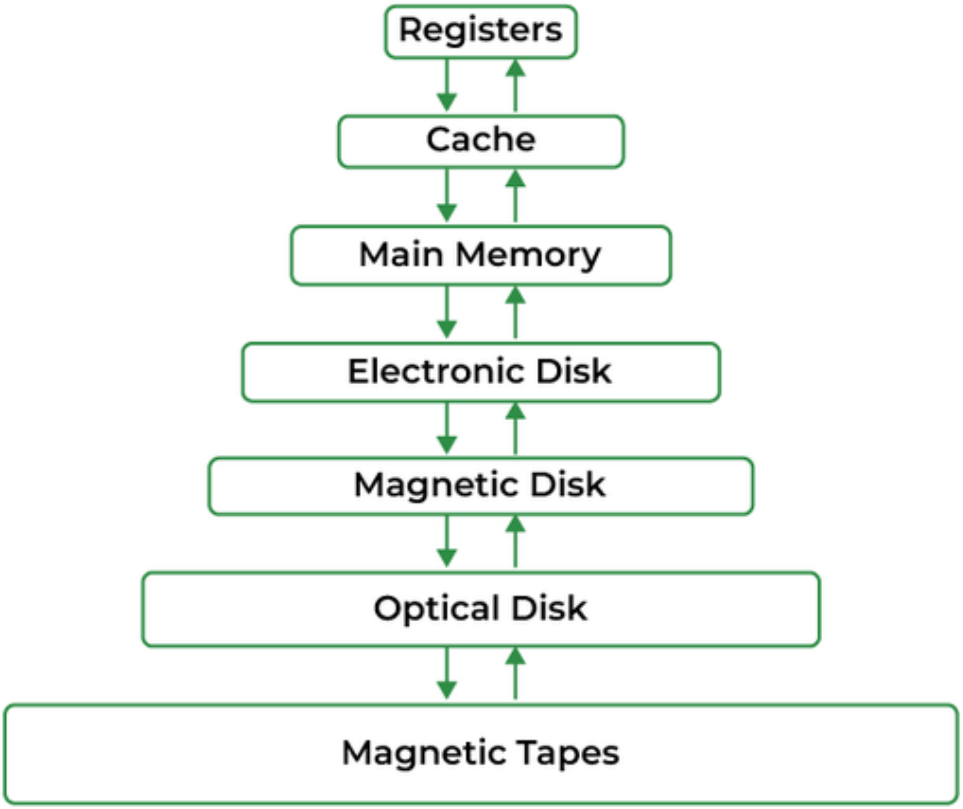
UNIT IV

Memory Management

What is Main Memory ?

- The main memory is central to the operation of a modern computer. Main Memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions.
- Main memory is a repository of rapidly available information shared by the CPU and I/O devices. Main memory is the place where programs and information are kept when the processor is effectively utilizing them.
- Main memory is associated with the processor, so moving instructions and information into and out of the processor is extremely fast. Main memory is also known as RAM(Random Access Memory). This memory is a volatile memory. RAM lost its data when a power interruption occurs.



What is Memory Management :

In a multiprogramming computer, the operating system resides in a part of memory and the rest is used by multiple processes. The task of subdividing the memory among different processes is called memory management. Memory management is a method in the operating system to manage operations between main memory and disk during process execution. The main aim of memory management is to achieve efficient utilization of memory.

Why Memory Management is required:

- Allocate and de-allocate memory before and after process execution.
- To keep track of used memory space by processes.
- To minimize fragmentation issues.

- To proper utilization of main memory.
- To maintain data integrity while executing of process.

Swapping:

Swapping is moving data between physical memory(RAM) and secondary memory. In computing, virtual memory is a management technique that combines a computer's hard disk space with its random access memory (RAM) to create a larger virtual address space. This can be useful if you have too many processes running on your system and not enough physical memory to store them.

While performing swapping the **operating system** needs to allocate a block of memory. It finds the first vacant block of physical memory.

As each new block of memory is allocated, it replaces the oldest block in physical memory. When a program attempts to access a page that has been swapped out, the operating system copies the page from the disk into physical memory and updates the page table entry.

The purpose of operating system swapping is to increase the degree of multiprogramming and increase main memory usage.

There are two steps to changing the operating system:

- **Swap-in**: A swap-in process in which a process moves from secondary storage / hard disk to main memory (RAM).
- **Swap out:** Swap out takes a process out of the main memory and places it in secondary memory.

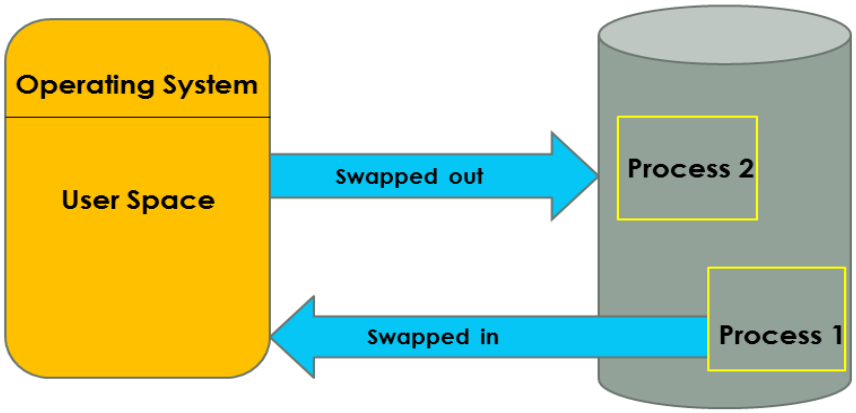
Advantages of Swapping

- Swapping can help to make more room and allow your programs to run more smoothly. Additionally, swap files can also be helpful in cases where you have multiple programs open at the same time.

