目录 Contents

1 Overviews
2 Process Description and Control
3 Threads and Kernel Architecture
4 Concurrency: Mutual Exclusion and Synchronization
5 Concurrency: Deadlock and Starvation
6 Memory Management and Virtual Memory
7 Uniprocessor Scheduling
8 I/O Management and Disk Scheduling
9 File Management

Memory Management and Virtual Memory

1 Memory Management Requirements
2 Memory Partitioning
3 Virtual Memory
4 Paging
5 Segmentation
6 Combined Paging and Segmentation
7 Operating System Software For VM

6.1 Memory Management Requirements

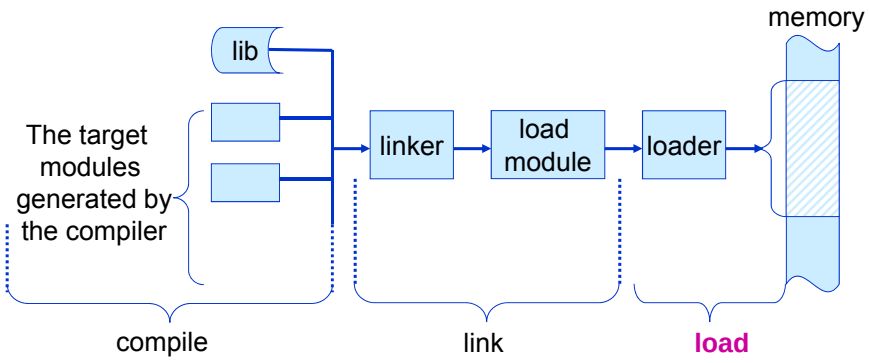
Memory Management Requirements

- Memory management is intended to satisfy the following requirements:
  - Relocation(重定位)
  - Protection
  - Sharing
  - Logical organization
  - Physical organization

Relocation

- Programmers typically do not know in advance which other programs will be resident in main memory at the time of execution of their program
- Active processes need to be able to be swapped in and out of main memory in order to maximize processor utilization
- Specifying that a process must be placed in the same memory region when it is swapped back in would be limiting
  - may need to relocate the process to a different area of memory

Source Programs To Executable Program



- Memory references must be translated in the code to actual physical memory address——**Relocation**.

Addressing Requirements for a Process

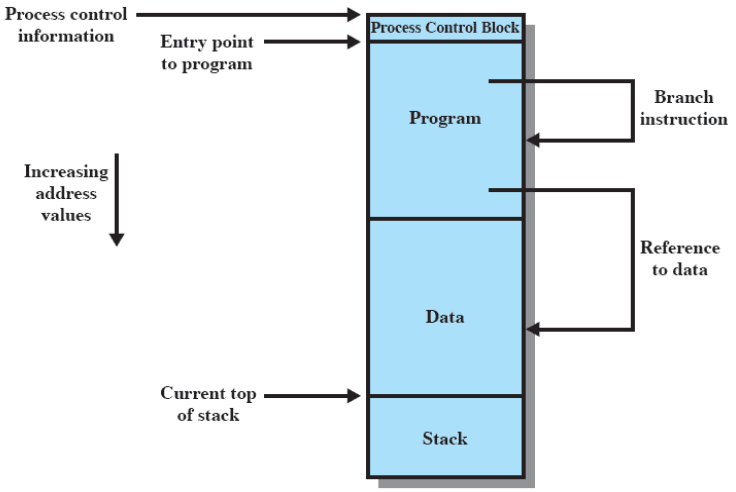


Figure 7.1 Addressing Requirements for a Process

Protection

- Processes should not be able to reference memory locations in another process without permission
  - Impossible to check absolute addresses at compile time to assure protection
  - Address reference must be checked at run time
- Memory protection requirement must be satisfied by the processor (hardware) rather than the operating system (software)
  - Operating system cannot anticipate all of the memory references a program will make

Sharing

- Allow several processes to access the same portion of memory
  - Better to allow each process access to the same copy of the program rather than have their own separate copy

Logical Organization

- main memory is organized as linear, or one-dimesion, address space, consisting of a sequence of bytes or words;
- Programs are written in modules
  - modules can be written and compiled independently
  - different degrees of protection given to modules (read-only, execute-only)
  - sharing on a module level corresponds to the user's way of viewing the problem
- Segmentation is the tool that most readily satisfies requirements

Physical Organization

- The memory hierarchy
  - cache memory
  - main memory
  - secondary memory(auxiliary memory)

capacity — access time

low ↓ high

cache	volatile, not stored permanently	expensive
main memory	volatile, not stored permanently	Medium
secondary memory	not volatile, stored permanently	cheap

fast ↑ slow

Physical Organization.

- the organization of the flow of information between main and secondary memory is a major system concern.
- Cannot leave the programmer with the responsibility to manage memory
  - Memory available for a program plus its data may be insufficient
    - **Overlaying** allows various modules to be assigned the same region of memory
  - Programmer does not know how much space will be available

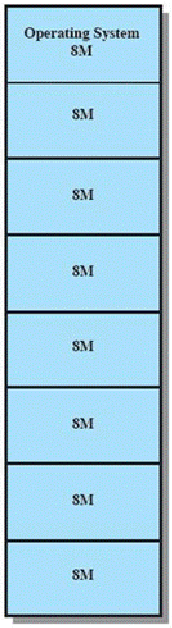
6.2 Memory Partitioning

1 Fixed Partitioning

- **Fixed Partitioning**: The memory is divided into a number of static partitions. A user program or process should be loaded into a partition or equal or greater.
  - Equal-size partitions
  - Unequal-size partitions

Equal-size Partitions

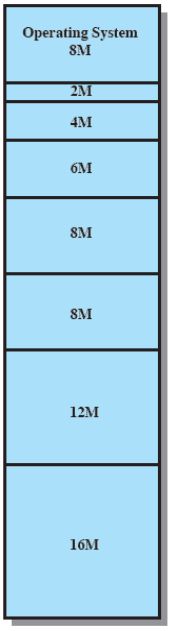
- any process whose size is less than or equal to the partition size can be loaded into an available partition.
- The operating system can swap out a process if all partitions are full and no process is in the Ready or Running state.



(a) Equal-size partitions

Unequal-size Partitions

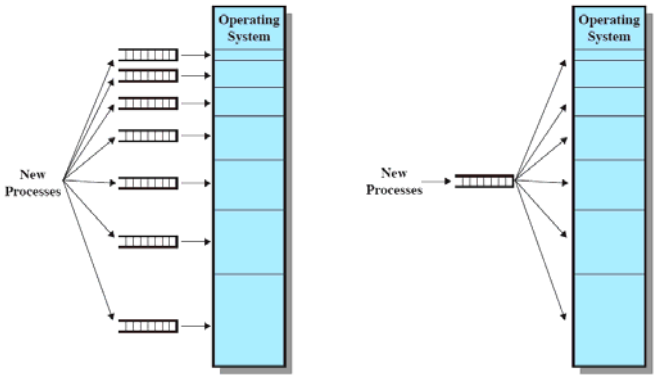
- Disadvantages of Equal-size Partitions
  - A program may be too big to fit in a partition
    - Program needs to be designed with the use of overlays
  - Main memory utilization is inefficient: any program, regardless of size, occupies an entire partition
    - **Internal fragmentation**: wasted space due to the block of data loaded being smaller than the partition
- Using unequal size partitions helps lessen the problems



(b) Unequal-size partitions

Placement Algorithm with Partitions

- Can assign each process to the smallest partition within which it will fit.
- Processes are assigned in such a way as to minimize wasted memory within a partition.

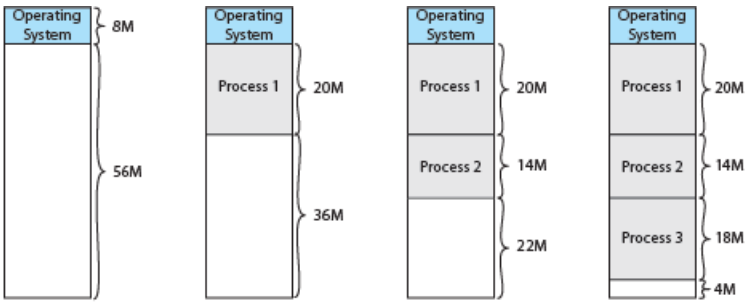


(a) One process queue per partition (b) Single queue

6.2 Memory Partitioning

2 Dynamic Partitioning

- **Dynamic Partitioning**: Partitions are of variable length and number
- Process is allocated exactly as much memory as required



(a) (b) (c) (d)

Dynamic Partitioning.

