Operating System Concepts

(22MCA12)

PART I

1. **Differentiate between internal and external fragmentation.**

OR

2. **Write notes on i) Swapping ii) Compaction.**

PART II

3. **What is Paging? Explain the paging hardware with a neat diagram.**

OR

4. **What is the need for segmentation? Explain the segmentation architecture with a neat diagram.**

PART III

5. **What is demand paging? Explain with a neat diagram.**

OR

6. **Consider the following reference string:**

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

How many page faults would occur in case of i)FIFO ii)Optimal algorithm?

PART IV

7. **What are the various access methods used to access files?**

OR

8. **Explain the different directory structures with neat diagrams.**

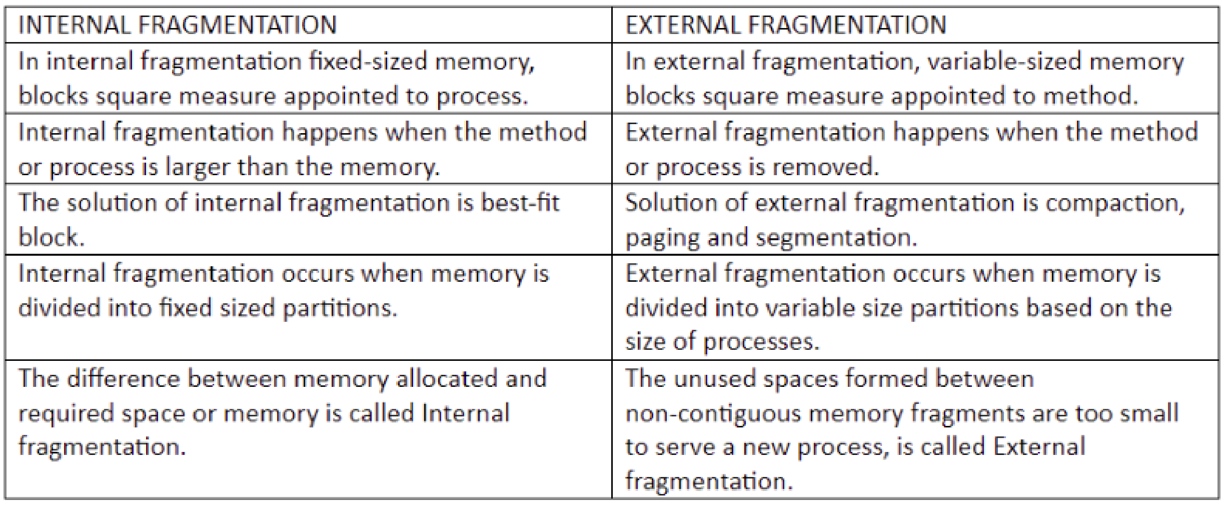
PARTV

9. **What is a page fault? Explain the steps involved in handling page faults.**

OR

10. **Explain the file allocation methods with neat diagrams.**

1.



INTERNAL FRAGMENTATION	EXTERNAL FRAGMENTATION
In internal fragmentation fixed-sized memory, blocks square measure appointed to process.	In external fragmentation, variable-sized memory blocks square measure appointed to method.
Internal fragmentation happens when the method or process is larger than the memory.	External fragmentation happens when the method or process is removed.
The solution of internal fragmentation is best-fit block.	Solution of external fragmentation is compaction, paging and segmentation.
Internal fragmentation occurs when memory is divided into fixed sized partitions.	External fragmentation occurs when memory is divided into variable size partitions based on the size of processes.
The difference between memory allocated and required space or memory is called Internal fragmentation.	The unused spaces formed between non-contiguous memory fragments are too small to serve a new process, is called External fragmentation.

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 into memory for continued execution.
- **Backing-store** is a fast disk which is large enough to accommodate copies of allmemory-images for all users.

- **Roll out/Roll in** is a swapping variant used for priority-based scheduling algorithms.
  - □ Lower-priority process is swapped out so that higher-priority process can beloaded and executed.
  - □ Once the higher-priority process finishes, the lower-priority process can beswapped back in and continued (Figure 3.12).

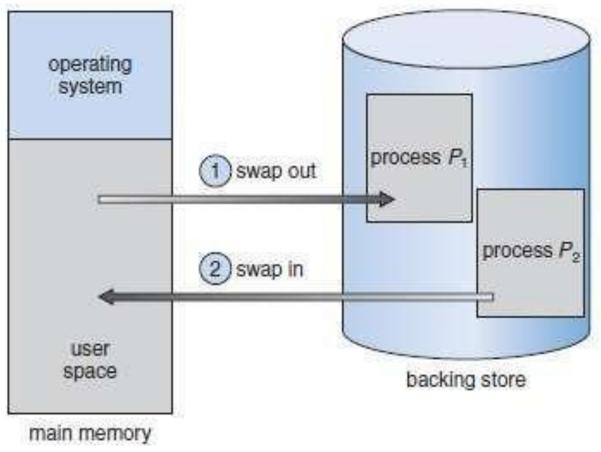


Figure 3.12 Swapping of two processes using a disk as a backing-store

- Swapping depends upon address-binding:
  1) If binding is done at load-time, then process cannot be easily moved to a different location.
  2) If binding is done at execution-time, then a process can be swapped into adifferent memory- space, because the physical-addresses are computed during execution-time.
- Major part of swap-time is transfer-time; i.e. total transfer-time is directly proportional to the amount of memory swapped.
- Disadvantages:

1) Context-switch time is fairly high.
2) If we want to swap a process, we must be sure that it is completely idle. Twosolutions:
    i) Never swap a process with pending I/O.
    ii) Execute I/O operations only into OS buffers.

