MODULE V

5.1 Memory Management

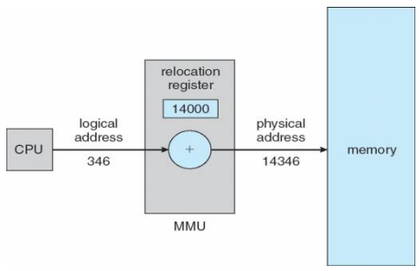
- Memory consists of a large array of words or bytes, each with its own address.
- The CPU fetches instructions from memory according to the value of the program counter.
- Main memory and the registers built into the processor itself are the only storage that the CPU can access directly.
- Each process has a separate memory space.
- There is a range of legal addresses that the process may access and to the process can access only these legal addresses.
- To provide this protection two registers are used.
  - Base register- holds the smallest legal physical memory address.
  - Limit register- specifies the size of the range.

Address Binding

- **Address binding** is the process of mapping the program's logical or virtual a**ddresses** to corresponding physical or main memory **addresses.**
- The binding of instructions and data to memory addresses can be done at any of the following step:
  - **Compile time**
  - **Load time**
  - **Execution time**

Logical versus Physical Address Space

- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management
  - **Logical address(Virtual address)** – generated by the CPU; also referred to as virtual address
  - **Physical address** – address seen by the memory unit
- Logical address space is the set of all logical addresses generated by a program
- Physical address space is the set of all physical addresses generated by a program
- **Memory-Management Unit (MMU) -** Hardware device that at run time maps virtual to physical address.

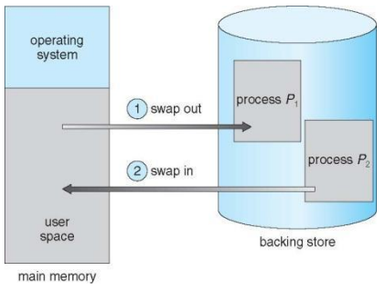


Dynamic relocation using relocation register.

5.2 Swapping

- A process must be in memory to be executed.
- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution
- Swapping variant used for priority-based scheduling algorithms are called roll out, roll in- lower-priority process is swapped out so higher-priority process can be loaded and executed.
- Swapping requires a backing store.
- The **backing store** is commonly a fast disk. It must be large enough to accommodate copies of all memory images for all users, and it must provide direct access to these memory images.

- The system maintains a ready queue consisting of all processes whose memory images are on the backing store or in memory and are ready to run.
- Whenever the CPU scheduler decides to execute a process, it calls the dispatcher.
- The dispatcher checks to see whether the next process in the queue is in memory. If it is not, and if there is no free memory region, the dispatcher swaps out a process currently in memory and swaps in the desired process. It then reloads registers and transfers control to the selected process.



Swapping of two processes using a disk as a backing store.

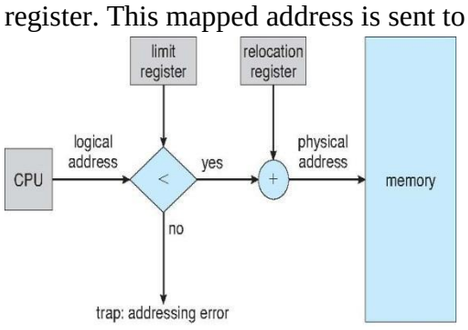
- The context-switch time in such a swapping system is fairly high.
- The major part of the swap time is transfer time. The total transfer time is directly proportional to the amount of memory swapped.
- Other constraints as well on swapping
  - Pending I/O – can't swap out as I/O would occur to wrong process
  - Or always transfer I/O to kernel space, then to I/O device
    - Known as **double buffering**, adds overhead
- Standard swapping not used in modern operating systems, but modified version is common
  - Swap only when free memory extremely low

5.3 Contiguous Memory Allocation

- The main memory must accommodate both the operating system and the various user processes.
- The memory is usually divided into two partitions:
  - one for the resident operating system
  - One for the user processes.
- We can place the operating system in either low memory or high memory.
- Since the interrupt vector is often in low memory, programmers usually place the operating system in low memory.
- Several user processes has to reside in memory at the same time.
- In. contiguous memory allocation, each process is contained in a single contiguous section of memory.

