Deadlock:

Overview

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.

Scope of the Article

- This article defines and explains a deadlock and the necessary conditions for a deadlock situation to occur.
- The article reflects on the various methods of handling deadlocks.
- The article lists the key differences between a deadlock and starvation.
- The article lists the advantages and disadvantages of a deadlock.

What is Deadlock in OS?

All the processes in a system require some resources such as central processing unit(CPU), file storage, input/output devices, etc to execute it. Once the execution is finished, the process releases the resource it was holding. However, when many processes run on a system they also compete for these resources they require for execution. This may arise a deadlock situation.

A deadlock 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. Therefore, none of the processes gets executed.

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 {

In the above figure, there are two processes and two resources. Process 1 holds "Resource 1" and needs "Resource 2" while Process 2 holds "Resource 2" and requires "Resource 1". This creates a situation of deadlock because none of the two processes can be executed. Since the resources are non-shareable they can only be used by one process at a time(Mutual Exclusion). Each process is holding a resource and waiting for the other process the release the resource it requires. None of the two processes releases their resources before their execution and this creates a circular wait. Therefore, all four conditions are satisfied.

Methods of Handling Deadlocks in Operating System

The first two methods are used to ensure the system never enters a deadlock.

Deadlock Prevention

This is done by restraining the ways a request can be made. Since deadlock occurs when all the above four conditions are met, we try to prevent any one of them, thus preventing a deadlock.

Deadlock Avoidance

When a process requests a resource, the deadlock avoidance algorithm examines the resource-allocation state. If allocating that resource sends the system into an unsafe state, the request is not granted.

Therefore, it requires additional information such as how many resources of each type is required by a process. If the system enters into an unsafe state, it has to take a step back to avoid deadlock.

Deadlock Detection and Recovery

We let the system fall into a deadlock and if it happens, we detect it using a detection algorithm and try to recover.

Some ways of recovery are as follows.

- Aborting all the deadlocked processes.
- Abort one process at a time until the system recovers from the deadlock.
- Resource Preemption: Resources are taken one by one from a process and assigned to higher priority processes until the deadlock is resolved.

Deadlock Ignorance

In the method, the system assumes that deadlock never occurs. Since the problem of deadlock situation is not frequent, some systems simply ignore it. Operating systems such as UNIX and Windows follow this approach. However, if a deadlock occurs we can reboot our system and the deadlock is resolved automatically.

CONCLUSION

- 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 are mutual exclusion, no preemption, hold and wait and circular set.
- There are four methods of handling deadlocks - deadlock avoidance, deadlock prevention, deadline detection and recovery and deadlock ignorance.
- We can prevent a deadlock by preventing any one of the four necessary conditions for a deadlock.
- There are different ways of detecting and recovering a deadlock in a system.
- A starvation is a situation in which lower priority processes are postponed indefinitely while higher priority processes are executed.
- The advantages of deadlock handling methods are that no preemption is needed and it is good for activities that require a single burst of activity.
- The disadvantages of deadlock handling methods are it delays process initiation and preemptions are frequently encountered in it.

BANKER’S ALGO

Bankers Algorithm in OS

Challenge Inside! : Find out where you stand! Try quiz, solve problems & win rewards!

Go to Challenge

Overview

Bankers algorithm in Operating System is used to avoid deadlock and for resource allocation safely to each process in the system. As the name suggests, it is mainly used in the banking system to check whether the loan can be sanctioned to a person or not. Bankers algorithm in OS is a combination of two main algorithms: safety algorithm (to check whether the system is in a safe state or not) and resource request algorithm (to check how the system behaves when a process makes a resource request).

Scope of the Article

- In this article, we will study the implementation of the Bankers algorithm in OS along with its examples, advantages, and disadvantages.
- We will also take a look at the safety algorithm and resource request algorithm in OS.

What is the Bankers Algorithm in OS?

A process in OS can request a resource, can use the resource, and can also release it. There comes a situation of deadlock in OS in which a set of processes is blocked because it is holding a resource and also requires some other resources at the same time that are being acquired by other processes. So to avoid such a situation of deadlock, we have the Bankers algorithm in Operating System.

Bankers algorithm in OS is a deadlock avoidance algorithm and it is also used to safely allocate the resources to each process within the system. It was designed by Edsger Dijkstra. As the name suggests, Bankers algorithm in OS is mostly used in the banking systems to identify whether a loan should be given to a person or not.

To understand this algorithm in detail, let us discuss a real-world scenario. Supposing there are 'n' account holders in a particular bank and the total sum of their money is 'x'. Now, if a person applies for a loan (let's say for buying a car), then the loan amount subtracted from the total amount available in the bank gives us the remaining amount and that should be greater than 'x', then only the loan will be sanctioned by the bank. It is done keeping in mind about the worst case where all the account holders come to withdraw their money from the bank at the same time. Thus, the bank always remains in a safe state.