<u>Paging</u>

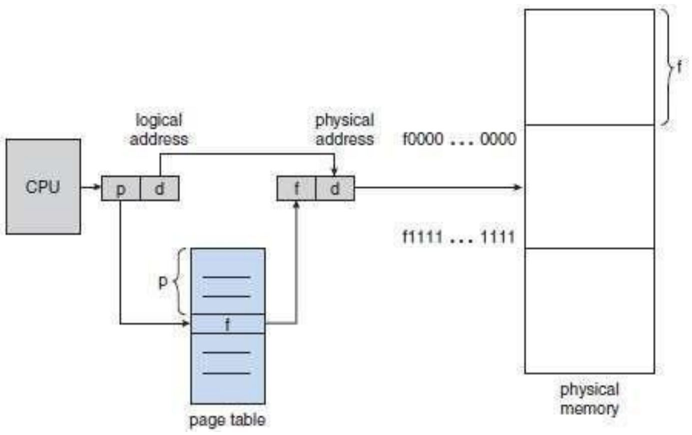
- Paging is a memory-management scheme.
- This permits the physical-address space of a process to be non-contiguous.
- This also solves the considerable problem of fitting memory-chunks of varying sizesonto the backing-store.
- Traditionally: Support for paging has been handled by hardware.Recent designs: The hardware & OS are closely integrated.

1) Basic Method

□ Physical-memory is broken into fixed-sized blocks called**frames** (Figure 3.16). Logical-memory is broken into same-sized blocks called **pages.**

□When a process is to be executed, its pages are loaded into any availablememory-frames from the backing-store.

□The backing-store is divided into fixed-sized blocks that are of the same size

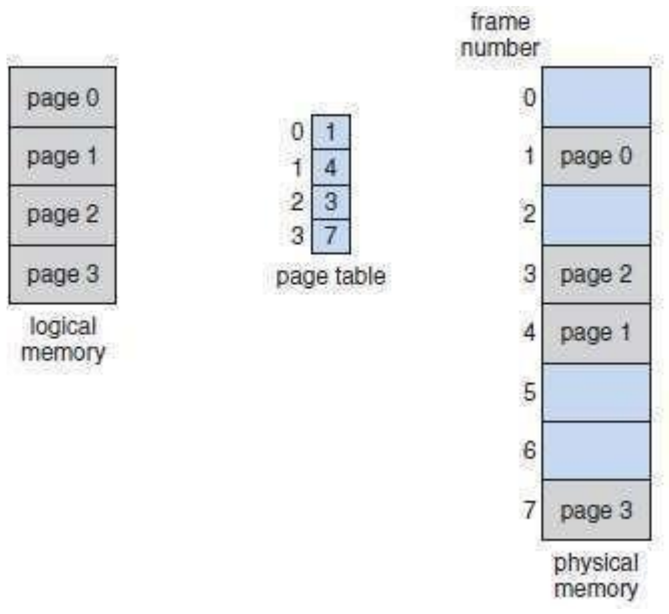


as the memory-frames.

Figure 3.16 Paging hardware

- The page-table contains the base-address of each page in physical-memory.
- Address generated by CPU is divided into 2 parts (Figure 3.17):
    1) **Page-number(p)** is used as an index to the page-table and

2) **Offset (d)** is combined with the base-address to definethe physical-address. This physical-address is sent



to the memory-unit.

Figure 3.17 Paging model of logical andphysical-memory

□The page-size (like the frame size) is defined by the hardware (Figure 3.18).

□If the size of the logical-address space is 2^m, and a page-size is 2^n addressing-units (bytes or words) then the high-order m-n bits of a logical-address designate thepage-number, and the n low-order bits designate the page-offset.

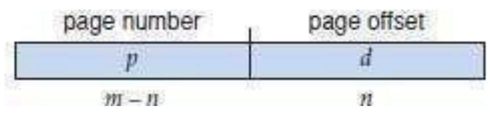
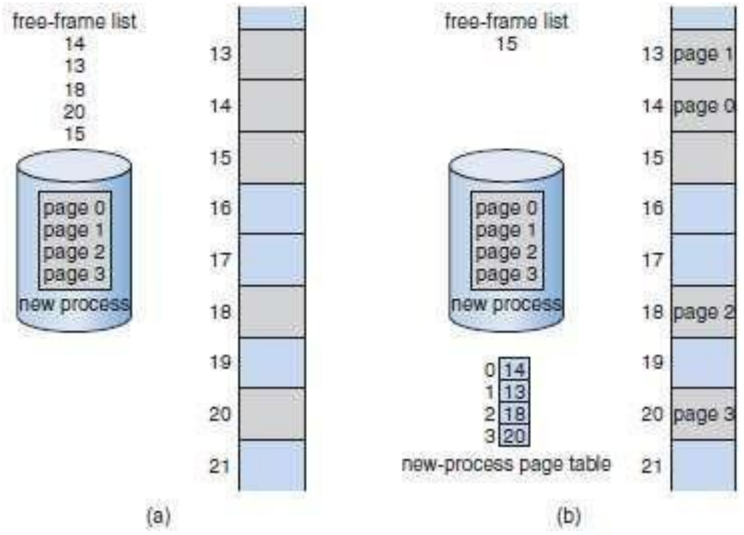


Figure 3.18 Free frames (a) before allocation and (b) after allocation



- Consider the following micro example, in which a process has 16 bytes of logical memory, mapped in 4 byte pages into 32 bytes of physical memory. (Presumably some other processes would be consuming the remaining 16 bytes of physical memory.)