- Using a swap file, you can ensure that each program has its own dedicated chunk of memory, which can help improve overall performance.
- Improve the degree of multi-programming.
- Better RAM utilization.

Disadvantages of Swapping

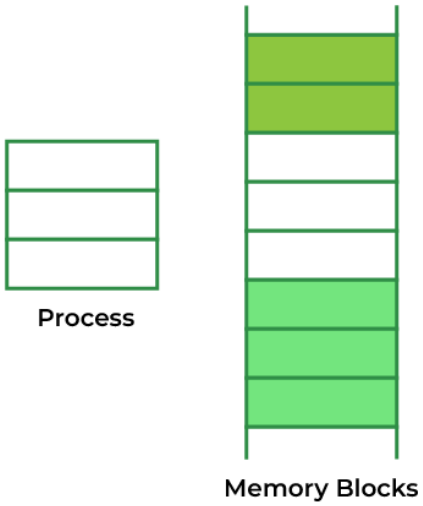
- If the computer system is turned off during high paging activity, the user may lose all information related to the program.
- The number of page faults increases, which can reduce overall processing performance.
- When you make a lot of transactions, users lose information and computers lose power.



Contiguous Memory Allocation:

It is the type of memory allocation method. When a process requests the memory, a single contiguous section of memory blocks is allotted depending on its requirements.

It is completed by partitioning the memory into fixed-sized partitions and assigning every partition to a single process. However, it will limit the degree of multiprogramming to the number of fixed partitions done in memory.



contiguous memory allocation

This allocation also leads to internal fragmentation. For example, suppose a fixed-sized memory block assigned to a process is slightly bigger than its demand. In that case, the remaining memory space in the block is referred to as internal fragmentation. When a process in a partition finishes, the partition becomes accessible for another process to run.

The OS preserves a table showing which memory partitions are free and occupied by processes in the variable partitioning scheme. Contiguous memory allocation speeds up process execution by decreasing address translation overheads.

Advantages and Disadvantages of Contiguous Memory Allocation

There are various advantages and disadvantages of contiguous memory allocation. Some of the advantages and disadvantages are as follows:

Advantages

1. It is simple to keep track of how many memory blocks are left, which determines how many more processes can be granted memory space.
2. The read performance of contiguous memory allocation is good because the complete file may be read from the disk in a single task.
3. The contiguous allocation is simple to set up and performs well.

Disadvantages

1. Fragmentation isn't a problem because every new file may be written to the end of the disk after the previous one.
2. When generating a new file, it must know its eventual size to select the appropriate hole size.
3. When the disk is filled up, it would be necessary to compress or reuse the spare space in the holes.

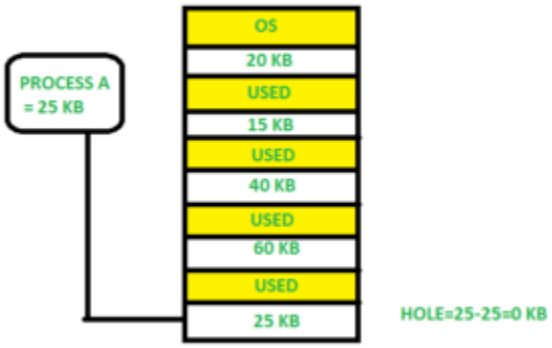
First fit:-

In the first fit, the first available free hole fulfills the requirement of the process allocated.

Here, in this diagram 40 KB memory block is the first available free hole that can store process A (size of 25 KB), because the first two blocks did not have sufficient memory space.

Best fit:-

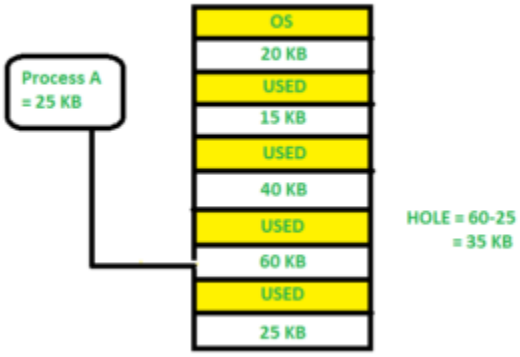
In the best fit, allocate the smallest hole that is big enough to process requirements. For this, we search the entire list, unless the list is ordered by size.



Here in this example, first, we traverse the complete list and find the last hole 25KB is the best suitable hole for Process A(size 25KB).

In this method memory utilization is maximum as compared to other memory allocation techniques.

