**DHANEKULA INSTITUTE OF ENGINEERING & TECHNOLOGY**

**(Approved by AICTE, Affiliated by JNTU, Kakinada)**

**GANGURU: VIJAYAWADA – 521 139**

**Department of CSE-AI&ML**

**Set No: II**

Name of the program: B.Techin Computer Science &Engineering

Name of the Course:Operating System

Year &Semester:III-I

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Q No** | **1** | **2** | **3.a** | **3.b** | **3.c** |
| **Marks** | 5 | 5 | 1 | 1 | 3 |
| **CO** | **R20C203.1** | **R20C203.2** | **R20C203.1**  **R20C203.2** | | **R20C203.3** |
| **Tot** | 5 | 5 | 2 | | 3 | 15 |

Name of the Exam: Mid-I

Academic year: 2023-24

Subject Code: R20C203

JNTUK Subject Code:R2021052

Exam Weightage:100%

Duration: 90 Min

Date: 20 /09/2023

|  |  |
| --- | --- |
| Course Outcome Number | Course Outcome |
| R20C203.1 | Describe various generations of Operating System and functions of Operating System |
| R20C203.2 | Analyze scheduling algorithms and inter-process communication methods of processes handled by operating systems through examples. |
| R20C203.3 | Distinguish various memory management strategies such as paging and segmentation, virtual memory, swapping and page replacement algorithms |

--------------------------------------------------------------------------------------------------------------

1. Explain Operating System Structures and Operating System Debugging with necessary example.

**[BTL2,3Understanding,AnalyzePO1, 2, 3, 4, 12/PSO 1,2 --- 5M]**

2. Analyze the concept of communication in client server systems with example and Dining philosophers problem with example

**[BTL2,Understanding, Analyze PO1, 2, 3, 4, 12/PSO 1,2 --- 5M]**

3. a) What is System boot?– 1M

b) Define Semaphores and Monitors with examples? – 1M

c) Explain Memory Management with example. -3M

**[BTL2, Understanding, PO1, 2, 3, 4, 12/PSO 1,2 --- 5M]**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Explain Operating System Structures and Operating System Debugging with necessary example.  **[BTL3, Applying, PO1/PSO 1,2 --- 5M]**

An operating system is a design that enables user application programs to communicate with the hardware of the machine. The operating system should be built with the utmost care because it is such a complicated structure and should be simple to use and modify. Partially developing the operating system is a simple approach to accomplish this. Each of these components needs to have distinct inputs, outputs, and functionalities.

Simple Structure

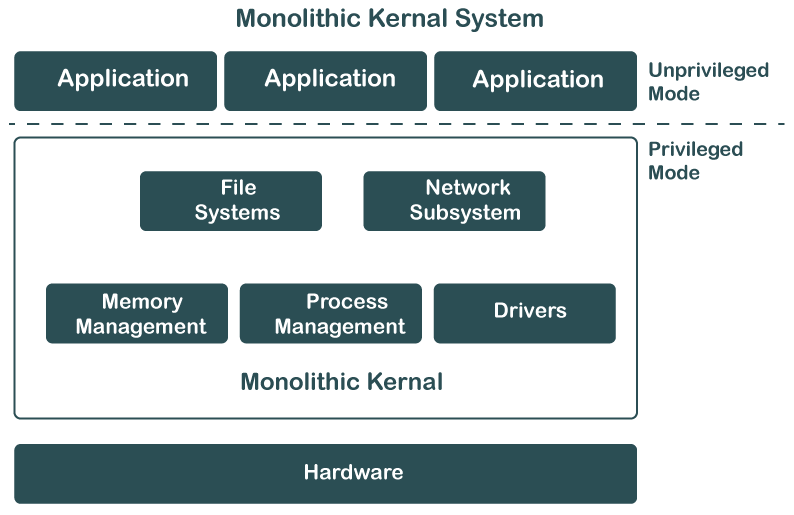
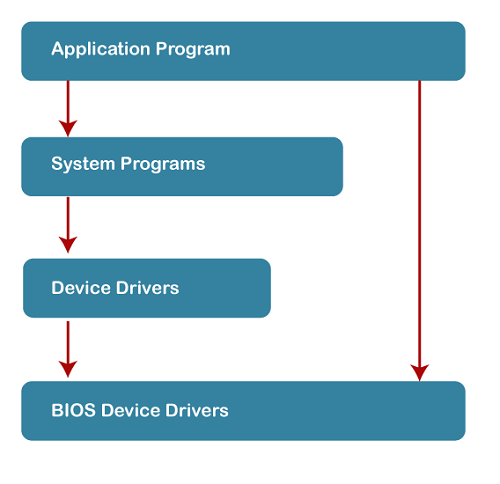
Monolithic Structure

