MEMORY MANAGEMENT

Main Memory Management Strategies

- Memory management is concerned with managing the primary memory.
- Memory consists of array of bytes or words each with its own address.
- Every program to be executed must be in memory. The instruction must be fetched from memory before it is executed.
- In multi-tasking OS memory management is complex, because as processes are swapped in and out of the CPU, their code and data must be swapped in and out of memory.

Basic Hardware

- Main memory, cache and CPU registers in the processors are the only storage spaces that CPU can access directly.
- The program and data must be bought into the memory from the disk, for the process to run. Each process has a separate memory space and must access only this range of legal addresses. Protection of memory is required to ensure correct operation. This prevention is provided by hardware implementation.
- Two registers are used - a base register and a limit register. The base register holds the smallest legal physical memory address; the limit register specifies the size of the range.
- For example, if the base register holds 300040 and limit register is 120900, then the program can legally access all addresses from 300040 through 420940 (inclusive).

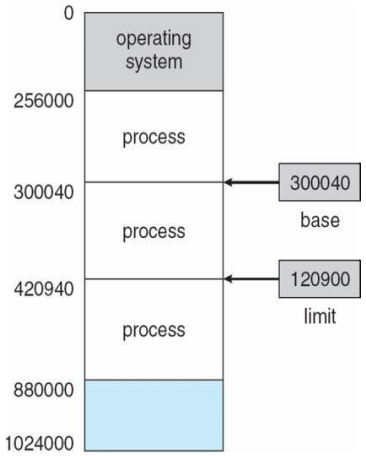


Figure: A base and a limit-register define a logical-address space

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

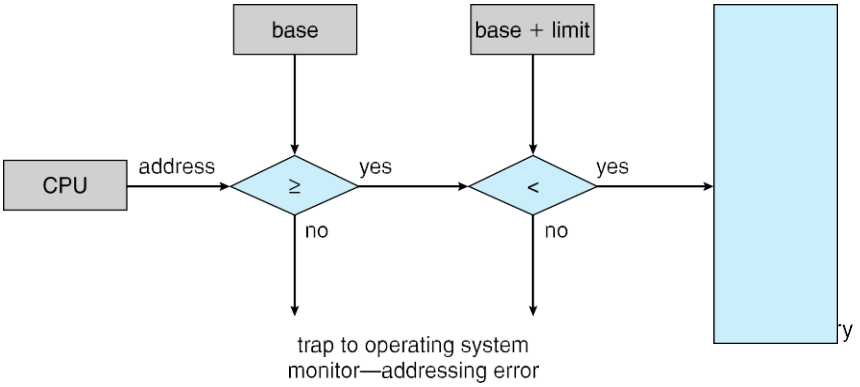


Figure: Hardware address protection with base and limit-registers

- ✓ **Protection of memory space** is accomplished by having the CPU hardware compare every address generated in user mode with the registers.
- ✓ 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** as shown in below **figure.**
- ✓ This prevents a user program from (accidentally or deliberately) modifying the code or data structures of either the operating system or other users.

Address Binding

- User programs typically refer to memory addresses with symbolic names. These symbolic names must be mapped or bound to physical memory addresses.
- Programs are stored on the secondary storage disks as binary executable files.
- When the programs are to be executed, they are brought into the main memory and placed within a process.
- The collection of processes on the disk waiting to enter the main memory forms the **input queue.**
- One of the processes which are to be executed is fetched from the queue and is loaded into main memory.
- During the execution it fetches instruction and data from main memory. After the process terminates it returns the memory space.
- During execution the process will go through several steps as shown in **the figure below.** and in each step the address is represented in different ways.
- In source program the address is symbolic. The compiler **binds** the symbolic address to re-locatable address. The loader will in turn bind this re-locatable address to the absolute address.
- Address binding of instructions to memory-addresses can happen at 3 different stages.
  1. **Compile Time** - If it is known at compile time where a program will reside in physical memory, then absolute code can be generated by the compiler, containing actual physical addresses. However, if the load address changes at some later time, then the program will have to be recompiled.
  2. **Load Time** - If the location at which a program will be loaded is not known at compile time, then the compiler must generate relocatable code, which references addressesrelative to the start of the program. If that starting address changes, then the program must be reloaded but not recompiled.
  3. **Execution Time** - If a program can be moved around in memory during the course of itsexecution, then binding must be delayed until execution time.

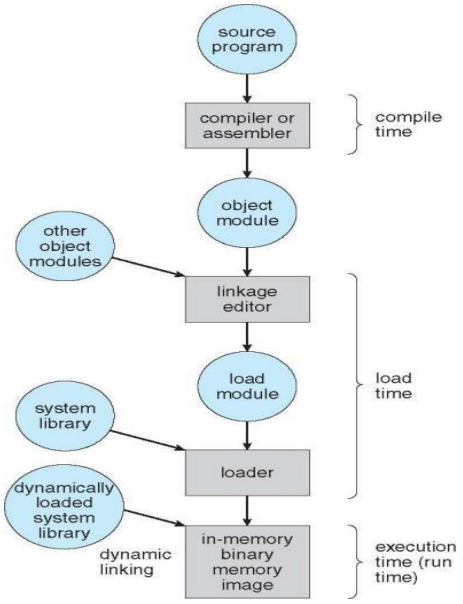


Figure: Multistep processing of a user program

Logical Versus Physical Address Space

- The address generated by the CPU is a logical address, whereas the memory addresswhere programs are actually stored is a physical address.
- The set of all logical addresses used by a program composes the logical address space,and the set of all corresponding physical addresses composes the physical address space.
- The run time mapping of logical to physical addresses is handled by the memory-management unit (MMU).
    - One of the simplest is a modification of the base-register scheme.
    - The base register is termed a relocation register
    - The value in the relocation-register is added to every address generated by auser-process at the time it is sent to memory.
    - The user-program deals with logical-addresses; it never sees the real physical-addresses.