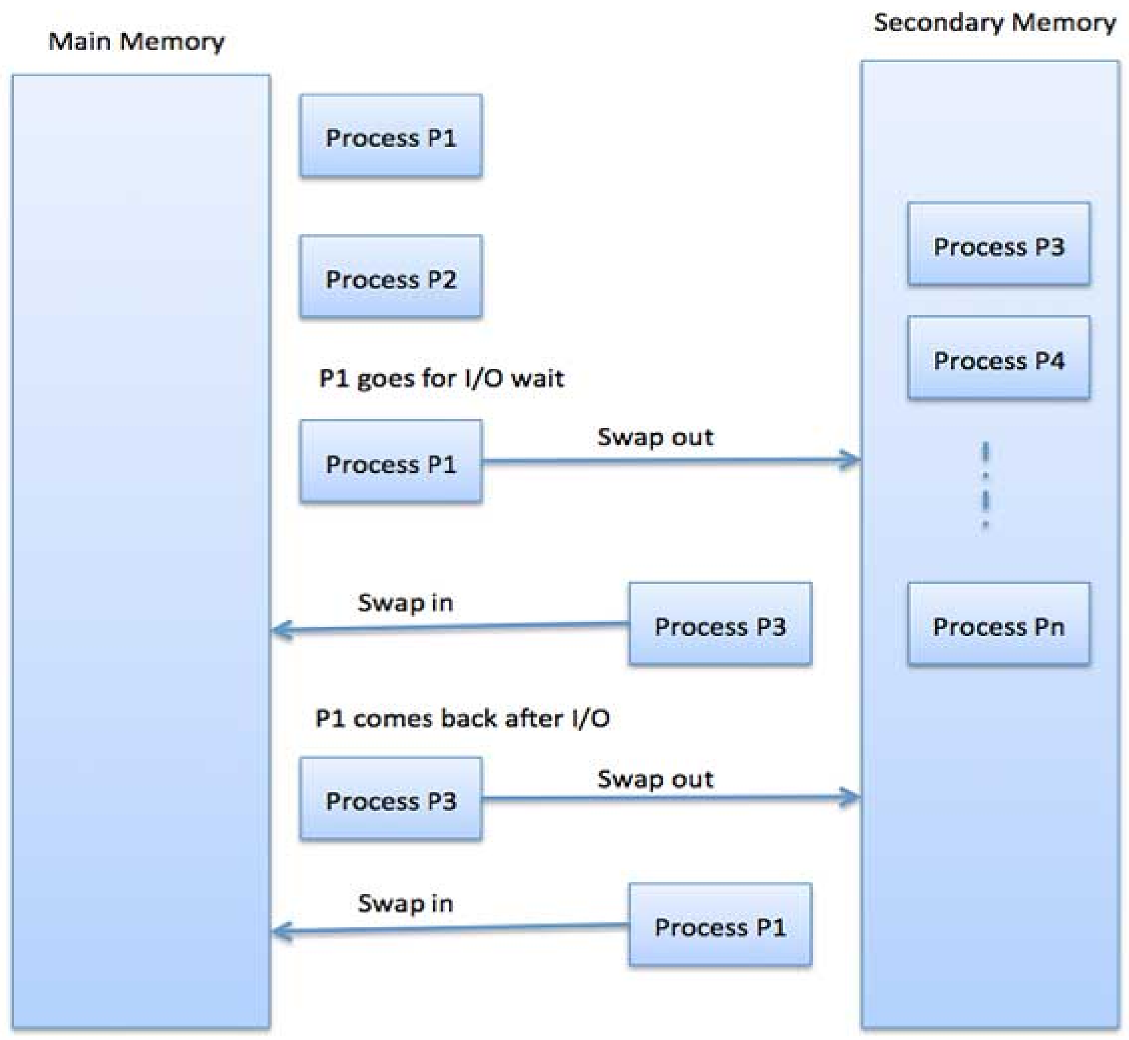
Bankers algorithm in OS works similarly. Each process within the system must provide all the important necessary details to the operating system like upcoming processes, requests for the resources, delays, etc. Based on these details, OS decides whether to execute the process or keep it in the waiting state to avoid deadlocks in the system. Thus, Bankers algorithm is sometimes also known as the Deadlock Detection Algorithm.

Swapping

Swapping is a mechanism in which a process **can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes.** **At some later time, the system swaps back the process from the secondary storage to main memory.**

