Memory Management

Vera Goebel
Department of Informatics, University of Oslo

with slides from: C. Griwodz (Ifi/UiO), P. Halvorsen (Ifi/UiO), K. Li (Princeton), A. Tanenbaum (VU Amsterdam), and M. van Steen (VU Amsterdam)

- Basic memory management:
  - Mono- and multi-programming
  - Fixed and varaible memory partitioning
- Swapping
- Paging
- Segmentation
- Virtual memory (paging)
- Page replacement algorithms
- Design issues for paging systems
- Implementation issues

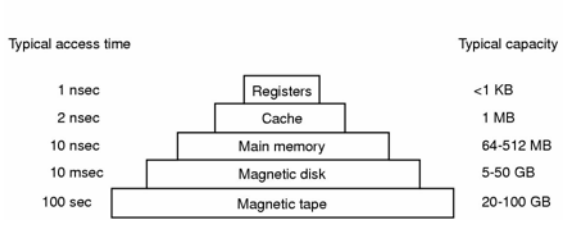
Motivation

- In project assignments so far
  - Program code is linked to kernel
  - Physical addresses are well-known
  - Not realistic
- In the real world
  - Programs are loaded dynamically
  - Physical addresses it will get are not known to program
  - Program size at run-time is not known to kernel

Memory Management

- Ideally programmers want memory that is
  - large
  - fast
  - non volatile
- Memory hierarchy
  - small amount of fast, expensive memory – cache
  - some medium-speed, medium price main memory
  - gigabytes of slow, cheap disk storage
- Memory manager handles the memory hierarchy

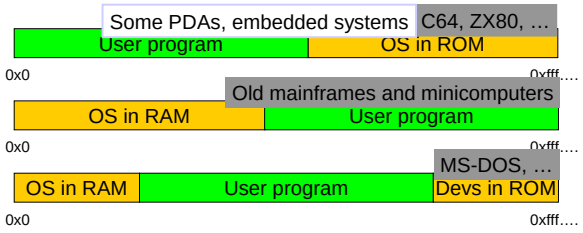
Computer Hardware Review



- Typical memory hierarchy
  - numbers shown are rough approximations

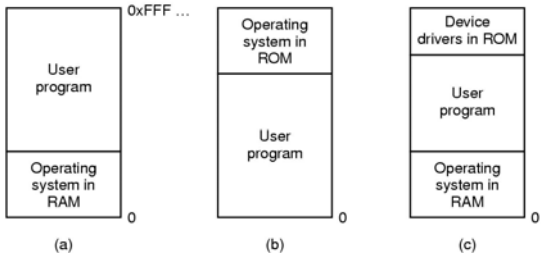
Memory Management for Monoprogramming

- Only one user program loaded
  - Program is entirely in memory
  - No swapping or paging
- Three simple ways of organizing memory



Basic Memory Management

Monoprogramming without Swapping or Paging



Three simple ways of organizing memory
- an operating system with one user process

Multiprogramming

- Processes have to wait for I/O
- Goal
  - Do other work while a process waits
  - Give CPU to another process
- Processes may be concurrently ready
- So
  - If I/O waiting probability for all processes is *p*
  - Probable CPU utilization can be estimated as

    CPU utilization = 1 - p^n

7

Multiprogramming

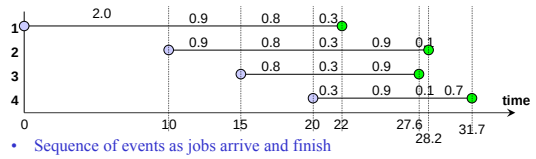
- Arrival and work requirements of 4 jobs
- CPU utilization for 1-4 jobs with 80% I/O wait

Job#	Arrival time	CPU use time	IO wait time
1	10:00	4	16
2	10:10	3	12
3	10:15	2	8
4	10:20	2	8