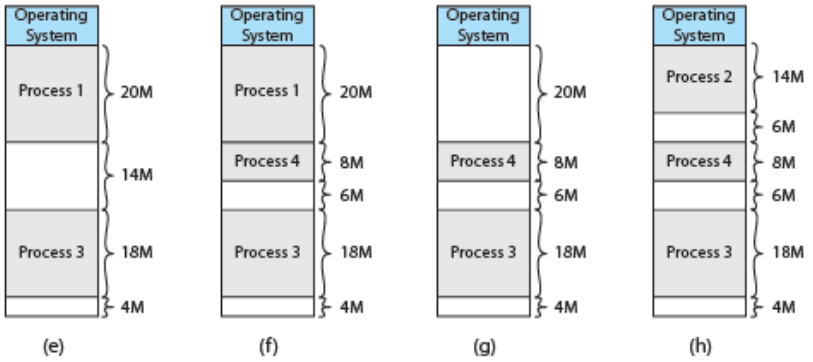


Figure 7.4 The Effect of Dynamic Partitioning

Disadvantages of Dynamic Partitioning

- Eventually get holes in the memory. This is called **external fragmentation** (外零头/外部碎片)
- Must use **compaction**(紧缩) to shift processes so they are contiguous and all free memory is in one block
  - time consuming and wastes CPU time

Dynamic Partitioning Placement Algorithm

- **First-fit(首次适配) algorithm**: Scans memory from the beginning and chooses the first available block that is large enough
  - May have many process loaded in the front end of memory that must be searched over when trying to find a free block
- **Next-fit(下次适配)**: Scans memory from the location of the last placement
  - More often allocate a block of memory at the end of memory where the largest block is found
  - The largest block of memory is broken up into smaller blocks
  - Compaction is required to obtain a large block at the end of memory

Dynamic Partitioning Placement Algorithm.

- **Best-fit(最佳适配) algorithm**: Chooses the block that is closest in size to the request
  - Since smallest block is found for process, the smallest amount of fragmentation is left
  - Memory compaction must be done more often
  - Worst performer overall

Example of Placement Algorithms

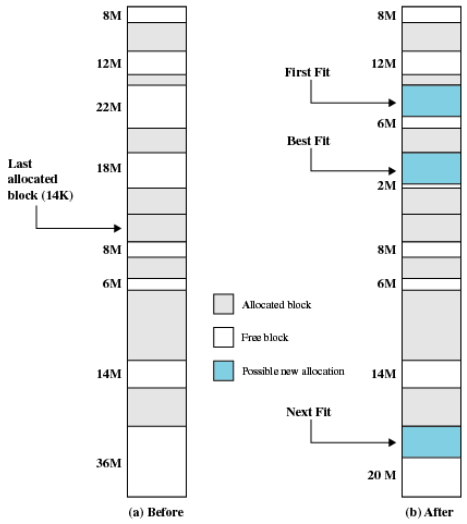


Figure 7.5 Example Memory Configuration Before and After Allocation of 16 Mbyte Block

6.2 Memory Partitioning

3 Buddy System

- **Buddy System(伙伴系统)**: Entire space available is treated as a single block of 2^U . If a request of size s such that $2^{U-1} < s \leqslant 2^U$, entire block is allocated, otherwise block is split into two equal buddies 2^{U-1}. Process continues until smallest block($2^k \geqslant 2^L$, 2^L = smallest size block that is allocated) greater than or equal to s is generated.
- Comprised of fixed and dynamic partitioning schemes

Example of Buddy System

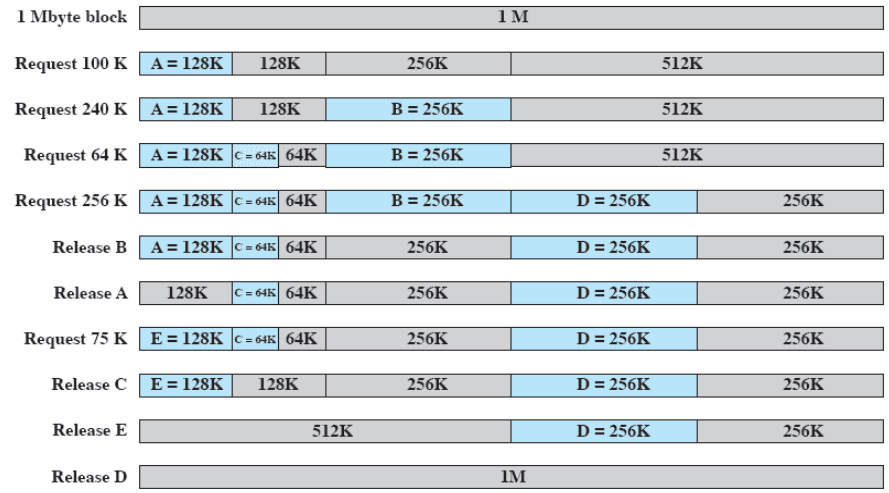


Figure 7.6 Example of Buddy System

Tree Representation of Buddy System

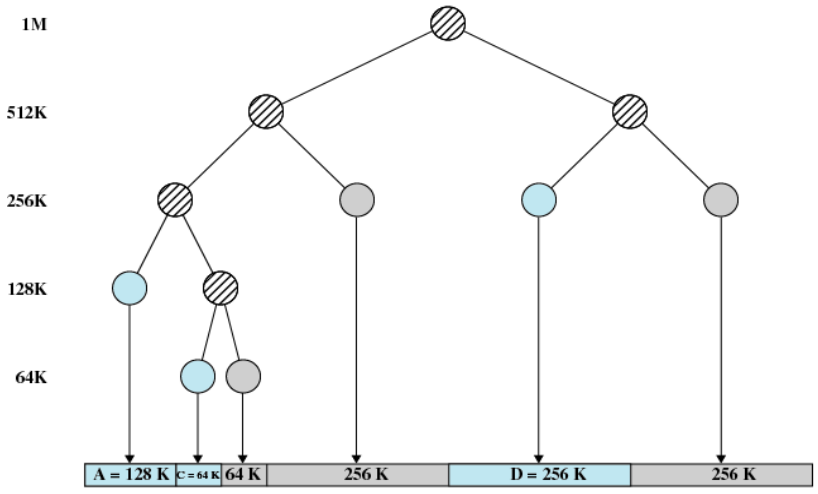


Figure 7.7 Tree Representation of Buddy System

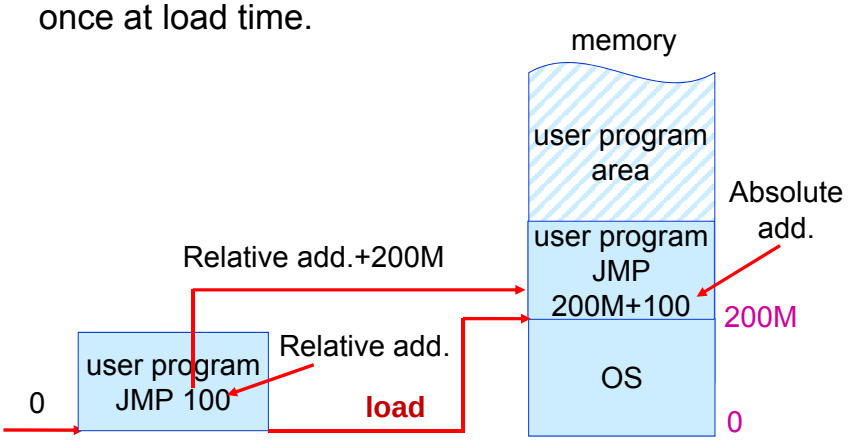
6.2 Memory Partitioning

4 Relocation(重定位)

- Addresses
  - **Logical address(逻辑地址)**: reference to a memory location independent of the current assignment of data to memory. A translation must be made to a physical address before the memory access can be achieved.
  - **Relative address(相对地址)**: address is expressed as a location relative to some known point.
  - **Physical or Absolute address**: is an actual location in main memory.

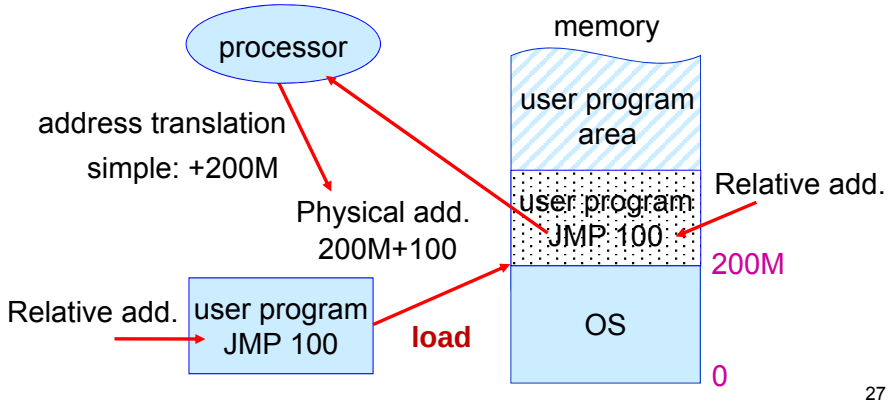
Static Relocation

- **Static relocation**: Address translation is done only once at load time.



Dynamic Relocation

- **Dynamic relocation**: Address translation is done by dynamic address translation mechanism (hardware) when processor access to main memory every time.