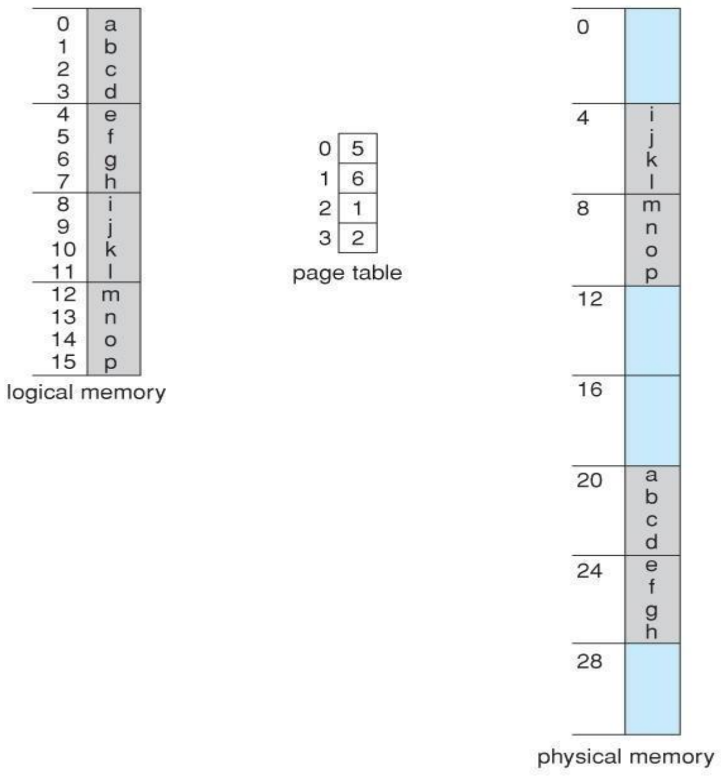


Figure 8.12 - Paging example for a 32-byte memory with 4-byte pages

- Note that paging is like having a table of relocation registers, one for each page of the logical memory.

- There is no external fragmentation with paging. All blocks of physical memory are used, and there are no gaps in between and no problems with finding the right sized hole for a particular chunk of memory.
- There is, however, internal fragmentation. Memory is allocated in chunks the size of a page, and on the average, the last page will only be half full, wasting on the average half a page of memory per process. (Possibly more, if processes keep their code and data in separate pages.)
- Larger page sizes waste more memory, but are more efficient in terms of overhead. Modern trends have been to increase page sizes, and some systems even have multiple size pages to try and make the best of both worlds.
- Page table entries (frame numbers) are typically 32 bit numbers, allowing access to 2^32 physical page frames. If those frames are 4 KB in size each, that translates to 16 TB of addressable physical memory. (32 + 12 = 44 bits of physical address space.)
- When a process requests memory (e.g. when its code is loaded in from disk), free frames are allocated from a free-frame list, and inserted into that process's page table.
- Processes are blocked from accessing anyone else's memory because all of their memory requests are mapped through their page table. There is no way for them to generate an address that maps into any other process's memory space.
- The operating system must keep track of each individual process's page table, updating it whenever the process's pages get moved in and out of memory, and applying the correct page table when processing system calls for a particular process. This all increases the overhead involved when swapping processes in and out of the CPU. (The currently active page table must be updated to reflect the process that is currently running.)

4

Segmentation

1) Basic Method

- This is a memory-management scheme that supports user-view of memory (Figure 3.26).
- A logical-address space is a collection of segments.
- Each segment has a name and a length.
- The addresses specify both
  - → segment-name and
  - → offset within the segment.
- Normally, the user-program is compiled, and the compiler automatically constructs segments reflecting the input program.

For ex:

→ The code
→ The heap, from which memory is allocated
→ The standard C library
→ Global variables
→ The stacks used by each thread

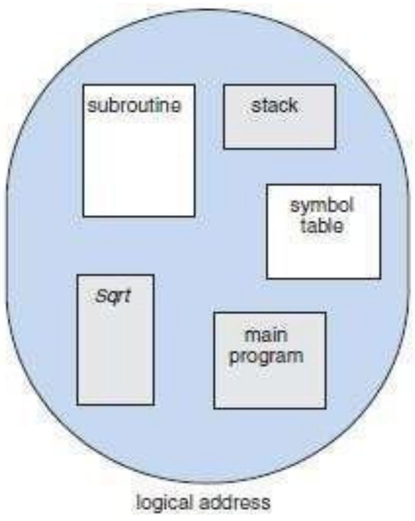
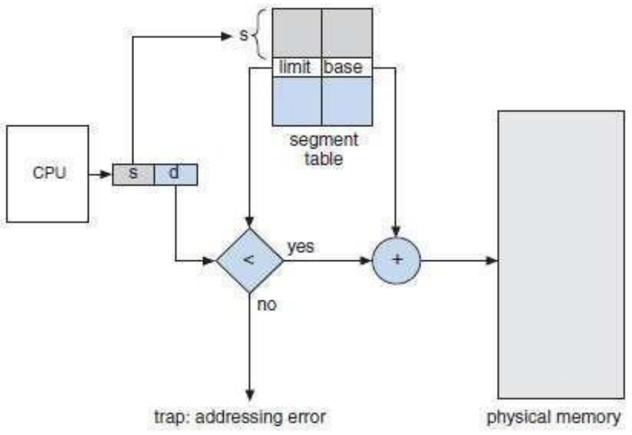