Remaining CPU time

1	2.0
2	1.9
3	1.8
4	1.7

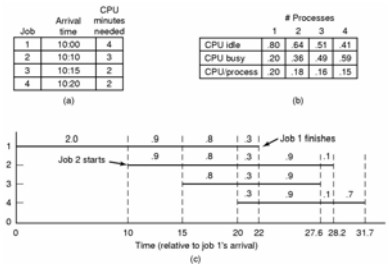
%	# processors 1	2	3	4
CPU idle	80	64	51	41
CPU busy	20	36	49	59
CPU per process	20	18	16	15



- Sequence of events as jobs arrive and finish
  - Note numbers show amount of CPU time jobs get each interval

8

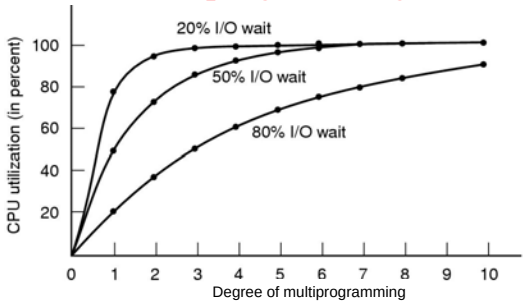
Analysis of Multiprogramming System Performance



- Arrival and work requirements of 4 jobs
- CPU utilization for 1 – 4 jobs with 80% I/O wait
- Sequence of events as jobs arrive and finish
  - note numbers show amout of CPU time jobs get in each interval

9

Multiprogramming



- CPU utilization as a function of number of processes in memory

10

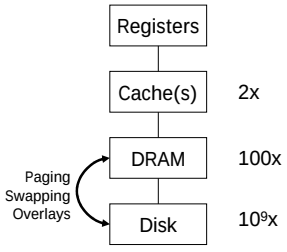
Multiprogramming

- Several programs
  - Concurrently loaded into memory
  - OS must arrange memory sharing
  - Memory partitioning
- Memory
  - Needed for different tasks within a process
  - Shared among processes
  - Process memory demand may change over time
- Use of secondary storage
  - Move (parts of) blocking processes from memory
  - Higher degree of multiprogramming possible
  - Makes sense if processes block for long times

11

Memory Management for Multiprogramming

- Process may not be entirely in memory
- Reasons
  - Other processes use memory
    - Their turn
    - Higher priority
    - Process is waiting for I/O
  - Too big
    - For its share
    - For entire available memory
- Approaches
  - Swapping
  - Paging
  - Overlays



12

Memory Management for Multiprogramming

- Swapping
  - Remove a process from memory
    - With all of its state and data
    - Store it on a secondary medium
      - Disk, Flash RAM, other slow RAM, historically also Tape
- Paging
  - Remove part of a process from memory
    - Store it on a secondary medium
    - Sizes of such parts are fixed
    - Page size
- Overlays
  - Manually replace parts of code and data
    - Programmer's rather than OS's work
    - Only for very old and memory-scarce systems

How to use these with Virtual Memory

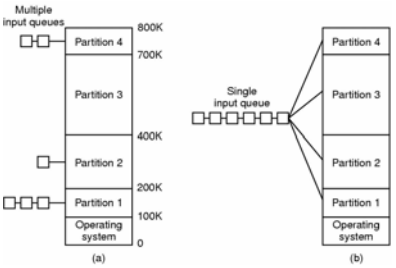
13

Memory Management Techniques

- Before details about moving processes out
  - Assign memory to processes
- Memory partitioning
  - Fixed partitioning
  - Dynamic partitioning
  - Simple paging
  - Simple segmentation
  - Virtual memory paging
  - Virtual memory segmentation

14

Multiprogramming with Fixed Partitions



- Fixed memory partitions
  - separate input queues for each partition
  - single input queue

15

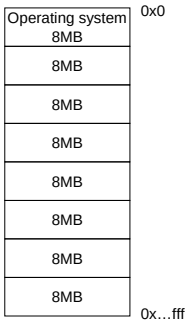
Fixed Partitioning

- Divide memory
  - Into static partitions
  - At system initialization time (boot or earlier)
- Advantages
  - Very easy to implement
  - Can support swapping process in and out