Worst fit:-In the worst fit, allocate the largest available hole to process. This method produces the largest leftover hole.



Here in this example, Process A (Size 25 KB) is allocated to the largest available memory block which is 60KB. Inefficient memory utilization is a major issue in the worst fit.

Paging:

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. The process of retrieving processes in the form of pages from the secondary storage into the main memory is known as paging. The basic purpose of paging is to separate each procedure into pages. Additionally, frames will be used to split the main memory.This scheme permits the physical address space of a process to be non – contiguous.

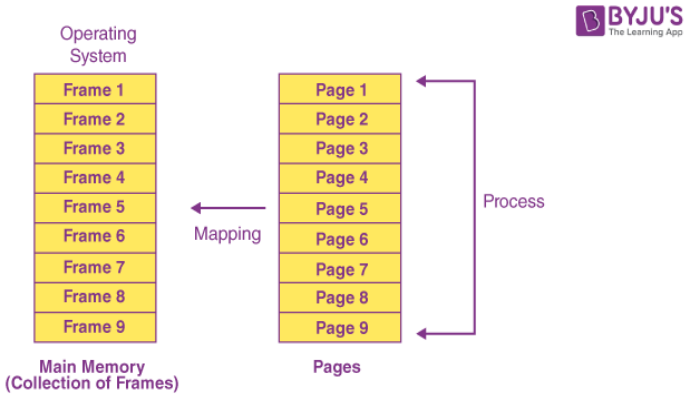
Let us look some important terminologies:

- Logical Address or Virtual Address (represented in bits): An address generated by the CPU
- Logical Address Space or Virtual Address Space(represented in words or bytes): The set of all logical addresses generated by a program
- Physical Address (represented in bits): An address actually available on memory unit
- Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses

Features of paging:

1. Mapping logical address to physical address.
2. Page size is equal to frame size.
3. Number of entries in a page table is equal to number of pages in logical address space.
4. The page table entry contains the frame number.

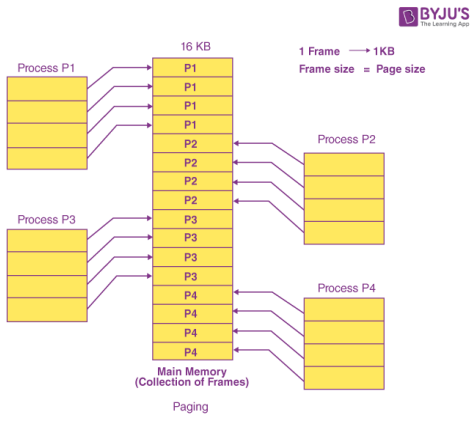
5. All the page table of the processes are placed in main memory.



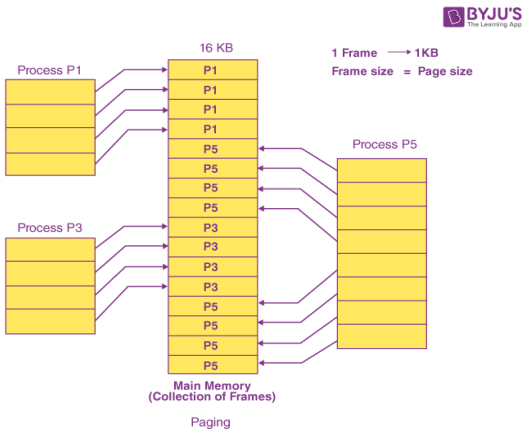
Examples:

Assuming that the main memory is 16 KB and the frame size is 1 KB, the main memory will be partitioned into a collection of 16 1 KB frames. P1, P2, P3, and P4 are the four processes in the system, each of which is 4 KB in size. Each process is separated into 1 KB pages, allowing one page to be saved in a single frame.

Because all of the frames are initially empty, the pages of the processes will be stored in a continuous manner. The graphic below depicts frames, pages, and the mapping between them.

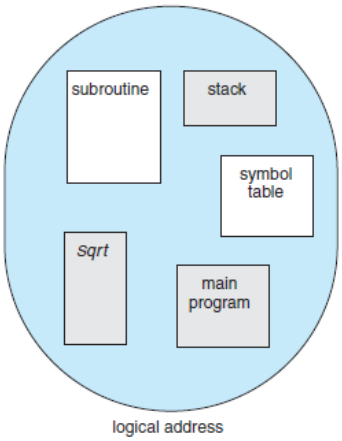


Consider the case when P2 and P4 are shifted to the waiting state after a period of time. Eight frames are now empty, allowing other pages to be loaded in their stead. Inside the ready queue is the process P5, which is 8 KB (8 pages) in size.Given that we have 8 noncontiguous frames accessible in memory, paging allows us to store the process in many locations. As a result, we can load the process P5 page instead of P2 and P4.



Segmentation:

Segmentation is a memory-management scheme that supports this programmer view of memory. A logical address space is a collection of segments.



Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment. The programmer therefore specifies each address by two quantities: a segment name and an offset.

For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name. Thus, a logical address consists of a two tuple*:*

<segment-number, offset>.

