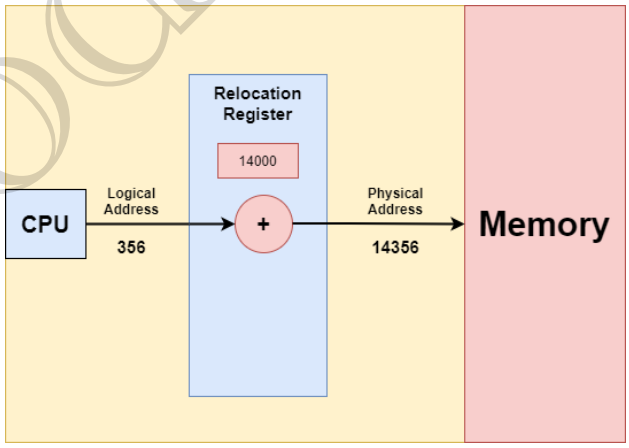
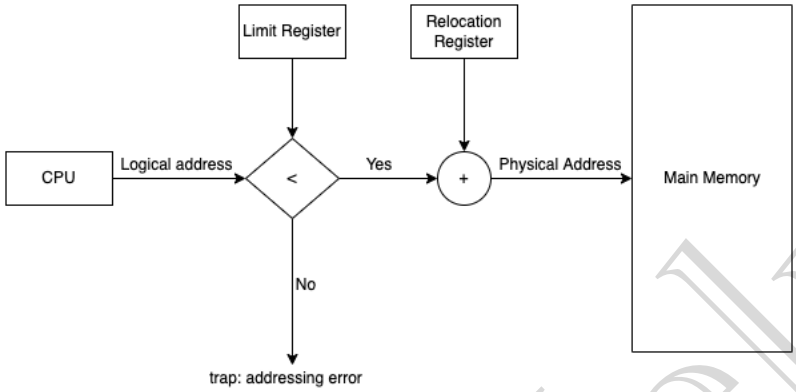
LEC-24: Memory Management Techniques | Contiguous Memory Allocation

1. In Multi-programming environment, we have multiple processes in the main memory (Ready Queue) to keep the CPU utilization high and to make computer responsive to the users.
2. To realize this increase in performance, however, we must keep several processes in the memory; that is, we must **share** the main memory. As a result, we must **manage** main memory for all the different processes.
3. **Logical versus Physical Address Space**
    - **a.** Logical Address
        - **i.** An address generated by the **CPU**.
        - **ii.** The logical address is basically the address of an instruction or data used by a process.
        - **iii.** User can access logical address of the process.
        - **iv.** User has indirect access to the physical address through logical address.
        - **v.** Logical address does not exist physically. Hence, aka, **Virtual address.**
        - vi. The set of all logical addresses that are generated by any program is referred to as Logical Address Space.
        - **vii. Range: 0 to *max*.**
    - b. Physical Address
        - i. An address loaded into the memory-address register of the physical memory.
        - ii. User can never access the physical address of the Program.
        - iii. The physical address is in the memory unit. It's a location in the main memory physically.
        - iv. A physical address can be accessed by a user indirectly but not directly.
        - v. The set of all physical addresses corresponding to the Logical addresses is commonly known as Physical Address Space.
        - vi. It is computed by the **Memory Management Unit (MMU).**
        - **vii. Range: (R + 0) to (R + max), for a base value R.**
    - **c. The runtime mapping from virtual to physical address is done by a hardware device called the memory-management unit (MMU).**
    - d. The user's program mainly generates the logical address, and the user thinks that the program is running in this logical address, but the program mainly needs physical memory in order to complete its execution.
    - e.