Memory Mapping and Protection

- These features are implemented by using a relocation register, together with a limit register. The relocation register contain the value of the smallest physical address; the limit register contains the range of logical addresses.
- With relocation and limit registers, each logical address must be less than the limit register.
- The MMU maps the logical address dynamically by adding the value in the relocation register. This mapped address is sent to memory



Hardware support for relocation and limit registers.

- ➢ When the 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.
- ➢ Because every address generated by a CPU is checked against these registers, we can protect both the operating system and the other users' programs and data from being modified by this running process.
- ➢ The relocation-register scheme provides an effective way to allow the operating system's size to change dynamically.

Memory Allocation

- ➢ A simplest method for allocating memory is to divide memory into several fixed-sized partitions.

  Each partition may contain exactly one process. Thus, the degree of multiprogramming is bound by the number of partitions. In this multiple partitioned method, when a partition is free; a process is

  selected from the input queue and is loaded into the free partition.When the process terminates, the partition becomes available for another process.
- ➢ In the variable partition scheme, the operating system keeps a table indicating which parts of memory are available and which are occupied. Initially, all memory is available for user processes and is considered one large block of available memory, a **hole**.Eventually, memory contains a set of holes of various sizes.
- ➢ As processes enter the system, they are put into an input queue. The operating system takes into account the memory requirements of each process and the amount of available memory space in determining which processes are allocated memory.
- ➢ When a process is allocated space, it is loaded into memory, and it can then compete for CPU time. When a process terminates, it releases its memory which the operating system may then fill with another process from the input queue.
- ➢ In general, the memory blocks available comprise a set of holes of various sizes scattered throughout memory. When a process arrives and needs memory, the system searches the set for a hole that is large enough for this process.If the hole is too large, it is split into two parts. One part is allocated to the arriving process; the other is returned to the set of holes.
- ➢ When a process terminates, it releases its block of memory, which is then placed back in the set of holes.
- ➢ If the new hole is adjacent to other holes, these adjacent holes are merged to form one larger hole. At this point, the system may need to check whether there are processes waiting for memory and whether this newly freed and recombined memory could satisfy the demands of any of these waiting processes.
- ➢ **Dynamic storage allocation problem** concerns how to satisfy a request of size n from a list of free holes. Solutions for this problems are
  - o **First fit**. Allocate the **first hole** that is big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough.
  - o **Best fit**. Allocate the **smallest hole** that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.
  - o **Worst fit**. Allocate the **largest hole**. Search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best-fit approach
- ➢ Both the first-fit and best-fit strategies for memory allocation suffer from external fragmentation.
- ➢ As processes are loaded and removed from memory, the free memory space is broken into little pieces.

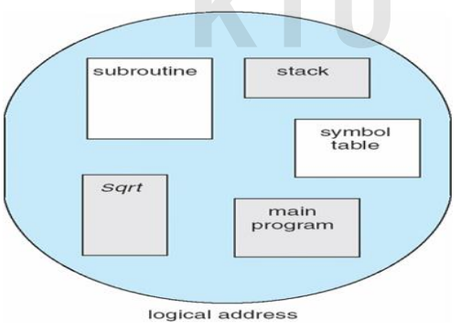
- **External fragmentation** exists when there is enough total memory space to satisfy a request but the available spaces are not contiguous; storage is fragmented into a large number of small holes.
- In the worst case, it could have a block of free (or wasted) memory between every two processes. If all these small pieces of memory were in one big free block instead, it might be able to run several more processes.
- Memory fragmentation can be internal as well as external.
- The general approach to avoiding this problem is to break the physical memory into fixed-sized blocks and allocate memory in units based on block size.
- The memory allocated to a process may be slightly larger than the requested memory. The difference between these two numbers is **internal fragmentation**; memory that is internal to a partition.
- **Compaction** is the solution for external fragmentation- shuffles the memory contents so as to place all free memory together in one large block.

5.4 Segmentation

- A Memory-management scheme that supports user view of memory.

Basic Method

- A program is a collection of segments here.
- A segment is a logical unit such as:
  - main program
  - procedure
  - function
  - method
  - object
  - local variables, global variables
  - common block
  - stack
  - symbol table
  - arrays



User's view of a program.