Normally, when a program is compiled, the compiler automatically constructs segments reflecting the input program. A C compiler might create separate segments for the following:

1. The code

2. Global variables

3. The heap, from which memory is allocated

4. The stacks used by each thread

5. The standard C library

Libraries that are linked in during compile time might be assigned separate segments. The loader would take all these segments and assign them segment numbers.

Advantages of Segmentation:

1. No internal fragmentation
2. Average Segment Size is larger than the actual page size.
3. Less overhead
4. It is easier to relocate segments than entire address space.
5. The segment table is of lesser size as compared to the page table in paging.

Disadvantages of Segmentation:

1. It can have external fragmentation.
2. it is difficult to allocate contiguous memory to variable sized partition.
3. Costly memory management algorithms.

Intel 32 :

IA-32 (short for "Intel Architecture, 32-bit", commonly called i386[1][2])[3] is the 32-bit version of the x86 instruction set architecture, designed by Intel and first implemented in the 80386 microprocessor in 1985.

Architectural Features :

The primary defining characteristic of IA-32 is the availability of 32-bit general-purpose processor registers (for example, EAX and EBX), 32-bit integer arithmetic and logical operations, 32-bit offsets within a segment in protected mode, and the translation of segmented addresses to 32-bit linear addresses. The designers took the opportunity to make other improvements as well. Some of the most significant changes (relative to the 16-bit 286 instruction set) are described below.

- 32-bit integer capability

  All general-purpose registers (GPRs) are expanded from 16 bits to 32 bits, and all arithmetic and logical operations, memory-to-register and register-to-memory operations, etc., can operate directly on 32-bit integers. Pushes and pops on the stack default to 4-byte strides, and non-segmented pointers are 4 bytes wide.

- More general addressing modes

  Any GPR can be used as a base register, and any GPR other than ESP can be used as an index register, in a memory reference. The index register value can be multiplied by 1, 2, 4, or 8 before being added to the base register value and displacement.

- Additional segment registers

  Two additional segment registers, FS and GS, are provided.

- Larger virtual address space

  The IA-32 architecture defines a 48-bit segmented address format, with a 16-bit segment number and a 32-bit offset within the segment. Segmented addresses are mapped to 32-bit linear addresses.

- Demand paging

  32-bit linear addresses are virtual addresses rather than physical addresses; they are translated to physical addresses through a page table. In the 80386, 80486, and the original Pentium processors, the physical address was 32 bits; in the Pentium Pro and later processors, the Physical Address Extension allowed 36-bit physical addresses, although the linear address size was still 32 bits.

Operation Mode:

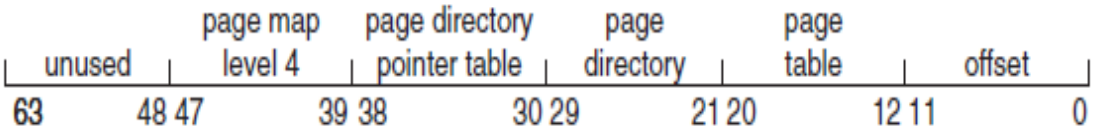
Operating mode	**Operating system required**	**Type of code being run**	**Default address size**	**Default operand size**	**Typical GPR width**
Protected mode	32-bit operating system or boot loader	32-bit protected-mode code	32 bits	32 bits	32 bits

	16-bit protected-mode operating system or boot loader, or 32-bit boot loader	16-bit protected-mode code	16 bits	16 bits	16 or 32 bits
Virtual 8086 mode	16- or 32-bit protected-mode operating system	16-bit real-mode code	16 bits	16 bits	16 or 32 bits
Real mode	16-bit real-mode operating system or boot loader, or 32-bit boot loader	16-bit real-mode code	16 bits	16 bits	16 or 32 bits
Unreal mode	16-bit real-mode operating system or boot loader, or 32-bit boot loader				

64-bit Architectures:

- ✓ Intel has had an interesting history of developing 64-bit architectures. Its initial entry was the IA-64 (later named Itanium) architecture, but that architecture was not widely adopted.
- ✓ Meanwhile, another chip manufacturer— AMD —began developing a 64-bit architecture known as x86-64 that was based on extending the existing IA-32 instruction set.

- ✓ The x86-64 supported much larger logical and physical address spaces, as well as several other architectural advances.
- ✓ Historically, AMD had often developed chips based on Intel'sarchitecture, but now the roles were reversed as Intel adopted AMD's x86-64 architecture. In discussing this architecture, rather than using the commercial namesAMD64 and Intel 64, we will use the more general term x86-64.
- ✓ Support for a 64-bit address space yields an astonishing 264 bytes of addressable memory—a number greater than 16 quintillion (or 16 exabytes).
- ✓ However, even though 64-bit systems can potentially address this much memory, in practice far fewer than 64bits are used for address representation in current designs.
- ✓ The x86-64 architecture currently provides a 48-bit virtual address with support for page sizes of 4 KB, 2 MB, or 1 GB using four levels of paging hierarchy.
- ✓ The representation of the linear address appears in image below diagram. Because this addressing scheme can use PAE, virtual addresses are 48 bits in size but support 52-bit physical addresses (4096 terabytes).



