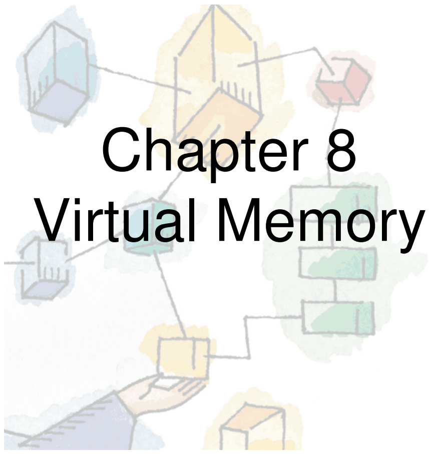
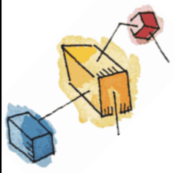
Operating Systems:
Internals and Design Principles, 6/E
William Stallings



Chapter 8
Virtual Memory

Dave Bremer
Otago Polytechnic, N.Z.



Roadmap

→ Hardware and Control Structures
- Operating System Software
- UNIX and Solaris Memory Management
- Linux Memory Management
- Windows Memory Management

Terminology

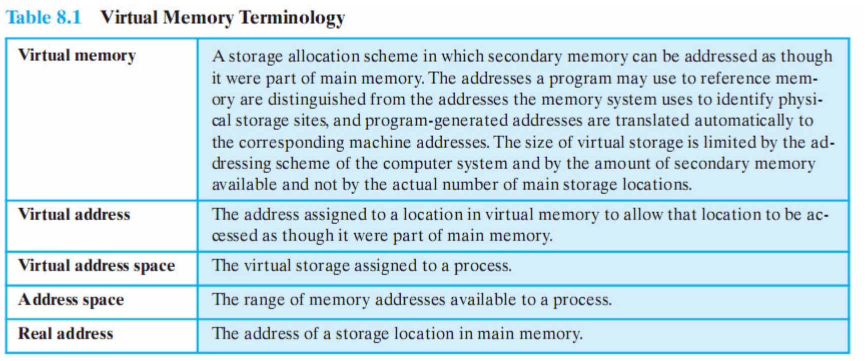


Table 8.1 **Virtual Memory Terminology**

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.

Key points in Memory Management

1) Memory references are logical addresses dynamically translated into physical addresses at run time
   – A process may be swapped in and out of main memory occupying different regions at different times during execution
2) A process may be broken up into pieces that do not need to located contiguously in main memory

Breakthrough in Memory Management

- **If both** of those two characteristics are present,
  - then it is not necessary that all of the pages or all of the segments of a process be in main memory during execution.
- If the next instruction, and the next data location are in memory then execution can proceed
  - at least for a time

Execution of a Process

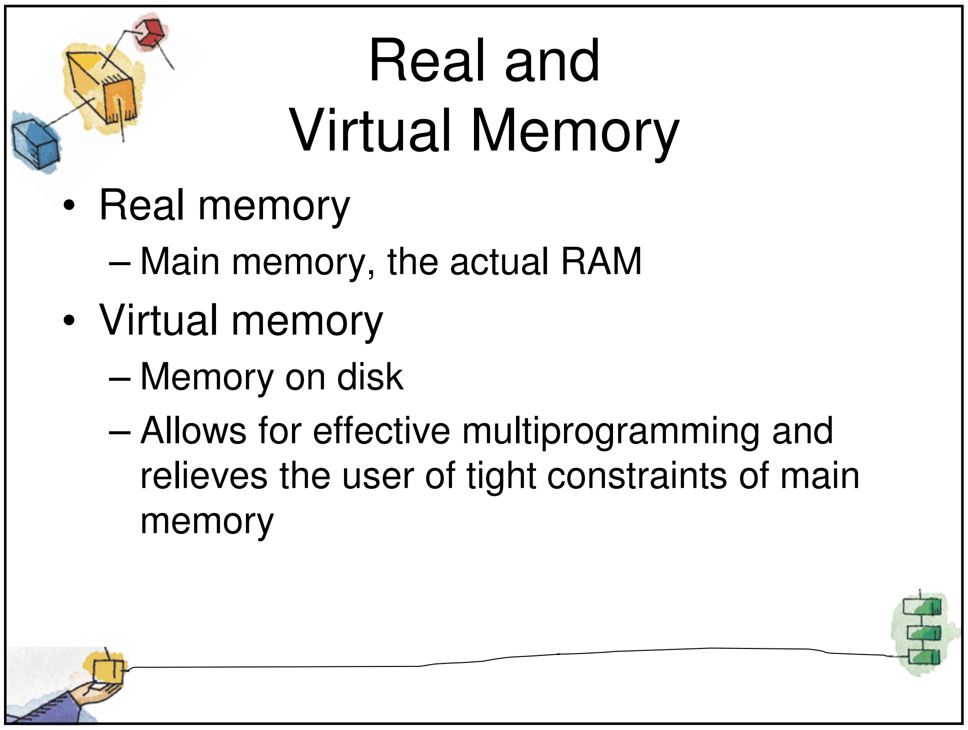
- Operating system brings into main memory a few pieces of the program
- Resident set - portion of process that is in main memory
- An interrupt is generated when an address is needed that is not in main memory
- Operating system places the process in a blocking state

Execution of a Process

- Piece of process that contains the logical address is brought into main memory
  - Operating system 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 operating system to place the affected process in the Ready state

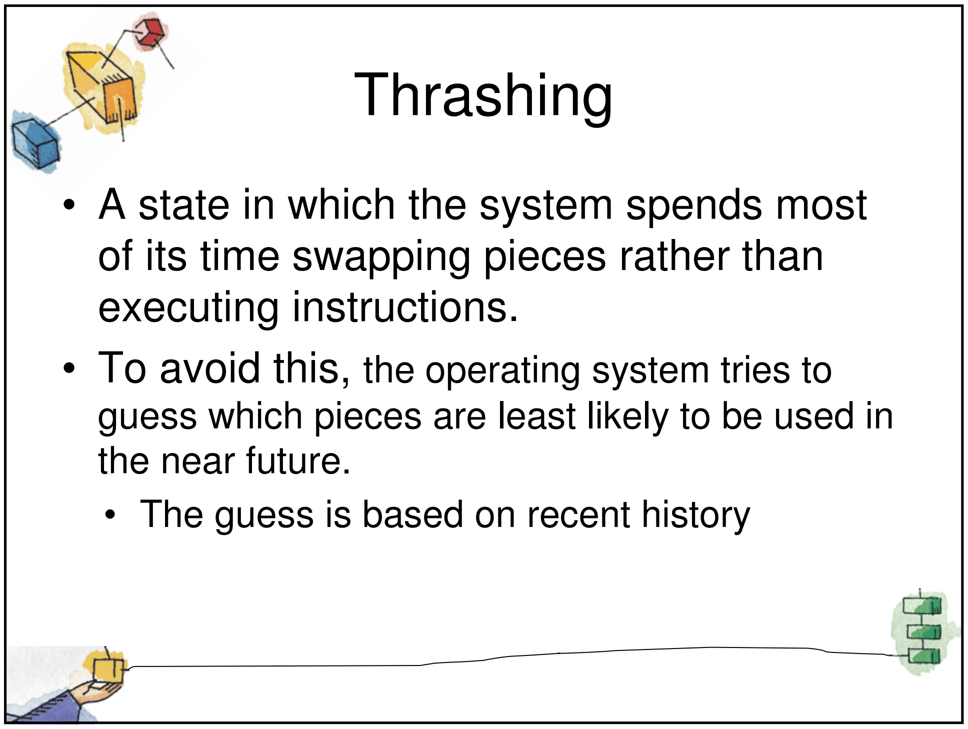
Implications of this new strategy

- 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



Real and Virtual Memory

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



Thrashing

- A state in which the system spends most of its time swapping pieces rather than executing instructions.
- To avoid this, the operating system tries to guess which pieces are least likely to be used in the near future.
  - The guess is based on recent history

Principle of Locality

- 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
- Therefore it is possible to make intelligent guesses about which pieces will be needed in the future
- This suggests that virtual memory may work efficiently

A Processes Performance in VM Environment

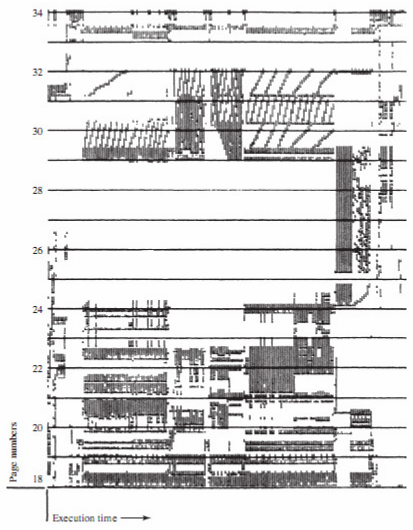


Figure 8.1 Paging Behavior

- Note that during the lifetime of the process, references are confined to a subset of pages.