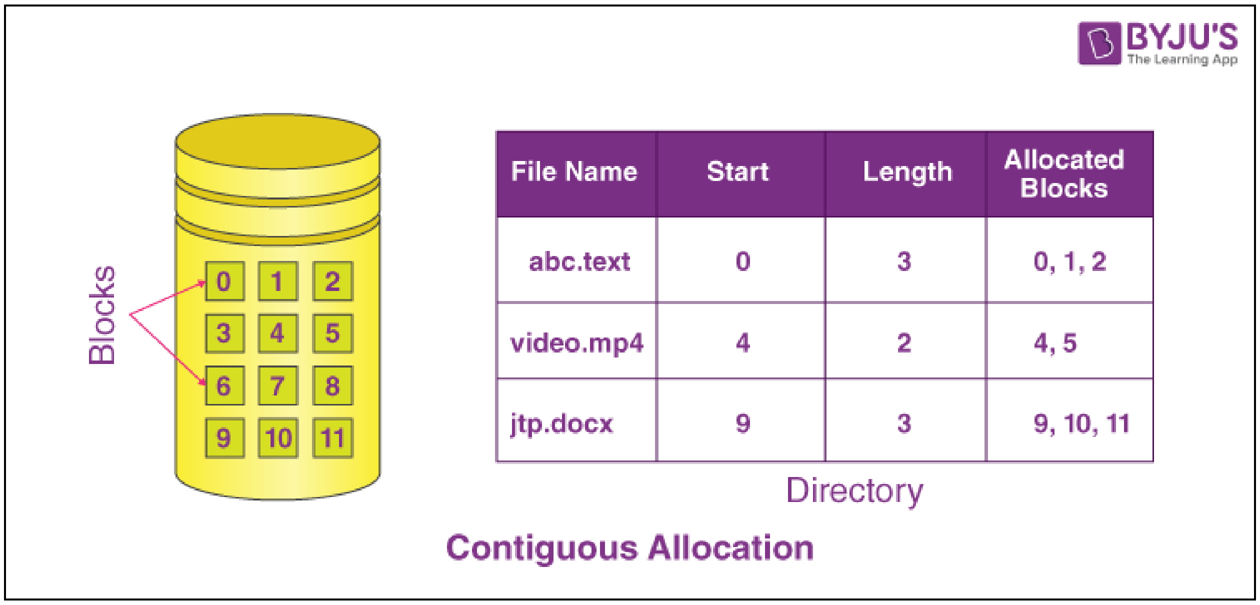
Though **performance is usually affected** by swapping process but it helps in running multiple and big processes in parallel and that's the reason Swapping is also known as a technique for memory compaction.

The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.



What is Contiguous Memory Allocation in Operating Systems?

Contiguous memory allocation refers to a memory management technique in which whenever there occurs a request by a user process for the memory, one of the sections of the contiguous memory block would be given to that process, in accordance with its requirement.



File Name	Start	Length	Allocated Blocks
abc.text	0	3	0, 1, 2
video.mp4	4	2	4, 5
jtp.docx	9	3	9, 10, 11

As you can see in the illustration shown above, three files are there in the directory. The starting block, along with the length of each file, is mentioned in the table. Thus, we can see in this table that all the contiguous blocks get assigned to every file as per their need.

Types of Partitions

Contiguous memory allocation can be achieved when we divide the memory into the following types of partitions:

1. Fixed-Sized Partitions

Another name for this is static partitioning. In this case, the system gets divided into multiple fixed-sized partitions. In this type of scheme, every partition may consist of exactly one process. This very process limits the extent at which multiprogramming would occur, since the total number of partitions decides the total number of processes. Read more on fixed-sized partitions here.

2. Variable-Sized Partitions

Dynamic partitioning is another name for this. The scheme allocation in this type of partition is done dynamically. Here, the size of every partition isn’t declared initially. Only once we know the process size, will we know the size of the partitions. But in this case, the size of the process and the partition is equal; thus, it helps in preventing internal fragmentation.

On the other hand, when a process is smaller than its partition, some size of the partition gets wasted (internal fragmentation). It occurs in static partitioning, and dynamic partitioning solves this issue. Read more on dynamic partitions here.

Pros of Contiguous Memory Allocation

1. It supports a user’s random access to files.

2. The user gets excellent read performance.

3. It is fairly simple to implement.

Cons of Contiguous Memory Allocation

1. Having a file grow might be somewhat difficult.

2. The disk may become fragmented

Paging

Paging is used to bring processes of large size than memory into memory for execution

Paging is a memory management technique in which process address space is broken into blocks of the same size called pages (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called frames and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.

Address Translation

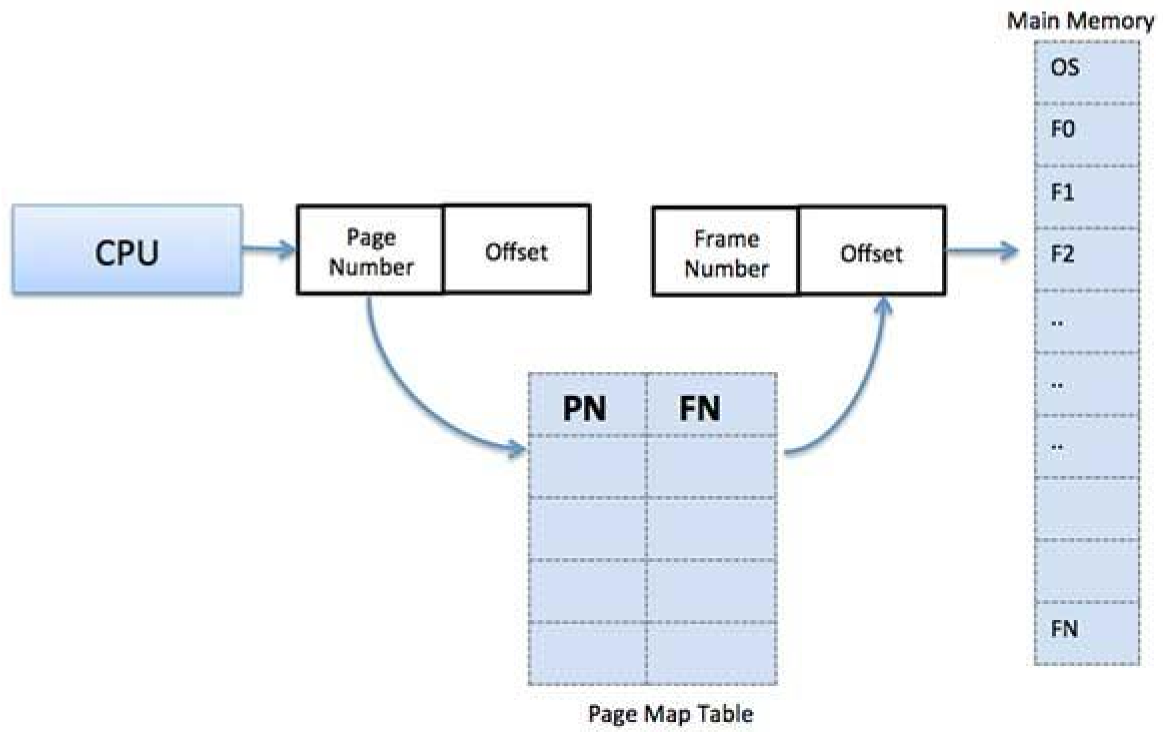
Page address is called logical address and represented by page number and the offset.