ARM Architecture:

Advanced RISC Machine (ARM) Processor is considered to be family of Central Processing Units that is used in music players, smartphones, wearables, tablets and other consumer electronic devices.

The architecture of ARM processor is created by Advanced RISC Machines, hence name ARM. This needs very few instruction sets and transistors. It has very small size.

This is reason that it is perfect fit for small size devices. It has less power consumption along with reduced complexity in its circuits.

They can be applied to various designs such as 32-bit devices and embedded systems. They can even be upgraded according to user needs.

The main features of ARM Processor are mentioned below :

1. **Multiprocessing Systems –**

   ARM processors are designed so that they can be used in cases of multiprocessing systems where more than one processors are used to process information. First AMP processor introduced by name of ARMv6K had ability to support 4 CPUs along with its hardware.

2. **Tightly Coupled Memory –**

   Memory of ARM processors is tightly coupled. This has very fast response time. It has low latency (quick response) that can also be used in cases of cache memory being unpredictable.

3. **Memory Management –**

   ARM processor has management section. This includes Memory Management Unit and Memory Protection Unit. These management systems become very important in managing memory efficiently.

4. **Thumb-2 Technology –**

   Thumb-2 Technology was introduced in 2003 and was used to create variable length instruction set. It extends 16-bit instructions of initial Thumb technology to 32-bit

instructions. It has better performance than previously used Thumb technology.

5. **One cycle execution time** – ARM processor is optimized for each instruction on CPU. Each instruction is of fixed length that allows time for fetching future instructions before executing present instruction. ARM has CPI (Clock Per Instruction) of one cycle.

6. **Pipelining** – Processing of instructions is done in parallel using pipelines. Instructions are broken down and decoded in one pipeline stage. The pipeline advances one step at a time to increase throughput (rate of processing).

7. **Large number of registers** – Large number of registers are used in ARM processor to prevent large amount of memory interactions. Registers contain data and addresses. These act as local memory store for all operations.

Virtual-Memory Management

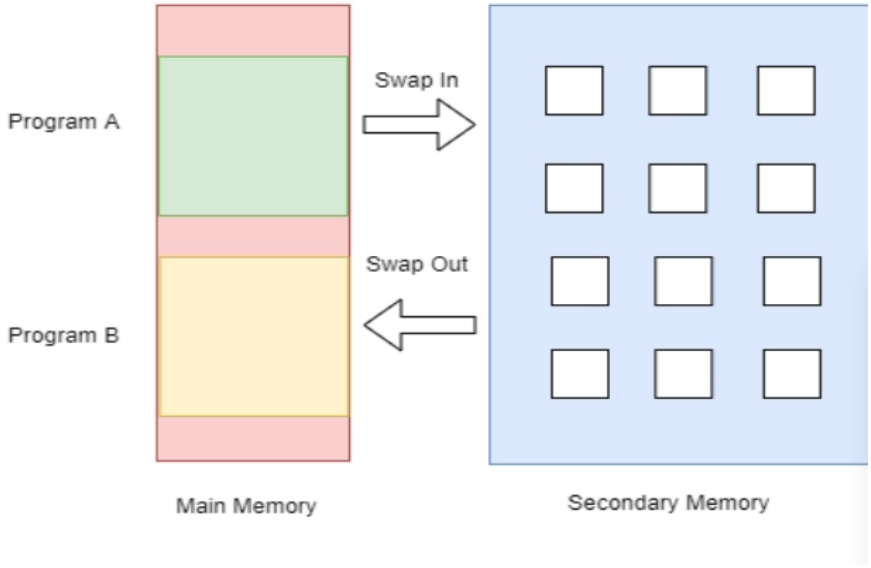
Demand Paging :

The demand paging system is similar to the swapping paging system in that processes are mostly stored in the main memory (usually on the hard disk). As a result, demand paging is a

procedure that addresses the problem above just by shifting pages on demand. Lazy swapper is another name for this (It never swaps the page into the memory unless it is needed).

A pager is a kind of swapper that deals with the individual pages of a process.

Demand Paging is a method in which a page is only brought into main memory when the CPU requests it. At first, just those pages are loaded directly required by the operation. Pages that are never accessed are thus never loaded into physical memory.



Advantages of Demand Paging :

Demand paging has the following benefits:

- Memory can be put to better use.
- If we use demand paging, then we can have a large virtual memory.
- By using demand paging, we can run programs that are larger than physical memory.
- In demand paging, there is no requirement for compaction.
- In demand paging, the sharing of pages is easy.
- Partition management is simple in demand paging because of the fixed partition size and the discontinuous loading.

Disadvantages of Demand Paging:

Demand paging has the following drawbacks:

- Internal fragmentation is a possibility with demand paging.
- It takes longer to access memory (page table lookup).
- Memory requirements

- Guarded page tables
- Inverted page tables

Common Terms in Demand Paging Operating System:

Following are some common terms in demand paging operating systems:

- Page Fault
- Swapping
- Thrashing

Page Fault

There will be a miss if the referenced page is not present in the main memory; this is known as a page miss or page fault.

The CPU must look up the missing page in secondary memory. When the number of page faults is significant, the system's effective access time increases dramatically.