Layered Approach Structure

Micro-Kernel Structure

Exo-Kernel Structure

Virtual Machines



Debugging is the process of finding the problems in a computer system and solving them. There are many different ways in which operating systems perform debugging. Some of these are −

**Log Files**

The log files record all the events that occur in an operating system. This is done by writing all the messages into a log file. There are different types of log files. Some of these are given as follows −

**Event Logs**

These stores the records of all the events that occur in the execution of a system. This is done so that the activities of all the events can be understood to diagnose problems.

**Transaction Logs**

The transaction logs store the changes to the data so that the system can recover from crashes and other errors. These logs are readable by a human.

**Message Logs**

These logs store both the public and private messages between the users. They are mostly plain text files, but in some cases they may be HTML files.

**Core Dump Files**

The core dump files contain the memory address space of a process that terminates unexpectedly. The creation of the core dump is triggered in response to program crashes by the kernel. The core dump files are used by the developers to find the program’s state at the time of its termination so that they can find out why the termination occurred.

The automatic creation of the core dump files can be disabled by the users. This may be done to improve performance, clear disk space or increase security.

**Crash Dump Files**

In the event of a total system failure, the information about the state of the operating system is captured in crash dump files. There are three types of dump that can be captured when a system crashes. These are −

**Complete Memory Dump**

The whole contents of the physical memory at the time of the system crash are captured in the complete memory dump. This is the default setting on the Windows Server System.

**Kernel Memory Dump**

Only the kernel mode read and write pages that are present in the main memory at the time of the system crash are stored in the kernel memory dump.

**Small Memory Dump**

This memory dump contains the list of device drivers, stop code, process and thread information, kernel stack etc.

**Trace Listings**

The trace listing record information about a program execution using logging. This information is used by programmers for debugging. System administrators and technical personnel can use the trace listings to find the common problems with software using software monitoring tools.

**Profiling**

This is a type of program analysis that measures various parameters in a program such as space and time complexity, frequency and duration of function calls, usage of specific instructions etc. Profiling is done by monitoring the source code of the required system program using a code profiler.

2.Analyze the concept of communication in client server systems with example and Dining philosophers problem with example**BTL2,Understanding, Analyze PO1, 2, 3, 4, 12/PSO 1,2 --- 5M]**

The Dining Philosopher Problem states that K philosophers are seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pick up the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.

**Semaphore Solution to Dining Philosopher**

Each philosopher is represented by the following pseudocode:

process P[i]

while true do

{ THINK;

PICKUP(CHOPSTICK[i], CHOPSTICK[i+1 mod 5]);

EAT;

PUTDOWN(CHOPSTICK[i], CHOPSTICK[i+1 mod 5])

}

There are three states of the philosopher: **THINKING, HUNGRY, and EATING**. Here there are two semaphores: Mutex and a semaphore array for the philosophers. Mutex is used such that no two philosophers may access the pickup or put it down at the same time. The array is used to control the behavior of each philosopher. But, semaphores can result in deadlock due to programming errors.

**Outline of a philosopher process:**

The Dining Philosopher Problem is a classic synchronization problem in computer science that involves multiple processes (philosophers) sharing a limited set of resources (forks) in order to perform a task (eating). In order to avoid deadlock or starvation, a solution must be implemented that ensures that each philosopher can access the resources they need to perform their task without interference from other philosophers.

One common solution to the Dining Philosopher Problem uses semaphores, a synchronization mechanism that can be used to control access to shared resources. In this solution, each fork is represented by a semaphore, and a philosopher must acquire both the semaphore for the fork to their left and the semaphore for the fork to their right before they can begin eating. If a philosopher cannot acquire both semaphores, they must wait until they become available.