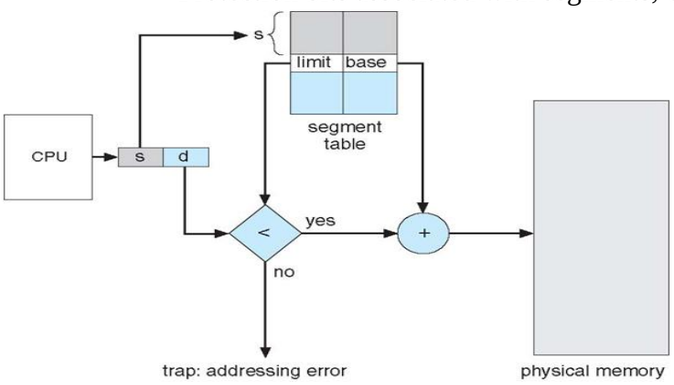
- Segmentation is a memory-management scheme that supports this user view of memory.
- A logical address space is a collection of segments.
- Each segment has a name and a length.
- The addresses specify both the segment name and the offset within the segment.
- a logical address consists of a two tuple:
  <segment-number, offset>.
- Normally, the user program is compiled, and the compiler automatically constructs segments reflecting the input program.

Hardware

- An implementation is needed to map two dimensional user-defined addresses into one-dimensional physical addresses. This mapping is affected by a segmentation table.
- each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory

- **Segment-table length register (STLR)** indicates number of segments used by a program;

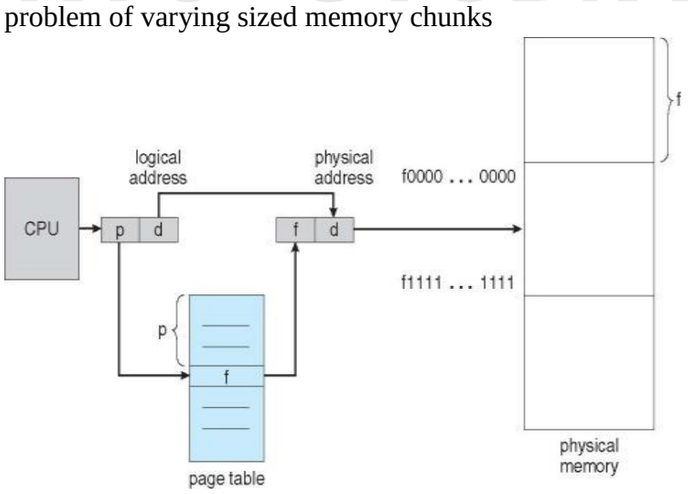
  Segment number ***s*** is legal if ***s*** < **STLR**
- **Protection**
  - With each entry in segment table associate:
    - validation bit = 0 ,means illegal segment
    - read/write/execute privileges
  - Protection bits associated with segments; code sharing occurs at segment



Segmentation hardware

5.5 Paging

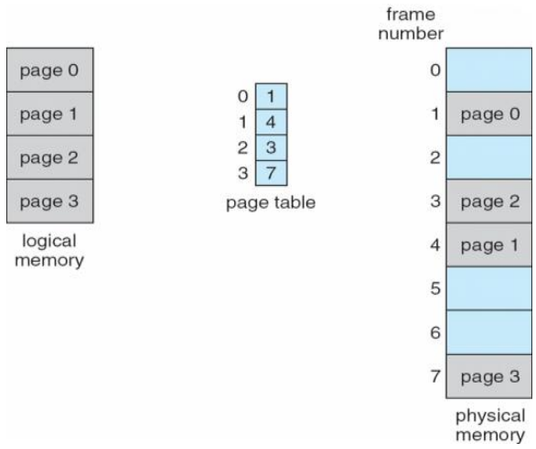
- Paging is a memory-management scheme that permits the physical address space a process to be noncontiguous.
- Paging avoids
  - External fragmentation and the need for compaction.
  - problem of varying sized memory chunks