Logical Address = Page number + page offset

Frame address is called physical address and represented by a frame number and the offset.

Physical Address = Frame number + page offset

A data structure called page map table is used to keep track of the relation between a page of a process to a frame in physical memory.



- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM

TWO LEVEL PAGING

When the size of the page table is less than the size of one Frame then we need not worry because we can directly put the page table in a frame of the main memory. Thus we can directly access the page table. But if the size of the page table is larger than the size of the Frame. Then the page table in return is to be divided into several pages and these pages of the page table are to be stored in the main memory. Thus, an Outer Page Table comes into the picture. This Outer Page Table would contain the address of the Frames which contain the pages of the **Inner Page Table** (i.e., Page Table one page) in the main memory. The size of this Outer Page is also calculated in the same way as explained above and was used to calculate the size of the inner page Table. Now if the size of the inner page Table is Less Than or Equal to the size of a Frame then we can stop here as we are able to keep the outermost table in a Single frame. This is called **Two Level Paging. Example:** Consider Given,

```
Physical Address Space = 2(44) B


Virtual Address Space = 2(32) B




Page Entry = 4B


Page Size = 4KB




So, No.of Frame = 2(32)


No. of Pages Of the Process = 2(20)
```

Page Table 1 size =2(20) * 4 B= 4 MB

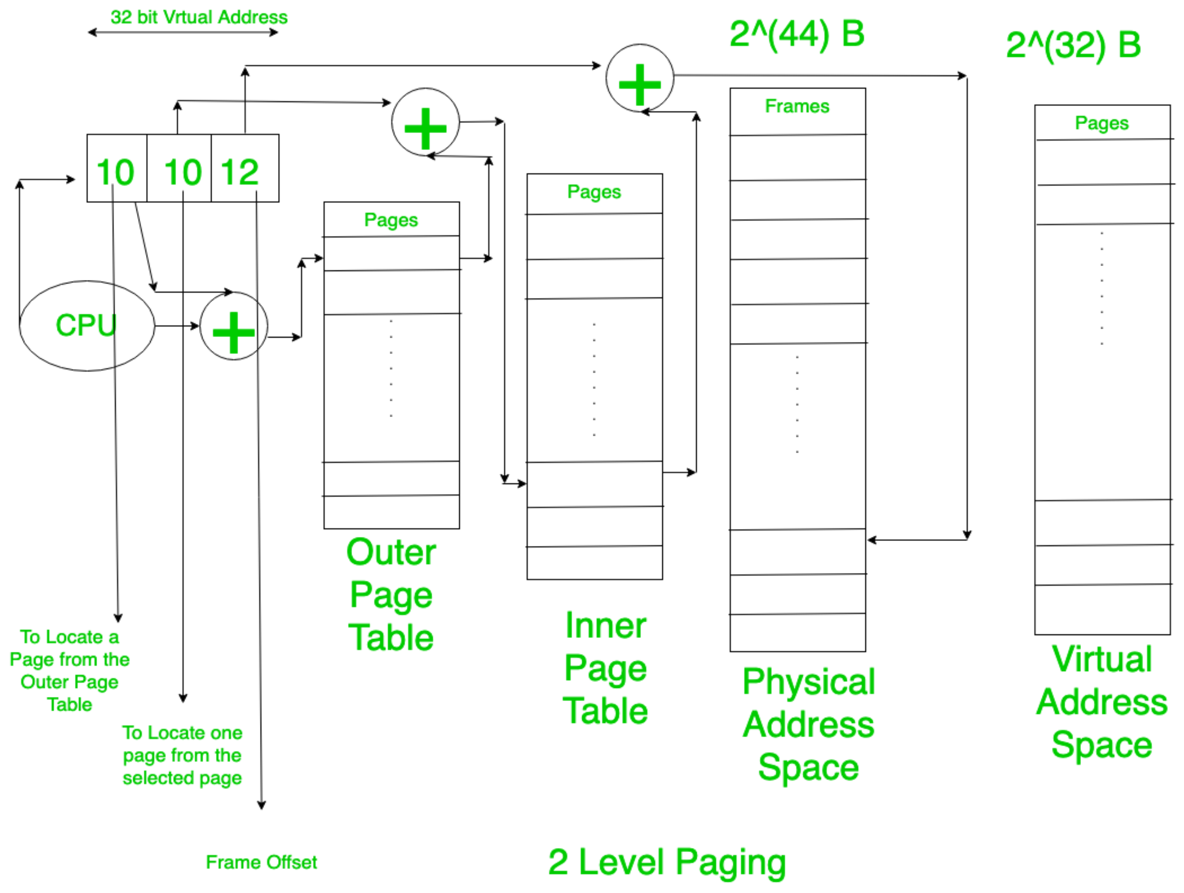
As it is larger than 4KB (Frame size). Thus, this Page Table has to be converted to pages. No. of pages of Page Table 2 (Outer Page Table)