Swapping

Swapping comprises either erasing all of the process's pages from memory or marking the pages so that we can remove them via the page replacement method.

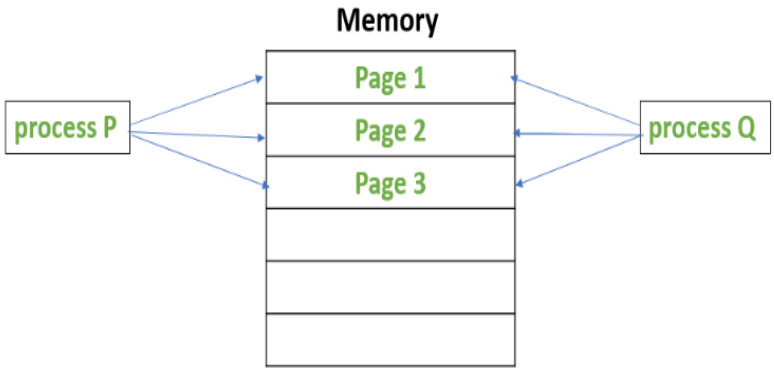
Thrashing

The effective access time will be the time needed by the CPU to read one word from the secondary memory if the number of page faults is equal to the number of referred pages or if the number of page faults is so high that the CPU is only reading pages from the secondary memory. This is known as Thrashing.

Copy-on-Write:

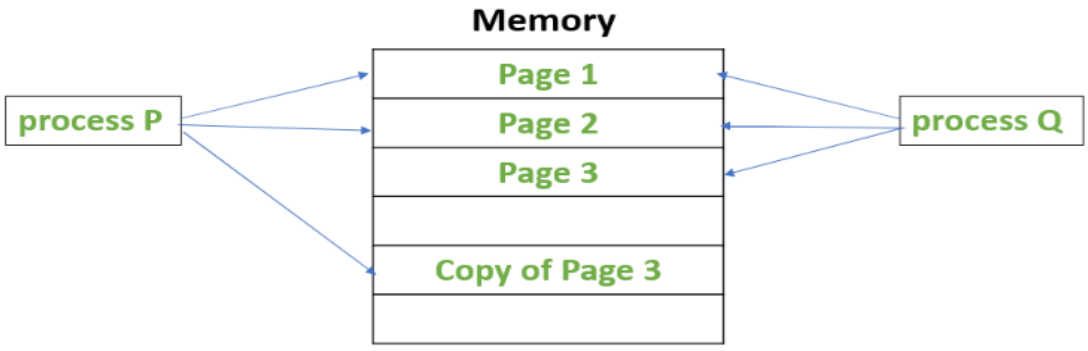
- ✓ **Copy on Write** or simply COW is a resource management technique. One of its main use is in the implementation of the fork system call in which it shares the virtual memory(pages) of the OS.
- ✓ In UNIX like OS, fork() system call creates a duplicate process of the parent process which is called as the child process.

- ✓ The idea behind a copy-on-write is that when a parent process creates a child process then both of these processes initially will share the same pages in memory and these shared pages will be marked as copy-on-write which means that if any of these processes will try to modify the shared pages then only a copy of these pages will be created and the modifications will be done on the copy of pages by that process and thus not affecting the other process.
- ✓ Suppose, there is a process P that creates a new process Q and then process Pmodifies page 3.The below figures shows what happens before and after process P modifies page



Before process P modifies Page 3

3.



After process P modifies Page 3

Page Replacement:

- ✓ Page replacement is needed in the operating systems that use virtual memory using Demand Paging. As we know that in Demand paging, only a set of pages of a process is

loaded into the memory. This is done so that we can have more processes in the memory at the same time.

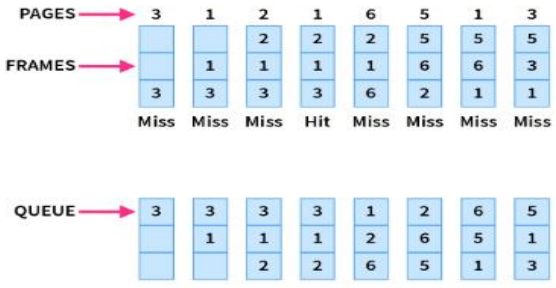
- ✓ When a page that is residing in virtual memory is requested by a process for its execution, the Operating System needs to decide which page will be replaced by this requested page. This process is known as page replacement and is a vital component in virtual memory management.

Page Replacement Algorithms in Operating Systems

First In First Out (FIFO)

- ✓ FIFO algorithm is the simplest of all the page replacement algorithms. In this, we maintain a queue of all the pages that are in the memory currently. The oldest page in the memory is at the front-end of the queue and the most recent page is at the back or rear-end of the queue.
- ✓ Whenever a page fault occurs, the operating system looks at the front-end of the queue to know the page to be replaced by the newly requested page. It also adds this newly requested page at the rear-end and removes the oldest page from the front-end of the queue.

Example: Consider the page reference string as 3, 1, 2, 1, 6, 5, 1, 3 with 3-page frames. Let's try to find the number of page faults:



- Initially, all of the slots are empty so page faults occur at 3,1,2.