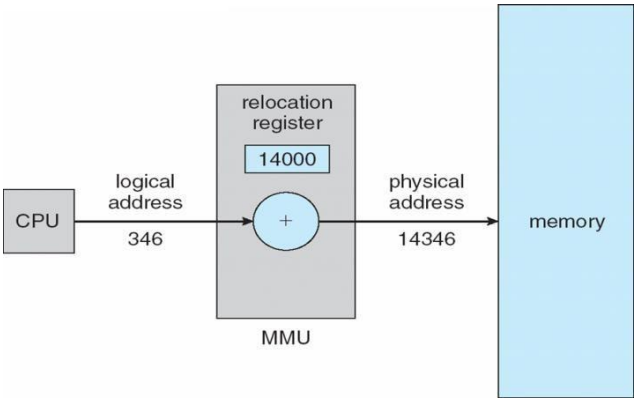


Figure: Dynamic relocation using a relocation-register

- **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**. The user program never sees the real physical addresses.
- The size of the process is thus limited to the size of the physical memory.

Dynamic Loading

- This can be used to obtain better memory-space utilization.
- A routine is not loaded until it is called.

This works as follows:

1. Initially, all routines are kept on disk in a relocatable-load format.
2. Firstly, the main-program is loaded into memory and is executed.
3. When a main-program calls the routine, the main-program first checks to see whether theroutine has been loaded.
4. If routine has been not yet loaded, the loader is called to load desired routine into memory.
5. Finally, control is passed to the newly loaded-routine.

Advantages:

1. An unused routine is never loaded.
2. Useful when large amounts of code are needed to handle infrequently occurring cases.
3. Although the total program-size may be large, the portion that is used (and hence loaded)may be much smaller.
4. Does not require special support from the OS.

Dynamic Linking and Shared Libraries

- With **static linking** library modules get fully included in executable modules, wasting both disk space and main memory usage, because every program that included a certain routine from the library would have to have their own copy of that routine linked into their executable code.
- With **dynamic linking**, however, only a stub is linked into the executable module, containing references to the actual library module linked in at run time.
  - The stub is a small piece of code used to locate the appropriate memory-resident library-routine.
  - This method saves disk space, because the library routines do not need to be fully included in the executable modules, only the stubs.
  - An added benefit of dynamically linked libraries (DLLs, also known as shared libraries or shared objects on UNIX systems) involves easy upgrades and updates.

Shared libraries

- A library may be replaced by a new version, and all programs that reference the librarywill automatically use the new one.
- Version info. is included in both program & library so that programs won't accidentally execute incompatible versions.

Swapping

- A process must be loaded into memory in order to execute.
- If there is not enough memory available to keep all running processes in memory at the same time, then some processes that are not currently using the CPU may have their memory swapped out to a fast local disk called the ***backing store***.
- *Swapping is the process of moving a process from memory to backing store and moving another process from backing store to memory*. Swapping is a very slow process compared to other operations.
- A variant of swapping policy is used for priority-based scheduling algorithms. If a higher-priority process arrives and wants service, the memory manager can swap out the lower-priority process and then load and execute the higher-priority process. When the higher-priority process finishes, the lower-priority process can be swapped back in and continued. This variant of swapping is ***called roll out, roll in.***
- Normally the process which is swapped out will be swapped back to the same memory space that is occupied previously and this depends upon address binding.
- The system maintains a **ready queue** consisting of all the processes whose memory images are on the backing store or in memory and are ready to run.

Swapping depends upon address-binding:

- If binding is done at load-time, then process cannot be easily moved to a different location.
- If binding is done at execution-time, then a process can be swapped into a different 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 theamount 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.Two solutions:

i) Never swap a process with pending I/O.

ii) Execute I/O operations only into OS buffers.

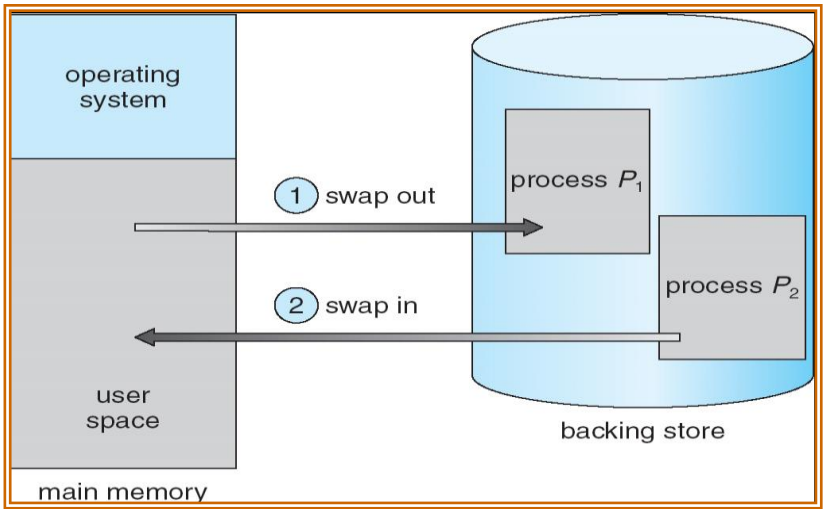


Figure: Swapping of two processes using a disk as a backing store

Example:

Assume that the user process is 10 MB in size and the backing store is a standard hard disk witha transfer rate of 40 MB per second.

The actual transfer of the 10-MB process to or from main memory takes

10000 KB/40000 KB per second = 1/4 second

= 250 milliseconds.

Assuming that no head seeks are necessary, and assuming an average latency of 8 milliseconds, the swap time is 258 milliseconds. Since we must both swap out and swap in, the total swap time is about 516 milliseconds.