Support Needed for Virtual Memory

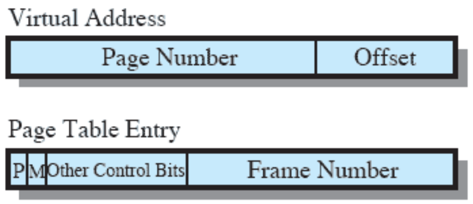
- Hardware must support paging and segmentation
- Operating system must be able to manage the movement of pages and/or segments between secondary memory and main memory

Paging

- Each process has its own page table
- Each page table entry contains the frame number of the corresponding page in main memory
- Two extra bits are needed to indicate:
  - whether the page is in main memory or not
  - Whether the contents of the page has been altered since it was last loaded

(see next slide)

Paging Table



(a) Paging only

Address Translation

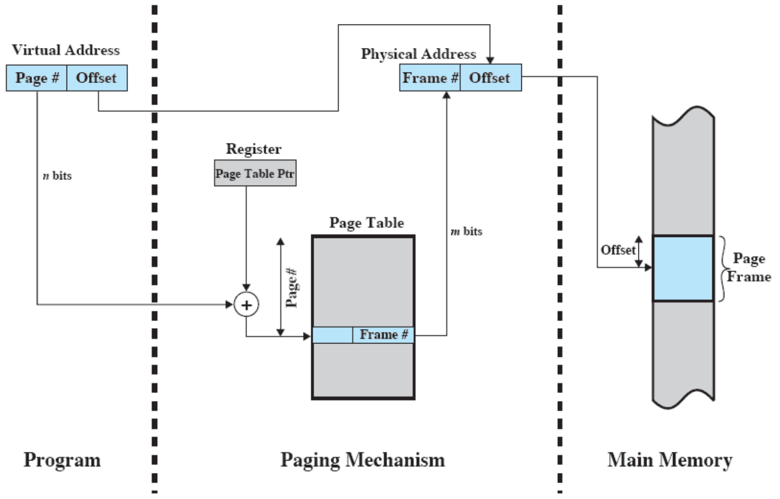


Figure 8.3 Address Translation in a Paging System

Page Tables

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

Two-Level Hierarchical Page Table

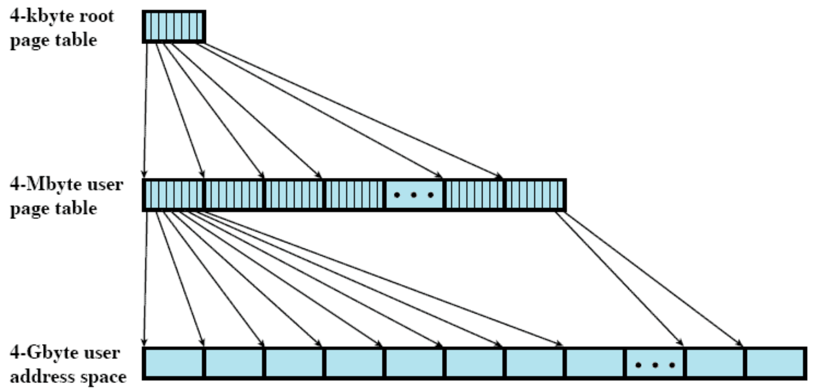


Figure 8.4 A Two-Level Hierarchical Page Table

Address Translation for Hierarchical page table

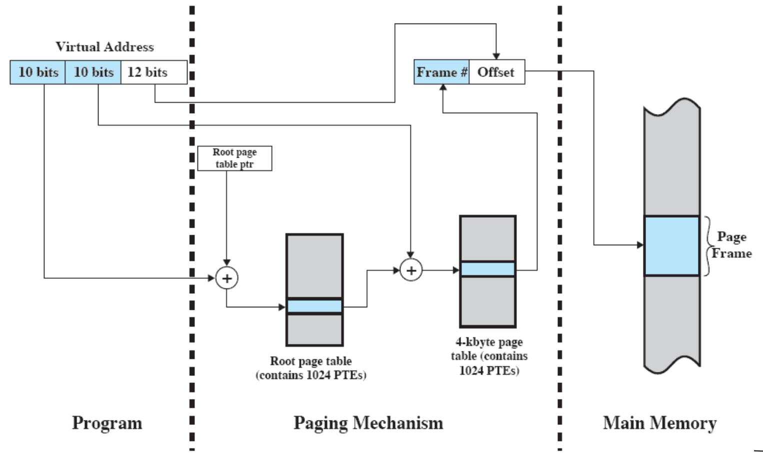


Figure 8.5 Address Translation in a Two-Level Paging System

Page tables grow proportionally

- A drawback of the type of page tables just discussed is that their size is proportional to that of the virtual address space.
- An alternative is Inverted Page Tables

Inverted Page Table

- Used on PowerPC, UltraSPARC, and IA-64 architecture
- Page number portion of a virtual address is mapped into a hash value
- Hash value points to inverted page table
- Fixed proportion of real memory is required for the tables regardless of the number of processes

Inverted Page Table

Each entry in the page table includes:

- Page number
- Process identifier
  - The process that owns this page.
- Control bits
  - includes flags, such as valid, referenced, etc
- Chain pointer
  - the index value of the next entry in the chain.

Inverted Page Table

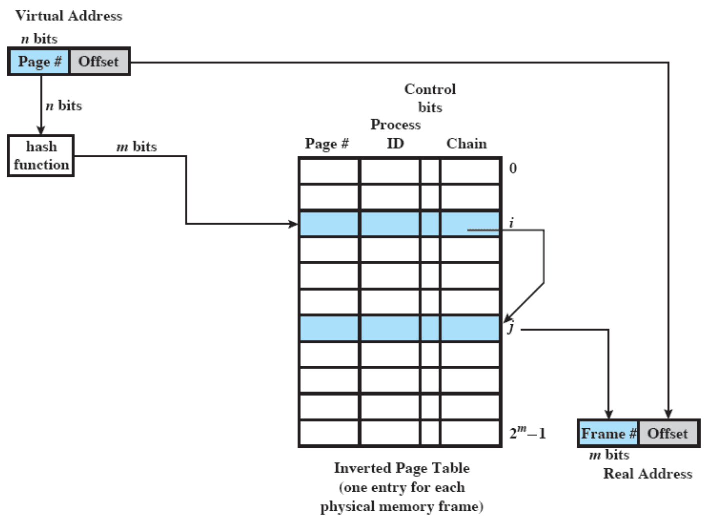


Figure 8.6 Inverted Page Table Structure

Translation Lookaside Buffer

- 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)
  - Contains page table entries that have been most recently used

TLB Operation

- 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

Looking into 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

Translation Lookaside Buffer

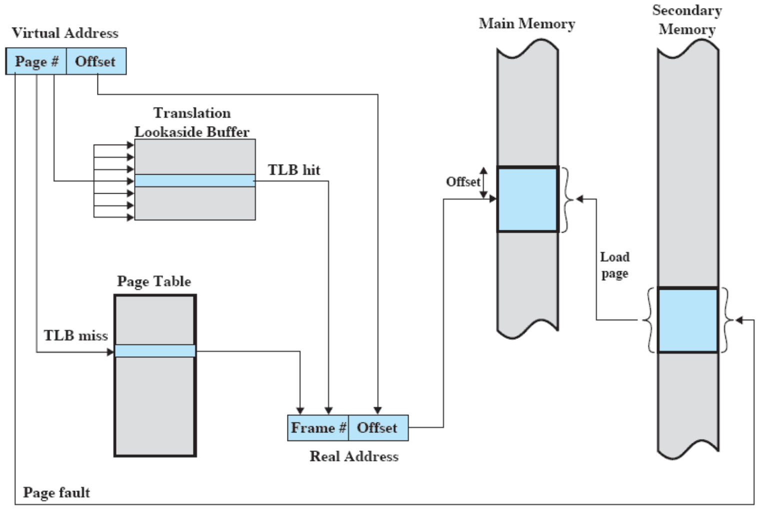


Figure 8.7 Use of a Translation Lookaside Buffer

TLB operation

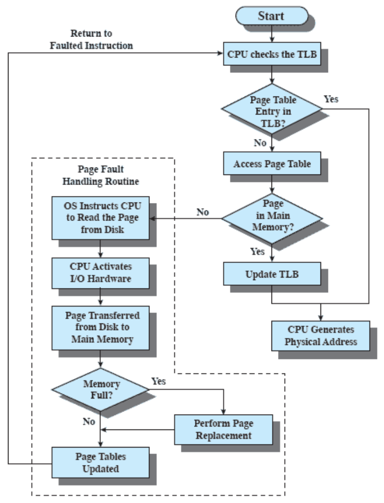

Start
Return to Faulted Instruction
CPU checks the TLB
Page Table Entry in TLB?
Yes
No
Access Page Table
Page Fault Handling Routine
OS Instructs CPU to Read the Page from Disk
No
Page in Main Memory?
Yes
Update TLB
CPU Activates I/O Hardware
Page Transferred from Disk to Main Memory
CPU Generates Physical Address
Memory Full?
Yes
No
Perform Page Replacement
Page Tables Updated

