(Deadlock theory questions not present in prev year SEE papers, but still there in syllabus)



Access to a page marked invalid causes a page fault. The paging hardware, in translating the address through the page table, will notice that the **invalid bit** is set, causing a **trap** to the operating system.

The procedure for handling this page fault is straightforward (Figure 9.6):

1. We check an internal table (usually kept with the process control block) for this process to determine whether the reference was a valid or an invalid memory access.

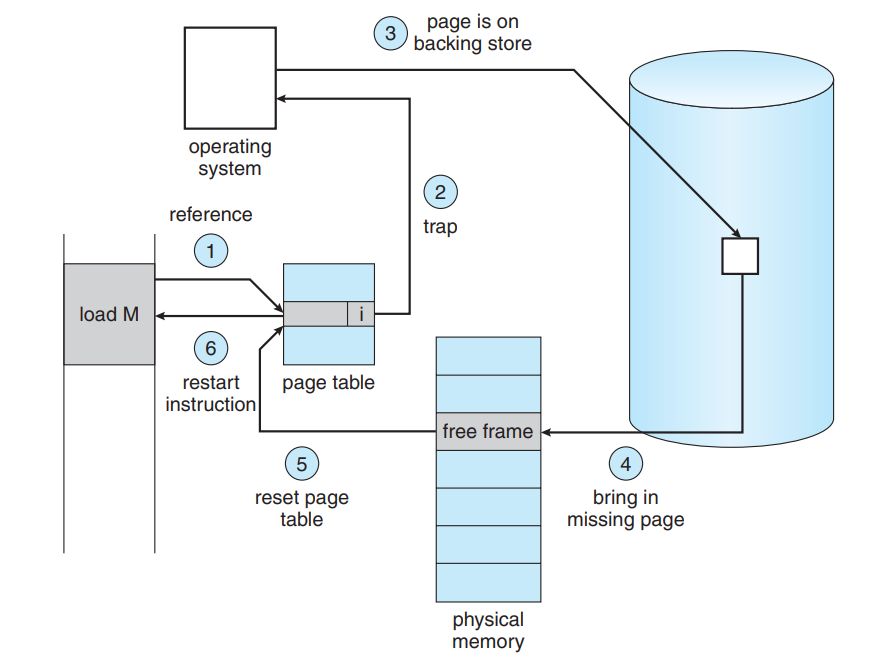
2. If the reference was invalid, we terminate the process. If it was valid but we have not yet brought in that page, we now page it in.

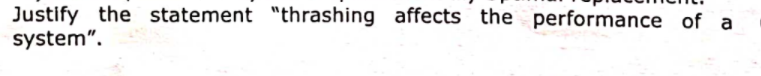
3. We find a free frame (by taking one from the free-frame list, for example).

4. We schedule a disk operation to read the desired page into the newly allocated frame.

5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.

6. We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory



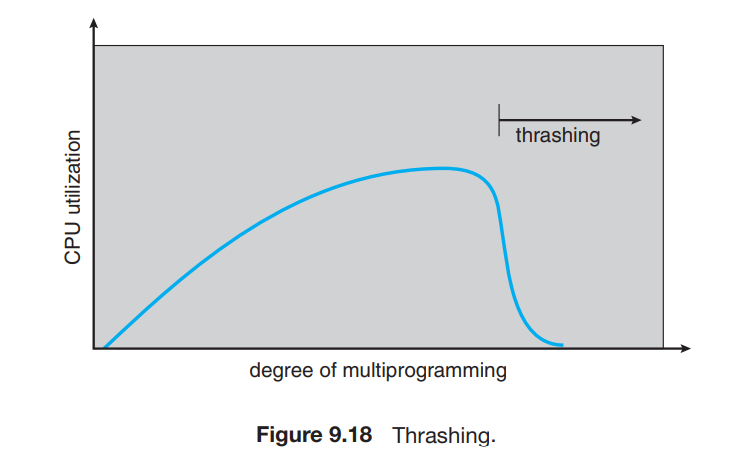


A process is **thrashing** if it is spending more time paging than executing

Thrashing results in severe performance problems. Consider the following scenario, the operating system monitors CPU utilization. If CPU utilization is too low, we **increase the degree of multiprogramming** (describes the maximum number of processes that a single-processor system can accommodate efficiently). Now suppose that a process enters a new phase in its execution and needs more frames. It starts faulting and taking frames away from other processes. These processes need those pages, however, and so they also fault, taking frames from other processes. These faulting processes must use the paging device to swap pages in and out. As they queue up for the paging device, the ready queue empties. As processes wait for the paging device, CPU utilization decreases.

The CPU scheduler sees the decreasing CPU utilization and increases the degree of multiprogramming as a result. The new process tries to get started by taking frames from running processes, causing more page faults and a longer queue for the paging device. As a result, CPU utilization drops even further, and the CPU scheduler tries to increase the degree of multiprogramming even more. Thrashing has occurred, and system throughput plunges. The pagefault rate increases tremendously. As a result, the effective memory-access time increases. No work is getting done, because the processes are spending all their time paging.