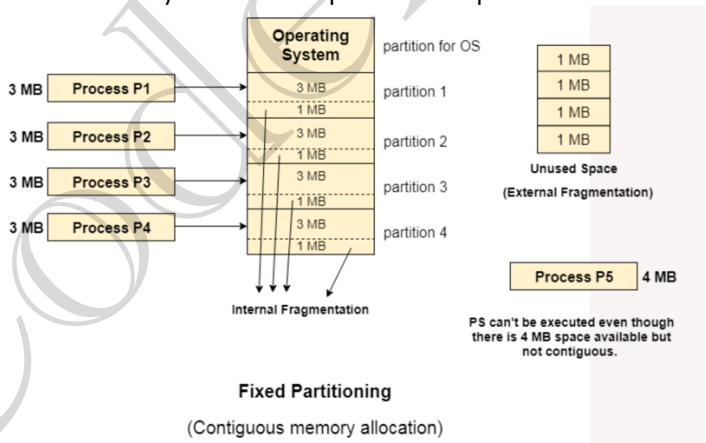


4. How OS manages the isolation and protect? **(Memory Mapping and Protection)**
    - a. OS provides this Virtual Address Space (VAS) concept.
    - b. To separate memory space, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses.
    - c. The relocation register contains value of smallest physical address (Base address [R]); the limit register contains the range of logical addresses (e.g., relocation = 100040 & limit = 74600).
    - d. Each logical address must be less than the limit register.

e. MMU maps the logical address dynamically by adding the value in the relocation register.
f. When CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of the context switch. Since every address generated by the CPU (Logical address) is checked against these registers, we can protect both OS and other users' programs and data from being modified by running process.
g. Any attempt by a program executing in user mode to access the OS memory or other uses' memory results in a trap in the OS, which treat the attempt as a fatal error.
h. **Address Translation**



5. **Allocation Method on Physical Memory**
    a. Contiguous Allocation
    b. Non-contiguous Allocation
6. **Contiguous Memory Allocation**
    a. In this scheme, each process is contained in a single contiguous block of memory.
    b. Fixed Partitioning
        i. The main memory is divided into partitions of equal or different sizes.

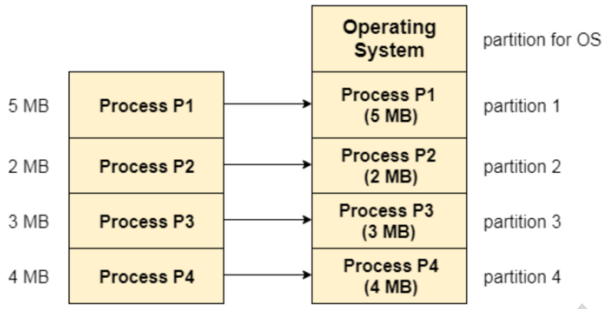


        ii.
        iii. Limitations:
            1. **Internal Fragmentation:** if the size of the process is lesser then the total size of the partition then some size of the partition gets wasted and remain unused. This is wastage of the memory and called internal fragmentation.
            2. **External Fragmentation:** The total unused space of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form.
            3. Limitation on process size: If the process size is larger than the size of maximum sized partition then that process cannot be loaded into the memory. Therefore, a limitation can be imposed on the process size that is it cannot be larger than the size of the largest partition.

4. Low degree of multi-programming: In fixed partitioning, the degree of multiprogramming is fixed and very less because the size of the partition cannot be varied according to the size of processes.

c. Dynamic Partitioning

i. In this technique, the partition size is not declared initially. It is declared at the time of process loading.



Dynamic Partitioning

(Process Size = Partition Size)

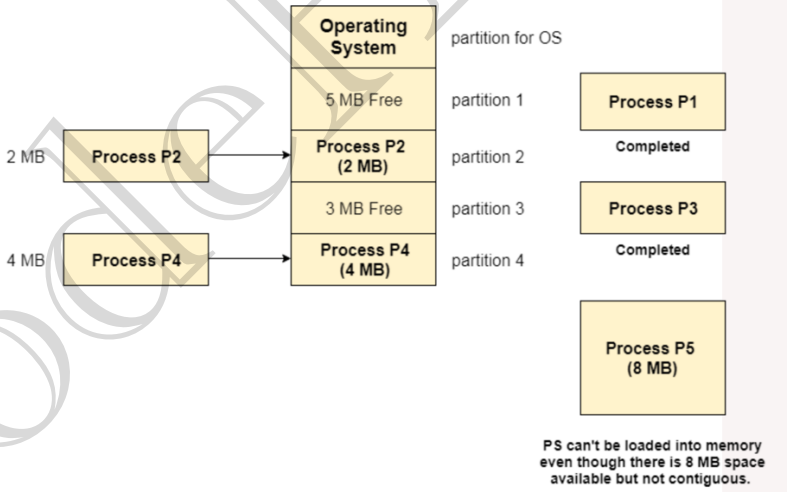
ii.

