CS 444 Operating Systems

Chapter 3 Memory Management

J. Holly DeBlois

September 14, 2024

Memory

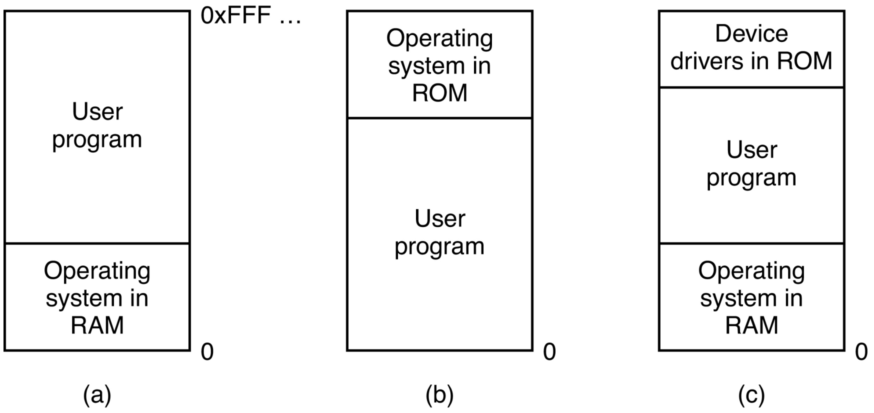
- Paraphrase of Parkinson's Law, "Programs expand to fill the memory available to hold them"
- Average home computer nowadays has 10,000 times more memory than the IBM 7094, the largest computer in the world in the early 1960s

Memory Hierarchy

- Cache: managed by hardware
- Main memory: managed by the OS, this chapter
- Disk: Chapter 5

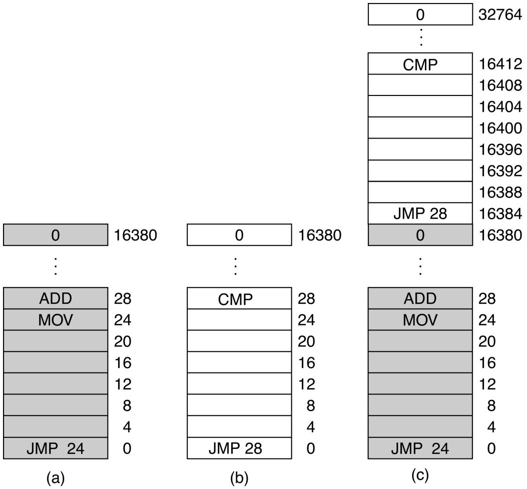
Without Memory Abstraction

- User programs use physical memory addresses
- Mainframes before 1960
- Minicomputers before 1970
- PC before 1980



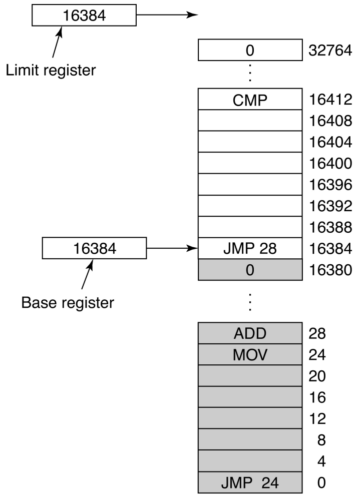
Multiprogramming Without Memory Abstraction

- Need to change absolute memory addresses
- Relocation
- Static relocation changes addresses at loading time
- Add 16384 to 28
  `JMP 16412`



Dynamic Relocation

- Base and limit registers
- OS sets these registers for each process
- Within a process, the base register is added to addresses

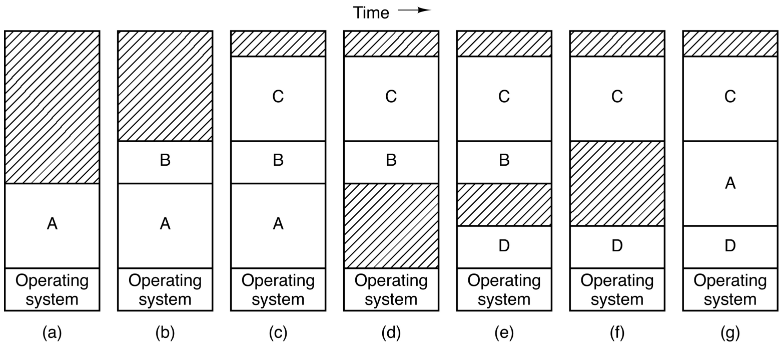


(c)

When Physical Memory is Too Small

- Swapping
  - Swap whole programs in and out of memory
  - Leave multiple holes in memory
  - Periodically compact the memory
  - Memory compaction is a slow operation
- Virtual memory