This phenomenon is illustrated in Figure 9.18, in which CPU utilization is plotted against the degree of multiprogramming. As the degree of multiprogramming increases, CPU utilization also increases, although more slowly, until a maximum is reached. If the degree of multiprogramming is increased even further, thrashing sets in, and CPU utilization drops sharply. At this point, to increase CPU utilization and stop thrashing, we must decrease the degree of multiprogramming

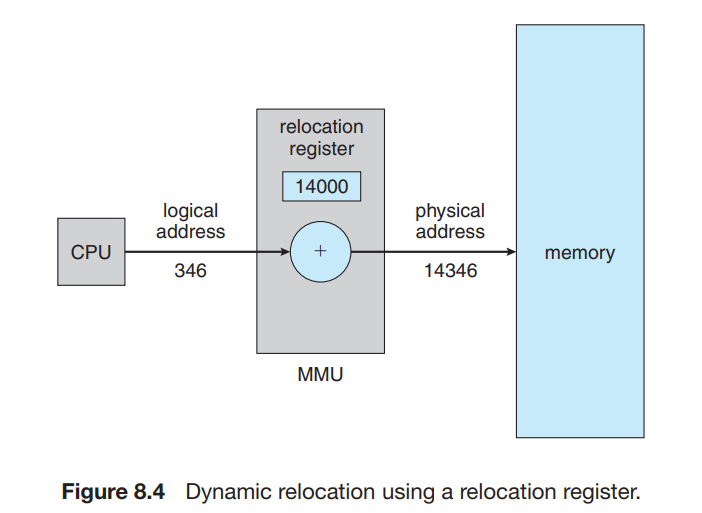


**With a neat diagram show the translation of a given virtual address to its equivalent physical address.**

The run-time mapping from virtual to physical addresses is done by a hardware device called the **memory-management unit (MMU)**.MMU will generate a relocation register (base register) for eg: 14000. 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 (see Figure 8.4).

For example, if the base is at 14000, then an attempt by the user to address location 0 is dynamically relocated to location 14000; an access to location 346 is mapped to location 14346.

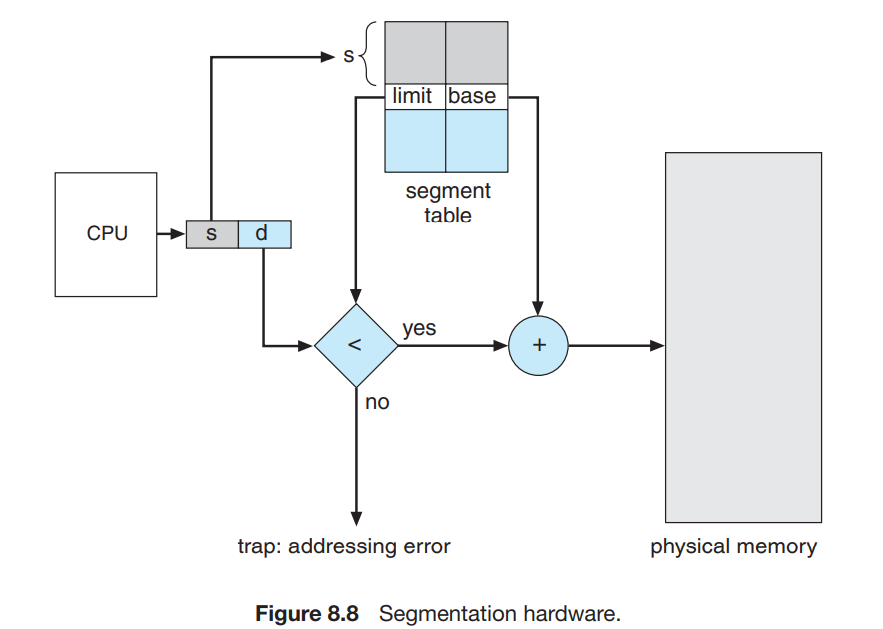
The user program generates only logical addresses. However, these logical addresses must be mapped to physical addresses before they are used.

****

**Illustrate with an example the address translation for segmentation technique of memory virtualization.**

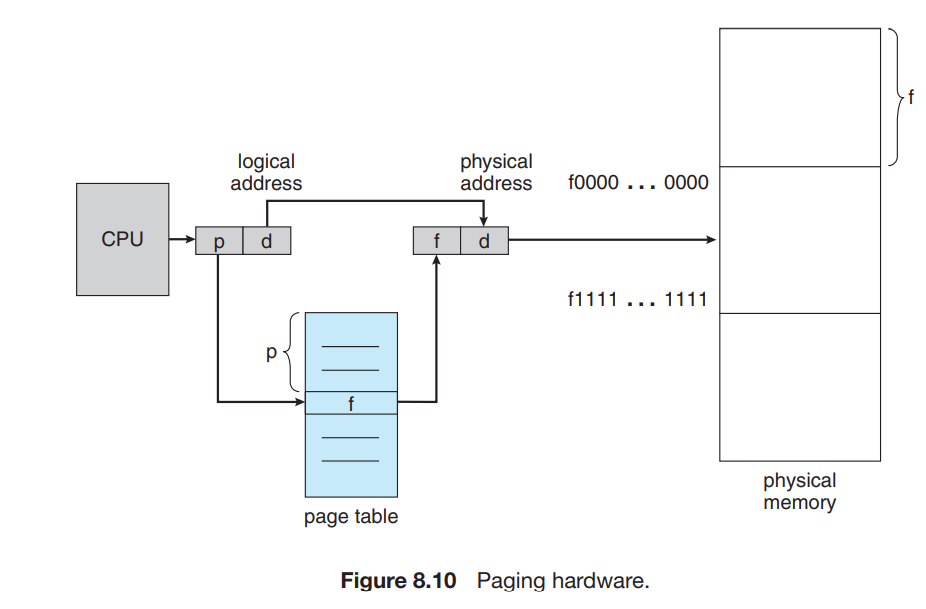
we define an implementation to map two-dimensional user-defined addresses into one-dimensional physical address. This mapping is effected by a **segment table**. Each entry in the segment table has a **segment base** and a **segment limit**. The segment base contains the starting physical address where the segment resides in memory, and the segment limit specifies the length of the segment