= 2(22)*2(-12)= 2(10) pages

So, the size of the Outer Page Table

= 2(10) * 4B = 4KB

Thus here our Outer Page Table (Page Table 2) can be stored in one frame. Therefore, we can stop here. This is Two-level Paging because here we got 2-page tables.



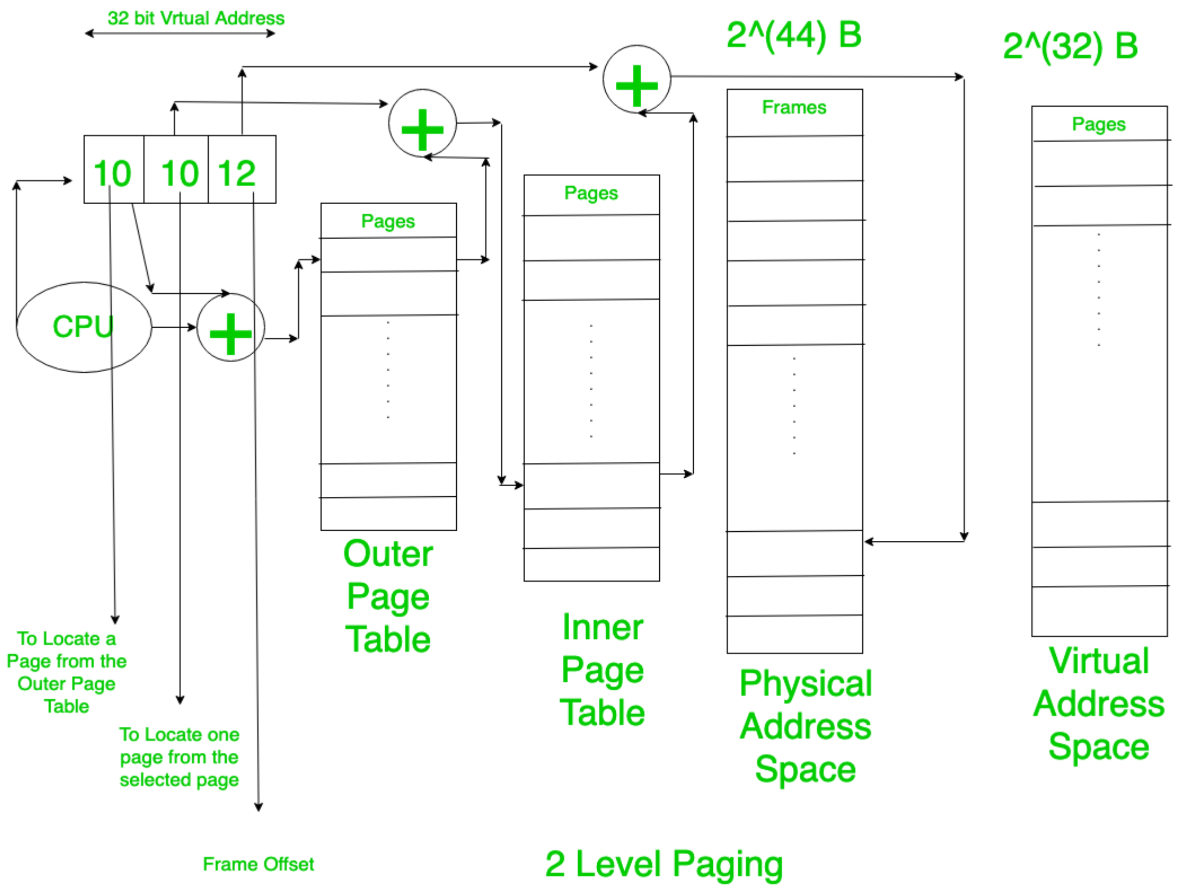
But If still the size of the page table is more than the Frame Size then we have to continue till we reach a stage where the size of Outer Most Table is less than the Frame Size. This Concept is called MultiLevel Paging. Our Target should be to keep the outermost Page Table in a single Frame.