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

Associative Mapping

- As the TLB only contains some of the page table entries we cannot simply index into the TLB based on the page number
  - Each TLB entry must include the page number as well as the complete page table entry
- The process is able to simultaneously query numerous TLB entries to determine if there is a page number match

Translation Lookaside Buffer

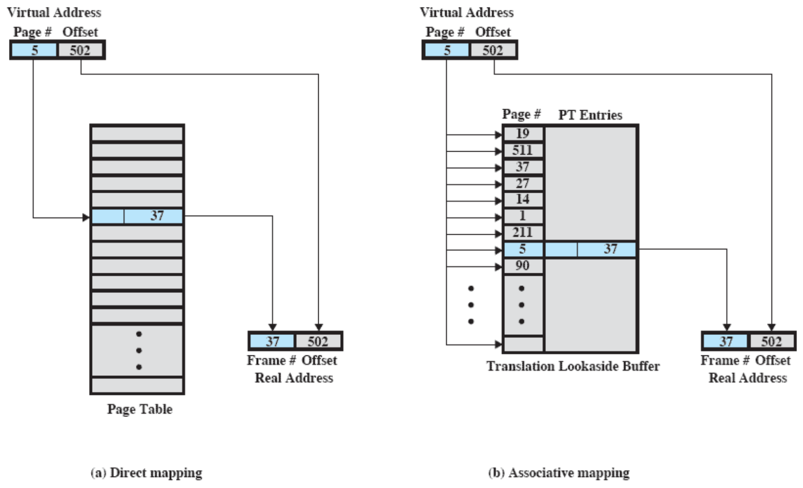


Figure 8.9 Direct Versus Associative Lookup for Page Table Entries

TLB and Cache Operation

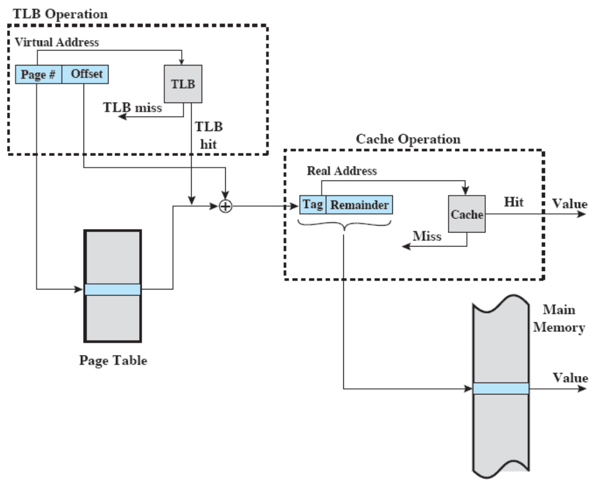


Figure 8.10 Translation Lookaside Buffer and Cache Operation

Page Size

- Smaller page size, less amount of internal fragmentation
- But 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

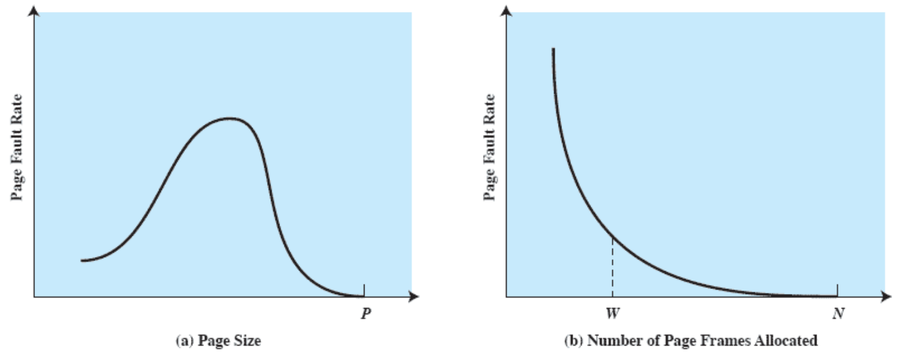
Page Size

- Secondary memory is designed to efficiently transfer large blocks of data so a large page size is better

Further complications to Page Size

- 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.

Page Size



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 Size

Table 8.3 Example Page Sizes

Computer	Page Size
Atlas	512 48-bit words
Honeywell-Multics	1024 36-bit word
IBM 370/XA and 370/ESA	4 Kbytes
VAX family	512 bytes
IBM AS/400	512 bytes
DEC Alpha	8 Kbytes
MIPS	4 Kbyes 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

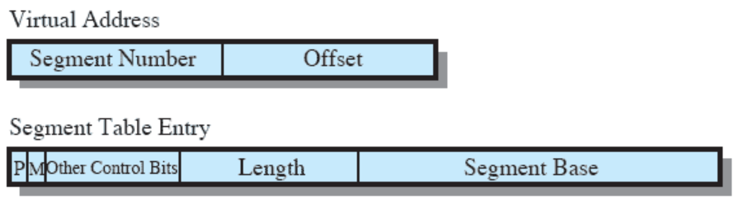
Segmentation

- Segmentation allows the programmer to view memory as consisting of multiple address spaces or segments.
  - May be unequal, dynamic size
  - 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 Organization

- Starting address corresponding segment in main memory
- Each entry contains the length of the segment
- A bit is needed to determine if 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

Segment Table Entries



(b) Segmentation only

Address Translation in Segmentation

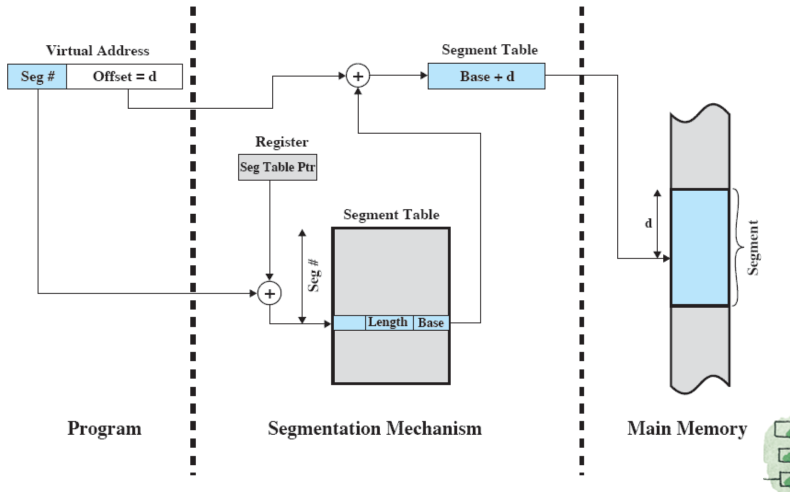
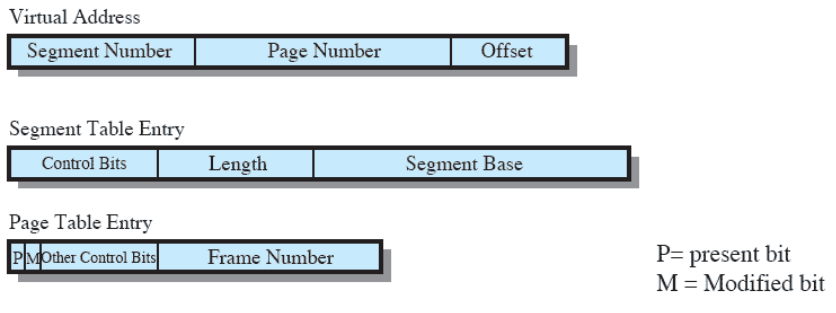


Figure 8.12 Address Translation in a Segmentation System

Combined Paging and Segmentation

- Paging is transparent to the programmer
- Segmentation is visible to the programmer
- Each segment is broken into fixed-size pages

Combined Paging and Segmentation



(c) Combined segmentation and paging

Address Translation

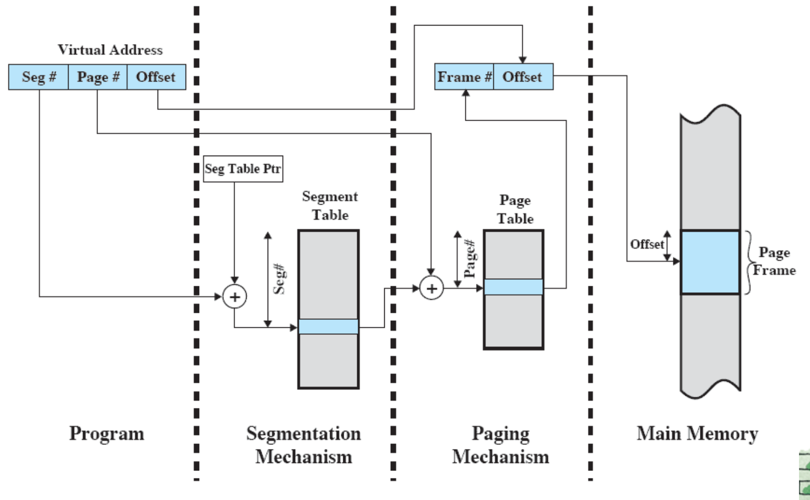


Figure 8.13 Address Translation in a Segmentation/Paging System

Protection and sharing

- Segmentation lends itself to the implementation of protection and sharing policies.
- As each entry has a base address and length, inadvertent memory access can be controlled
- Sharing can be achieved by segments referencing multiple processes

