What is page fault? Explain the handling of the validity page fault

- A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM.
- This can happen for various reasons, such as:
  - Page Not Present: The page that the program is trying to access has not been loaded into physical memory.
  - Page Protection Violation: The program is trying to access a page of memory for which it does not have the necessary permissions (e.g., trying to write to a read-only page).
  - Invalid Memory Access: The program is trying to access memory that is outside the bounds of its allocated address space.
- There are two types of page faults: **validity faults and protection faults**. The fault handlers may have to read a page from disk to memory and sleep during the I/O operation, therefore, they are an exception to the general rule that interrupt handlers cannot sleep.

Validity Fault :

- If a process attempts to access a page whose valid bit is not set, it incurs a validity fault and then the kernel invokes the validity fault handler.

Validity Fault Handler :

Step-by-step explanation of the **vfault** algorithm:

1. **Input**: The algorithm takes the address where the process faulted as input.
2. **Find Region and Page Table Entry**: The algorithm first finds the memory region to which the faulting address belongs and locates the corresponding page table entry.
3. **Check Address Validity**: It checks if the faulting address is outside the valid virtual address space of the process. If so, it sends a segmentation violation signal (**SIGSEGV**) to the process, indicating an invalid memory access, and exits (**goto out**).
4. **Check Page Validity**: If the page table entry indicates that the page is now valid (i.e., another process may have loaded it while waiting), the algorithm exits (**goto out**).
5. **Handle Page in Cache**:
   - If the page is found in the cache, it is removed from the cache, and the page table entry is adjusted.
   - If the page's contents are not yet valid (i.e., another process faulted first), the algorithm sleeps until the contents become valid.
6. **Handle Page Not in Cache**:
   - If the page is not in the cache, a new page is assigned to the memory region, and the page is put into the cache.
   - If the page was not previously loaded and is marked as "demand zero," the assigned page is cleared to zero.
   - Otherwise, if the page is on a swap device or in an executable file, it is read from disk, and the algorithm sleeps until the I/O operation is complete.

7. **Wake Up Processes**: Once the page contents are valid, any processes waiting for the page are awakened.
8. **Set Page Valid Bit**: Finally, the page's valid bit is set in the page table, indicating that it is now present in memory.
9. **Clear Modify Bit and Page Age**: Optionally, the algorithm clears the page modify bit and page age for the newly loaded page.
10. **Recalculate Process Priority**: Optionally, the algorithm may recalculate the priority of the process that incurred the page fault based on the newly loaded page.
11. **Unlock Region**: The region lock is released.

This algorithm ensures that the requested page is brought into memory from disk if necessary and coordinates the necessary operations to make the page accessible to the requesting process.

What is demand paging? Explain the data structures used for demand paging.
Explain the data structures for demand paging.

- Demand paging can be described as a memory management technique that is used in operating systems to improve memory usage and system performance.
- Demand paging is a technique used in virtual memory systems where pages enter main memory only when requested or needed by the CPU.
- In demand paging, the operating system loads only the necessary pages of a program into memory at runtime, instead of loading the entire program into memory at the start.
- A page fault occurred when the program needed to access a page is not currently in memory. It is managed transparently by the operating system while the process continues execution.
- Machines whose memory architectures is based on pages and whose CPU has restartable instructions can support a kernel that implements a demand paging algorithm, swapping pages of memory between main memory and a swap device.

Working Set Model Approximation:

- Systems use a working set model, approximating the pages a process needs based on recent references.
- A reference bit is set when a page is accessed, marking it as part of the working set.
- Periodic sampling helps track recently referenced pages, ensuring they stay in memory.

The kernel contains 4 major data structures to support low-level memory management functions and demand paging:

1. page table entries
2. disk block descriptors
3. page frame data table (called pfdata for short)
4. swap-use table

Page Table Entries:

- Each process has its own page table that maps virtual addresses to physical addresses.
- Page table entries contain information such as the physical address of the page, protection bits, and additional bit fields for demand paging support.

Disk Block Descriptors:

- Associated with each page table entry.
- Describes the disk copy of the virtual page.
- Contains information about the location of the page contents on the disk, including logical device numbers, block numbers, and special conditions like "demand fill" or "demand zero."

Page Frame Data Table (pfdata):

- Describes each page of physical memory and is indexed by page number.
- Contains fields such as:
  - State of the page: Indicates whether the page is on a swap device, executable file, or if DMA (Direct Memory Access) is underway for the page.
  - Reference count: Number of valid page table entries referencing the page.
  - Logical device and block number containing a copy of the page.
  - Pointers to other pfdata table entries on a list of free pages and on a hash queue of pages.

