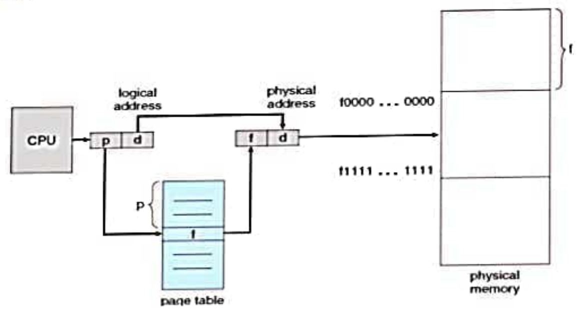
17

2. Non-contiguous allocation-Paging and Segmentation

Paging
A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.
Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes).The size of the process is measured in the number of pages. Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.

Paging Hardware



Address Translation
Page address is called **logical address** and represented by **page number** and the **offset**.

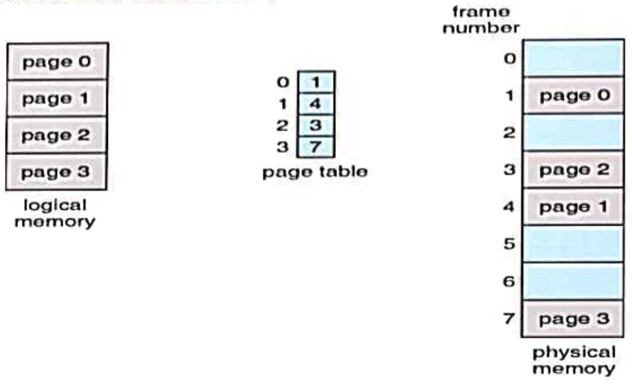
Logical Address = Page number + page offset

Frame address is called **physical address** and represented by **a frame number** and the **offset**.

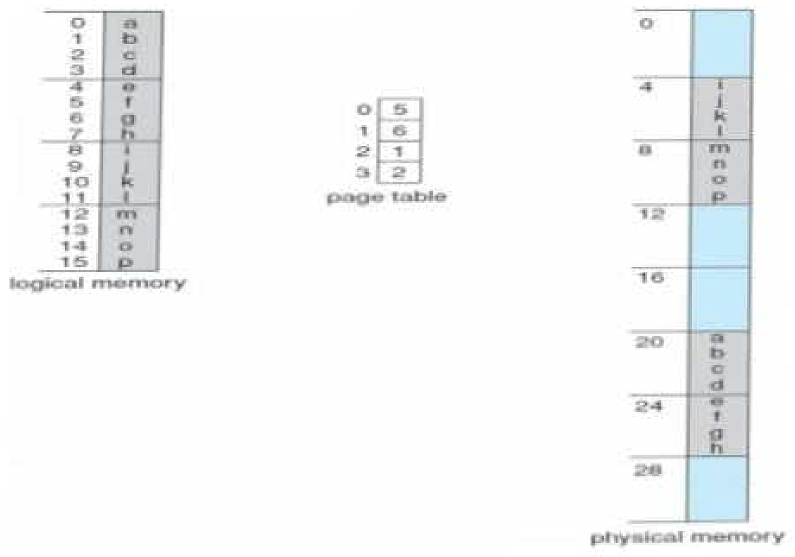
Physical Address = Frame number + page offset

A data structure called **page map table** is used to keep track of the relation between a pageof a process to a frame in physical memory.

Paging Model of Logical and Physical Memory



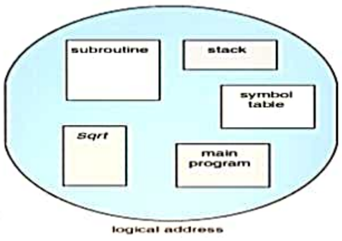
Paging Example



Segmentation

- Memory-management scheme that supports user view of memory A program is acollection of segments
  A segment is a logical unit such as:
  main program
  Procedure
  function method
  object