Hardware Support for Relocation

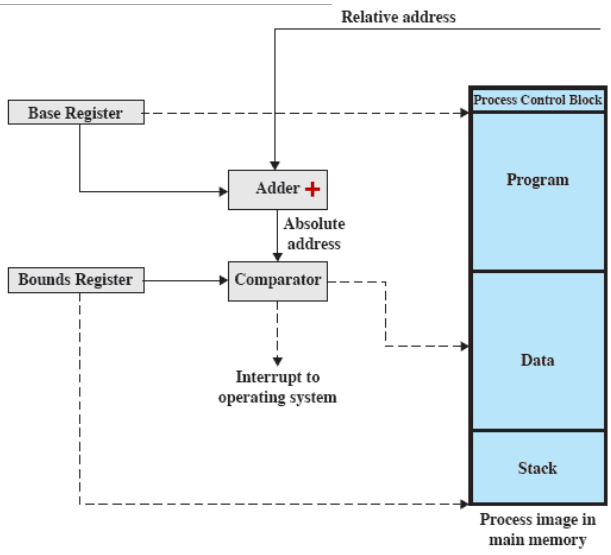


Figure 7.8 Hardware Support for Relocation

6.3 Virtual Memory

1 Why Virtual Memory?

- Many years ago people were first confronted with programs that were too big to fit in the available memory.

- **Overlay(覆盖)**: split a program into fragments called overlays. The program does not require that all overlays be loaded at runtime. For example, overlay 0 runs first and another overlay is called when it ends, or several overlays run first and then other overlays are called.

Principle of Locality(局部性原理)

- Program and data references within a process tend to **cluster**. Only a few pieces of a process will be needed **over a short period of time**.
  - **Time locality**: Once an instruction is executed or data is accessed, the instruction may be executed again shortly thereafter and the data may be accessed again—cyclic program
  - **Spatial locality**: Once a program accesses an unit, it is not long before nearby units are also accessed—The sequential execution of a program
- Possible to make intelligent guesses about which pieces will be needed in the future.

Advantages of Breaking Up A Process

- More processes may be maintained in main memory
  - Only load in some of the pieces of each process
  - With so many processes in main memory, it is very likely a process will be in the Ready state at any particular time
- A process may be larger than all of main memory
  - A process may be swapped in and out of main memory partially.
- Overlay: Although the actual work of swapping overlays in and out was done by the system, the decision of how to split the program into pieces had to be done by the programmer.
- Virtual Memory: controlled by OS

Basic Idea of Virtual Memory Management

- Memory references are dynamically translated into physical addresses at runtime—**Dynamic Relocation(动态重定位)**
  - A process may be swapped in and out of main memory such that it occupies different regions
- A process may be broken up into pieces that do not need to located contiguously(连续的) in main memory—**Paging or Segmentation(分页/分段)**
- All pieces of a process do not need to be loaded in main memory during execution—**Page Fault(缺页故障)**

Execution of A Program

- Operating system only brings into main memory a few pieces of the program
  - **Resident set(驻留集)** - portion of process that is in main memory
- **An interrupt(Page Fault, 页面故障)** is generated when an address is needed that is not in main memory. OS places the process in blocking state.
  - OS issues a disk I/O Read request
  - Another process is dispatched to run while the disk I/O takes place
  - An interrupt is issued when disk I/O complete which causes the OS to place the affected process in the Ready state

Support Needed for Virtual Memory

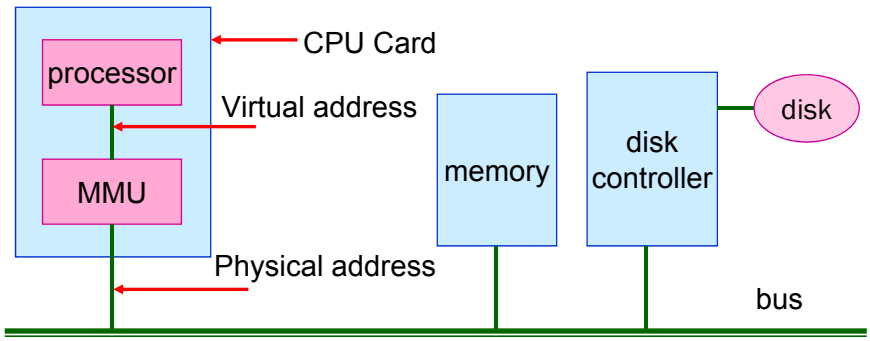
- Real memory: Main memory
- Virtual memory: Memory on disk
  - Allows for effective multiprogramming and relieves the user of tight constraints of main memory

Support Needed for Virtual Memory

- Hardware must support paging and segmentation
- Operating system must be able to manage the movement of pages and/or segments between secondary memory and main memory

Memory Manage Unit

- **Memory manage unit(MMU)**: consists of one or a group of chips, usually a base address register and a program virtual address register, whose function is to translate virtual addresses to physical addresses.



Virtual Memory Terminology

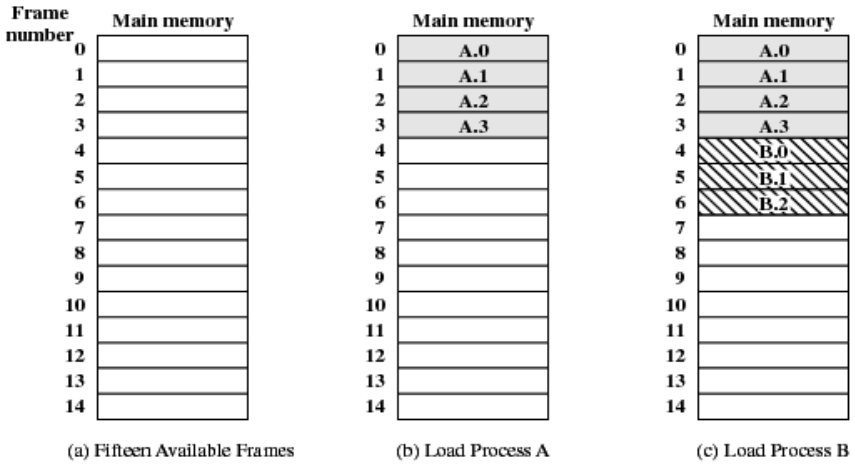
Virtual memory	A storage allocation scheme in which secondary memory can be addressed as though it were part of main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites, and program-generated addresses are translated automatically to the corresponding machine addresses.The size of virtual storage is limited by the addressing scheme of the computer system and by the amount of secondary memory available and not by the actual number of main storage locations.
Virtual address	The address assigned to a location in virtual memory to allow that location to be accessed as though it were part of main memory.
Virtual address space	The virtual storage assigned to a process.
Address space	The range of memory addresses available to a process.
Real address	The address of a storage location in main memory.

6.4 Paging

1 Paging

- Partition memory into small equal fixed-size chunks and divide each process into the same size chunks
- The chunks of a process are called **pages(页)** and chunks of memory are called **frames(页框)**
- Operating system maintains a **page table** for each process
  - Contains the frame location for each page in the process
  - Memory address consist of a page number and offset within the page

Assignment of Process Pages to Free Frames



(a) Fifteen Available Frames (b) Load Process A (c) Load Process B

Assignment of Process Pages to Free Frames.

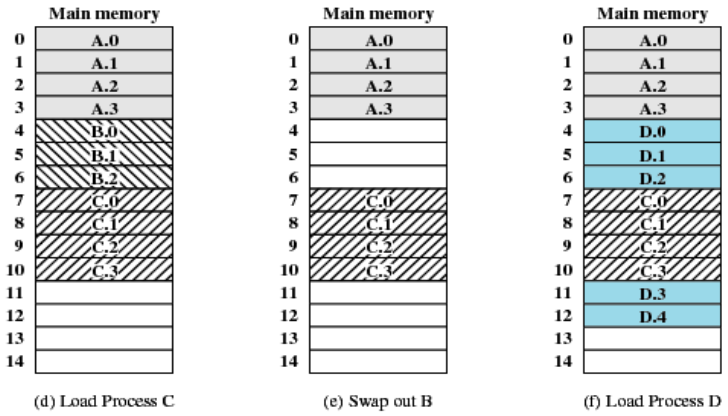


Figure 7.9 Assignment of Process Pages to Free Frames

Page Tables for Example

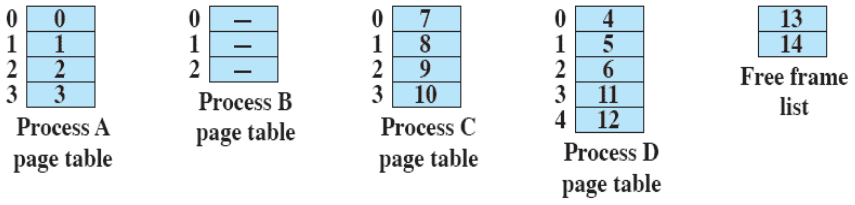
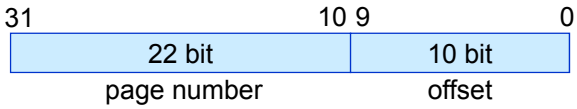


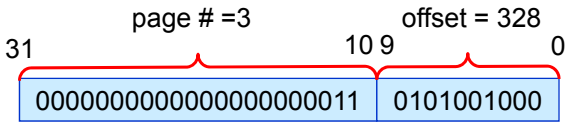
Figure 7.10 Data Structures for the Example of Figure 7.9 at Time Epoch (f)

Virtual Address Formats Based Paging

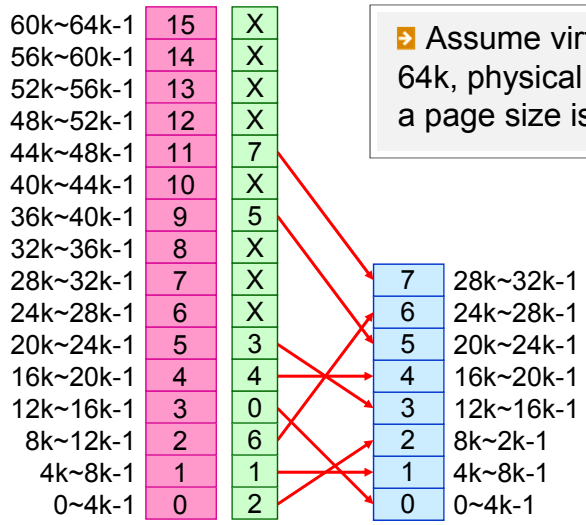
- Assume address space is 32bit, that is, 2^{32}=4G, a page size is 1k=2^{10}.



