1-Introduction

1.1 Computer memory is the storage space in a computer where data is to be processed and instructions required for processing are stored.

1.2 Memory types:

A- Primary memory Consist of two types :

1- RAM (Random Access memory)

- It's also called read/write memory or main memory.The CPU can load instructions only from memory, so any programs to run must be stored there.In other word : Program must be brought (from disk) into memory and placed within a process for it to be run.
- When the computer is switched on, the operating system is loaded into RAM.
- When you start working on any application it's loaded into RAM and we are actually working on this copy.
- RAM is also called volatile memory meaning it will retain data as long as the electricity is available.

2- ROM (read-only memory)

- Because ROM cannot be changed, only static programs, such as the bootstrap program described earlier, are stored there. Without ROM, we cannot load the operating system and therefore we cannot work on the computer.
- EEPROM (electrically erasable programmable read-only memory).It can be changed but cannot be changed frequently and so contains mostly static programs. For example, smartphones have EEPROM to store their factory-installed programs.

3-**Cache Memory** is a special, very high-speed memory. It is used to speed up and synchronise with high-speed CPUs. Cache memory is costlier than main memory or disk memory. Cache memory is a fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed. Cache memory is used to reduce the average time to access data from the Main memory.

4- **Registers:** Registers are small amounts of high-speed memory contained within the CPU. They are used by the processor to store small amounts of data that are needed during processing, such as: the address of the next instruction to be executed.

B- Secondary memory is also called auxiliary memory, external memory, Backup memory.

Types of secondary memory:

1- **Hard Disk**: This is one of the primary types of non-volatile memory used to store a large amount of data on the computer, it can reside inside the computer or additionally outside.

2- **Compact Disk (CD)**:Is another type of non-volatile memory once widely used.Its portable and cheaper.

3- **Digital Versatile Disk(DVD)** It's the same as the compact disk but it is capable of storing six times of data stored in the CD. In addition, it can be single side or double sides increasing the capacity further.

4- **USB Drive:** It's also called a flash drive or pen drive, this can be connected to the usb(universal serial bus port) of the computer, this is similar to the hard disk but it is portable and very smaller in size and less expensive.

5- **Memory card** : Is very small in size and used to store data in various electronic devices such as digital cameras, smart phones, MP3 players and many more.

The reasons of using secondary memory are:

1- The primary memory has limited capacity.

2- RAM volatile memory and ROM is unchangeable memory.

1.3 Memory addresses

All forms of memory provide an array of bytes. Each byte has its own address. Interaction is achieved through a sequence of load or store instructions to specific memory addresses.

The load instruction moves a byte or word from the main memory to an internal register within the CPU, whereas the store instruction moves the content of a register to the main memory. Aside from loads and stores, the CPU automatically loads instructions from main memory for execution.

A typical instruction–execution cycle, as executed on a system, first fetches an instruction from memory and stores that instruction in the instruction register. The instruction is then decoded and then the instruction to be executed, the result may be stored back in memory.

Important Background

- Program must be brought (from disk) into memory and placed within a process for it to be run.
- Main memory and registers are only storage CPU can access directly.
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests.
- Register access in one CPU clock (or less).
- Main memory can take many cycles, causing a stall of CPU.
- Cache sits between main memory and CPU registers.
- Protection of memory required to ensure correct operation.

1.4 Protection of memory

Base and Limit Registers:

- We first need to make sure that each process has a separate memory space.
- Separate per-process memory space protects the processes from each other and is fundamental to having multiple processes loaded in memory for concurrent execution.

- To separate memory spaces, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses.
- We can provide this protection by using two registers, usually a base and a limit, as illustrated in Figure1 below.
- The base register holds the smallest legal physical memory address; the limit register specifies the size of the range or the number of bytes in the allocation.

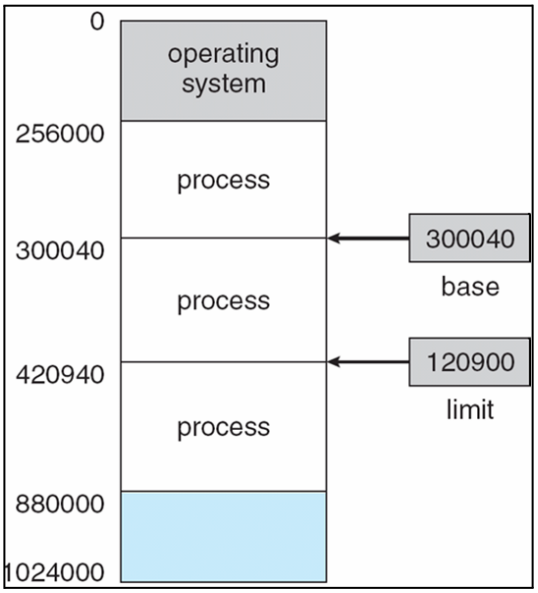


Figure 1 -A base and a limit register define a logical address space.

Note1: Protection of memory space is accomplished by having the CPU hardware compare every address generated in user mode with the registers.

Note2: Any attempt by a program executing in user mode to access operating-system memory or other users' memory results in a trap to the operating system, which treats the attempt as a fatal error (Figure2).

Note3: This scheme prevents a user program from (accidentally or deliberately) modifying the code or data structures of either the operating system or other users.

Note4: The base and limit registers can be loaded only by the operating system, which uses a special privileged instruction. Since privileged instructions can be executed only in kernel mode, and since only the operating system executes in kernel mode, only the operating system can load the base and limit registers.

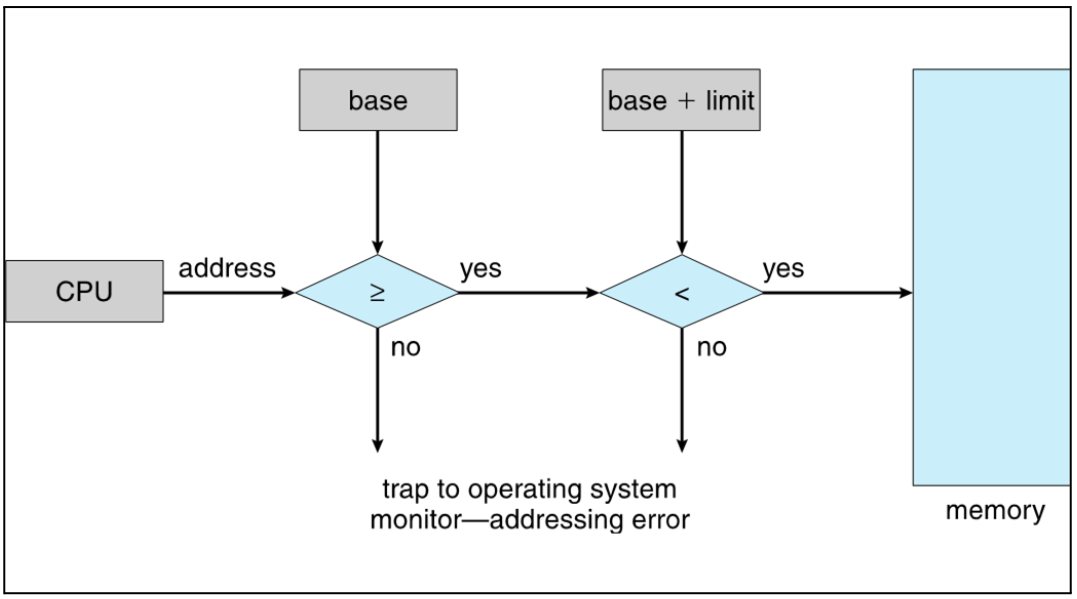


Figure2 :Hardware address protection with base and limit registers.

Example : If the base register address of a process is 200 and the limit register is 50, that means the legal addresses are from 200 to 249.

1.5 Address binding:

- Usually, a program resides on a disk as a binary executable file. To be executed, the program must be brought into memory and placed within a process.
- Depending on the memory management in use, the process may be moved between disk and memory during its execution.
- Most systems allow a user process to reside in any part of the physical memory. Thus, although the address space of the computer may start at 00000, the first address of the user process need not be 00000.
- In most cases, a user program goes through several steps before being executed.
- Further, addresses represented in different ways at different stages of a program's life:
  - Source code addresses are usually symbolic. i.e count.
  - Compiled code addresses **bind** to relocatable address (virtual address) i.e. "14 bytes from beginning of this module".
  - Linker or loader will bind relocatable addresses to absolute addresses(physical address) i.e. 74014.

1.6 Binding of Instructions and Data to Memory

Address binding of instructions and data to memory addresses can happen at three different stages:

Compile time: If memory location for a process is known a priori, absolute code can be generated; must recompile code if starting location changes.

Load time: Must generate relocatable code if memory location of a process is not known at compile time.

Execution time: Binding delayed until run time if the process can be moved during its execution from one memory segment to another. This need hardware support for address maps (e.g., base and limit registers)

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 – 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.
- 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.
- The run-time mapping from virtual to physical addresses is done by a hardware device called the memory-management unit (MMU).