Figure 3.26 Programmer's view of a program

2) Hardware Support

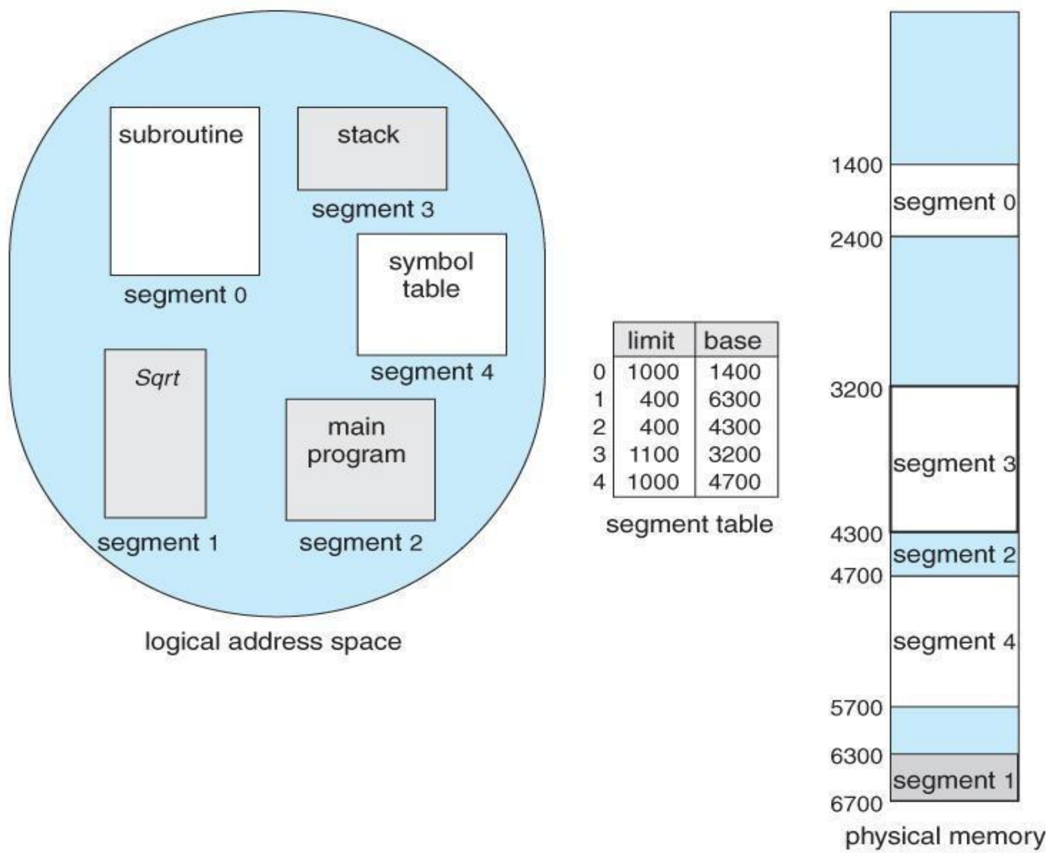
- Segment-table maps 2 dimensional user-defined addresses into one-dimensional physical-addresses.
- In the segment-table, each entry has following 2 fields:
    8) **Segment-base** contains starting physical-address where the segment resides in memory.
    9) **Segment-limit** specifies the length of the segment (Figure 3.27).
- A logical-address consists of 2 parts:
    1) **Segment-number(s)** is used as an index to the segment-table .
    2) **Offset(d)** must be between 0 and the segment-limit.
- If offset is not between 0 & segment-limit, then we trap to the OS(logical-addressing attempt beyond end of segment).
- If offset is legal, then it is added to the segment-base to produce the physical-memory



address.

Figure 3.27 Segmentation hardware

- Example :



	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

- We have 5 segments (0-4), which are stored in physical memory as shown. The segmenttable has a separate entry for each segment, giving the base address of the segment & its length i.e., limit.
- Example: Segment 2 is 400 bytes long and begins at location 4300. Thus the referenceto byte 53 of segment 2 is mapped to 4300+53 = 4353.

5

4.1 Demand Paging

- A demand-paging system is similar to a paging-system with swapping (Figure 4.2).
- Processes reside in secondary-memory (usually a disk).
- When we want to execute a process, we swap it into memory.
- Instead of swapping in a whole process, **lazy swapper** brings only those necessary pages into memory.

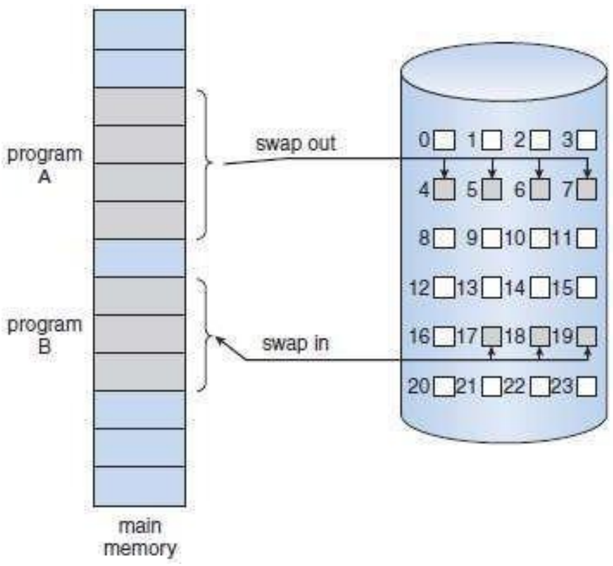
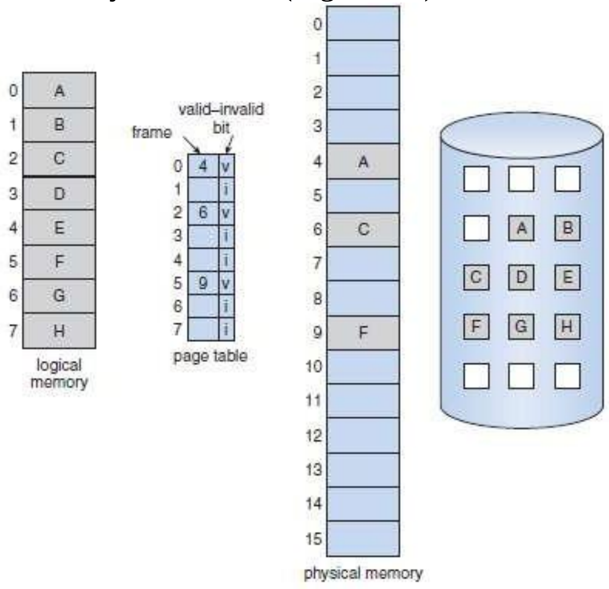