Protection Relationships

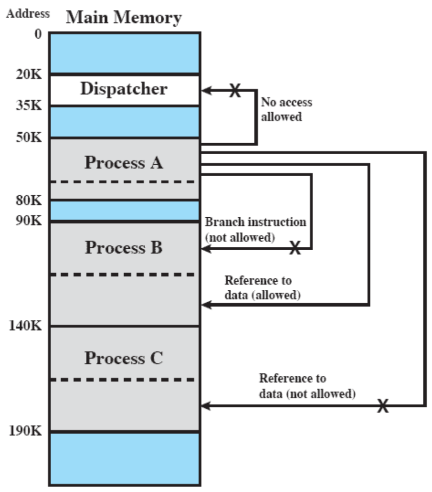


Figure 8.14 Protection Relationships Between Segments

Roadmap

- Hardware and Control Structures
- Operating System Software
- UNIX and Solaris Memory Management
- Linux Memory Management
- Windows Memory Management

Memory Management Decisions

- 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

Key Design Elements

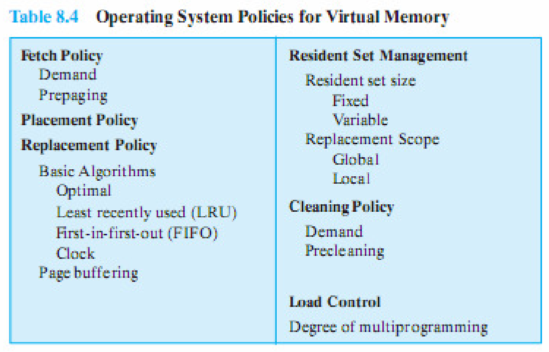


Table 8.4 Operating System Policies for Virtual Memory

Fetch Policy	**Resident Set Management**
Demand	Resident set size
Prepaging	Fixed
Placement Policy	Variable
Replacement Policy	Replacement Scope
Basic Algorithms	Global
Optimal	Local
Least recently used (LRU)	**Cleaning Policy**
First-in-first-out (FIFO)	Demand
Clock	Precleaning
Page buffering	
	Load Control
	Degree of multiprogramming

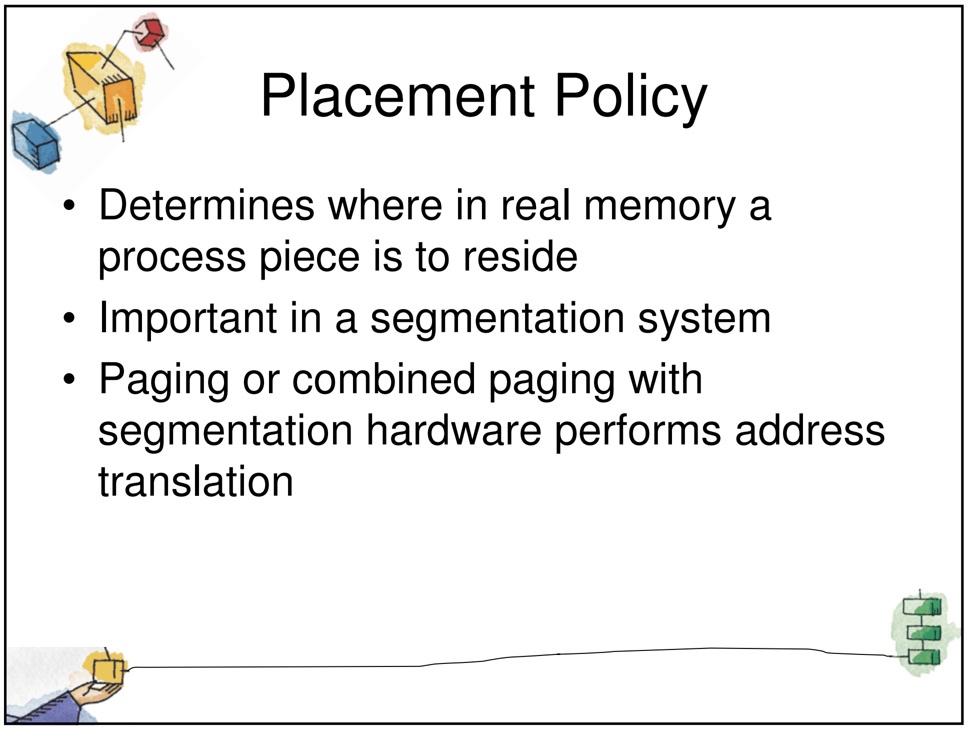
- Key aim: Minimise page faults
  - No definitive best policy

Fetch Policy

- Determines when a page should be brought into memory
- Two main types:
  - Demand Paging
  - Prepaging

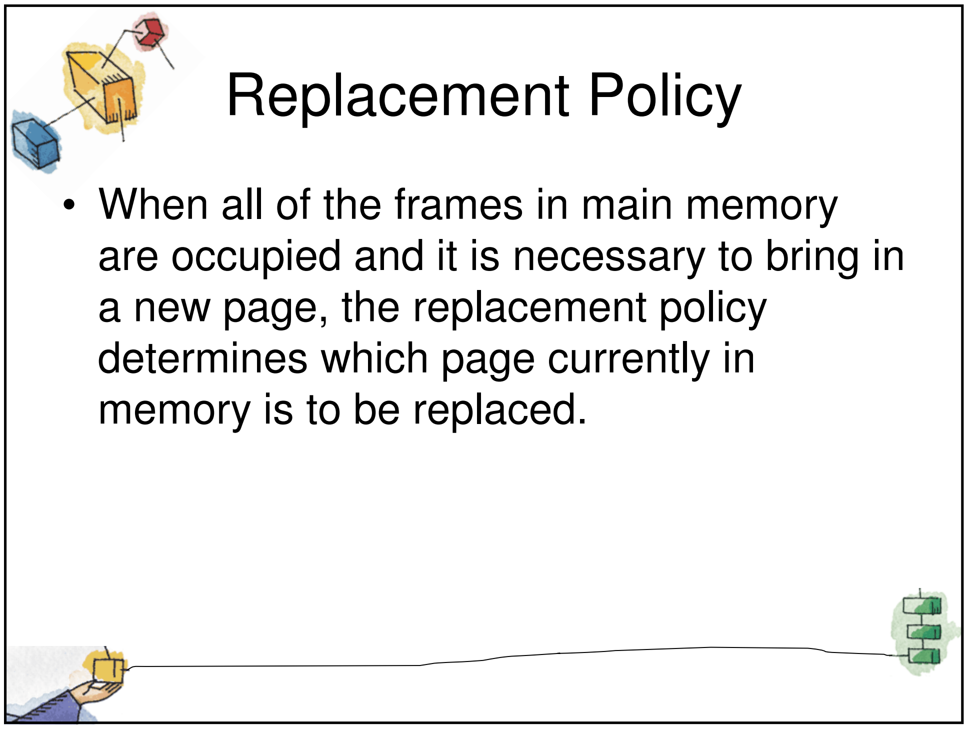
Demand Paging and Prepaging

- **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
- **Prepaging**
  - brings in more pages than needed
  - More efficient to bring in pages that reside contiguously on the disk
  - Don’t confuse with “swapping”



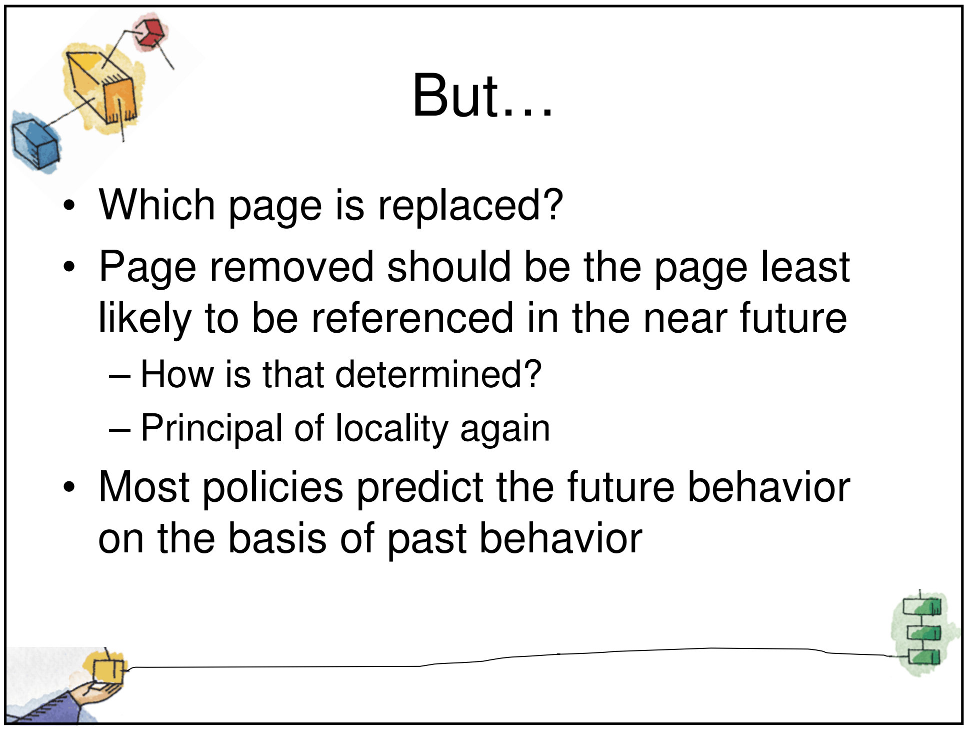
Placement Policy

- Determines where in real memory a process piece is to reside
- Important in a segmentation system
- Paging or combined paging with segmentation hardware performs address translation



