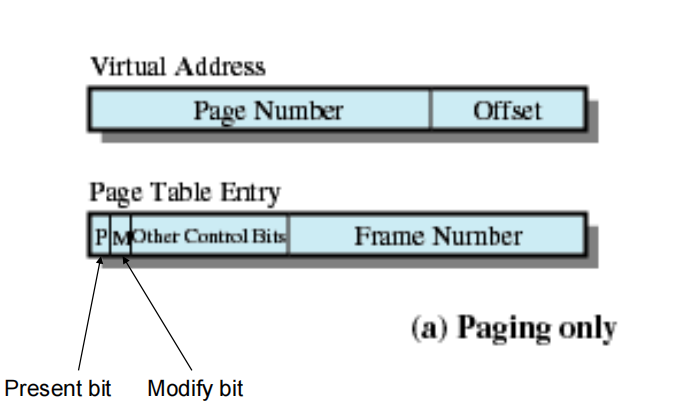
loaded (M)

• When somebody else comes to replace me, I will not need to

be written back to disk

– Some control bits

• For example protection or sharing at page level



◼ Page-table base register (PTBR) points to the page table

◼ Page-table length register (PRLR) indicates size of the page table

Translation Lookaside Buffer ( Cache )

Each virtual memory reference can cause two physical memory

accesses

❑ One to fetch appropriate page table

❑ One to fetch appropriate data

◼ Effect of doubling the memory access time!

◼ To overcome this problem a high-speed cache is set up for page

table entries

❑ Called a Translation Lookaside Buffer (TLB)

❑ Contains page table entries that have been most recently used

TLB Operation

◼ Given a virtual address, processor examines the TLB

◼ If page table entry is present (TLB hit), the frame number is retrieved

and the real address is formed

◼ If page table entry is not found in the TLB (TLB miss), the page

number is used to index the process page table

▪ First checks if page is already in main memory

• If yes, go ahead, update TLB to include this new entry

• If no, a page fault is issued to get the page (OS takes

over), page table is updated, instruction is re-executed

◼ It have entries between 64 to 1024

Page Size

◼ Secondary memory is designed to efficiently transfer large blocks of

data so a large page size is better

◼ Small page size, large number of pages will be found in main

memory

◼ As time goes on during execution, the pages in memory will all

contain portions of the process near recent references => Page

faults low.

◼ Increased page size causes pages to contain locations further from

any recent reference => Page faults rise.

Effect of page size on number of page faults

• With very small page size

– More pages in memory, thus lesser page faults

– Greater effect of principle of locality

• One page refers to nearby locations

• As page size increases

– Lesser pages in memory, thus more page faults

– Effect of locality reduced

• Each page will contain locations further and further away

from recent references

• Page fault rate is also determined by the number of frames

allocated to a process

VM + Segmentation

• Earlier segmentation: each process has its own segment table

– When all of its segments are loaded into main memory, segment

table is created and loaded

• With VM, STEs become more complex

– P bit (present/absent)

– M bit (modify)

– Other control bits

• For example to support sharing or protection at segment

level

Page Replacement Policies

Basic Page Replacement

1) Find the location of the desired page on disk

2) Find a free frame:

- If there is a free frame, use it

- If there is no free frame, use a page replacement

algorithm to select a **victim** frame

3) Bring the desired page into the (newly) free frame; update

the page and frame tables

4) Restart the process

**1. First-in, first-out (FIFO)**

❑ Treats page frames allocated to a process as a circular buffer

❑ Pages are removed in round-robin style

❑ Simplest replacement policy to implement

❑ Page that has been in memory the longest is replaced

❑ These pages may be needed again very soon

❑ How would you implement FIFO strategy?