iii. Advantages over fixed partitioning

1. No internal fragmentation
2. No limit on size of process
3. Better degree of multi-programming

iv. Limitation

1. External fragmentation



External Fragmentation in Dynamic Partitioning

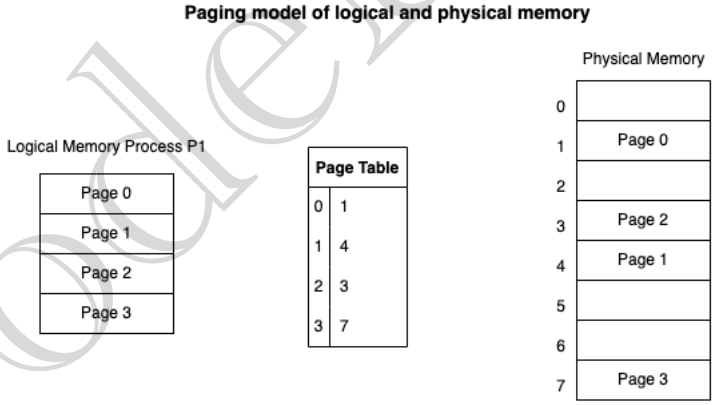
X

LEC-25: Free Space Management

1. **Defragmentation/Compaction**
    a. Dynamic partitioning suffers from external fragmentation.
    b. Compaction to minimize the probability of external fragmentation.
    c. All the free partitions are made contiguous, and all the loaded partitions are brought together.
    d. By applying this technique, we can store the bigger processes in the memory. The free partitions are merged which can now be allocated according to the needs of new processes. This technique is also called **defragmentation**.
    e. The efficiency of the system is decreased in the case of compaction since all the free spaces will be transferred from several places to a single place.
2. **How free space is stored/represented in OS?**
    a. Free holes in the memory are represented by a free list (Linked-List data structure).
3. **How to satisfy a request of a of n size from a list of free holes?**
    a. Various algorithms which are implemented by the Operating System in order to find out the holes in the linked list and allocate them to the processes.
    b. First Fit
        i. Allocate the first hole that is big enough.
        ii. Simple and easy to implement.
        iii. Fast/Less time complexity
    c. Next Fit
        i. Enhancement on First fit but starts search always from last allocated hole.
        ii. Same advantages of First Fit.
    d. Best Fit
        i. Allocate smallest hole that is big enough.
        ii. Lesser internal fragmentation.
        iii. May create many small holes and cause major external fragmentation.
        iv. Slow, as required to iterate whole free holes list.
    e. Worst Fit
        i. Allocate the largest hole that is big enough.
        ii. Slow, as required to iterate whole free holes list.
        iii. Leaves larger holes that may accommodate other processes.