Paging hardware

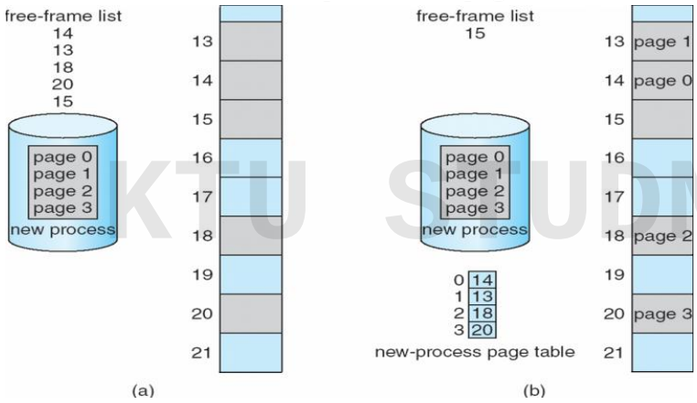
Basic Method

- Breaks physical memory into fixed-sized blocks called **frames** and logical memory into blocks of the same size called **pages**.When a process is to be executed, its pages are loaded into any available memory frames from their source.
- Address generated by cpu is divided into
  - Page number(p)- The page number is used as an index into a **page table**.
  - Page offset(d)- – combined with base address to define the physical memory address that is sent to the memory unit
- The page table contains the base address of each page in physical memory.



Paging model of logical and physical memory.

- ➢ **Paging causes some internal fragmentation**- the last frame allocated may not be completely full for a process.
- ➢ When a process arrives in the system to be executed, its size, expressed in pages, is examined.
- ➢ Each page of the process needs one frame. Thus, if the process requires n pages, at least n frames must be available in memory. If n frames are available, they are allocated to this arriving process.
- ➢ The first page of the process is loaded into one of the allocated frames, and the frame number is put in the page table for this process. The next page is loaded into another frame; its frame number is put into the page table, and so on.



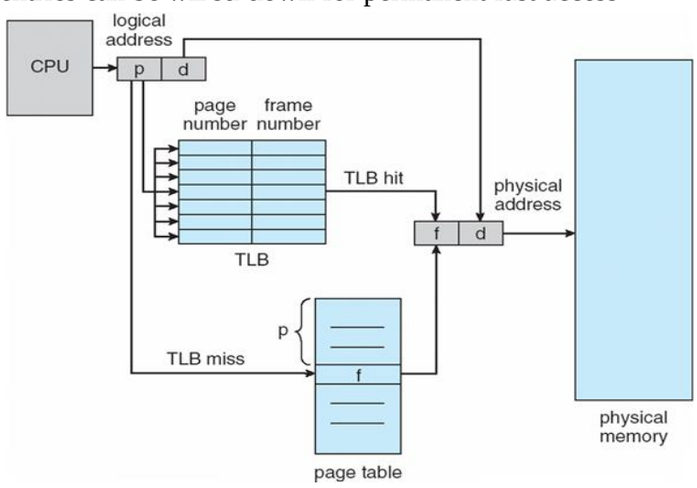
Free frames (a) before allocation and (b) after allocation.

- ➢ An important aspect of paging is the clear separation between the user's view of memory and the actual physical memory. The user program views memory as one single space, containing only this one program.

Hardware Support

- ➢ Each operating system has its own methods for storing page tables. Most allocate a page table for each process. A pointer to the page table is stored with the other register values in the process control block.
- ➢ The hardware implementation of the page table can be done in several ways.
  - o As a set of dedicated registers.
  - o kept in main memory
    - **Page-table base register** (**PTBR**) points to the page table
    - **Page-table length register** (**PTLR**) indicates size of the page table
    - In this scheme every data/instruction access requires two memory accesses
      - One for the page table and one for the data / instruction
      - The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers** (**TLBs**)

- Some TLBs store **address-space identifiers (ASIDs)** in each TLB entry – uniquely identifies each process to provide address-space protection for that process
- Otherwise need to flush at every context switch
- TLBs typically small (64 to 1,024 entries)
- On a TLB miss, value is loaded into the TLB for faster access next time
- Replacement policies must be considered
- Some entries can be **wired down** for permanent fast access



Paging hardware with TLB.

