**Operating System for Placement**

1. OS
   1. <https://www.geeksforgeeks.org/last-minute-notes-operating-systems/>

Resource – GFG

* <https://www.geeksforgeeks.org/difference-between-multitasking-multithreading-and-multiprocessing/>
* **Difference between Multiprogramming, multitasking, multithreading and multiprocessing**

1. **Multiprogramming –** A computer running more than one program at a time (like running Excel and Firefox simultaneously).
2. **Multiprocessing –** A computer using more than one CPU at a time.
3. **Multitasking –** Tasks sharing a common resource (like 1 CPU).
4. **Multithreading** is an extension of multitasking.

* <https://www.geeksforgeeks.org/random-access-memory-ram-and-read-only-memory-rom/>
* <https://www.geeksforgeeks.org/difference-32-bit-64-bit-operating-systems/>

**System Structure :**

* [Introduction of System Call](https://www.geeksforgeeks.org/operating-system-introduction-system-call/)
* [Dual Mode operations in OS](https://www.geeksforgeeks.org/dual-mode-operations-os/)
* [Privileged and Non-Privileged Instructions](https://www.geeksforgeeks.org/operating-system-privileged-and-non-privileged-instructions/)

**CPU Scheduling & Process**

1. [Process | (Introduction and different states)](https://www.geeksforgeeks.org/gate-notes-operating-system-process-management-introduction/)
2. [States of a process](https://www.geeksforgeeks.org/operating-systems-states-process/)
3. [Process Table and Process Control Block (PCB)](https://www.geeksforgeeks.org/operating-system-process-table-process-control-block-pcb/)
4. [Process Scheduler](https://www.geeksforgeeks.org/gate-notes-operating-system-scheduler/)
5. [CPU Scheduling](https://www.geeksforgeeks.org/gate-notes-operating-system-process-scheduling/)
6. [Preemptive and Non-Preemptive Scheduling](https://www.geeksforgeeks.org/preemptive-and-non-preemptive-scheduling/)
7. [Difference between dispatcher and scheduler](https://www.geeksforgeeks.org/operating-system-difference-dispatcher-scheduler/)
8. [Belady’s Anomaly](https://www.geeksforgeeks.org/operating-system-beladys-anomaly/)

**Process Synchronization :**

1. [Process Synchronization | Introduction](https://www.geeksforgeeks.org/process-synchronization-set-1/)
2. [Process Synchronization | Set 2](https://www.geeksforgeeks.org/operating-system-process-synchronization/)
3. [Critical Section](https://www.geeksforgeeks.org/g-fact-70/)
4. [Inter Process Communication](https://www.geeksforgeeks.org/inter-process-communication/)
5. [Interprocess Communication: Methods](https://www.geeksforgeeks.org/interprocess-communication-methods/)
6. [IPC through shared memory](https://www.geeksforgeeks.org/ipc-shared-memory/)
7. [IPC using Message Queues](https://www.geeksforgeeks.org/ipc-using-message-queues/)
8. [Message based Communication in IPC (inter process communication)](https://www.geeksforgeeks.org/message-based-communication-in-ipc-inter-process-communication/)
9. [Communication between two process using signals in C](https://www.geeksforgeeks.org/signals-c-set-2/)
10. [Semaphores in operating system](https://www.geeksforgeeks.org/semaphores-operating-system/)
11. [Mutex vs. Semaphore](https://www.geeksforgeeks.org/mutex-vs-semaphore/)
12. [Producer Consumer Problem using Semaphores | Set 1](https://www.geeksforgeeks.org/producer-consumer-problem-using-semaphores-set-1/)
13. [Readers-Writers Problem | Set 1 (Introduction and Readers Preference Solution)](https://www.geeksforgeeks.org/readers-writers-problem-set-1-introduction-and-readers-preference-solution/)

**Processes & Threads :**

* **Process vs Thread?**
  + The primary difference is that threads within the same process run in a shared memory space, while processes run in separate memory spaces.
  + Threads are not independent of one another like processes are, and as a result threads share with other threads their code section, data section, and OS resources (like open files and signals). But, like process, a thread has its own program counter (PC), register set, and stack space.
* ***Advantages of Thread over Process***  
  *1. Responsiveness:* If the process is divided into multiple threads, if one thread completes its execution, then its output can be immediately returned.
* *2. Faster context switch:*Context switch time between threads is lower compared to process context switch. Process context switching requires more overhead from the CPU.
* *3. Effective utilization of multiprocessor system:* If we have multiple threads in a single process, then we can schedule multiple threads on multiple processor. This will make process execution faster.
* *4. Resource sharing:* Resources like code, data, and files can be shared among all threads within a process.  
  Note: stack and registers can’t be shared among the threads. Each thread has its own stack and registers.
* *5. Communication:*Communication between multiple threads is easier, as the threads shares common address space. while in process we have to follow some specific communication technique for communication between two process.