The use of a segment table is illustrated in Figure 8.8. A logical address consists of two parts: a **segment number, s,** and an **offset into that segment**, **d**. The segment number is used as an index to the segment table. The offset d of the logical address must be between 0 and the segment limit. If it is not, we release a trap to the operating system (logical addressing attempt beyond end of segment). When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte



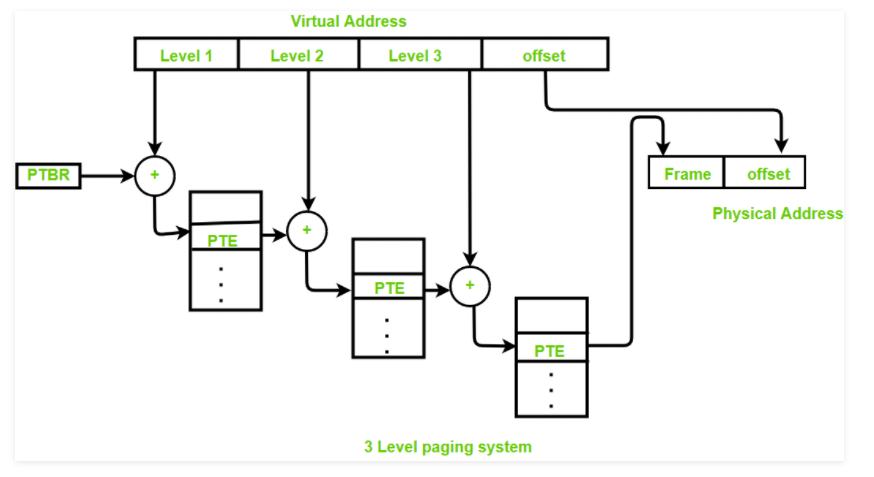
**With a code snippet, explain how to access memory using paging technique.**

Every address generated by the CPU is divided into two parts: a **page number (p)** and a **page offset (d).** The page number is used as an index into a page table. The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.

****

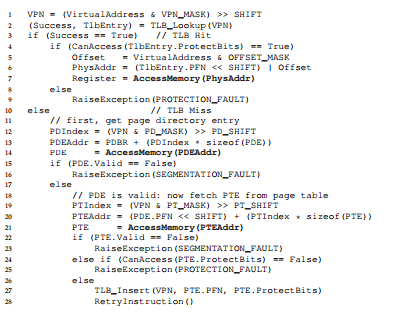
**Code snippet??**

**Illustrate multi level page table with a suitable example.**



Multilevel Paging is a paging scheme which consists of two or more levels of page tables in a hierarchical manner. It is also known as hierarchical paging. The entries of the level 1 page table are pointers to a level 2 page table and entries of the level 2 page tables are pointers to a level 3 page table and so on. The entries of the last level page table store actual frame information. Level 1 contains a single page table and the address of that table is stored in PTBR (Page Table Base Register).

In multilevel paging whatever may be levels of paging all the page tables will be stored in main memory.So it requires more than one memory access to get the physical address of page frame. One access for each level needed. Each page table entry except the last level page table entry contains base address of the next level page table.

**Write Multi-level Page Table Control Flow algorithm**

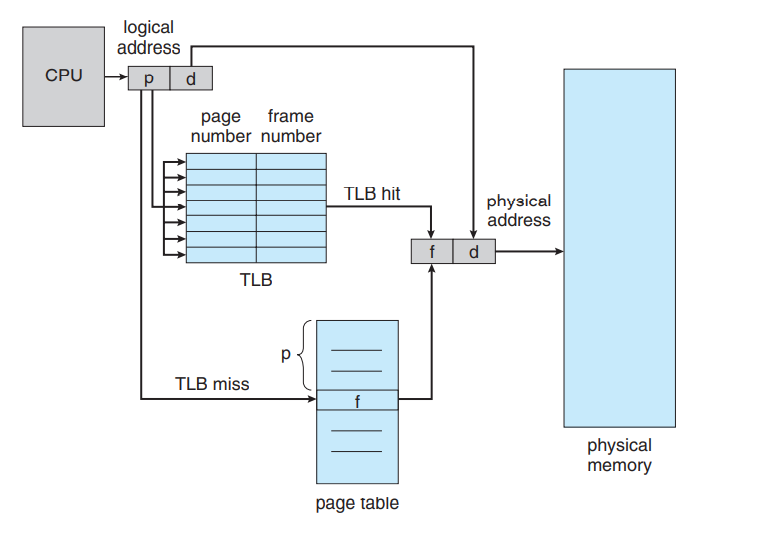
(I hope we don’t have this algorithm in syllabus…)(Pretty Sure We don’t)