Contiguous Memory Allocation

- The main memory must accommodate both the operating system and the various user processes. Therefore we need to allocate the parts of the main memory in the most efficient way possible.
- Memory is usually divided into 2 partitions: One for the resident OS. One for the user processes.
- Each process is contained in a single contiguous section of memory.

1. Memory Mapping and Protection

- Memory-protection means protecting OS from user-process and protecting user-processes from one another.
- Memory-protection is done using
  - Relocation-register: contains the value of the smallest physical-address.
  - Limit-register: contains the range of logical-addresses.
- Each logical-address must be less than the limit-register.
- The MMU maps the logical-address dynamically by adding the value in the relocation-register. This mapped-address is sent to memory
- When the CPU scheduler selects a process for execution, the dispatcher loads therelocation and limit-registers with the correct values.
- Because every address generated by the CPU is checked against these registers, we canprotect the OS from the running-process.
- The relocation-register scheme provides an effective way to allow the OS size to change dynamically.

- Transient OS code: Code that comes & goes as needed to save memory-space and overhead for unnecessary swapping.

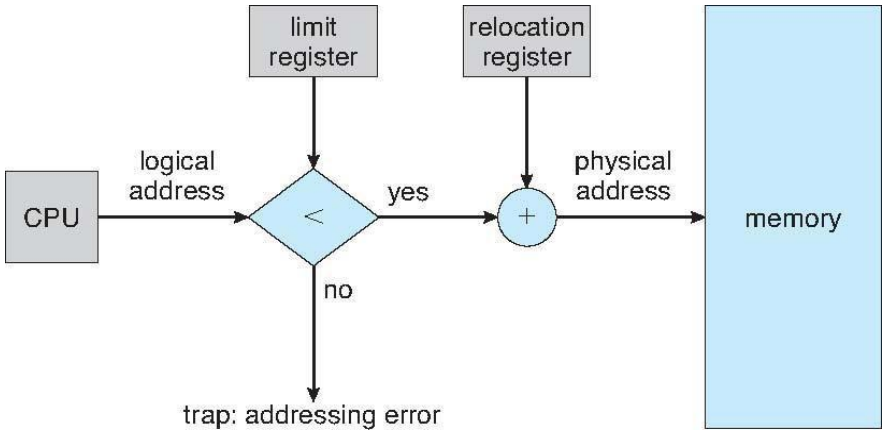


Figure: Hardware support for relocation and limit-registers

2. Memory Allocation

Two types of memory partitioning are:

1. Fixed-sized partitioning
2. Variable-sized partitioning

1. Fixed-sized Partitioning

- The memory is divided into fixed-sized partitions.
- Each partition may contain exactly one process.
- The degree of multiprogramming is bound by the number of partitions.
- When a partition is free, a process is selected from the input queue and loaded into thefree partition.
- When the process terminates, the partition becomes available for another process.

2.Variable-sized Partitioning

- The OS keeps a table indicating which parts of memory are available and which parts are occupied.
- A hole is a block of available memory. Normally, memory contains a set of holes of various sizes.
- Initially, all memory is available for user-processes and considered one large hole.
- When a process arrives, the process is allocated memory from a large hole.
- If we find the hole, we allocate only as much memory as is needed and keep the remaining memory available to satisfy future requests.

Three strategies used to select a free hole from the set of available holes:

1. **First Fit:** Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended.
2. **Best Fit:** Allocate the smallest hole that is big enough. We must search the entire list,unless the list is ordered by size. This strategy produces the smallest leftover hole.
3. **Worst Fit:** Allocate the largest hole. Again, we must search the entire list, unless it issorted by size. This strategy produces the largest leftover hole.

First-fit and best fit are better than worst fit in terms of decreasing time and storage utilization.

a. What are the drawbacks of contiguous memory allocation? Given five memory partitions of 100KB, 500KB, 200KB, 300KB and 600KB (in order), how would each of the first fit, best fit and worst fit algorithms place processes of 212 KB, 417 KB, 112KB and 426 KB (in order)? Which algorithm makes the most efficient use of memory? **(06 Marks)**

1a. Solution:

First Fit

100K	
	212K
500K	112K
200K	
300K	
600K	417K

426 must wait

Best Fit

100K	
500K	417K
200K	112K
300K	212K
600K	426K

Worst Fit

100K	
500K	417K
200K	
300K	212K
600K	112K

426 must wait

The Best Fit is the efficient algorithm.

3. Fragmentation

Two types of memory fragmentation:

1. Internal fragmentation
2. External fragmentation

1. Internal Fragmentation

- The general approach is to break the physical-memory into fixed-sized blocks andallocate memory in units based on block size.
- The allocated-memory to a process may be slightly larger than the requested-memory.
- The difference between requested-memory and allocated-memory is called internal fragmentation i.e. Unused memory that is internal to a partition.

2. External Fragmentation

- External fragmentation occurs when there is enough total memory-space to satisfy a request but the available-spaces are not contiguous. (i.e. storage is fragmented into alarge number of small holes).
- Both the first-fit and best-fit strategies for memory-allocation suffer from external fragmentation.
- Statistical analysis of first-fit reveals that given N allocated blocks, another 0.5 N blocks will be lost to fragmentation. This property is known as the 50-percent rule.

Two solutions to external fragmentation:

- Compaction: The goal is to shuffle the memory-contents to place all free memorytogether in one large hole. Compaction is possible only if relocation is dynamic and doneat execution-time
- Permit the logical-address space of the processes to be non-contiguous. This allows a process to be allocated physical-memory wherever such memory is available. Twotechniques achieve this solution: 1) Paging and 2) Segmentation.

Paging

- 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.

Basic Method of Paging

- The basic method for implementing paging involves breaking physical memory into fixed-sized blocks called **frames** and breaking logical memory into blocks of the same size called **pages**.
- When a process is to be executed, its pages are loaded into any available memory framesfrom the backing store.
- The backing store is divided into fixed-sized blocks that are of the same size as the memory frames.

