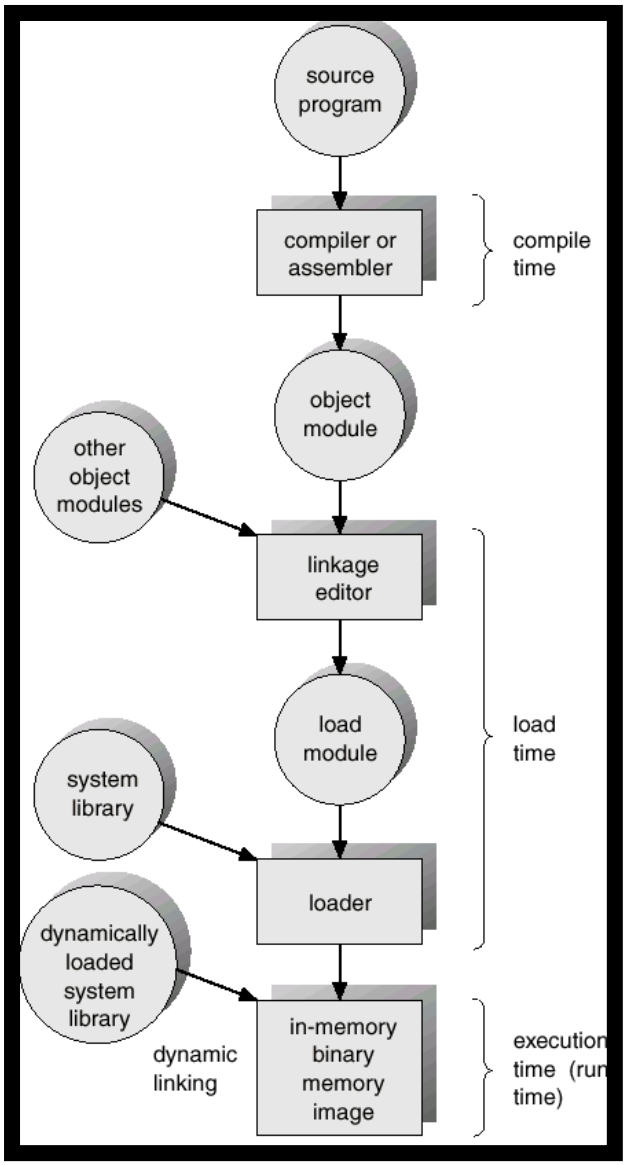
- Program must be brought into memory and placed within a process for it to be run.
- Input queue or job queue – collection of processes on the disk that are waiting to be brought into memory to run the program
- User programs go through several steps before being run.

Multistep process of a User program



Logical vs. Physical Address Space

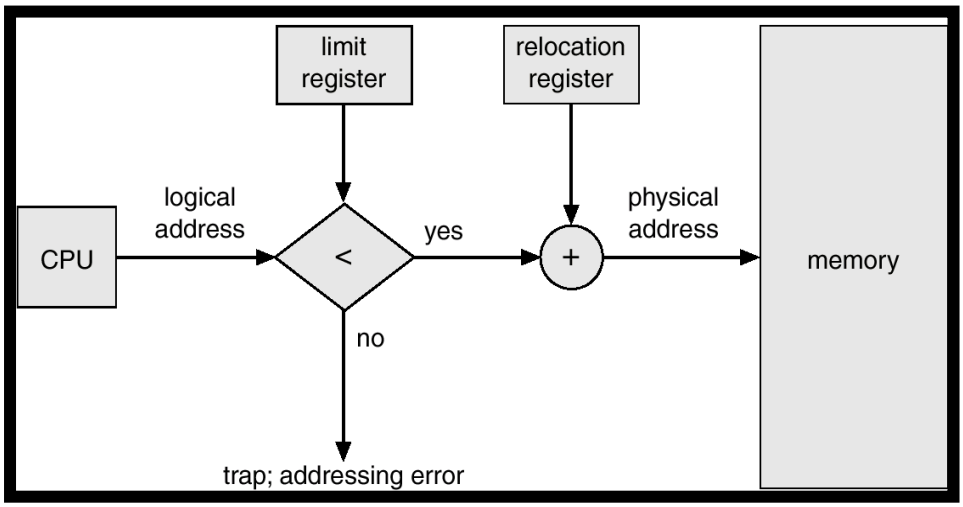
- ✓ The concept of a logical address space that is bound to a separate physical address space is central to proper memory management
- ✓ Logical address – generated by the CPU; also referred to as virtual address
- ✓ Physical address – address seen by the memory unit
- ✓ Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme

Memory-Management Unit (MMU)

- ✓ Hardware device that maps virtual to physical address

- ✓ In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
- ✓ The user program deals with logical addresses; it never sees the real physical addresses

Dynamic relocation using relocation register



Limit register contains range of logical addresses – each logical address must be less than the limit register.

Memory Management is the process of controlling and coordinating computer memory, assigning portions known as blocks to various running programs to optimize the overall performance of the system.