1.7 Memory-management unit (MMU)

- The user program generates only logical addresses and thinks that the process runs in locations 0 to max. However, these logical addresses must be mapped to physical addresses before they are used. The concept of a logical address space that is bound to a separate physical address space is central to proper memory management.
- Memory-management unit is a hardware device that at run time maps virtual address to physical address.
- We can choose from many different methods to accomplish such mapping. We illustrate this mapping with a simple MMU scheme that is a generalization of the base-register scheme. The base register is now called a relocation register.

1.8 Dynamic loading

- It is necessary that the entire program and all the process data in the physical memory for the process to execute.
- To obtain better memory-space utilization, we can use dynamic loading.
- With dynamic loading, a routine is not loaded until it is called.
- All routines are kept on disk in a relocatable load format.
- The main program is loaded into memory and is executed. When a routine needs to call another routine, the calling routine first checks to see whether the other routine has been loaded. If it has not, the relocatable linking loader is called to load the desired routine into memory and to update the program's address tables to reflect this change. Then control is passed to the newly loaded routine.
- The advantage of dynamic loading is that a routine is loaded only when it is needed. This method is particularly useful when large amounts of code are needed to handle infrequently occurring cases, such as error routines. In this case, although the total program size may be large, the portion that is used (and hence loaded) may be much smaller.
- The value in the relocation register is added to every address generated by a user process at the time the address is sent to memory (see Figure 3).

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

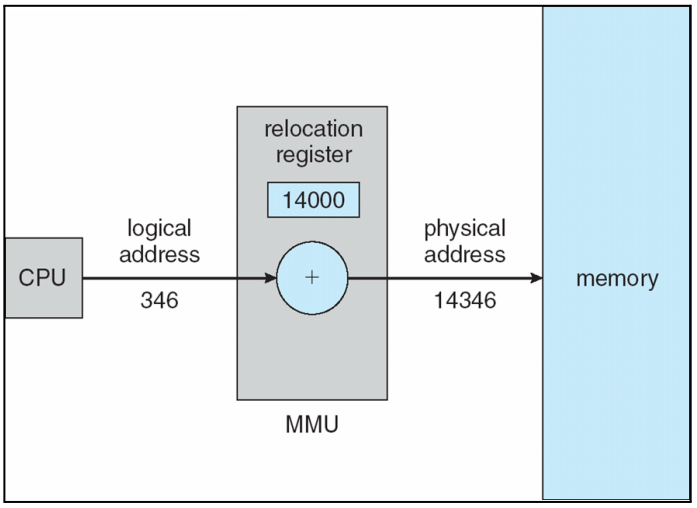


Figure3 Dynamic relocation using a relocation register.

1.9 Dynamic Linking and shared libraries

- Dynamically linked libraries are system libraries that are linked to user programs when the programs are run. See (figure 4).
- Static linking – system libraries and program code combined by the loader into the binary program image.

- Dynamic linking –linking postponed until execution time.
- This feature is usually used with system libraries, such as language subroutine libraries. Without this facility, each program on a system must include a copy of its language library in the executable image. This requirement wastes both disk space and main memory.
- Small piece of code, **stub**, included in the image of each library routine reference, is used to locate the appropriate memory-resident library routine. Stub checks to see whether the needed routine is already in memory. If it is not, the program loads the routine into memory.
- Dynamic linking is particularly useful for libraries.
- Under this scheme, all processes that use a language library execute only one copy of the library code so it's also known as shared libraries.

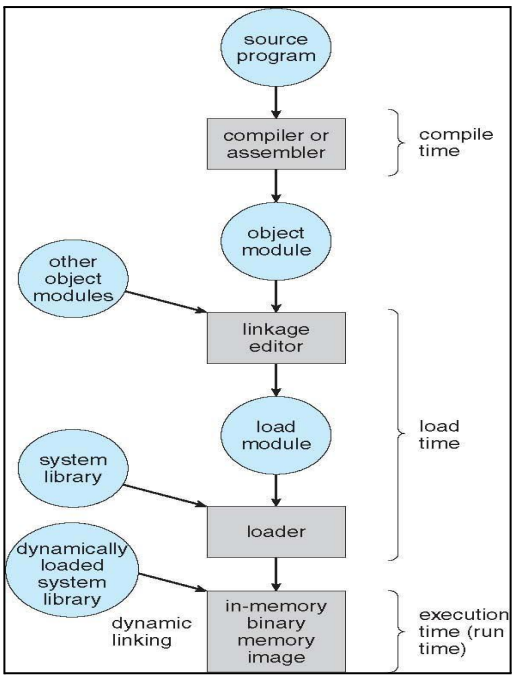


Figure 4 (Multistep Processing of a User Program)

1.10 Swapping

- Swapping : is a memory management scheme in which any process can be temporarily swapped from main memory to secondary so that the main memory can be available for other processes.
- It is used to improve main memory utilization.
- The disk used to swap out is called a backing store.
- The operation of transferring a process from main memory to backing store is called swap out or roll out.
- The operation of transferring a process from the backing store to main memory is called swap in or roll in.
- See figure 5.
- The swapping between processes occurs depending on scheduling mechanism, for example, if the mechanism scheduling is the priority, then the process with the highest priority is inserted and the lowest priority is removed from the main memory (ready queue).

- The context-switch time in such a swapping system is fairly high. To get an idea of the context-switch time, let's assume that the user process is 100 MB in size and the backing store is a standard hard disk with a transfer rate of 50 MB

per second. The actual transfer of the 100-MB process to or from main memory takes :

100 MB/50 MB per second = 2 seconds (2000 milliseconds).

Since we must swap both out and in, the total swap time is about 4,000 milliseconds.

- Notice that the major part of the swap time is transfer time. The total transfer time is directly proportional to the amount of memory swapped.

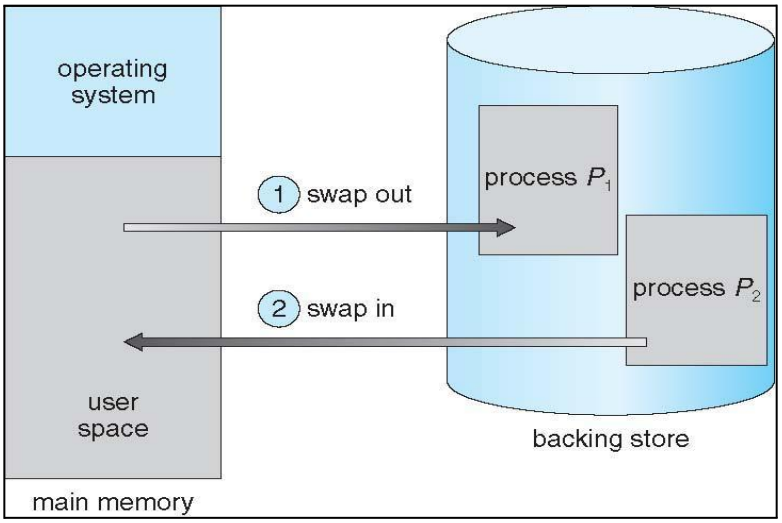


Figure 5: Swapping of two processes using a disk as a backing store.

1.11 Segmentation

1-11-a Method

- When writing a program, a programmer thinks of it as a main program with a set of methods, procedures, or functions. It may also include various data structures: objects, arrays, stacks, variables, and so on. Each of these modules or data elements is referred to by name.
- The programmer talks about “the stack,” “the math library,” and “the main program” without caring what addresses in memory these elements occupy.
- Segmentation is a memory-management scheme that supports this programmer view of memory.
- A logical address space is a collection of segments.

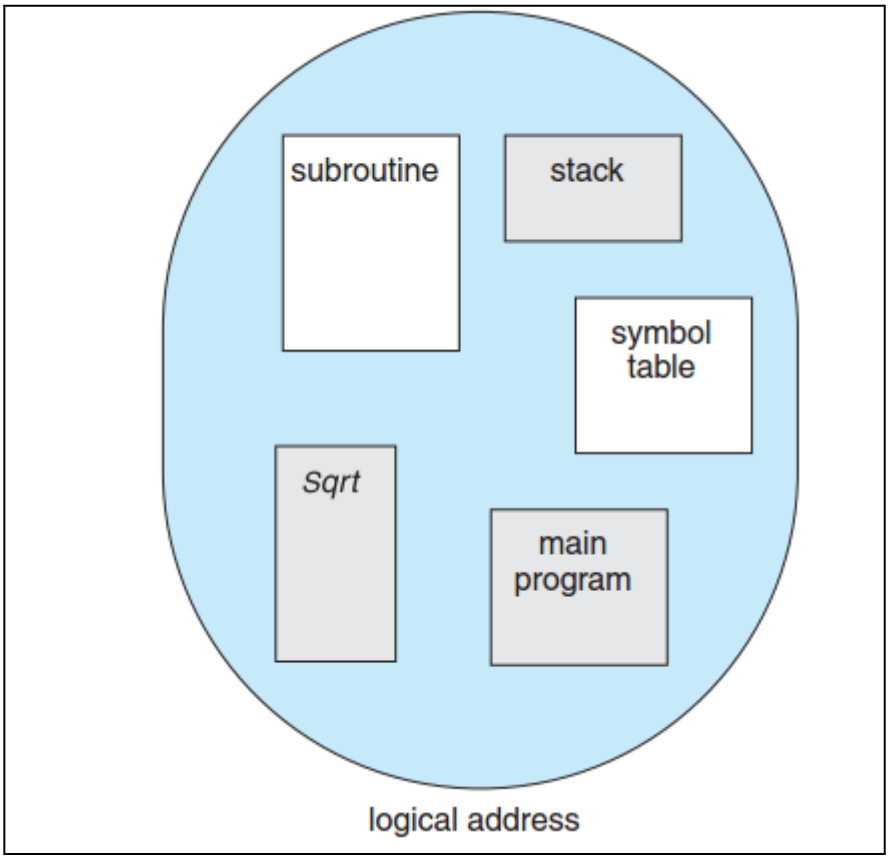


Figure 6: Programmer’s view of a program.

- Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment.
- For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name. Thus, a logical address consists of a two tuple: **<segment-number, offset>**

1.11.b Segmentation Hardware

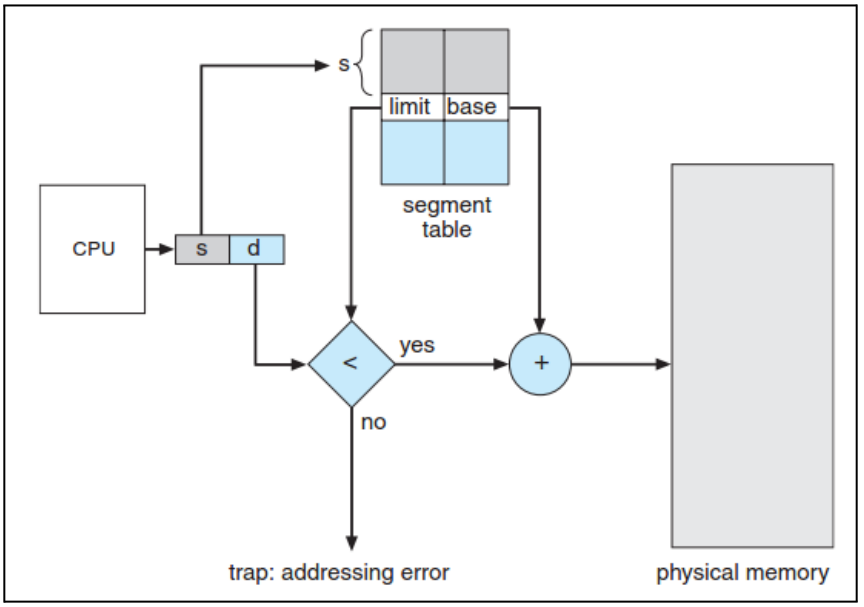


Figure 7: segmentation hardware

- The segment table is thus essentially an array of base–limit register pairs. It maps two-dimensional physical addresses. Each entry in the segment table has:

  -**base** : contains the starting physical address where the segments reside in memory.

  - **limit** : specifies the length of the segment.

- A logical address consists of two parts: a segment number, **s**, and an offset into that segment, **d**.

- The segment number **s** is used as an index to the segment table.

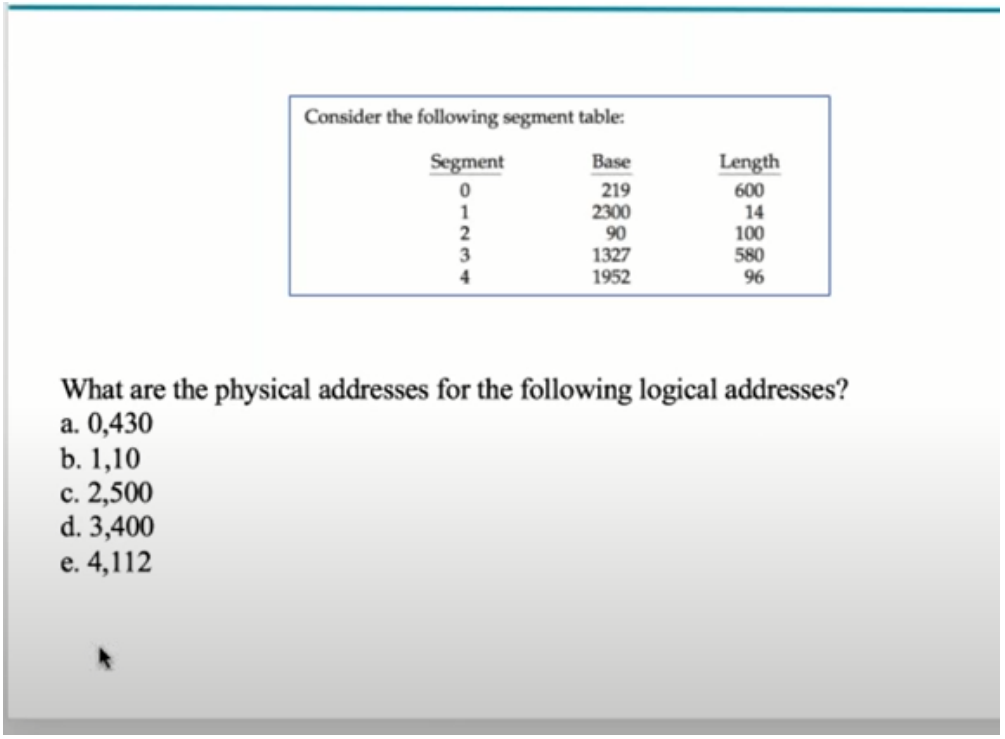
- The offset **d** of the logical address must be between 0 and the segment limit. If it is not, we trap the operating system. When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte.

- 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**

correct offset + segment base = address in Physical memory

Example of mapping logical addresses using segmentation scheme:



Consider the following segment table:

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What are the physical addresses for the following logical addresses?
a. 0,430
b. 1,10
c. 2,500
d. 3,400
e. 4,112

11.12 Paging

- The paging technique divides the physical memory(main memory) into fixed-size blocks that are known as Frames and also divides the logical memory(secondary memory) into blocks of the same size that are known as Pages.
- The Frame has the same size as that of a Page. A frame is basically a place where a (logical) page can be (physically) placed.
- Each process is mainly divided into parts where the size of each part is the same as the page size.There is a possibility that the size of the last part may be less than the page size.(internal fragmentation).
- One page of a process is mainly stored in one of the frames of the memory. Also, the pages can be stored at different locations of the memory but always the main priority is to find contiguous frames(contiguous allocation).

11.12.1 Translation of Logical Address into Physical Address

Important notes:

- The CPU always generates a logical address.
- In order to access the main memory, a physical address is needed.

- The logical address consists of two parts:
  1. **Page Number(p)** used as an index into a page table which contains the base address of each page in physical memory.In this scheme the base address is frame number.
  2. **Page offset (d)** –used to specify the specific word on the page that the CPU wants to read. It combined with base address to define the physical memory address that is sent to the memory unit.

11.12.2 Page Table

- ☐ The Page table mainly contains the base address of each page in the Physical memory.
- ☐ The base address is then combined with the page offset in order to define the physical memory address which is then sent to the memory unit.
- ☐ The page table mainly provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.
- ☐ As we have told you above that the frame number is combined with the page offset and forms the required physical address.
- ☐ So, The physical address consists of two parts:
  - Frame Number(f)
  - Page offset(d)

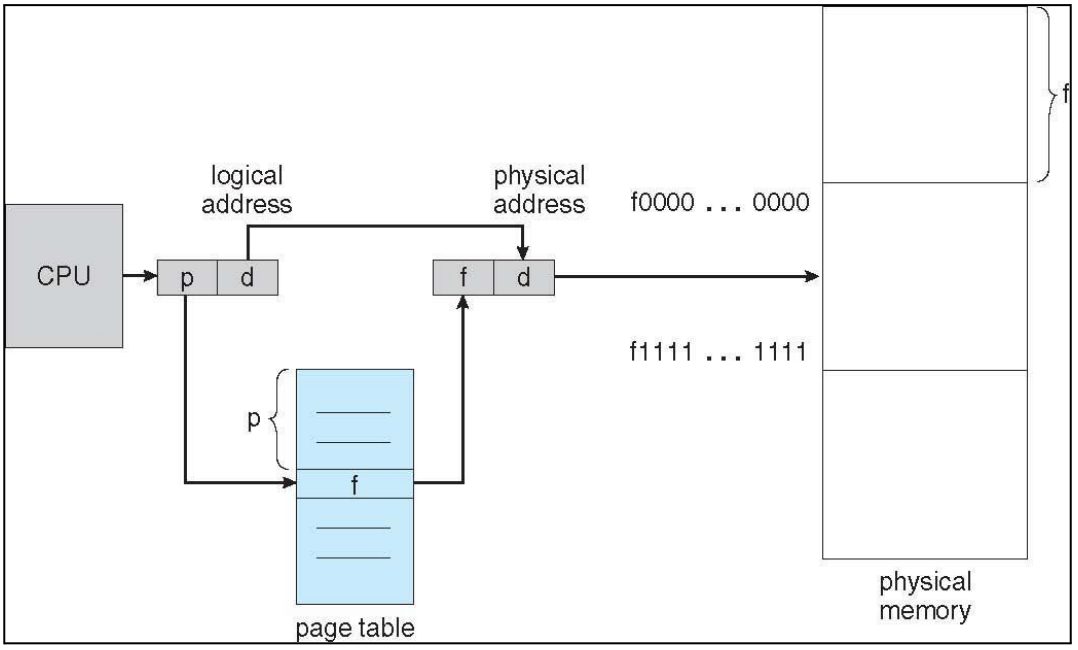


Figure 8: paging hardware

Examples: What is the physical address ?

