The Operating Systems Handbook

For Interview Preparation

Pavan Areti

National Institute of Technology , Silchar

LinkedIn
November 8, 2022

Contents

Chapter 1

Basics of OS

1.1 What is an OS?

- An operating system is a software that helps the users to interact with the computer hardware. It also acts as the interface between the user and the hardware.
- When user runs any application. And when any applications need to be executed, then it interacts with the operating system. And then Operating System provides all the tools and also manages the execution of the application in the CPU.
- Android OS, Mac OS (Macintosh), Linux, Windows, Unix, etc are different kinds of Operating systems.

1.2 Functionality of Operating System

1. Resource Management
2. Process Management
3. Storage Management (File System) and Memory Management (RAM)
4. Security and privacy

1.3 Services offered by an OS

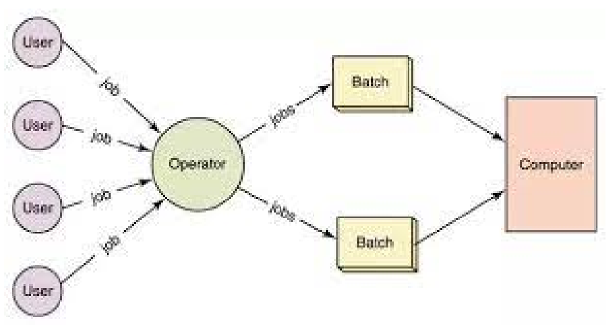
1. User interface
2. Program execution
3. I/O operations
4. File system management
5. Communications
6. Error detection
7. Resource allocation
8. Security

reference

1.4 Types of OS

Batch OS

Same type of jobs batch together and execute at a time



Demerits:

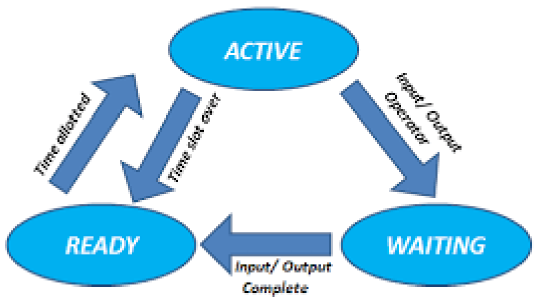
1. CPU is being often idle, because the speed of the mechanical I/O devices is slower than the CPU

2. It is difficult to provide the desired priority.

Time Sharing/Multi tasking OS

Each task is given some time for execution, if the task is not completed on time , it moves to the next process and schedules the previous task for later, The amount of time that each task gets is called as quantum Example of Time sharing OS : Unix

Figure 1.1: Time Sharing OS (src:bansalwiki.blogspot.com)



Merits:

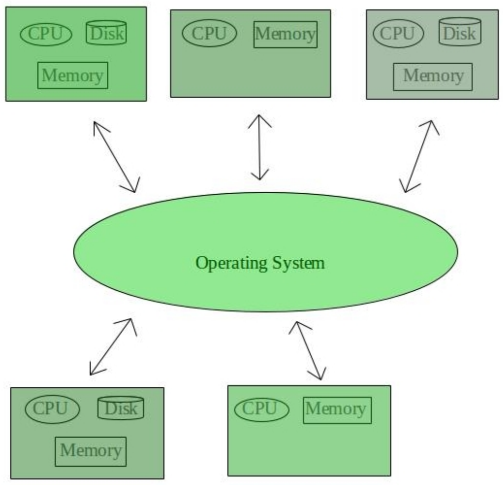
1. CPU idleness can be reduced

2. Each task gets equal opportunity

Distributed OS

In this kind of operating system, every system has its own environment and Various autonomous interconnected computers communicate with each other using a shared communication network.These are referred to as loosely coupled systems.The major benefit is that a user can also access files which are not locally stored in his system but in some other system in the same network

Figure 1.2: Distributed OS (src:geeksforgeeks.com)



Network OS

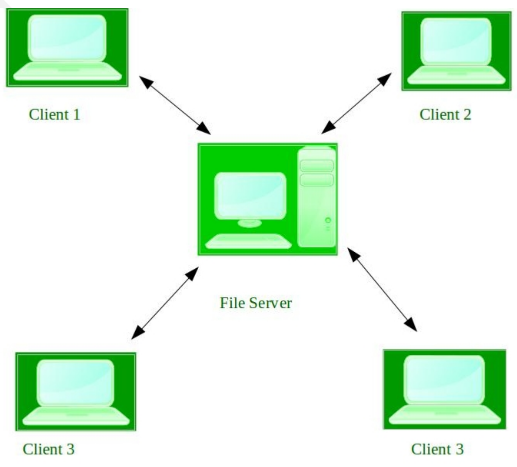


Figure 1.3: Network OS (src:geeksforgeeks.com)

These systems run on a server and provide the capability to manage data, users, groups, security, applications, and other networking functions, these systems are also called as tightly coupled systems

Advantages of Network Operating System:

1. Security concerns are handled through servers
2. New technologies and hardware up-gradation are easily integrated into the system
3. Server access is possible remotely from different locations and types of systems

Disadvantages of Network Operating System:

1. server costs are high
2. User needs to depend on a central server for most operations
3. Maintenance and updates are required regularly for efficient usage

Real time OS

In these systems , the time duration required to respond to inputs is very low , used when time restrictions are strict
example : robots , traffic control system , rockets , missiles etc
Two types of Real-Time Operating System which are as follows:

Hard Real-Time Systems:

These OS are meant for applications where time constraints are very strict and even the shortest possible delay is not acceptable.

Soft Real-Time Systems:

These OS are for applications where time-constraint is less strict.

1.5 Multi-programming, multitasking, multi-threading and multiprocessing

Multi-programming

When multiple programs execute at a time on a single device, it is multi-programming.when one process went to do I/O task , instead of CPU being idle , another process enters CPU and utilises it

Multi-Processing

Multiprocessing refers to processing of multiple processes at same time by multiple CPU's.Say processes P1, P2, P3 and P4 are waiting for execution. Now in a single processor system, firstly one process will execute, then the other, then the other and so on.But with multiprocessing, each process can be assigned to a different processor for its execution.

Multi-Tasking

Multitasking is nothing but multi-programming with a Round-robin scheduling algorithm. Multitasking is a logical extension of multi programming. The major way in which multitasking differs from multi programming is that multi programming works solely on the concept of context switching whereas multitasking is based on time sharing alongside the concept of context switching.

Multi-Threading

Multi-threading enables us to run multiple threads concurrently. We can think of threads as child processes that share the parent process resources but execute independently. Now take the case of a GUI. Say we are performing a calculation on the GUI (which is taking very long time to finish). Now we can not interact with the rest of the GUI until this command finishes its execution. To be able to interact with the rest of the GUI, this command of calculation should be assigned to a separate thread. So at this point of time, 2 threads will be executing i.e. one for calculation, and one for the rest of the GUI. Hence here in a single process, we used multiple threads for multiple functionality.

Advantages:

1. Benefits of Multi threading include increased responsiveness. Since there are multiple threads in a program, so if one thread is taking too long to execute or if it gets blocked, the rest of the threads keep executing without any problem. Thus the whole program remains responsive to the user by means of remaining threads.

2. Another advantage of multi threading is that it is less costly.

1.6 RAM, ROM and its differences

RAM

The full form of RAM is Random Access Memory. The information stored in this type of memory is lost when the power supply to the PC or laptop is switched off. It is generally known as the main memory, or temporary memory or cache memory or volatile memory of the computer system.

ROM

The full form of ROM is a Read-Only Memory. It is a permanent type of memory. Its content are not lost when the power supply is switched off.

The biggest advantage of RAM is that it does not have any moving parts while the biggest advantage of Rom is that it is not lost when power is switched off.

What is the Difference Between RAM and ROM?

The difference between RAM (Random Access Memory) and ROM (Read Only Memory) is RAM is volatile. So, its contents are lost when the device is powered off. Whereas ROM stores all the application which is needed to boot the computer initially, it only allows for reading.

Types of RAM

1. DRAM -Dynamic RAM must be continuously refreshed, or otherwise, all contents are lost.

2. SRAM – Static RAM is faster, needs less power but is more expensive. However, it does not need to be refreshed like DRAM.

3. Synchronous Dynamic RAM (SDRAM) – This type of RAM can run at very high clock speeds.

4. DDR – Double Data Rate provide synchronous Random Access Memory

Types of ROM

1. EPROM: The full form of EPROM is Erasable Programmable Read-only memory. It stores instructions, but you can erase only by exposing the memory to ultraviolet light.

2. PROM: The full form of PROM is Programmable Read-Only memory. This type of ROM is written or programmed using a particular device.

3. EEPROM: stands for electrically Erasable Programmable Read-Only Memory. It stores and deletes instructions on a special circuit.

4. Mask ROM: is a full form of MROM is a type of read-only memory (ROM) whose contents can be programmed only by an integrated circuit manufacturer.

Detailed Reference

1.7 Few important terms in OS

Compiler

A compiler is a computer program that changes source code written in a high-level language into low-level machine language.