It is the functionality of an OS which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free

Reasons for using memory management:

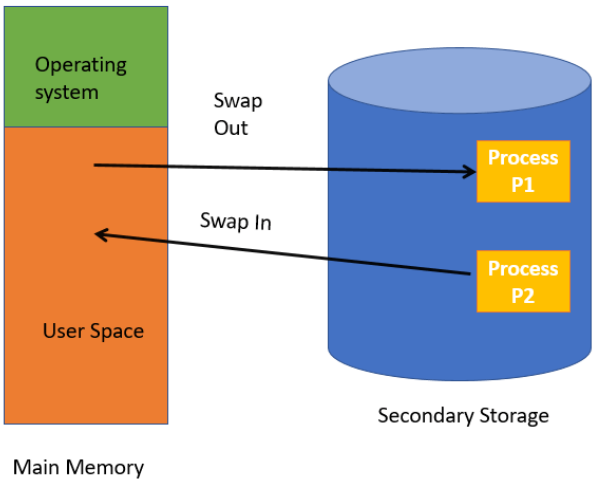
- It allows checking how much memory needs to be allocated to processes.
- Tracks whenever inventory gets freed or unallocated. According to it will update the status.
- It allocates the space to application routines.
- It also makes sure that these applications do not interfere with each other.
- Helps protect different processes from each other
- It places the programs in memory so that memory is utilized to its full extent.

Memory Management Strategies

SWAPPING

Swapping is a method in which the process should be swapped temporarily from the main memory to the backing store. It will be later brought back into the memory for continue execution.

Backing store is a hard disk or some other secondary storage device that should be big enough in order to accommodate copies of all memory images for all users. It is also capable of offering direct access to these memory images.



A variant of swapping policy is used for priority based scheduling algorithms. If a higher priority process arrives and wants service, the memory manager can swap out the lower-priority process and then load and execute the higher-priority process. When it finishes, the low priority task can be swapped back in and continued. This is sometimes called **roll out, roll in.**

Benefits of Swapping

- It offers a higher degree of multiprogramming.
- Allows dynamic relocation. For example, if address binding at execution time is being used, then processes can be swap in different locations. Else in case of compile and load time bindings, processes should be moved to the same location.
- It helps to get better utilization of memory.
- Minimum wastage of CPU time on completion so it can easily be applied to a priority-based scheduling method to improve its performance.

If we want to swap a process, we must be sure that it is completely idle. Never swap a process with pending I/O, or execute I/O operations only into operating-system buffers.

CONTIGUOUS MEMORY ALLOCATION

The memory is usually divided into two partitions: one for the resident operating system, and one for the user processes. We may place the operating system in either low memory or high memory.

To allocate available memory to the processes that is in the input queue waiting to be brought into memory. With this approach each process is contained in a single contiguous section of memory.

(The OS is usually loaded low, because that is where the interrupt vectors are located, but on older systems part of the OS was loaded high to make more room in low memory (within the 640K barrier) for user processes.)

Memory Protection

- The system shown in Figure 8.6 below allows protection against user programs accessing areas that they should not, allows programs to be relocated to different memory starting addresses as needed, and allows the memory space devoted to the OS to grow or shrink dynamically as needs change.

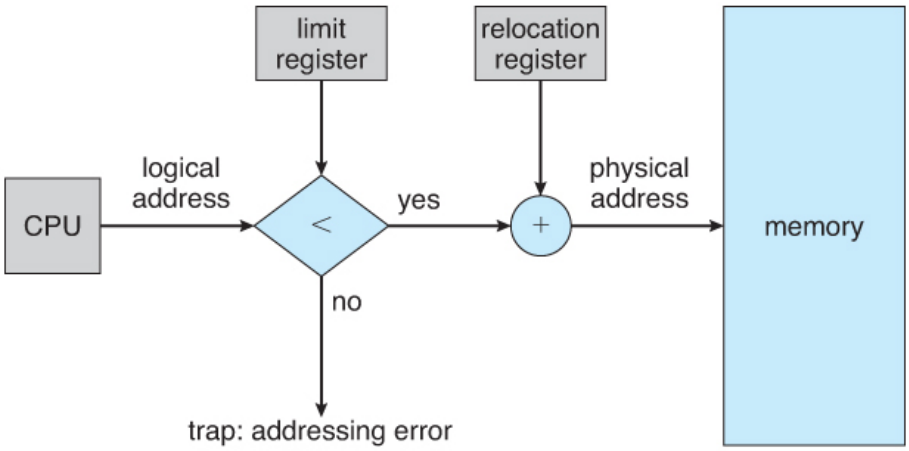


Figure 8.6 - Hardware support for relocation and limit registers

Memory Allocation

One of the simplest methods for memory allocation is to divide memory into several fixed-sized partitions. Each partition may contain exactly one process. In this multiple-partition 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. The operating system keeps a table indicating which parts of memory are available and which are occupied. Finally, when a process arrives and needs memory, a memory section large enough for this process is provided.