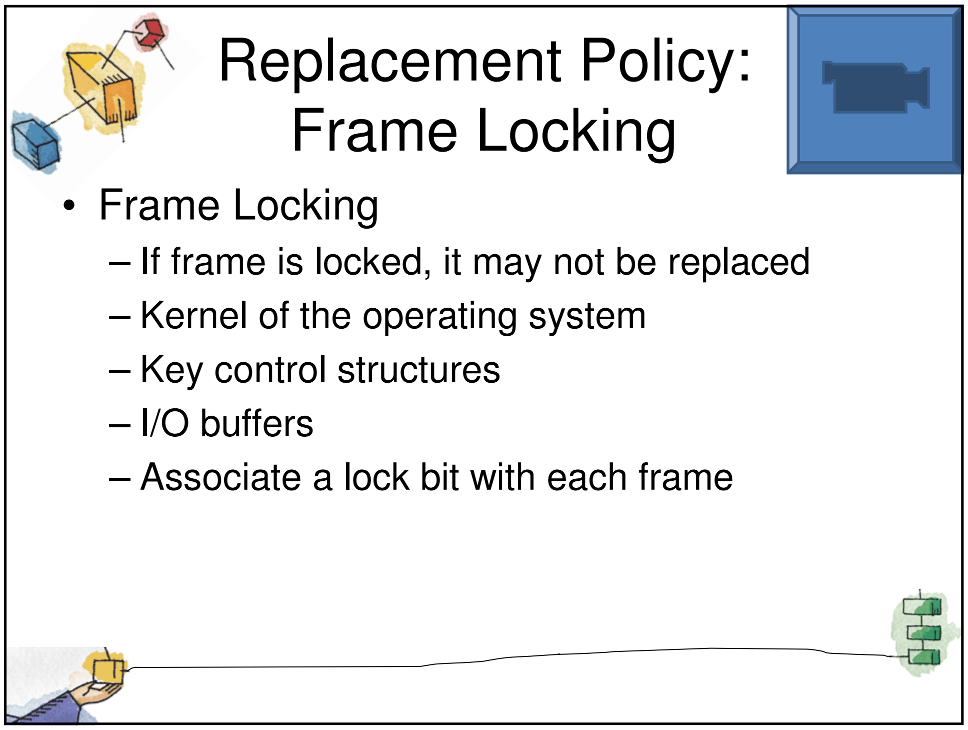
Replacement Policy

- When all of the frames in main memory are occupied and it is necessary to bring in a new page, the replacement policy determines which page currently in memory is to be replaced.



But…

- Which page is replaced?
- Page removed should be the page least likely to be referenced in the near future
  - How is that determined?
  - Principal of locality again
- Most policies predict the future behavior on the basis of past behavior



Replacement Policy: Frame Locking

- Frame Locking
  - If frame is locked, it may not be replaced
  - Kernel of the operating system
  - Key control structures
  - I/O buffers
  - Associate a lock bit with each frame

Basic Replacement Algorithms

- There are certain basic algorithms that are used for the selection of a page to replace, they include
  - Optimal
  - Least recently used (LRU)
  - First-in-first-out (FIFO)
  - Clock
- Examples

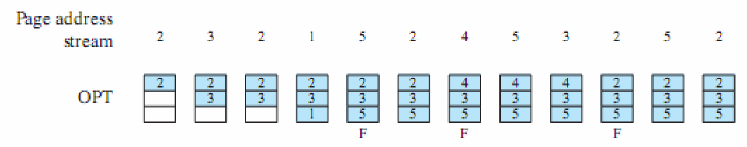
Examples

- An example of the implementation of these policies will use a page address stream formed by executing the program is
  - 2 3 2 1 5 2 4 5 3 2 5 2
- Which means that the first page referenced is 2,
  - the second page referenced is 3,
  - And so on.

Optimal policy

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

Optimal Policy Example



Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2	2	2	2	2	4	4	4	2	2	2
		3	3	3	3	3	3	3	3	3	3	3
				1	5	5	5	5	5	5	5	5
					F		F			F		

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

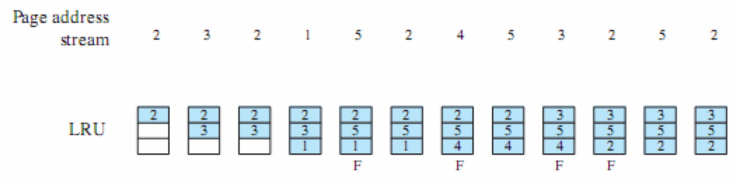
Figure 8.15 Behavior of Four Page Replacement Algorithms

- The optimal policy produces three page faults after the frame allocation has been filled.

Least Recently Used (LRU)

- Replaces the page that has not been referenced for the longest time
- By the principle of locality, this should be the page least likely to be referenced in the near future
- Difficult to implement
  - One approach is to tag each page with the time of last reference.
  - This requires a great deal of overhead.

LRU Example



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

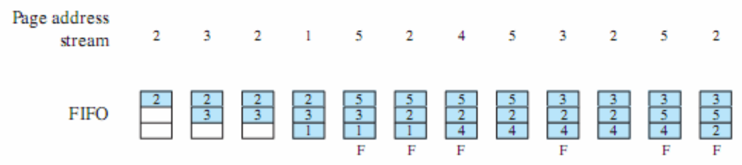
Figure 8.15 **Behavior of Four Page Replacement Algorithms**

- The LRU policy does nearly as well as the optimal policy.
  - In this example, there are four page faults

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
  - But, these pages may be needed again very soon if it hasn't truly fallen out of use

FIFO Example



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

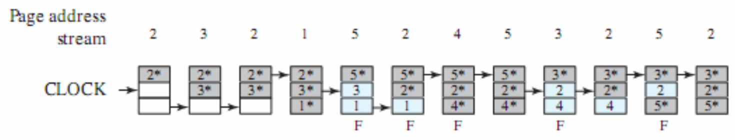
Figure 8.15 Behavior of Four Page Replacement Algorithms

- The FIFO policy results in six page faults.
  - Note that LRU recognizes that pages 2 and 5 are referenced more frequently than other pages, whereas FIFO does not.

Clock Policy

- Uses and additional bit called a “use bit”
- When a page is first loaded in memory or referenced, the use bit is set to 1
- When it is time to replace a page, the OS scans the set flipping all 1’s to 0
- The first frame encountered with the use bit already set to 0 is replaced.

Clock Policy Example

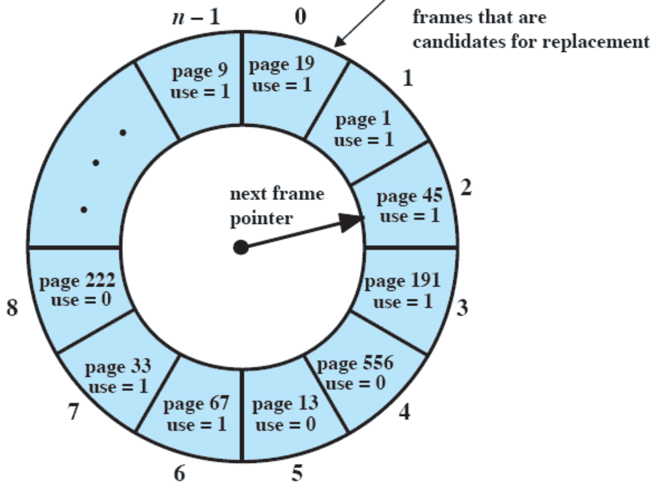


F= page fault occurring after the frame allocation is initially filled

Figure 8.15 **Behavior of Four Page Replacement Algorithms**

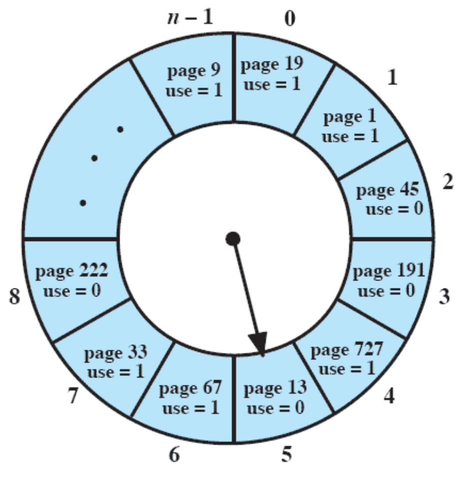
- Note that the clock policy is adept at protecting frames 2 and 5 from replacement.