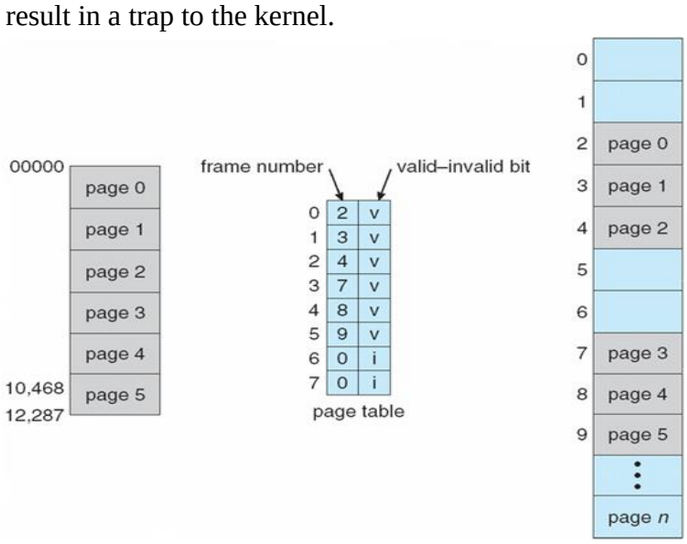
➢ The percentage of times that a particular page number is found in the TLB is called the **hit ratio.**

Protection

➢ Memory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed
  - Can also add more bits to indicate page execute-only, and so on

➢ Valid-invalid bit attached to each entry in the page table:
  - "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page
  - "invalid" indicates that the page is not in the process' logical address space
  - Or use page-table length register (PTLR), to indicate the size of the page table. This value is checked against every logical address to verify that the address is in the valid range for the process.

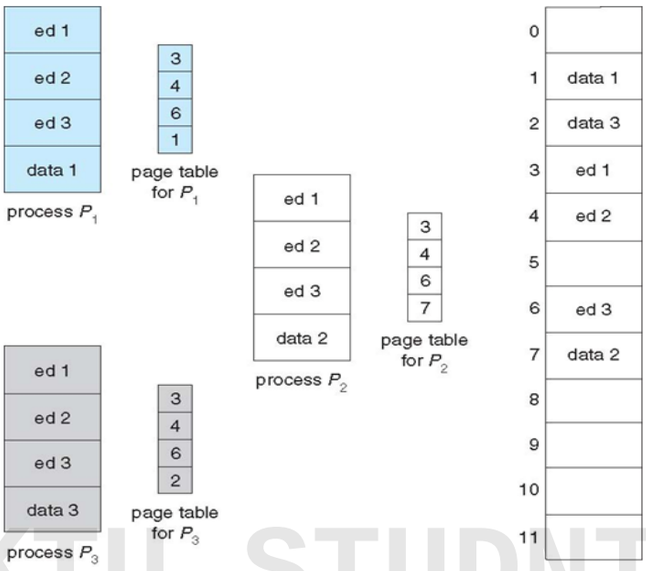
➢ Any violations result in a trap to the kernel.



Valid (v) or invalid (i) bit in a page table.

Shared Pages

- An advantage of paging is the possibility of sharing common code.
- **Shared code**
  - One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems)
  - Similar to multiple threads sharing the same process space
  - Also useful for interprocess communication if sharing of read-write pages is allowed
- **Private code and data**
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space.



Sharing of code in a paging environment

Structure of the Page Table

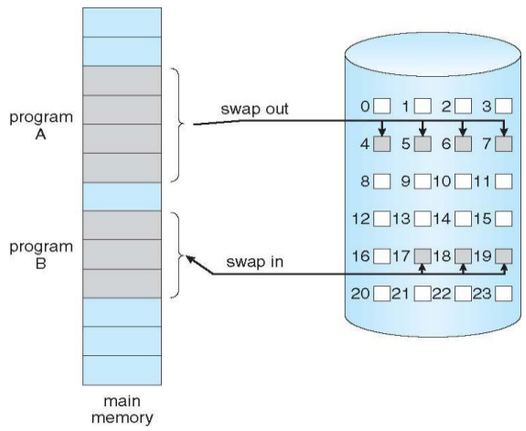
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Virtual memory

- Virtual memory is a technique that allows the execution of processes that are not completely in memory. One major advantage of this scheme is that programs can be larger than physical memory.
- Further, virtual memory abstracts main memory into an extremely large, uniform array of storage, separating logical memory as viewed by the user from physical memory. This technique frees programmers from the concerns of memory-storage limitations.
- Virtual memory also allows processes to share files easily and to implement shared memory

5.6 Demand paging

- Loading the entire program in physical memory at program execution time is not so good because the user doesn't *need* the entire program in memory at a time.
- An alternative strategy is to load pages only as they are needed. This technique is known as **demand paging.**
- With demand-paged virtual memory, pages are only loaded when they are demanded during program execution; pages that are never accessed are thus never loaded into physical memory.