An alternate approach is to keep a list of unused (free) memory blocks (holes), and to find a hole of a suitable size whenever a process needs to be loaded into memory. There are many different strategies for finding the "best" allocation of memory to processes, including the three most commonly discussed:

1. **First fit** - Search the list of holes until one is found that is big enough to satisfy the request, and assign a portion of that hole to that process. Whatever fraction of the hole not needed by the request is left on the free list as a smaller hole. Subsequent requests may start looking either from the beginning of the list or from the point at which this search ended.
2. **Best fit** - Allocate the *smallest* hole that is big enough to satisfy the request. This saves large holes for other process requests that may need them later, but the resulting unused portions of holes may be too small to be of any use, and will therefore be wasted. Keeping the free list sorted can speed up the process of finding the right hole.
3. **Worst fit** - Allocate the largest hole available, thereby increasing the likelihood that the remaining portion will be usable for satisfying future requests.

Simulations show that either first or best fit are better than worst fit in terms of both time and storage utilization. First and best fits are about equal in terms of storage utilization, but first fit is faster.

Camlin | Page
Date / /

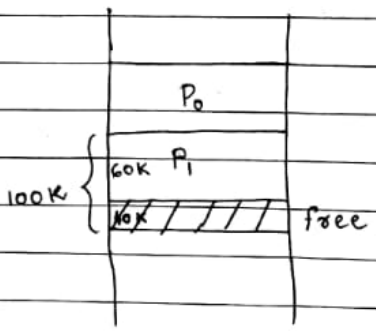
FRAGMENTATION.

As processes are loaded and removed from memory, the free memory space is broken into small pieces/partitions. It happens that often sometimes processes cannot be allocated to partitions because of their small size and memory partition remains unused. This problem is known as fragmentation.

Two types

① Internal fragmentation

When the memory allocated to the process is slightly larger than the memory size of process, this creates free space, left unused in the allocated block/partition, as it cannot be used by another process, causing internal fragmentation.

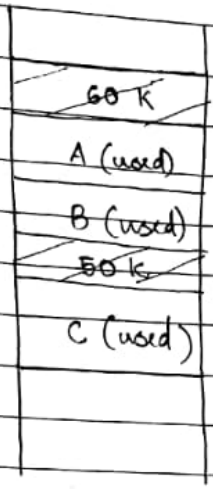


It occurs in fixed partitioning allocation method.

② External fragmentation

If total memory is scattered into small pieces (non-contiguous) and a request for memory is not satisfied although the sum of available memory in fragments is greater than the request amount of memory of process, then external fragmentation occurs.

It occurs in dynamic / variable size partition

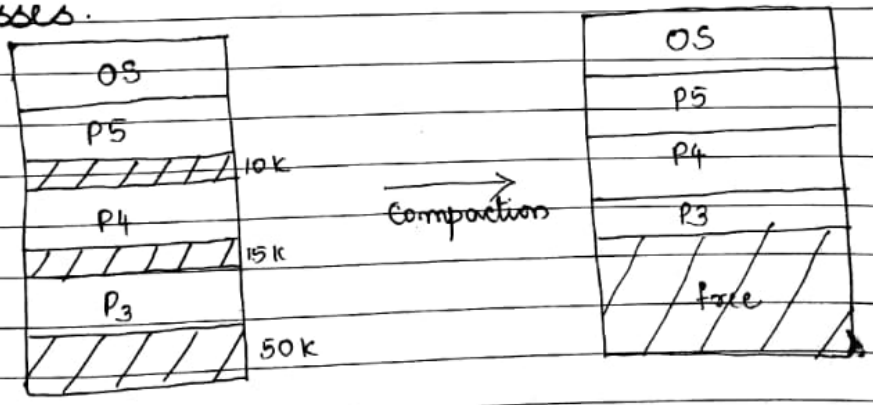


Total free space = 60+50 = 110

D (100 K) remains unallocated because of lack of contiguous space.

Solution : COMPACTION.

It is the process in which the free space is collected in a large memory block / partition to make space available for other processes.



Compaction involves shifting of memory from one place to another. This is done by I/O operations and hence reduce the CPU utilization and efficiency of the system.

PAGING

Paging is a memory management scheme that permits the physical address space of a process to be non-contiguous.

The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called frames and breaking 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 the backing store.

Frame table: It has one entry for each physical page frame, indicating whether the frame is free or allocated. If it is allocated, to which page of which processes.