- A virtual address = 3400



Mapping From Virtual Add. To Physical Add.



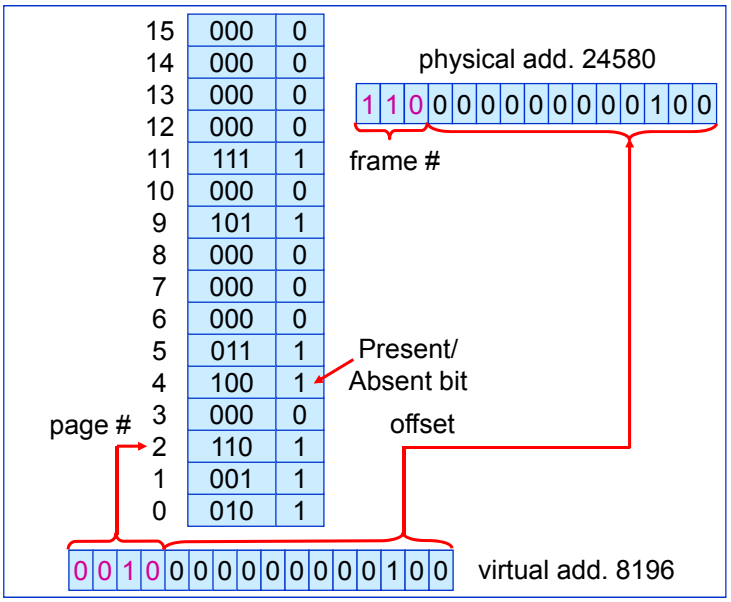
- Assume virtual add. space is 64k, physical add. space is 32k, a page size is 4k.

```
mov reg, 0
mov reg, 8192
mov reg, 20500
```

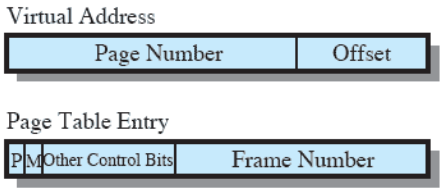
0→8192
8192→24576
20500→12308

```
mov reg, 32780
```

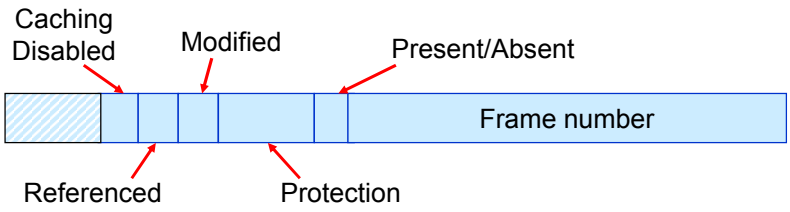
Address Translation In MMU



Virtual Address and Page Table Entry



(a) Paging only



Page Table Entry in Virtual Memory

Paging In Virtual Memory

- Each process has its own **page table**
- Each **page table entry(PTE)** contains the frame number of the corresponding page in main memory.
  - A bit is needed in PTE to indicate whether the page is in main memory or not
  - Modify bit is needed to indicate if the page has been altered since it was last loaded into main memory
  - If no change has been made, the page does not have to be written to the disk when it needs to be swapped out

Address Translation(Mapping)

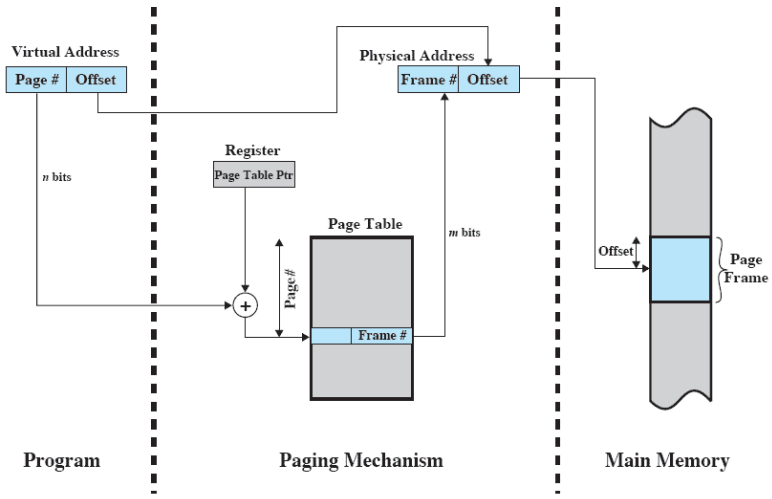


Figure 8.3 Address Translation in a Paging System

6.4 Paging

2 Multi-Level Paging

- **Entire page table** may take up too much main memory.

- Page tables are also stored in virtual memory, when a process is running, part of its page table is in main memory.

- Assume address space is 32bit, that is, 2^{32}=4G, a page size is 4k=2^{12}. Using a two-level page table, a page table block=2^{10}, an entry need 4 byte.

10bit	10bit	12bit
PT1	PT2	Offset
top level page #	2nd level page #	offset

Two-Level Scheme for 32-bit Address

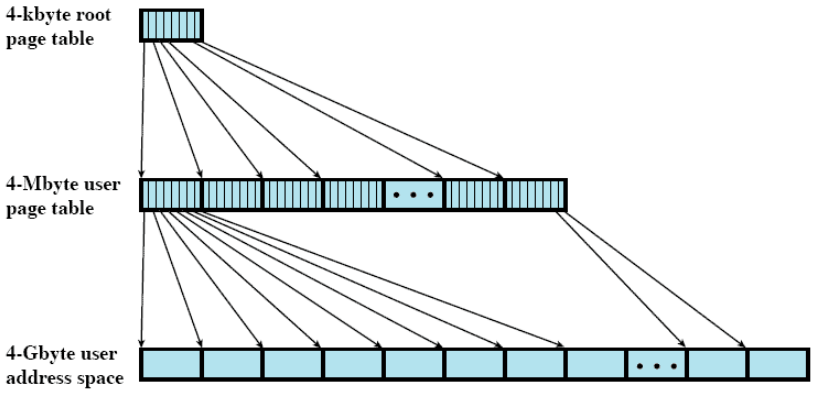
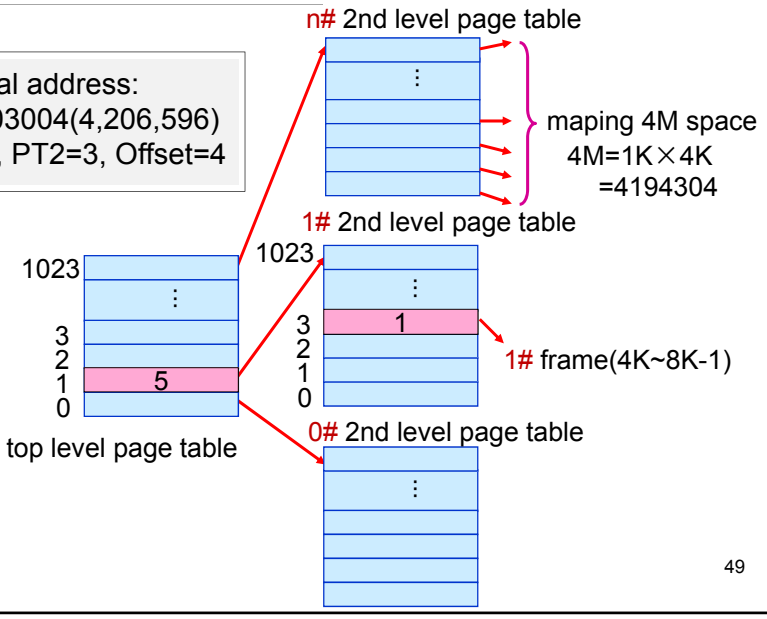


Figure 8.4 A Two-Level Hierarchical Page Table

Address Translation In Two-Level Scheme

- Virtual address: 0x00403004(4,206,596) PT1=1, PT2=3, Offset=4



Address Translation In Two-Level Scheme.