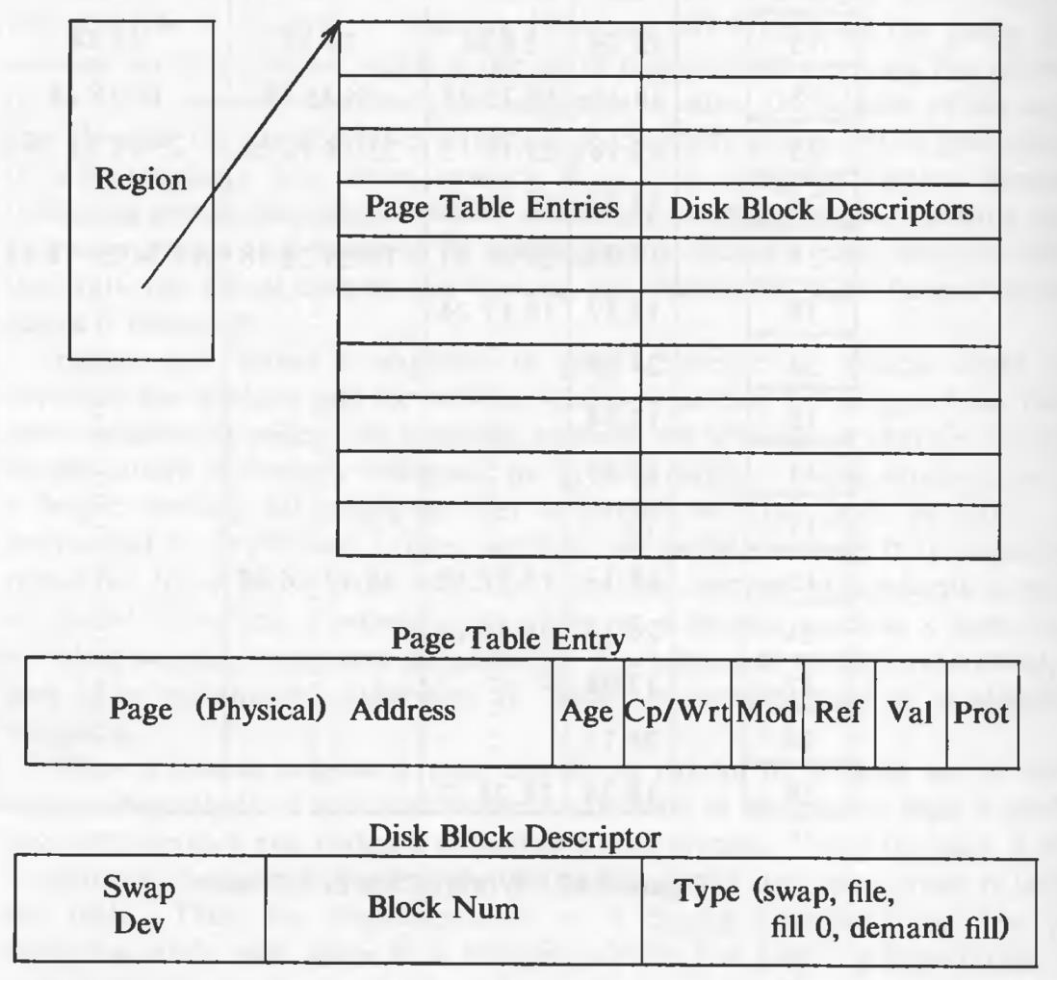
Swap-Use Table:

- Contains an entry for every page on a swap device.
- Keeps track of how many page table entries point to a page on a swap device using a reference count.

The region contains page tables to access physical memory. Each entry of a page table contains the physical address of the page, protection bits indicating whether processes can read, write or execute from the page, and the following bit fields to support demand paging:

- **Valid** - This bit indicates whether the contents of the page are currently legal or valid.
- **Reference** - The reference bit is used to track whether a process has recently accessed or referenced the page.
- **Modify** - This bit indicates whether the contents of the page have been modified by a process since it was last loaded into memory.
- **Copy on write -** The copy-on-write bit is used during process forking
- **Age -** The age bits are used to track how long a page has been in memory or how recently it has been accessed.

The structure of the region is given below:



Explain the working of the page stealer process

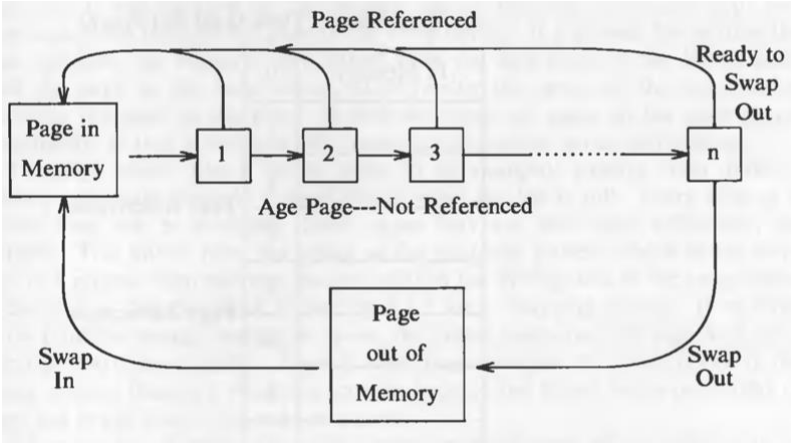
- The pages that are eligible for swapping are found by the Page-Stealer' and places the page numbers in a list which contains the pages to be swapped.
- The page stealer is a kernel process that swaps out memory pages that are no longer part of the working set of a process.
- It is created on system initialization and is invoked throughout the system lifetime when system is low on free pages.
- It examines every active, unlocked region and increments the age field of all valid pages. The kernel locks a region when a process faults on a page in the region, so that the page stealer cannot steal the page being faulted in.