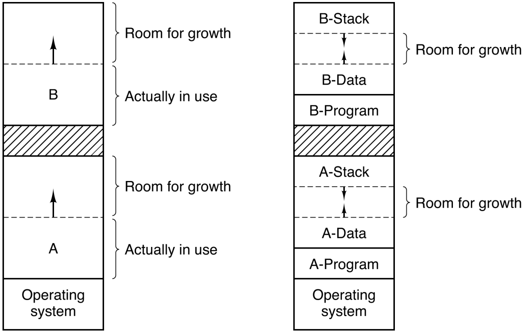
Swapping



- Allocation changes as processes come into memory and leave it
- The shaded regions are unused memory
- OS needs to compact memory when what's available is too fragmented for an incoming process

Processes May Grow the Data and Stack Segments

- Processes may grow the data segment by `malloc()`
- Processes may grow the stack segment by function calls
- When space runs out, a process must be swapped out or killed

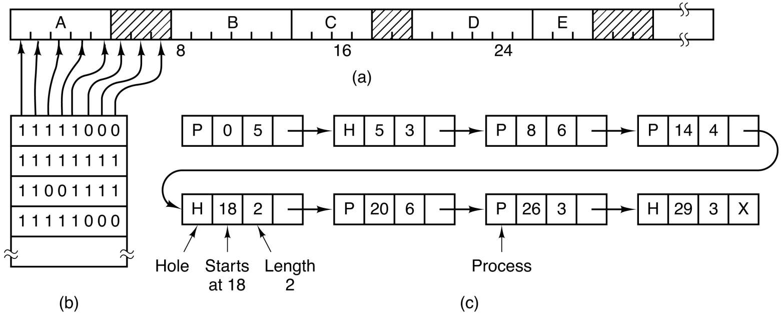


(a) (b)

Managing Free Memory

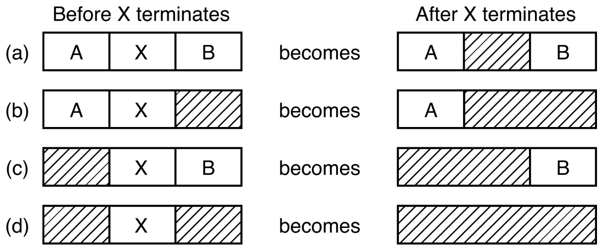
- Bitmaps
- Linked list

Memory Management with Bitmaps



- The size of the allocation unit is an important design issue
- We can easily determine whether a memory unit is in use
- But searching for a run of a given length is slow

Memory Management with Linked Lists



- Sorted by address
- Double-linked
- Two memory lists: processes, holes
  - Hole list can be sorted by sizes, or
  - Use multiple hole lists
- Merge consecutive holes

Memory Management Algorithms

- First fit
- Next fit
- Best fit — leaves many tiny holes
- Worst fit
- Quick fit uses multiple lists of specific hole sizes

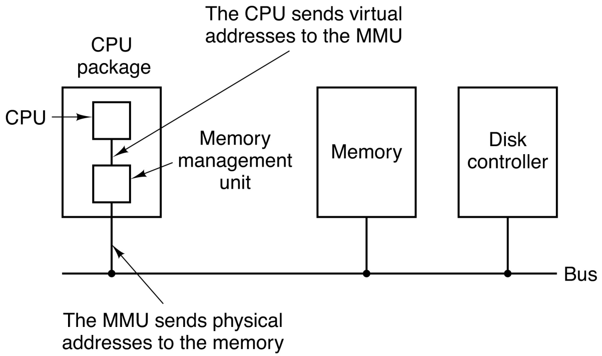
Virtual Memory

- There is a need to run programs that are too large to fit in memory
- Solution adopted in the 1960s
  - Split programs into little pieces, called overlays
  - They are kept on the disk, swapped in and out of memory
- Virtual memory
  - Each program has its own address space, broken up into chunks called pages
  - A generalization of the base/limit register idea

Paging

- The addresses in user programs are virtual addresses
- The virtual address space is divided into pages
- The physical memory is divided into page frames
- Memory management unit (MMU) maps virtual addresses to physical addresses

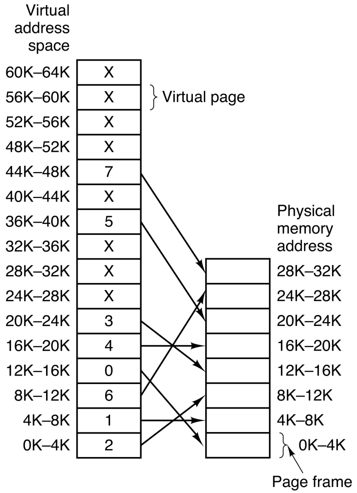
MMU


The CPU sends virtual addresses to the MMU
CPU package
CPU
Memory management unit
Memory
Disk controller
Bus
The MMU sends physical addresses to the memory