Clock Policy



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

Clock Policy



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

Figure 8.16 Example of Clock Policy Operation

Clock Policy

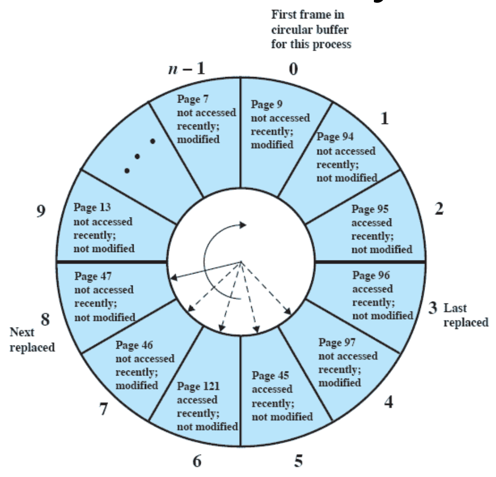
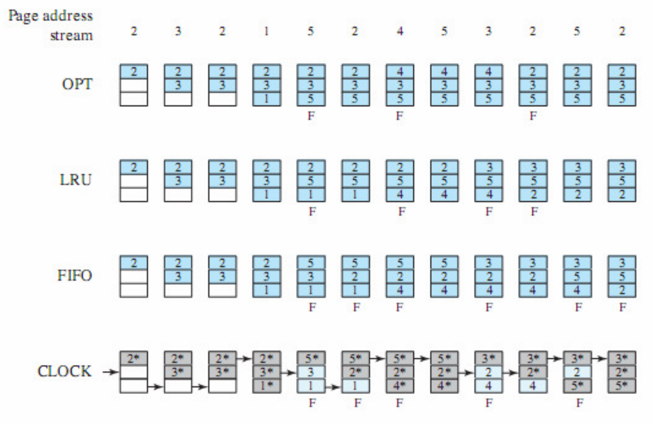

First frame in circular buffer for this process
0
1
2
3 Last replaced
4
5
6
7
8 Next replaced
9
n – 1
Page 9 not accessed recently; modified
Page 94 not accessed recently; not modified
Page 95 accessed recently; not modified
Page 96 accessed recently; not modified
Page 97 not accessed recently; modified
Page 45 accessed recently; not modified
Page 121 accessed recently; not modified
Page 46 not accessed recently; modified
Page 47 not accessed recently; not modified
Page 13 not accessed recently; not modified
Page 7 not accessed recently; modified

Figure 8.18 The Clock Page-Replacement Algorithm [GOLD89]

Combined Examples



Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2 3	2 3	2 3 1	2 3 5 F	2 3 5	4 3 5 F	4 3 5	4 3 5	2 3 5 F	2 3 5	2 3 5
LRU	2	2 3	2 3	2 3 1	2 5 1 F	2 5 1	2 5 4 F	2 5 4	3 5 4 F	3 5 2 F	3 5 2	3 5 2
FIFO	2	2 3	2 3	2 3 1	5 3 1 F	5 2 1 F	5 2 4 F	5 2 4	3 2 4 F	3 2 4	3 5 4 F	3 5 2 F
CLOCK	2*	2* 3*	2* 3*	2* 3* 1*	5* 3 1 F	5* 2* 1 F	5* 2* 4* F	5* 2* 4*	3* 2 4 F	3* 2* 4	3* 2 5* F	3* 2* 5*

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

Figure 8.15 Behavior of Four Page Replacement Algorithms

Comparison

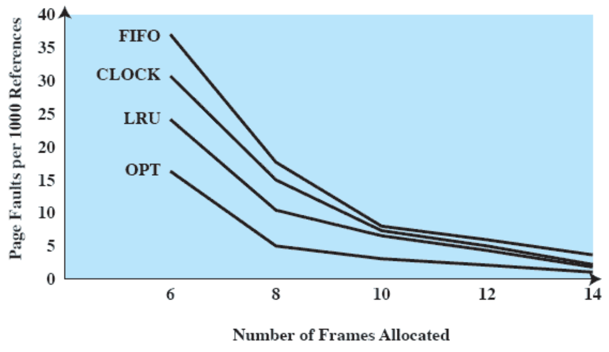


Figure 8.17 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

Page Buffering

- LRU and Clock policies both involve complexity and overhead
  - Also, replacing a modified page is more costly than unmodified as needs written to secondary memory
- Solution: Replaced page is added to one of two lists
  - Free page list if page has not been modified
  - Modified page list

Replacement Policy and Cache Size

- Main memory size is getting larger and the locality of applications is decreasing.
  - So, cache sizes have been increasing
- With large caches, replacement of pages can have a performance impact
  - improve performance by supplementing the page replacement policy with a with a policy for page placement in the page buffer

Resident Set Management

- The OS must decide how many pages to bring into main memory
  - The smaller the amount of memory allocated to each process, the more processes that can reside in memory.
  - Small number of pages loaded increases page faults.
  - Beyond a certain size, further allocations of pages will not affect 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
  - Number of pages allocated to a process varies over the lifetime of the process

Replacement Scope

- The scope of a replacement strategy can be categorized as *global* or *local*.
  - Both types are activated by a page fault when there are no free page frames.
  - A local replacement policy chooses only among the resident pages of the process that generated the page fault
  - A global replacement policy considers all unlocked pages in main memory

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
  - Increased processor idle time or
  - Increased swapping.

Variable Allocation, Global Scope

- Easiest to implement
  - Adopted by many operating systems
- 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
  - Therein lies the difficulty … which to replace.

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
- Reevaluate allocation from time to time

Resident Set Management Summary

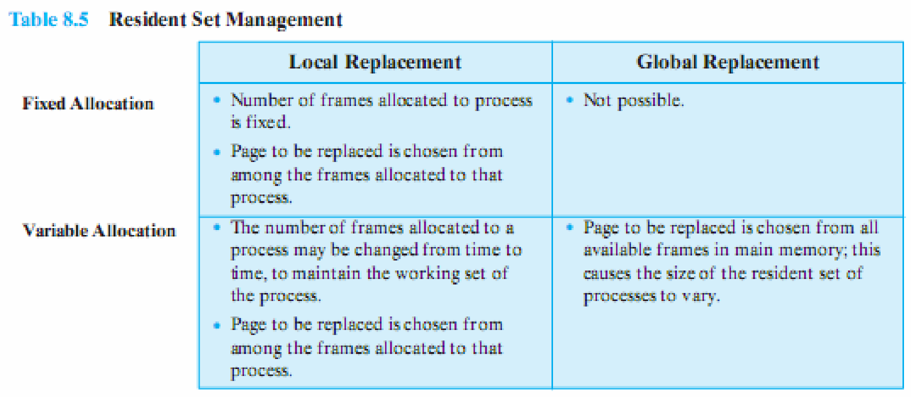
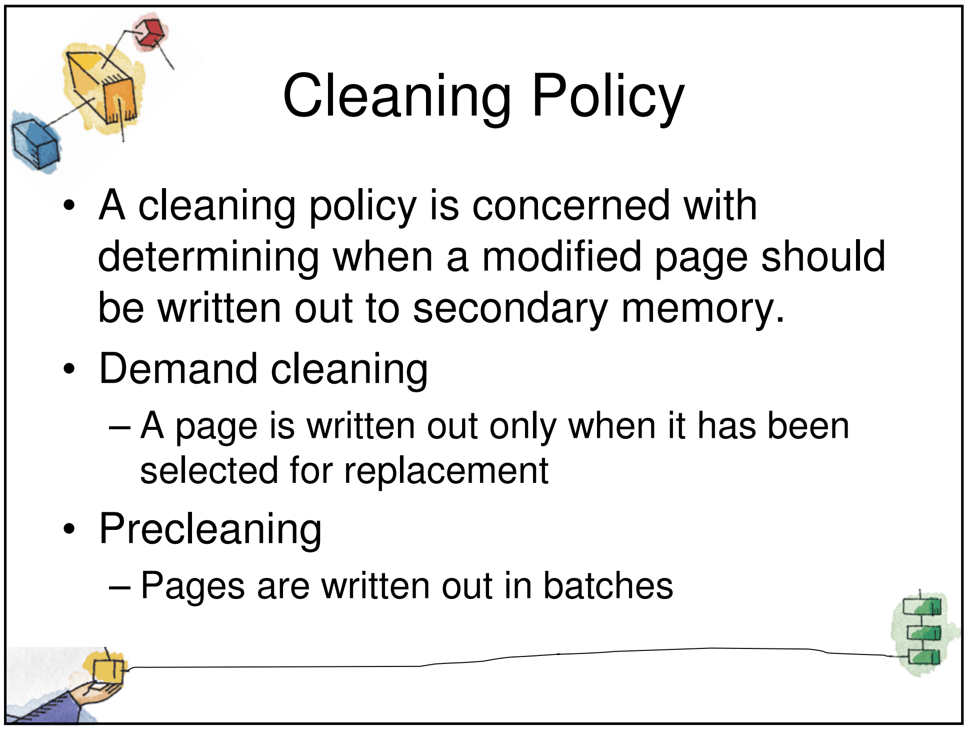