Loader

In computer systems a loader is the part of an operating system that is responsible for loading programs and libraries. It is one of the essential stages in the process of starting a program, as it places programs into memory and prepares them for execution.

Assembler

The Assembler is a Software that converts an assembly language code to machine code. It takes basic Computer commands and converts them into Binary Code that Computer's Processor can use to perform its Basic Operations.

Interpreter

A command interpreter is the part of a computer operating system that understands and executes commands that are entered interactively by a human being or from a program. In some operating systems, the command interpreter is called the shell.

System calls

A system call is a way for a user program to interface with the operating system. The program requests several services, and the OS responds by invoking a series of system calls to satisfy the request. A system call can be written in assembly language or a high-level language like C or Pascal. System calls are predefined functions that the operating system may directly invoke if a high-level language is used.

API

An application programming interface is a way for two or more computer programs to communicate with each other. It is a type of software interface, offering a service to other pieces of software.

Kernel

It is the core that provides basic services for all other parts of the OS. It is the main layer between the OS and underlying computer hardware, and it helps with tasks such as process and memory management, file systems, device control and networking.

Shell

A shell is an environment in which we can run our commands, programs, and shell scripts.
(or)
A shell is basically an interface present between the kernel and the user. A kernel is the very core of a typical OS. Meaning. A shell is a CLI (command-line interpreter). A kernel is a type of low-level program that has its interfacing with the hardware on top of which all the applications run

Booting

Booting is a startup sequence that starts the operating system of a computer when it is turned on.

1.8 What happens when we turn on computer?

The first thing a computer has to do when it is turned on is to start up a special program called an operating system.

1. PC is ON

2. CPU initializes itself and looks for a firmware program (BIOS) stored in BIOS Chip

3. CPU runs the BIOS which tests and initializes system hardware. Bios loads configuration settings.
   This is called POST (Power on self-test) process.

4. BIOS looked at the MBR (master boot record), a special boot sector at the beginning of a disk , MBR contains the code that loads the rest of the OS known as Boot-Loader

5. The BIOS executes boot-loader , which begins booting the actual operating system

6. Boot-loader is a small program which has to do large task of booting,windows uses boot-loader named windows boot manager (Bootmgr.exe)

Chapter 2

Processes

2.1 what is a process?

A process is a program in execution. Process is not as same as program code but a lot more than it. A process is an 'active' entity as opposed to program which is considered to be a 'passive' entity. Attributes held by process include hardware state, memory, CPU etc.

2.2 Types of Process

Event-specific based category of process

CPU Bound Process

Processes that spend the majority of their time simply using the CPU (doing calculations).

I/O Bound process

Processes that are associated with input/output-based activity like reading from files, etc.

Category of processes based on their nature

Independent Process

A process that does not need any other external factor to get triggered is an independent process.

Co-operative Process

A process that works on the occurrence of any event and the outcome affects any part of the rest of the system is a cooperating process.

2.3 What are the process states ?

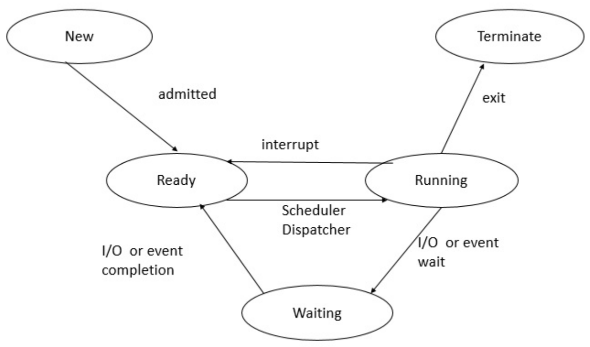


Figure 2.1: Process states (src:www.tutorialspoint.com)

1. Whenever a new process is created, it is admitted into ready state.

2. If no other process is present at running state, it is dispatched to running based on scheduler dispatcher.

3. If any higher priority process is ready, the uncompleted process will be sent to the waiting state from the running state.

4. Whenever I/O or event is completed the process will send back to ready state based on the interrupt signal given by the running state.
5. Whenever the execution of a process is completed in running state, it will exit to terminate state, which is the completion of process.

2.4 PCB structure

1. While creating a process to identify the processes, it assigns a process identification number (PID) to each process.
2. As the operating system supports multi-programming, it needs to keep track of all the processes. For this task, the process control block (PCB) is used to track the process's execution status.
3. Each block of memory contains information about the process state, program counter, stack pointer, status of opened files, scheduling algorithms, etc.
4. A process control block (PCB) contains information about the process. The process table is an array of PCB's, that means logically contains a PCB for all of the current processes in the system.

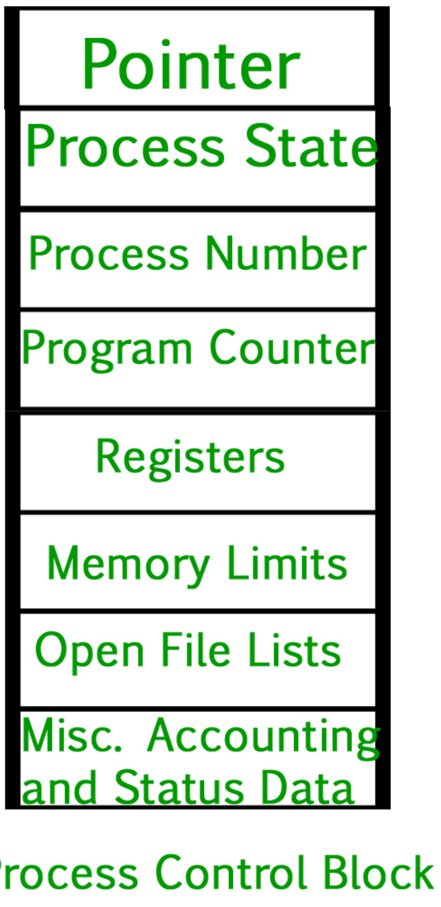
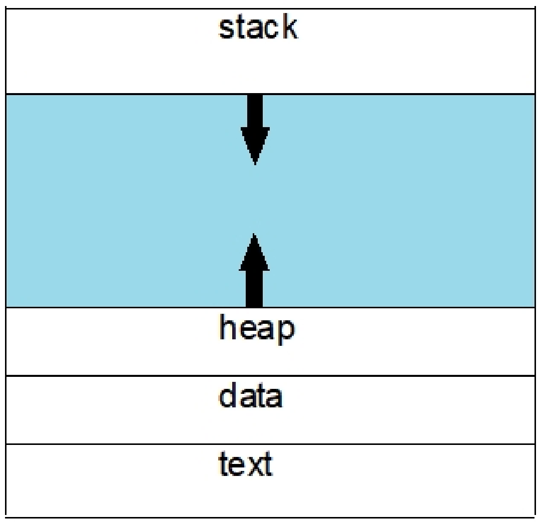


Figure 2.2: Process Control Block (src:www.geeksforgeeks.com)

2.5 How does a process look like in memory?



Text

Code or a code segment is known as Text Section. This section of memory contains the executable instructions of a program.

Data

Data Section segment of memory contains the global and static variables that are initialized by the programmer prior to the execution of a program.

Heap

To allocate memory for variables whose size cannot be statically determined by the compiler before program execution, requested by the programmer, there is a requirement of dynamic allocation of memory which is done in heap segment.

stack

A process generally also includes the process stack, which contains temporary data i.e. function parameters, return addresses, and local variables.

2.6 Process vs threads

Process:

Process is basically a program under execution , A process can create other processes which are known as Child Processes. The process takes more time to terminate and it is isolated means it does not share the memory with any other process.
The process can have the following states new, ready, running, waiting, terminated, and suspended.

Thread:

Thread is the segment of a process which means a process can have multiple threads and these multiple threads are contained within a process. A thread has three states: Running, Ready, and Blocked.

Few differences :

1. If one process is blocked then it will not affect the execution of other processes,If a user-level thread is blocked, then all other user-level threads are blocked.

2. Processes wont share memory , but threads will

3. System call is involved in process whereas API is Involved in threads

4. Threads take less time for context switching compared to processes

2.7 Process Scheduling

Introduction:

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.

Categories of Scheduling

Non-preemptive:

Here the resource can't be taken from a process until the process completes execution. The switching of resources occurs when the running process terminates and moves to a waiting state.

Preemptive:

Here the OS allocates the resources to a process for a fixed amount of time. During resource allocation, the process switches from running state to ready state or from waiting state to ready state. This switching occurs as the CPU may give priority to other processes and replace the process with higher priority with the running process.

Process Scheduling Queues

The OS maintains all Process Control Blocks (PCBs) in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

Job queue

This queue keeps all the processes in the system.

Ready queue

This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.

Device queues

The processes which are blocked due to unavailability of an I/O device constitute this queue.

2.8 Schedulers:

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run.

Long term Scheduler:

It is also called a job scheduler. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

Short Term Scheduler

It is also called as CPU scheduler.It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Medium Term Scheduler

It is a part of swapping. some running process requires to perform I/O operation , in this situation process suspends from main memory and placed on the secondary memory and these processes after a while reloaded in memory and continued where they left earlier , swap in and swap out is done by medium term scheduler.