- The position and function of the MMU
- Here the MMU is shown as being a part of the CPU chip
- Logically it can be a separate chip and was years ago

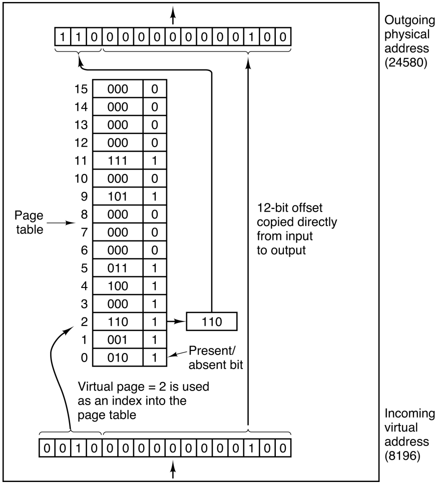
Mapping Virtual Address to Physical Address

- Page table contains the relation between virtual addresses and physical memory addresses
- This example has 16 virtual pages and 8 page frames
- Every page begins on a multiple of 4,096 and ends 4,095 addresses higher
- So 4K-8K really means 4096-8191, and 8K-12K means 8192-12287

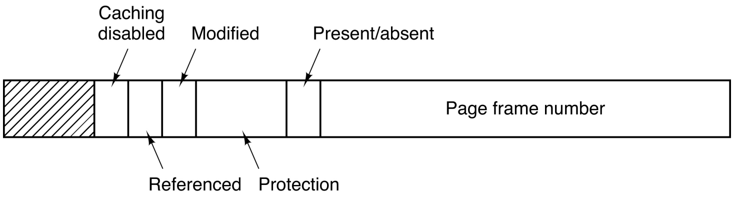


Page Table

- The virtual address has 16 bits
- The lower 12 bits are offsets within a page — 4KB
- The higher 4 bits are virtual page number
- Page table maps the 4-bit virtual page number to the 3-bit page frame number
- The physical address has 15 bits



Structure of a Page Table Entry



- Common size: 32 bits for one entry
- 1 bit for present/absent
- 1 bit for modified (dirty)
- 1 bit for referenced
- 1 bit for caching disabled
- 1 or 3 bits for protection; 1 bit: read/write or read-only; 3 bits: `rwx`

Speeding Up Paging

- Major issues faced
  1. The mapping from virtual address to physical address must be fast
  2. If the virtual address space is large, the page table will be large
- Approaches
  1. All page table in special hardware, too expensive
  2. All page table in RAM, too slow
  3. Hybrid: translation lookaside buffer

Translation Lookaside Buffers

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

- TLB resides inside the MMU, with up to 256 entries
- Associative memory
- Given a virtual address, TLB searches all entries in parallel, checks protection, and retrieves the page frame number

Hardware TLB Management

1. MMU consults with TLB
    - If a TLB hit, get the page frame number
    - If a TLB miss, Step 2
2. MMU consults page table (in RAM)
    - Evicts one entry from TLB
    - Replaces it with the page table entry that was just looked up

Software TLB Management

1. MMU consults with TLB
    - If a TLB hit, get the page frame number
    - If a TLB miss (fault), Step 2
2. Software searches page table (in RAM)
    - The page (in RAM) holding the page table entry may not be in TLB
        - Additional TLB faults
    - Keep a special page (in RAM) of TLB entries, keep this page permanently in TLB

Different Kinds of Misses

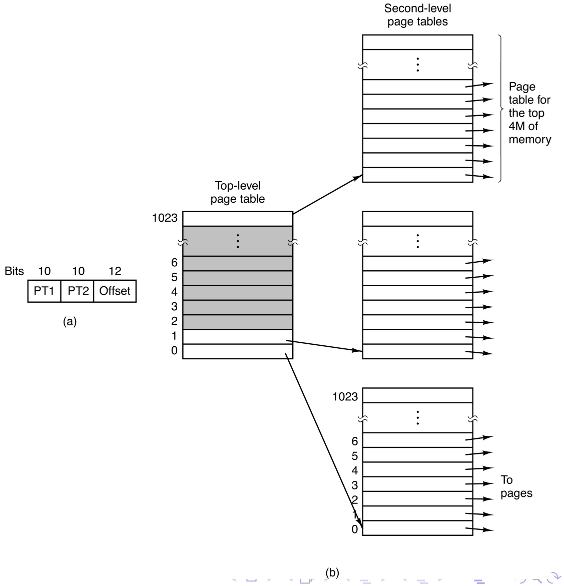
- Soft (TLB) miss: Page table entry not in TLB, but in RAM
- Hard (TLB) miss: Page table entry not in RAM
- Minor page fault: Page in RAM, but not in the page table of this process
  - Such as a dynamically loaded library
  - Add it to the page table of the process that needs it
- Major page fault: Page not in RAM
- Segmentation fault: Invalid address or prohibited operation