LEC-26: Paging | Non-Contiguous Memory Allocation

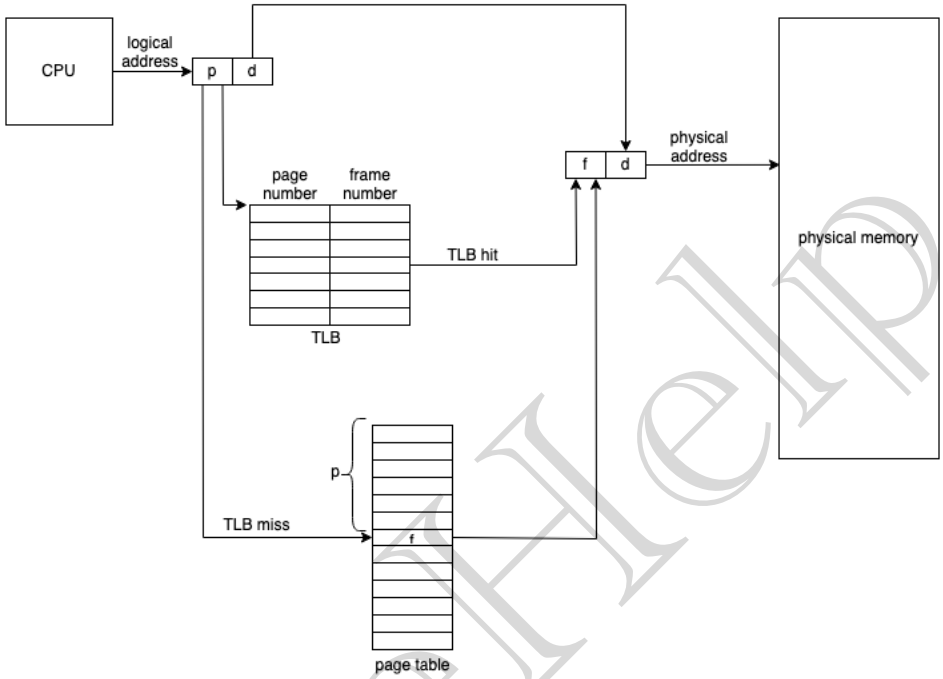
1. The main **disadvantage of Dynamic partitioning is External Fragmentation.**
    a. Can be removed by Compaction, but with overhead.
    b. We need more dynamic/flexible/optimal mechanism, to load processes in the partitions.
2. **Idea behind Paging**
    a. If we have only two small non-contiguous free holes in the memory, say 1KB each.
    b. If OS wants to allocate RAM to a process of 2KB, in contiguous allocation, it is not possible, as we must have contiguous memory space available of 2KB. (External Fragmentation)
    c. What if we divide the process into 1KB-1KB blocks?
3. Paging
    a. Paging is a memory-management scheme that permits the physical address space of a process to be non-contiguous.
    b. It avoids external fragmentation and the need of compaction.
    c. Idea is to divide the physical memory into fixed-sized blocks called **Frames**, along with divide logical memory into blocks of same size called **Pages**. (# Page size = Frame size)
    d. **Page size** is usually determined by the processor architecture. Traditionally, pages in a system had uniform size, such as 4,096 bytes. However, processor designs often allow two or more, sometimes simultaneous, page sizes due to its benefits.
    e. Page Table
        i. A Data structure stores which page is mapped to which frame.
        ii. The page table contains the base address of each page in the physical memory.
    f. Every address generated by CPU (logical address) is divided into two parts: a page number (p) and a page offset (d). The p is used as an index into a page table to get base address the corresponding frame in physical memory.



    g. Page table is stored in main memory at the time of process creation and its base address is stored in process control block (**PCB**).
    h. A page table base register (**PTBR**) is present in the system that points to the current page table. Changing page tables requires only this one register, at the time of context-switching.
4. **How Paging avoids external fragmentation?**
    a. Non-contiguous allocation of the pages of the process is allowed in the random free frames of the physical memory.
5. **Why paging is slow and how do we make it fast?**
    a. There are too many memory references to access the desired location in physical memory.
6. **Translation Look-aside buffer (TLB)**
    a. A Hardware support to speed-up paging process.
    b. It's a hardware cache, high speed memory.
    c. TBL has key and value.

