CSC 553 Operating Systems

Lecture 8 - Virtual Memory

What is Virtual Memory?

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.

Hardware and Control Structures

- Two characteristics fundamental to memory management:
  1) All memory references are logical addresses that are dynamically translated into physical addresses at run time
  2) A process may be broken up into a number of pieces that don't need to be contiguously located in main memory during execution
- If these two characteristics are present, it is not necessary that all of the pages or segments of a process be in main memory during execution

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 mainmemory
- Operating system places the process in a blocking state

Execution of a Process

- After the page fault , a 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 is complete, which causes the operating system to place the affected process in the Ready state

Implications of Virtual Memory

- More processes may be maintained in main memory
  - Because only some of the pieces of any particular process are loaded, there is room for more processes
  - This leads to more efficient utilization of the processor because it is more likely that at least one of the more numerous processes will be in a Ready state at any particular time

Implications of Virtual Memory

- A process may be larger than all of main memory
  - If the program being written is too large, the programmer must devise ways to structure the program into pieces that can be loaded separately in some sort of overlay strategy
  - With virtual memory based on paging or segmentation, that job is left to the OS and the hardware
  - The OS automatically loads pieces of a process into main memory as required

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

Simple Paging	Virtual Memory Paging	Simple Segmentation	Virtual Memory Segmentation
Main memory partitioned into small fixed-size chunks called frames		Main memory not partitioned	
Program broken into pages by the compiler or memory management system		Program segments specified by the programmer to the compiler (i.e., the decision is made by the programmer)	
Internal fragmentation within frames		No internal fragmentation	
No external fragmentation		External fragmentation	
Operating system must maintain a page table for each process showing which frame each page occupies		Operating system must maintain a segment table for each process showing the load address and length of each segment	
Operating system must maintain a free frame list		Operating system must maintain a list of free holes in main memory	
Processor uses page number, offset to calculate absolute address		Processor uses segment number, offset to calculate absolute address	
All the pages of a process must be in main memory for process to run, unless overlays are used	Not all pages of a process need be in main memory frames for the process to run. Pages may be read in as needed	All the segments of a process must be in main memory for process to run, unless overlays are used	Not all segments of a process need be in main memory for the process to run. Segments may be read in as needed
	Reading a page into main memory may require writing a page out to disk		Reading a segment into main memory may require writing one or more segments out to disk

Characteristics of Paging and Segmentation

Thrashing

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

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
- Avoids thrashing

Support Needed for Virtual Memory

- For virtual memory to be practical and effective:
  - Hardware must support paging and segmentation
  - Operating system must include software for managing the movement of pages and/or segments between secondary memory and main memory

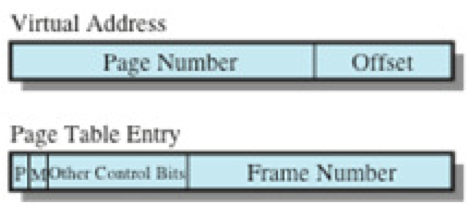
Paging

- The term *virtual memory* is usually associated with systems that employ paging
- Use of paging to achieve virtual memory was first reported for the Atlas computer

Paging

- Each process has its own page table
  - Each page table entry (PTE) contains the frame number of the corresponding page in main memory
  - A page table is also needed for a virtual memory scheme based on paging