The hardware support for paging is illustrated in Figure.

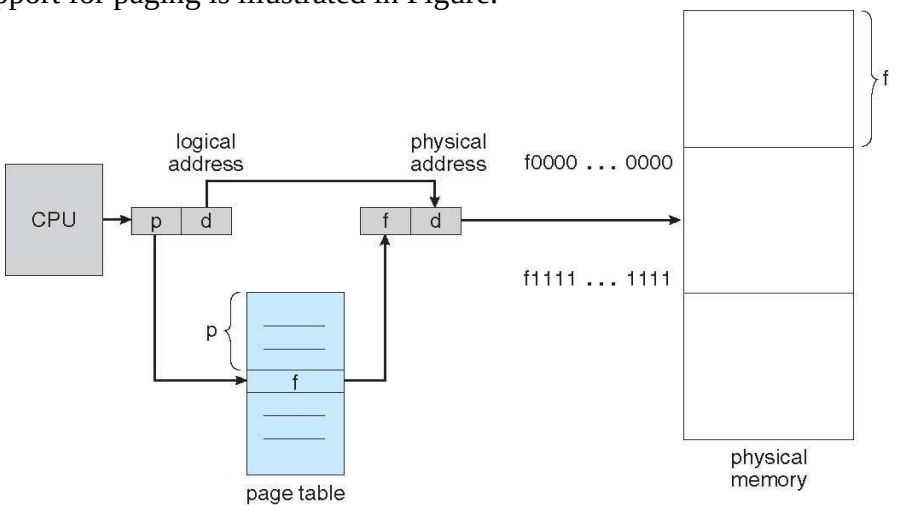


Figure : Paging hardware

- Address generated by CPU is divided into 2 parts (Figure 2):
  1. Page-number (p) is used as an index to the page-table. The page-table contains thebase-address of each page in physical-memory.
  2. Offset (d) is combined with the base-address to define the physical-address. This physical-address is sent to the memory-unit.
- The page table maps the page number to a frame number, to yield a physical address
- The page table maps the page number to a frame number, to yield a physical addresswhich also has two parts: The frame number and the offset within that frame.
- The number of bits in the frame number determines how many frames the system can address, and the number of bits in the offset determines the size of each frame.

The paging model of memory is shown in Figure

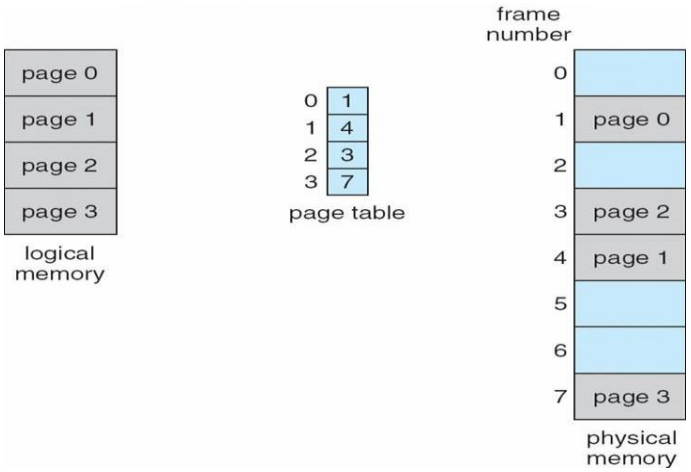


Figure : Paging model of logical and physical memory.

- The page size (like the frame size) is defined by the hardware.
- The size of a page is typically a power of 2, varying between 512 bytes and 16 MB per page, depending on the computer architecture.
- The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset.
- If the size of 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 the page number, and the n low-order bits designate the page offset.

Thus, the logical address is as follows:

page number	page offset
p	d
m -n	n

✓ **Ex:**

✓ Consider a logical address space of 64 pages of 1,024 words each, mapped onto a physical memory of 32 frames.
a. How many bits are there in the logical address?
Solution: 64 pages of 1,024 words each= $2^6*2^{10} = 2^{16}$ hence we need 16 bits
b. How many bits are there in the physical address?
Solution: 32 frames of 1,024 words each= $2^5*2^{10} = 2^{15}$ hence we need 15 bits

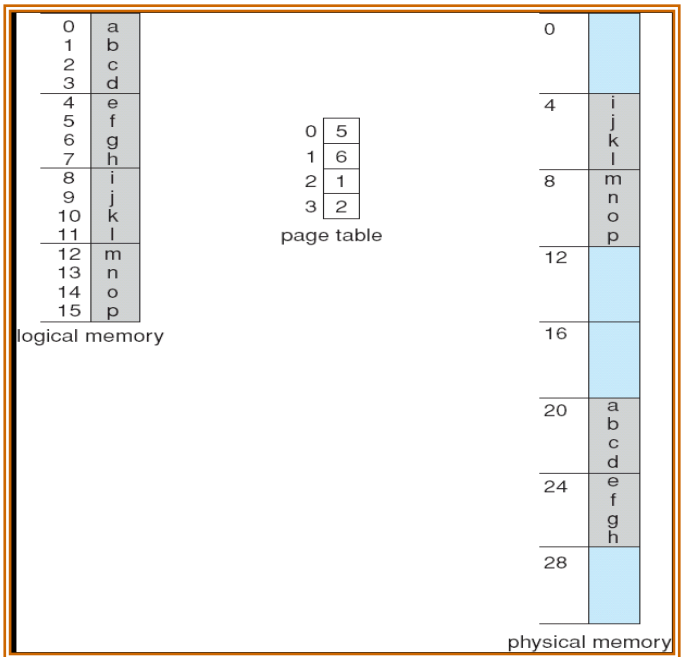
✓ To show how to map logical memory into physical memory, consider a page size of 4 bytes and physical memory of 32 bytes (8 pages) as shown in below **figure**.