$= 2^{(22)} * 2^{(-12)} = 2^{(10)}$ pages

So, the size of the Outer Page Table

$= 2^{(10)} * 4B = 4KB$

Thus here our Outer Page Table (Page Table 2) can be stored in one frame. Therefore, we can stop here. This is Two-level Paging because here we got 2-page tables.



But If still the size of the page table is more than the Frame Size then we have to continue till we reach a stage where the size of Outer Most Table is less than the Frame Size. This Concept is called MultiLevel Paging. Our Target should be to keep the outermost Page Table in a single Frame.

VIRTUAL MEMORY

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard disk that's set up to emulate the computer's RAM.

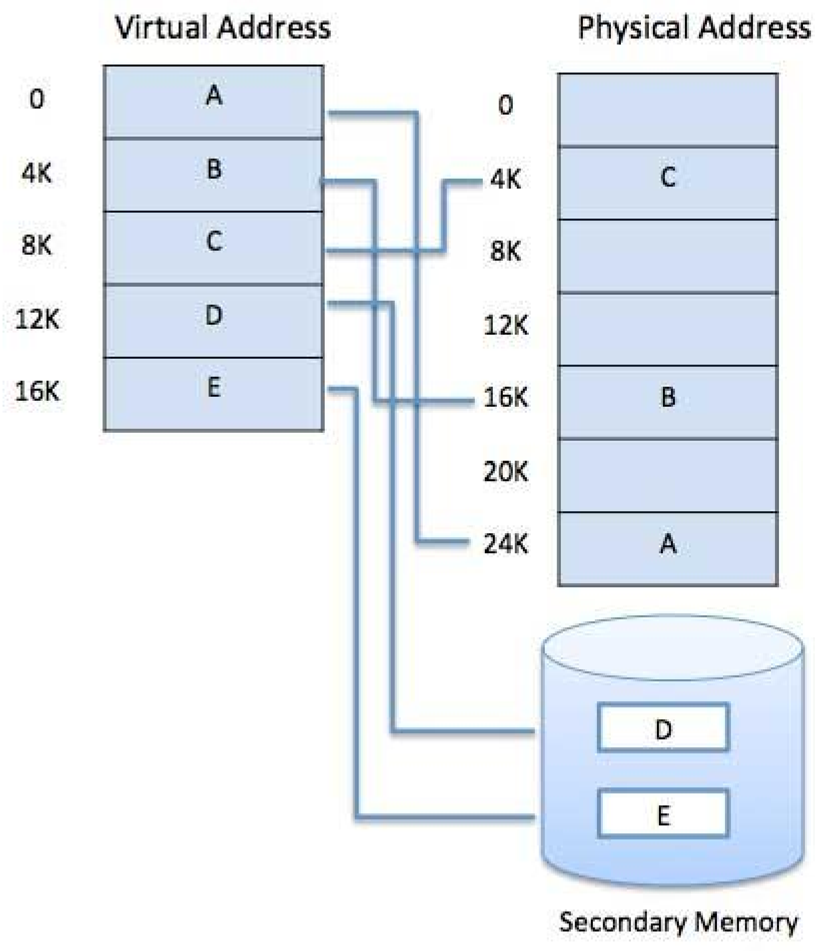
The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Following are the situations, when entire program is not required to be loaded fully in main memory.

- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.
- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability to execute a program that is only partially in memory would counter many benefits.
- Less number of I/O would be needed to load or swap each user program into memory.
- A program would no longer be constrained by the amount of physical memory that is available.

- Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below –



Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.

While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a page fault and transfers control from the program to the operating system to demand the page back into the memory.