16

Fixed Partitioning

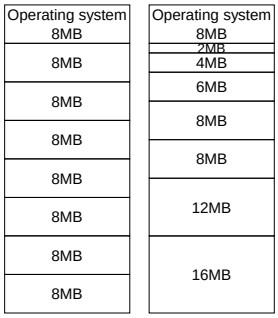
- Two fixed partitioning schemes
  - Equal-size partitions
  - Unequal-size partitions
- Equal-size partitions
  - Big programs can not be executed
    - Unless program parts are loaded from disk
  - Small programs use entire partition
    - A problem called "internal fragmentation"



17

Fixed Partitioning

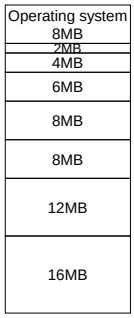
- Two fixed partitioning schemes
  - Equal-size partitions
  - Unequal-size partitions
- Unequal-size partitions
  - Bigger programs can be loaded at once
  - Smaller programs can lead to less internal fragmentation
  - Advantages require assignment of jobs to partitions



18

Fixed Partitioning

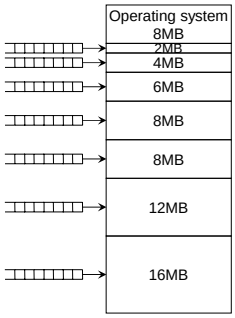
- Approach
  - Has been used in mainframes
  - Uses the term job for a running program
  - Jobs run as batch jobs
  - Jobs are taken from a queue of pending jobs
- Problem with unequal partitions
  - Choosing a job for a partition



19

Fixed Partitioning

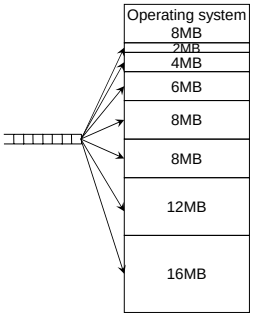
- One queue per partition
  - Internal fragmentation is minimal
  - Jobs wait although sufficiently large partitions are available



20

Fixed Partitioning

- Single queue
  - Jobs are put into next sufficiently large partition
  - Waiting time is reduced
  - Internal fragmentation is bigger
  - A swapping mechanism can reduce internal fragmentation
    - Move a job to another partition



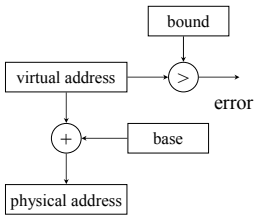
21

Problems: Relocation and Protection

- Cannot be sure where program will be loaded in memory
  - address locations of variables, code routines cannot be absolute
  - must keep a program out of other processes' partitions
- Use base and limit values
  - address locations added to base value to map to physical addr
  - address locations larger than limit value is an error

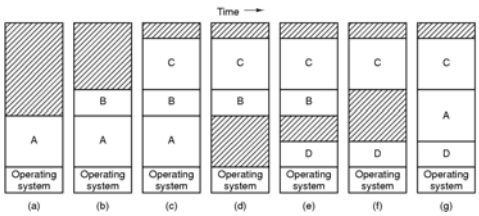
22

2 Registers: Base and Bound



- Built in Cray-1
- A program can only access physical memory in [base, base+bound]
- On a context switch: save/restore base, bound registers
- Pros: Simple
- Cons: fragmentation, hard to share, and difficult to use disks

Swapping (1)



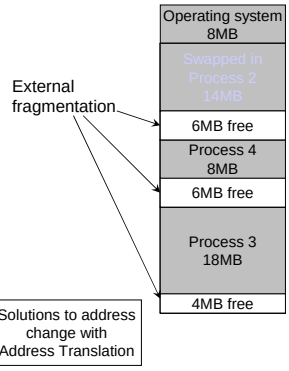
Memory allocation changes as
- processes come into memory
- leave memory