Figure 4.2 Transfer of a paged memory to contiguous disk space

4.2.1 Basic Concepts

- Instead of swapping in a whole process, the pager brings only those necessary pages into memory.
- Advantages:
  1) Avoids reading into memory-pages that will not be used,
  2) Decreases the swap-time and
  3) Decreases the amount of physical-memory needed.
- The valid-invalid bit scheme can be used to distinguish between
  → pages that are in memory and
  → pages that are on the disk.
  1) If the bit is set to **valid**, the associated page is both legal and in memory.
  2) If the bit is set to **invalid**, the page either
     → is not valid (i.e. not in the logical-address space of the process) or
     → is valid but is currently on the disk (Figure 4.3).



Solution:

(i) LRU with 3 frames:

Frames	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
1	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
2		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
3			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
No. of Page faults	√	√	√	√		√		√	√	√	√			√		√		√		

No of page faults=12

(ii) FIFO with 3 frames:

Frames	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
1	7	0	1	2	2	3	0	4	2	3	0	0	0	1	2	2	2	7	0	1
2		7	0	1	1	2	3	0	4	2	3	3	3	0	1	1	1	2	7	0
3			7	0	0	1	2	3	0	4	2	2	2	3	0	0	0	1	2	7
No. of Page faults	√	√	√	√		√	√	√	√	√	√			√	√			√	√	√

No of page faults=15

(iii) Optimal with 3 frames:

Frames	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	2	7	7	7
2		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
3			1	1	1	3	3	3	3	3	3	3	3	1	1	1	1	1	1	1
No. of Page faults	√	√	√	√		√		√			√			√				√		

No of page faults=9

Conclusion: The optimal page replacement algorithm is most efficient among three algorithms, as it has lowest page faults i.e. 9.

7

4.8 Access Methods

4.8.1 Sequential Access

- This is based on a tape model of a file.
- This works both on
  - → sequential-access devices and
  - → random-access devices.
- Info. in the file is processed in order (Figure 4.15).
  For ex: editors and compilers
- Reading and writing are the 2 main operations on the file.
- File-operations:
  1) **read next**
     - ➢ This is used to
       - → read the next portion of the file and
       - → advance a file-pointer, which tracks the I/O location.
  2) **write next**
     - ➢ This is used to
       - → append to the end of the file and
       - → advance to the new end of file.

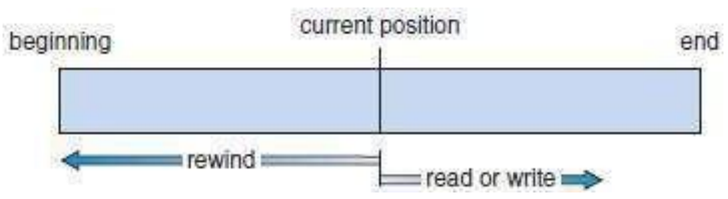


Figure 4.15 Sequential-access file

4.8.2 Direct Access (Relative Access)

- This is based on a disk model of a file (since disks allow random access to any file-block).
- A file is made up of fixed length logical records**.**
- Programs can read and write records rapidly in no particular order.
- Disadvantages:
  1) Useful for immediate access to large amounts of info.
  2) Databases are often of this type.
- File-operations include a relative block-number as parameter.
- The **relative block-number** is an index relative to the beginning of the file.
- File-operations (Figure 4.16):

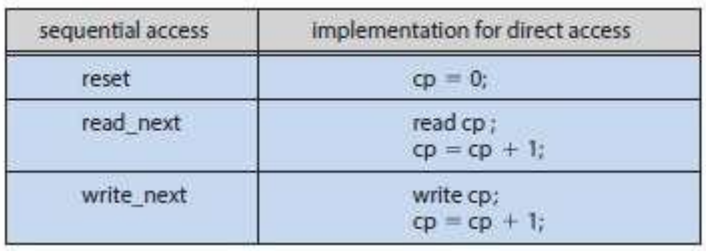
  1) read n

  2) write n

  where n is the block-number
- Use of relative block-numbers:

  → allows OS to decide where the file should be placed and

  → helps to prevent user from accessing portions of file-system that may not be part of his file.



sequential access	implementation for direct access
reset	cp = 0;
read_next	read cp ; cp = cp + 1;

write_next	write cp;
	cp = cp + 1;

Figure 4.16 Simulation of sequential access on a direct-access file

4.8.3 Other Access Methods

- These methods generally involve constructing a **file-index**.
- The index contains pointers to the various blocks (like an index in the back of a book).
- To find a record in the file(Figure 4.17):
    1) First, search the index and
    2) Then, use the pointer to
        → access the file directly and
        → find the desired record.
- Problem: With large files, the index-file itself may become too large to be kept in memory.

Solution: Create an index for the index-file. (The primary index-file may contain pointers to secondary index-files, which would point to the actual data-items).