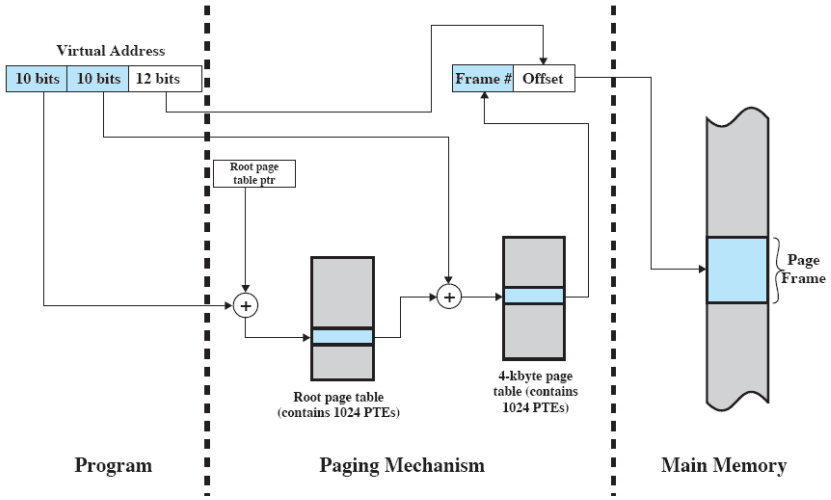


Figure 8.5 Address Translation in a Two-Level Paging System

Translation Lookaside Buffer

- How many time will physical memory accesses occur for each virtual memory reference?

- Each virtual memory reference can cause two physical memory accesses.
  - One to fetch the page table
  - One to fetch the data
- To overcome this problem a high-speed cache is set up for page table entries, called a **Translation Lookaside Buffer(TLB)(转换检测缓冲区)**.
- TLB contains page table entries that have been most recently used.

Translation Lookaside Buffer.

- Most programs access to a few pages at a period

Present	Page #	Modified	Protection	Frame #
1	140	1	rw	31
1	20	0	rx	38
1	130	1	rw	29
1	129	1	rw	62
1	19	0	rx	50
1	21	0	rx	45
1	860	1	rw	14
1	861	1	rw	75

Access A Virtual Address Steps Using TLB

- Given a virtual address, processor examines the TLB
- If page table entry is present (TLB hit), the frame number is retrieved and the real address is formed
- If page table entry is not found in the TLB (TLB miss), the page number is used to index the process page table
- First checks if page is already in main memory
  - If not in main memory a page fault is issued
- The TLB is updated to include the new page entry

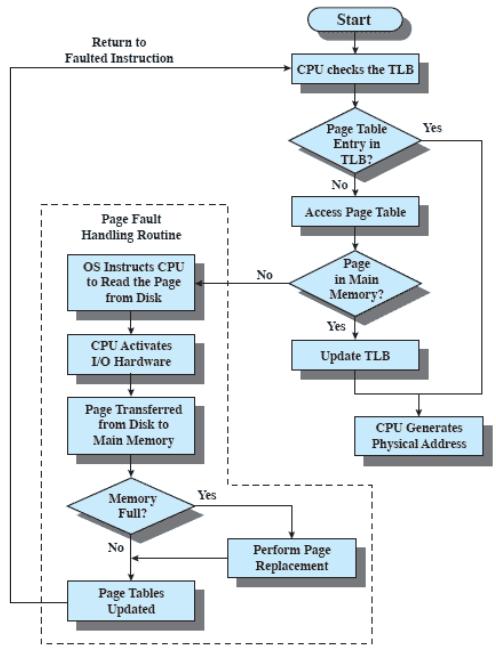


Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

Use of A TLB

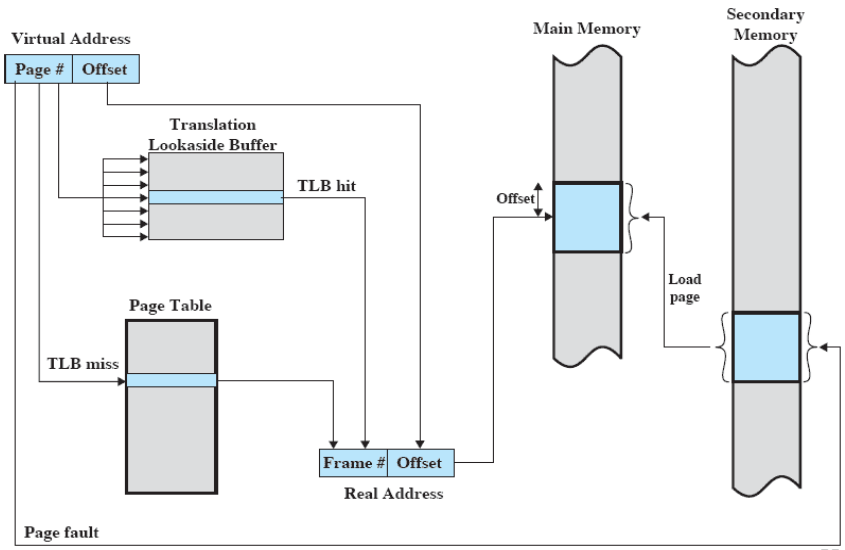


Figure 8.7 Use of a Translation Lookaside Buffer

Associative Mapping(关联映射)

- The TLB only contains some of the page table entries so we cannot simply index into the TLB based on page number
  - each TLB entry must include the page number as well as the complete page table entry
- The processor is equipped with hardware that allows it to interrogate simultaneously a number of TLB entries to determine if there is a match on page number

Direct Versus Associative Lookup

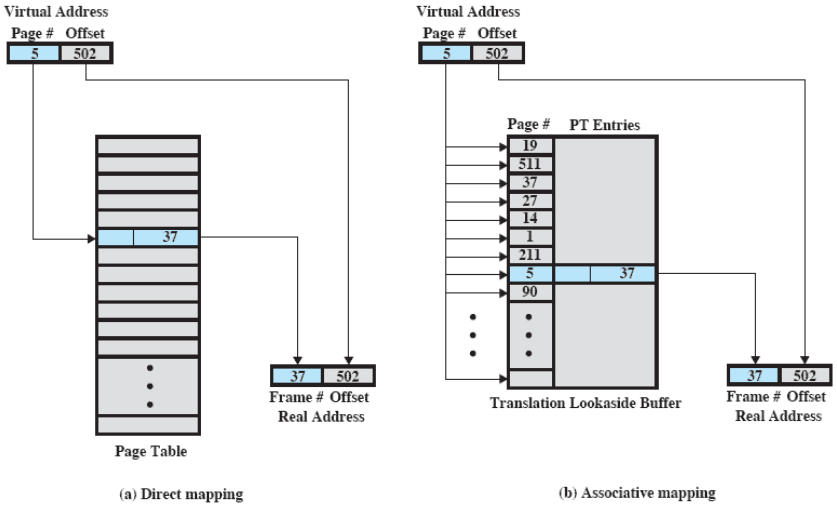


Figure 8.9 Direct Versus Associative Lookup for Page Table Entries

TLB And Cache Operation

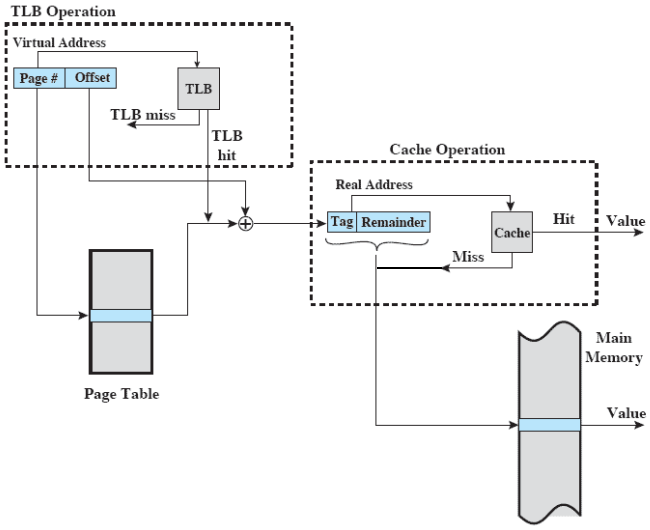


Figure 8.10 Translation Lookaside Buffer and Cache Operation

6.4 Paging

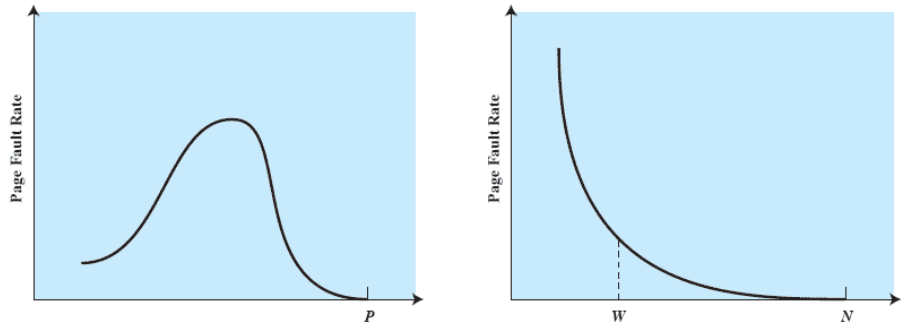
3 Page Size

- Smaller page size
  - advantage: less amount of internal fragmentation
  - disadvantage
    - Smaller page size, more pages required per process, more pages per process means larger page tables
    - Larger page tables means large portion of page tables in virtual memory, may be a double page fault
- Large page size
  - advantage: Secondary memory is designed to efficiently transfer large blocks of data so a large page size is better

the Effect of Page Size on Page Faults

- Small page size, large number of pages will be found in main memory, as time goes on during execution, the pages in memory will all contain portions of the process near recent references—Page faults low.
- Increased page size causes pages to contain locations further from any recent reference—Page faults rise.
- However, the page fault rate will begin to fall as the size of a page approaches the size of the entire process (point P in the diagram).