### The steps for the Dining Philosopher Problem solution using semaphores are as follows

1. Initialize the semaphores for each fork to 1 (indicating that they are available).

2. Initialize a binary semaphore (mutex) to 1 to ensure that only one philosopher can attempt to pick up a fork at a time.

3. For each philosopher process, create a separate thread that executes the following code:

* While true:
  + Think for a random amount of time.
  + Acquire the mutex semaphore to ensure that only one philosopher can attempt to pick up a fork at a time.
  + Attempt to acquire the semaphore for the fork to the left.
* If successful, attempt to acquire the semaphore for the fork to the right.
* If both forks are acquired successfully, eat for a random amount of time and then release both semaphores.
* If not successful in acquiring both forks, release the semaphore for the fork to the left (if acquired) and then release the mutex semaphore and go back to thinking.

4. Run the philosopher threads concurrently.

By using semaphores to control access to the forks, the Dining Philosopher Problem can be solved in a way that avoids deadlock and starvation. The use of the mutex semaphore ensures that only one philosopher can attempt to pick up a fork at a time, while the use of the fork semaphores ensures that a philosopher can only eat if both forks are available.

3. a) What is System boot?– 1M**[BTL2, Understanding, PO1, 2, 3, 4, 12/PSO 1,2 --- 5M]**

b) Define Semaphores and Monitors with examples? – 1M

c) Explain Memory Management with example. -3M

**System boot**

Bootstrapping, commonly known as booting, is a crucial procedure that allows an operating system to load and initialize on a computer. It involves turning on the computer, loading the operating system into the main memory, and preparing it to accept user commands.

b) Define Semaphores and Monitors with examples?

A semaphore is an integer variable that allows many processes in a parallel system to manage access to a common resource like a multitasking OS. On the other hand, a monitor is a synchronization technique that enables threads to mutual exclusion and the wait() for a given condition to become true.

c) Explain Memory Management with example

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

Process Address Space

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

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

****

**DHANEKULA INSTITUTE OF ENGINEERING & TECHNOLOGY**

**(Approved by AICTE, Affiliated by JNTU, Kakinada)**

**GANGURU: VIJAYAWADA – 521 139**

**Department of Computer Science&Engineering**

**Set No: I**

Name of the program : B.Techin Computer Science &Engineering

Name of the Course :Operating System

Year &Semester :II-I

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Q No** | **1** | **2** | **3.a** | **3.b** | **3.c** |
| **Marks** | 5 | 5 | 1 | 1 | 3 |
| **CO** | **R20C203.4** | **R20C203.5** | **R20C203.4** | **R20C203.5** | **R20C203.3** |
| **Tot** | 5 | 5 | 1 | 1 | 3 | 15 |

Name of the Exam : Mid-II

Academic year: 2023-24

Subject Code: R20C302

Exam Weightage :100%

Duration : 90 Min

Date : 28/11/2023

Name of the Faculty :

|  |  |  |
| --- | --- | --- |
| Course Outcome Number | Course Outcome | Blooms Taxonomy Levels |
| R20C203.4 | Solve variousmemorymanagementstrategiessuchas paging andsegmentation, virtual memory,swappingandpagereplacement algorithms. | Applying(BTL3) |
| R20C203.5 | Identifydeadlockdetection,recovery,preventionandavoidance algorithms along with thediskstructure,diskscheduling andstorage implementation. | Applying(BTL3) |
| R20C203.6 | Listvarioussecuritymeasuresandsystemprotectiontechniques | Analyze(BTL4) |

--------------------------------------------------------------------------------------------------------------

Answer all the Questions

1. Apply Bankers Algorithm for Deadlock Prevention with suitable example?

**[BTL3,Applying, PO1/PSO 1,2 --- 5M]**

1. Explain Program, System and network threats?

**[BTL2, Understanding,PO 2/PSO 1,2 --- 5M]**

1. a)Explain necessary Conditions for resource deadlocks?

**[BTL2, Understanding, PO1/PSO 1,2 --- 1M]**

b) Explain goals of protection?

**[BTL2, Understanding,PO 2/PSO 1,2 --- 1M]**

c) What is Demand paging and Apply different types of Page replacement algorithms with example?