**explain how TLB can speed up memory access.**

TLB (Translation lookaside buffer) is associative, high-speed memory. Each entry in the TLB consists of two parts: **a key (page number)** and **a value(frame number)**. When the associative memory is presented with an item, the item is compared with all keys simultaneously. If the item is found, the corresponding value field is returned. The search is fast

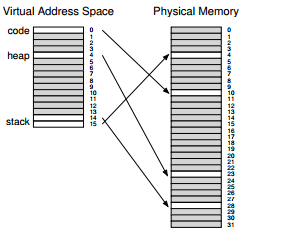
The TLB contains only a few of the page-table entries. When a logical address is generated by the CPU, its page number is presented to the TLB. If the page number is found, its frame number is immediately available and is used to access memory. If the page number is not in the TLB (known as a **TLB miss**), a memory reference to the page table must be made and the TLB must be updated with the new frame number accordingly.

In addition, we add the page number and frame number to the TLB, so that they will be found quickly on the next reference. If the TLB is already full of entries, an existing entry must be selected for replacement.

**Differentiate between fixed-size and variable-size techniques of partitioning the memory.**

| S.NO. | Fixed partitioning | Variable partitioning |
| --- | --- | --- |
| **1.** | **In multi-programming with fixed partitioning the main memory is divided into fixed sized partitions.** | **In multi-programming with variable partitioning the main memory is not divided into fixed sized partitions.** |
| **2.** | **Only one process can be placed in a partition.** | **In variable partitioning, the process is allocated a chunk of free memory.** |
| **3.** | **It does not utilize the main memory effectively.** | **It utilizes the main memory effectively.** |
| **4.** | **There is presence of internal fragmentation and external fragmentation.** | **There is external fragmentation.** |
| **5.** | **Degree of multi-programming is less.** | **Degree of multi-programming is higher.** |
| **6.** | **It is more easier to implement.** | **It is less easier to implement.** |
| **7.** | **There is limitation on size of process.** | **There is no limitation on size of process.** |

**With an example show how hybrid approach of paging minimizes the space required to store page table.**

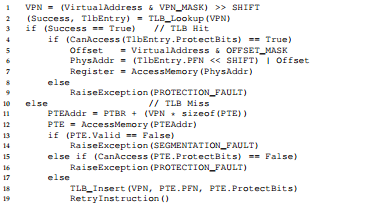


Hybrid uses both paging and segmentation techniques.

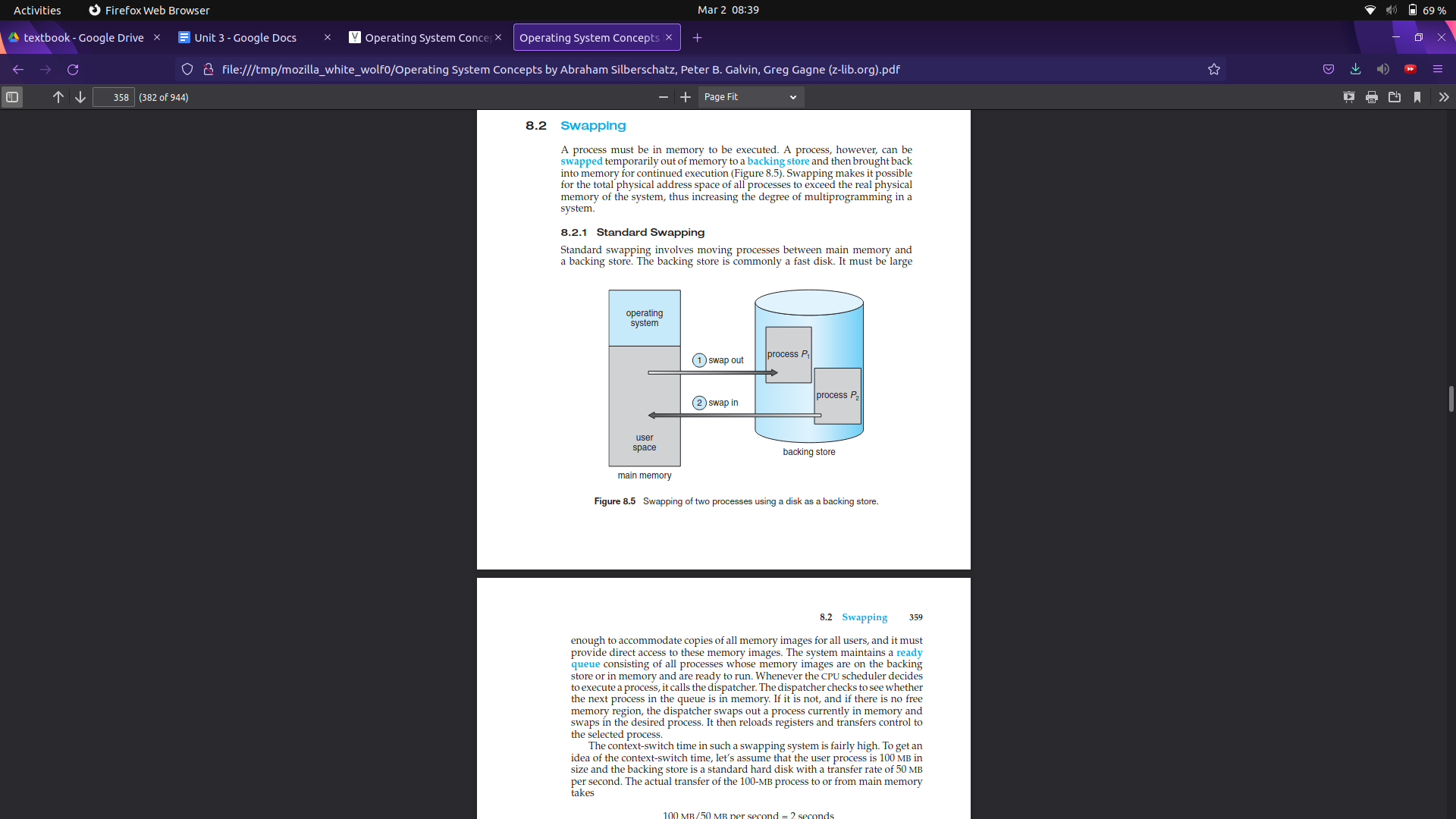
It divides virtual memory into no of pages and then categorizes pages into segments.