1- Memory size= 32 byte, Logical address=0, page size=4 , no. of pages=8

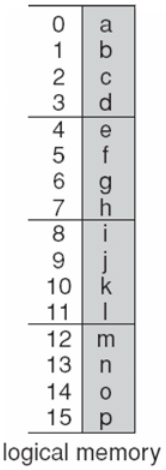
Sol:

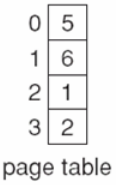
1- find page no.= logical address / page size

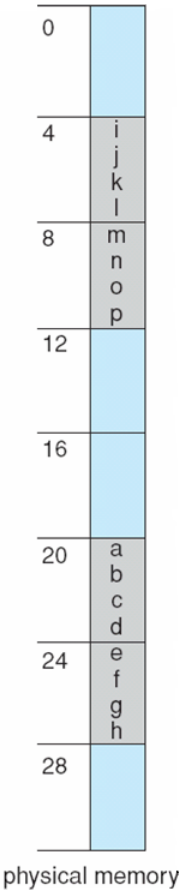
Then, get the frame number from page table

2- find offset= logical address % page size

Then , the physical address= frame no.,offset in main memory







- ☐ Page table is 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. Thus memory access is slower by a factor of 2 and in most cases, this scheme slowed by a factor of 2.
- ☐ The two memory access problems can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs).**

Advantages of Paging:

1-Paging mainly allows storage of parts of a single process in a non-contiguous fashion.

2- With the help of Paging, the problem of external fragmentation is solved.

3-Paging is one of the simplest algorithms for memory management.

Disadvantages of Paging

- In Paging, sometimes the page table consumes more memory.
- Internal fragmentation is caused by this technique.
- There is an increase in time taken to fetch the instruction since now two memory accesses are required.

Background:

The requirement that instructions must be in physical memory to be executed seems necessary ; but that is a disadvantage, since it limits the size of a program to the size of physical memory. In fact, in many cases, the entire program is not needed. For instance, consider the following:

- Programs often have code to handle unusual error conditions. Since these errors seldom, if ever, occur in practice, this code is almost never executed.
- Arrays, lists, and tables are often allocated more memory than they actually need. An array may be declared 100 by 100 elements, even though it is seldom larger than 10 by 10 elements.

Even in those cases where the entire program is needed, it may not all be needed at the same time. The ability to execute a program that is only partially in memory would confer many benefits:

- A program would no longer be constrained by the amount of physical memory that is available. Users would be able to write programs for an extremely large virtual address space, simplifying the programming task.
- Because each user program could take less physical memory, more programs could be run at the same time, with a corresponding increase in CPU utilization and throughput but with no increase in response time or turnaround time.
- Less I/O would be needed to load or swap user programs into memory, so each user program would run faster.

Virtual memory

- Virtual Memory is a storage scheme that provides users with an illusion of having a very big main memory. This is done by treating a part of secondary memory as the main memory. It is a section of a hard disk that's set up to emulate the computer's RAM.
- Virtual memory involves the separation of logical memory as perceived by users from physical memory. This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available.
- Virtual memory makes the task of programming much easier, because the programmer no longer needs to worry about the amount of physical memory available.

Virtual memory can be implemented via:

-Demand paging

-Demand segmentation

Demand paging:

- Is a strategy to load pages into memory only as they are needed. This technique is commonly used in virtual memory systems.
- With demand-paged virtual memory, pages are loaded only when they are demanded during program execution. Pages that are never accessed are thus never loaded into physical memory.
- A demand-paging system is similar to a paging system with swapping (Figure 1) where processes reside in secondary memory (usually a disk).
- When we want to execute a page-s of a process, we swap it into memory. Rather than swapping the entire process into memory, though, we use a lazy swapper.
- A lazy swapper never swaps a page into memory unless that page will be needed.
- In the context of a demand-paging system, we use “pager,” rather than “swapper”, because a swapper manipulates entire processes, whereas a pager is manipulated with the individual pages of a process.
- The pager brings only those pages into memory. Thus, it avoids reading into memory pages that will not be used anyway, decreasing the swap time and the amount of physical memory needed.

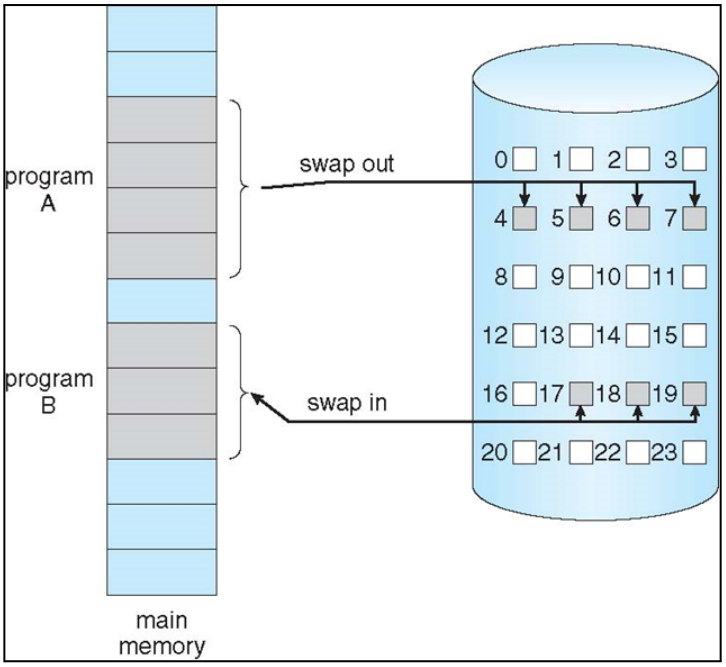


Figure-1 Transfer of a paged memory to contiguous disk space.

valid–invalid bit

- With the demand paging scheme, we need some form of hardware support to distinguish between the pages that are in memory and the pages that are on the disk. So valid-invalid bit is used for this purpose.
- valid–invalid bit is located in the page table, it associated with each page table entry:
  - ➔ when this bit is set to "valid,"(v) the associated page is both legal and in memory.
  - ➔ If the bit is set to invalid (i), the page may be in one of two states:
    - ◆ is not valid (that is, not in the logical address space of the process). Leads to page abort.
    - ◆ or is valid but is currently on the disk. Leads to page faults.
- See (Figure 2) for these cases.

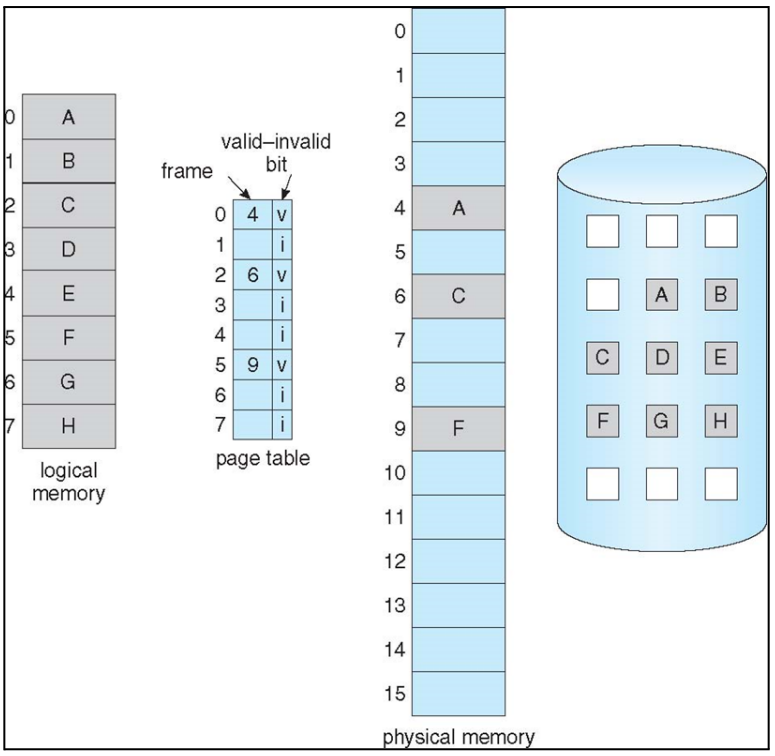


Figure 2 Page Table When Some Pages Are Not in Main Memory

Page fault:

- If the process tries to access a page that was not brought into memory (the bit value is i and it is not in the memory), this case will cause page fault.
- The paging hardware, in translating the address through the page table, will notice that the invalid bit (i) is set, causing a trap to the operating system. This trap is the result of the operating system's failure to bring the desired page into memory.
- The procedure for handling this page fault is straightforward (Figure 3):
    1. Checking the page reference in the page table, if it's a valid reference but not in the memory yet .
    2. Trap the operating system.
    3. Find a free frame (by taking one from the free-frame list, for example).
    4. Getting the desired page from disk into the allocated memory frame.
    5. When the disk read is complete, the page table must be updated 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.