# Threads and its types in Operating System

Each thread has

1. A program counter
2. A register set
3. A stack space

# Threads are not independent of each other as they share the code, data, OS resources etc.

* **User Level thread (ULT) –**
  + Is implemented in the user level library, they are not created using the system calls. Thread switching does not need to call OS and to cause interrupt to Kernel. Kernel doesn’t know about the user level thread and manages them as if they were single-threaded processes.

**Advantages of ULT –**

* Can be implemented on an OS that does’t support multithreading.
* Simple representation since thread has only program counter, register set, stack space.
  + Thread switching is fast since no OS calls need to be made.

**Disadvantages of ULT –**

* + No or less co-ordination among the threads and Kernel.
  + If one thread causes a page fault, the entire process blocks.
* **Kernel Level Thread (KLT) –**
  + Kernel knows and manages the threads. Instead of thread table in each process, the kernel itself has thread table (a master one) that keeps track of all the threads in the system. In addition kernel also maintains the traditional process table to keep track of the processes. OS kernel provides system call to create and manage threads.

**Advantages of KLT –**

* + Since kernel has full knowledge about the threads in the system, scheduler may decide to give more time to processes having large number of threads.
  + Good for applications that frequently block.

**Disadvantages of KLT –**

* + Slow and inefficient.
  + It requires thread control block so it is an overhead.
  + They are created by the kernel via the kernel\_thread function.
  + They run as part of the kernel and are not associated with any userspace program/process/thread. They have full access to the machine. Devices, MMU, etc. Kernel threads run in the highest privilege level: ring 0. They also run in the kernel's address space and not the address space of any user process/thread.

**Summary:**

1. Each ULT has a process that keeps track of the thread using the Thread table.
2. Each KLT has Thread Table (TCB) as well as the Process Table (PCB).

**Multitasking programming is of two types –**

1. Process-based Multitasking
2. Thread-based Multitasking.

**Process Based Multitasking Programming –**

* In process based multitasking two or more processes and programs can be run concurrently.
* In process based multitasking a process or a program is the smallest unit.

# **Example –** We can listen to music and browse internet at the same time. The processes in this example are the music player and browser.

# **Thread Based Multitasking Programming –**

* In thread based multitasking two or more threads can be run concurrently.
* In thread based multitasking a thread is the smallest unit.
* Thread based multitasking requires less overhead.
* Threads share same address space.

# Example -  word-processing application like MS Word, we can type text in one thread and spell checker checks for mistakes in another thread.

# Multi Threading Models in Process Management

# ****Many to Many Model**** –

# In this model, we have multiple user threads multiplex to same or lesser number of kernel level threads. Number of kernel level threads are specific to the machine, advantage of this model is if a user thread is blocked we can schedule others user thread to other kernel thread. Thus, System doesn’t block if a particular thread is blocked.

**Many to One Model**

* In this model, we have multiple user threads mapped to one kernel thread. In this model when a user thread makes a blocking system call entire process blocks. As we have only one kernel thread and only one user thread can access kernel at a time, so multiple threads are not able access multiprocessor at the same time.

**One to One Model**

* In this model, one to one relationship between kernel and user thread. In this model multiple thread can run on multiple processor. Problem with this model is that creating a user thread requires the corresponding kernel thread.
* As each user thread is connected to different kernel , if any user thread makes a blocking system call, the other user threads won’t be blocked.

**Memory Management :**

In the operating system, the following are four common memory management techniques.

**Single contiguous allocation:** Simplest allocation method used by MS-DOS. All memory (except some reserved for OS) is available to a process.

**Partitioned allocation:** Memory is divided into different blocks or partitions. Each process is allocated according to the requirement.

**Paged memory management:** Memory is divided into fixed-sized units called page frames, used in a virtual memory environment.

**Segmented memory management:** Memory is divided into different segments (a segment is a logical grouping of the process’ data or code).In this management, allocated memory doesn’t have to be contiguous.

In **Partition Allocation**, when there is more than one partition freely available to accommodate a process’s request, a partition must be selected. To choose a particular partition, a partition allocation method is needed. A partition allocation method is considered better if it avoids internal fragmentation.