Large Virtual Address Space

- Two ways to work with a large virtual address space
- Multilevel page tables
- Inverted page tables

Multilevel Page Tables

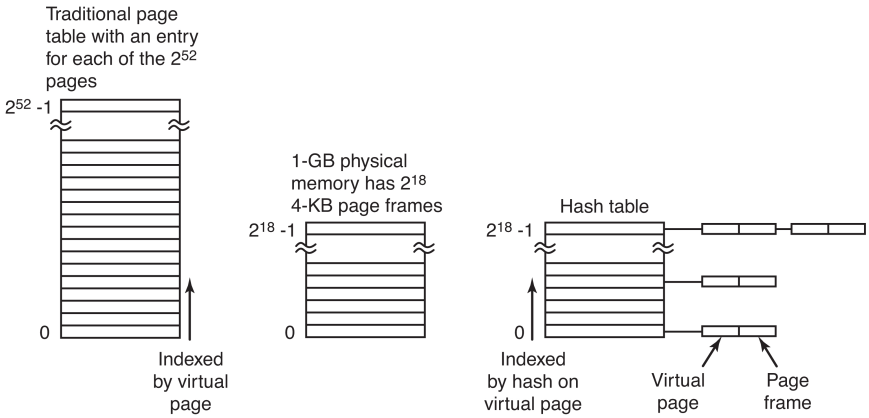
- 1 page table entry is 4B
- 1M pages for 32-bit address
- 1M entries if one page table is used
- Use 2-level page tables
- Only 4 pages for page table are needed, each having 1K entries



x86-64 Implementation of Multilevel Page Tables

- Each page has 4KB (12 bits)
- 4 levels of page tables
- Each level has 512 entries (9 bits)
- Total virtual address space: 48 bits (4 x 9 + 12)
- Allowable physical address: 52 bits (40 + 12)

Inverted Page Tables


Traditional page table with an entry for each of the 2^{52} pages
2^{52} -1
0
Indexed by virtual page
1-GB physical memory has 2^{18} 4-KB page frames
2^{18} -1
0
Hash table
2^{18} -1
0
Indexed by hash on virtual page
Virtual page
Page frame

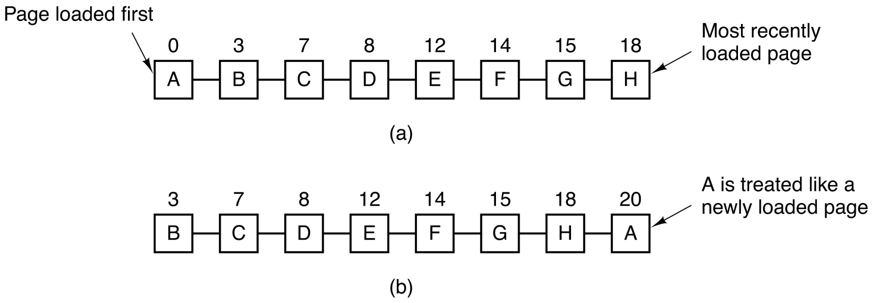
Page Replacement Algorithms

- Optimal algorithm (for benchmark)
- Not recently used algorithm
- First-in, first-out algorithm (rarely used)
- Second-chance algorithm
- Clock algorithm
- Least recently used (LRU) algorithm
- Working set algorithm
- WSClock algorithm

Not Recently Used Algorithm

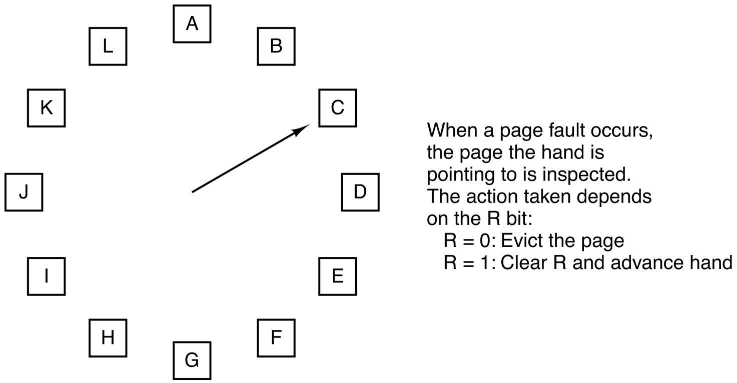
- R: referenced (periodically reset to 0)
- M: modified
- At page fault, system inspects pages
- Categories of pages based on the current values of their R and M bits:
- Class 0: not referenced, not modified
- Class 1: not referenced, modified
- Class 2: referenced, not modified
- Class 3: referenced, modified
- Choose a victim from the lowest nonempty class