**[BTL3, Applying, PO1/PSO 1,2 ---3M]**

**Scheme of Evaluation**

1) Bankers Algorithm - 3M

Safety algorithm – 2M **[BTL2,Understanding,PO 2/PSO 1,2]**

2)Program threats -1M

System threats – 2M

Network threats- 2M **[BTL3, Applying,PO1/PSO 1,2]**

3) a)Conditions for resource deadlocks – 1M **[BTL2,Understanding,PO1/PSO 1,2]**

b)Goals of protection -1M **[BTL2,Understanding,PO 2/PSO 1,2]**

c) Demand paging -1M

Types of Page replacement algorithms – 2M **[BTL3, Applying, PO1/PSO 1,2]**

1. Apply Bankers Algorithm for Deadlock Prevention with suitable example?

**[BTL3, Applying, PO1/PSO 1,2 --- 5M]**

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that simulates resource allocation for predetermined maximum possible amounts of all resources before performing an "s-state" check to look for potential activities and determining whether allocation should be permitted to continue.

The Banker's Algorithm is implemented using the following data structures:

Let n be the number of processes in the system and m be the number of resource kinds.

**Available:**

This 1-d array of size 'm' lists the number of resources of each category that are currently available.

There are 'k' instances of the resource type if Available[j] = Rj

**Max:**

The maximum demand of each process in a system is specified by a 2-d array of size 'n\*m'.

Process Pi may request a maximum of 'k' instances of resource type Rj if Max[i, j] = k.

**Allocation:**

The quantity of resources of each kind currently assigned to each process is specified by a 2-d array of size 'n\*m'.

Process is shown by Allocation[i, j] = k. Pi has been given 'k' instances of the resource type at this time. Rj

**Need:**

The remaining resource needs of each process are shown in a 2-d array of size 'n\*m'.

Process is shown by Need[i, j] = k. Right now, Pi requires "k" instances of the resource type. Rj

Allocation[i, j] - Maximum[i, j] = Need[i, j]

The resources that are now allotted to process Pi are identified by Allocationi, while the extra resources that process Pi could yet need in order to do its work are identified by Needi.

The safety algorithm and the resource request algorithm make up the banker's algorithm.

**Algorithm for Safety**

The following is a description of the method for determining if a system is in a safe state:

1) Assume Work and Finish are two vectors, each with lengths of m and n.

Initialize: Work = Available

Finish[i] is **false** when i=1, 2, 3, 4...n

2) Find an I such that both

a) Finish[i] = **false**

b) Needi <= Work

**if** such an I does not exist. **goto**  step (4)

3) Work = Work + Allocation[i]

Finish[i] = **true**

go to step (2)

4) If Finish[i] = **true** **for** each and every i

then system is in a secure state.

**Algorithm for Resource Requests**

Let Requesti represent the process Pi request array. Process Pi requests k instances of resource type Rj, which is indicated by Requesti [j] = k. The following things happen when process Pi makes a request for resources:

1) Proceed to step 2 **if** Requesti >= Needi; otherwise, report an error condition because the process has made more claims than it can handle.

2) Proceed to step (3) **if** Requesti <= Accessible; otherwise, Pi will have to wait since the resources are not available.

3) Change the state in such a way that the system appears to have given Pi the required resources:

Requested - Available = Available

Allocationi = Allocationi + Requesti

Needi = Needi - Requesti

1. Explain Program, System and network threats?**[BTL2, Understanding,PO 2/PSO 1,2 --- 5M]**

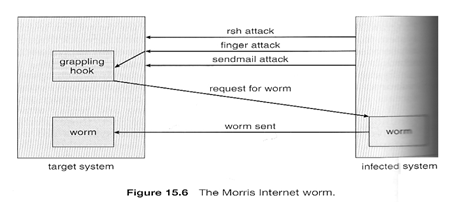
**Introduction:-**Most of the threats described above are termed program threats, because they attack specific programs or are carried and distributed in programs. The threats in this section attack the operating system or the network itself, or leverage those systems to launch their attacks.

**Worms:**-A worm is a process that uses the fork / spawn process to make copies of itself in order to wreak havoc on a system. Worms consume system resources, often blocking out other, legitimate processes. Worms that propagate over networks can be especially problematic, as they can tie up vast amounts of network resources and bring down large-scale systems.