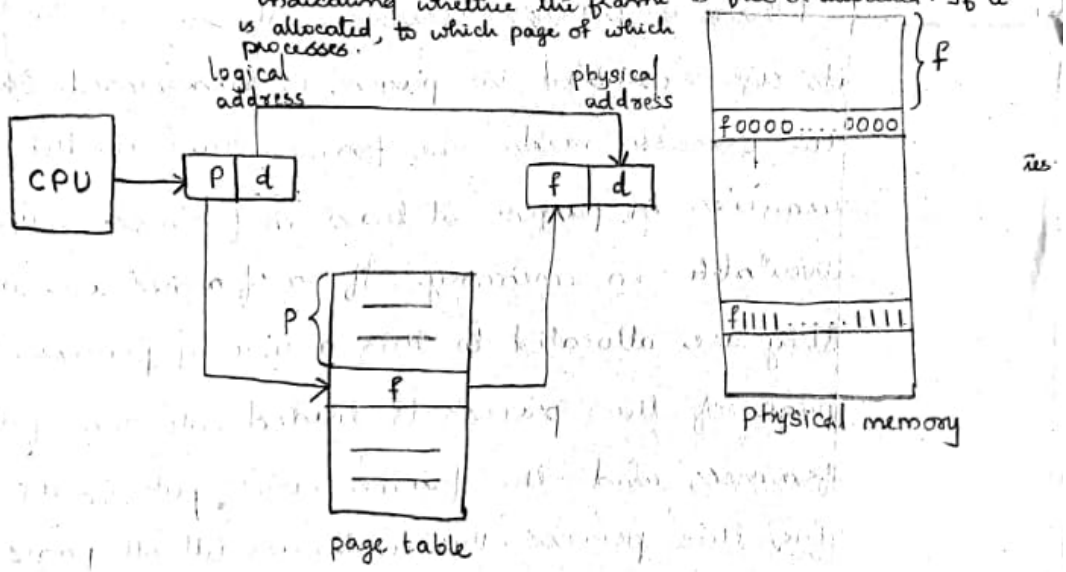
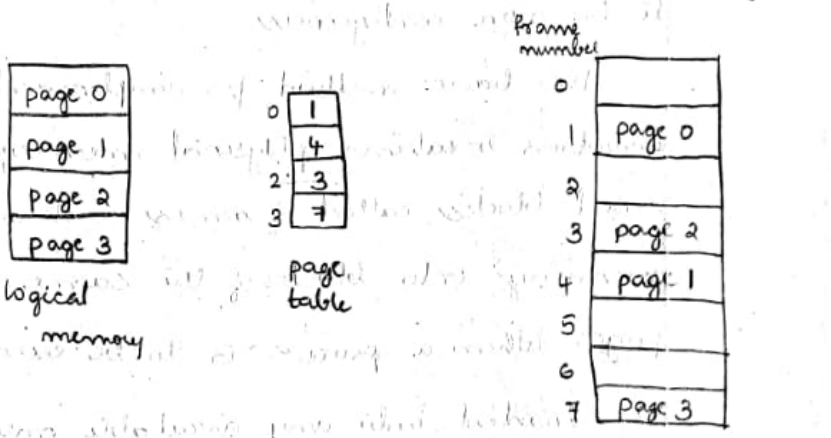


Fig: Paging Hardware

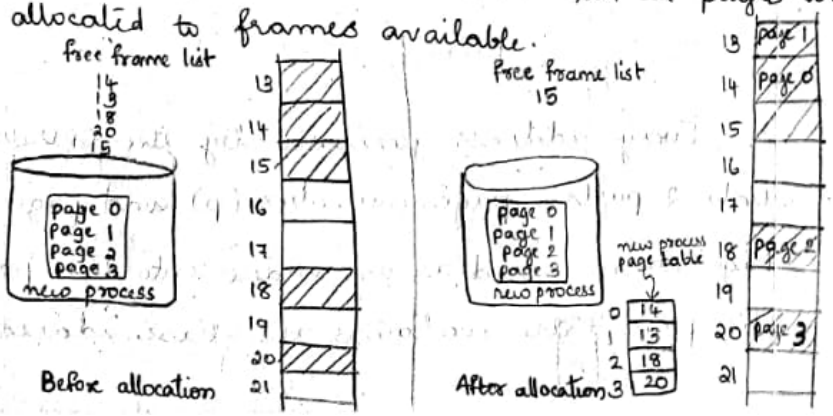
Every address generated by the CPU is divided into 2 parts: page number (p) and page offset (d). Page no. is used as an index into the page table. The page table contains the base address of each

(2) 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

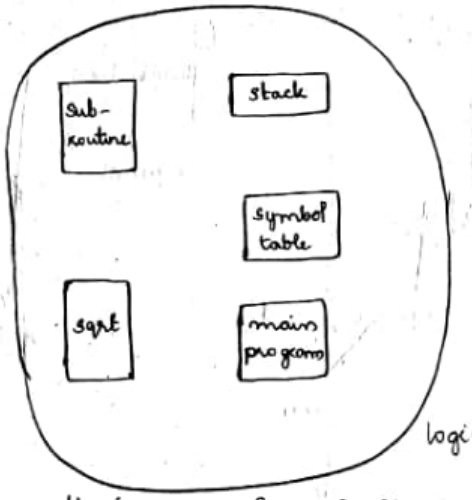


Paging model of logical & physical memory.

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 no is put in the page table for this process. This continues till all pages are allocated to frames available.