Paging Behavior of A Program



(a) Page Size

(b) Number of Page Frames Allocated

P = size of entire process
W = working set size
N = total number of pages in process

Figure 8.11 Typical Paging Behavior of a Program

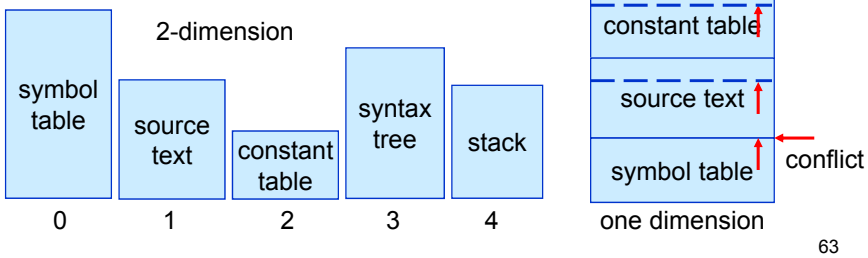
Example Page Sizes

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit words
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 Kbytes to 16 Mbytes
UltraSPARC	8 Kbytes to 4 Mbytes
Pentium	4 Kbytes or 4 Mbytes
IBM POWER	4 Kbytes
Itanium	4 Kbytes to 256 Mbytes

6.5 Segmentation

Why Segmentation?

- Two main deficiency about paging:
  - One dimension linear address space
  - Sharing



Segmentation

- **Segment(段)**: A segment is a logical entity. A program can be subdivided into segments, each segment contains only one type of object.
  - may vary in length
  - there is a maximum length
- **Segmentation(分段)**: allows the programmer to view memory as consisting of multiple address spaces or segments, segments may be unequal, dynamic size. To translate an address to a physical address, a segment table should be used.
- Addressing consist of two parts - a **segment number** and an **offset**.

Advantages of Segmentation

- since segments are not equal, segmentation is similar to dynamic partitioning, eliminates internal fragmentation
- simplifies handling of growing data structures
- allows programs to be altered and recompiled independently
- lends itself to sharing data among processes
- lends itself to protection

Segment Table Entry

- Each segment table entry contains the starting address of the corresponding segment in main memory and the length of the segment
- A bit is needed to determine if the segment is already in main memory
- Another bit is needed to determine if the segment has been modified since it was loaded in main memory

Address Translation in Segmentation System

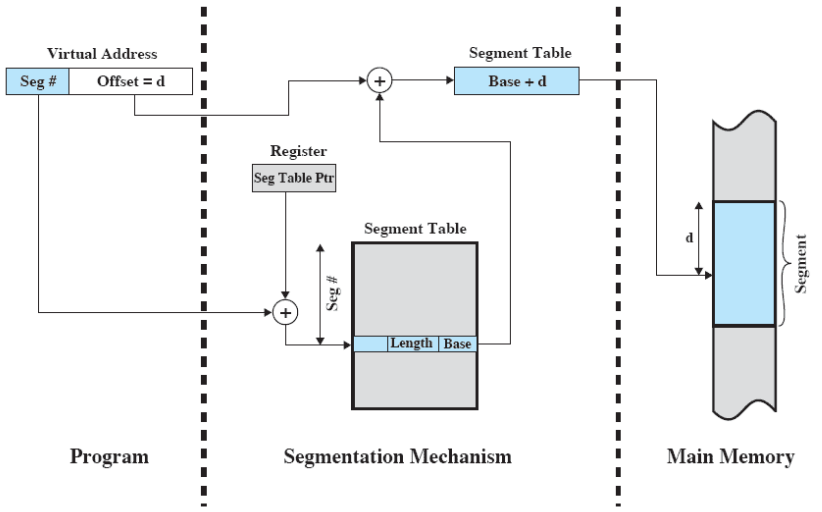


Figure 8.12 Address Translation in a Segmentation System

Address Translation in Segmentation System.

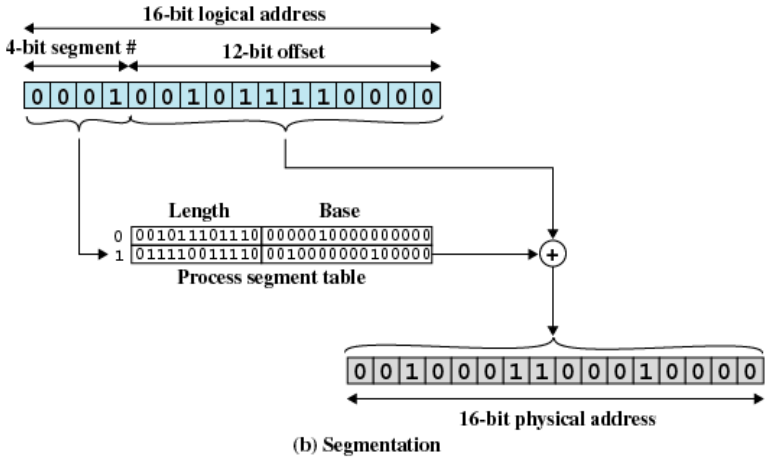
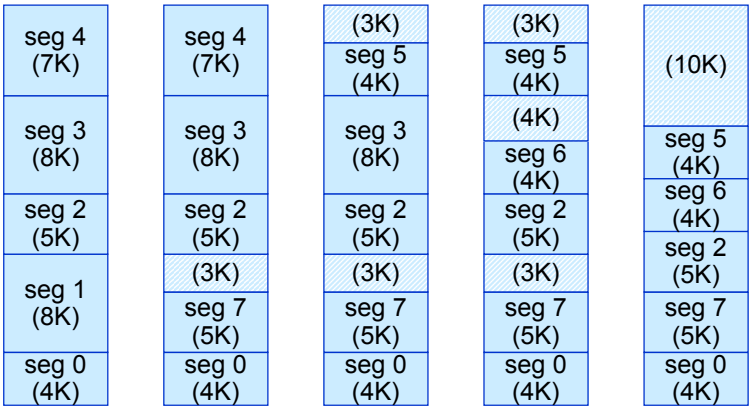


Figure 7.12 Examples of Logical-to-Physical Address Translation

Disadvantages of Segmentation



- External fragmentation(外零头)
- Segment may be too big to fit in the available memory.

69

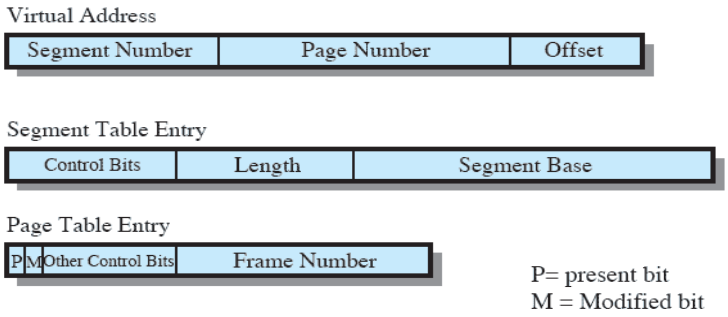
6.6 Combined Paging and Segmentation

How to combine paging and segmentation?

- Paging is transparent(透明) to the programmer
- Segmentation is visible to the programmer.

- In a combined paging/segmentation system a user's address space is broken up into a number of segments. Each segment is broken up into a number of fixed-sized pages which are equal in length to a main memory frame.

Virtual Address And Table Entry



(c) Combined segmentation and paging

Address Translation

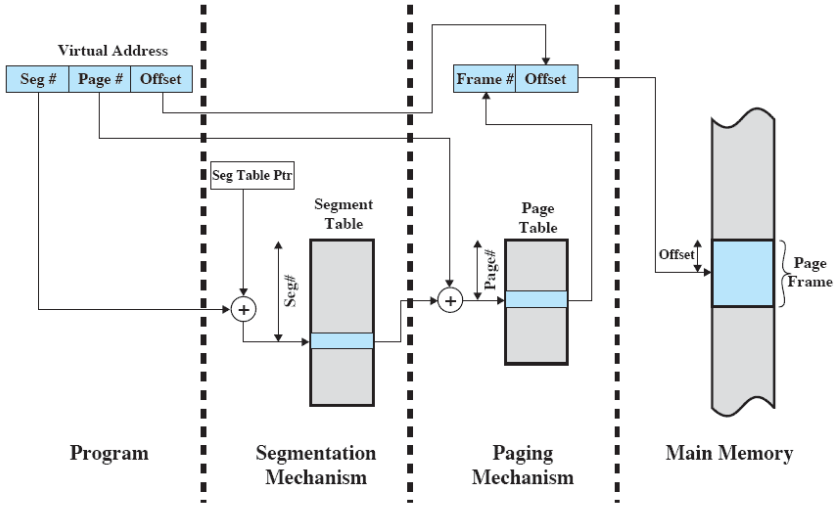
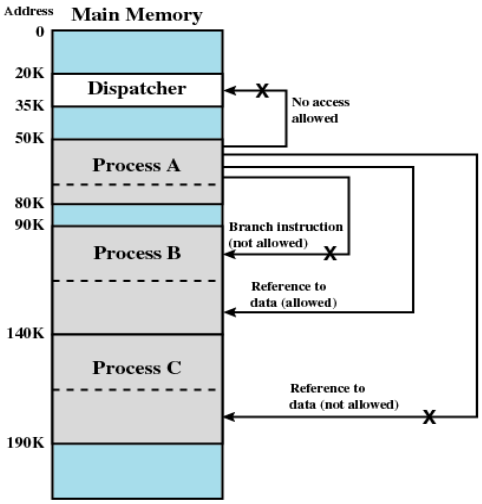


Figure 8.13 Address Translation in a Segmentation/Paging System

Protection and Sharing



- Each entry has a base address and length so inadvertent memory access can be controlled
- Sharing can be achieved by segments referencing multiple processes.