-

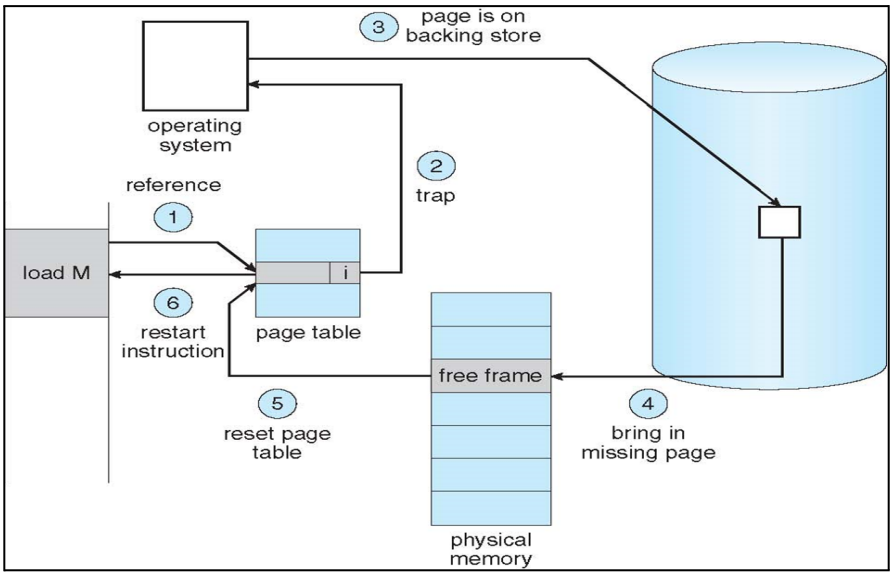


Figure 3: Steps in handling page fault

Performance of demand paging

- Page fault time is the time it takes to fetch the page from disk to main memory, it is can be calculated as :

  Page fault time = trap OS + swap out + swap in + update page table

- To calculate the performance of demand paging ,we need to find effective memory access time :

  EMT = (1-p)*(ma) + (p)* (pft)

  Where :

  - P : the probability of page fault occurring (0<=p<=1).
  - pft: page fault time.
  - ma: memory access.

Example:

Memory Access Time = 200 nanoseconds

Average Page Fault Service Time = 8 milliseconds

EAT = (1-p)*200+p(8 milliseconds)

= (1-p)*200+p*8000000

= 200+p*7999800

Page replacement :

- A page fault happens when a running program accesses a memory page that is mapped into the virtual address space but not loaded in physical memory.

- In an operating system that uses paging for memory management, when page fault occurs and there is no free frame in main memory, Page replacement is needed to find one (frame) that is not currently being used and free it .

Basic Page Replacement:

The page replacement approach is implemented as follow :

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. Continue the user process from where the page fault occurred.

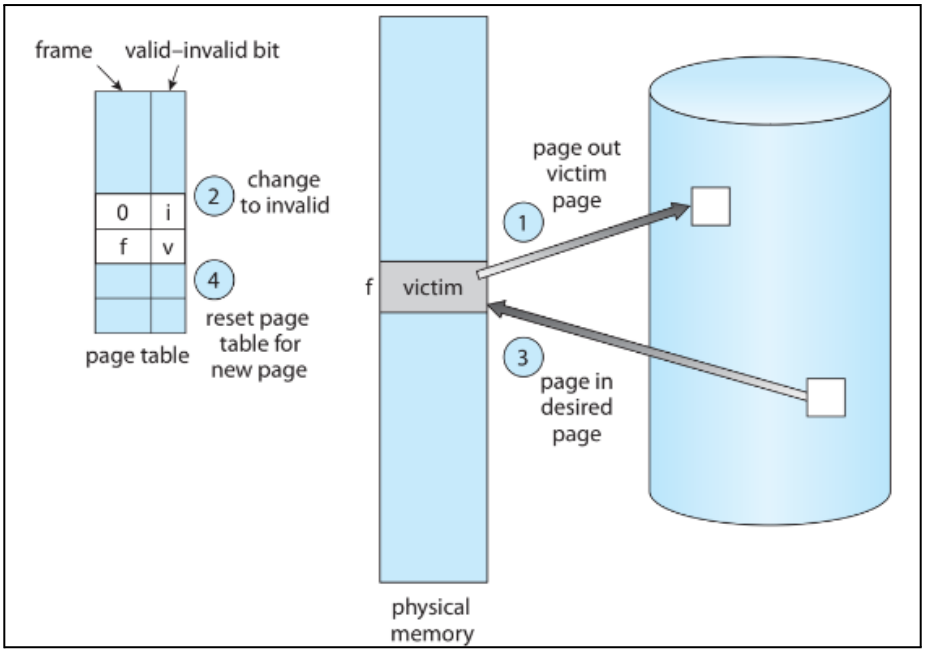


Figure 4: Page replacement

Modify bit (dirty bit):

- If no frames are free, two page transfers (one out and one in) are required.
- This situation effectively doubles the page-fault service time and increases the effective access time.
- We can reduce this overhead by using a modify bit (or dirty bit).
- When this scheme is used, each page or frame has a modify bit associated with it.
- The modify bit for a page is set whenever any byte in the page is written into, indicating that the page has been modified.
- When we select a page for replacement, we examine its modify bit. If the bit is set, we know that the page has been modified since it was read in from the disk. In this case, we must write the page to the disk.
- If the modify bit is not set, however, the page has not been modified since it was read into memory. In this case, we need not write the memory page to the disk: it is already there.
- This scheme can significantly reduce the time required to service a page fault, since it reduces I/O time by one-half if the page has not been modified.

Page replacement algorithms:

There are many different page-replacement algorithms. Every operating system probably has its own replacement scheme. How do we select a particular replacement algorithm? In general, we want the one with the lowest page-fault rate.

The choice of which page to replace is specified by page replacement algorithms:

1- FIFO Page Replacement algorithm

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced, the page in the front of the queue is selected for removal.

Example:

F F F F H **F F F F F F** H H **F F** H H **F F F**

Reference string: **7,0,1,2,0, 3,0,4,2, 3, 0, 3, 2, 1,2, 0, 1, 7,0, 1**

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

Page faults= 15 , **Fault rate= no.page fault/no. of reference string**

Fault rate=15/20= 0.75

Belady's Anomaly

- Generally, on increasing the number of frames to a process' virtual memory, its execution becomes faster as fewer page faults occur. Sometimes the reverse happens, i.e. more page faults occur when more frames are allocated to a process. This most unexpected result is termed Belady's Anomaly. This case may occur with FIFO page replacement algorithm.
- Example: if we consider reference strings 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4, and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10-page faults.

F F F F F F F H H F F H

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

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

F F F F H H F F F F F F

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

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

2- Optimal page replacement algorithm

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Example1: Consider the page references string:

F F F F H **F** H **F** H H **F** H H **F** H H H **F** H H

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

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

Page faults=9 , fault rate = 9/20 =0.45

Example 2: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frames. Find number of page faults.

F F F F H F H F H H H H H H

Page reference 7,0,1,2, 0,3,0, 4,2 ,3, 0, 3, 2, 3 No. of page frame=4

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

Total page fault=6 , fault rate =6/14= 0.4

Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

3- Least Recently Used(LRU): In this algorithm, the page will be replaced which is least recently used.

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.
- LRU and OPT are cases of stack algorithms that don't have Belady's Anomaly.

Example: if we have the pages reference string:

F F F F H **F** H **F F F F** H H **F** H **F** H **F** H H

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

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

Faults pages = 12 , fault rate=12/20 =0.6

1.11 Segmentation

1-11-a Method

- When writing a program, a programmer thinks of it as a main program with a set of methods, procedures, or functions. It may also include various data structures: objects, arrays, stacks, variables, and so on. Each of these modules or data elements is referred to by name.
- The programmer talks about “the stack,” “the math library,” and “the main program” without caring what addresses in memory these elements occupy.
- Segmentation is a memory-management scheme that supports this programmer view of memory.
- A logical address space is a collection of segments.