SEGMENTATION



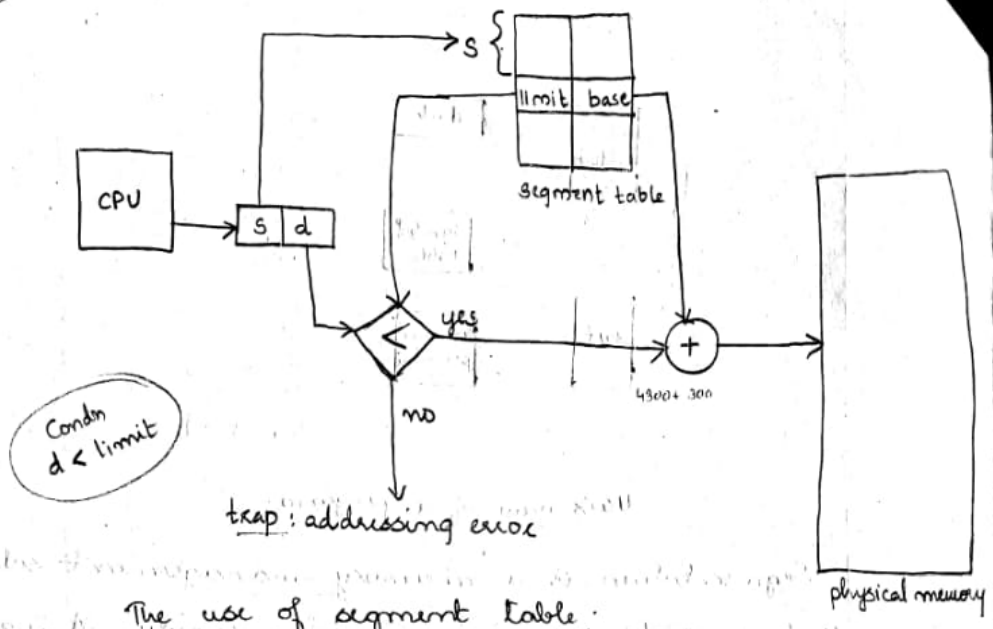
User's view of a program.

Segmentation is a memory management scheme that supports the 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. The user specifies each address by 2 quantities segment name and offset.

A logical address consists of a 2 tuples.

<segment_no, offset>

Normally, the user program is compiled, and the compiler automatically constructs segments reflecting the input program. The C compiler might create separate segments for code, global variables, heap, stack, standard C library. libraries that are linked in during compile time might be assigned separate segments. The loader would take all these segments and assign them segment numbers.

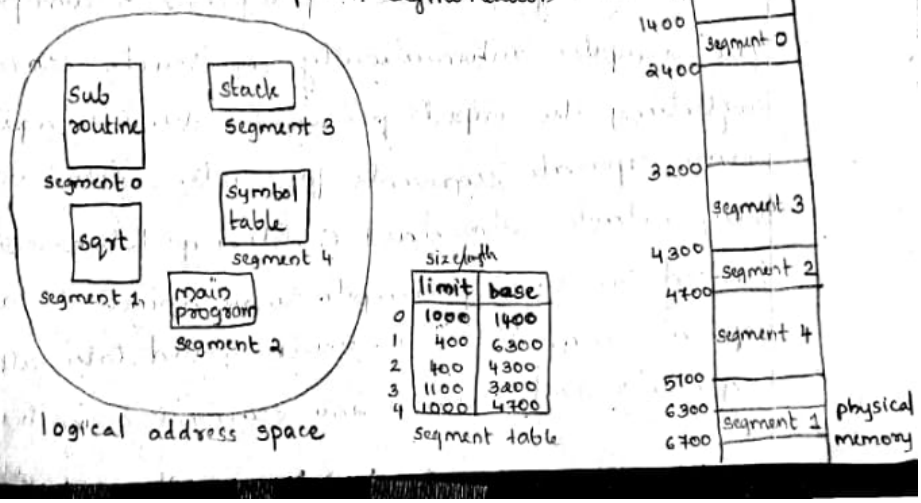


The use of segment table.

The mapping of logical address to physical add. use a segment table. A logical address consists of 2 parts
- segment no (s) – index to segment table
- offset (d) – between 0 and segment limit
  If not, we trap to OS (error)

When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte.

Example of segmentation:



Camlin Page
Date / /

PAGE REPLACEMENT

It is the technique used by the OS to decide which memory pages to swap out. It is also decided that in memory how many frames to allocate to each process. When page replacement is required, we must select frames that are to be replaced.

Reference string:

String of memory references is called reference string.

eg: (7, 0, 1, 2)

Page fault

It is a type of interrupt raised when a running process access a memory page that is mapped into virtual memory but not loaded in main memory.

DIAGRAM OF PAGE REPLACEMENT.

Steps in Page Replacement:

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.