Advantages

Following are the advantages of Demand Paging –

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

Page Replacement Algorithm

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults,

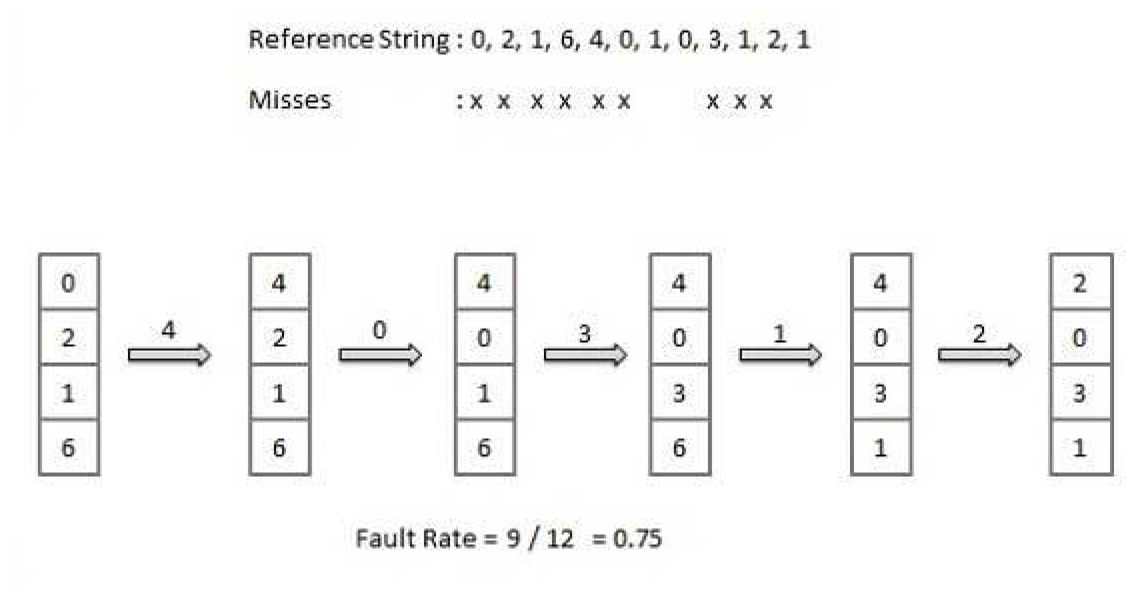
Reference String

The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data, where we note two things.

- For a given page size, we need to consider only the page number, not the entire address.
- If we have a reference to a page p, then any immediately following references to page p will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.
- For example, consider the following sequence of addresses – 123,215,600,1234,76,96
- If page size is 100, then the reference string is 1,2,6,12,0,0

First In First Out (FIFO) algorithm

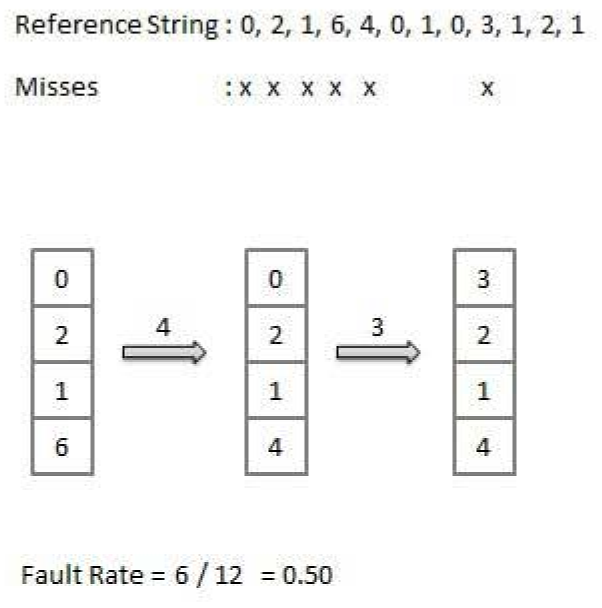
- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.



Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1
Misses : x x x x x x x x x
0 2 1 6
4
4 2 1 6
0
4 0 1 6
3
4 0 3 6
1
4 0 3 1
2
2 0 3 1
Fault Rate = 9 / 12 = 0.75

Optimal Page algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.



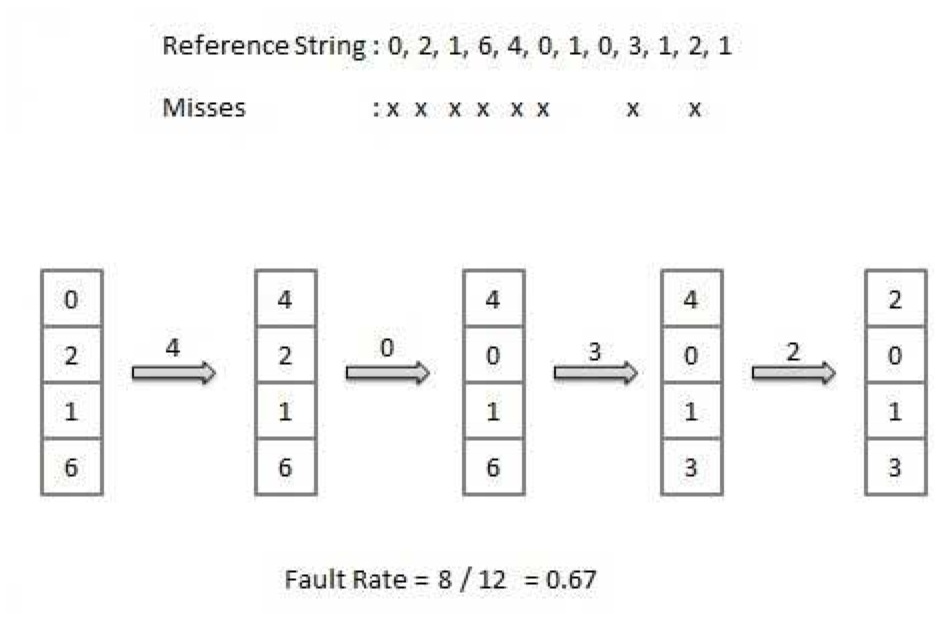
Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x

Fault Rate = 6 / 12 = 0.50

Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.


Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1
Misses : x x x x x x x x
0 2 1 6
4
4 2 1 6
0
4 0 1 6
3
4 0 1 3
2
2 0 1 3
Fault Rate = 8 / 12 = 0.67

Page Buffering algorithm

- To get a process start quickly, keep a pool of free frames.
- On page fault, select a page to be replaced.
- Write the new page in the frame of free pool, mark the page table and restart the process.
- Now write the dirty page out of disk and place the frame holding replaced page in free pool.