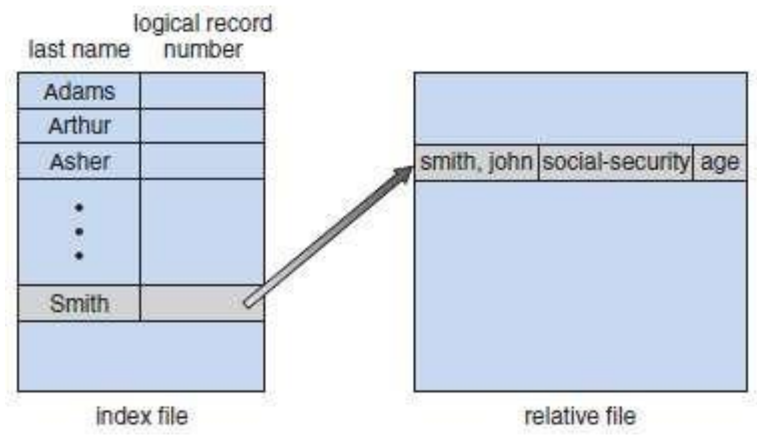


Figure 4.17 Example of index and relative files

4.9.1 Single Level Directory

- All files are contained in the same directory (Figure 4.19).
- Disadvantages (Limitations):
    1) Naming problem: All files must have unique names.
    2) Grouping problem: Difficult to remember names of all files, as number of files increases.

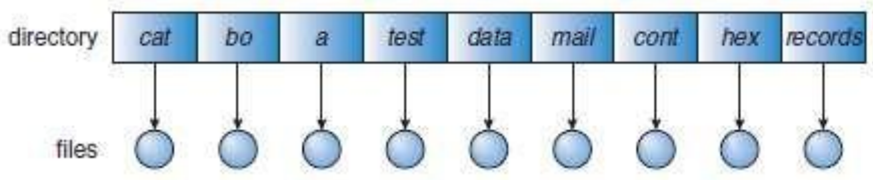


Figure 4.19 Single-level directory

4.9.2 Two Level Directory

- A separate directory for each user.
- Each user has his own UFD (user file directory).
- The UFDs have similar structures.
- Each UFD lists only the files of a single user.
- When a user job starts, the system's MFD is searched (MFD=master file directory).
- The MFD is indexed by user-name.
- Each entry in MFD points to the UFD for that user (Figure 4.20).

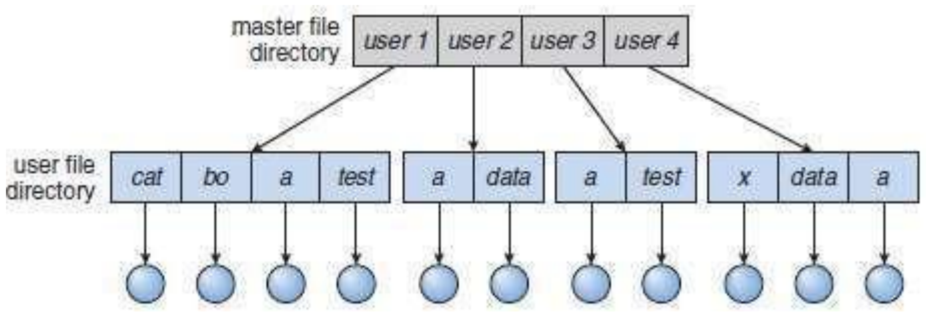


Figure 4.20 Two-level directory-structure

- To create a file for a user,
  the OS searches only that user's UFD to determine whether another file of that name exists.
- To delete a file,
  the OS limits its search to the local UFD. (Thus, it cannot accidentally delete another user's file that has the same name).
- Advantages:
  1) No filename-collision among different users.
  2) Efficient searching.
- Disadvantage:
  1) Users are isolated from one another and can‟t cooperate on the same task.

4.9.3 Tree Structured Directories

- Users can create their own subdirectories and organize files (Figure 4.21).
- A tree is the most common directory-structure.
- The tree has a root directory.
- Every file in the system has a unique path-name.

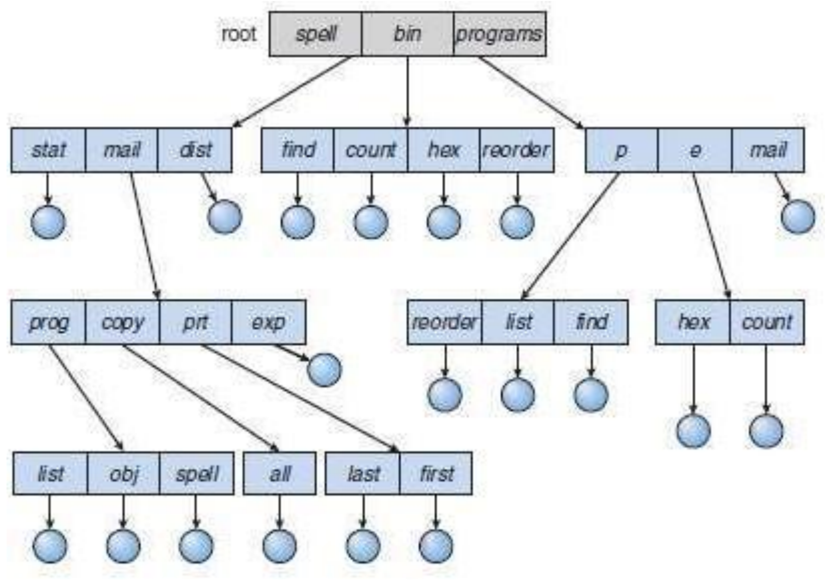


Figure 4.21 Tree-structured directory-structure

- A directory contains a set of files (or subdirectories).
- A directory is simply another file, but it is treated in a special way.
- In each directory-entry, one bit defines as
  file (0) or
  subdirectory (1).
- Path-names can be of 2 types:
- Two types of path-names:
  1) **Absolute path-name** begins at the root.
  2) **Relative path-name** defines a path from the current directory.
- How to delete directory?
  1) To delete an **empty directory**:
     → Just delete the directory.
  2) To delete a **non-empty directory**:
     → First, delete all files in the directory.
     → If any subdirectories exist, this procedure must be applied recursively to them.
- Advantage:
  1) Users can be allowed to access the files of other users.
- Disadvantages:
  1) A path to a file can be longer than a path in a two-level directory.
  2) Prohibits the sharing of files (or directories).