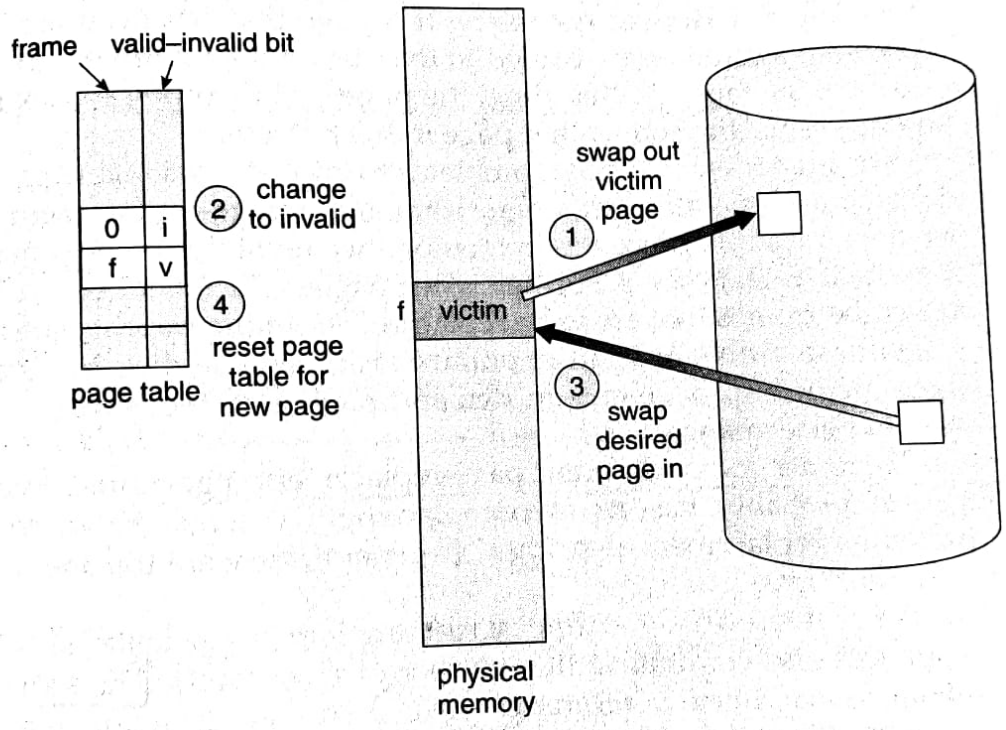
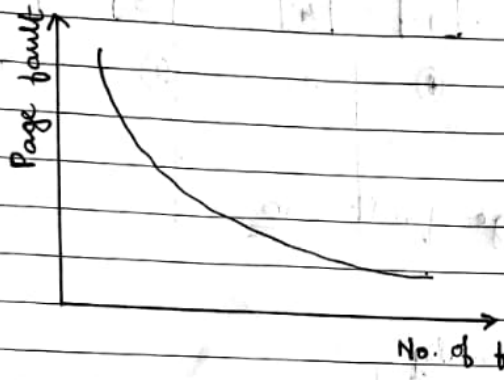


Figure 9.10 Page replacement.

3. Read the desired page into the newly freed frame, change the page and frame tables.
4. Restart the user process.

Graph of Page fault vs No. of frames



PAGE REPLACEMENT ALGORITHM

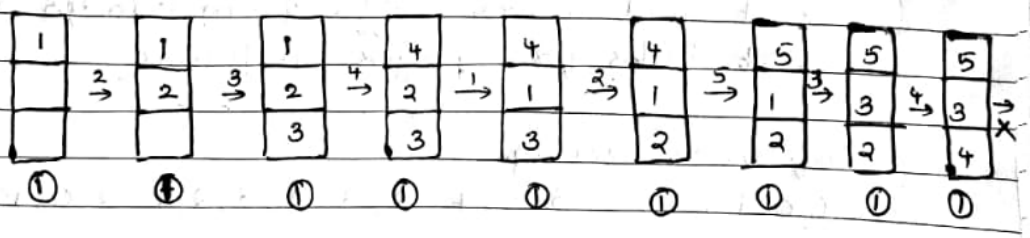
I. First In First Out (FIFO)

* Simple to implement.
* Oldest page is replaced.
* Performance is not always good.

Case 1 3 frames

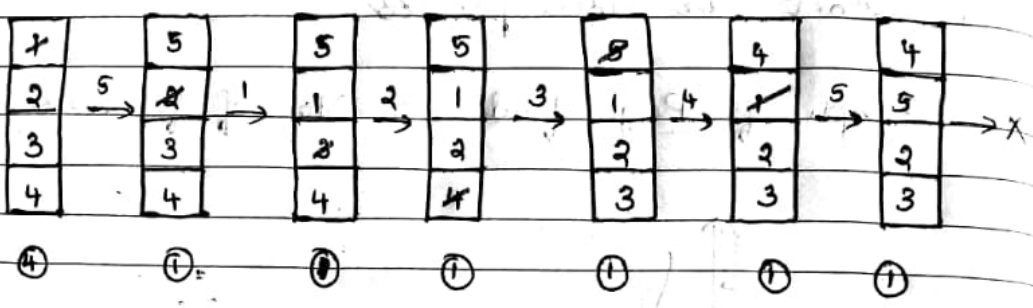
Reference string

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5



No. of page fault = 9

Case 2 4 frames



No. of page fault = 10

Belady's Anomaly

It states that for some page replacement algorithms, the page fault rate may increase as the no. of allocated frames increases. It is commonly observed in FIFO page replacement algorithms.

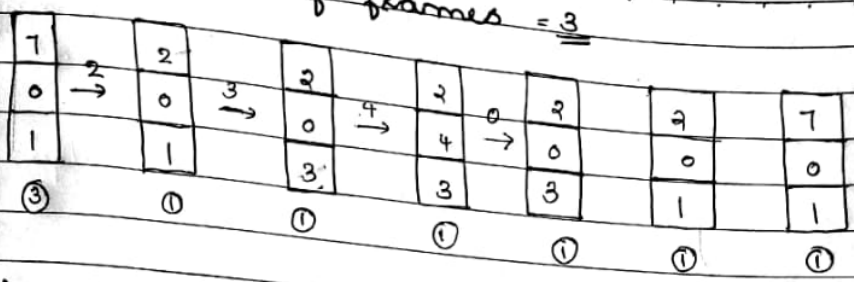
II. Optimal Page Replacement algorithm

* It is the best method becoz of lowest page fault.
* Replace the page that will not be used for the longest period of time.
* It will never suffer from Belady's anomaly
* It looks for pages in future.