a. Logical address 0 is page 0 and offset 0 and Page 0 is in frame 5. The **logical address 0** maps to physical address **[(5*4) + 0]=20.**

b. **Logical address 3** is page 0 and offset 3 and Page 0 is in frame 5. The **logical address 3** maps to **physical address [(5*4) + 3]= 23.**

c. **Logical address 4** is page 1 and offset 0 and page 1 is mapped to frame 6. So logical address 4 maps to **physical address [(6*4) + 0]=24**.

d. **Logical address 13** is page 3 and offset 1 and page 3 is mapped to frame 2. So logical address 13 maps to **physical address [(2*4) + 1]=9.**



- ✓ In paging scheme, there is **no external fragmentation**. Any free frame can be allocated to a process that needs it. But there may exist **internal fragmentation**.
- ✓ If the memory requirements of a process do not happen to coincide with page boundaries, the last frame allocated may not be completely full.
- ✓ **For example**, if page size is 2,048 bytes, a process of 72,766 bytes will need 35 pages plus 1,086 bytes. It will be allocated 36 frames, resulting in **internal fragmentation** of 2,048 - 1,086= **962 bytes**.
- ✓ When a process arrives in the system to be executed, its size expressed in pages is examined. Each page of the process needs one frame. Thus, if the process requires **n** pages, at least **n** frames must be available in memory. If n frames are available, they are allocated to this arriving process.
- ✓ The first page of the process is loaded in to one of the allocated frames, and the frame number is put in the page table for this process. The next page is loaded into another frame and its frame number is put into the page table and so on, as shown in below **figure. (a) before allocation, (b) after allocation**.

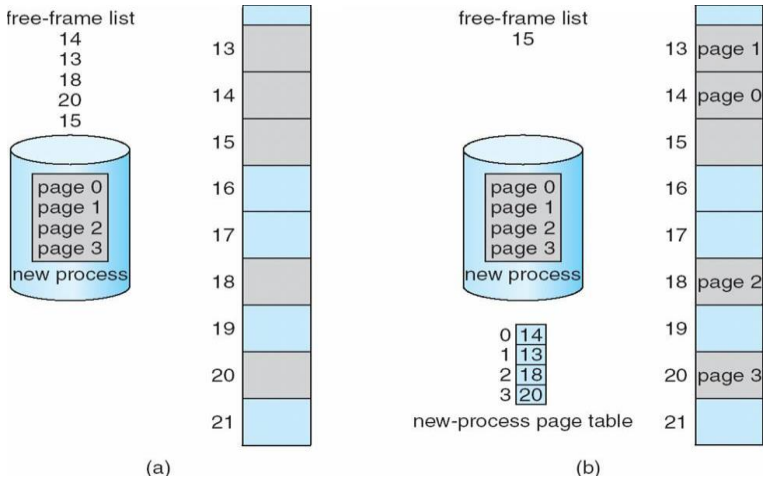


Figure: Free frames (a) before allocation and (b) after allocation.

Hardware Support

Translation Look aside Buffer

- A special, small, fast lookup hardware cache, called a translation look-aside buffer(TLB).
- Each entry in the TLB consists of two parts: a key (or tag) and a value.
- When the associative memory is presented with an item, the item is compared with all keys

simultaneously. If the item is found, the corresponding value field is returned. The search is fast; the hardware, however, is expensive. Typically, the number of entries in a TLB is small, often numbering between 64 and 1,024.

- The TLB contains only a few of the page-table entries.

Working:

- When a logical-address is generated by the CPU, its page-number is presented to theTLB.
- If the page-number is found (TLB hit), its frame-number is immediately available and used to access memory
- If page-number is not in TLB (TLB miss), a memory-reference to page table must be made. The obtained frame-number can be used to access memory (Figure 1)

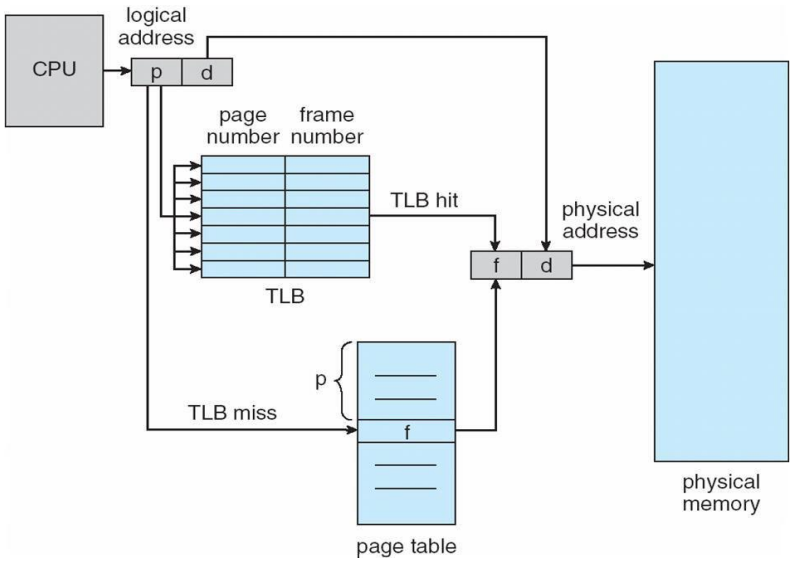


Figure : Paging hardware with TLB

- In addition, we add the page-number and frame-number to the TLB, so that they will befound quickly on the next reference.
- If the TLB is already full of entries, the OS must select one for replacement.
- Percentage of times that a particular page-number is found in the TLB is called hit ratio.

Advantage: Search operation is fast.

Disadvantage: Hardware is expensive.

- Some TLBs have wired down entries that can't be removed.
- Some TLBs store ASID (address-space identifier) in each entry of the TLB that uniquely identify each process and provide address space protection for that process.