9

- A **page-fault** occurs when the process tries to access a page that was not brought into memory**.**
- Procedure for handling the page-fault (Figure 4.4):
  1) Check an internal-table to determine whether the reference was a valid or an invalid memory access.
  2) If the reference is invalid, we terminate the process.
     If reference is valid, but we have not yet brought in that page, we now page it in.
  3) Find a free-frame (by taking one from the free-frame list, for example).

4) Read the desired page into the newly allocated frame.
5) Modify the internal-table and the page-table to indicate that the page is now in memory.
6) Restart the instruction that was interrupted by the trap.

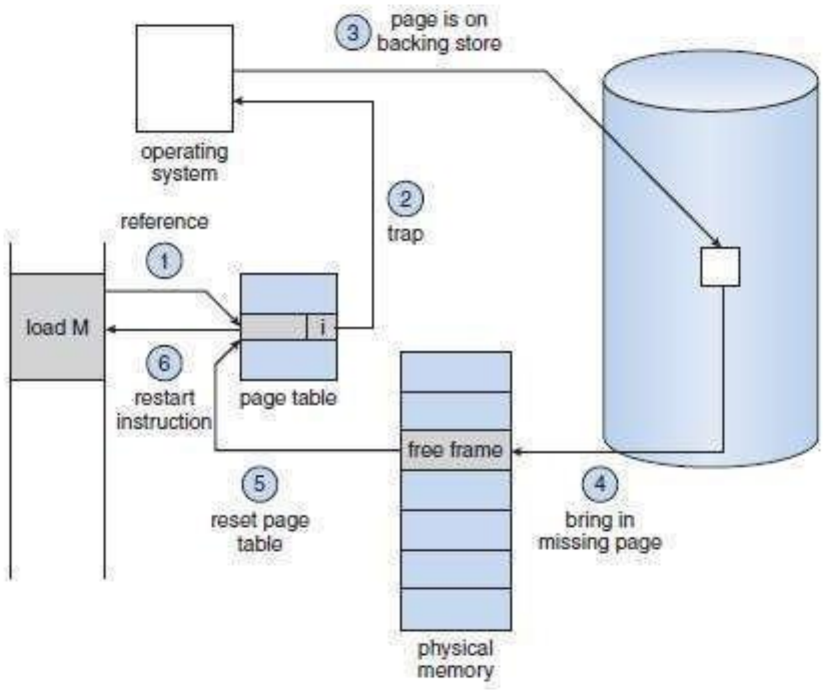


Figure 4.4 Steps in handling a page-fault

4.16 Allocation Methods

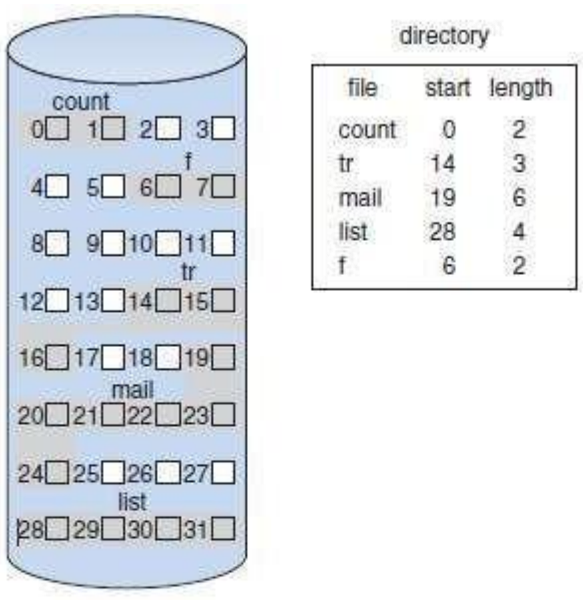
- The direct-access nature of disks allows us flexibility in the implementation of files.
- In almost every case, many files are stored on the same disk.
- Main problem:
  How to allocate space to the files so that
  → disk-space is utilized effectively and
  → files can be accessed quickly.
- Three methods of allocating disk-space:
  1) Contiguous
  2) Linked and
  3) Indexed
- Each method has advantages and disadvantages.
- Some systems support all three (Data General's RDOS for its Nova line of computers).

4.16.1 Contiguous Allocation

- Each file occupies a set of contiguous-blocks on the disk (Figure 4.30).
- Disk addresses define a linear ordering on the disk.
- The number of disk seeks required for accessing contiguously allocated files is minimal.
- Both sequential and direct access can be supported.
- Problems:
  1) Finding space for a new file
  ➢ External fragmentation can occur.
  2) Determining how much space is needed for a file.
  ➢ If you allocate too little space, it can't be extended.
  Two solutions:
  i) The user-program can be terminated with an appropriate error-message. The user must then allocate more space and run the program again.
  ii) Find a larger hole,

copy the contents of the file to the new space and
release the previous space.

- To minimize these drawbacks:
    1) A contiguous chunk of space can be allocated initially and
    2) Then when that amount is not large enough, another chunk of contiguous space (known as an „extent‟) is added.



file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

Figure 4.30 Contiguous allocation of disk-space

4.16.2 Linked Allocation

- Each file is a linked-list of disk-blocks.
- The disk-blocks may be scattered anywhere on the disk.
- The directory contains a pointer to the first and last blocks of the file (Figure 4.31).
- To create a new file, just create a new entry in the directory (each directory-entry has a pointer to the disk-block of the file).