local variables, global variables
common block
stack
symbol table
arrays

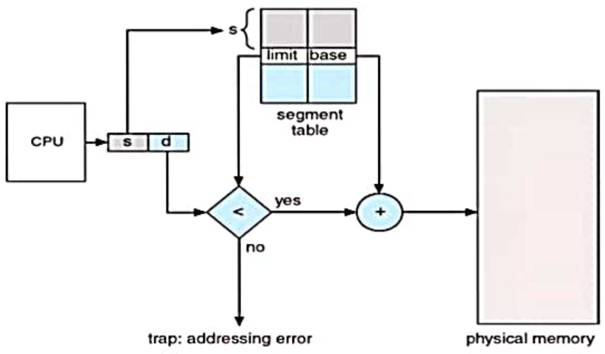


User's View of a Program

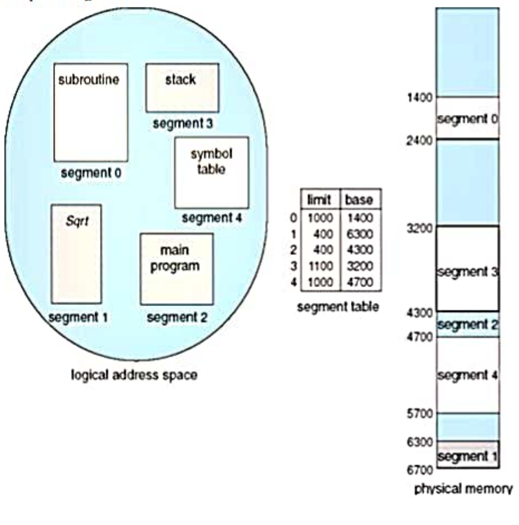
Segmentation Architecture

- Logical address consists of a two tuple o <segment-number, offset>,
- **Segment table** – maps two-dimensional physical address; **base** – contains the starting physical address where the segments reside in memory**limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory **Segment-table length register (STLR)** indicates number of segments used by a program;segment number **s** is legal if **s** < **STLR**
- Protection With each entry in segment table associate:
- validation bit = 0 Þ illegal segmentread/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem A segmentation example is shown in the following diagram

Segmentation Hardware



Example of Segmentation



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

segment table

2a

Virtual Memory

Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as large virtual memory is provided for user programs when a very small physical memory is there.

In real scenarios, most processes never need all their pages at once, for following reasons :

- Error handling code is not needed unless that specific error occurs, some of whichare quite rare.
- Arrays are often over-sized for worst-case scenarios, and only a small fraction ofthearrays are actually used in practice.
- Certain features of certain programs are rarely used.

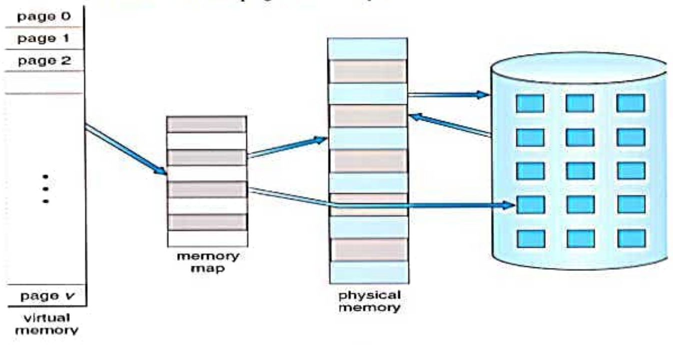


Fig. Diagram showing virtual memory that is larger than physical memory.

Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Benefits of having Virtual Memory :

1. Large programs can be written, as virtual space available is huge compared tophysical memory.

2. Less I/O required, leads to faster and easy swapping of processes.
3. More physical memory available, as programs are stored on virtual memory, so theyoccupy very less space on actual physical memory.

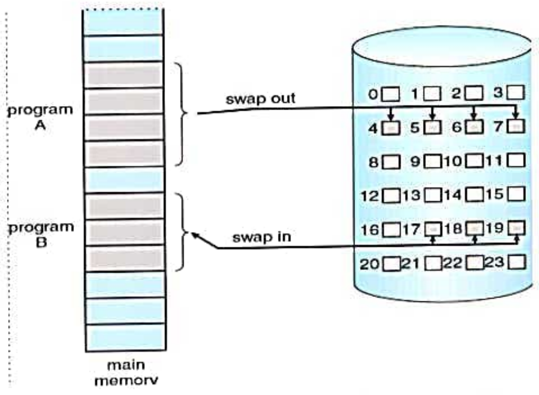
Demand Paging

A demand paging is similar to a paging system with swapping(Fig 5.2). When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory.

When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed.

Hardware support is required to distinguish between those pages that are in memory and those pages that are on the disk using the valid-invalid bit scheme. Where valid and invalid pages can be checked checking the bit and marking a page will have no effect if the process never attempts to access the pages. While the process executes and accesses pages that are memory resident, execution proceeds normally.

Fig. Transfer of a paged memory to continuous disk space



Access to a page marked invalid causes a page-fault trap. This trap is the result of the operating system's failure to bring the desired page into memory.

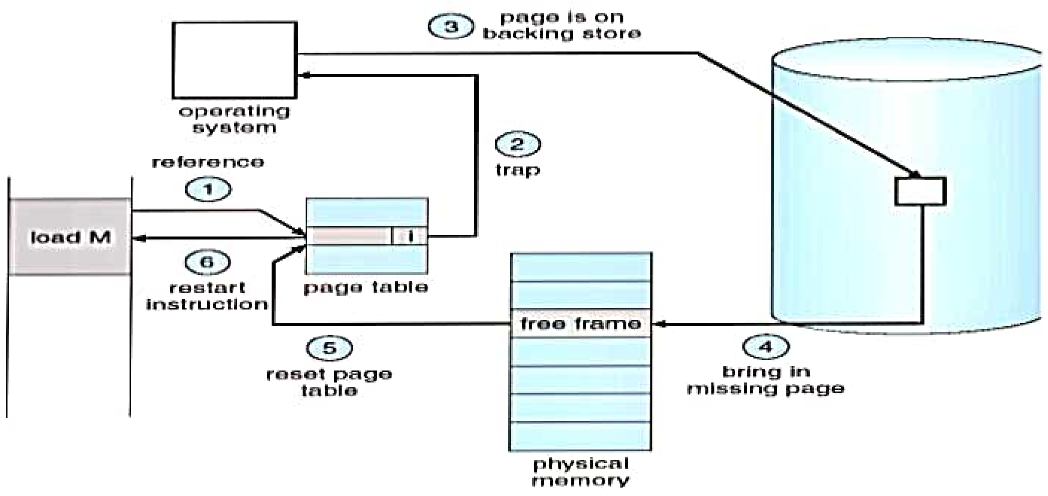
Initially only those pages are loaded which will be required the process immediately.