•

Keep a queue and do round robin

**2.Optimal policy**

❑ Selects for replacement that page for which the time to the next

reference is the longest

❑ Impossible to have perfect knowledge of future events

**3.Least Recently Used (LRU)**

❑ Replaces the page that has not been referenced for the longest time

❑ By the principle of locality, this should be the page least likely to be

referenced in the near future

❑ Each page could be tagged with the time of last reference. This would

require a great deal of overhead

Global vs. Local Allocation

•

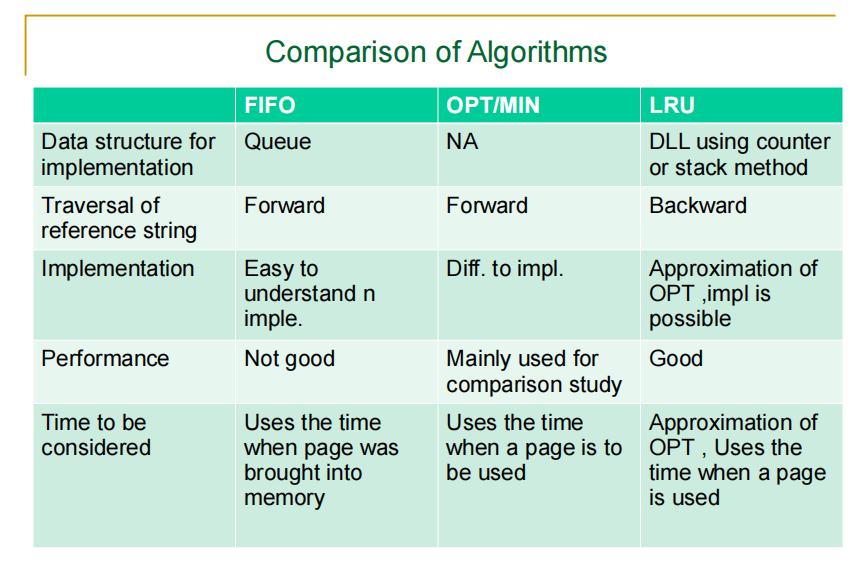
Global replacement – process selects a replacement frame from the set of

all frames; one process can take a frame from another.

•

Local replacement – each process selects from only its own set of allocated

frames.



Thrashing

◼ If a process does not have “enough” pages, the page-fault rate is very high.

This leads to:

❑ low CPU utilization

❑ operating system thinks that it needs to increase the degree of multiprogramming

❑ another process added to the system

◼ **Thrashing**  a process is busy swapping pages in and out

◼ Swapping out a piece of a process just before that piece is needed

◼ The processor spends most of its time swapping pieces rather than

executing user instructions

# UNIT 6 :I/O management:

Categories of I/O Devices

◼ Human readable/Interfacing

❑ Used to communicate with the user

❑ Ex:

❑ video display terminals

❑ keyboard

❑ mouse

❑ printer

◼ Machine readable/storage devices

❑ Used to communicate with electronic equipment

❑ Ex:

❑ Disk drives

❑ Tape drives

❑ Controllers

❑ Sensors

❑ Actuators

❑ USB keys

Categories of I/O Devices

◼ Communication/transmission

❑ Used to communicate with remote devices

❑ Ex:

❑ digital line drivers

❑ Modems

❑ controllers

I/O Buffering

<https://www.youtube.com/watch?v=1cSou52VdE0&t=345s>

◼ There is a speed mismatch between I/O devices and CPU. This

leads to inefficiency in processes being completed.

◼ To increase the efficiency, it may be convenient to perform input

transfers in advance of requests being made and to perform output

transfers some time after the request is made. This technique is

known as buffering.

◼ A buffer is a memory area that stores data while they are transferred

b/n two devices or b/n a device & an application.

Single Buffer (input)

• Operating system assigns a buffer in system portion of the main

memory for an I/O request

• Example in case of block-oriented devices

• Input transfers made to the system buffer

• When transfer is complete, process moves the block to user

space

• Immediately requests another block

• Called reading ahead or anticipated input

• Fair enough assumption except for the “last” block

• User process can now be processing the previous block while the

next block is being read

• OS can feel free to swap because I/O is taking place with system

memory, not process memory

◼ Block-oriented

❑ user process can process one block of data while next

block is read in

❑ swapping can occur since input is taking place in

system memory, not user memory

❑ operating system keeps track of assignment of system

buffers to user processes

❑ output is accomplished by the user process writing a

block to the buffer and later actually written out

◼ Stream-oriented

❑ in a line- at- a time Fashion or a byte-at-a-time

fashion.

❑ user input from a terminal is one line at a time with

carriage return signaling the end of the line

❑ output to the terminal is one line at a time

❑ For eg. Scroll-mode terminals, line printer

❑ In case of byte-at-a-time ,each keystroke is important

in sensor,controller.

❑ OS & user process follows producer-consumer model

Double Buffer

◼ Use two system buffers instead of one

◼ A process can transfer data to or from one buffer while the operating system

empties or fills the other buffer.

◼ known as **buffer swapping**

Circular Buffer

◼ Double buffering may be inadequate if the process performs rapid bursts of I/O.

◼ More than two buffers are used

◼ Each individual buffer is one unit in a circular buffer

◼ Used when I/O operation must keep up with process

◼ OS & user process follows producer-consumer model

Disk Scheduling

Notes are remaining for this part