Least frequently Used(LFU) algorithm

- The page with the smallest count is the one which will be selected for replacement.
- This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.

Most frequently Used(MFU) algorithm

- This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

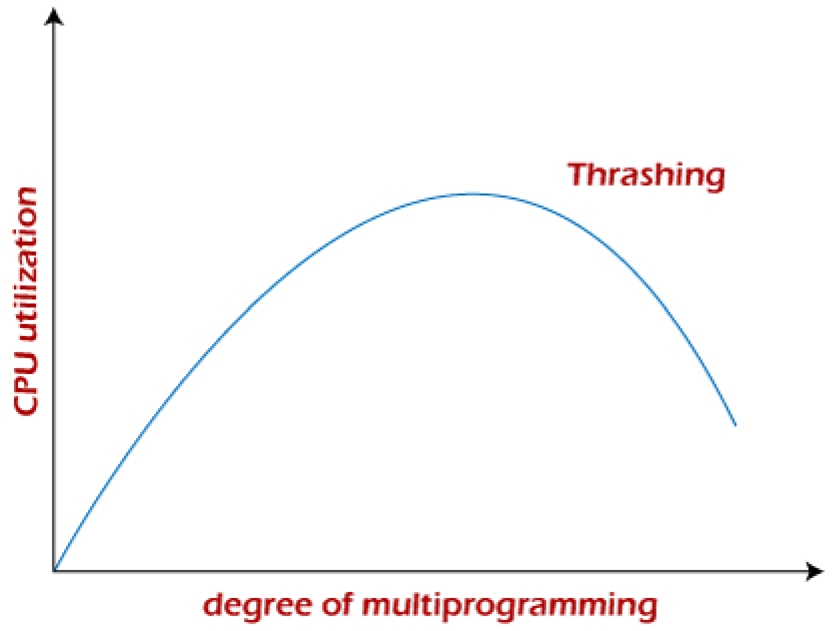
Thrashing

- **Page fault:** We know every program is divided into some pages. A page fault occurs when a program attempts to access data or code in its address space but is not currently located in the system RAM.
- **Swapping:** Whenever a page fault happens, the operating system will try to fetch that page from secondary memory and try to swap it with one of the pages in RAM. This process is called swapping.

Thrashing is when the page fault and swapping happens very frequently at a higher rate, and then the operating system has to spend more time swapping these pages. This state in the operating system is known as thrashing. Because of thrashing, the CPU utilization is going to be reduced or negligible.

The basic concept involved is that if a process is allocated too few frames, then there will be too many and too frequent page faults. As a result, no valuable work would be done by the CPU, and the CPU utilization would fall drastically.

The long-term scheduler would then try to improve the CPU utilization by loading some more processes into the memory, thereby increasing the degree of multiprogramming. Unfortunately, this would result in a further decrease in the CPU utilization, triggering a chained reaction of higher page faults followed by an increase in the degree of multiprogramming, called thrashin



What is Segmentation?

Segmentation is another memory management technique, just like Paging, that divides the addressable memory in distinct segments. However, there is a difference between a Page and a Segment. While Paging divides the memory into blocks of fixed size, Segmentation divides the memory into segments of variable sizes that can grow or shrink as per requirement. A computer that uses segmentation as the memory management technique would have a logical address space that can be viewed as multiple segments.

Advantages of Segmentation

Segmentation supports user's view of memory by dividing a process into modules that provide better visualization. There is no internal fragmentation in Segmentation. A Segment Table keeps track of all the segments, which occupies less space as compared to an equivalent Paging Table that keeps a record of all the pages.

Disadvantages of Segmentation

Segmentation is an expensive technique because it involves a lot of overhead for maintaining a separate segment table for each process. Since the segments are of unequal length, they are not suitable for swapping. Segmentation leads to external fragmentation as the available memory space is broken down into smaller chunks as the processes are being loaded and removed.

Difference between Demand Paging and Segmentation

The following table highlights the major differences between Demand Paging and Segmentation –

Key	Demand Paging	Segmentation
Definition	Paging is a memory management technique in which process address space is broken into blocks of the same size called "pages".	Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions.

Block Size	The block size is fixed in case of Demand Paging.	In Segmentation, the process address space is broken in varying sized blocks which are called as "sections". So block size is not fixed in case of Segmentation.
Block size dependency	In Demand Paging, the size of blocks is dependent on system memory and gets assigned accordingly.	In Segmentation, the size is not dependent on system memory and is all up to user's choice that of what size blocks are needed.
Performance	In context of performance, Demand Paging is faster as compared to Segmentation.	Segmentation is slower in speed as compared to Pagination.
Data Load	In case of Demand Paging, pages get loaded in the main memory at runtime when the user demands it.	In case of Segmentation, all the sections get loaded at the time of compilation.
Data Record	In Demand Paging, there is a Page map table that	In case of Segmentation, there is a Segment map table that manages

	manages the record of pages in memory.	every segment address in the memory.

Conclusion

The most important point that you should note here is that Paging is a faster technique than Segmentation. In Paging, the pages are of fixed size and it is decided by the hardware. In contrast, the size of segments in Segmentation can vary as per the user's requirements