When it is time to load a process into the main memory and if there is more than one free block of memory of sufficient size then the OS decides which free block to allocate.

There are different Placement Algorithm:

A. First Fit

B. Best Fit

C. Worst Fit

D. Next Fit

There are two Memory Management Techniques: **Contiguous**, and **Non-Contiguous**. In Contiguous Technique, executing process must be loaded entirely in main-memory. Contiguous Technique can be divided into:

1. Fixed (or static) partitioning
2. Variable (or dynamic) partitioning

**Fixed Partitioning:**  
This is the oldest and simplest technique used to put more than one processes in the main memory. In this partitioning, number of partitions (non-overlapping) in RAM are **fixed but size** of each partition may or **may not be same**. As it is **contiguous** allocation, hence no spanning is allowed. Here partition are made before execution or during system configure.

**Disadvantages of Fixed Partitioning –**

1. **Internal Fragmentation:**  
   Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This can cause internal fragmentation.
2. **External Fragmentation:**  
   The total unused space (as stated above) of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form (as spanning is not allowed).

**Variable Partitioning –**  
It is a part of Contiguous allocation technique. It is used to alleviate the problem faced by Fixed Partitioning. In contrast with fixed partitioning, partitions are not made before the execution or during system configure. Various **features** associated with variable Partitioning-

1. Initially RAM is empty and partitions are made during the run-time according to process’s need instead of partitioning during system configure.
2. The size of partition will be equal to incoming process.
3. The partition size varies according to the need of the process so that the internal fragmentation can be avoided to ensure efficient utilisation of RAM.
4. Number of partitions in RAM is not fixed and depends on the number of incoming process and Main Memory’s size.

**Advantages of Variable Partitioning –**

1. **No Internal Fragmentation:**

**Disadvantages of Variable Partitioning –**

1. **Difficult Implementation:**  
   Implementing variable Partitioning is difficult as compared to Fixed Partitioning as it involves allocation of memory during run-time rather than during system configure.
2. **External Fragmentation:**

# Non-Contiguous Allocation in Operating System

[Paging](https://www.geeksforgeeks.org/operating-system-paging/) and [Segmentation](https://www.geeksforgeeks.org/operating-systems-segmentation/) are the two ways which allow a process’s physical address space to be non-contiguous. It has **advantage** of reducing memory wastage but it increases the overheads due to address translation. It slows the execution of the memory because time is consumed in address translation.

In non-contiguous allocation, Operating system needs to maintain the table which is called **Page Table** for each process which contains the base address of the each block which is acquired by the process in memory space. In non-contiguous memory allocation, different parts of a process is allocated different places in Main Memory. Spanning is allowed which is not possible in other techniques like Dynamic or Static Contiguous memory allocation. That’s why paging is needed to ensure effective memory allocation. Paging is done to remove External Fragmentation.

But, in what manner we divide a process to allocate them into main memory is very important to understand. Process is divided after analysing the number of empty spaces and their size in main memory. Then only we divide our process. It is very time consuming process. Their number as well as their sizes changing every time due to execution of already present processes in main memory.

In order to avoid this time consuming process, we divide our process in secondary memory in advance before reaching the main memory for its execution. Every process is divided into various parts of equal size called Pages. We also divide our main memory into different parts of equal size called Frames. It is important to understand that:

Size of page in process = Size of frame in memory

----- consider empty main memory having size of each frame is 2 KB, and two processes P1 and P2 are 2 KB each.

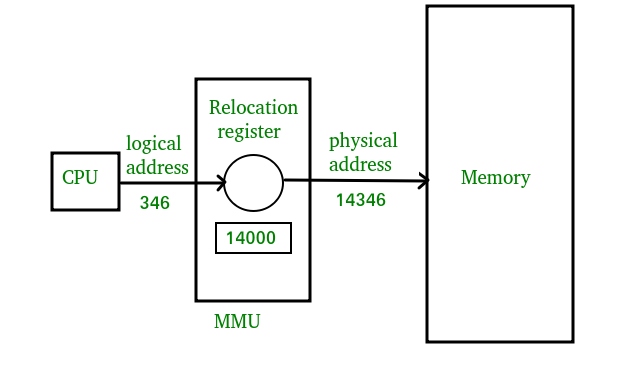


# Logical and Physical Address in Operating System

**Logical Address** is generated by CPU while a program is running. The logical address is virtual address as it does not exist physically, therefore, it is also known as Virtual Address. This address is used as a reference to access the physical memory location by CPU. The term Logical Address Space is used for the set of all logical addresses generated by a program’s perspective. 