Page faults = 3

- When page 1 comes, it is in the memory so no page fault occurs.

Page faults = 3

- When page 6 comes, it is not present and a page fault occurs. Since there are no empty slots, we remove the front of the queue, i.e 3.

Page faults = 4

- When page 5 comes, it is also not present and hence a page fault occurs. The front of the queue i.e 1 is removed.

Page faults = 5

- When page 1 comes, it is not found in memory and again a page fault occurs. The front of the queue i.e 2 is removed.

Page faults = 6

- When page 3 comes, it is again not found in memory, a page fault occurs, and page 6 is removed being on top of the queue

Total page faults = 7

Belady's anomaly: Generally if we increase the number of frames in the memory, the number of page faults should decrease due to obvious reasons. Belady's anomaly refers to the phenomena where increasing the number of frames in memory, increases the page faults as well.

Advantages

- Simple to understand and implement
- Does not cause more overhead

Disadvantages

- Poor performance
- Doesn't use the frequency of the last used time and just simply replaces the oldest page.
- Suffers from Belady's anomaly.

Allocation of Frames:

An important aspect of operating systems, virtual memory is implemented using demand paging. Demand paging necessitates the development of a page-replacement algorithm and a **frame allocation algorithm**. Frame allocation algorithms are used if you have multiple processes; it helps decide how many frames to allocate to each process.

There are various constraints to the strategies for the allocation of frames:

- ✓ You cannot allocate more than the total number of available frames.
- ✓ At least a minimum number of frames should be allocated to each process. This constraint is supported by two reasons. The first reason is, as less number of frames are allocated, there is an increase in the page fault ratio, decreasing the performance of the execution of the process. Secondly, there should be enough frames to hold all the different pages that any single instruction can reference.

Frame allocation algorithms –

The two algorithms commonly used to allocate frames to a process are:

1. **Equal allocation:** In a system with x frames and y processes, each process gets equal number of frames, i.e. x/y. For instance, if the system has 48 frames and 9 processes, each process will get 5 frames. The three frames which are not allocated to any process can be used as a free-frame buffer pool.
   - **Disadvantage:** In systems with processes of varying sizes, it does not make much sense to give each process equal frames. Allocation of a large number of frames to a small process will eventually lead to the wastage of a large number of allocated unused frames.
2. **Proportional allocation:** Frames are allocated to each process according to the process size. For a process p_i of size s_i, the number of allocated frames is $\mathbf{a_i = (s_i/S)*m}$, where S is the sum of the sizes of all the processes and m is the number of frames in the system. For

instance, in a system with 62 frames, if there is a process of 10KB and another process of 127KB, then the first process will be allocated (10/137)*62 = 4 frames and the other process will get (127/137)*62 = 57 frames.

- **Advantage:** All the processes share the available frames according to their needs, rather than equally.

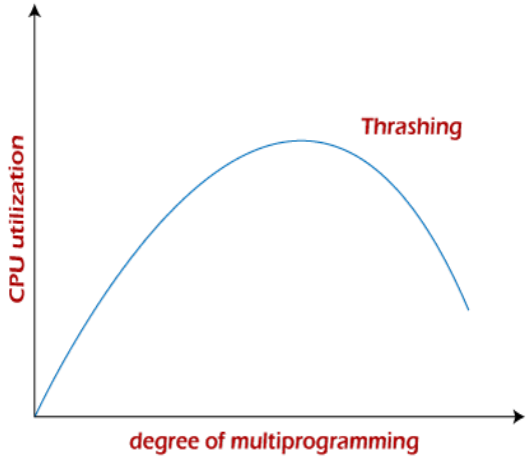
Global vs Local Allocation –

The number of frames allocated to a process can also dynamically change depending on whether you have used **global replacement** or **local replacement** for replacing pages in case of a page fault.

1. **Local replacement:** When a process needs a page which is not in the memory, it can bring in the new page and allocate it a frame from its own set of allocated frames only.
   - **Advantage:** The pages in memory for a particular process and the page fault ratio is affected by the paging behavior of only that process.
   - **Disadvantage:** A low priority process may hinder a high priority process by not making its frames available to the high priority process.
2. **Global replacement:** When a process needs a page which is not in the memory, it can bring in the new page and allocate it a frame from the set of all frames, even if that frame is currently allocated to some other process; that is, one process can take a frame from another.
   - **Advantage:** Does not hinder the performance of processes and hence results in greater system throughput.
   - **Disadvantage:** The page fault ratio of a process can not be solely controlled by the process itself. The pages in memory for a process depends on the paging behavior of other processes as well.

Thrashing:

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.



Algorithms during Thrashing:

Whenever thrashing starts, the operating system tries to apply either the Global page replacement Algorithm or the Local page replacement algorithm.

1. Global Page Replacement

Since global page replacement can bring any page, it tries to bring more pages whenever thrashing is found. But what actually will happen is that no process gets enough frames, and as a result, the thrashing will increase more and more. Therefore, the global page replacement algorithm is not suitable when thrashing happens.