Memory Management Format - Paging



(a) Paging only

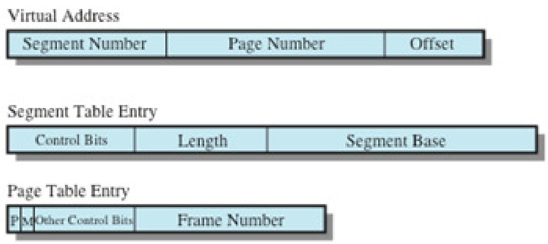
P = Present
M = Modified

Memory Management Format - Segmentation

(b) Segmentation only

P = Present
M = Modified

Memory Management Format – Combined Paging and Segmentation



P= present bit
M = Modified bit

(c) Combined segmentation and paging

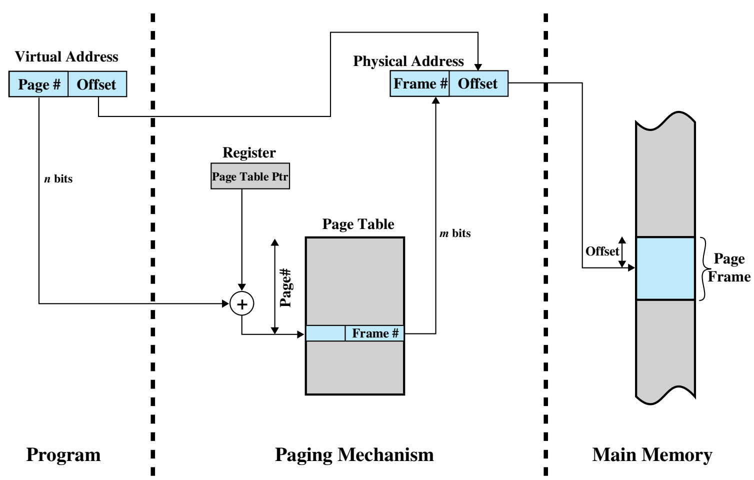


Figure 8.2 Address Translation in a Paging System

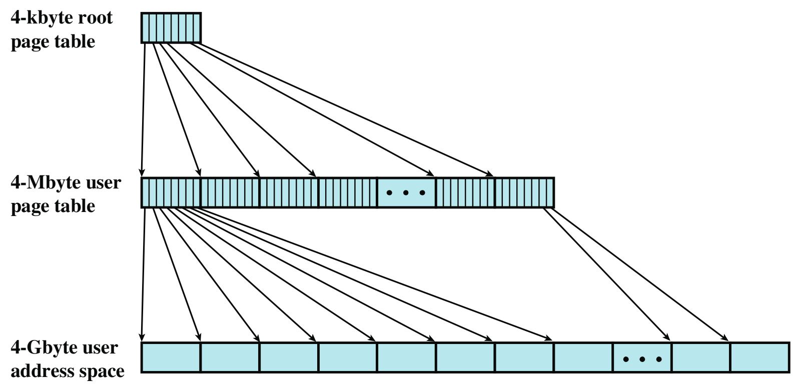


Figure 8.3 A Two-Level Hierarchical Page Table

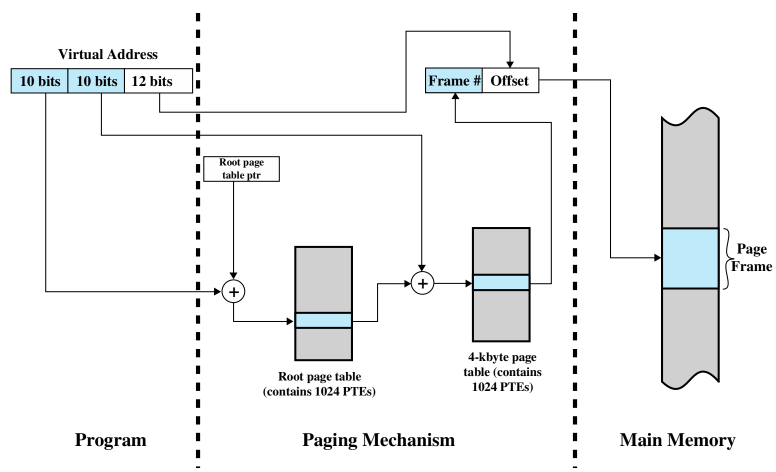


Figure 8.4 Address Translation in a Two-Level Paging System

Inverted Page Table

- 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 or virtual pages supported
- Structure is called ***inverted*** because it indexes page table entries by frame number rather than by virtual page number

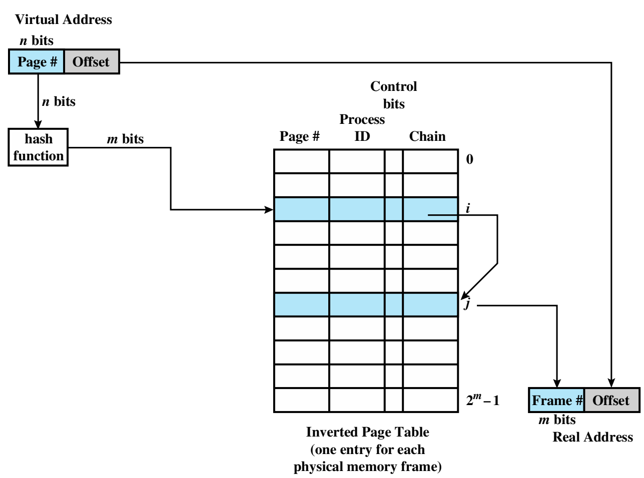


Figure 8.5 Inverted Page Table Structure

Inverted Page Table

- Each entry in the page table includes:
  - Page number - page number portion of the virtual address
  - Process identifier - process that owns this page
  - Control bits - Includes flags and protection and locking information
  - Chain pointer - The index value of the next entry in the chain