✓ **For example,** an **80-percent hit ratio** means that we find the desired page number in the TLB 80 percent of the time. If it takes **20 nanoseconds** to search the TLB and **100 nanoseconds to access memory**, then a mapped-memory access takes **120 nanoseconds when the page number is in the TLB**. If we fail to find the page number in the TLB (20 nanoseconds), then we must first access memory for the page table and frame number (100 nanoseconds) and then access the desired byte in memory (100 nanoseconds), for a total of **220 nanoseconds**. Thus the effective access time is,

Effective Access Time (EAT) = 0.80 x 120 + 0.20 x 220
= **140 nanoseconds.**

In this example, we suffer a **40-percent slowdown** in memory-access time (from 100 to 140 nanoseconds).

✓ For a **98-percent hit ratio** we have
Effective Access Time (EAT) = 0.98 x 120 + 0.02 x 220
= **122 nanoseconds.**

✓ This increased hit rate produces only a **22 percent slowdown** in access time.

Protection

- Memory-protection is achieved by protection-bits for each frame.
- The protection-bits are kept in the page-table.
- One protection-bit can define a page to be read-write or read-only.
- Every reference to memory goes through the page-table to find the correct frame-number.
- Firstly, the physical-address is computed. At the same time, the protection-bit is checkedto verify that no writes are being made to a read-only page.
- An attempt to write to a read-only page causes a hardware-trap to the OS (or memory protection violation).

Valid Invalid Bit

- This bit is attached to each entry in the page-table.
- Valid bit: "valid" indicates that the associated page is in the process' logical addressspace, and is thus a legal page
- Invalid bit: "invalid" indicates that the page is not in the process' logical address space
- Illegal addresses are trapped by use of valid-invalid bit.
- The OS sets this bit for each page to allow or disallow access to the page.

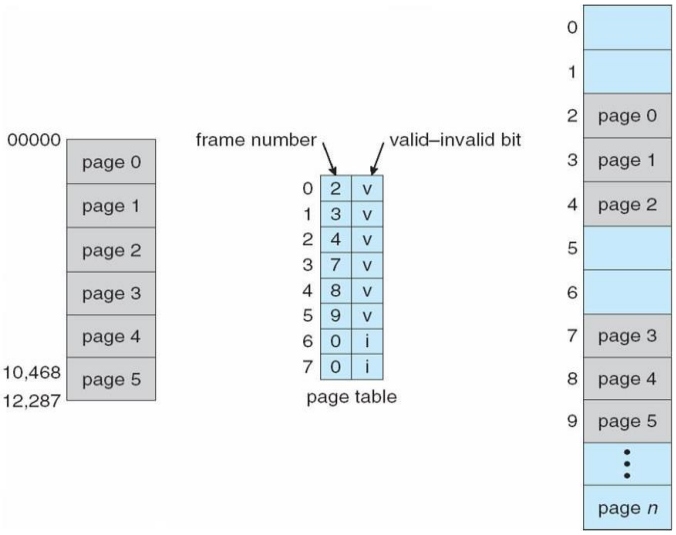


Figure: Valid (v) or invalid (i) bit in a page-table

Shared Pages

- An advantage of paging is the possibility of sharing common code.
- Re-entrant code (Pure Code) is non-self-modifying code, it never changes duringexecution.
- Two or more processes can execute the same code at the same time.
- Each process has its own copy of registers and data-storage to hold the data for the process's execution.
- The data for 2 different processes will be different.
- Only one copy of the editor need be kept in physical-memory
- Each user's page-table maps onto the same physical copy of the editor, but data pages are mapped onto different frames.

Disadvantage:

Systems that use inverted page-tables have difficulty implementing shared-memory.

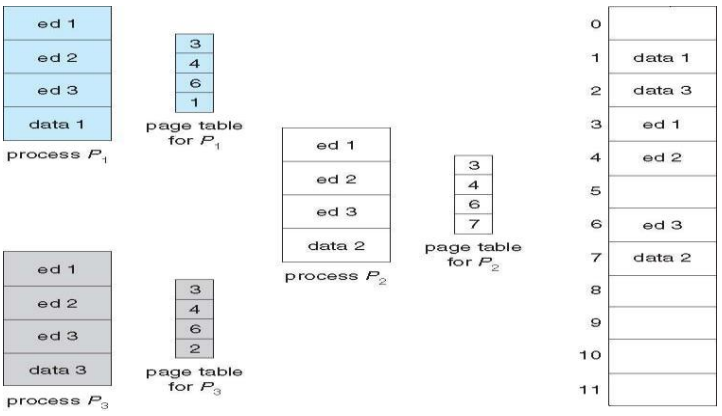


Figure: Sharing of code in a paging environment

Structure of the Page Table

The most common techniques for structuring the page table:

1. Hierarchical Paging
2. Hashed Page-tables
3. Inverted Page-tables

1. Hierarchical Paging

- Problem: Most computers support a large logical-address space (232 to 264). In these systems, the page-table itself becomes excessively large.
- Solution: Divide the page-table into smaller pieces.

Two Level Paging Algorithm:

- The page-table itself is also paged.
- This is also known as a forward-mapped page-table because address translation worksfrom the outer page-table inwards.