2. Local Page Replacement

Unlike the global page replacement algorithm, local page replacement will select pages which only belong to that process. So there is a chance to reduce the thrashing. But it is proven that

there are many disadvantages if we use local page replacement. Therefore, local page replacement is just an alternative to global page replacement in a thrashing scenario.

Causes of Thrashing:

Programs or workloads may cause thrashing, and it results in severe performance problems, such as:

- If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new system. A global page replacement algorithm is used. The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming.
- CPU utilization is plotted against the degree of multiprogramming.
- As the degree of multiprogramming increases, CPU utilization also increases.
- If the degree of multiprogramming is increased further, thrashing sets in, and CPU utilization drops sharply.
- So, at this point, to increase CPU utilization and to stop thrashing, we must decrease the degree of multiprogramming.

Eliminate Thrashing:

Thrashing has some negative impacts on hard drive health and system performance. Therefore, it is necessary to take some actions to avoid it. To resolve the problem of thrashing, here are the following methods, such as:

- **Adjust the swap file size:**If the system swap file is not configured correctly, disk thrashing can also happen to you.
- **Increase the amount of RAM:** As insufficient memory can cause disk thrashing, one solution is to add more RAM to the laptop. With more memory, your computer can handle tasks easily and don't have to work excessively. Generally, it is the best long-term solution.
- **Decrease the number of applications running on the computer:** If there are too many applications running in the background, your system resource will consume a lot. And the

remaining system resource is slow that can result in thrashing. So while closing, some applications will release some resources so that you can avoid thrashing to some extent.

- **Replace programs:** Replace those programs that are heavy memory occupied with equivalents that use less memory.

Allocating Kernel Memory:

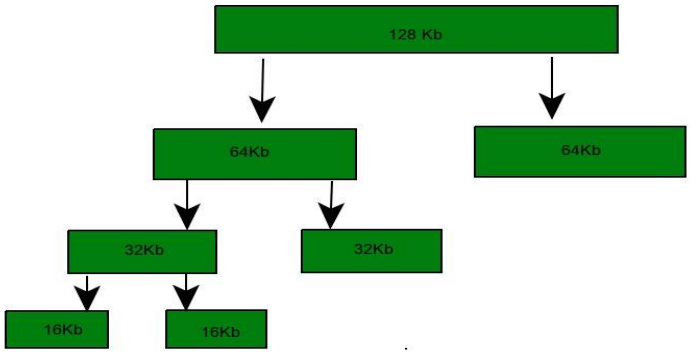
Kernel memory is often allocated from a free-memory pool different from the list used to satisfy ordinary user-mode processes.

Two strategies for managing free memory that is assigned to kernel processes:

1. Buddy system –

Buddy allocation system is an algorithm in which a larger memory block is divided into small parts to satisfy the request. This algorithm is used to give best fit. The two smaller parts of block are of equal size and called as buddies. In the same manner one of the two buddies will further divide into smaller parts until the request is fulfilled. Benefit of this technique is that the two buddies can combine to form the block of larger size according to the memory request.

Example – If the request of 25Kb is made then block of size 32Kb is allocated.



Four Types of Buddy System –

1. Binary buddy system
2. Fibonacci buddy system
3. Weighted buddy system
4. Tertiary buddy system

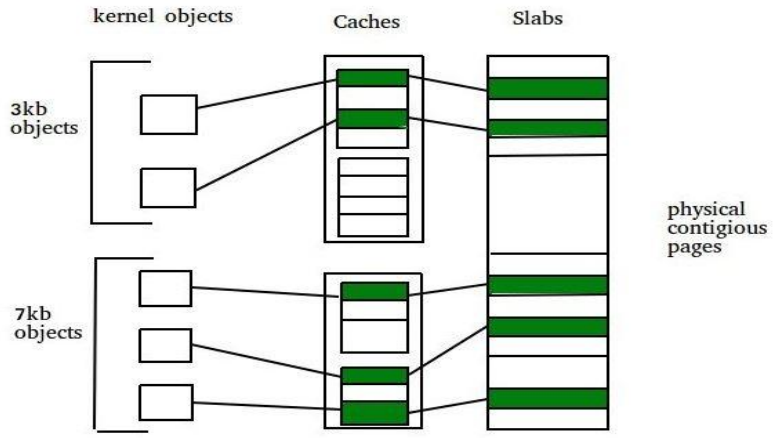
2. Slab Allocation –

A second strategy for allocating kernel memory is known as slab allocation. It eliminates fragmentation caused by allocations and deallocations. This method is used to retain allocated memory that contains a data object of a certain type for reuse upon subsequent allocations of objects of the same type.

In slab allocation memory chunks suitable to fit data objects of certain type or size are preallocated. Cache does not free the space immediately after use although it keeps track of data which are required frequently so that whenever request is made the data will reach very fast.

Two terms required are:

- **Slab –** A slab is made up of one or more physically contiguous pages. The slab is the actual container of data associated with objects of the specific kind of the containing cache.
- **Cache –** Cache represents a small amount of very fast memory. A cache consists of one or more slabs. There is a single cache for each unique kernel data structure.



Example –

- A separate cache for a data structure representing processes descriptors
- Separate cache for file objects
- Separate cache for semaphores etc.