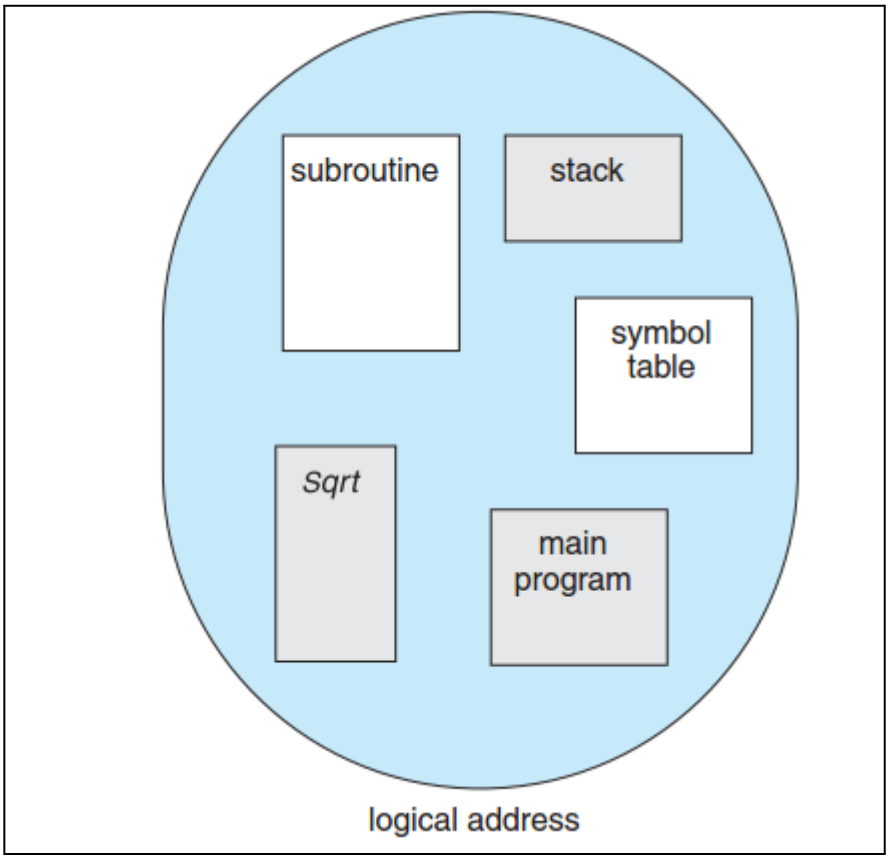


Figure 6: Programmer’s view of a program.

- Each segment has a name and a length. The addresses specify both the segment name and the offset within the segment.
- For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name. Thus, a logical address consists of a two tuple: **<segment-number, offset>**

1.11.b Segmentation Hardware

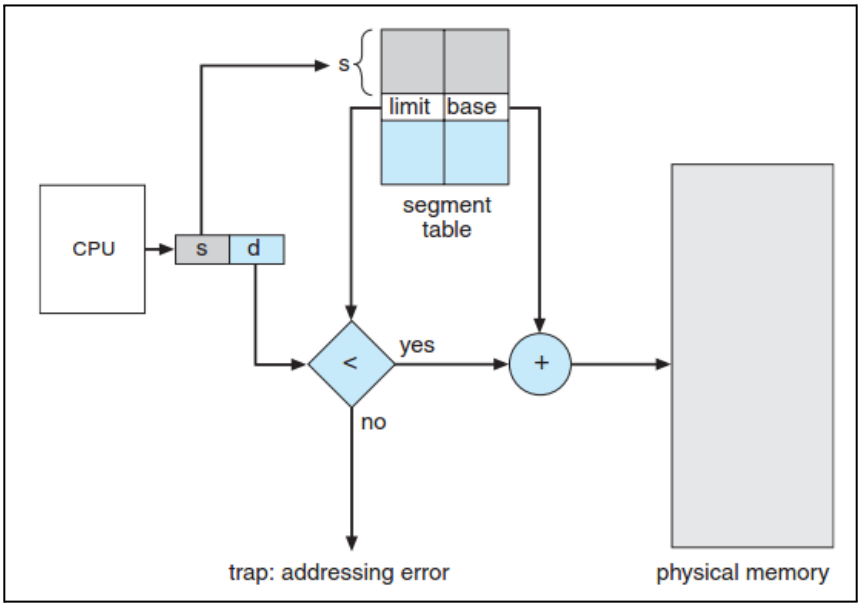


Figure 7: segmentation hardware

- The segment table is thus essentially an array of base–limit register pairs. It maps two-dimensional physical addresses. Each entry in the segment table has:

  -**base** : contains the starting physical address where the segments reside in memory.

  - **limit** : specifies the length of the segment.

- A logical address consists of two parts: a segment number, **s**, and an offset into that segment, **d**.

- The segment number **s** is used as an index to the segment table.

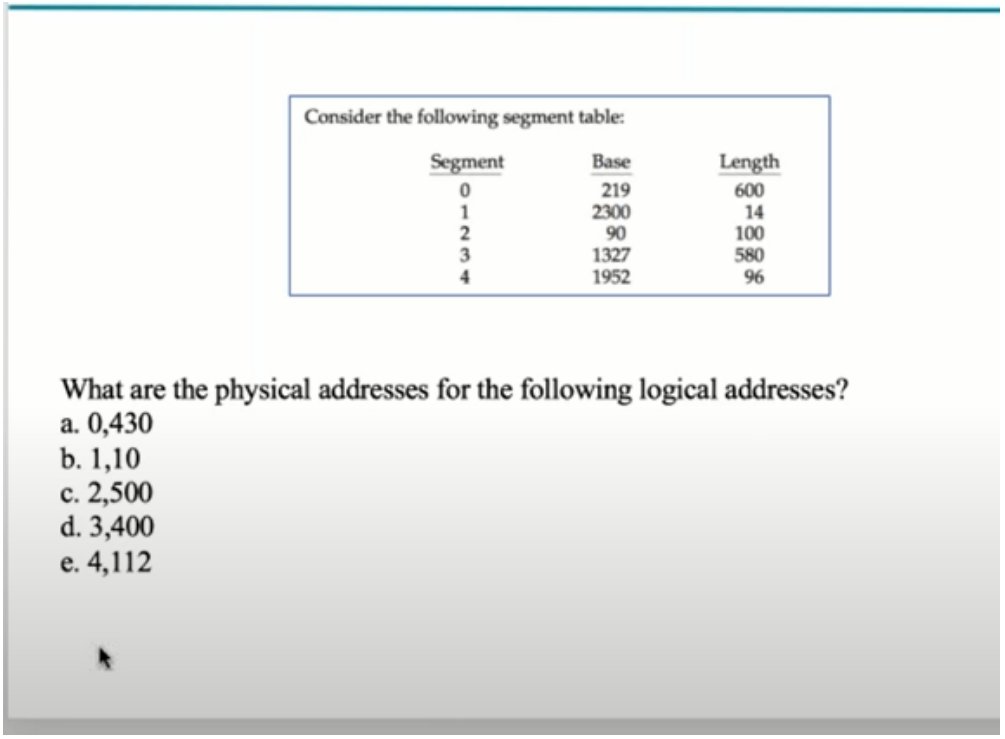
- The offset **d** of the logical address must be between 0 and the segment limit. If it is not, we trap the operating system. When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte.

- 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**

correct offset + segment base = address in Physical memory

Example of mapping logical addresses using segmentation scheme:



Consider the following segment table:

Segment	Base	Length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

What are the physical addresses for the following logical addresses?
a. 0,430
b. 1,10
c. 2,500
d. 3,400
e. 4,112

11.12 Paging

- The paging technique divides the physical memory(main memory) into fixed-size blocks that are known as Frames and also divides the logical memory(secondary memory) into blocks of the same size that are known as Pages.
- The Frame has the same size as that of a Page. A frame is basically a place where a (logical) page can be (physically) placed.
- Each process is mainly divided into parts where the size of each part is the same as the page size.There is a possibility that the size of the last part may be less than the page size.(internal fragmentation).
- One page of a process is mainly stored in one of the frames of the memory. Also, the pages can be stored at different locations of the memory but always the main priority is to find contiguous frames(contiguous allocation).

11.12.1 Translation of Logical Address into Physical Address

Important notes:

- The CPU always generates a logical address.
- In order to access the main memory, a physical address is needed.

- The logical address consists of two parts:
  1. **Page Number(p)** used as an index into a page table which contains the base address of each page in physical memory.In this scheme the base address is frame number.
  2. **Page offset (d)** –used to specify the specific word on the page that the CPU wants to read. It combined with base address to define the physical memory address that is sent to the memory unit.

11.12.2 Page Table in OS

- ☐ The Page table mainly contains the base address of each page in the Physical memory.
- ☐ The base address is then combined with the page offset in order to define the physical memory address which is then sent to the memory unit.
- ☐ The page table mainly provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.
- ☐ As we have told you above that the frame number is combined with the page offset and forms the required physical address.
- ☐ So, The physical address consists of two parts:
  - Page offset(d)
  - Frame Number(f)

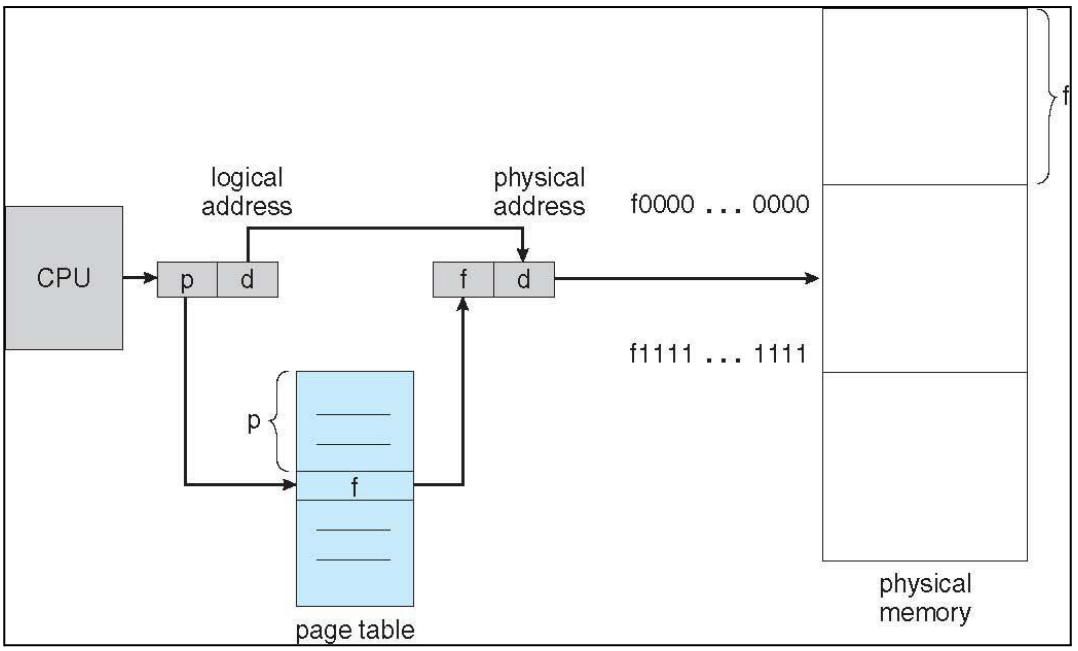


Figure 8: paging hardware

Examples: what is the physical address ?