This worm consisted of two parts:-

1. A small program called a grappling hook, which was deposited on the target system through one of three vulnerabilities, and

2. The main worm program, which was transferred onto the target system and launched by the grappling hook program.



The three vulnerabilities exploited by the Morris Internet worm were as follows:

1. rsh ( remote shell ) is a utility that was in common use at that time for accessing remote systems without having to provide a password. If a user had an account on two different computers ( with the same account name on both systems ), then the system could be configured to allow that user to remotely connect from one system to the other without having to provide a password. Many systems were configured so that any user (except root) on system A could access the same account on system B without providing a password.

2. finger is a utility that allows one to remotely query a user database, to find the true name and other information for a given account name on a given system. For example "finger joeUser@somemachine.edu" would access the finger daemon at somemachine.edu and return information regarding joeUser. Unfortunately the finger daemon ( which ran with system privileges ) had the buffer overflow problem, so by sending a special 536-character user name the worm was able to fork a shell on the remote system running with root privileges.

3. sendmail is a routine for sending and forwarding mail that also included a debugging option for verifying and testing the system. The debug feature was convenient for administrators, and was often left turned on. The Morris worm exploited the debugger to mail and execute a copy of the grappling hook program on the remote system.

Once in place, the worm undertook systematic attacks to discover user passwords:

* First it would check for accounts for which the account name and the password were the same, such as "guest", "guest".
* Then it would try an internal dictionary of 432 favorite password choices.
* Finally it would try every word in the standard UNIX on-line dictionary to try and break into user accounts.

Once it had gotten access to one or more user accounts, then it would attempt to use those accounts to rsh to other systems, and continue the process.

With each new access, the worm would check for already running copies of itself, and 6 out of 7 times if it found one it would stop.

Fortunately the same rapid network connectivity that allowed the worm to propagate so quickly also quickly led to its demise - Within 24 hours remedies for stopping the worm propagated through the Internet from administrator to administrator, and the worm was quickly shut down.

**Port Scanning**

Port Scanning is technically not an attack, but rather a search for vulnerabilities to attack. The basic idea is to systematically attempt to connect to every known ( or common or possible ) network port on some remote machine, and to attempt to make contact. Once it is determined that a particular computer is listening to a particular port, then the next step is to determine what daemon is listening, and whether or not it is a version containing a known security flaw that can be exploited.

  Because port scanning is easily detected and traced, it is usually launched from zombie systems, i.e. previously hacked systems that are being used without the knowledge or permission of their rightful owner. For this reason it is important to protect "innocuous" systems and accounts as well as those that contain sensitive information or special privileges.

  There are also port scanners available that administrators can use to check their own systems, which report any weaknesses found but which do not exploit the weaknesses or cause any problems. Two such systems are nmap ( http://www.insecure.org/nmap ) and nessus ( http://www.nessus.org ). The former identifies what OS is found, what firewalls are in place, and what services are listening to what ports. The latter also contains a database of known security holes, and identifies any that it finds.

**Denial of Service**

Denial of Service (DOS) attacks do not attempt to actually access or damage systems, but merely to clog them up so badly that they cannot be used for any useful work. Tight loops that repeatedly request system services are an obvious form of this attack. DOS attacks can also involve social engineering, such as the Internet chain letters that say "send this immediately to 10 of your friends, and then go to a certain URL", which clogs up not only the Internet mail system but also the web server to which everyone is directed. ( Note: Sending a "reply all" to such a message notifying everyone that it was just a hoax also clogs up the Internet mail service, just as effectively as if you had forwarded the thing. )Security systems that lock accounts after a certain number of failed login attempts are subject to DOS attacks which repeatedly attempt logins to all accounts with invalid passwords strictly in order to lock up all accounts.

3a) Explain necessary Conditions for resource deadlocks?

**[BTL2, Understanding, PO1/PSO 1,2 --- 1M]**

b) Explain goals of protection?

**[BTL2, Understanding,PO 2/PSO 1,2 --- 1M]**

c) What is Demand paging and Apply different types of Page replacement algorithms with example?

here are four conditions because of which Deadlock is going to occur in Operating Systems. The Conditions are:

Hold and Wait

No Pre Emption

Mutual Exclusion

Circular Wait

Now, let us explain about each and every Condition in detail.