Translation Lookaside Buffer (TLB)

- Each virtual memory reference can cause two physical memory accesses:
  - One to fetch the page table entry
  - One to fetch the data

Translation Lookaside Buffer (TLB)

- To overcome the effect of doubling the memory access time, most virtual memory schemes make use of a special high-speed cache called a ***translation lookaside buffer*** (TLB)
  - This cache functions in the same way as a memory cache and contains those page table entities that have been most recently used

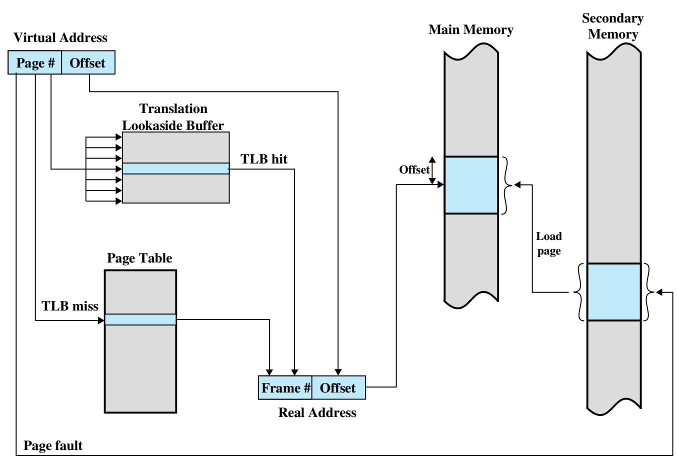


Figure 8.6 Use of a Translation Lookaside Buffer

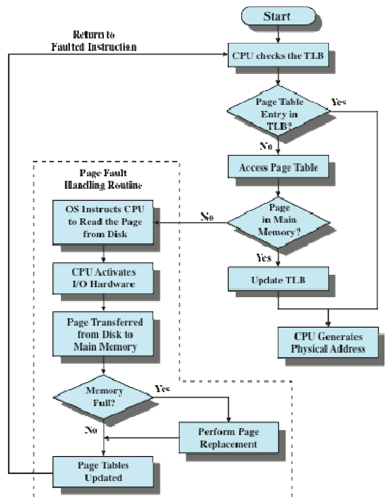


Figure 8.7 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

Associative Mapping

- 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

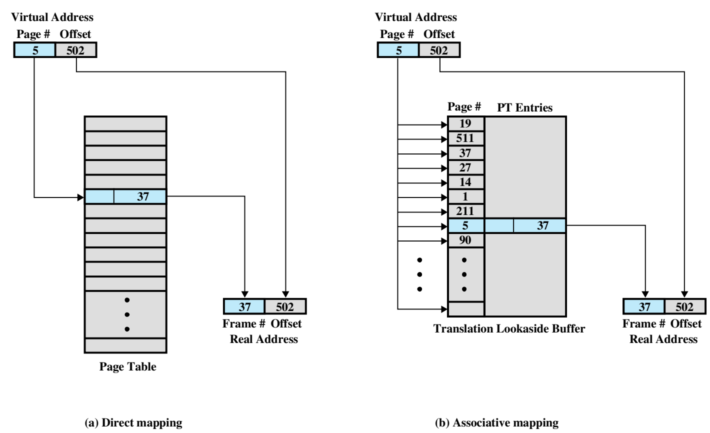


Figure 8.8 Direct Versus Associative Lookup for Page Table Entries

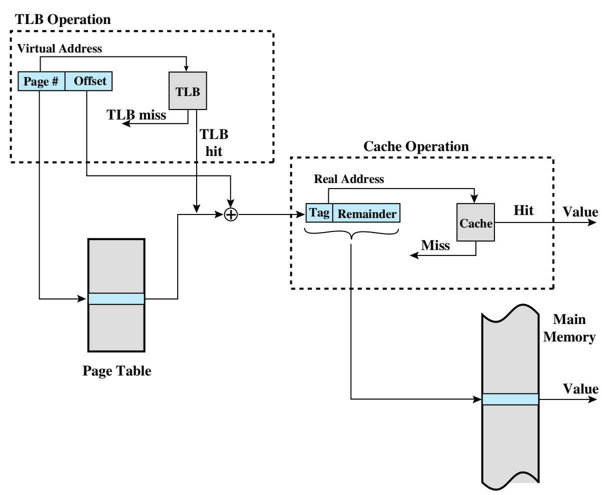
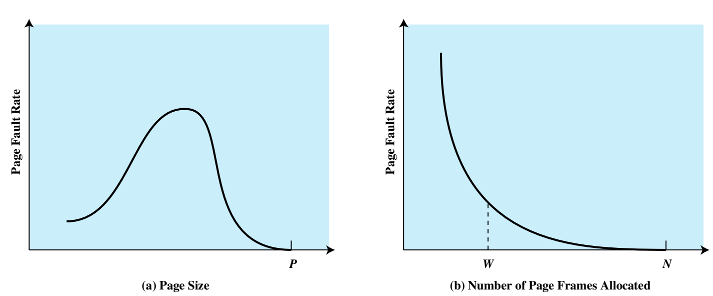


Figure 8.9 Translation Lookaside Buffer and Cache Operation

Page Size

- The smaller the page size, the lesser the amount of internal fragmentation
  - However, more pages are required per process
    More pages per process means larger page tables
  - For large programs in a heavily multiprogrammed environment some portion of the page tables of active processes must be in virtual memory instead of main memory