Schedulers in simple words

LTS : secondary memory to main memory
STS : main memory to CPU
MTS : swapping

What is Context switching ?

Context Switching involves storing the context or state of a process so that it can be reloaded when required and execution can be resumed from the same point as earlier. This is a feature of a multitasking operating system and allows a single CPU to be shared by multiple processes.

Context switching triggers

1. Multitasking
2. User and kernal mode switching
3. Interrupt handling

2.9 IPC (Inter Process Communication)

Reference
Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions. The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other through both:

1. Shared Memory
2. Message passing

Shared Memory

Producer consumer problem

There are two processes: Producer and Consumer. The producer produces some items and the Consumer consumes that item. The two processes share a common space or memory location known as a buffer where the item produced by the Producer is stored and from which the Consumer consumes the item if needed.

Message Passing

In this method, processes communicate with each other without using any kind of shared memory. If two processes p1 and p2 want to communicate with each other, they proceed as follows:

- Establish a communication link (if a link already exists, no need to establish it again.)
- Start exchanging messages using basic primitives.
- We need at least two primitives:
  - send(message, destination) or send(message)
  - receive(message, host) or receive(message)

2.10 Zombie process and Orphan Process

Zombie Process

A zombie process is a process whose execution is completed but it still has an entry in the process table. Zombie processes usually occur for child processes, as the parent process still needs to read its child's exit status.

Orphan Process

A process whose parent process no more exists i.e. either finished or terminated without waiting for its child process to terminate is called an orphan process.

Chapter 3

Thread Concepts

3.1 What is a Thread ?

Thread is often referred to as a lightweight process. The process can be split down into so many threads.

Types of Threads

- User threads
- Kernel threads

User Thread

User threads are above the kernel and without kernel support. These are the threads that application programmers use in their programs.

Kernel Thread

Kernel threads are supported within the kernel of the OS itself. All modern OS's support kernel-level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.

Reference for types of threads

3.2 Why do we need Multi-threading ?

A thread is also known as lightweight process. The idea is to achieve parallelism by dividing a process into multiple threads. MS Word uses multiple threads: one thread to format the text, another thread to process inputs, etc. More advantages of multi-threading are discussed below

Advantages of Multi-threading

1. Responsiveness (Faster context switch)
2. Resource Sharing (Easier communication)
3. Economy
4. Effective utilization of multiprocessor system

3.3 Process vs Thread

The primary difference is that threads within the same process run in a shared memory space, while processes run in separate memory spaces. Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals). But, like process, a thread has its own program counter (PC), register set, and stack space.

3.4 Multi-Threading models

https://www.studytonight.com/operating-system/multithreading

The user threads must be mapped to kernel threads, by one of the following strategies:

1. Many to One
2. One to One

3. Many to Many

Many to One Model

In the Many-to-one model many user-level threads are mapped onto a single kernel level thread

One to One Model

In this a Kernel level thread is created for every user level thread ,This model provides more concurrency compared to previous models , windows OS uses this

Many to Many Model

Many to many model multiplexes any number of user threads to equal or smaller number of kernel threads , this model combines the best of previous 2 models

Blocking the kernel system call wont block the entire process

Chapter 4

Process Scheduling

4.1 What is Process Scheduling ?

Process scheduling is a task of OS that schedules different states, Process scheduling allows OS to allocate a time interval of CPU for each process

Reference for CPU Scheduling

4.2 Types of CPU scheduling

- Pre-emptive Scheduling
- Non Pre-emptive Scheduling

Pre-emptive Scheduling

Preemptive scheduling is used when a process switches from running state to ready state or from the waiting state to ready state. The resources (mainly CPU cycles) are allocated to the process for a limited amount of time and then taken away, and the process is again placed back in the ready queue if that process still has CPU burst time remaining. That process stays in the ready queue till it gets its next chance to execute.

Non Pre-emptive Scheduling

Non-preemptive Scheduling is used when a process terminates, or a process switches from running to the waiting state. In this scheduling, once the resources (CPU cycles) are allocated to a process, the process holds the CPU till it gets terminated or reaches a waiting state. In the case of non-preemptive scheduling does not interrupt a process running CPU in the middle of the execution. Instead, it waits till the process completes its CPU burst time, and then it can allocate the CPU to another process.

Few important points regarding preemptive and non preemptive scheduling

1. In preemptive scheduling, the CPU is allocated to the processes for a limited time whereas, in Non-preemptive scheduling, the CPU is allocated to the process till it terminates or switches to the waiting state.

2. In preemptive scheduling, if a high-priority process frequently arrives in the ready queue then the process with low priority has to wait for a long, and it may have to starve. , in the non-preemptive scheduling, if CPU is allocated to the process having a larger burst time then the processes with small burst time may have to starve.

3. Preemptive scheduling attains flexibility by allowing the critical processes to access the CPU as they arrive into the ready queue, no matter what process is executing currently. Non-preemptive scheduling is called rigid as even if a critical process enters the ready queue the process running CPU is not disturbed.

4.3 What is a Dispatcher ?

It is a module that provides control of the CPU to the process. The Dispatcher should be fast so that it can run on every context switch. Dispatch latency is the amount of time needed by the CPU scheduler to stop one process and start another.

Functions performed by dispatcher :

1. Context switching

2. Switching to user mode

4.4 Criterion's related to CPU scheduling

Burst Time/Execution Time

It is a time required by the process to complete execution. It is also called running time.

Throughput

It is the number of process completed execution for unit amount of time

Arrival Time

The time a process enters the ready state

Finish Time/Completion Time

The time process complete and exit from a system

Turn around time

The difference between the time of completion and the time of arrival is known as the Turn Around Time of the process.

Waiting time

The total time spent by the process in the ready state waiting for CPU.

TAT = CT-AT
WT = TAT - BT

Response time

It is the time spent between the Arrival time and getting the CPU for the first time.

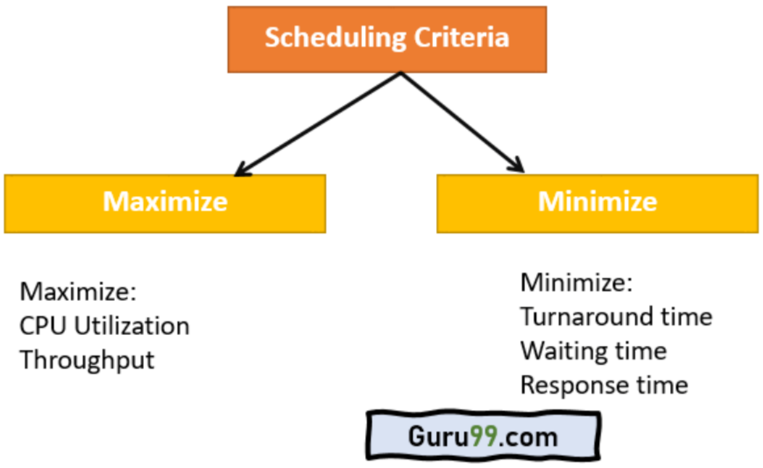


Figure 4.1: Scheduling Criteria (src:https://www.guru99.com/)

4.5 Scheduling Algorithms

First Come First Serve (FCFS)

In this type of algorithm, the process which requests the CPU gets the CPU allocation first. This scheduling method can be managed with a FIFO queue.

Shortest Job First (SJF)

SJF is a full form of (Shortest job first) is a scheduling algorithm in which the process with the shortest execution time should be selected for execution next.

Priority-based

Priority scheduling is a method of scheduling processes based on priority. In this method, the scheduler selects the tasks to work as per the priority.The processes with higher priority should be carried out first, whereas jobs with equal priorities are carried out on a round-robin or FCFS basis.

Round-Robin

Round Robin is the preemptive process scheduling algorithm.Each process is provided a fix time to execute, it is called a quantum.Once a process is executed for a given time period, it is preempted and other process executes for a given time period.Context switching is used to save states of preempted processes.

Multi level Queues Scheduling

Multiple-level queues are not an independent scheduling algorithm. They make use of other existing algorithms to group and schedule jobs with common characteristics.

- Multiple queues are maintained for processes with common characteristics.
- Each queue can have its own scheduling algorithms.
- Priorities are assigned to each queue.

For example, CPU-bound jobs can be scheduled in one queue and all I/O-bound jobs in another queue. The Process Scheduler then alternately selects jobs from each queue and assigns them to the CPU based on the algorithm assigned to the queue.

Multi level feedback queue

Reference
CPU Scheduling is like Multilevel Queue(MLQ) Scheduling but in this processes can move between the queues. And thus, much more efficient than multilevel queue scheduling.

4.6 Starvation and Aging

Reference

Starvation

Starvation or indefinite blocking is a phenomenon associated with the Priority scheduling algorithms, in which a process ready for the CPU (resources) can wait to run indefinitely because of low priority.

Ageing

Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time. For example, if priority range from 127(low) to 0(high), we could increase the priority of a waiting process by 1 Every 15 minutes.