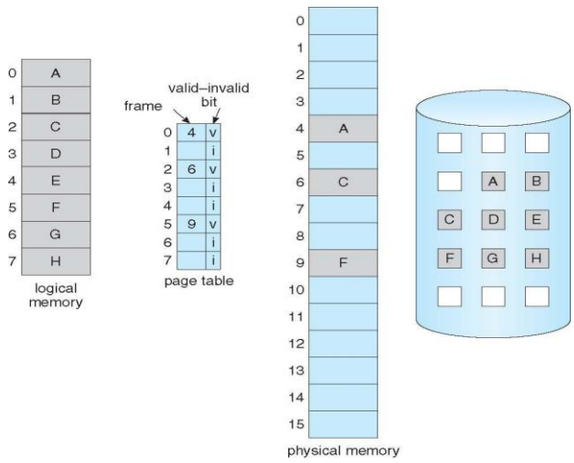


Transfer of a paged memory to contiguous disk space.

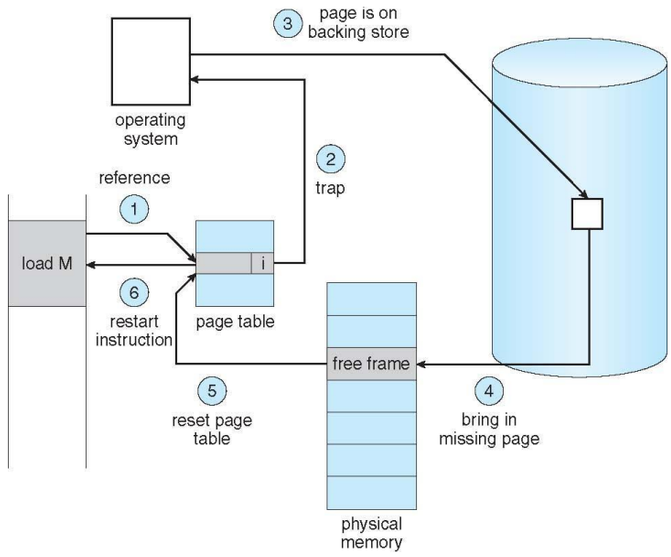
- A demand-paging system is similar to a paging system with swapping.
- But here it uses a lazy swapper; where it never swaps a page into memory unless that page will be needed.
- In Demand paging, **pager i**s more suitable than swapper .

Basic Concepts

- With swapping, pager guesses which pages will be used before swapping out again
- Instead, pager brings in only those pages into memory
- some form of hardware support is needed to distinguish between the pages that are in memory and the pages that are on the disk.
- The valid -invalid bit scheme can be used for this purpose.
- With each page table entry a valid–invalid bit is associated
  (v ⇒ in-memory – memory resident, i ⇒ not-in-memory)
- Initially valid–invalid bit is set to i on all entries
- During MMU address translation, if valid–invalid bit in page table entry is i then a page fault occurs.



Page table when some pages are not in main memory



Steps in handling a page fault.

- If there is a reference to a page, first reference to that page will trap to operating system:
  - page fault
- The procedure for handling this page fault is
  1. 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, terminate the process. If it was valid, but have not yet brought in that page, page it in.
  3. Find a free frame.
  4. Schedule a disk operation to read the desired page into the newly allocated frame.
  5. When the disk read is complete, modify the internal table kept with the process and the page table to indicate that the page is now in memory.
  6. Restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.
- In the extreme case, we can start executing a process with no pages in memory. When the operating system sets the instruction pointer to the first 404 Chapter 9 Virtual Memory instruction of the process, which is on a non-memory-resident page, the process immediately faults for the page. After this page is brought into memory, the process continues to execute, faulting as necessary until every page that it needs is in memory. At that point, it can execute with no more faults. This scheme is **pure demand paging**: never bring a page into memory until it is required
- Theoretically, some programs could access several new pages of memory with each instruction execution, possibly causing multiple page faults per instruction. This situation would result in unacceptable system performance. Locality of reference can be used for such situations.
- Hardware support needed for demand paging
  • **Page table**. This table has the ability to mark an entry invalid through a valid–invalid bit or a special value of protection bits.
  • **Secondary memory**. This memory holds those pages that are not present in main memory. The secondary memory is usually a high-speed disk. It is known as the swap device, and the section of disk used for this purpose is known as swap space

Performance of Demand Paging

- Demand paging can significantly affect the performance of a computer system.
- the effective access time is

Effective access time= (1 - p) x ma + p x page fault time.