Second-Chance Algorithm



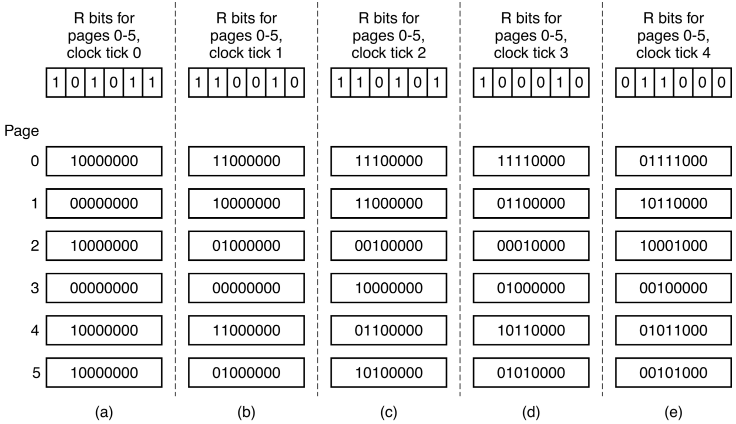
- Pages sorted in FIFO order
- When a page fault occurs at time 20 and page A has its R bit set, clear the R bit and give it a second chance
- The numbers above the pages are their load times

Clock Page Replacement Algorithm


A
B
C
D
E
F
G
H
I
J
K
L
When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:
R = 0: Evict the page
R = 1: Clear R and advance hand

- Just like second-chance
- But organize the linked list as a circular list
- More efficient

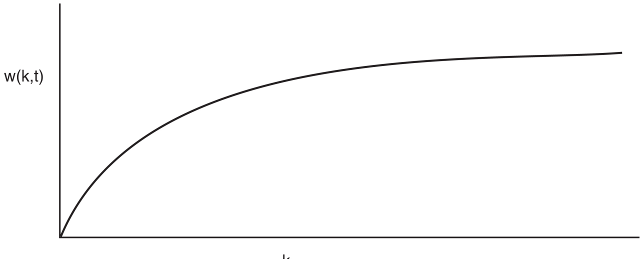
Simulating LRU in Software



	R bits for pages 0-5, clock tick 0	R bits for pages 0-5, clock tick 1	R bits for pages 0-5, clock tick 2	R bits for pages 0-5, clock tick 3	R bits for pages 0-5, clock tick 4
	1 0 1 0 1 1	1 1 0 0 1 0	1 1 0 1 0 1	1 0 0 0 1 0	0 1 1 0 0 0
Page					
0	10000000	11000000	11100000	11110000	01111000
1	00000000	10000000	11000000	01100000	10110000
2	10000000	01000000	00100000	00010000	10001000
3	00000000	00000000	10000000	01000000	00100000
4	10000000	11000000	01100000	10110000	01011000
5	10000000	01000000	10100000	01010000	00101000
	(a)	(b)	(c)	(d)	(e)

- The aging algorithm simulates LRU in software
- 6 pages for 5 clock ticks

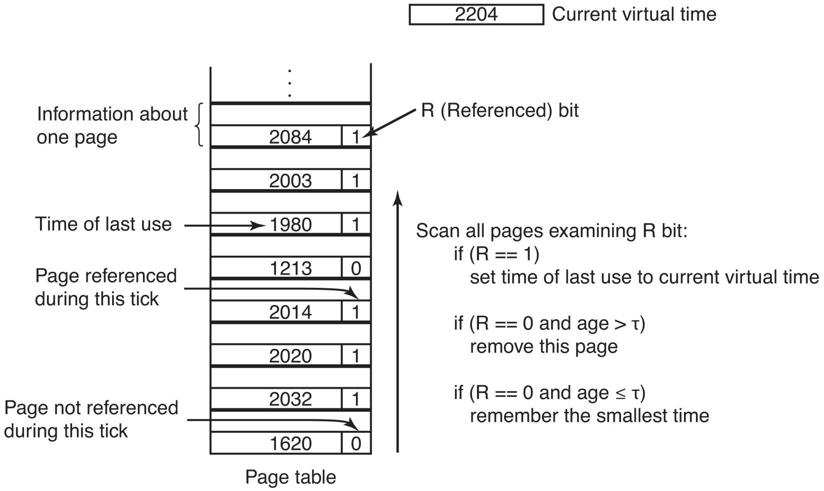
Working Set Concept



- Processes exhibit locality of reference
- The set of pages that a process is currently using is its working set
- A program causing page faults every few instructions is said to be thrashing
- The function $w(k, t)$ is the size of the working set at time t
- The set of pages used by the k most recent memory references