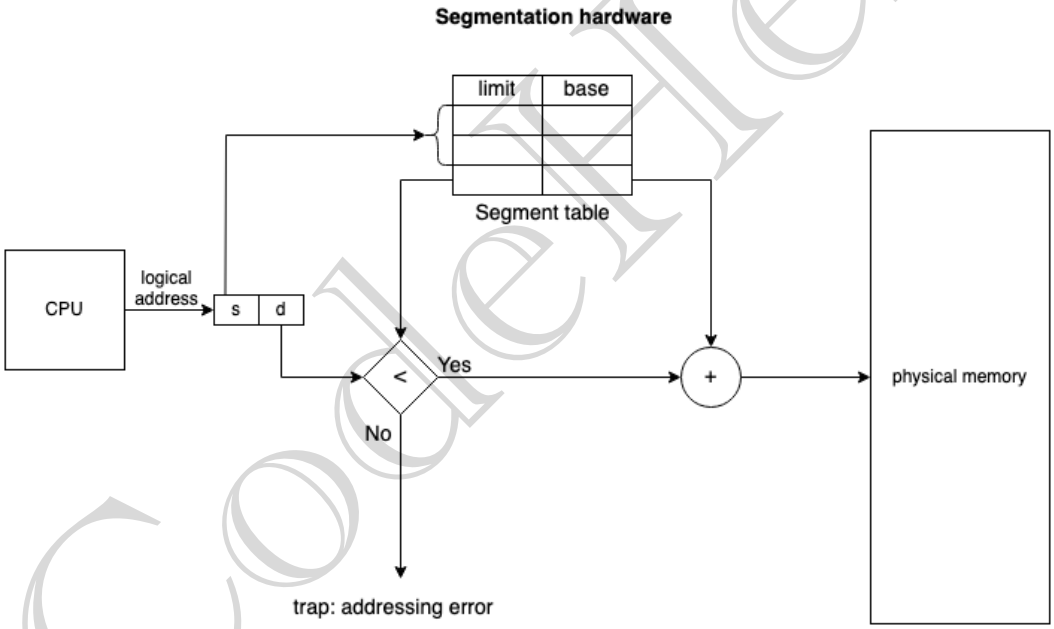
d. Page table is stores in main memory & because of this when the memory references is made the translation is slow.
e. When we are retrieving physical address using page table, after getting frame address corresponding to the page number, we put an entry of the into the TLB. So that next time, we can get the values from TLB directly without referencing actual page table. Hence, make paging process faster.

Paging hardware with TLB



f. TLB hit, TLB contains the mapping for the requested logical address.
g. **Address space identifier (ASIDs)** is stored in each entry of TLB. ASID uniquely identifies each process and is used to **provide address space protection and allow to TLB to contain entries for several different processes.** When TLB attempts to resolve virtual page numbers, it ensures that the ASID for the currently executing process matches the ASID associated with virtual page. If it doesn't match, the attempt is treated as TLB miss.

LEC-27: Segmentation | Non-Contiguous Memory Allocation

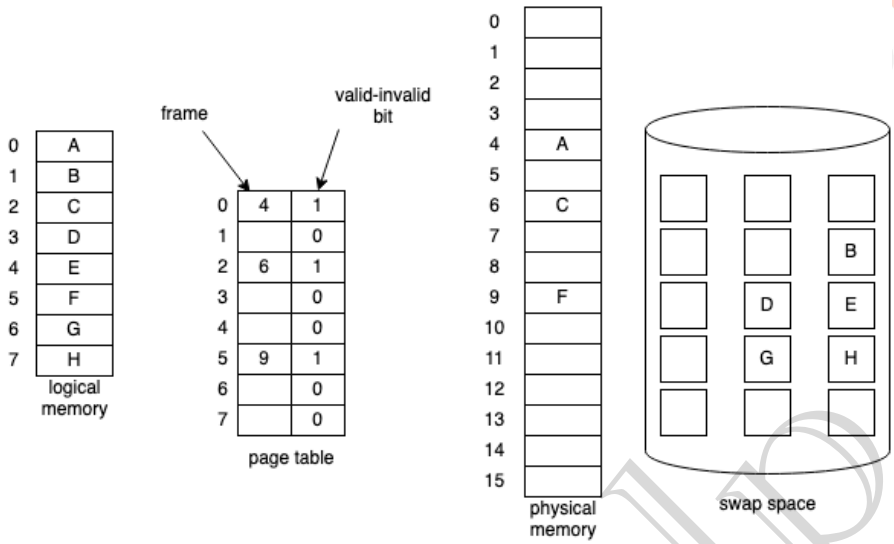
1. An important aspect of memory management that become unavoidable with paging is separation of user's view of memory from the actual physical memory.
2. Segmentation is memory management technique that supports the **user view of memory.**
3. A logical address space is a collection of segments, these segments are based on **user view** of logical memory.
4. Each segment has **segment number and offset,** defining a segment.
   <segment-number, offset> {s,d}
5. Process is divided into **variable segments based on user view.**
6. **Paging** is closer to the Operating system rather than the **User**. It divides all the processes into the form of pages although a process can have some relative parts of functions which need to be loaded in the same page.
7. Operating system doesn't care about the **User's view** of the process. It may **divide the same function into different page**s and those **pages may or may not be loaded at the same time into the memory.** It decreases the efficiency of the system.
8. It is better to have segmentation which divides the process into the segments. Each segment contains the same type of functions such as the main function can be included in one segment and the library functions can be included in the other segment.
9. 