Chapter 5

Synchronisation

5.1 Introduction

As we have seen earlier , there are two types of process

- Independent process
- Co-operative process

In co-operative process execution of one process affects the execution of another process,These processes need to be synchronized so that the order of execution can be guaranteed.

5.2 Process Synchronisation

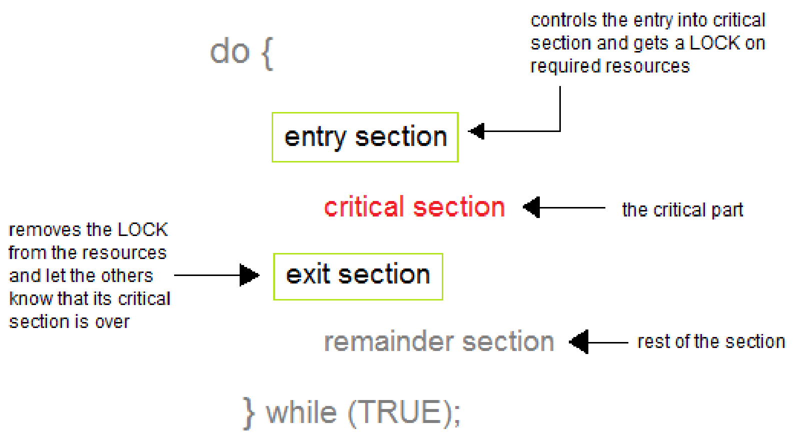
It is the phenomenon of coordinating the execution of process in such a way that no 2 process can have access to same shared memory at same time.In order to synchronize the processes, there are various synchronization mechanisms.

Process Synchronization is mainly needed in a multi-processing system when multiple processes are running together, and more than one processes try to gain access to the same shared resource or any data at the same time

5.3 Critical section

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

The entry to the critical section is mainly handled by wait() function while the exit from the critical section is controlled by the signal() function.



do {
controls the entry into critical section and gets a LOCK on required resources
entry section
critical section
the critical part
removes the LOCK from the resources and let the others know that its critical section is over
exit section
remainder section
rest of the section
} while (TRUE);

Figure 5.1: Critical Section (src:https://www.studytonight.com/)

5.4 Race Condition

At some point of time if more than one process which use same critical section are being executed then it may lead to wrong output , this scenario is called as race condition

But this condition in critical sections can be avoided if the critical section is treated as an atomic instruction. Proper thread synchronization using locks or atomic variables can also prevent race conditions.

5.5 Conditions to solve critical section problem

Mutual Exclusion

when multiple processes are running at a time , only one process can have access to critical section

Progress

If critical section is empty and if one or more process wants to execute their critical section code then any one process or thread must be allowed

Bounded Waiting

After a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section, before this process's request is granted. So after the limit is reached, the system must grant the process permission to get into its critical section.

5.6 Solution to critical section problem

The critical section plays an important role in Process Synchronization so that the problem must be solved.

1. Peterson's Solution

With the help of this solution whenever a process is executing in any critical state, then the other process only executes the rest of the code, and vice-versa can happen. This method also helps to make sure of the thing that only a single process can run in the critical section at a specific time.

this solution preserves all the 3 required conditions i.e mutual exclusion, progress , and bounded waiting

When flag[i] is set to true we will give turn to j, so that i will check whether j is present in critical section or not

```
*** Petersons Solution ***

do{
   flag[i] = true;
   turn = j;
   while(flag[j] && turn==j);//waiting
   //critical section
   flag[i] = false;
}while(1)
```

Disadvantages of Peterson's solution

1. Busy Waiting (wastage of cpu cycles for while loop)

2. Only limited to 2 processes

2. Mutex locks

In this approach, in the entry section of code, a LOCK is acquired over the critical resources modified and used inside the critical section, and in the exit section that LOCK is released. As the resource is locked while a process executes its critical section hence no other process can access it.

5.7 Semaphores

1. A semaphore is a signaling mechanism and a thread that is waiting on a semaphore can be signaled by another thread. This is different than a mutex as the mutex can be signaled only by the thread that is called the wait function.

2. A semaphore uses two atomic operations, wait and signal for process synchronization.

3. A Semaphore is an integer variable, which can be accessed only through two operations wait() and signal().

4. semaphore always has non-negative number and Can have many different critical sections with different semaphores.

Wait

This Operation mainly helps you to control the entry of a task into the critical section. In the case of zero value, no operation is executed. wait() operation was originally termed as P; so it is also known as P(S) operation.

Signal

This Operation is mainly used to control the exit of a task from the critical section.signal() operation was originally termed as V; so it is also known as V(S) operation.

```
P(S): if S >= 1 then S := S - 1
      else <block and enqueue the process>;

V(S): if <some process is blocked on the queue>
        then <unblock a process>
      else S := S + 1;
```

5.8 Types of Semaphores

There are two types of semaphores

1. Binary Semaphore
2. Counting Semaphore

Binary Semaphore

It is a special form of semaphore used for implementing mutual exclusion, hence it is often called a Mutex. A binary semaphore is initialized to 1 and only takes the values 0 and 1 during the execution of a program. In Binary Semaphore, the wait operation works only if the value of semaphore = 1, and the signal operation succeeds when the semaphore= 0. Binary Semaphores are easier to implement than counting semaphores.

Counting Semaphores

The semaphore can be initialized to the number of instances of the resource. Whenever a process wants to use that resource, it checks if the number of remaining instances is more than zero, i.e., the process has an instance available. Then, the process can enter its critical section thereby decreasing the value of the counting semaphore by 1. After the process is over with the use of the instance of the resource, it can leave the critical section thereby adding 1 to the number of available instances of the resource.
One advantage of semaphore is that, there is no busy waiting

Busy Waiting

Busy waiting, also known as spinning, or busy looping is a process synchronization technique in which a process/task waits and constantly checks for a condition to be satisfied before proceeding with its execution.
spin-lock is a kind of busy waiting

Deadlock

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

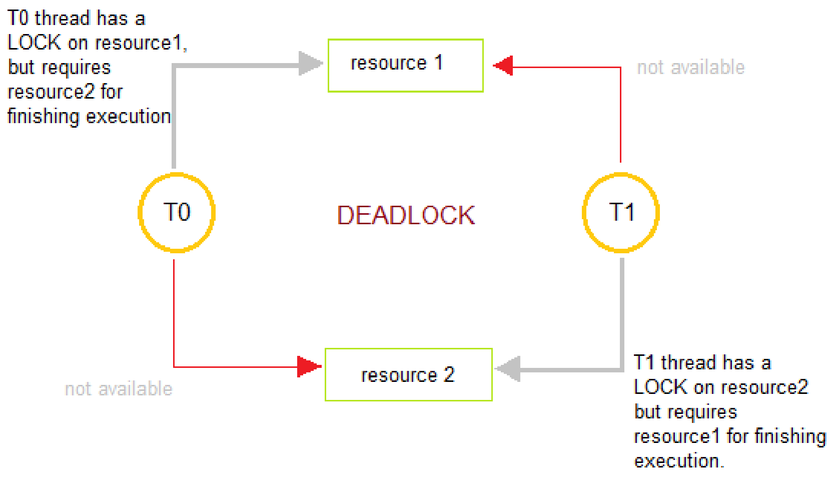

T0 thread has a LOCK on resource1, but requires resource2 for finishing execution
resource 1
not available
T0
DEADLOCK
T1
not available
resource 2
T1 thread has a LOCK on resource2 but requires resource1 for finishing execution.

Figure 5.2: Deadlock (src:https://www.guru99.com/)

Starvation vs Deadlock

starvation is when all the low priority process got blocked due to usage of resources by high priority process
Deadlock is when one of the processes got blocked.
Starvation is not an infinite process but deadlock is a infinite process

we will study about deadlock in detail in next chapter

5.9 Classical Synchronisation Problems

Below are some of the classical problem depicting flaws of process synchronization in systems where cooperating processes are present.

Bounded Buffer Problem (Producer-Consumer Problem)

Problem

There is a buffer of n slots and each slot is capable of storing one unit of data. There are two processes running, namely, producer and consumer, which are operating on the buffer.A producer tries to insert data into an empty slot of the buffer. A consumer tries to remove data from a filled slot in the buffer. As you might have guessed by now, those two processes won't produce the expected output if they are being executed concurrently.

Solution

1. One solution of this problem is to use semaphores. The semaphores which will be used here are:
2.
   - m, a binary semaphore which is used to acquire and release the lock.
   - empty, a counting semaphore whose initial value is the number of slots in the buffer, since, initially all slots are empty.
   - full, a counting semaphore whose initial value is 0.
3. At any instant, the current value of empty represents the number of empty slots in the buffer and full represents the number of occupied slots in the buffer.

```
        // ********  Producer Code *******
do
{
    // wait until empty > 0 and then decrement 'empty'
    wait(empty);
    // acquire lock
    wait(mutex);
    /* perform the insert operation in a slot */
    // release lock
    signal(mutex);
    // increment 'full'
    signal(full);
}
while(TRUE)
```

```
          //*********** Consumer Code**********
do
{
    // wait until full > 0 and then decrement 'full'
    wait(full);
    // acquire the lock
    wait(mutex);
    /* perform the remove operation in a slot */
    // release the lock
    signal(mutex);
    // increment 'empty'
    signal(empty);
}
while(TRUE);
```

Dining Philosophers Problem

It is used to evaluate situations where there is a need of allocating multiple resources to multiple processes.