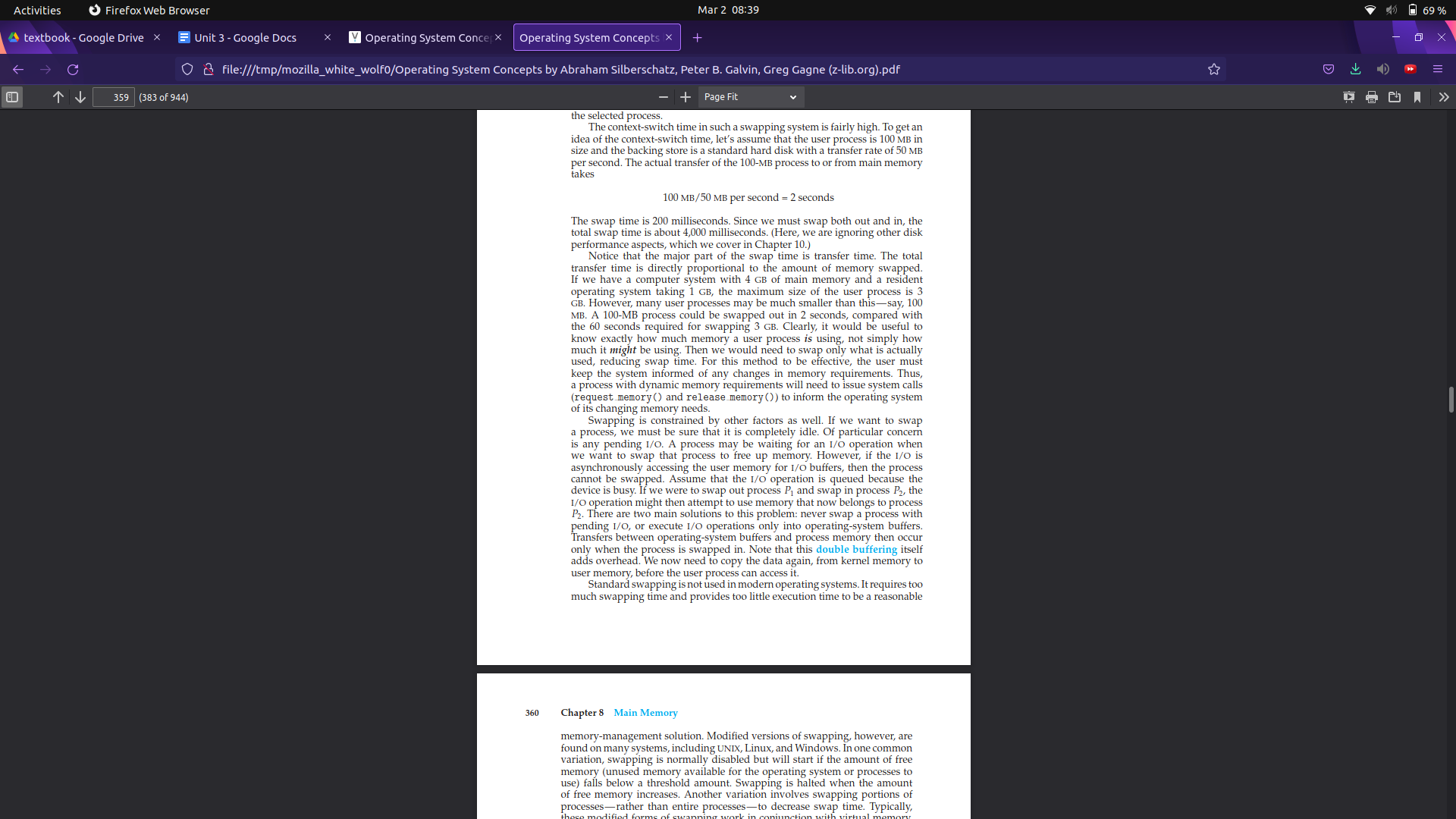
For address translation first identifies the segment and then the page in that segment.