The hardware device called **Memory-Management Unit is used for mapping logical address to its corresponding physical address**.

**Physical Address** identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address. The user program generates the logical address and thinks that the program is running in this logical address but the program needs physical memory for its execution, therefore, the logical address must be mapped to the physical address by MMU before they are used. The term Physical Address Space is used for all physical addresses corresponding to the logical addresses in a Logical address space.



**Differences Between Logical and Physical Address in Operating System**

| Parameter | LOGICAL ADDRESS | PHYSICAL ADDRESS |
| --- | --- | --- |
| Basic | generated by CPU | location in a memory unit |
| Address Space | Logical Address Space is set of all logical addresses generated by CPU in reference to a program. | Physical Address is set of all physical addresses mapped to the corresponding logical addresses. |
| Visibility | User can view the logical address of a program. | User can never view physical address of program. |
| Generation | generated by the CPU | Computed by MMU |
| Access | The user can use the logical address to access the physical address. | The user can indirectly access physical address but not directly. |

* Identical logical addresses are generated by Compile-time and Load time address binding methods whereas they differs from each other in run-time address binding method.
* The logical address is generated by the CPU while the program is running whereas the physical address is computed by the Memory Management Unit (MMU).

# Paging in Operating System

Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory. This scheme permits the physical address space of a process to be non – contiguous.

* 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

**Example:**

* If Logical Address = 31 bit, then Logical Address Space = 231 words = 2 G words (1 G = 230)
* If Logical Address Space = 128 M words = 27 \* 220 words, then Logical Address = log2 227 = 27 bits
* If Physical Address = 22 bit, then Physical Address Space = 222 words = 4 M words (1 M = 220)
* If Physical Address Space = 16 M words = 24 \* 220 words, then Physical Address = log2 224 = 24 bits

The mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device and this mapping is known as paging technique.

* The Physical Address Space is conceptually divided into a number of fixed-size blocks, called **frames**.
* The Logical address Space is also splitted into fixed-size blocks, called **pages**.
* Page Size = Frame Size

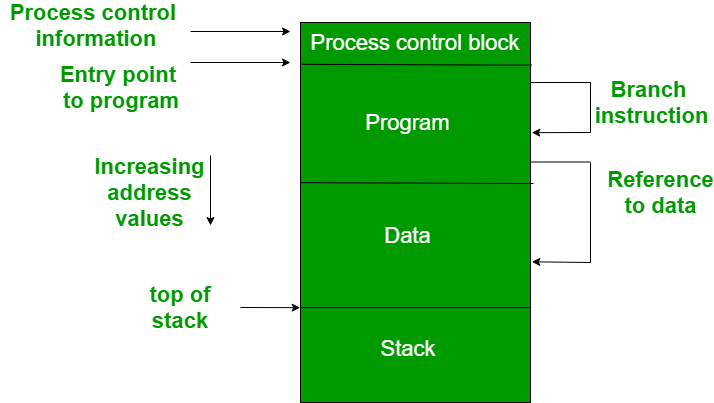
# Requirements of Memory Management System

Memory management keeps track of the status of each memory location, whether it is allocated or free. It allocates the memory dynamically to the programs at their request and frees it for reuse when it is no longer needed

These Requirements of memory management are:

**Relocation –** The available memory is generally shared among a number of processes in a multiprogramming system, so it is not possible to know in advance which other programs will be resident in main memory at the time of execution of his program. Swapping the active processes in and out of the main memory enables the operating system to have a larger pool of ready-to-execute process.

When a program gets swapped out to a disk memory, then it is not always possible that when it is swapped back into main memory then it occupies the previous memory location, since the location may still be occupied by another process. We may need to **relocate** the process to a different area of memory. Thus there is a possibility that program may be moved in main memory due to swapping.



The operating system will need to know many things including the location of process control information, the execution stack, and the code entry. Within a program, there are memory references in various instructions and these are called logical addresses.

After loading of the program into main memory, the processor and the operating system must be able to translate logical addresses into physical addresses. Branch instructions contain the address of the next instruction to be executed. Data reference instructions contain the address of byte or word of data referenced.

**Protection –** There is always a danger when we have multiple programs at the same time as one program may write to the address space of another program. So every process must be protected against unwanted interference when other process tries to write in a process whether accidental or incidental. Between relocation and protection requirement a trade-off occurs as the satisfaction of relocation requirement increases the difficulty of satisfying the protection requirement.