  - The physical characteristics of most secondary-memory devices favor a larger page size for more efficient block transfer of data



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

Figure 8.10 Typical Paging Behavior of a Program

Examples of 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

Page Size

- The design issue of page size is related to the size of physical main memory and program size
  - Main memory is getting larger and address space used by applications is also growing
  - Most obvious on personal computers where applications are becoming increasingly complex
- Contemporary programming techniques used in large programs tend to decrease the locality of references within a process

Segmentation

- Segmentation allows the programmer to view memory as consisting of multiple address spaces or segments
- Advantages:
  - 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

- 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

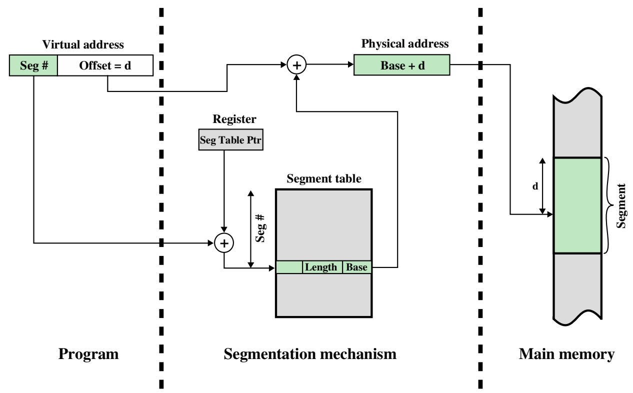


Figure 8.11 Address Translation in a Segmentation System

Combined Paging and Segmentation

- 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
  - Segmentation is visible to the programmer
  - Paging is transparent to the programmer