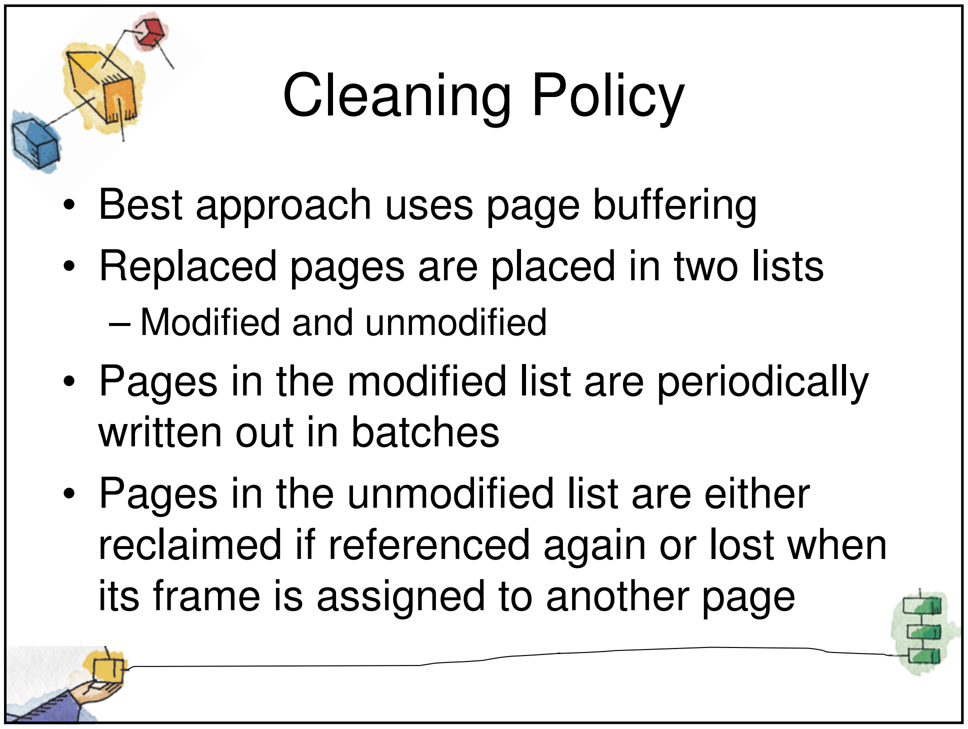
Table 8.5 **Resident Set Management**

	Local Replacement	**Global Replacement**
Fixed Allocation	• Number of frames allocated to process is fixed. • Page to be replaced is chosen from among the frames allocated to that process.	• Not possible.
Variable Allocation	• The number of frames allocated to a process may be changed from time to time, to maintain the working set of the process. • Page to be replaced is chosen from among the frames allocated to that process.	• Page to be replaced is chosen from all available frames in main memory; this causes the size of the resident set of processes to vary.



Cleaning Policy

- A cleaning policy is concerned with determining when a modified page should be written out to secondary memory.
- Demand cleaning
  - A page is written out only when it has been selected for replacement
- Precleaning
  - Pages are written out in batches



Cleaning Policy

- Best approach uses 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

Load Control

- Determines the number of processes that will be resident in main memory
  - The *multiprogramming* level
- 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

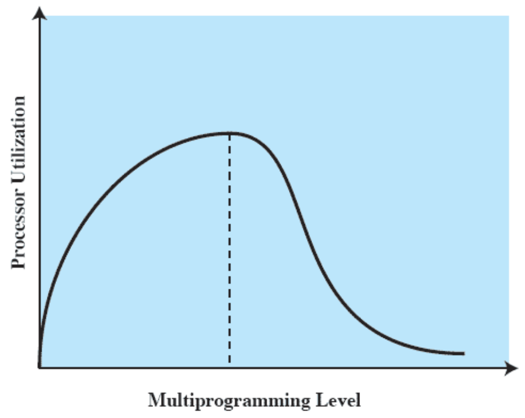


Figure 8.21 Multiprogramming Effects

Process Suspension

- If the degree of multiprogramming is to be reduced, one or more of the currently resident processes must be suspended (swapped out).
- Six possibilities exist…

Suspension policies

- 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

Suspension policies cont.

- 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

Roadmap

- Hardware and Control Structures
- Operating System Software
- UNIX and Solaris Memory Management
- Linux Memory Management
- Windows Memory Management

Unix

- Intended to be machine independent so implementations vary
  - Early Unix: variable partitioning with no virtual memory to paged
  - Recent Unix (SVR4 & Solaris) using paged virtual memory
- SVR4 uses two separate schemes:
  - *Paging system* and a *kernel memory allocator*.

Paging System and Kernel Memory Allocator

- Paging system provides a virtual memory capability that allocates page frames in main memory to processes
  - Also allocates page frames to disk block buffers.
- Kernel Memory Allocator allocates memory for the kernel
  - The paging system is less suited for this task

Paged VM Data Structures

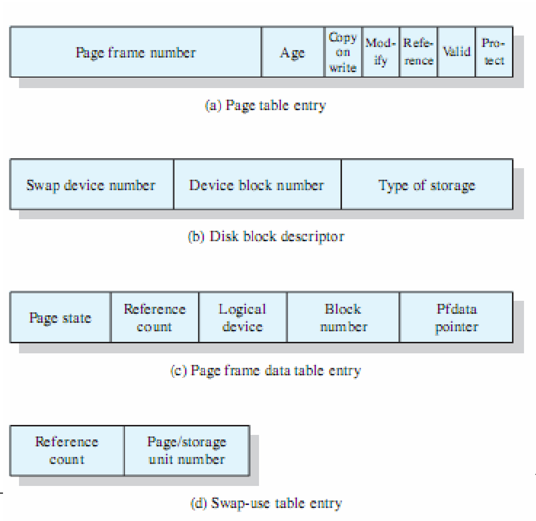
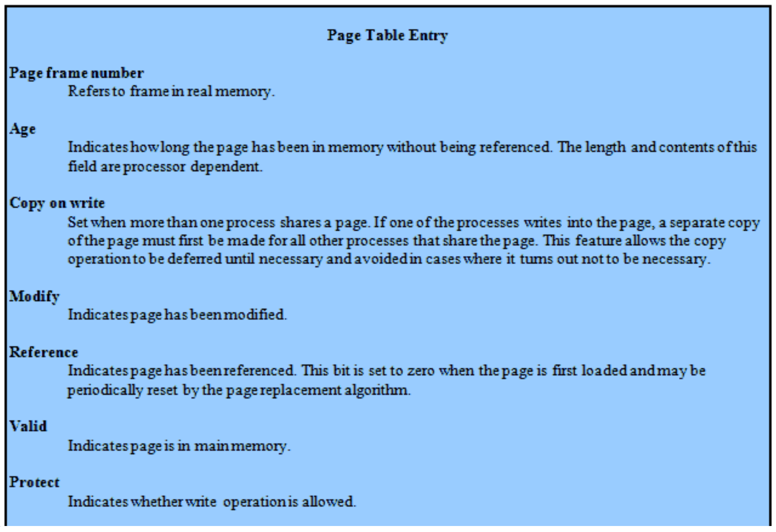


Figure 8.22 **UNIX SVR4 Memory Management Formats**

Page Table Entry Fields



Page Table Entry

Page frame number
Refers to frame in real memory.

Age
Indicates how long the page has been in memory without being referenced. The length and contents of this field are processor dependent.

Copy on write
Set when more than one process shares a page. If one of the processes writes into the page, a separate copy of the page must first be made for all other processes that share the page. This feature allows the copy operation to be deferred until necessary and avoided in cases where it turns out not to be necessary.

Modify
Indicates page has been modified.

Reference
Indicates page has been referenced. This bit is set to zero when the page is first loaded and may be periodically reset by the page replacement algorithm.

Valid
Indicates page is in main memory.

Protect
Indicates whether write operation is allowed.

Disk Block Descriptor Fields

Disk Block Descriptor

Swap device number
Logical device number of the secondary device that holds the corresponding page. This allows more than one device to be used for swapping.

Device block number
Block location of page on swap device.