- where,
  - p be the probability of a page fault (0 <=p <=1).
  - ma be the memory-access time

Page Replacement

- It prevents **over-allocation** of memory by modifying page-fault service routine to include page replacement

Basic Page Replacement

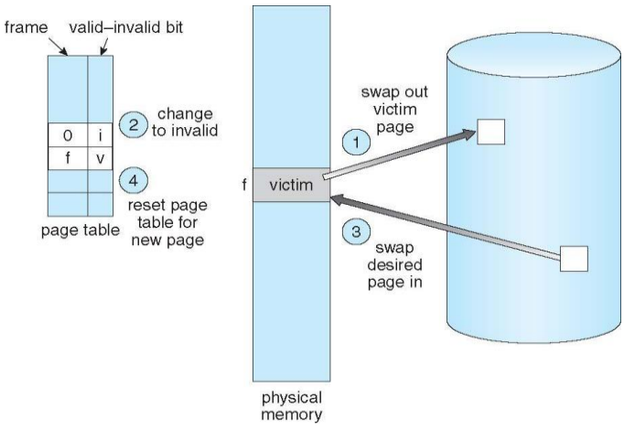
- Page replacement takes the following approach.

1. Find the location of the desired page on the disk.
2. Find a free frame:

   a. If there is a free frame, use it.

   b. If there is no free frame, use a page-replacement algorithm to select a victim frame

   c. Write the victim frame to the disk; change the page and frame tables accordingly.
3. Read the desired page into the newly freed frame; change the page and frame tables.
4. Restart the user process.

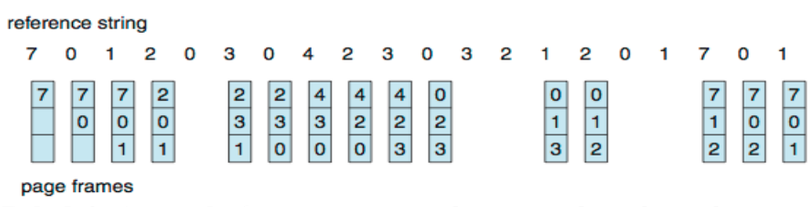


page replacement

- Use **modify** (**dirty**) **bit** to reduce overhead of page transfers – only modified pages are written to disk
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.
- two major problems to implement demand paging
  - Frame allocation algorithm
  - page replacement algorithm
- **Frame-allocation algorithm** determines
  - How many frames to give each process
  - Which frames to replace
- **Page-replacement algorithm**
  - Want lowest page-fault rate on both first access and re-access
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
  - String is just page numbers, not full addresses
  - Repeated access to the same page does not cause a page fault
  - Results depend on number of frames available

First-In-First-Out (FIFO) Algorithm

- A FIFO replacement algorithm associates with each page the time when that page was brought into memory.
- When a page must be replaced, the oldest page is chosen.
- Example:
  - Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
  - 3 frames (3 pages can be in memory at a time per process)



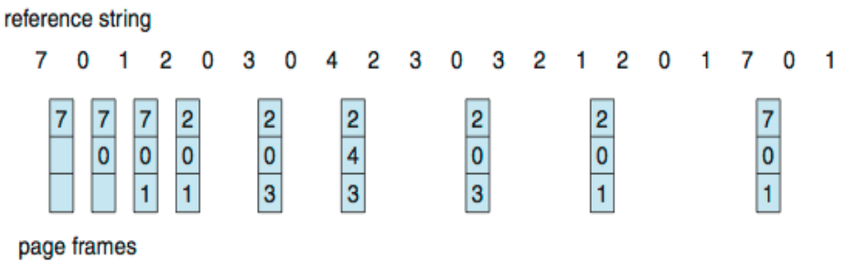
- **Belady's Anomaly**-for some page-replacement algorithms, the page-fault rate may increase as the number of allocated frames increases.

- Example:For the reference string 1,2,3,4,1,2,5,1,2,3,4,5

  page fault is 9 with frames 3 and page fault is 10 with frames 4.