10. **Advantages**:
    a. No internal fragmentation.
    b. One segment has a contiguous allocation, hence efficient working within segment.
    c. The size of segment table is generally less than the size of page table.
    d. It results in a more efficient system because the compiler keeps the same type of functions in one segment.
11. **Disadvantages**:
    a. External fragmentation.
    b. The different size of segment is not good that the time of swapping.
12. Modern System architecture provides both segmentation and paging implemented in some hybrid approach.

LEC-28: What is Virtual Memory? || Demand Paging || Page Faults

1. **Virtual memory** is a technique that allows the execution of processes that are not completely in the memory. It provides user an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory. (Swap-space)
2. **Advantage** of this is, programs can be larger than physical memory.
3. It is required that instructions must be in physical memory to be executed. But it limits the size of a program to the size of physical memory. In fact, in many cases, the entire program is not needed at the same time. So, we want an ability to execute a program that is only partially in memory would give many benefits:
    a. A program would no longer be constrained by the amount of physical memory that is available.
    b. Because each user program could take less physical memory, more programs could be run at the same time, with a corresponding **increase in CPU utilization and throughput.**
    c. Running a program that is not entirely in memory would benefit **both the system and the user.**
4. Programmer is provided very large virtual memory when only a smaller physical memory is available.
5. **Demand Paging** is a popular method of **virtual memory management.**
6. In demand paging, the pages of a process which are least used, get stored in the secondary memory.
7. A page is copied to the main memory when its demand is made, or **page fault** occurs. There are various **page replacement algorithms** which are used to determine the pages which will be replaced.
8. Rather than swapping the entire process into memory, we use **Lazy Swapper.** A lazy swapper never swaps a page into memory unless that page will be needed.
9. We are viewing a process as a sequence of pages, rather than one large contiguous address space, using the term **Swapper is technically incorrect**. A swapper manipulates entire processes, whereas a **Pager** is concerned with individual pages of a process.
10. **How Demand Paging works?**
    a. When a process is to be swapped-in, the pager guesses which pages will be used.
    b. Instead of swapping in a whole process, the pager brings only those pages into memory. This, it avoids reading **into memory pages that will not be used anyway.**
    c. Above way**, OS decreases the swap time and the amount of physical memory needed.**
    d. The **valid-invalid bit scheme in the page table** is used to distinguish between pages that are in memory and that are on the disk.
        i. Valid-invalid bit **1** means, the associated page is both legal and in memory.
        ii. Valid-invalid bit **0** means, the page either is not valid (not in the LAS of the process) or is valid but is currently on the disk.

Page table when some pages are not in memory



e.

f. If a process never attempts **to access some invalid bit page, the process will be executed successfully without even the need pages present in the swap space.**

g. What happens if the process tries to access a page that was not brought into memory, access to a page marked invalid causes **page fault.** Paging hardware noticing invalid bit for a demanded page will cause **a trap to the OS.**

h. The procedure to handle the page fault:

i. Check an internal table (in PCB of the process) to determine whether the reference was valid or an invalid memory access.