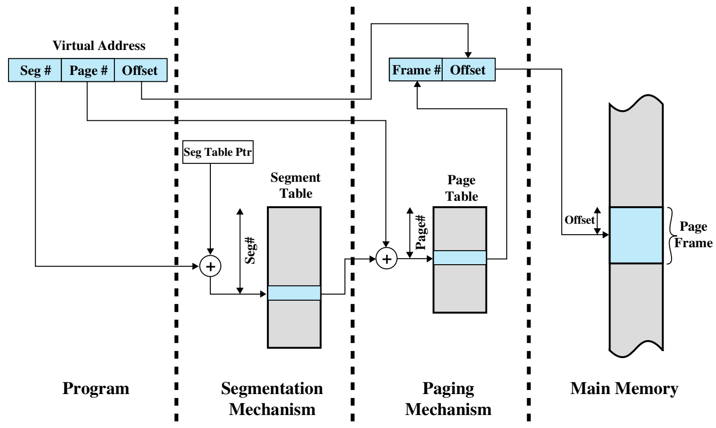


Figure 8.12 Address Translation in a Segmentation/Paging System

Virtual Address

Segment Number	Page Number	Offset

Segment Table Entry

Control Bits	Length	Segment Base

Page Table Entry

P	M	Other Control Bits	Frame Number

P= present bit
M = Modified bit

(c) Combined segmentation and paging

Protection and Sharing

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

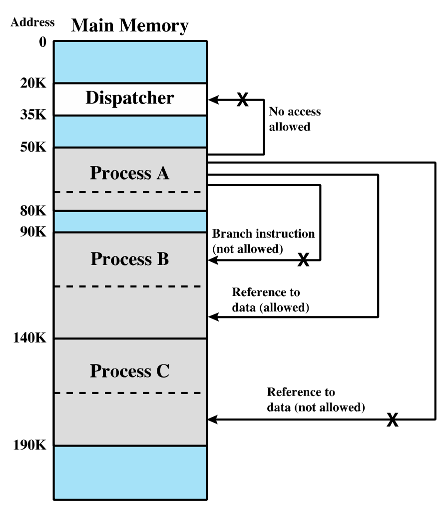


Figure 8.13 Protection Relationships Between Segments

Operating System Software

- 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

Operating System Policies for Virtual Memory

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

Fetch Policy

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

- Pages other than the one demanded by a page fault are brought in
- 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”

Placement Policy

- Determines where in real memory a process piece is to reside
- Important design issue in a segmentation system
- Paging or combined paging with segmentation placing is irrelevant because hardware performs functions with equal efficiency
- For NUMA systems an automatic placement strategy is desirable

Replacement Policy

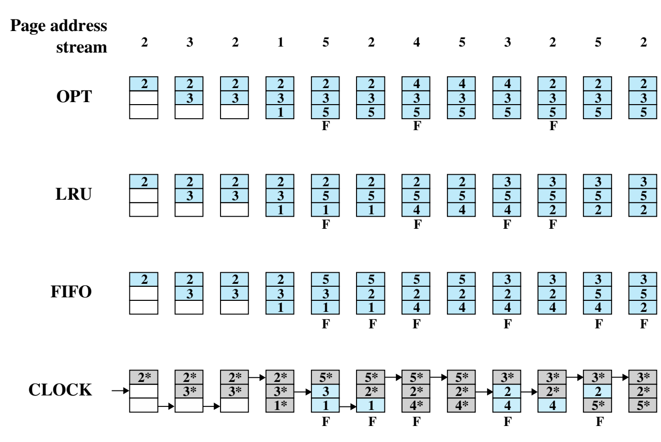
- Deals with the selection of a page in main memory to be replaced when a new page must be brought in
  - Objective is that the page that is removed be the page least likely to be referenced in the near future
- The more elaborate the replacement policy the greater the hardware and software overhead to implement it

Frame Locking

- When a frame is locked the page currently stored in that frame may not be replaced
  - Kernel of the OS as well as key control structures are held in locked frames
  - I/O buffers and time-critical areas may be locked into main memory frames
  - Locking is achieved by associating a lock bit with each frame

Basic Algorithms

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



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

Figure 8.14 Behavior of Four Page-Replacement Algorithms

Optimal Policy

- The **optimal policy** selects for replacement that page for which the time to the next reference is the longest.
  - It can be shown that this policy results in the fewest number of page faults.
  - Clearly, this policy is impossible to implement, but, it does serve as a standard against which to judge real-world algorithms.

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

First-in-First-out (FIFO)

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

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 or referenced, the use bit is set to 1
- The set of frames is considered to be a circular buffer
- Any frame with a use bit of 1 is passed over by the algorithm

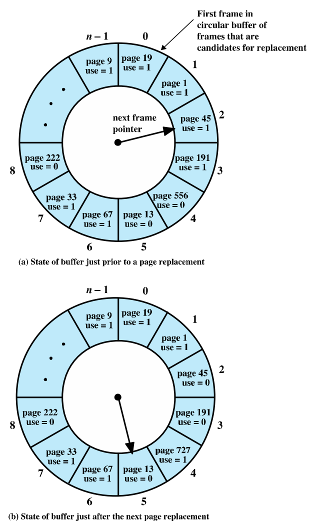


Figure 8.15 Example of Clock Policy Operation

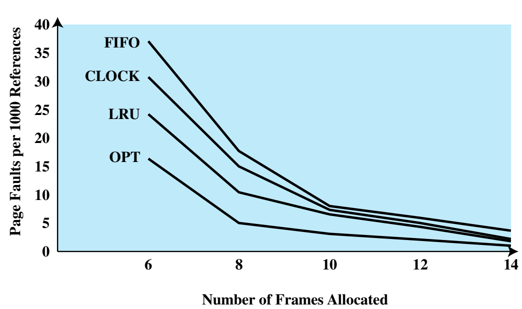


Figure 8.16 Comparison of Fixed-Allocation, Local Page Replacement Algorithms

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** - list of page frames available for reading in pages
  - **Modified page list** - pages are written out in clusters