**Hardware Page-Fault Control Flow Algorithm. (Not in syllabus mostly)**



**Explain Swapping of two processes using backing store**



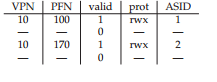
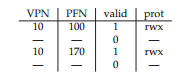


**Differentiate between fine-grained and coarse-grained segmentation techniques.**

**??**

**Discuss the issues and solutions related to the context switching, in implementing Translation Look Aside Buffer (TLB). (Is it there in syllabus???)**

The TLB contains virtual-to-physical translations that are only valid for the currently running process; these translations are not meaningful for other processes. As a result, when switching from one process to another, the hardware or OS (or both) must be careful to ensure that the about-to-be-run process does not accidentally use translations from some previously run process.



**Given five memory partitions of 150 KB, 600 KB, 250 KB, 350 KB, and 650 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of 258 KB, 477 KB, 136 KB, and 524 KB (in order)? Which algorithm makes the most efficient use of memory?**

First Fit

258 477 136 524

600 650 150 nil

Best Fit

258 477 136 524

350 600 150 650

Worst Fit

258 477 136 524

650 600 350 nil

Least Wastage and Max Efficiency =Best Fit

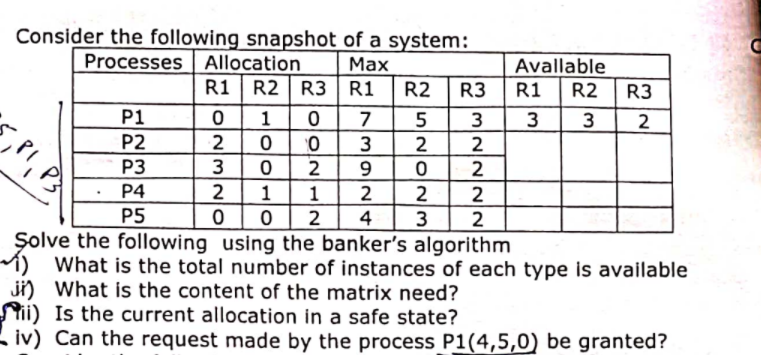
Faster=First Fit

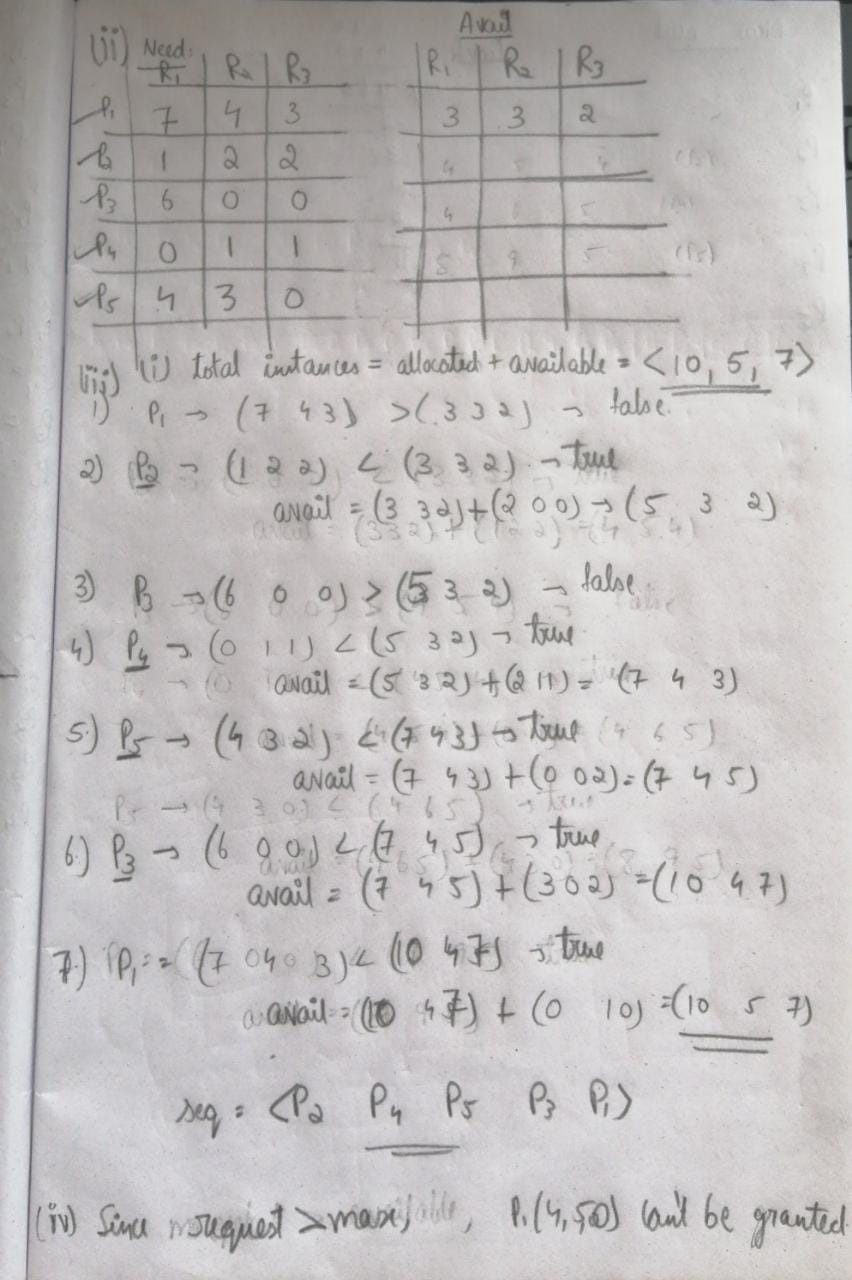
**With a neat diagram, describe briefly address space layout created by OS to use abstraction of physical memory.**