Problem Statement:

Consider there are five philosophers sitting around a circular dining table. The dining table has five chopsticks and a bowl of rice in the middle as shown in the below figure.At any instant, a philosopher is either eating or thinking. When a philosopher wants to eat, he uses two chopsticks -

one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place. if all five philosophers are hungry simultaneously, and each of them pickup one chopstick, then a deadlock situation occurs because they will be waiting for another chopstick forever.

Solution:

```
     //***** Dining Philosopher Solution *****
while(TRUE)
{
    // allow to eat only after getting 2 sticks
    wait(stick[i]);
    wait(stick[(i+1) % 5]);
    /* eat */
    signal(stick[i]);
    signal(stick[(i+1) % 5]);
    /* think */
}
This is not a valid solution as there is dead lock.
```

- A philosopher must be allowed to pick up the chopsticks only if both the left and right chopsticks are available.

- another solution is Allow only four philosophers to sit at the table. That way, if all the four philosophers pick up four chopsticks, there will be one chopstick left on the table.

- **The original solution is all philosphers should pick right chopstick while one should pick their left one first, then deadlock can be solved**

Reader-Writer Problem

Problem Statement:

from DBMS we already know that Read-Read never cause any synchronisation problems, but all other combinations like read-write, write-write ,write-read will cause synchronisation problems

Solution:

- as writer is always causing problem just dont allow any other operation when writer is in critical section , so it will have simple mutex code
- allow multiple readers but don't allow any writer
- when one reader went to critical section it sends wait() for writer so that no writer enters critical section
- when all the readers came out of critical section it will signal() to writer

```
       \\***** writers Code *****
while(TRUE)
{
   wait(wrt);
   /*perform the write operation*/
   signal(wrt);
}
```

```
       \\ ***** readers code *****
while(TRUE)
{
    //acquire lock as read_count is shared variable
    wait(m);
    read_count++;
    if(read_count == 1)
        wait(wrt);
    //release lock
    signal(m);

    /* perform the reading operation */

    // acquire lock
    wait(m);
    read_count--;
    if(read_count == 0)
        signal(wrt);
    // release lock
    signal(m);
}
```

Chapter 6

Deadlocks

6.1 What is Deadlock ?

In a multi-programming system a number of process compete for a limited number of resources and if a resource is not available at that instance process enters into waiting state , if a process is unable to change its waiting state indefinitely because the resource requested by it are held by another waiting process, then it is said to be deadlock

6.2 What happens to other process when deadlock ?

Only that processes which are in deadlock will block.
And it's one of the reasons that most of modern personal computers ignore it. (Since deadlock prevention,avoidance,detection and recovery are expensive)

6.3 Conditions required for deadlock

All these conditions should satisfy simultaneously for deadlock to occur

Mutual Exclusion

According to this condition, atleast one resource should be non-shareable (non-shareable resources are those that can be used by one process at a time.)

Hold and wait

According to this condition, A process is holding atleast one resource and is waiting for additional resources.

No Pre-emption

Resources cannot be taken from the process because resources can be released only voluntarily by the process holding them.

Circular Wait

In this condition, the set of processes are waiting for each other in the circular form.

6.4 Methods to handle deadlock

Deadlock Prevention

The main aim of the deadlock prevention method is to violate any one condition among the four; because if any of one condition is violated then the problem of deadlock will never occur. As the idea behind this method is simple but the difficulty can occur during the physical implementation of this method in the system.

Deadlock Avoidance

This method is used by the operating system in order to check whether the system is in a safe state or in an unsafe state. This method checks every step performed by the operating system. Any process continues its execution until the system is in a safe state. Once the system enters into an unsafe state, the operating system has to take a step back.

Deadlock detection and recovery

We will wait till it occur then detect and try to recover it

Deadlock Ignorance

Ignore the Deadlock

6.5 Deadlock Prevention

we need to violate at least one condition from 4 required conditions

mutual exclusion cant be violated because that's based on the hardware

Hold and Wait :

- conservative approach : process is allowed to start if and only if it has acquired all the resources (less efficient)
- do not hold : process will acquire only desired resources but before making any fresh request it must release all the previous resources that its holding
- do not wait : we set fixed amount till which it will wait after which it releases all the resources

Preemption : we allow a process to forcefully preempt the resource hold by other process , this can be used only in very high priority processes

Circular wait: Assign a priority number to each resource. There will be a condition that any process cannot request for a lesser priority resource. This method ensures that not a single process can request a resource that is being utilized by any other process and due to which no cycle will be formed.
Example: Assume that R5 resource is allocated to P1, if next time P1 asks for R4, R3 that are lesser than R5; then such request will not be granted. Only the request for resources that are more than R5 will be granted.

problems with prevention:

Puts different types of restrictions or conditions on the processes and resources and because of which system becomes slow and reduced system throughput

6.6 Deadlock Avoidance

For deadlock avoidance we keep on checking if we perform an operation whether it will be in safe state or not , we will only perform if it will be in safe state

Safe state is checked using bankers algorithm

Reference for Bankers Algorithm

6.7 Deadlock Detection and Recovery

Here we do not previously check safety and whenever any process request for some resources they are allocated immediately if available so there is a possibility of deadlock which needs to be detected using 2 methods

1. Active method
    a) in this we run the checking algorithm at regular intervals
2. Lazy method
    a) in this we run the checking algorithm when system performance is degraded

Recovery from deadlock

When a detection algorithm determines that a deadlock exists then there are several available alternatives. There one possibility and that is to inform the operator about the deadlock and let him deal with this problem manually.

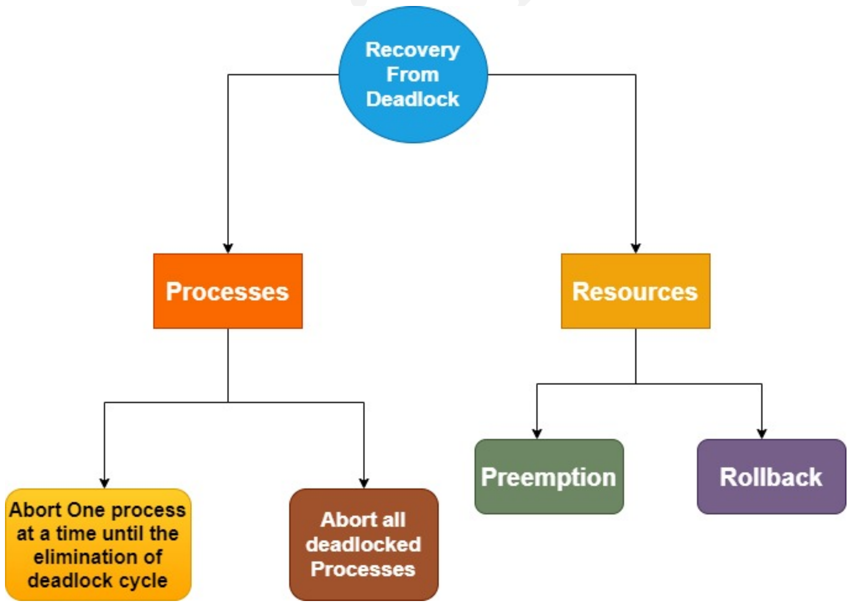

Recovery From Deadlock
Processes
Resources
Abort One process at a time until the elimination of deadlock cycle
Abort all deadlocked Processes
Preemption
Rollback

Figure 6.1: Deadlock Recovery (src:https://www.studytonight.com/)

Chapter 7

Memory Management

7.1 Introduction to Memory Management

The task of subdividing the memory among different processes is called memory management.

To achieve a degree of multiprogramming and proper utilization of memory, memory management is important.

Main Memory refers to a physical memory that is the internal memory to the computer.Main memory is also known as RAM.The computer is able to change only data that is in main memory. Therefore, every program we execute and every file we access must be copied from a storage device into main memory.