Working Set Algorithm

- Parameters k, τ
- WSClock: implementing working set with a circular list

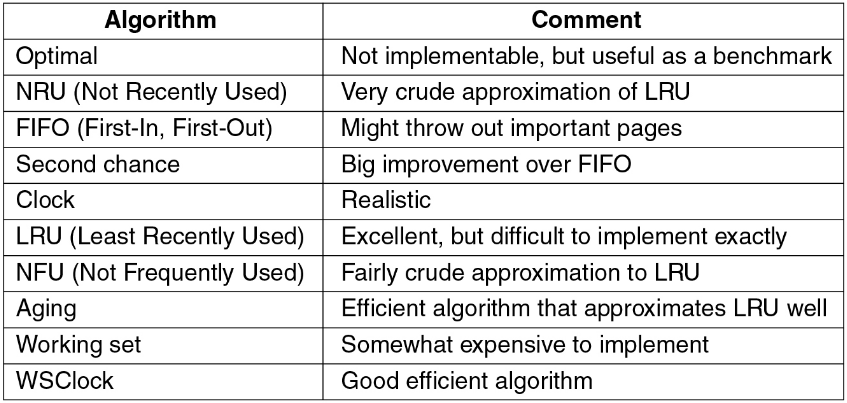


```
Scan all pages examining R bit:
    if (R == 1)
        set time of last use to current virtual time

    if (R == 0 and age > τ)
        remove this page

    if (R == 0 and age ≤ τ)
        remember the smallest time
```

Summary of Page Replacement Algorithms



Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude approximation of LRU
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm

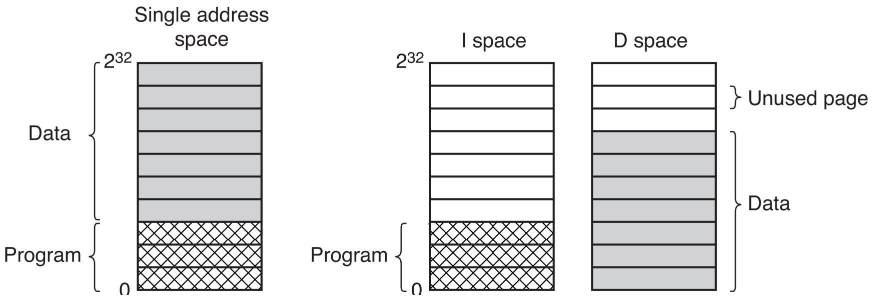
Design Issues in Page Memory

- Local versus global allocation policies
- Load control
- Page size
  - Internal fragmentation
  - s: process size, e: page table entry size, p: page size
    overhead $= se/p + p/2$
  - $s = 1\text{MB}$, $e = 8\text{B}$, $p = 4\text{KB}$
- Separate instruction and data spaces, `rwx`
- Shared pages, shared libraries (DLL)
- Paging daemon periodically removes pages

Local versus Global Allocation Policies

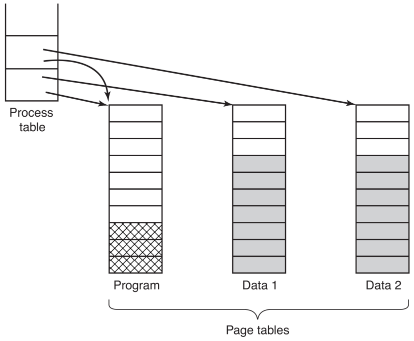
(a)	Age	(b)	(c)
A0	10	A0	A0
A1	7	A1	A1
A2	5	A2	A2
A3	4	A3	A3
A4	6	A4	A4
A5	3	A6	A5
B0	9	B0	B0
B1	4	B1	B1
B2	6	B2	B2
B3	2	B3	A6
B4	5	B4	B4
B5	6	B5	B5
B6	12	B6	B6
C1	3	C1	C1
C2	5	C2	C2
C3	6	C3	C3

Separate Instruction and Data Spaces

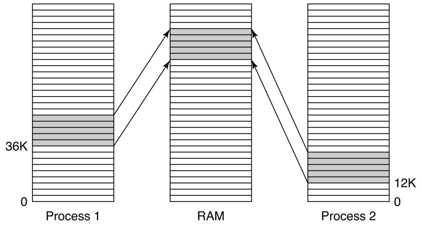


Shared Pages and Libraries

- Shared pages



- Shared libraries



Implementation Issues

- Loading a new process
- Page fault handling
- Instruction backup
- Locking pages in memory
    - In particular, I/O buffers
- Backing store

Loading a New Process

- Create a page table in RAM, initialize it
- Create a swap area on disk, initialize it
    - Program text and data, ready to swap in
- Enter info of page table and swap area in the process table
- When the process is scheduled to run
    - Reset MMU
    - Flush TLB
    - Load pointer to page table to a register