Reference string

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Assume, no. of frames = 3



No. of page faults = 9

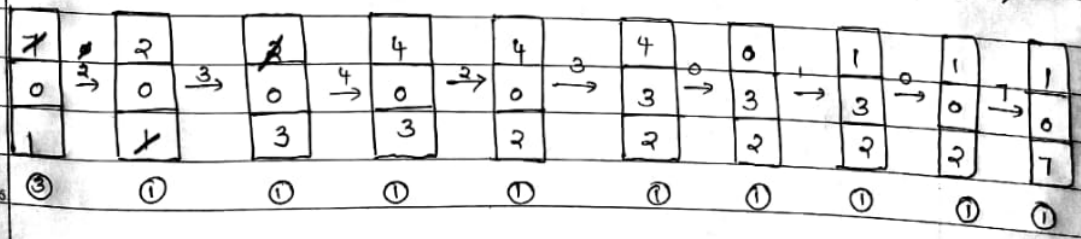
* It is difficult to implement (becoz it requires future knowledge of the reference string)

III. Least Recently Used (LRU) Page Replacement.

* LRU chooses the page that has not been used for the longest period of time.

Reference string

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



No. of page faults = 12

* It is like optimal page-replacement algorithm looking backwards in time.

(5)

FILE SYSTEM

FILE CONCEPT

A file system consists of two distinct parts:

- a collection of files - storing related data
- a directory structure - which organizes & provides information about all the files in the system.

A file is a named collection of related information that is recorded on secondary storage. Files are mapped by the OS onto physical devices. These storage devices are usually non-volatile, so the contents are persistent through power failures and system reboots.

A file has a certain defined structure, which depends on its type. A text file is a sequence of characters organized into lines. A source file is a sequence of subroutines and functions each of which is further organized as declarations followed by executable statements. An object file is a sequence of bytes organized into blocks understandable by the system's linker. An executable file is a series of code sections that the loader can bring into memory and execute.

File Attributes

- Name - symbolic name kept in human readable form.

- Identifier – a unique tag (number) which identifies the file within the file system
- Type – specifies type of file
- Location – a pointer to the device and to the location of the file on that device
- Size – size of file
- Protection – access control information
- Time, date and user identification – kept for creation, last modification and last use.

This data is useful for protection & security

File Operations

The operations that can be performed on files are:

① Creating a file – 2 steps

- find a space in the file system
- entry for the new file must be made in the directory.

② Writing a file

- Make a system call specifying both the name of the file and information to be written to the file

eg: write (a, "ABC")

③ Reading a file

- Use a system call that specifies the name of the file and where (in memory) the next block of the file should be put.

eg: read (a).

④ Repositioning within a file - File seek

- Does not involve any actual I/O.
- The directory is searched for the appropriate entry and the current-file-position pointer is repositioned to a given value.

⑤ Deleting a file

- Search directory for the named file, release all file space and erase directory entry.

⑥ Truncating a file

- To erase the contents of the file but to keep its attributes.

⑦ Appending - adding new information to the end of existing file.

⑧ Renaming an existing file.

File Types

File type is included as part of the filename. The name is split into two parts - a name and an extension usually separated by a period character.

eg: resume.doc

Filetype	extension	Function
executable	exe, com, bin	ready to run machine language programs.
object	obj, o	compiled, machine language, not linked
source code	c, java, asm, php	source code in various languages.

batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word processor formats
archive	arc, zip, tar	compressed formats
multimedia	mpeg, mov, mp3	binary file containing audio or A/V information.

FILE ACCESS METHODS

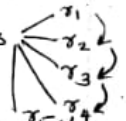
File store information. When it is used, this information must be accessed and read into computer memory.

3 METHODS:

(1) Sequential Access

→ Emulates magnetic tape operation

→ Information in the file is processed in order — one record after the other.



eg: editors and compilers access in this fashion.

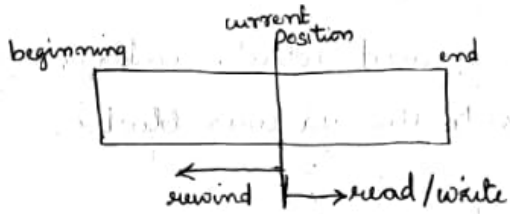
→ Supports following operations:

(i) read next – read record and moves pointer to next position.

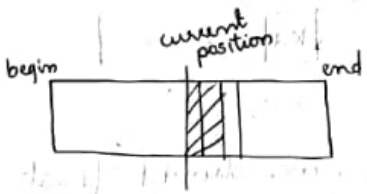
(ii) write next – write and advance position

appends

(iii) rewind - moving back to the earlier location



(iv) skip n records



- to skip 2 records from current location
- program may be able to skip forward and backward.

② Direct Access :

It is based on disk model. It allows random access. User can jump to any record and access that record. Following operations are supported:

(i) read n - Reading record no. 'n'

(ii) write n - Writing record no. 'n'.