  1) A **write** to the file causes a free block to be found. This new block is then written to and linked to the eof (end of file).

  2) A **read** to the file causes moving the pointers from block to block.
- Advantages:

  1) No external fragmentation, and any free block on the free-space list can be used to satisfy a request.

  2) The size of the file doesn't need to be declared on creation.

  3) Not necessary to compact disk-space.
- Disadvantages:

  1) Can be used effectively only for sequential-access files.

  2) Space required for the pointers.

  Solution: Collect blocks into multiples (called „clusters‟) & allocate clusters rather than blocks.

  3) Reliability: Problem occurs if a pointer is lost(or damaged).

  Partial solutions: i) Use doubly linked-lists.

  ii) Store file name and relative block-number in each block.

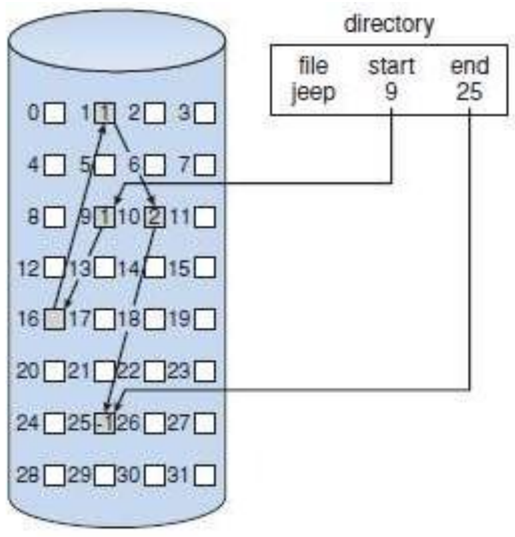


Figure 4.31 Linked allocation of disk-space

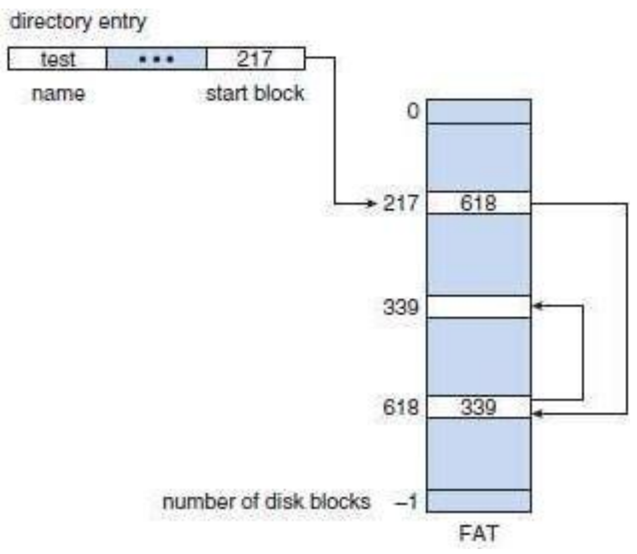


Figure 4.32 File-allocation table

- FAT is a variation on linked allocation (FAT=File Allocation Table).
- A section of disk at the beginning of each partition is set aside to contain the table (Figure 4.32).
- The table

  → has one entry for each disk-block and

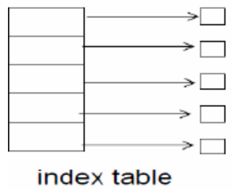
  → is indexed by block-number.
- The directory-entry contains the block-number of the first block in the file.
- The table entry indexed by that block-number then contains the block-number of the next block in the file.
- This chain continues until the last block, which has a special end-of-file value as the table entry.
- Advantages:

  1) Cache can be used to reduce the no. of disk head seeks.

  2) Improved access time, since the disk head can find the location of any block by reading the info in the FAT.

4.16.3 Indexed Allocation

- Solves the problems of linked allocation (without a FAT) by bringing all the pointers together into an index block.
- Each file has its own index block, which is an array of disk-block addresses.



Logical view of the Index Table

- The ith entry in the index block points to the ith file block (Figure 4.33).
- The directory contains the address of the index block.

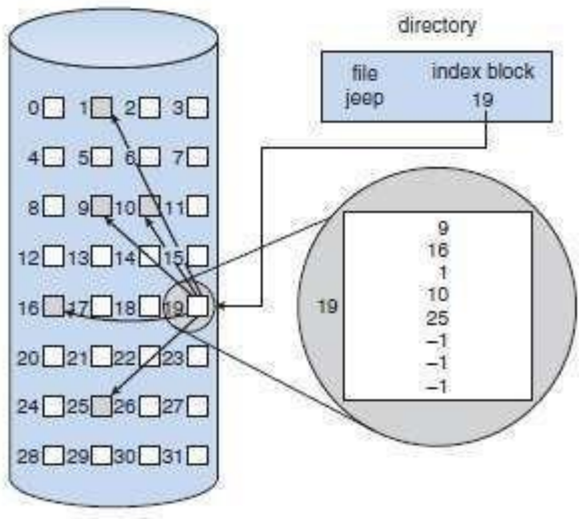


Figure 4.33 Indexed allocation of disk space

- When the file is created, all pointers in the index-block are set to nil.
- When writing the ith block, a block is obtained from the free-space manager, and its address put in the ith index-block entry,
- Problem: If the index block is too small, it will not be able to hold enough pointers for a large file, Solution: Three mechanisms to deal with this problem:

1) Linked Scheme

➢ To allow for large files, link several index blocks,

2) Multilevel Index

➢ A first-level index block points to second-level ones, which in turn point to the file blocks,

3) Combined Scheme

➢The first few pointers point to direct blocks (i.e. they contain addresses of blocks that contain data of the file).

- The next few pointers point to indirect blocks.