The pages that are not moved into the memory are marked as invalid in the page table. For

an invalid entry the rest of the table is empty. In case of pages that are loaded in the memory, they are marked as valid along with the information about where to find the swapped out page.
When the process requires any of the page that is not loaded into the memory, a page fault trap is triggered and following steps are followed,

1. The memory address which is requested by the process is first checked, to verify the request made by the process.
2. If its found to be invalid, the process is terminated.
3. In case the request by the process is valid, a free frame is located, possibly from a free-frame list, where the required page will be moved.
4. A new operation is scheduled to move the necessary page from disk to the specified memory location. (This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime.)
5. When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to valid.

Fig. Steps in handling a page fault



6. The instruction that caused the page fault must now be restarted from the beginning. There are cases when no pages are loaded into the memory initially, pages are only loadedwhen demanded by the process by generating page faults. This is called **Pure Demand Paging.**

The only major issue with Demand Paging is, after a new page is loaded, the process starts execution from the beginning. It is not a big issue for small programs, but for larger programsit affects performance drastically.

What is dirty bit?

When a bit is modified by the CPU and not written back to the storage, it is called as a dirtybit. This bit is present in the memory cache or the virtual storage space.

Advantages of Demand Paging:

1. Large virtual memory.
2. More efficient use of memory.
3. Unconstrained multiprogramming. There is no limit on degree of multiprogramming.

Disadvantages of Demand Paging:

1. Number of tables and amount of processor over head for handling page interrupts are greater than inthe case of the simple paged management techniques.
2. due to the lack of an explicit constraints on a jobs address space size.

2b **Page Fault**

- What if the process refers to (i.e., tries to access) a page not in-memory ?
  - The [first] reference (i.e., address) to that **invalid** page will trap to operating system and causes a **page fault**

- **Procedure for handling a page fault**

1. OS checks an internal table to see if reference is valid or invalid memory access
2. If
   - Invalid reference ⇒ abort the process
     - address is not in logical address space of process
   - Just not in memory ⇒ page in the referred page from the disk
     - logical address is valid but page is simply not in-memory
3. Read the referred page into this allocated frame via scheduled disk operation
4. Update both internal table and page-table by setting validation bit = **v**
5. Restart the instruction that caused the page fault and resume process execution

Internal fragmentation and External fragmentation.

Internal fragmentation

When memory is free internally, that is inside a process but it cannot be used, we call that fragment as internal fragment. For example say a hole of size 18464 bytes is available. Let the size of the process be 18462. If the hole is allocated to this process, then two bytes are left which is not used. These two bytes which cannot be used forms the internal fragmentation. The worst part of it is that the overhead to maintain these two bytes is more than two bytes.

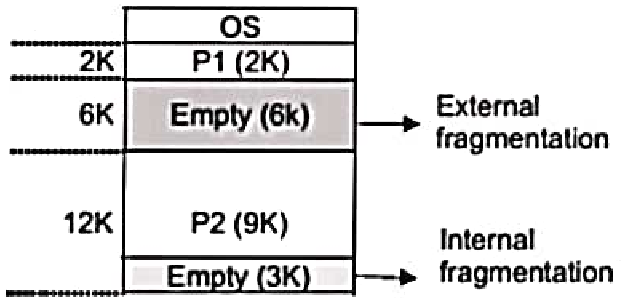
External fragmentation

All the three dynamic storage allocation methods discussed above suffer external fragmentation. When the total memory space that is got by adding the scattered holes is sufficient to satisfy a request but it is not available contiguously, then this type of fragmentation is called external fragmentation.

The solution to this kind of external fragmentation is compaction. **Compaction** is a method by which all free memory that are scattered are placed together in one large memory block. It is to be noted that compaction cannot be done if relocation is done at compile time or assembly time. It is possible only if dynamic relocation is done, that is relocation at execution time.

One more solution to external fragmentation is to have the logical address space and physical address space to be non contiguous. Paging and Segmentation are popular non contiguous allocation methods.

Example for internal and external fragmentation



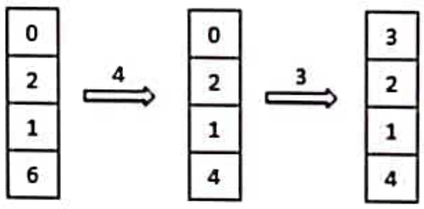
Optimal Page algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.