**??**

[A Memory Abstraction: Address Spaces](https://gcallah.github.io/OperatingSystems/AddressSpaces.html#:~:text=A%20solution%20is%2C%20to%20use,from%20actual%20physical%20memory%20location).

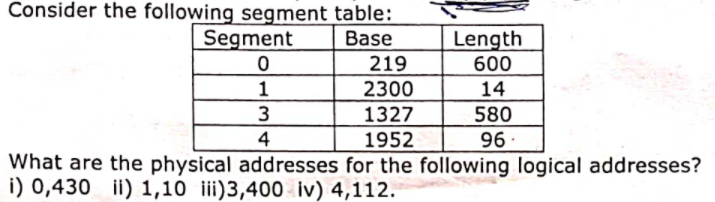
I think we need to explain how the OS maps logical addresses ( virtual) to physical address.





According to the resource-request algorithm(pg 332 text book ,under deadlock chapter)

If the request P1(4,5,0) is granted,it exceeds the maximum claim, hence it cant be granted.



1) base 219

Offset 430<length(600)

Physical address=219+430=649

2) base 2300

Offset 10<length(14)

Physical address=2300+10=2310

3) base 1327

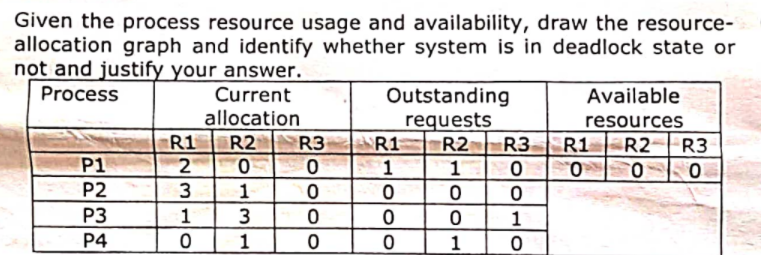
Offset 400<length(580)