Replacement Policy and Cache Size

- With large caches, replacement of pages can have a performance impact
  - If the page frame selected for replacement is in the cache, that cache block is lost as well as the page that it holds
  - In systems using page buffering, cache performance can be improved with a policy for page placement in the page buffer
  - Most operating systems place pages by selecting an arbitrary page frame from 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 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 frames in main memory within which to execute
  - When a page fault occurs, one of the pages of that process must be replaced
- Variable-Allocation
  - Allows the number of page frames allocated to a process to be varied 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
- 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

	Local Replacement	Global Replacement
Fixed Allocation	•Number of frames allocated to a 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.

Fixed Allocation, Local Scope

- Necessary to 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
  - Increased time spent in swapping

Variable Allocation Global Scope

- Easiest to implement
  - Adopted in a number of operating systems
- OS maintains a list of free frames
- Free frame is added to resident set of process when a page fault occurs
- If no frames are available the OS must choose a page currently in memory
- One way to counter potential problems is to use page buffering

Variable Allocation - Local Scope

- When a new process is loaded into main memory, allocate to it a certain number of page frames as its resident set
- When a page fault occurs, select the page to replace from among the resident set of the process that suffers the fault
- Reevaluate the allocation provided to the process and increase or decrease it to improve overall performance

Variable Allocation - Local Scope

- Decision to increase or decrease a resident set size is based on the assessment of the likely future demands of active processes
- Key elements:
  - Criteria used to determine resident set size
  - The timing of changes

Sequence of Page References	Window Size, Δ: 2	3	4	5
24	24	24	24	24
15	24 15	24 15	24 15	24 15
18	15 18	24 15 18	24 15 18	24 15 18
23	18 23	15 18 23	24 15 18 23	24 15 18 23
24	23 24	18 23 24	•	•
17	24 17	23 24 17	18 23 24 17	15 18 23 24 17
18	17 18	24 17 18	•	18 23 24 17
24	18 24	•	24 17 18	•
18	•	18 24	•	24 17 18
17	18 17	24 18 17	•	•
17	17	18 17	•	•
15	17 15	17 15	18 17 15	24 18 17 15
24	15 24	17 15 24	17 15 24	•
17	24 17	•	•	17 15 24
24	•	24 17	•	•
18	24 18	17 24 18	17 24 18	15 17 24 18

Figure 8.17 Working Set of Process as Defined by Window Size

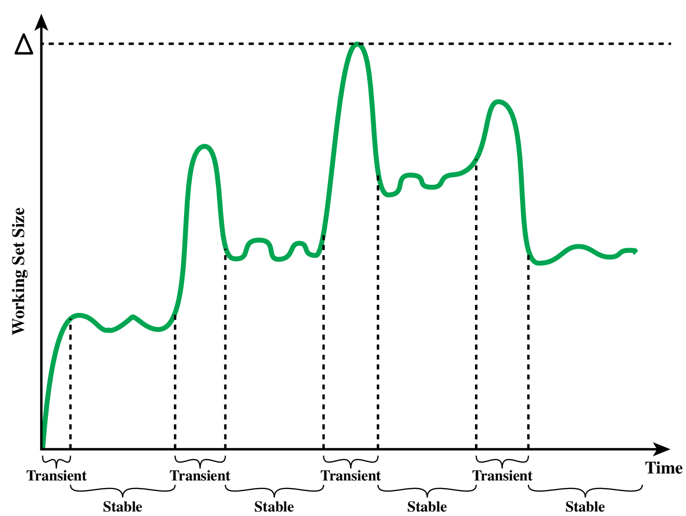


Figure 8.18 Typical Graph of Working Set Size [MAEK87]

Page Fault Frequency (PFF)

- Requires a use bit to be associated with each page in memory
- Bit is set to 1 when that page is accessed
- When a page fault occurs, the OS notes the virtual time since the last page fault for that process
- Does not perform well during the transient periods when there is a shift to a new locality

Variable-Interval Sampled Working Set (VSWS)