- Replace the page that will not be used for the longest period of time. Use the timewhen a page is to be used.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x



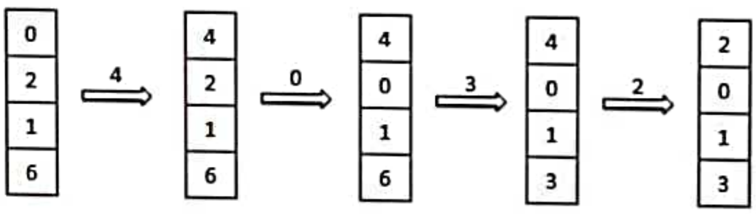
Fault Rate = 6 / 12 = 0.50

Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the onewhich will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x

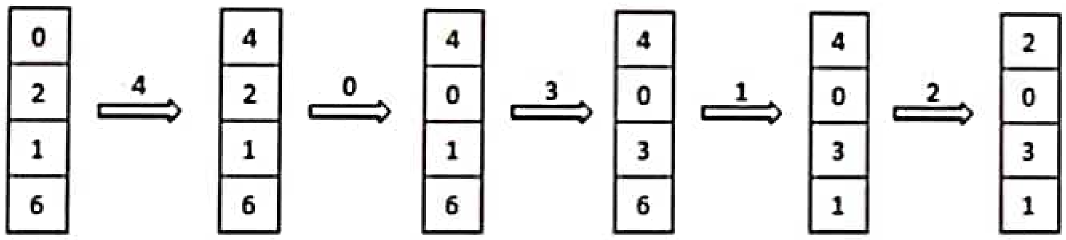


Fault Rate = 8 / 12 = 0.67

- 4 If page size is 100, then the reference string is1,2,6,12,0,0 FirstInFirstOut(FIFO)algorithm
- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages atthe head.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses :x x x x x x x x x



Fault Rate = 9 / 12 = 0.75

Optimal Page algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.

6 1. **Contiguous Allocation-MVT,MFT**

Contiguous memory allocation is one of the efficient ways of allocating main memory to the processes. The memory is divided into two partitions. One for the Operating System and another for the user processes. Operating System is placed in low or high memory depending on the interrupt vector placed. In contiguous memory allocation each process is contained ina single contiguous section of memory.

Memory protection

Memory protection is required to protect Operating System from the user processes and userprocesses from one another. A relocation register contains the value of the smallest physical address for example say 100040. The limit register contains the range of logical address for example say 74600. Each logical address must be less than limit register. If a logical addressis greater than the limit register, then there is an addressing error and it is trapped. The limit register hence offers memory protection.

The MMU, that is, Memory Management Unit maps the logical address dynamically, that is at run time, by adding the logical address to the value in relocation register. This added valueis the physical memory address which is sent to the memory.

The CPU scheduler selects a process for execution and a dispatcher loads the limit and relocation registers with correct values. The advantage of relocation register is that it providesan efficient way to allow the Operating System size to change dynamically.

Memory allocation- **[Refer diagrams for MVT and MFT]**

There are two methods namely,
1. **multiple partition method-** MVT stands for Multiprogrammingwith a Variable number of Tasks
2. **multiple fixed partition method -** MFT stands for Multiprogramming with a Fixed number of Tasks.

In multiple partition method, the memory is divided into several fixed size partitions. One process occupies each partition. This scheme is rarely used nowadays. Degree of multiprogramming depends on the number of partitions. Degree of multiprogramming is the number of programs that are in the main memory. The CPU is never left idle in multiprogramming. This was used by IBM OS/360 called MFT. MFT stands for Multiprogramming with a Fixed number of Tasks.

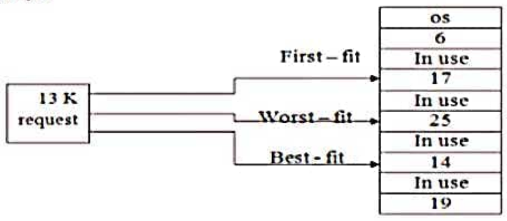
Generalization of fixed partition scheme is used in MVT. MVT stands for Multiprogrammingwith a Variable number of Tasks. The Operating System keeps track of which parts of memory are available and which is occupied. This is done with the help of a table that is maintained by the Operating System. Initially the whole of the available memory is treated as one large block of memory called a **hole**. The programs that enter a system are maintained in an input queue. From the hole, blocks of main memory are allocated to the programs in the input queue. If the hole is large, then it is split into two, and one half is allocated to the arriving process and the other half is returned. As and when memory is allocated, a set of holes in scattered. If holes are adjacent, they can be merged.
Now there comes a general dynamic storage allocation problem.
The following are the solutions to the **dynamic storage allocation problem**.

- First fit: The first hole that is large enough is allocated. Searching for the holes starts from the beginning of the set of holes or from where the previous first fit search ended.

- Best fit: The smallest hole that is big enough to accommodate the incoming process is allocated. If the available holes are ordered, then the searching can be reduced.

- Worst fit: The largest of the available holes is allocated.