7.2 Few important terminologies

Internal fragmentation

when more space is allotted than required then is said to be internal fragmentation

External fragmentation

External fragmentation happens when there's a sufficient quantity of area within the memory to satisfy the memory request of a method. However, the process's memory request cannot be fulfilled because the memory offered is in a non-contiguous manner

7.3 Memory Management Techniques

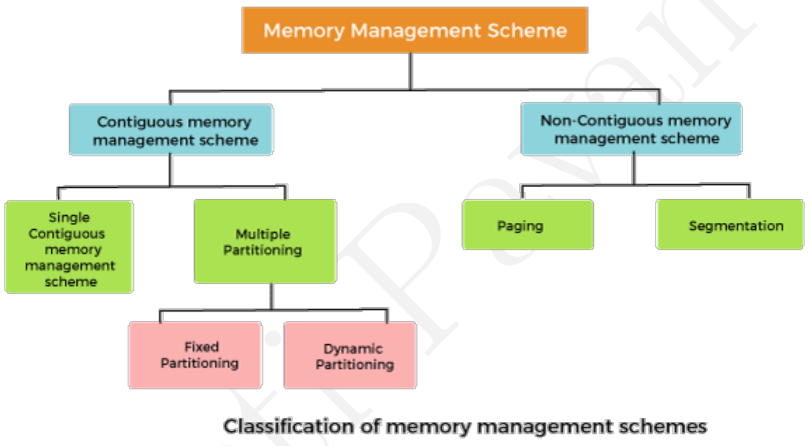

Memory Management Scheme
Contiguous memory management scheme
Non-Contiguous memory management scheme
Single Contiguous memory management scheme
Multiple Partitioning
Paging
Segmentation
Fixed Partitioning
Dynamic Partitioning
Classification of memory management schemes

Figure 7.1: Memory Management Techniques (java-t-point)

7.4 Continuous Memory Allocation

Fixed Partitioning (static partitioning)

1. Number of partitions is fixed
2. But size of each partition can be same or not
3. Continuous allocation , so spanning is not allowed
4. Internal Fragmentation and external fragmentation is there

5. There is limitation on number of process

Variable sized partitioning

1. We will assign size only when some process comes for execution
2. There is no internal fragmentation , external fragmentation still exists as when some process execution is completed and left the ram , we have enough space in non continuous manner
3. Degree of multi-programming is increased

Memory allocation algorithms

First fit:

First location that is big enough is allocated

Next fit:

Same as first fit but starts searching from last allocated hole

Best fit:

Searches for smallest hole which is big enough to allocate to process

Worst fit:

Searches for largest hole which is big enough to allocate to process

7.5 Non-Contiguous Memory Allocation

Need for paging

In Non-Contiguous Memory Allocation holes are generated dynamically and and not continuous so we divide each process into parts in secondary memory itself called as page and RAM is also divided into parts called frame , **Size of page is equal to size of frame**

7.6 Paging

We have a process which is divided into pages, we have ram which is divided into frames

Size of page = process size / number of pages
Size of frame = = Size of page
no of frames = RAM size / Size of frame

CPU don't know where is nth byte of a process is stored in the main memory , it can be anywhere so we will need a mapping table called page table , this is done by MMU (memory management unit) , every process has its own page table.

Mapping is done from page to frame , CPU depends on logical address and it consists of 2 things , page number and page offset and both of them are mentioned in binary format, physical address also contains 2 things frame number and frame offset and page table maps logical address to physical address

Page table entry in page table

1. Frame number : It gives the frame number in which the current page you are looking for is present.

2. Present/absent bit (0/1) : Present or absent bit says whether a particular page you are looking for is present or absent.

3. Protection bit : Protection bit says that what kind of protection you want on that page (read, write etc).

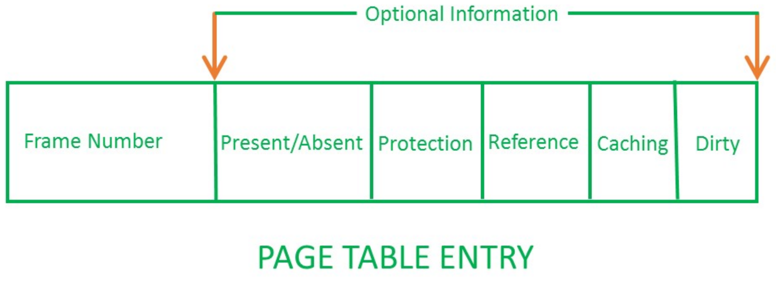


Figure 7.2: Page table entry (geeksforgeeks.com)

4. Referenced bit : Referenced bit will say whether this page has been referred in the last clock cycle or not.

5. Cache on or off : Says whether to store cache or not

6. Modified bit : Modified bit says whether the page has been modified or not. If a page is modified, then whenever you should replace that page with some other page, then the modified information should be kept on the hard disk. It is set to 1 by hardware on write-access to page which is used to avoid writing when swapped out. Sometimes this modified bit is also called as the Dirty bit.

Multi Level Paging

when page table size is greater than frame size then we use this technique , Multilevel Paging is a paging scheme that consists of two or more levels of page tables in a hierarchical manner. It is also known as hierarchical paging. The entries of the level 1 page table are pointers to a level 2 page table and entries of the level 2 page tables are pointers to a level 3 page table and so on. The entries of the last level page table store actual frame information.

reference : Gate Smashers Video

Page table entry in page table

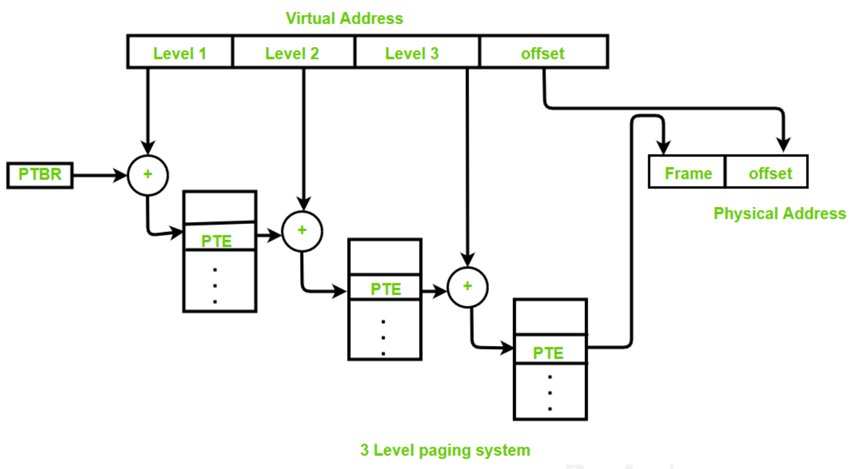


Figure 7.3: multi level paging (geeksforgeeks.com)

Inverted Paging

In normal paging , if at least one page of a process is in main memory then its page table also needs to be in main memory , so we will use inverted paging to reduce space usage by maintaining a global page table which looks like this.

In normal page table there is mapping from page number of each process to frame , but in inverted page table we will have mapping from frame number to page number and process number

7.7 Thrashing

Page fault

For example we have one page of all processes in CPU and multi-programming is maximised , now if cpu requests for page that is not in CPU this is called page fault when page fault occurs it takes time to bring the required page from secondary memory , this consumes time and hence reduces the CPU utilisation , this is called as thrashing

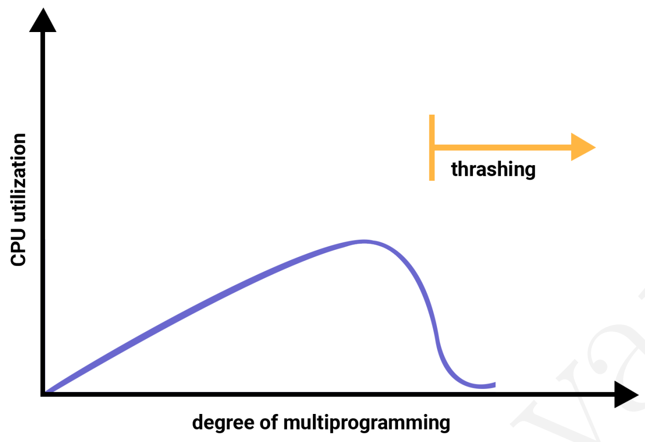


Figure 7.4: Thrashing (studytonight.com)

7.8 Segmentation

unlike paging , in segmentation size of each block is not fixed , process is divided into small segments of different sizes and are kept in main memory , using this we are putting related data together

1. logical address consists of 2 fields , s and d (segment name and offset respectively)

2. s is the index that it has to check in segment table

3. segment table consists of base address and size of that segment in main memory, and d represents the size that needs to be accessed