Prediction of the location of a program in main memory is not possible, that’s why it is impossible to check the absolute address at compile time to assure protection. Most of the programming language allows the dynamic calculation of address at run time. The memory protection requirement must be satisfied by the processor rather than the operating system because the operating system can hardly control a process when it occupies the processor. Thus it is possible to check the validity of memory references.

# Mapping Virtual Addresses to Physical Addresses

**MMU(Memory Management Unit) :**

The run time mapping between Virtual address and Physical Address is done by a hardware device known as MMU.  
In memory management, the Operating System will handle the processes and move the processes between disk and memory for execution . It keeps track of available and used memory.

1. CPU will generate logical address for eg: 346
2. MMU will generate a relocation register (base register) for eg: 14000
3. In memory, the physical address is located eg:(346+14000= 14346)

The value in the relocation register is added to every address generated by a user process at the time the address is sent to memory. The user program never sees the real physical addresses. The program can create a pointer to location 346, store it in memory, manipulate it, and compare it with other addresses—all like the number 346.   
The user program generates only logical addresses. However, these logical addresses must be mapped to physical addresses before they are used.

**Address binding :**  
Address binding is the process of mapping from one address space to another address space. Logical address is an address generated by the CPU during execution, whereas Physical Address refers to the location in the memory unit(the one that is loaded into memory).The logical address undergoes translation by the MMU or address translation unit in particular. The output of this process is the appropriate physical address or the location of code/data in RAM.

An address binding can be done in three different ways :

**Compile Time –**   
If you know that during compile time, where process will reside in memory, then an absolute address is generated. i.e The physical address is embedded to the executable of the program during compilation. Loading the executable as a process in memory is very fast. But if the generated address space is preoccupied by other processes, then the program crashes and it becomes necessary to recompile the program to change the address space.

**Load time –**   
If it is not known at the compile time where the process will reside, then a relocatable address will be generated. The loader translates the relocatable address to an absolute address. The base address of the process in main memory is added to all logical addresses by the loader to generate an absolute address. In this, if the base address of the process changes, then we need to reload the process again.

**Execution time –**   
The instructions are in memory and are being processed by the CPU. Additional memory may be allocated and/or deallocated at this time. This is used if a process can be moved from one memory to another during execution(dynamic linking-Linking that is done during load or run time). e.g – Compaction.

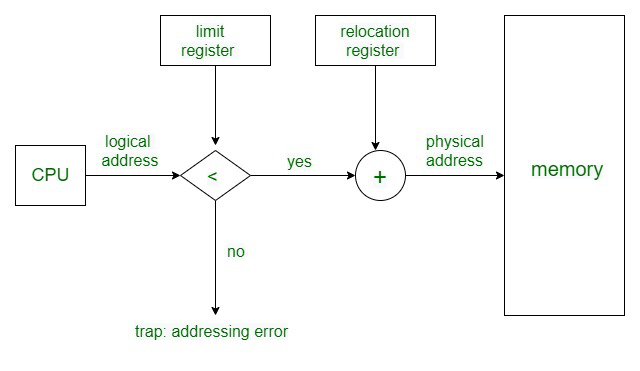
**Mapping Virtual Addresses to Physical Addresses :**  
In Contiguous memory allocation mapping from virtual addresses to physical addresses is not a difficult task, because if we take a process from secondary memory and copy it to the main memory, the addresses  will be stored in a contiguous manner, so if we know the base address of the process, we can find out the next addresses.

The Memory Management Unit is a combination of 2 registers –

1. Base Register (Relocation Register)
2. Limit Register.

**Base Register –**contains the starting physical address of the process.  
**Limit Register** -mentions the limit relative to the base address on the region occupied by the process.

The logical address generated by the CPU is first checked by the limit register, If the value of the logical address generated is less than the value of the limit register, the base address stored in the relocation register is added to the logical address to get the physical address of the memory location.  
If the logical address value is greater than the limit register, then the CPU traps to the OS, and the OS terminates the program by giving fatal error.



# Virtual Memory in Operating System

Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program generated addresses are translated automatically to the corresponding machine addresses.  
The size of virtual storage is limited by the addressing scheme of the computer system and amount of secondary memory is available not by the actual number of the main storage locations.

It is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.