Example:

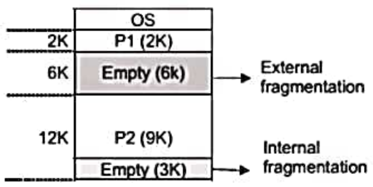


First and best fits decrease time and storage utilization. First fit is generally faster.Fragmentation
The disadvantage of contiguous memory allocation is **fragmentation**. There are twotypes of fragmentation, namely, internal fragmentation and External fragmentation.
Internal fragmentation

When memory is free internally, that is inside a process but it cannot be used, we call that fragment as internal fragment. For example say a hole of size 18464 bytes is available. Let the size of the process be 18462. If the hole is allocated to this process, then two bytes are left which is not used. These two bytes which cannot be used forms the internal fragmentation. The worst part of it is that the overhead to maintain these two bytes is more than two bytes.
External fragmentation
All the three dynamic storage allocation methods discussed above suffer external fragmentation. When the total memory space that is got by adding the scattered holes is sufficient to satisfy a request but it is not available contiguously, then this type of fragmentation is called external fragmentation.

The solution to this kind of external fragmentation is compaction. **Compaction** is a method by which all free memory that are scattered are placed together in one large memory block.It is to be noted that compaction cannot be done if relocation is done at compile time or assembly time. It is possible only if dynamic relocation is done, that is relocation at execution time.

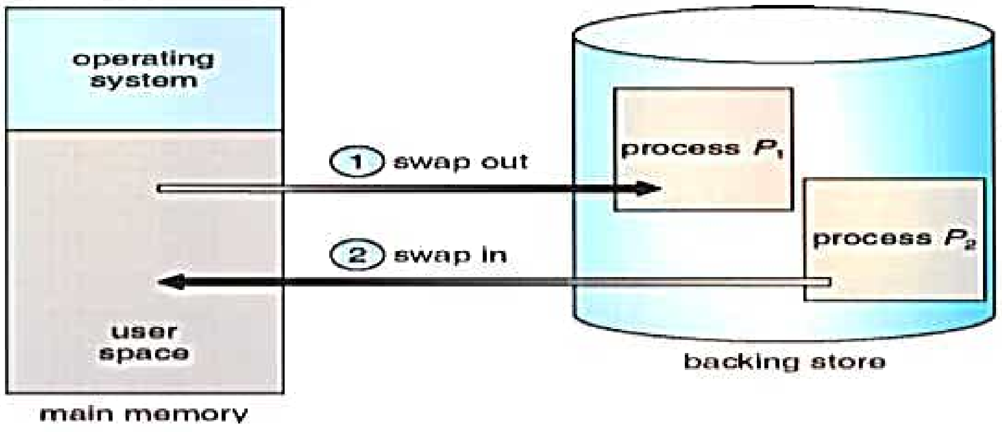
One more solution to external fragmentation is to have the logical address space and physical address space to be non contiguous. Paging and Segmentation are popular non contiguous allocation methods.
Example for internal and external fragmentation



8a **Swapping**

A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution **Backing store** – fast disk large enough to accommodate copies of allmemory images for all users; must provide direct access to these memory images **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped and Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Schematic View of Swapping



Main memory usually into two partitions:

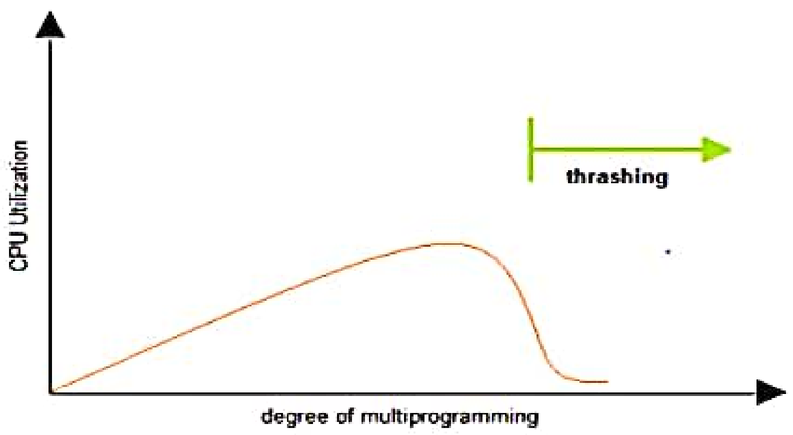
- Resident operating system, usually held in low memory with interrupt vector
- User processes then held in high memory and Relocation registers used to protect user processes from eachother, and from changing operating-system code and data Base register contains value of smallest physical address
-
- Limit register contains range of logical addresses – each logical address must be less than the limitregister
- MMU maps logical address dynamically

Thrashing

If the number of frames allocated to a low-priority process falls below the minimum number required by the computer architecture, we must suspend that process' execution. We should then page out its remaining pages, freeing all its allocated frames. This provision introduces a swap-in, swap-out level of intermediate CPU scheduling.