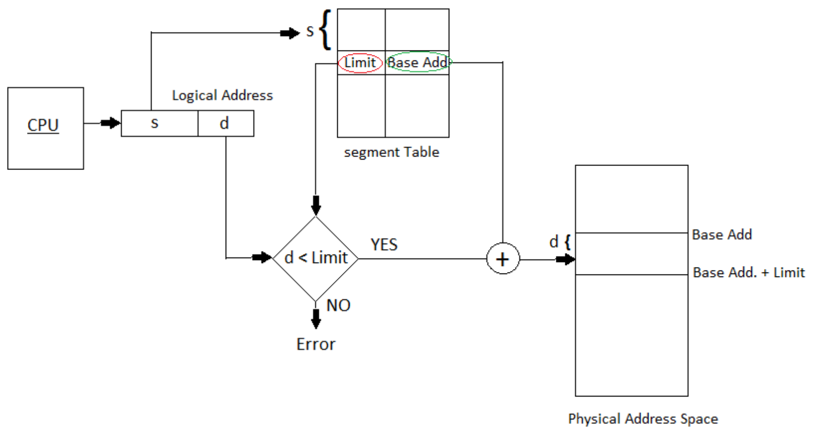
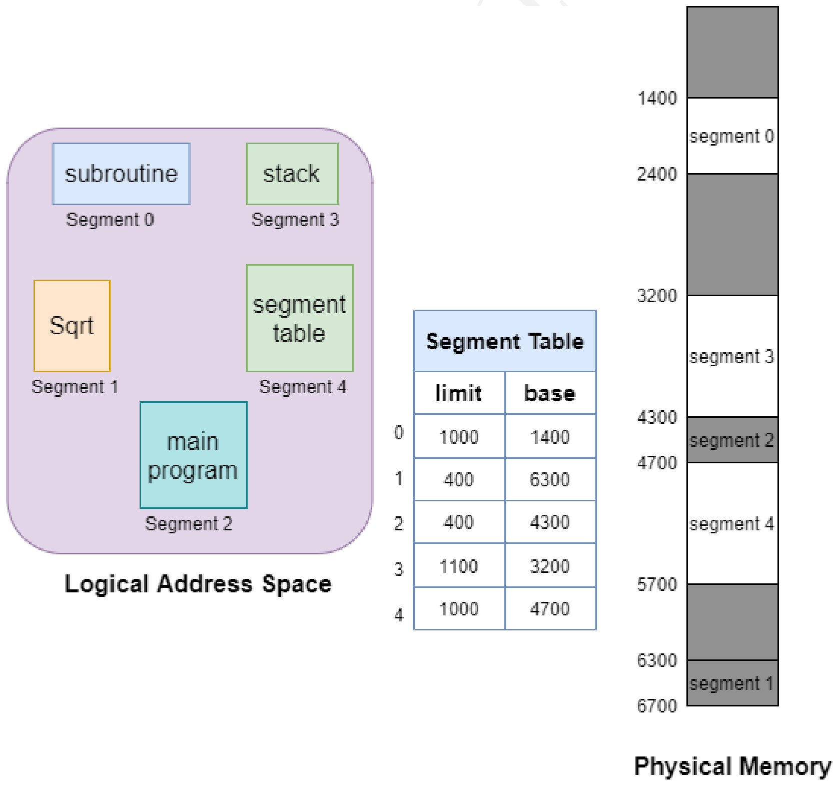


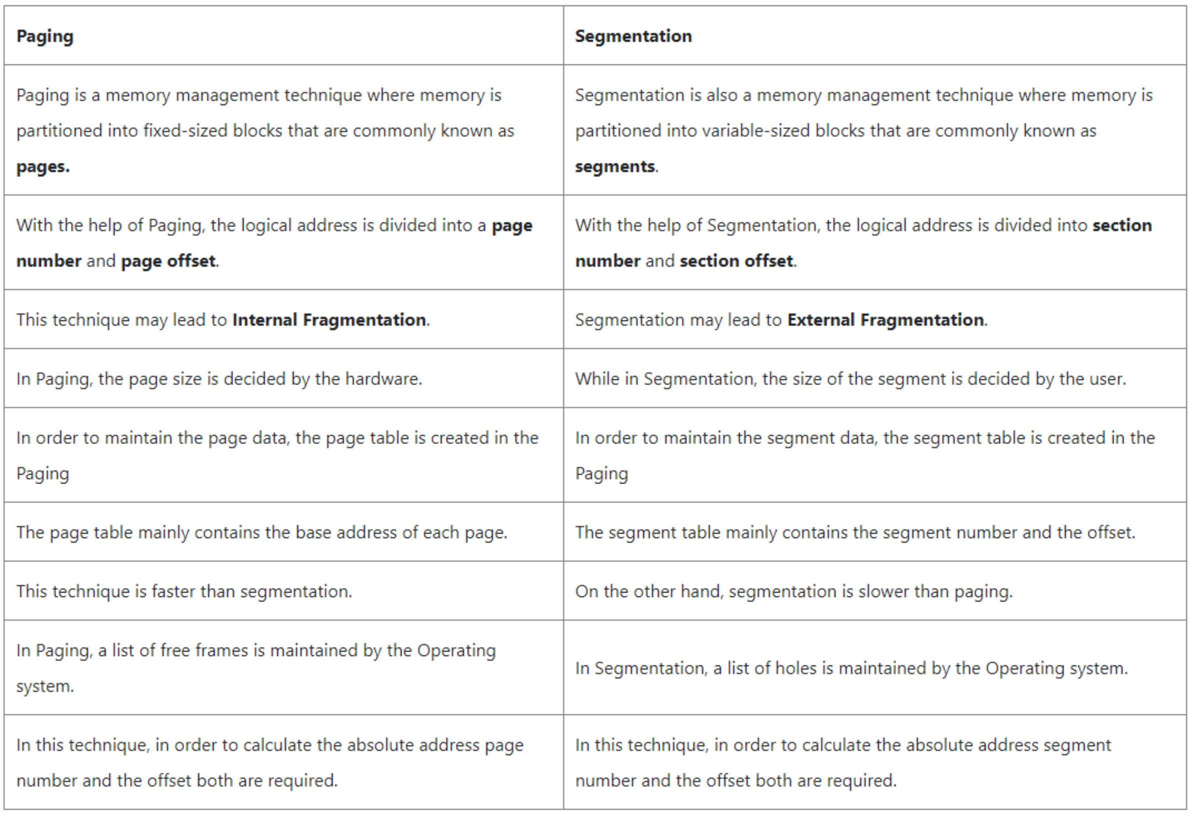
Figure 7.5: segmentation (geeksforgeeks.com)



Segment Table		
	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Figure 7.6: Logical address to Physical Address (studytonight.com)

7.9 Paging vs Segmentation



Paging	**Segmentation**
Paging is a memory management technique where memory is partitioned into fixed-sized blocks that are commonly known as **pages.**	Segmentation is also a memory management technique where memory is partitioned into variable-sized blocks that are commonly known as **segments**.
With the help of Paging, the logical address is divided into a **page number** and **page offset**.	With the help of Segmentation, the logical address is divided into **section number** and **section offset**.
This technique may lead to **Internal Fragmentation**.	Segmentation may lead to **External Fragmentation**.
In Paging, the page size is decided by the hardware.	While in Segmentation, the size of the segment is decided by the user.
In order to maintain the page data, the page table is created in the Paging	In order to maintain the segment data, the segment table is created in the Paging
The page table mainly contains the base address of each page.	The segment table mainly contains the segment number and the offset.
This technique is faster than segmentation.	On the other hand, segmentation is slower than paging.
In Paging, a list of free frames is maintained by the Operating system.	In Segmentation, a list of holes is maintained by the Operating system.
In this technique, in order to calculate the absolute address page number and the offset both are required.	In this technique, in order to calculate the absolute address segment number and the offset both are required.

Figure 7.7: Paging vs Segmentation (studytonight.com)

Chapter 8

Virtual Memory Management

Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of the main memory. It gives an illusion that even if the process size is greater than main memory size , it can still be executed. In this instead of bringing all pages of a process to main memory , We will bring required page along with few more pages , this is called **Demand Paging (paging + swapping)**

8.1 Page Fault

1. If the page that CPU requires is not present in main memory then it is called as page fault.
2. When page fault occurs , a trap signal is generated
3. Then the page is leaded from secondary memory into the main memory and it is updated in the page table
4. And then the CPU instruction is executed
5. The diagram for above steps is shown below
6. P = no of page faults
7. Memory Access time = (P)*(page fault service time)+(1-P)*(main memory access time)