Figure 8.14 Protection Relationships Between Segments

6.7 Operating System Software For VM

1 The Design of The Memory Management

- The design of the memory management portion of an operating system depends on three fundamental areas of choice:
  - whether or not to use virtual memory techniques
  - the use of paging or segmentation or both
  - the **algorithms** employed for various aspects of memory management

Policies for Virtual Memory

- Key issue: Performance
  - Minimize the rate of page faults
  - Overhead
    - Replacement: deciding which resident page or pages to replace
    - I/O of exchanging pages
    - Process switch during page I/O

6.7 Operating System Software For VM

2 Fetch Policy(提取策略)

- Determines **when** a page should be brought into memory
- **Demand paging**: only brings pages into main memory when a reference is made to a location on the page
  - Many page faults when process first started
  - principle of locality suggests that as more and more pages are brought in, most future references will be to pages that have recently been brought in, and page faults should drop to a very low level

Prepaging

- **Prepaging(预取)**: brings in more pages than needed
  - exploits the characteristics of most secondary memory devices. If pages of a process are stored contiguously in secondary memory it is more efficient to bring in a number of pages at one time
  - ineffective if extra pages are not referenced
  - should not be confused with “swapping”

6.7 Operating System Software For VM

3 Placement Policy(放置策略)

- Where should the segments and/or pages be placed when brought into memory?

- For a segmented memory without paging (hardly ever used) we have to deal with some appropriate placement policy such as best-fit, first-fit, etc.
- Paging or combined paging with segmentation placing is irrelevant because hardware performs functions with equal efficiency
- For NonUniform Memory Access(NUMA) systems an automatic placement strategy is desirable

6.7 Operating System Software For VM

4 Replacement Policy(置换策略)

- Which page should be selected in main memory to be replaced when a new page must be brought in?

- The major concern is that page removed should be the page least likely to be referenced in the **near future**.
- Most policies predict the **future behavior** on the basis of **past behavior**.
- The more elaborate the replacement policy the greater the hardware and software overhead to implement it.

Frame Locking

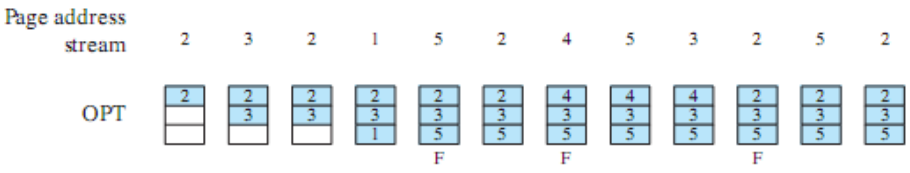
- Some of the frames in main memory may be locked. When a frame is locked, it may not be replaced
  - Kernel of the operating system
  - Key control structures
  - I/O buffers and time-critical areas
- Locking is achieved by associating a lock bit with each frame
- **Thrashing(颠簸/抖动)**: Swapping out a piece of a process just before that piece is needed
  - The processor spends most of its time swapping pieces rather than executing user instructions

Basic Replacement Algorithms

- Algorithms used for the selection of a page to replace:
  - Optimal
  - Least recently used (LRU)
  - First-in-first-out (FIFO)
  - Clock

(1) Optimal Policy(最佳算法)

- Selects for replacement that page for which the time to **the next reference** is the **longest**
- Difficult to implement: Impossible to have perfect knowledge of **future events**



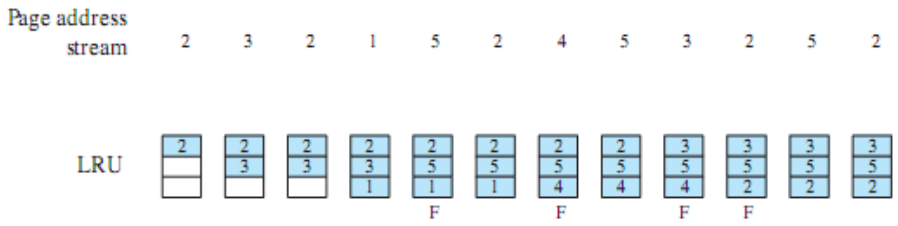
F = page fault occurring after the frame allocation is initially filled

Figure 8.15 Behavior of Four Page-Replacement Algorithms

(2) Least Recently Used (LRU, 最近最少使用)

- Replaces the page that has **not been referenced** for the **longest** time**(最久未使用算法)**
- By the principle of locality, this should be the page least likely to be referenced in the near future
- Difficult to implement: Each page could be tagged with the time of last reference. This would require a great deal of **overhead**.

LRU Example



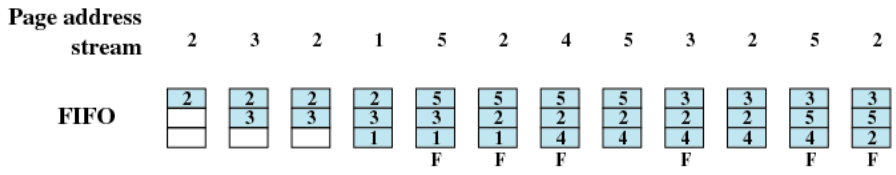
F = page fault occurring after the frame allocation is initially filled

Figure 8.15 Behavior of Four Page-Replacement Algorithms

(3) First-In-First-Out (FIFO)

- Treats page frames allocated to a process as a circular buffer
- Pages are removed in round-robin style
- Simplest replacement policy to implement
- Page that has been **in memory** the **longest** is replaced

FIFO Example



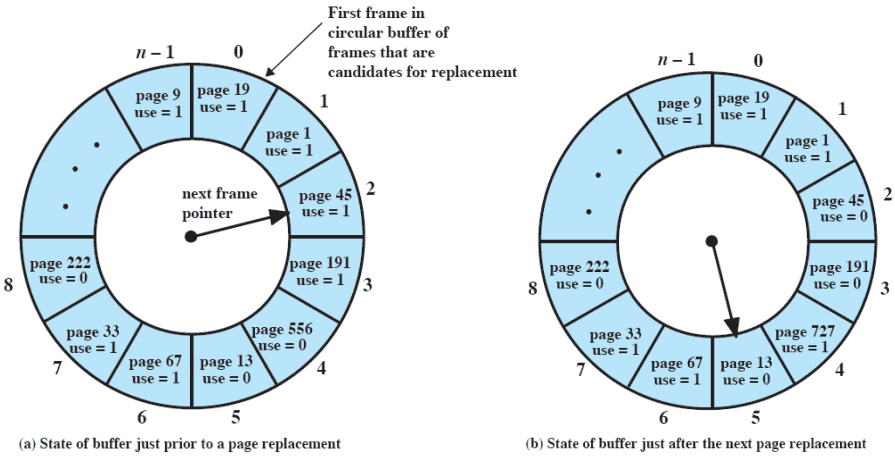
F = page fault occurring after the frame allocation is initially filled

Figure 8.15 Behavior of Four Page-Replacement Algorithms

(4) Clock Policy

- Requires the association of an additional bit with each frame, referred to as the **use bit**
- When a page is first loaded in memory, the use bit is set to 1
- When the page is referenced, the use bit is set to 1
- When it is time to replace a page, the first frame encountered with the use bit set to 0 is replaced.
- During the search for replacement, each use bit set to 1 is changed to 0(二次机会)

Clock Policy.