In fact, look at any process that does not have "enough" frames. Although it is technically possible to reduce the number of allocated frames to the minimum, there is some (larger) number of pages in active use. If the process does not have this number of frames, it will quickly page fault. At this point, it must replace some page. However, since all its pages are in active use, it must replace a page that will be needed again right away. Consequently, it quickly faults again, and again, and again. The process continues to fault, replacing pages for which it then faults and brings back in right away.

This high paging activity is called **thrashing**. A process is thrashing if it is spending more time paging than executing.



8b**Worst-Fit Memory Allocation** :

In this allocation technique, the process traverses the whole memory and always search for the largest hole/partition, and then the process is placed in that hole/partition. It is a slow process because it has to traverse the entire memory to search the largest hole.

Here is an example to understand Worst Fit-Allocation –

Process Number	Process Size
P1	30K
P2	100K
P3	45K

MEMORY LOCATION	MEMORY BLOCK SIZE	PROCESS NUMBER	PROCESS SIZE	STATUS	INTERNAL FRAGMENTATION
12345	50K	P3	45K	Busy	5K
45871	100K	P2	100K	Busy	None
1245	400K	P1	30K	Busy	370K
TOTAL AVAILABLE:	550K	TOTAL USED:	175K		375K

Here Process P1=30K is allocated with the Worst Fit-Allocation technique, so it traverses the entire memory and selects memory size 400K which is the largest, and hence there is an internal fragmentation of 370K which is very large and so many other processes can also utilize this leftover space.

First-Fit Memory Allocation: This method keeps the free/busy list of jobs organized by memory location, low-ordered to high-ordered memory. In this method, first job claims the first available memory with space more than or equal to it's size. The operating system doesn't search for appropriate partition but just allocate the job to the nearest memory partition available with sufficient
size.

Job Number	Memory Requested
J1	20 K
J2	200 K
J3	500 K
J4	50 K

Memory location	Memory block size	Job number	Job size	Status	Internal fragmentation
10567	200 K	J1	20 K	Busy	180 K
30457	30 K			Free	30
300875	700 K	J2	200 K	Busy	500 K
809567	50 K	J4	50 K	Busy	None
Total available :	980 K	Total used :	270 K		710 K

As illustrated above, the system assigns J1 the nearest partition in the memory. As a result, there is no partition with sufficient space is available for J3 and it is placed in the waiting list.The processor ignores if the size of partition allocated to the job is very large as compared to the size of job or not. It just allocates the memory. As a result, a lot of memory is wasted and many jobs may not get space in the memory, and would have to wait for another job to complete.

Best-Fit Memory Allocation:

This method keeps the free/busy list in order by size – smallest to largest. In this method, the operating system first searches the whole of the memory according to the size of the given job and allocates it to the closest-fitting free partition in the memory, making it able to use memory efficiently. Here the jobs are in the order from smallest job to largest job.

Job Number	Memory Requested
J1	20 K
J2	200 K
J3	500 K
J4	50 K

Memory location	Memory block size	Job number	Job size	Status	Internal fragmentation
10567	30 K	J1	20 K	Busy	10 K
30457	50 K	J4	50 K	Busy	None
300875	200 K	J2	200 K	Busy	None
809567	700 K	J3	500 K	Busy	200 K
Total available :	980 K	Total used :	770 K		210 K

As illustrated in above figure, the operating system first search throughout the memory and allocates the job to the minimum possible memory partition, making the memory allocation efficient.

Structure of the Page Table

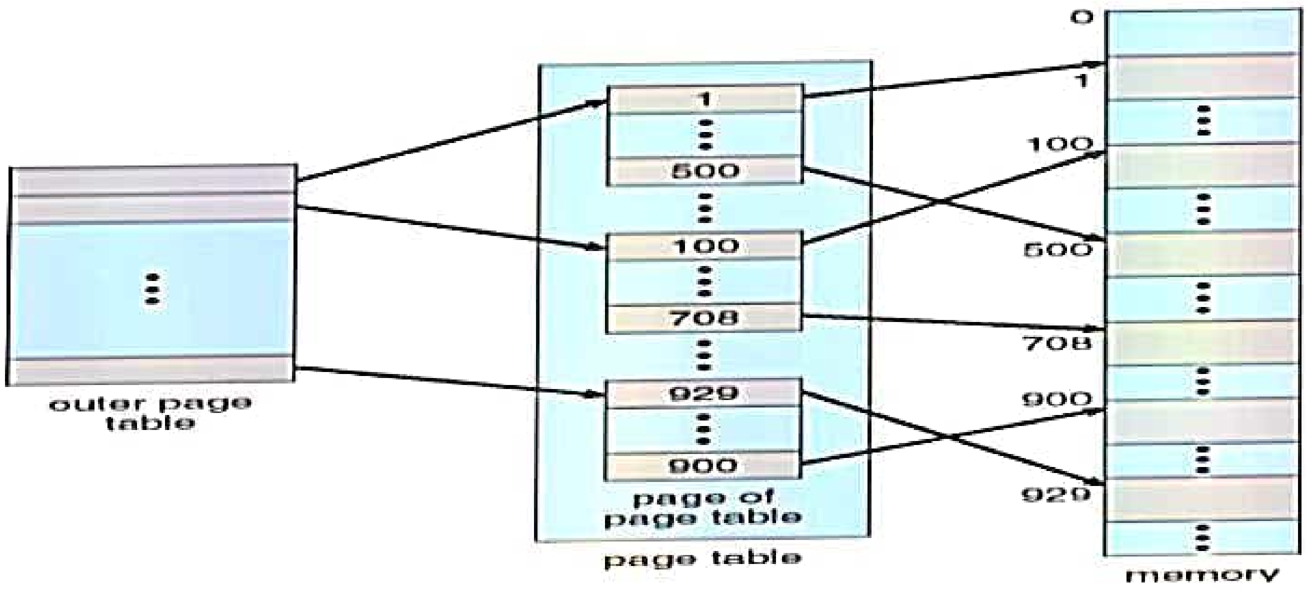
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