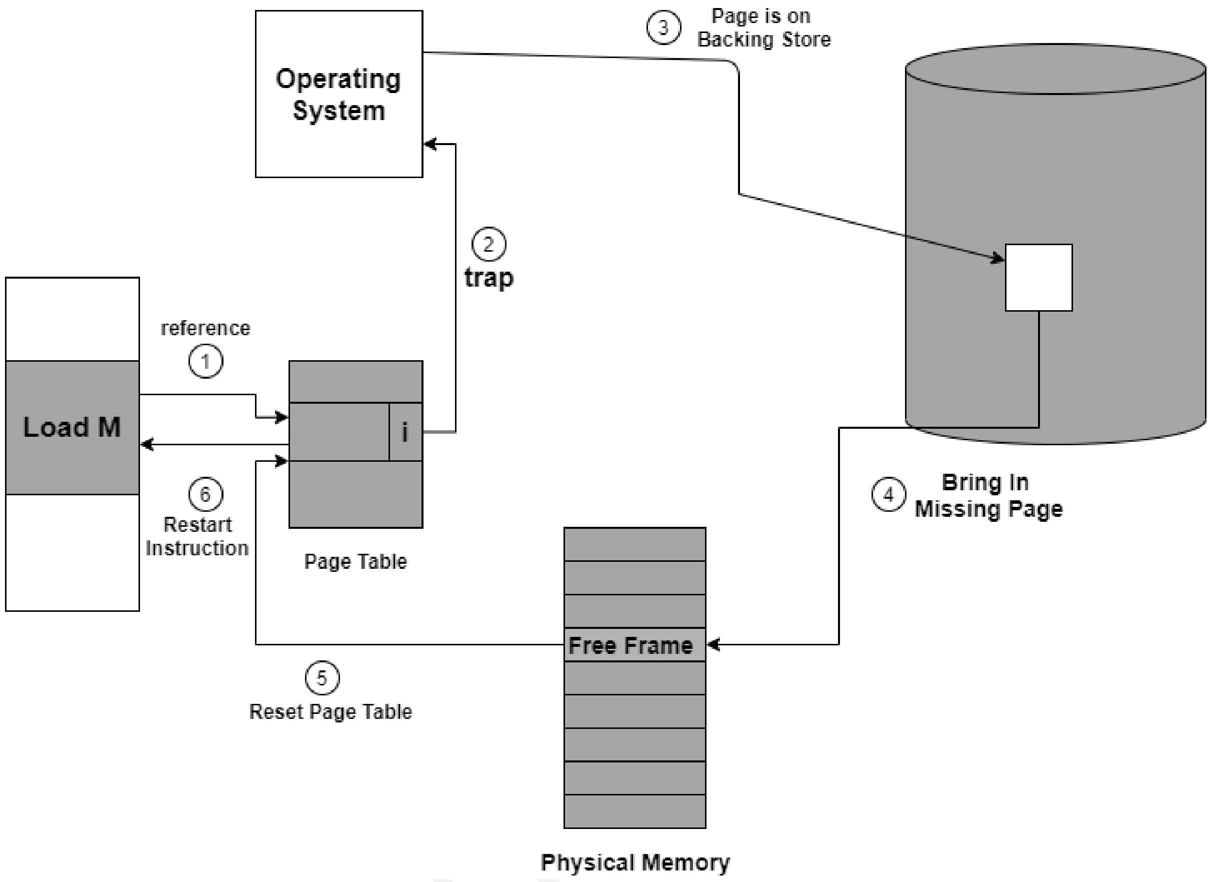


Figure 8.1: Page fault (studytonight.com)

8.2 Advantages of Virtual memory

1. User can write program for extremely large virtual address space
2. (CPU Utilisation and Throughput) increases (response time and turn around time remains same)
3. Less I/O would be needed to load or swap user programs into memory , so each user program would run faster
4. Degree of multi-programming increases

8.3 Translation look aside buffer (TLB)(Cache)

A translation look-aside buffer (TLB) is a memory cache that stores recent translations of virtual memory to physical addresses for faster retrieval. When a virtual memory address is referenced by a program, the search starts in the CPU.

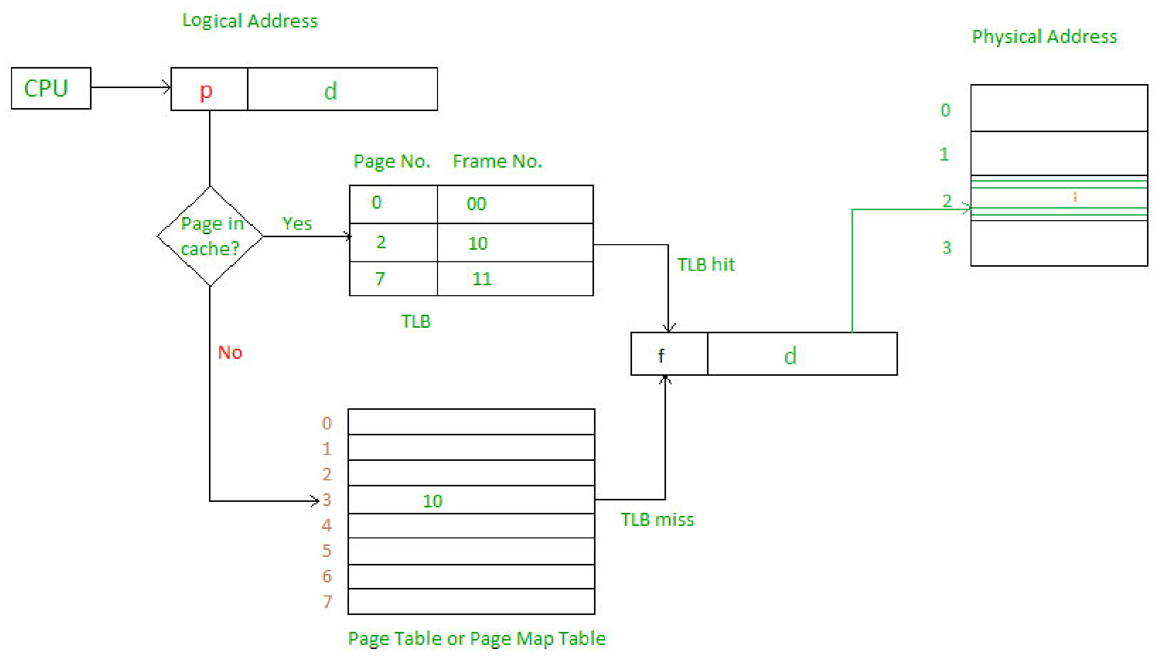


Figure 8.2: TLB functioning (studytonight.com)

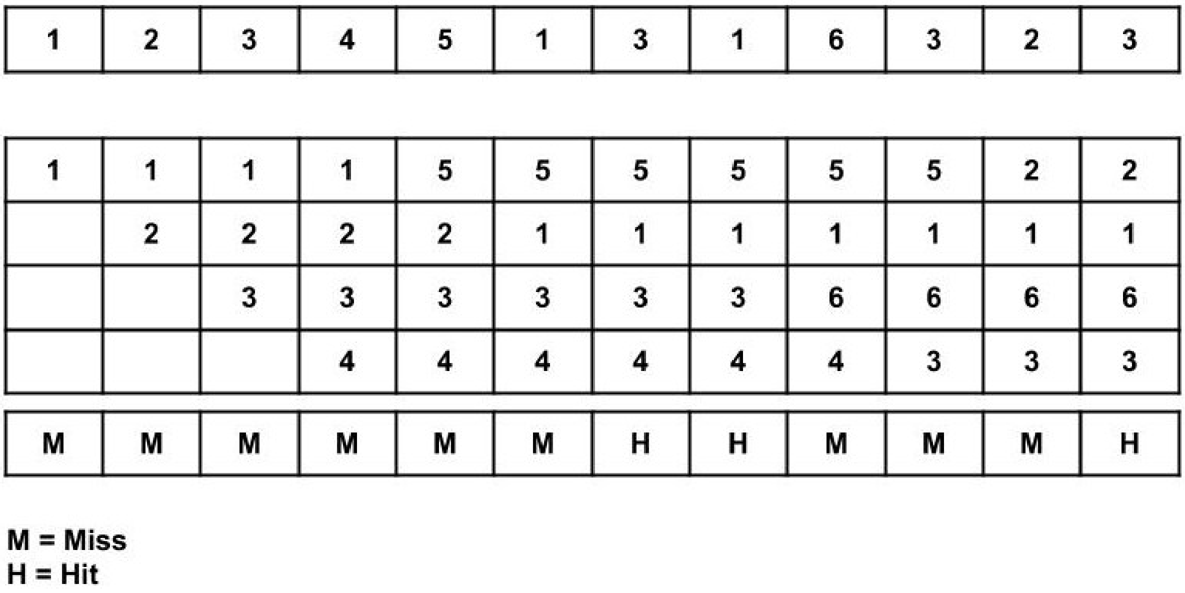
1. we call it a TLB hit when the requested address is present in TLB
2. we call it a TLB miss when the requested address is not found inside the TLB

8.4 Page Replacement Algorithms

reference

FIFO

when page fault occurs we will bring a page from secondary memory,then if main memory is full we will swap out the first entered page and swap in the new page , if already the requested page is in main memory , we call it a hit , else we call it miss



1	2	3	4	5	1	3	1	6	3	2	3

1	1	1	1	5	5	5	5	5	5	2	2
	2	2	2	2	1	1	1	1	1	1	1
		3	3	3	3	3	3	6	6	6	6
			4	4	4	4	4	4	3	3	3

M	M	M	M	M	M	H	H	M	M	M	H

M = Miss
H = Hit

Figure 8.3: FIFO page replacement algorithm (afteracademy.com)

Belady’s anomaly

when we increase the no of frames we expect hits will be increased and faults will be decreased but its not true always, when faults increased its called belady’s anomaly

Optimal Page Replacement Algorithm

Replace the page which is used last in the future, check after how much time each page is called , pick the one which is going to be called at last and replace that page

LRU (Least Recently Used)

Replace the least recently used page in the past

8.5 Thrashing

Definition:

Thrashing is a condition or a situation when the system is spending a major portion of its time servicing the page faults, but the actual processing done is very negligible.

When the CPU's usage is low, the process scheduling mechanism tries to load multiple processes into memory at the same time, increasing the degree of Multi programming. In this case, the number of processes in the memory exceeds the number of frames available in the memory. Each process is given a set number of frames to work with.so when the high priority process arrives and has no frames left to occupy it will do swapping , the procedure begins to take a long time to swap in the required pages. Because most of the processes are waiting for pages, the CPU utilization drops again.

Handling Thrashing

1. Use priority based replacement algorithm

2. Allocate the exact number of frames that are actually required