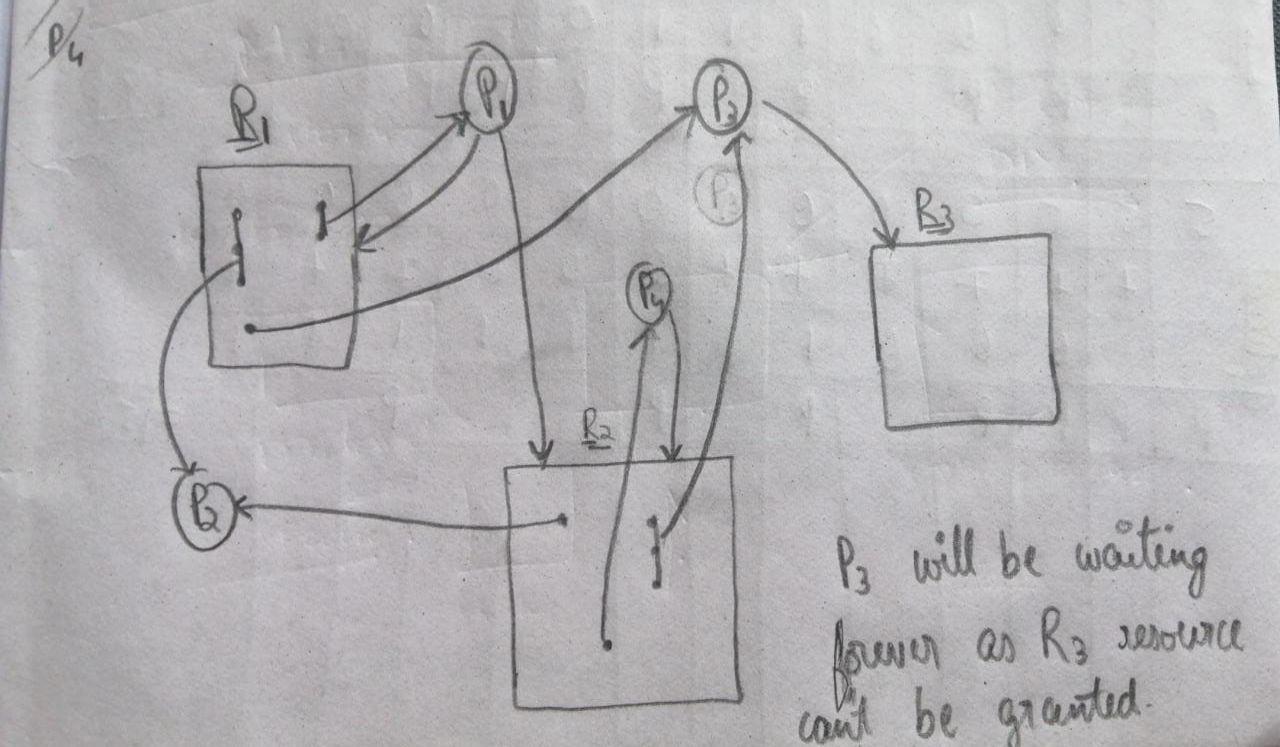
Physical address=1327+400=1727

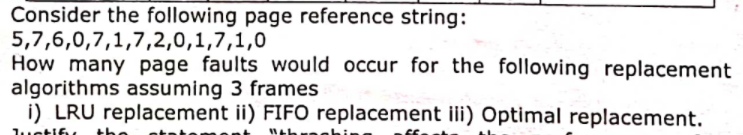
4) base 1952

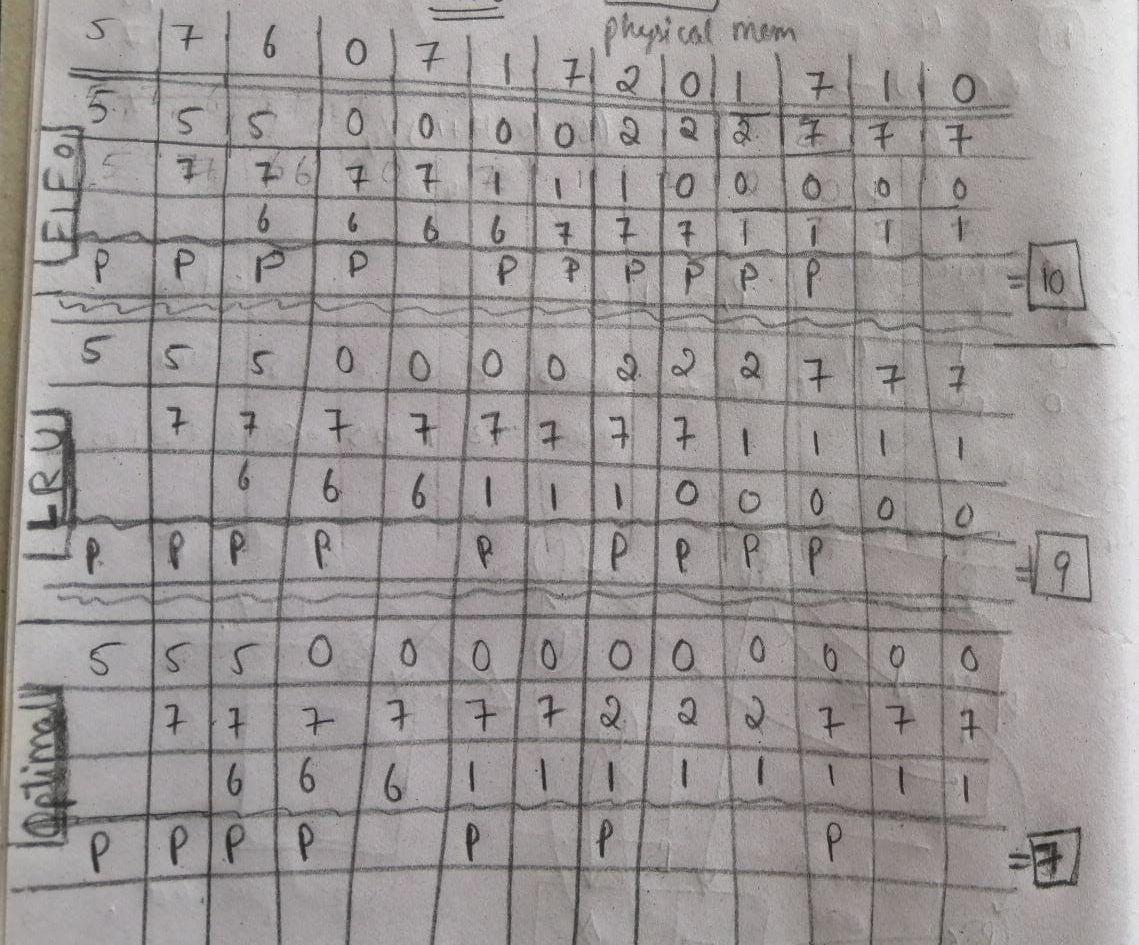
Offset 112>length(96)

No Possible Address









**A paging system has the following parameters: 2^32 bytes of physical memory, paging size of 2^10 bytes, 2^16 pages of logical address space and each page entry takes 4bytes. Answer the following using these data**

**i) How many bits are there in logical address?**

**ii)What is the size of logical and physical memory?**

**ii) How many bits are there in physical address?**

**iii) How many entries are there in page table?**

**iv) What is the size of the page table?**

**Imagine in a multilevel paging scheme, system maintains a small address space of size 16KB, with 64-byte pages. The pages 0, 1, 25, 254 and 255 contains valid data. Answer the following with respect to multilevel paging.**

**i) Write the no of bits for each filed in virtual address.**

**iii) How many page tables are possible and how many entries are in each page table?**

**iv) What is the size of the page table?**

**v) How many entries are there in directory?**

**vi) Find the physical address of the valid pages.**