1- Memory size= 32 byte, Logical address=0, page size=4 , no. of pages=8

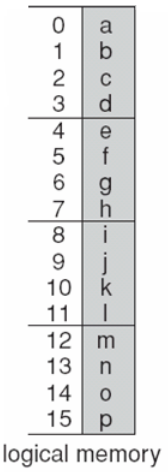
Sol:

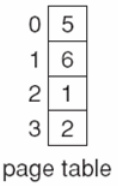
1- find page no.= logical address / page size

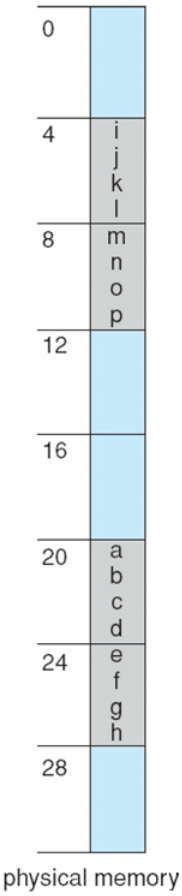
Then, get the frame number from page table

2- find offset= logical address % page size

Then , the physical address= frame no.,offset in main memory







- ☐ Page table is 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. Thus memory access is slower by a factor of 2 and in most cases, this scheme slowed by a factor of 2.
- ☐ 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).**

Advantages of Paging:

1-Paging mainly allows storage of parts of a single process in a non-contiguous fashion.

2- With the help of Paging, the problem of external fragmentation is solved.

3-Paging is one of the simplest algorithms for memory management.

Disadvantages of Paging

- In Paging, sometimes the page table consumes more memory.
- Internal fragmentation is caused by this technique.
- There is an increase in time taken to fetch the instruction since now two memory accesses are required.

1- File concept :

- Computers can store information on various storage media, such as magnetic disks, magnetic tapes, and optical disks. So that the computer system will be convenient to use, the operating system provides a uniform logical view of stored information.
- The logical storage unit is the file. Files are mapped by the operating system onto physical devices. These storage devices are usually nonvolatile, so the contents are persistent between system reboots.
- A file is a named collection of related information that is recorded on secondary storage. From a user's perspective, a file is the smallest allotment of logical secondary storage; that is, data cannot be written to secondary storage unless they are within a file.
- Commonly, files represent programs and data.
- Many different types of information may be stored in a file—source or executable programs, numeric or text data, photos, music, video, and so on.
- A file has a certain defined structure, which depends on its type. A text file is a sequence of characters organized into lines (and possibly pages). A source file is a sequence of functions, each of which is further organized as declarations followed by executable statements.
- An executable file is a series of code sections that the loader can bring into memory and execute.

2- File Attributes

- A file is named , and is referred to by its name. A name is usually a string of characters, such as example.c.
- When a file is named, it becomes independent of the process, the user, and even the system that created it. For instance, one user might create the file example.c, and another user might edit that file by specifying its name. The file's owner might write the file to a USB disk, send it as an email attachment, or copy it across a network, and it could still be called example.c on the destination system.

A file's attributes vary from one operating system to another but typically consist of these:

- **Name**. The symbolic file name is the only information kept in human-readable form.
- **Identifier**. This unique tag, usually a number, identifies the file within the file system; it is the non-human-readable name for the file.
- **Type**. This information is needed for systems that support different types of files.
- **Location**. This information is a pointer to a device and to the location of the file on that device.
- **Size**. The current size of the file (in bytes, words, or blocks) and possibly the maximum allowed size are included in this attribute.
- **Protection**. Access-control information determines who can do reading, writing, executing, and so on.
- **Time, date, and user identification**. This information may be kept for creation, last modification, and last use. These data can be useful for protection, security, and usage monitoring.

The information about all files is kept in the directory structure, which also resides on secondary storage. Typically, a directory entry consists of the file's name and its unique identifier. The identifier in turn locates the other file attributes. It may take more than a kilobyte to record this information for each file. In a system with many files, the size of the directory itself may be megabytes. Because directories, like files, must be nonvolatile, they must be stored on the device and brought into memory, as needed.

3- File Operations

- Creating a file. Two steps are necessary to create a file. First, space in the file system must be found for the file. Second, an entry for the new file must be made in the directory(check if the file is not found in the directory).

- Writing a file. To write a file, we make a system call specifying both the name of the file and the information to be written to the file. Given the name of the file, the system searches the directory to find the file's location.The system must keep a write pointer to the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.

- Reading a file. To read from a file, we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put. Again, the directory is searched for the associated entry, and the system needs to keep a read pointer to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated. Both the read and write operations use the same pointer, saving space and reducing system complexity.

- Deleting a file. To delete a file, we search the directory for the named file. Having found the associated directory entry, we release all file space, so that it can be reused by other files, and erase the directory entry.

- Truncating a file. The user may want to erase the contents of a file but keep its attributes. Rather than forcing the user to delete the file and then recreate it, this function allows all attributes to remain unchanged—except for file length—but lets the file be reset to length zero and its file space released.

Most of the file operations mentioned involve searching the directory for the entry associated with the named file. To avoid this constant searching, many systems require that an **open()** system call be made before a file is first used. The operating system keeps a table, called the open-file table, containing information about all open files. When a file operation is requested, the file is specified via an index into this table, so no searching is required. When the file is no longer being actively used, it is closed by the process, and the operating system removes its entry from the open-file table. **create()** and **delete()** are system calls that work with closed rather than open files.

In summary, several pieces of information are associated with an open file.

- **File pointer**. File pointer - records the current position in the file, for the next read or write access.
- **File-open count** - How many times has the current file been opened (simultaneously by different processes) and not yet closed?
    - Multiple processes may have opened a file, and the system must wait for the last file to close before removing the open-file table entry.
    - As files are closed, the operating system must free its open-file table entries to save space in the table.
    - When this counter reaches zero the file entry can be removed from the table.

- **Disk location of the file**. Most file operations require the system to modify data within the file. The information needed to locate the file on disk is kept in memory so that the system does not have to read it from disk for each operation.

- **Access rights**. Each process opens a file in an access mode. This information is stored on the per-process table so the operating system can allow or deny subsequent I/O requests.

4- File types

- When we design a file system—indeed, an entire operating system—we always consider whether the operating system should recognize and support

file types. If an operating system recognizes the type of a file, it can then operate on the file in reasonable ways.

- A common technique for implementing file types is to include the type as part of the file name. The name is split into two parts—a name and an extension, usually separated by a period (Figure 1). In this way, the user and the operating system can tell from the name alone what the type of a file is. Most operating systems allow users to specify a file name as a sequence of characters followed by a period and terminated by an extension made up of additional characters. Examples include resume.docx, server.c, and ReaderThread.cpp.

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

Figure 1: Common file types

5- Access methods

Files store information. When it is used, this information must be accessed and read into computer memory. The information in the file can be accessed in several ways.

A-Sequential Access

The simplest access method is sequential access. Information in the file is processed in order, one record after the other.

A read operation—read next()—reads the next portion of the file and automatically advances a file pointer, which tracks the I/O location. Similarly, the write operation—write next()—appends to the end of the file and advances to the end of the newly written material (the new end of file).

Sequential access, which is depicted in Figure , is based on a tape model of a file and works as well on sequential-access devices.