Break up the logical address space into multiple page tables

A simple techniqueis a two-level page table

Two-Level Page-Table Scheme

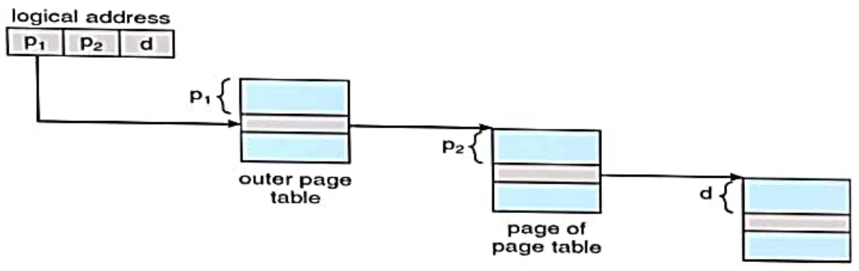


Two-Level Paging Example

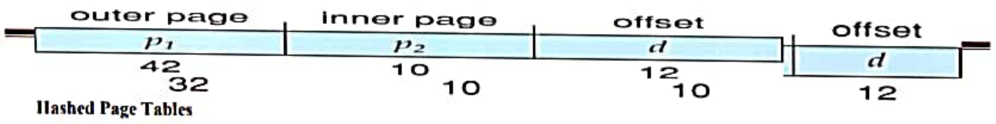
- A logical address (on 32-bit machine with 1K page size) is dividedinto: a page number consisting of 22 bits
- a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
- a 12-bit page number a 10-bit page offsetThus, a logical address is as follows:
- where pi is an index into the outer page table, and p2 is the displacement within the page of theouter page table
-
-

Page number		page offset
p_1 12	p_2 10	d_{10}

Address-Translation Scheme



Three-level Paging Scheme



Hashed Page Tables

- Common in address spaces > 32 bits
- The virtual page number is hashed into a page table
- This page table contains a chain of elements hashing to the same location Virtual page numbers are compared in
- this chain searching fora match
- If a match is found, the corresponding physical frame is extracted

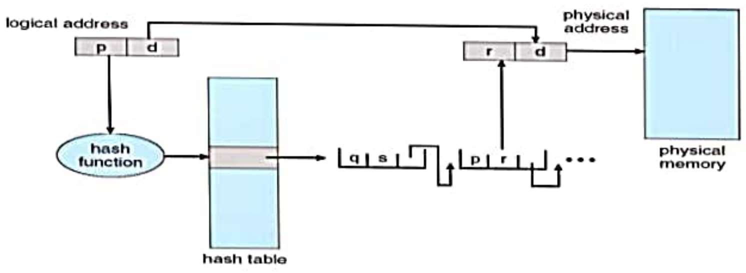
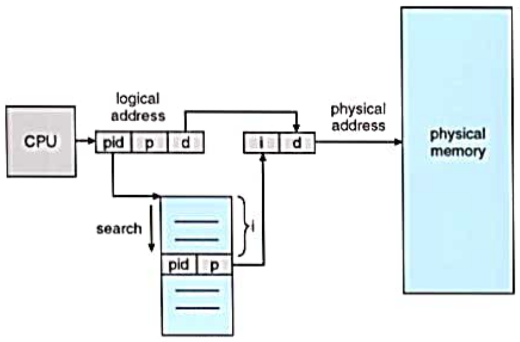


Fig:Hashed Page Table

Inverted Page Table

- One entry for each real page of memory
- Entry consists of the virtual address of the page stored in that real memory location, with informationabout the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the tablewhen a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries

Inverted Page Table Architecture



Advantages an

Here is a list of advantages and disadvantages of paging –

- Paging reduces external fragmentation, but still suffers from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory managementtechnique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system havingsmall RAM.

10 In Operating System (Memory Management Technique: Paging), for each process page table will be created, which will contain a Page Table Entry (PTE). This PTE will contain information like frame number (The address of the main memory where we want to refer), and some other useful bits (e.g., valid/invalid bit, dirty bit, protection bit, etc). This page table entry (PTE) will tell where in the main memory the actual page is residing.