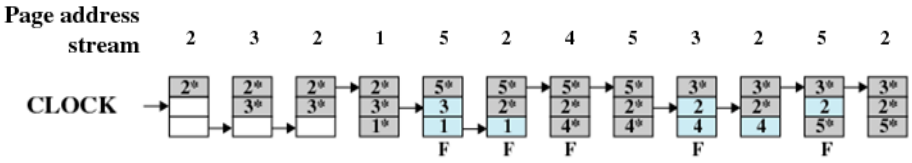


(a) State of buffer just prior to a page replacement

(b) State of buffer just after the next page replacement

Figure 8.16 Example of Clock Policy Operation

Clock Policy Example



F = page fault occurring after the frame allocation is initially filled

Figure 8.15 Behavior of Four Page-Replacement Algorithms

Comparison of Placement Algorithms

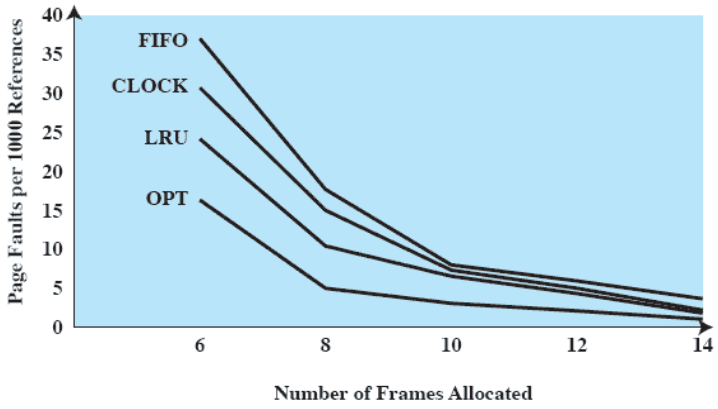
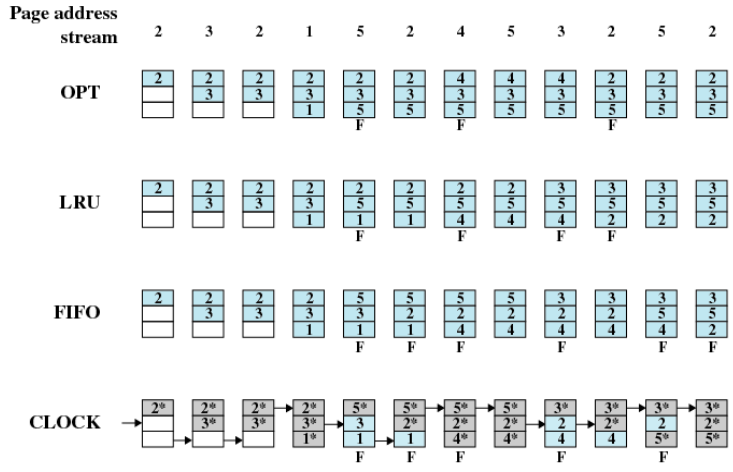


Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

Combined Examples



F = page fault occurring after the frame allocation is initially filled

Figure 8.15 Behavior of Four Page-Replacement Algorithms

(5) Clock Page-Replacement Algorithm

- FIFO is a special case of clock policy in which the use bit is not used.
- More bits can be used to improve the performance, such as a modify bit(also known as Not Recently Used, 最近未使用算法):
  - u=0, m=0
  - u=0, m=1
  - u=1, m=0
  - u=1, m=1

Low

priority

High

Clock Page-Replacement Algorithm.

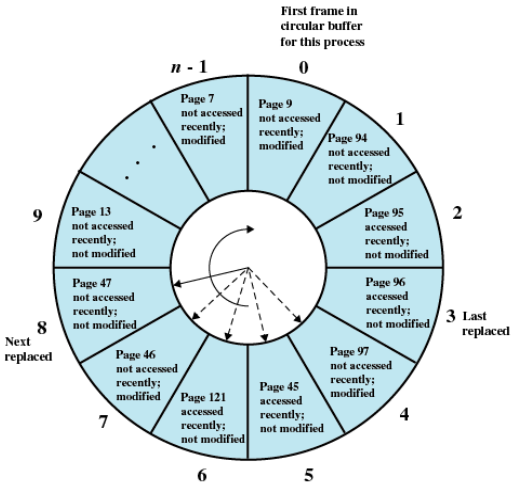


Figure 8.18 The Clock Page-Replacement Algorithm [GOLD89]

Page Buffering

- Page buffering improves paging performance and allows the use of a simpler page replacement policy.
- A replaced page is not lost, but rather assigned to one of two lists:
  - Free page list if page has not been modified: list of page frames available for reading in pages
  - Modified page list: pages are written out in clusters

6.7 Operating System Software For VM

5 Resident Set Management

- The OS must decide how many pages to bring into main memory(Resident Set, 驻留集)
  - the smaller the amount of memory allocated to each process, the more processes can reside in memory
  - small number of pages loaded increases page faults
  - beyond a certain size, further allocations of pages will not effect the page fault rate

Resident Set Size

- Fixed-allocation
  - Gives a process a fixed number of pages within which to execute
  - When a page fault occurs, one of the pages of that process must be replaced
- Variable-allocation
  - The number of pages allocated to a process varies over the lifetime of the process

Replacement Scope

- Local: chooses only among the resident pages of the process that generated the page fault
- Global: considers all unlocked pages in main memory

Resident Set Management

- Fixed Allocation, Local Scope: Decide ahead of time the amount of allocation to give a process
  - If allocation is too small, there will be a high page fault rate
  - If allocation is too large there will be too few programs in main memory
- Variable Allocation,Global Scope: Operating system keeps list of free frames. Free frame is added to resident set of process when a page fault occurs.If no free frame, replaces one from another process
  - Easiest to implement
  - Adopted by many operating systems

Resident Set Management.

- Variable Allocation,Local Scope
  - When new process added, allocate number of page frames based on application type, program request, or other criteria.
  - When page fault occurs, select page from among the resident set of the process that suffers the fault
  - From time to time, reevaluate the allocation provided to the process, and increase or decrease it to improve overall performance

6.7 Operating System Software For VM

6 Cleaning Policy

- Demand cleaning: A page is written out only when it has been selected for replacement
- Precleaning: Pages are written out in batches
- Best approach incorporates page buffering
  - Replaced pages are placed in two lists: Modified and unmodified
  - Pages in the modified list are periodically written out in batches
  - Pages in the unmodified list are either reclaimed if referenced again or lost when its frame is assigned to another page

6.7 Operating System Software For VM

7 Load Control

- **Load Control**(负载控制) determines the number of processes that will be resident in main memory
- Too few processes, many occasions when all processes will be blocked and much time will be spent in swapping
- Too many processes will lead to thrashing

Multiprogramming Effects

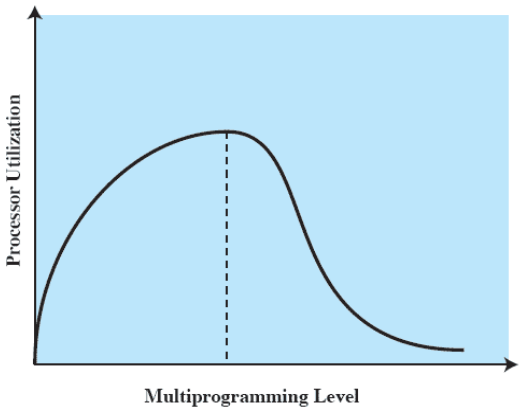


Figure 8.21 Multiprogramming Effects

Process Suspension

- If the degree of multiprogramming is to be reduced, one or more of the currently resident processes must be swapped out
  - Lowest priority process
  - Faulting process: This process does not have its working set in main memory so it will be blocked anyway
  - Last process activated: This process is least likely to have its working set resident
  - Process with smallest resident set: This process requires the least future effort to reload
  - Largest process: Obtains the most free frames
  - Process with the largest remaining execution window

6.8 Linux Memory Management

Virtual Memory Addressing

- Three-level page table structure
  - Page directory
  - Page middle directory
  - Page table
- Page Allocation: buddy system is used
- Page Replacement Algorithm: Based on the Clock Algorithm.
  - use 8-bit age variable to record the number of accessed time.

Address Translation in Linux

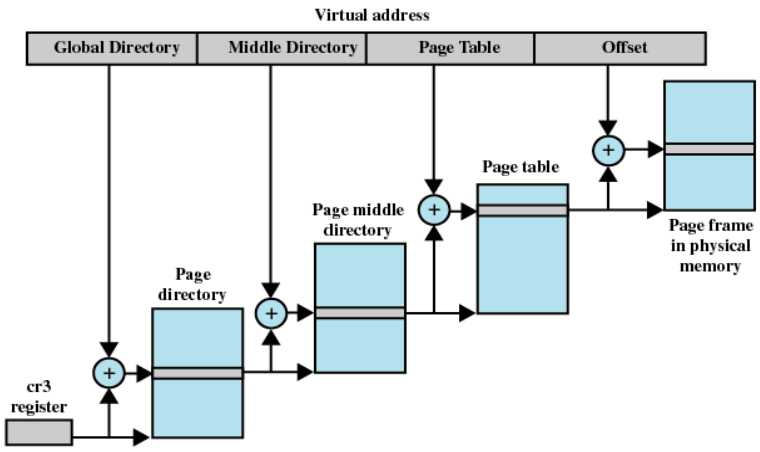


Figure 8.25 Address Translation in Linux Virtual Memory Scheme

Terminology

- Relocation
  - static relocation; dynamic relocation
- logical organization
- physical organization
- fixed partitioning
- dynamic partitioning
  - best-fit; first-fit; next-fit
  - compaction
- internal fragmentation; external fragmentation
- Buddy System
- logical address; relative address; physical address
- overlay

Terminology

- Locality
- virtual memory
- virtual address; virtual address space
- address space; real address
- paging
  - page; frames; page table; page fault;
  - demand paging; prepaging;
  - resident set; working set;
  - thrashing
  - translation look-aside buffer(TLB)
- segmentation
  - segment; segment table

Terminology

- fetch policy
- placement policy
- replacement policy
  - Optimal policy
  - Least Recently Used(LRU)
  - First-In-First-Out(FIFO)
  - Clock policy
  - Clock Page-Replacement Algorithm