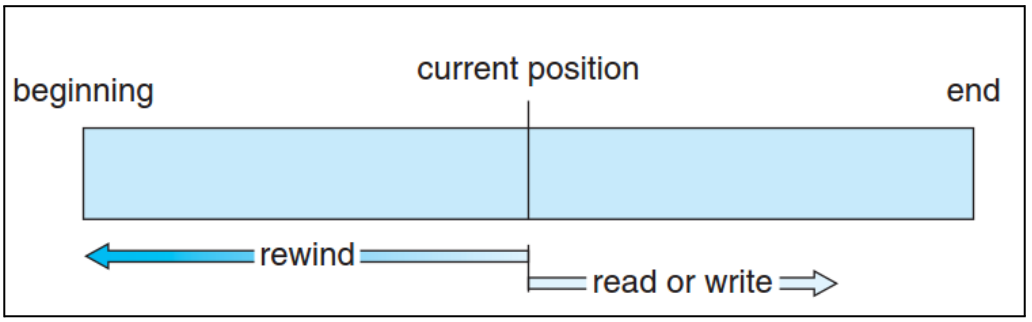


Figure2 : Sequential-access file

B-Direct Access

Another method is direct access (or relative access). Here, a file is made up of fixed-length logical records that allow programs to read and write records rapidly in no particular order.

For direct access, the file is viewed as a numbered sequence of blocks or records. Thus, we may read block 14, then read block 53, and then write block 7. There are no restrictions on the order of reading or writing for a direct-access file.

Direct-access files are of great use for immediate access to large amounts of information. Databases are often of this type. When a query concerning a particular subject arrives, we compute which block contains the answer and then read that block directly to provide the desired information.

For the direct-access method, the file operations must be modified to include the block number as a parameter. Thus, we have read(n), where n is the block number, rather than read next(), and write(n) rather than write next().

6- Directory and Disk Structure

- Directory is a collection of nodes containing information about files.such as name, location, size and type for all files in that directory.
- Computer systems may have zero or more file systems, and the file systems may be of varying types.
- The organization of directories must allow us to insert entries, to delete entries, to search for a named entry, and to list all the entries in the directory.
- Using directories are satisfy:
  - Efficiency – locating a file quickly
  - Naming – convenient to users
    - Two users can have same name for different files
    - The same file can have several different names
  - Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, ...)
- In this section, we examine several schemes for defining the logical structure of the directory system:

1. **Single-Level Directory**
   - Is the simplest directory structure. All files are contained in the same directory.
   - A single directory for all users cause:
     - Naming problem
     - Grouping problem

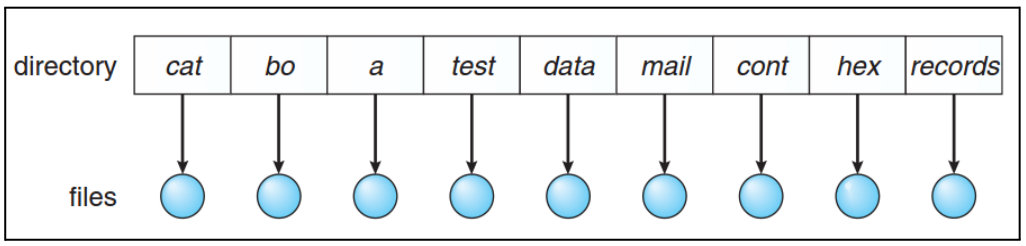


Figure 3 Single-Level Directory

2. **Two-Level Directory**

- The standard solution is to create a separate directory for each user.
- In this structure, each user has his own user file directory (UFD).
- The UFDs have similar structures, but each lists only the files of a single user.
- When a user refers to a particular file, only his own UFD is searched. Thus, different users may have files with the same name, as long as all the file names within each UFD are unique.
- This structure have the following properties:

- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

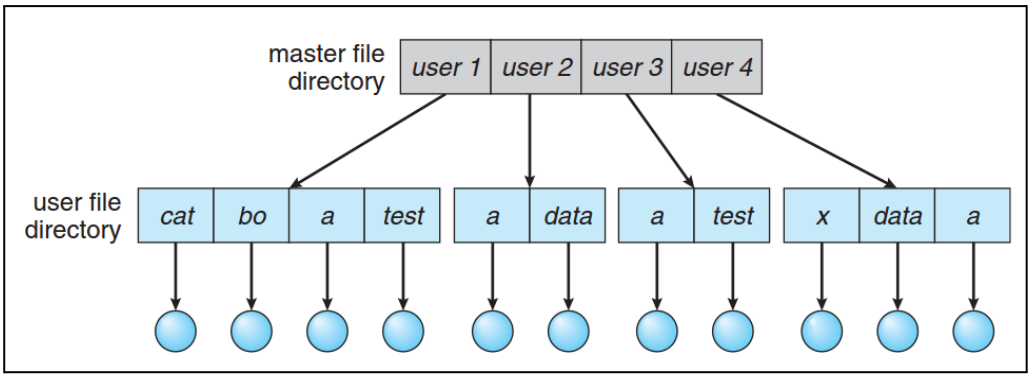


Figure 4 Two-level directory structure

3. Tree-Structured Directories

- The natural generalization is to extend the directory structure to a tree.
- The tree has a root directory, and every file in the system has a unique path name.
- Path names can be of two types: **absolute** and **relative**. An absolute path name begins at the root and follows a path down to the specified file, giving the directory names on the path. A relative path name defines a path from the current directory. For example, in the tree-structured file system of Figure 4 if the current directory is root/spell/mail, then the relative path name prt/first refers to the same file as does the absolute path name root/spell/mail/prt/first.

- This structure have the following properties:
  - Efficient searching
  - Grouping Capability
  - Current directory (working directory)
    cd /spell/mail/prog

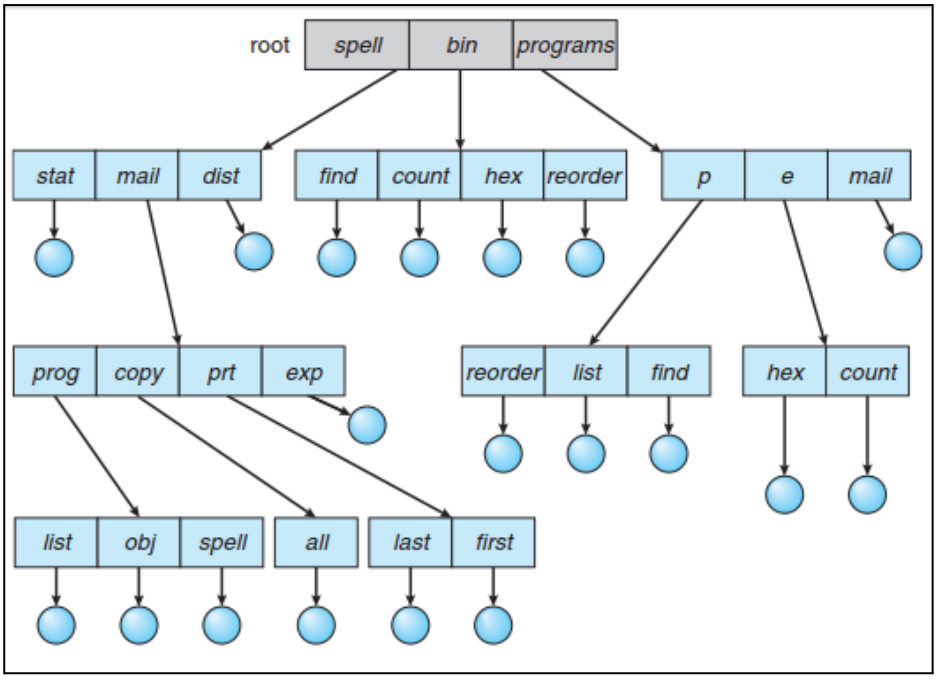


Figure 5 : Tree-structured directory structure.

4. Acyclic-Graph Directories

- This structure allows the shared files or subdirectory with users.

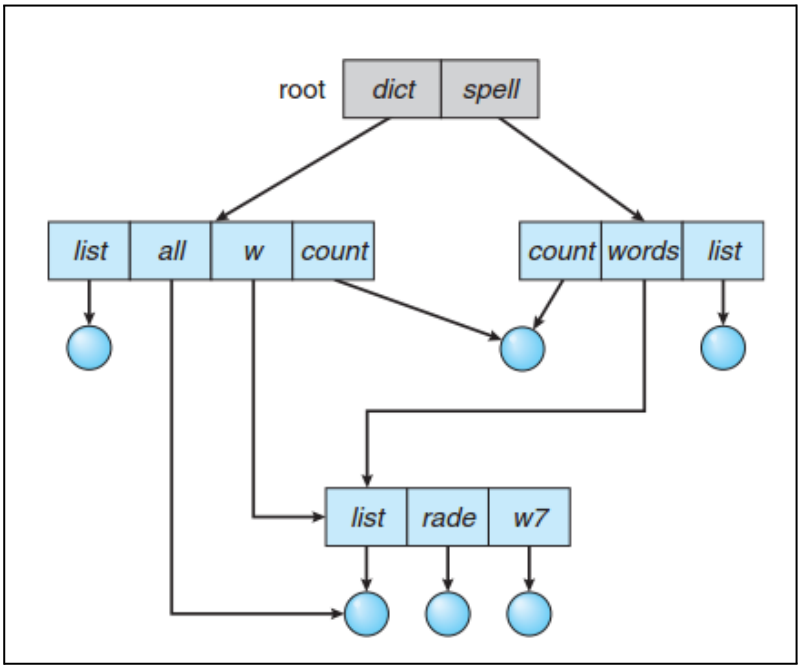


Figure 6: Acyclic-graph directory structure

8- File-System Mounting

9- File sharing