1. All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that a process can be swapped in and out of main memory such that it occupies different places in main memory at different times during the course of execution.
2. A process may be broken into number of pieces and these pieces need not be continuously located in the main memory during execution. The combination of dynamic run-time address translation and use of page or segment table permits this.

If these characteristics are present then, it is not necessary that all the pages or segments are present in the main memory during execution. This means that the required pages need to be loaded into memory whenever required. Virtual memory is implemented using Demand Paging or Demand Segmentation.

**Demand Paging :**  
The process of loading the page into memory on demand (whenever page fault occurs) is known as demand paging.  
The process includes the following steps :

virtual_mem

1. If CPU try to refer a page that is currently not available in the main memory, it generates an interrupt indicating memory access fault.
2. The OS puts the interrupted process in a blocking state. For the execution to proceed the OS must bring the required page into the memory.
3. The OS will search for the required page in the logical address space.
4. The required page will be brought from logical address space to physical address space. The page replacement algorithms are used for the decision making of replacing the page in physical address space.
5. The page table will updated accordingly.
6. The signal will be sent to the CPU to continue the program execution and it will place the process back into ready state.

Hence whenever a page fault occurs these steps are followed by the operating system and the required page is brought into memory.

**Advantages :**

* More processes may be maintained in the main memory: Because we are going to load only some of the pages of any particular process, there is room for more processes. This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes will be in the ready state at any particular time.
* A process may be larger than all of main memory: One of the most fundamental restrictions in programming is lifted. A process larger than the main memory can be executed because of demand paging. The OS itself loads pages of a process in main memory as required.
* It allows greater multiprogramming levels by using less of the available (primary) memory for each process.

**Thrashing-**

* Thus when the OS brings one page in, it must throw another out. If it throws out a page just before it is used, then it will just have to get that page again almost immediately. Too much of this leads to a condition called Thrashing. The system spends most of its time swapping pages rather than executing instructions. So a good page replacement algorithm is required.

**Causes of Thrashing :**

1. **High degree of multiprogramming**: If the number of processes keeps on increasing in the memory than number of frames allocated to each process will be decreased. So, less number of frames will be available to each process. Due to this, page fault will occur more frequently and more CPU time will be wasted in just swapping in and out of pages and the utilization will keep on decreasing.
2. **Lacks of Frames**:If a process has less number of frames then less pages of that process will be able to reside in memory and hence more frequent swapping in and out will be required. This may lead to thrashing. Hence sufficient amount of frames must be allocated to each process in order to prevent thrashing.

**Recovery of Thrashing :**

* Do not allow the system to go into thrashing by instructing the long term scheduler not to bring the processes into memory after the threshold.
* If the system is already in thrashing then instruct the mid term schedular to suspend some of the processes so that we can recover the system from thrashing.

1. [Memory Interleaving](https://www.geeksforgeeks.org/operating-system-memory-interleaving/)
2. [Virtual Memory Questions](https://www.geeksforgeeks.org/virtual-memory-questions/)
3. [Page Fault Handling](https://www.geeksforgeeks.org/operating-system-page-fault-handling/)
4. [Fixed (or static) Partitioning in Operating System](https://www.geeksforgeeks.org/fixed-or-static-partitioning-in-operating-system/)
5. [Segmentation](https://www.geeksforgeeks.org/operating-systems-segmentation/)
6. [Memory Segmentation in 8086 Microprocessor](https://www.geeksforgeeks.org/memory-segmentation-8086-microprocessor/)
7. [Program for Next Fit algorithm in Memory Management](https://www.geeksforgeeks.org/program-next-fit-algorithm-memory-management/)
8. [Page Replacement Algorithms](https://www.geeksforgeeks.org/operating-system-page-replacement-algorithm/)
9. [Program for Page Replacement Algorithms | Set 1 ( LRU)](https://www.geeksforgeeks.org/program-page-replacement-algorithms-set-1-lru/)
10. [Program for Optimal Page Replacement Algorithm](https://www.geeksforgeeks.org/program-optimal-page-replacement-algorithm/)
11. [LFU (Least Frequently Used) Cache Implementation](https://www.geeksforgeeks.org/lfu-least-frequently-used-cache-implementation/)
12. [Techniques to handle Thrashing](https://www.geeksforgeeks.org/operating-system-techniques-handle-thrashing/)
13. [Static and Dynamic Libraries | Set 1](https://www.geeksforgeeks.org/static-vs-dynamic-libraries/)
14. [Named Pipe or FIFO with example C program](https://www.geeksforgeeks.org/named-pipe-fifo-example-c-program/)