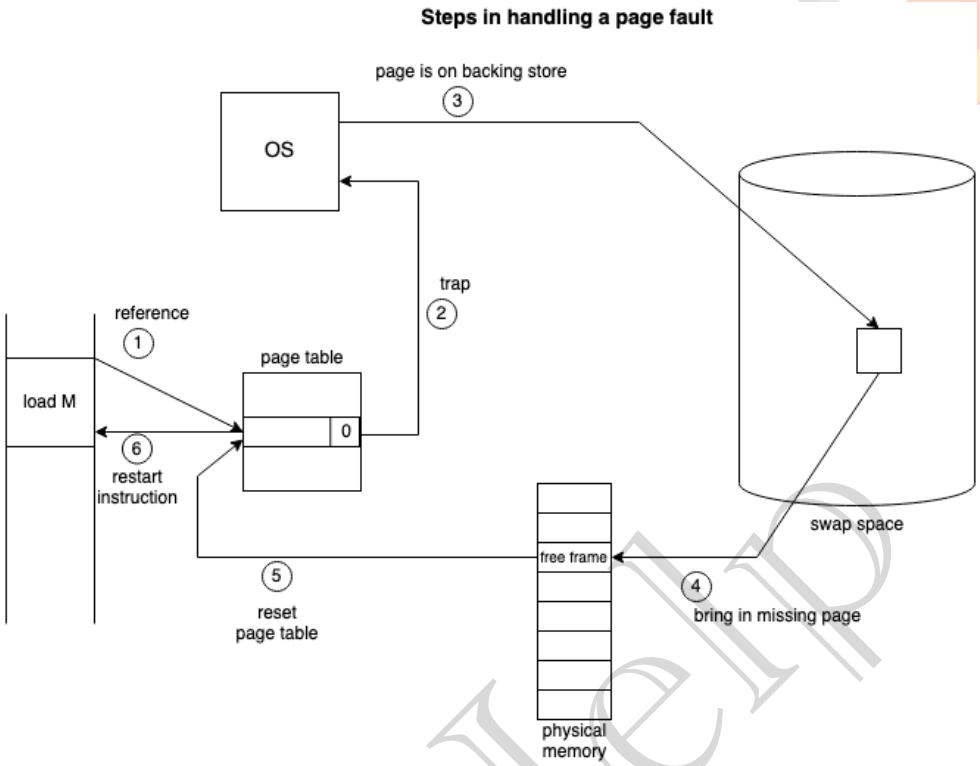
ii. If ref. was invalid process throws exception.
If ref. is valid, pager will swap-in the page.

iii. We find a free frame (from free-frame list)

iv. Schedule a disk operation to read the desired page into the newly allocated frame.

v. When disk read is complete, we modify the page table that, the page is now in memory.

vi. Restart the instruction that was interrupted by the trap. The process can now access the page as through it had always been in memory.



i.

j. **Pure Demand Paging**
    i. In extreme case, we can start executing a process with no pages in memory. When OS sets the instruction pointer to the first instruction of the process, which is not in the memory. The process immediately faults for the page and page is brought in the memory.
    ii. Never bring a page into memory until it is required.
k. We use **locality of reference** to bring out reasonable performance from demand paging.

11. **Advantages of Virtual memory**
    a. The degree of multi-programming will be increased.
    b. User can run large apps with less real physical memory.
12. **Disadvantages of Virtual Memory**
    a. The system can become slower as swapping takes time.
    b. **Thrashing** may occur.