- Evaluates the working set of a process at sampling instances based on elapsed virtual time
- Driven by three parameters:
  - The minimum duration of the sampling interval
  - The maximum duration of the sampling interval
  - The number of page faults that are allowed to occur between sampling instances

Cleaning Policy

- Concerned with determining when a modified page should be written out to secondary memory
- Demand Cleaning - A page is written out to secondary memory only when it has been selected for replacement
- Precleaning - Allows the writing of pages in batches

Load Control

- Determines the number of processes that will be resident in main memory
  - *Multiprogramming* level
- Critical in effective memory management
- 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

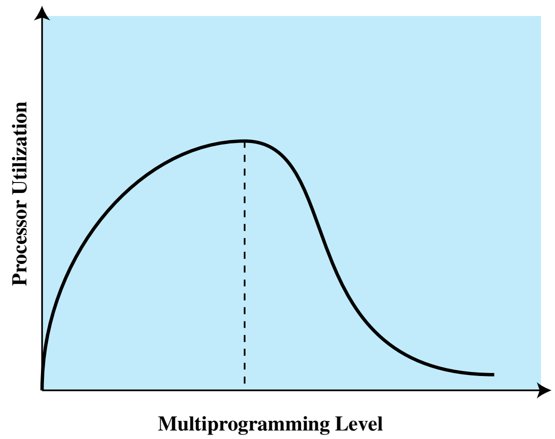


Figure 8.19 Multiprogramming Effects

Process Suspension

- If the degree of multiprogramming is to be reduced, one or more of the currently resident processes must be swapped out
- Six possibilities exist:
  - Lowest-priority process
  - Faulting process
  - Last process activated
  - Process with the smallest resident set
  - Largest process
  - Process with the largest remaining execution window

UNIX

- Intended to be machine independent so its memory management schemes will vary
  - Early UNIX: variable partitioning with no virtual memory scheme
  - Current implementations of UNIX and Solaris make use of paged virtual memory
- SVR4 and Solaris use two separate schemes:
  - Paging system
  - Kernel memory allocator

Paging system & Kernel memory allocator

- Paging System
  - Provides a virtual memory capability that allocates page frames in main memory to processes
  - Allocates page frames to disk block buffers
- Kernel Memory Allocator
  - Allocates memory for the kernel

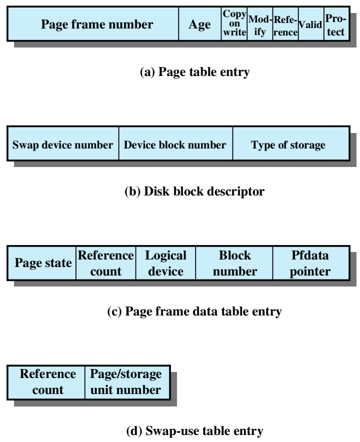


Figure 8.20 UNIX SVR4 Memory Management Formats

UNIX SVR4 Memory Management Parameters

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 0 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

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 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.

UNIX SVR4 Memory Management Parameters

Page Replacement

- The page frame data table is used for page replacement
- Pointers are used to create lists within the table
  - All available frames are linked together in a list of free frames available for bringing in pages
  - When the number of available frames drops below a certain threshold, the kernel will steal a number of frames to compensate

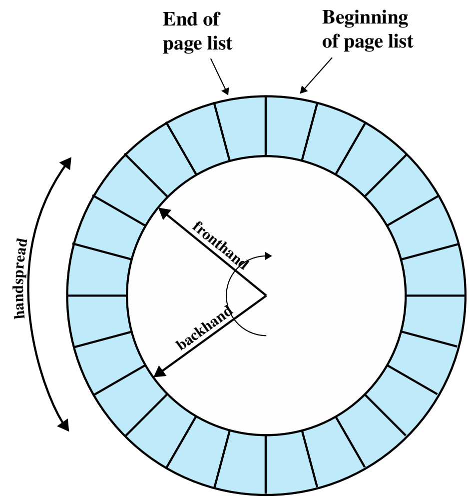


Figure 8.21 Two-Handed Clock Page-Replacement Algorithm

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 are significantly smaller than typical pages (therefore paging would be inefficient)
- Allocations and free operations must be made as fast as possible

Lazy Buddy

- Technique adopted for SVR4
- 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
- Defers coalescing until it seems likely that it is needed, and then coalesces as many blocks as possible

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 8.22 Lazy Buddy System Algorithm