Page Fault Handling

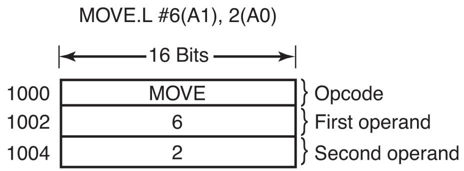
1. The hardware traps to kernel, saving program counter on stack
2. Assembly code routine started to save general registers and other volatile info
3. System discovers page fault has occurred, tries to discover which virtual page needed
4. Once virtual address caused fault is known, system checks to see if address valid and the protection consistent with access
5. If the selected frame is dirty, the page is scheduled for transfer to disk, context switch takes place, suspending faulting process

Page Fault Handling

6. As soon as the frame is clean, OS looks up disk address where needed page is, schedules disk operation to bring it in
7. When disk interrupt indicates the page has arrived, page table is updated to reflect position, and the frame is marked as being in normal state
8. Faulting instruction backed up to state it had when it began and program counter is reset
9. Faulting process is scheduled, operating system returns to routine that called it
10. Routine reloads registers and other state information, returns to user space to continue execution

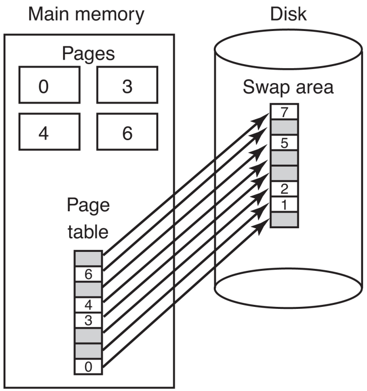
Instruction Backup

- An instruction may cause a page fault

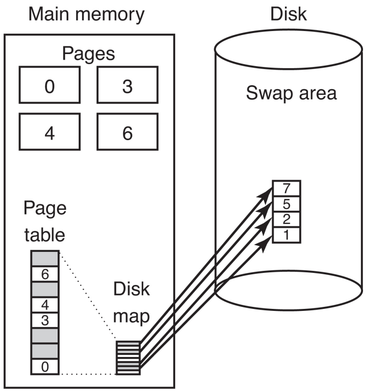


Backing Store

- Paging to a static swap area
- Backing up pages dynamically



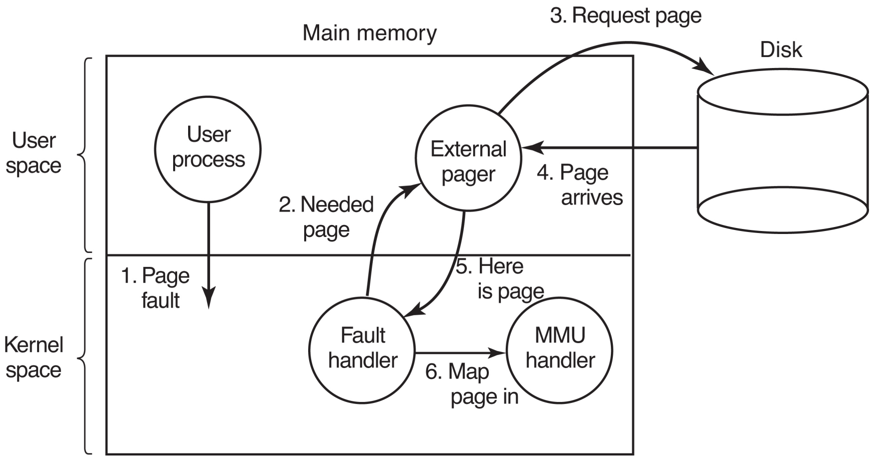
(a)



(b)

Separation of Policy and Mechanism

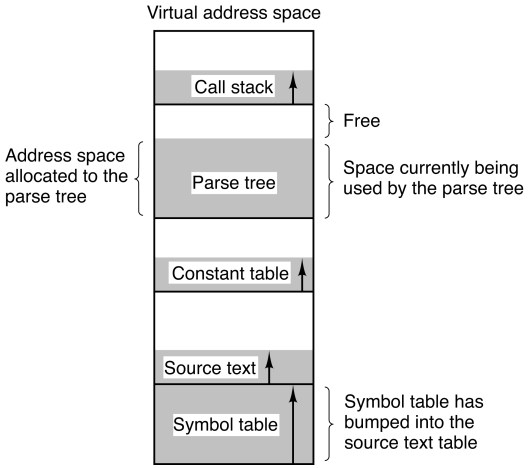
- Page fault handling with an external pager



Tables Generated by Compiler

- The source text being saved for the printed listing
- The symbol table, names and attributes of variables
- The table containing integer and floating-point constants used
- The parse tree, syntactic analysis of the program
- The stack used for procedure calls within compiler