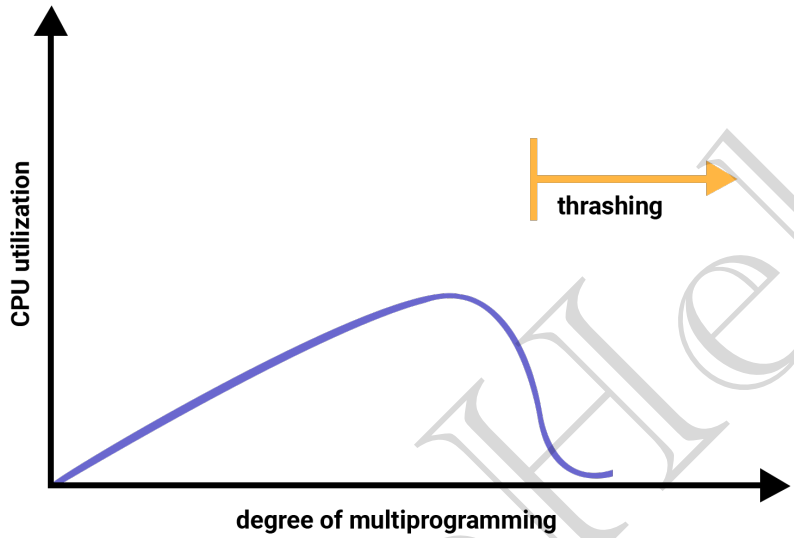
LEC-29: Page Replacement Algorithms

1. Whenever **Page Fault** occurs, that is, a process tries to access a page which is not currently present in a frame and OS must bring the page from swap-space to a frame.
2. OS must do page replacement to accommodate new page into a free frame, but there might be a possibility the system is working in high utilization and all the frames are busy, in that case OS must replace one of the pages allocated into some frame with the new page.
3. The **page replacement algorithm** decides which memory page is to be replaced. Some allocated page is swapped out from the frame and new page is swapped into the freed frame.
4. **Types** of Page Replacement Algorithm: (**AIM** is to have minimum page faults)
    a. **FIFO**
        i. Allocate frame to the page as it comes into the memory by **replacing the oldest page.**
        ii. Easy to implement.
        iii. Performance is **not** always good
            1. The page replaced may be an initialization module that was used long time ago **(Good replacement candidate)**
            2. The page may contain a heavily used variable that was initialized early and is in content use. (**Will again cause page fault)**
        iv. **Belady's** anomaly is present.
            1. **In the case of LRU and optimal page replacement algorithms, it is seen that the number of page faults will be reduced if we increase the number of frames.** However, Balady found that, In FIFO page replacement algorithm, the number of page faults will get increased with the increment in number of frames.
            2. This is the strange behavior shown by FIFO algorithm **in some of the cases.**
    b. **Optimal** page replacement
        i. Find if a page that is never referenced in future. If such a page exists, replace this page with new page.
        If no such page exists, find a page that is **referenced farthest in future**. Replace this page with new page.
        ii. **Lowest** page fault rate among any algorithm.
        iii. Difficult to implement as **OS requires future knowledge of reference string** which is kind of impossible. (Similar to SJF scheduling)
    c. Least-recently used (**LRU**)
        i. We can use recent past as an approximation of the near future then we can replace the page that has not been used for the longest period.
        ii. Can be implemented by two ways
            1. **Counters**
                a. Associate time field with each page table entry.
                b. Replace the page with smallest time value.
            2. **Stack**
                a. Keep a stack of page number.
                b. Whenever page is referenced, it is removed from the stack & put on the top.
                c. By this, most recently used is always on the top, & least recently used is always on the bottom.
                d. As entries might be removed from the middle of the stack, so Doubly linked list can be used.
    d. **Counting**-based page replacement – Keep a counter of the number of references that have been made to **each** page. (Reference counting)

i. Least frequently used (**LFU**)
    1. Actively used pages should have a large reference count.
    2. Replace page with the smallest count.
ii. Most frequently used (**MFU**)
    1. Based on the argument that the page with the smallest count was probably just brought in and has yet to be used.
iii. Neither MFU nor LFU replacement is common.

LEC-30: Thrashing

1. **Thrashing**
    a. If the process doesn't have the number of frames it needs to support pages in active use, it will quickly page-fault. At this point, it must replace some page. However, since all its pages are in active use, it must replace a page that will be needed again right away. Consequently, it quickly faults again, and again, and again, replacing pages that it must bring back in immediately.
    b. This **high paging activity is called Thrashing.**
    c. A system is Thrashing when it **spends more time servicing the page faults than executing processes.**



    d. **Technique to Handle Thrashing**
        i. **Working set model**
            1. This model is based on the concept of the **Locality** Model.
            2. The basic principle states that if we allocate enough frames to a process to accommodate its current locality, it will only fault whenever **it moves to some new locality.** But if the allocated frames are lesser than the size of the current locality, **the process is bound to thrash.**
        ii. **Page Fault frequency**
            1. **Thrashing** has a high page-fault rate.
            2. We want to **control** the page-fault rate.
            3. When it is too high, the process needs more frames. Conversely, if the page-fault rate is too low, then the process may have too many frames.
            4. We establish upper and lower bounds on the desired page fault rate.
            5. If pf-rate exceeds the upper limit, allocate the process another frame, if pf-rate fails falls below the lower limit, remove a frame from the process.
            6. By controlling pf-rate, thrashing can be prevented.