Type of storage
Storage may be swap unit or executable file. In the latter case, there is an indication as to whether the virtual memory to be allocated should be cleared first.

Page Frame and Swap Use fields

Page Frame Data Table Entry

Page State
Indicates whether this frame is available or has an associated page. In the latter case, the status of the page is specified: on swap device, in executable file, or DMA in progress.

Reference count
Number of processes that reference the page.

Logical device
Logical device that contains a copy of the page.

Block number
Block location of the page copy on the logical device.

Pfdata pointer
Pointer to other pfdata table entries on a list of free pages and on a hash queue of pages.

Swap-Use Table Entry

Reference count
Number of page table entries that point to a page on the swap device.

Page/storage unit number
Page identifier on storage unit.

Page Replacement

- The page frame data table is used for page replacement
- Pointers used to create several lists within the table
  - Free frame list
  - When the number of free frames drops below a threshold, the kernel will steal a number of frames to compensate.

"Two Handed" Clock Page Replacement

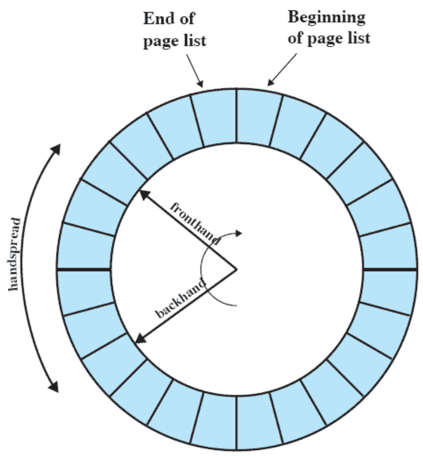


Figure 8.23 Two-Handed Clock Page-Replacement Algorithm

Parameters for Two Handed Clock

- Scanrate:
  - The rate at which the two hands scan through the page list, in pages per second
- Handspread:
  - The gap between fronthand and backhand
- Both have defaults set at boot time based on physical memory

Kernel Memory Allocator

- The kernel generates and destroys small tables and buffers frequently during the course of execution, each of which requires dynamic memory allocation.
- Most of these blocks significantly smaller than typical pages,
  - Therefore normal paging would be inefficient
- Variation of “buddy system” is used

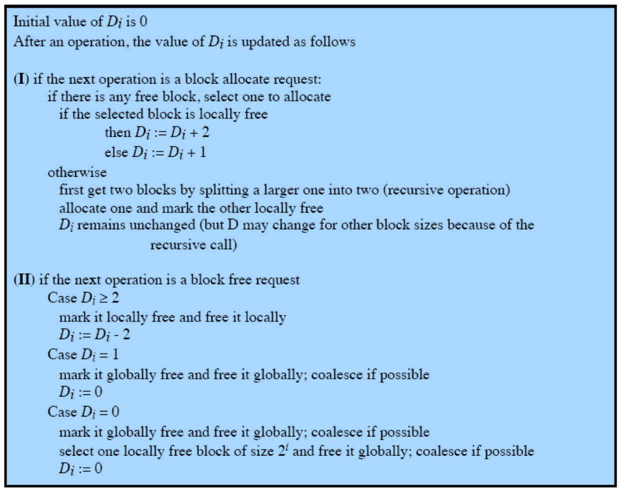
Lazy Buddy

- UNIX often exhibits steady-state behavior in kernel memory demand;
  - i.e. the amount of demand for blocks of a particular size varies slowly in time.
- To avoid unnecessary joining and splitting of blocks,
  - the lazy buddy system defers coalescing until it seems likely that it is needed, and then coalesces as many blocks as possible.

Lazy Buddy System Parameters

- N_i = current number of blocks of size 2^i
- A_i = current number of blocks of size 2^i that are allocated (occupied).
- G_i = current number of blocks of size 2^i that are globally free.
- L_i = current number of blocks of size 2^i that are locally free

Lazy Buddy System Allocator



```
Initial value of $D_i$ is 0
After an operation, the value of $D_i$ is updated as follows

(I) if the next operation is a block allocate request:
      if there is any free block, select one to allocate
        if the selected block is locally free
                then $D_i := D_i + 2$
                else $D_i := D_i + 1$
      otherwise
        first get two blocks by splitting a larger one into two (recursive operation)
        allocate one and mark the other locally free
        $D_i$ remains unchanged (but D may change for other block sizes because of the
                        recursive call)

(II) if the next operation is a block free request
      Case $D_i \geq 2$
        mark it locally free and free it locally
        $D_i := D_i - 2$
      Case $D_i = 1$
        mark it globally free and free it globally; coalesce if possible
        $D_i := 0$
      Case $D_i = 0$
        mark it globally free and free it globally; coalesce if possible
        select one locally free block of size $2^i$ and free it globally; coalesce if possible
        $D_i := 0$
```

Figure 8.24 Lazy Buddy System Algorithm

Linux Memory Management

- Shares many characteristics with Unix
  - But is quite complex
- Two main aspects
  - Process virtual memory, and
  - Kernel memory allocation.

Linux Memory Management

- Page directory
- Page middle directory
- Page table

Linux Virtual Memory

- Three level page table structure
  - Each table is the size of one page
- Page directory
  - Each process has one page directory
  - 1 page in size, must be in main memory
- Page middle directory:
  - May be multiple pages, each entry points to one page in the page table

Linux Memory cont

- Page table
  - May also span multiple pages.
  - Each page table entry refers to one virtual page of the process.

Address Translation

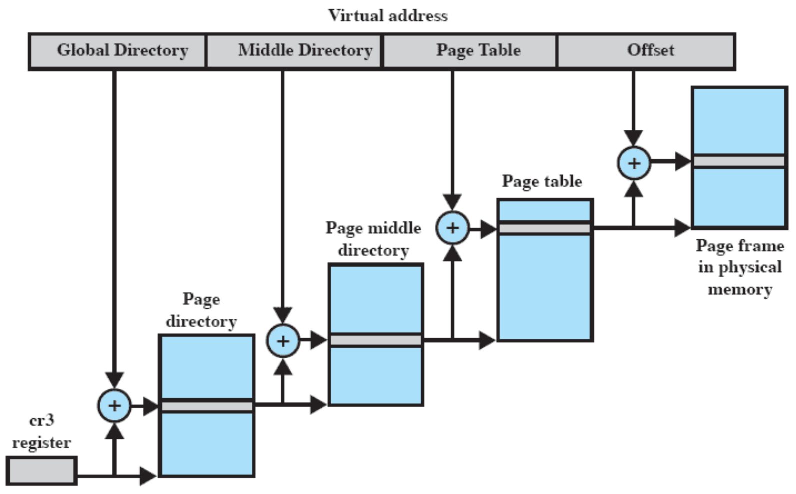


Figure 8.25 Address Translation in Linux Virtual Memory Scheme

Page Replacement

- Based on the clock algorithm
- The “use bit” is replace with an 8-bit age variable
  - Incremented with each page access
- Periodically decrements the age bits
  - Any page with an age of 0 is “old” and is a candidate for replacement
- A form of Least Frequently Used policy

Windows
Memory Management

- The Windows virtual memory manager controls how memory is allocated and how paging is performed.
- Designed to operate over a variety of platforms
  - uses page sizes ranging from 4 Kbytes to 64 Kbytes.

Windows Virtual Address Map

- On 32 bit platforms each user process sees a separate 32 bit address space
  - Allowing 4G per process
- Some reserved for the OS,
  - Typically each user process has 32G of available virtual address space
  - With all processes sharing the same 2G system space

32 bit Windows Address Space

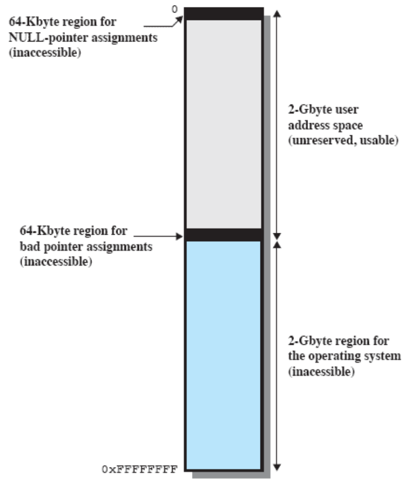

0
64-Kbyte region for NULL-pointer assignments (inaccessible)
2-Gbyte user address space (unreserved, usable)
64-Kbyte region for bad pointer assignments (inaccessible)
2-Gbyte region for the operating system (inacessible)
0xFFFFFFFF

Figure 8.26 Windows Default 32-bit Virtual Address Space

Windows Paging

- On creation, a process can make use of the entire user space of almost 2 Gbytes.
- This space is divided into fixed-size pages managed in contiguous regions allocated on 64Kbyte boundaries
- Regions may be in one of three states
  - Available
  - Reserved
  - Committed

Resident Set Management System

- Windows uses “variable allocation, local scope”
- When activated a process is assigned data structures to manage its working set
- Working sets of active processes are adjusted depending on the availability of main memory