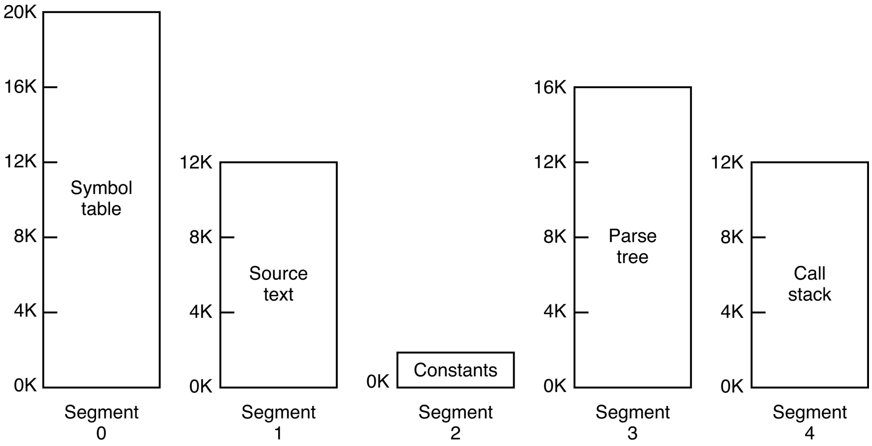
Issues with Contiguous Virtual Memory


Virtual address space
Call stack
Free
Address space allocated to the parse tree
Parse tree
Space currently being used by the parse tree
Constant table
Source text
Symbol table
Symbol table has bumped into the source text table

- A 1-dimensional address space with growing tables
- One table may bump into another

Segmentation

- A segmented memory allows each table to grow or shrink independently of the other tables

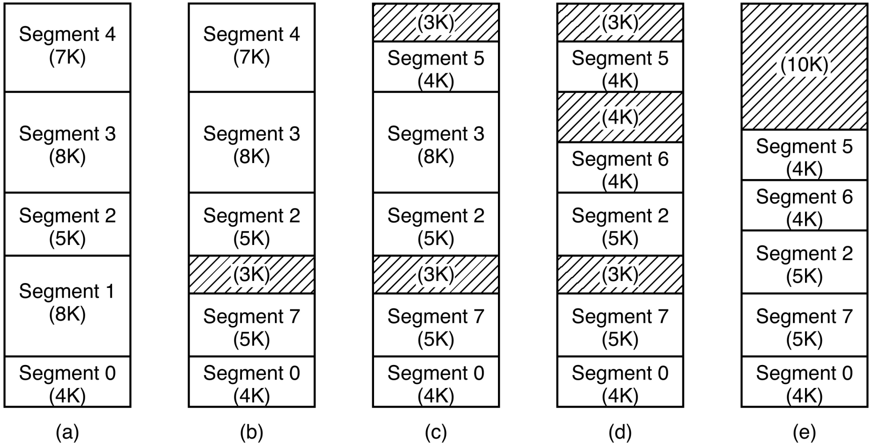


Comparison of Paging and Segmentation

Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection

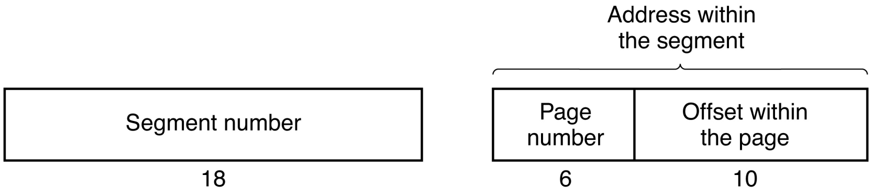
Pure Segmentation

- Lead to checkerboarding
- Use compaction to remove checkerboarding



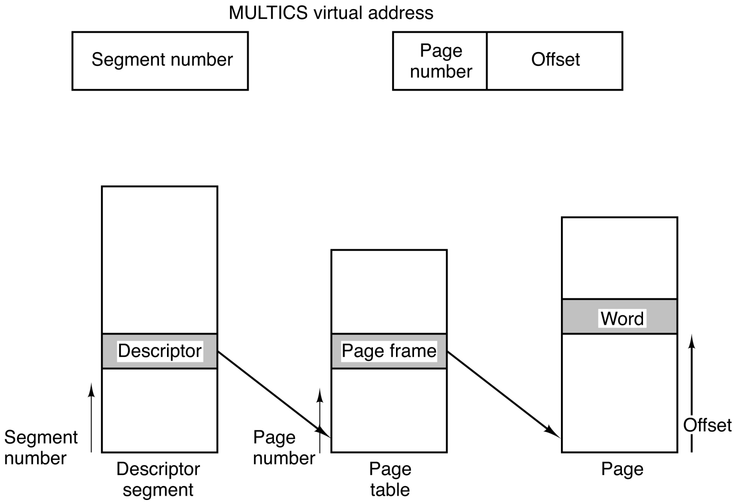
MULTICS

- Segmentation with paging
- 2 sizes of pages — a source of complication
- A descriptor segment that points to the page tables of the segments



MULTICS Virtual Address

- 2 steps of translation



The x86 Segmentation with Paging

- Mainly x86-32
- Obsolete now
- x86-64 is primarily paging