(iii) jump to record n -

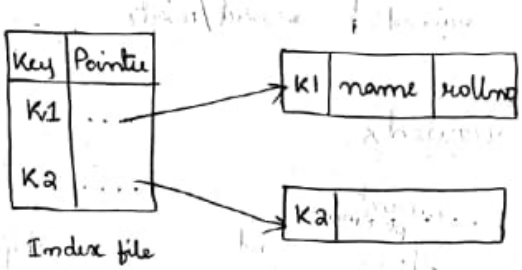
eg: jump to 10 from 5

Current position is at record 5 and to move to record 10, so records in between 5 and 10 are skipped. It can be 0 or end of file.

(iv) Query current record - used to return back to this record later.

③ Indexed Access

Index is created which contains a key field and pointers to the various blocks.



→ To access a file, we should first access index file for the key field and from the pointer, access the memory location where the file is stored.

FILE ALLOCATION METHODS.

It is used to allocate space to the files so that disk space is utilized in an efficient manner.

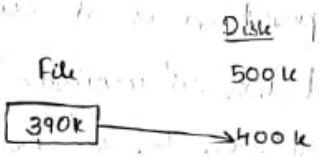
★ Factors to consider:

↳ Processing speed of — sequential access / random access.

↳ Ability to use multisector and multi track transfer.

↳ Disk space utilization - choose the disk space with less wastage

eg:



↳ Main memory requirement - File allocation should be such that main memory requirement is least.

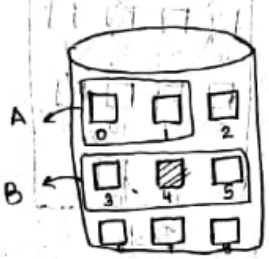
Contiguous Allocation

→ Each file occupies a set of contiguous addresses on disk.

→ Linear ordering

→ Location of a file is defined by the disk address of the first block and its length.

→ Both sequential and direct / random access are supported.



Disk.

File	start	len
A	0	2
B	3	3

• For random access, (b+i)

eg: 2nd record of B

3 + 1 = 4 direct access

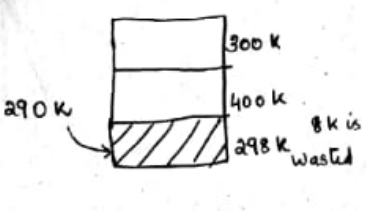
• For seq access, FS remembers the last block disk address and reads the next block.

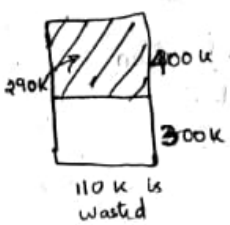
Disadvantages

* finding space for new file - need to keep track the vacant space.
* external fragmentation

Contiguous Allocation Application : Dynamic Storage allocation

• Best fit • First fit • Worst fit.





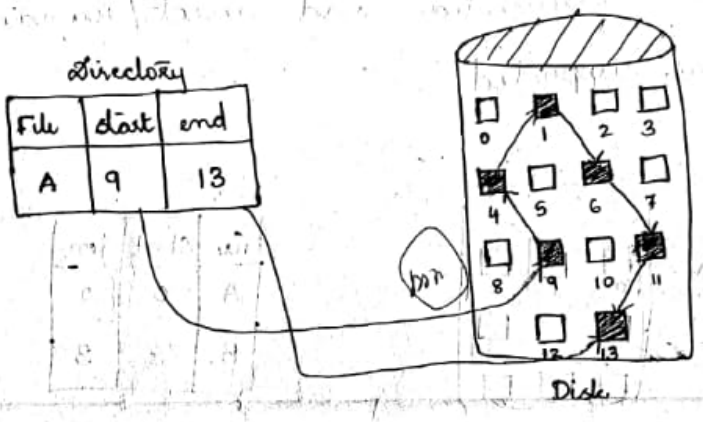
② Linked Allocation

→ Solves all problems of contiguous allocation. Each file is a linked list of disk blocks.

→ No external fragmentation

Disadv

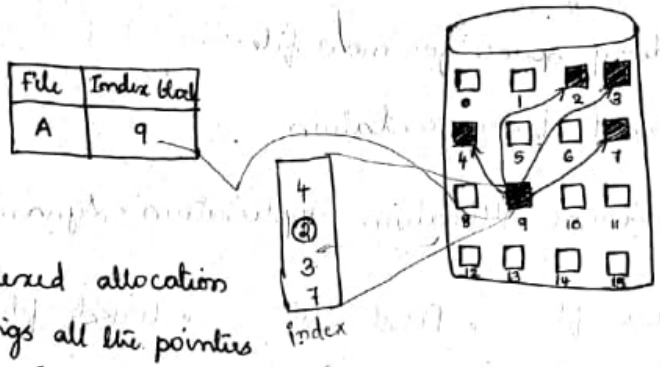
* can be used only for sequential access file. ie, To access 6, 9 should start from 9 through till 6.



③ Indexed allocation – allows random access

→ Solves the problems of linked allocation (No random access)

→ In this all the pointers are brought together into one location called index block. Each file has its own index block.



• Indexed allocation brings all the pointers together into one location ie, Index block.