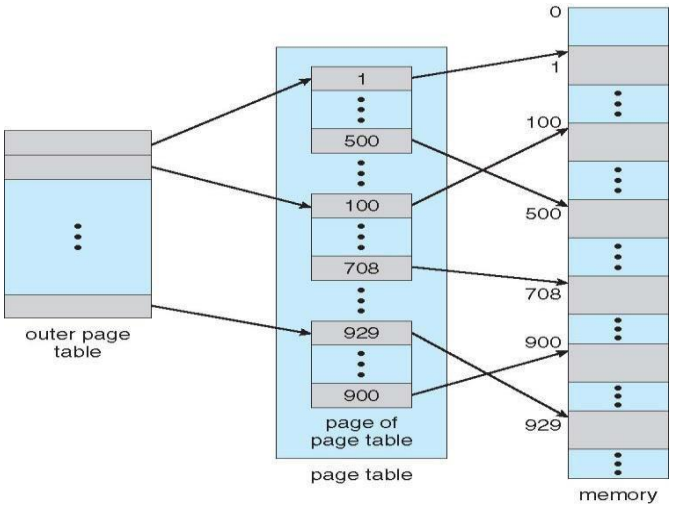


Figure: A two-level page-table scheme

For example:

Consider the system with a 32-bit logical-address space and a page-size of 4 KB.A logical-address is divided into

→ 20-bit page-number and

→ 12-bit page-offset.

Since the page-table is paged, the page-number is further divided into

→ 10-bit page-number and

→ 10-bit page-offset.

Thus, a logical-address is as follows:

page number		page offset
p_1	p_2	d
12	10	10

- where p_1 is an index into the outer page table, and p_2 is the displacement within thepage of the inner page table

The address-translation method for this architecture is shown in below figure. Because address translation works from the outer page table inward, this scheme is also known as a forward- mapped page table.

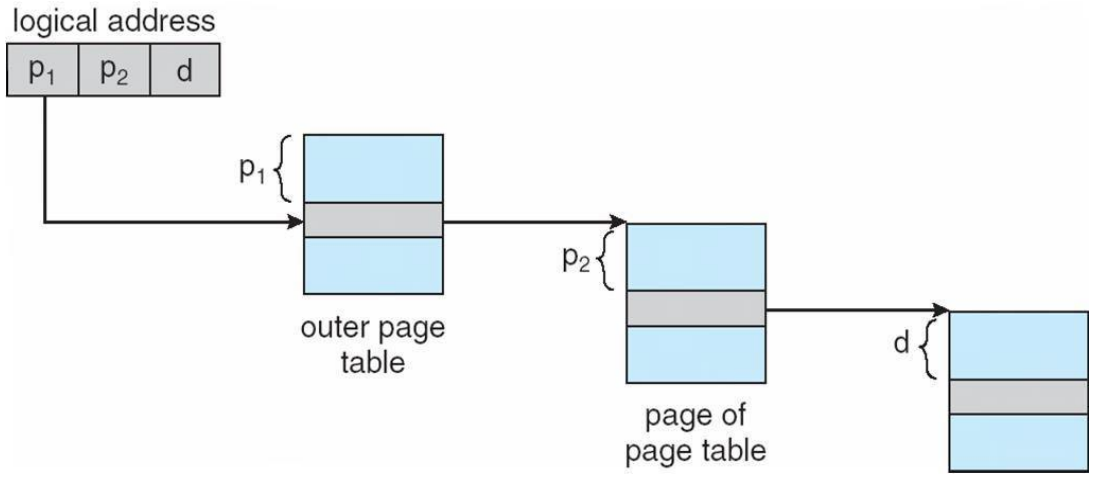


Figure: Address translation for a two-level 32-bit paging architecture

2. Hashed Page Tables

- This approach is used for handling address spaces larger than 32 bits.
- The hash-value is the virtual page-number.
- Each entry in the hash-table contains a linked-list of elements that hash to the same location (to handle collisions).
- Each element consists of 3 fields:
  1. Virtual page-number
  2. Value of the mapped page-frame and
  3. Pointer to the next element in the linked-list.

The algorithm works as follows:

- The virtual page-number is hashed into the hash-table.
- The virtual page-number is compared with the first element in the linked-list.
- If there is a match, the corresponding page-frame (field 2) is used to form the desired physical-address.
- If there is no match, subsequent entries in the linked-list are searched for a matchingvirtual page-number.

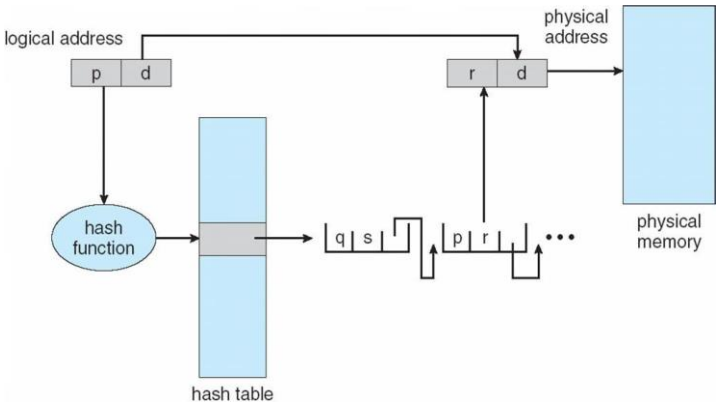


Figure: Hashed page-table

3. Inverted Page Tables

- Has one entry for each real page of memory.
- Each entry consists of virtual-address of the page stored in that real memory-locationand information about the process that owns the page.
- Each virtual-address consists of a triplet <process-id, page-number, offset>.

- Each inverted page-table entry is a pair <process-id, page-number>

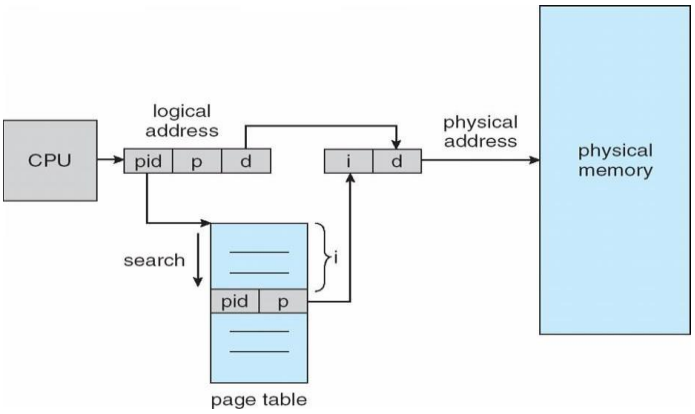


Figure: Inverted page-table

The algorithm works as follows:

1. When a memory-reference occurs, part of the virtual-address, consisting of <process-id,page-number>, is presented to the memory subsystem.
2. The inverted page-table is then searched for a match.
3. If a match is found, at entry i-then the physical-address <i, offset> is generated.
4. If no match is found, then an illegal address access has been attempted.