Working of page stealer –

1. **Page Aging and State Transition**:
    - Pages in memory are categorized into two states: aging and eligible for swapping. The aging state indicates that a page has been recently accessed by a process. The page stealer keeps track of the number of passes since a page was last referenced.
    - Substates within the aging state correspond to the number of passes made by the page stealer. When the pass count exceeds a threshold, the page transitions to the state where it becomes eligible for swapping.
2. **Triggering Page Stealer**:
    - The kernel wakes up the page stealer when the available memory falls below a low-water mark. It continues swapping out pages until the available memory rises above a high-water mark.
3. **Determining Pages for Swapping**:
    - The page stealer decides which pages to swap out based on factors such as pass count and memory usage.
    - If a page has not been modified since being loaded into memory and has a copy on a swap device, its page table entry's valid bit is cleared, and the page is put on the free list for future allocation.
    - If a page has been modified, it is scheduled for swapping, and its space on the swap device is freed.
4. **Swapping Pages**:
    - The page stealer maintains a list of pages to be swapped out. When this list is full, the kernel writes the pages to the swap device. If contiguous space is not available, pages are swapped out one at a time, leading to fragmentation.
    - When a page is written to the swap device, its page table entry's valid bit is turned off, and its reference count in the pfdata table is decremented. If the count reaches 0, the pfdata table entry is placed on the free list.
    - The kernel allocates swap space, saves the swap address in the disk block descriptor, and increments the swap-use table count for the page.

5. **Handling Page Faults**:
    - If a process incurs a page fault while the page is on the free list, the kernel can retrieve the page from memory instead of the swap space.
    - However, if the page is on the swap list, it will still be swapped out if needed.
6. **Example Operation**:
    - For example, if the page stealer swaps out pages from multiple processes and writes them to the swap device, the sequence of swapping operations is determined based on factors such as memory usage and access patterns.

The figures show how a page ages:



Explain the swapping of a process between swap space and main memory.

- Swapping is a fundamental memory management technique utilized within Unix operating systems to manage multiple processes efficiently when the physical memory (RAM) is either insufficient or needs to be optimized for performance.
- It is a technique of removing a process from the main memory and storing it into secondary memory, and then bringing it back into the main memory for continued execution.
- Swapping in OS is one of those schemes which fulfill the goal of maximum utilization of CPU and memory management by swapping in and swapping out processes from the main memory.

The Unix Swap Space :

- Swap or paging space is basically a portion of the hard disk that the operating system can use as an extension of the available RAM. This space can be allocated with a partition or a simple file.
- This space is pre-configured during the system setup and acts as an overflow area when the main memory is fully utilized.
- On UNIX platforms, you must ensure that you have adequate free disk space that is configured to be used as swap space.

Triggering the Swapping Process:

- Swapping typically occurs when there is no sufficient RAM available to handle the current workload of active processes.
- This situation can lead to swapping out less critical processes to the swap space, thereby freeing up RAM for higher priority processes

There are two important concepts in the process of swapping which are as follows:

1. Swap In

2. Swap Out

Swap In and Swap Out in OS

Swap In:

The method of removing a process from secondary memory (Hard Drive) and restoring it to the main memory (RAM) for execution is known as the Swap In method.

Swap Out:

- It is a method of bringing out a process from the main memory(RAM) and sending it to the secondary memory(hard drive) so that the processes with higher priority or more memory consumption will be executed known as the Swap Out method.
- During the swap out process, the operating system selects processes based on certain criteria (like least recently used), and moves them from the main memory to the swap space

Management of Swap Space:

- Unix systems typically manage swap space using either a dedicated swap partition or a swap file on the system.
- The operating system continuously monitors the usage of RAM and swap space to decide whether to perform swapping and which process to swap in or out.

The advantages of the swapping method are listed as follows:

- Swapping in OS helps in achieving the goal of Maximum CPU Utilization.
- Swapping ensures proper memory availability for every process that needs to be executed.
- Swapping helps avoid the problem of process starvation means a process should not take much time for execution so that the next process should be executed.
- CPU can perform various tasks simultaneously with the help of swapping so that processes do not have to wait much longer before execution.
- Swapping ensures proper RAM(main memory) utilization.
- Swapping creates a dedicated disk partition in the hard drive for swapped processes which is called swap space.