Optimal Page Replacement

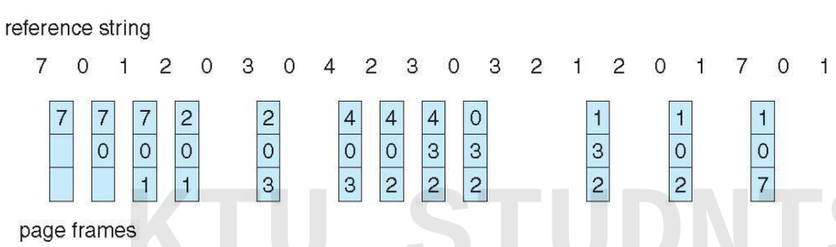
- Replace page that will not be used for longest period of time
- **It has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly-hence known as OPT or MIN.**



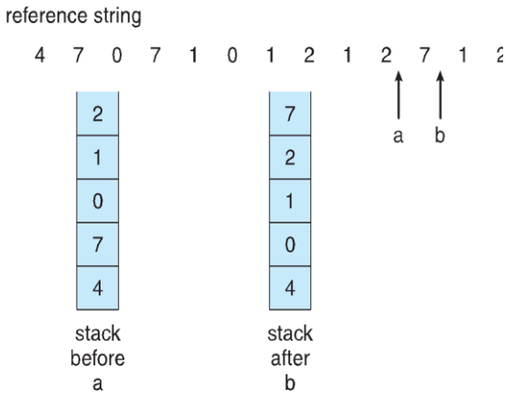
- It is difficult to implement, because it requires future knowledge of the reference string.

LRU Page Replacement

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page



- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used
- Two methods of implementations:
  - Counter implementation
    - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
      - When a page needs to be changed, look at the counters to find smallest value
        - Search through table needed
  - Stack implementation
    - Keep a stack of page numbers in a double link form:
    - Page referenced:
      - move it to the top
      - requires 6 pointers to be changed
    - But each update more expensive
    - No search for replacement
  - LRU and OPT are cases of **stack algorithms** that don't have Belady's Anomaly



Use of a stack to record the most recent page references.

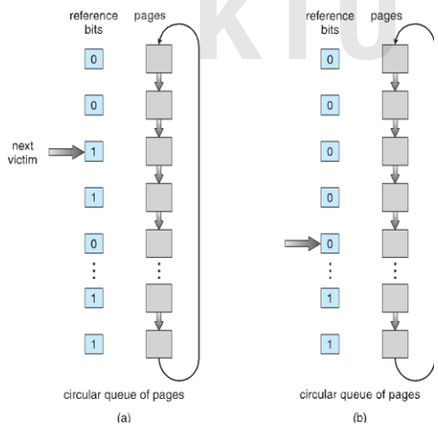
LRU-Approximation Page Replacement

Additional-Reference-Bits Algorithm

- With each page associate a bit, initially = 0
- When page is referenced bit set to 1
- Replace any with reference bit = 0 (if one exists)

Second-Chance Algorithm

- The basic algorithm of second-chance replacement is a FIFO replacement algorithm. When a page has been selected, inspect its reference bit.
- If the value is 0, proceed to replace this page; but if the reference bit is set to 1, give the page a second chance and move on to select the next FIFO page.
- When a page gets a second chance, its reference bit is cleared, and its arrival time is reset to the current time. Thus, a page that is given a second chance will not be replaced until all other pages have been replaced.
- One way to implement the second-chance algorithm (the clock algorithm) is as a circular queue. A pointer indicates which page is to be replaced next.
- When a frame is needed, the pointer advances until it finds a page with a 0 reference bit



Second-chance (clock) page-replacement algorithm

Enhanced Second-Chance Algorithm

- Improve algorithm by using reference bit and modify bit (if available) in concert
- Take ordered pair (reference, modify)

1. (0, 0) neither recently used not modified – best page to replace
2. (0, 1) not recently used but modified – not quite as good, must write out before replacement
3. (1, 0) recently used but clean – probably will be used again soon
4. (1, 1) recently used and modified – probably will be used again soon and need to write out before replacement

- When page replacement called for, use the clock scheme but use the four classes replace page in lowest non-empty class
  - Might need to search circular queue several times