Advantage:

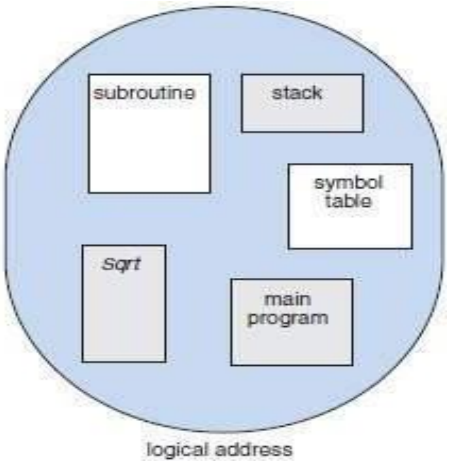
1. Decreases memory needed to store each page-table

Disadvantages:

1. Increases amount of time needed to search table when a page reference occurs.
2. Difficulty implementing shared-memory

Segmentation

Basic Method of Segmentation



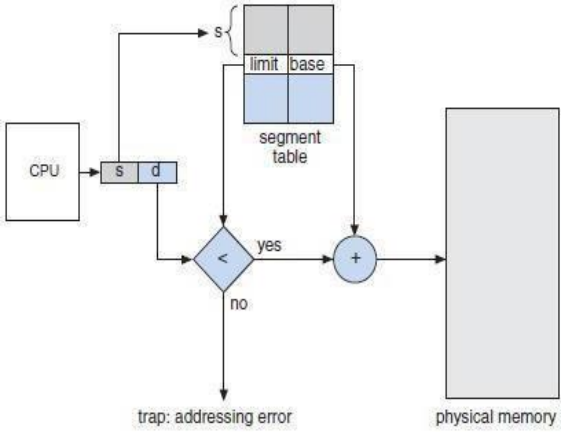
- This is a memory-management scheme that supports user-view of memory (Figure 1).
- A logical-address space is a collection of segments.
- Each segment has a name and a length.
- The addresses specify both segment-name and offset within the segment.
- Normally, the user-program is compiled, and the compiler automatically constructssegments reflecting the input program.
- For ex: The code, Global variables, The heap, from which memory is allocated, Thestacks used by each thread, The standard C library

Hardware support for Segmentation

- Segment-table maps 2 dimensional user-defined addresses into one-dimensional physical addresses.
- In the segment-table, each entry has following 2 fields:
  1. Segment-base contains starting physical-address where the segment resides inmemory.
  2. Segment-limit specifies the length of the segment (Figure 2).
- 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



✓ **For example**, consider the below **figure.** We have **five segments** numbered from 0 through 4. Segment 2 is 400 bytes long and begins at location 4300. Thus, a reference to byte 53 of segment 2 is mapped onto location 4300 +53= 4353. A reference byte 852 of segment 3, is mapped to 3200 (the base of segment 3) + 852 = 4052. A reference to byte 1222 of segment 0 would result in a trap to the operating system, as this segment is only 1,000 bytes long.

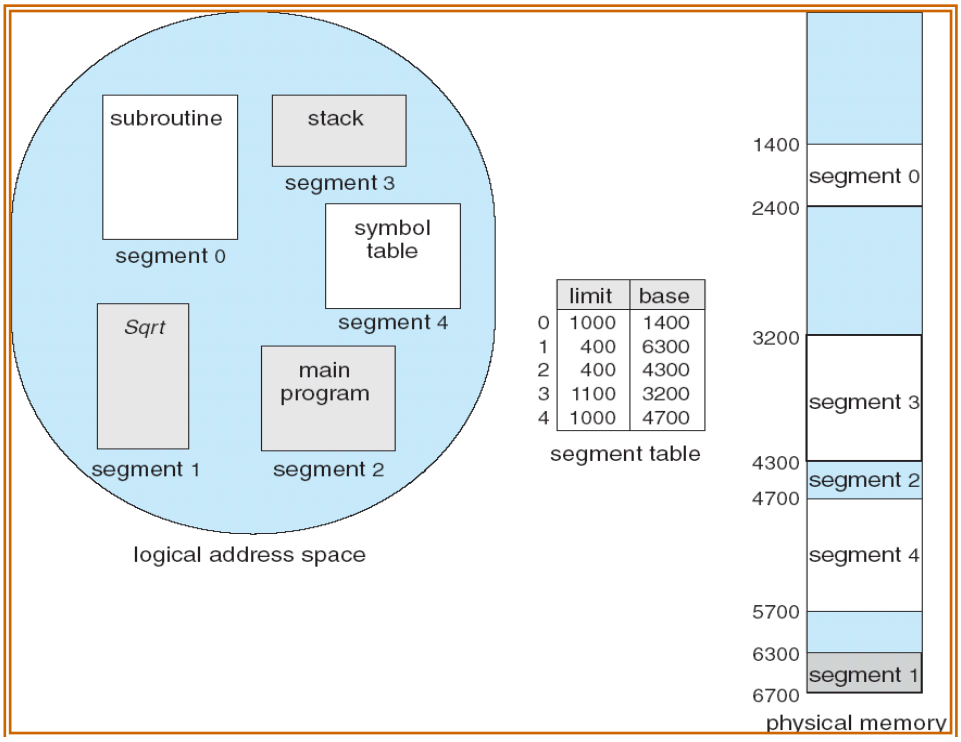


Figure: Segmentation hardware