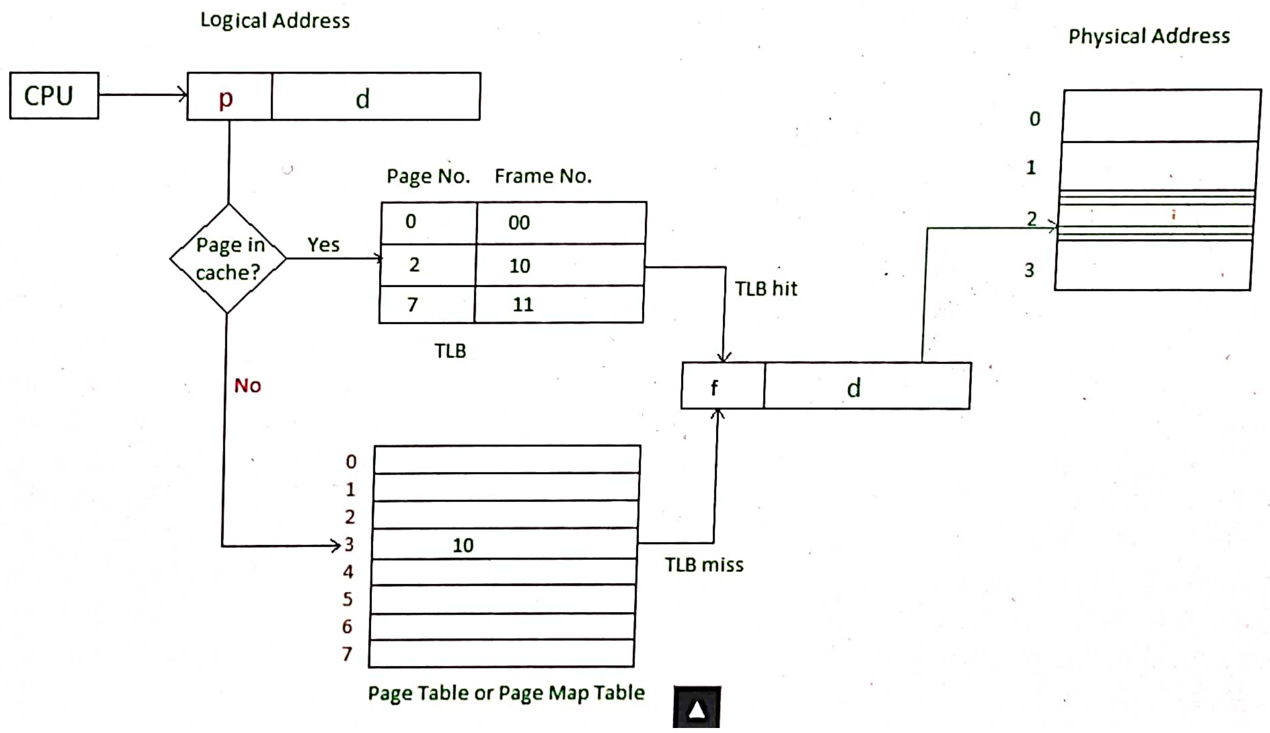
Now the question is where to place the page table, such that overall access time (or reference time) will be less. The problem initially was to fast access the main memory content based on the address generated by the CPU (i.e. logical/virtual address). Initially, some people thought of using registers to store page tables, as they are high-speed memory so access time will be less.

The idea used here is, to place the page table entries in registers, for each request generated from the CPU (virtual address), it will be matched to the appropriate page number of the page table, which will now tell where in the main memory that corresponding page resides. Everything seems right here, but the problem is registered size is small (in practice, it can accommodate a maximum of 0.5k to 1k page table entries) and the process size may be big hence the required page table will also be big (let's say this page table contains 1M entries), so registers may not hold all the PTE's of the Page table. So this is not a practical approach.

The entire page table was kept in the main memory to overcome this size issue. but the problem here is two main memory references are required:

1. To find the frame number
2. To go to the address specified by frame number

To overcome this problem a high-speed cache is set up for page table entries called a Translation Lookaside Buffer (TLB). Translation Lookaside Buffer (TLB) is a special cache used to keep track of recently used transactions. TLB contains page table entries that have been most recently used. Given a virtual address, the processor examines the TLB if a page table entry is present (TLB hit), the frame number is retrieved and the real address is formed. If a page table entry is not found in the TLB (TLB miss), the page number is used as an index while processing the page table. TLB first checks if the page is already in main memory, if not in main memory a page fault is issued then the TLB is updated to include the new page entry.



Steps in TLB hit

1. CPU generates a virtual (logical) address.
2. It is checked in TLB (present).
3. The corresponding frame number is retrieved, which now tells where the main memory page lies.

Steps in TLB miss

1. CPU generates a virtual (logical) address.
2. It is checked in TLB (not present).
3. Now the page number is matched to the page table residing in the main memory (assuming the page table contains all PTE).
4. The corresponding frame number is retrieved, which now tells where the main memory page lies.
5. The TLB is updated with new PTE (if space is not there, one of the replacement techniques comes into the picture i.e either FIFO, LRU or MFU etc).