**Hold and Wait**

Now, let us understand how Hold and Wait is going to cause Deadlock in Operating Systems.

Hold and Wait is process where a process stops a resource from sharing the resource to anyone or any process. Because of this holding, the processes tend to come into or enter waiting state. So, because of this waiting state the processes stop executing.

**No Pre-Emption**

When a process transitions from the running state to the ready state or from the waiting state to the ready state, Pre Emption is employed. The resources, mostly CPU cycles, are given to the process for a set period of time before being removed; if the process still has CPU burst time left, it is then put back in the ready queue. Until it has its subsequent opportunity to run, the process remains in the ready queue.

**Mutual Exclusion**

Now, let us understand what Mutual Exclusion is and how Mutual Exclusion is going to cause Deadlock in the Operating System.

Mutual Exclusion is the process by which only a single process is allowed to access the resources present in the Operating System.

If this condition is going on in the Operating System, the Hold and Wait condition occurs.

So, due to this Infinite waiting might occur and this is going to cause Deadlock to the Process Executed in the Operating System.

**Circular Wait.**

Circular Wait is the condition where the one process depends upon the other processes and cause deadlock in the Operating System.

b) Explain goals of protection?

To ensure that each shared resource is used only in accordance with system policies, which may be set either by system designers or by system administrators. To ensure that errant programs cause the minimal amount of damage possible.

c) What is Demand paging and Apply different types of Page replacement algorithms with example?

Paging is a storage mechanism. Paging is used to retrieve processes from secondary memory to primary memory.

The main memory is divided into small blocks called pages. Now, each of the pages contains the process which is retrieved into main memory and it is stored in one frame of memory.

It is very important to have pages and frames which are of equal sizes which are very useful for mapping and complete utilization of memory.

**Virtual Memory in Operating Systems (OS)**

A storage method known as virtual memory gives the user the impression that their main memory is quite large. By considering a portion of secondary memory as the main memory, this is accomplished.

By giving the user the impression that there is memory available to load the process, this approach allows them to load larger size programs than the primary memory that is accessible.

The Operating System loads the many components of several processes in the main memory as opposed to loading a single large process there.

By doing this, the level of multiprogramming will be enhanced, which will increase CPU consumption.

**Demand Paging**

The Demand Paging is a condition which is occurred in the Virtual Memory. We know that the pages of the process are stored in secondary memory. The page is brought to the main memory when required. We do not know when this requirement is going to occur. So, the pages are brought to the main memory when required by the Page Replacement Algorithms.

So, the process of calling the pages to main memory to secondary memory upon demand is known as Demand Paging.

**Frame Allocation in Virtual Memory**

Demand paging is used to implement virtual memory, an essential component of operating systems. A page-replacement mechanism and a frame allocation algorithm must be created for demand paging. If you have numerous processes, frame allocation techniques are utilized to determine how many frames to provide to each process.

A Physical Address is required by the Central Processing Unit (CPU) for the frame creation and the physical Addressing provides the actual address to the frame created. For each page a frame must be created.

**Frame Allocation Constraints**

The Frames that can be allocated cannot be greater than total number of frames.

Each process should be given a set minimum amount of frames.

When fewer frames are allocated then the page fault ration increases and the process execution becomes less efficient

There ought to be sufficient frames to accommodate all the many pages that a single instruction may refer to

**Frame Allocation Algorithms**

There are three types of Frame Allocation Algorithms in Operating Systems. They are:

**1) Equal Frame Allocation Algorithms**

Here, in this Frame Allocation Algorithm we take number of frames and number of processes at once. We divide the number of frames by number of processes. We get the number of frames we must provide for each process.

his means if we have 36 frames and 6 processes. For each process 6 frames are allocated.

It is not very logical to assign equal frames to all processes in systems with processes of different sizes. A lot of allocated but unused frames will eventually be wasted if a lot of frames are given to a little operation.

**2) Proportionate Frame Allocation Algorithms**

Here, in this Frame Allocation Algorithms we take number of frames based on the process size. For big process more number of frames is allocated. For small processes less number of frames is allocated by the operating system.

The problem in the Proportionate Frame Allocation Algorithm is number of frames are wasted in some rare cases.