Shaded regions are unused memory

24

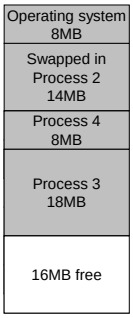
Dynamic Partitioning

- Divide memory
  - Partitions are created dynamically for jobs
  - Removed after jobs are finished
- External fragmentation
  - Problem increases with system running time
  - Occurs with swapping as well
  - Addresses of process 2 changed



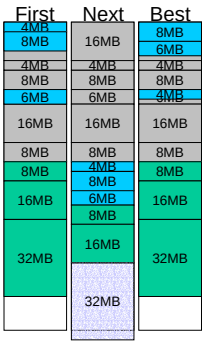
Dynamic Partitioning

- Reduce external fragmentation
  - Compaction
- Compaction
  - Takes time
  - Consumes processing resources
- Reduce compaction need
  - Placement algorithms



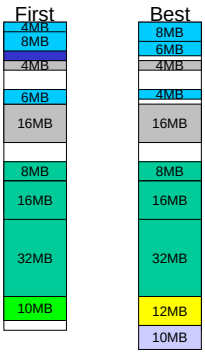
Dynamic Partitioning: Placement Algorithms

- Use most suitable partition for process
- Typical algorithms
  - First fit
  - Next fit
  - Best fit



Dynamic Partitioning: Placement Algorithms

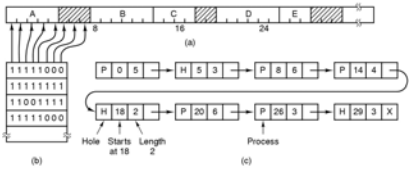
- Use most suitable partition for process
- Typical algorithms
  - First fit
  - Next fit
  - Best fit



Dynamic Partitioning: Placement Algorithms

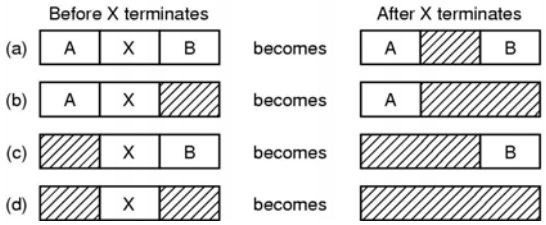
- Comparison of First fit, Next fit and Best fit
- Example is naturally artificial
  - First fit
    - Simplest, fastest of the three
    - Typically the best of the three
  - Next fit
    - Typically slightly worse than first fit
    - Problems with large segments
  - Best fit
    - Slowest
    - Creates lots of small free blocks
    - Therefore typically worst

Memory Management with Bit Maps



- Part of memory with 5 processes, 3 holes
  - tick marks show allocation units
  - shaded regions are free
- Corresponding bit map
- Same information as a list

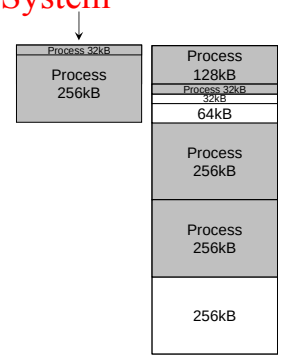
Memory Management with Linked Lists



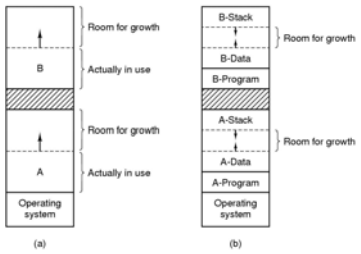
Four neighbor combinations for the terminating process X

Buddy System

- Mix of fixed and dynamic partitioning
  - Partitions have sizes 2^k, $L \le k \le U$
- Maintain a list of holes with sizes
- Assign a process
  - Find smallest k so that process fits into 2^k
  - Find a hole of size 2^k
  - If not available, split smallest hole larger than 2^k
    - Split recursively into halves until two holes have size 2^k



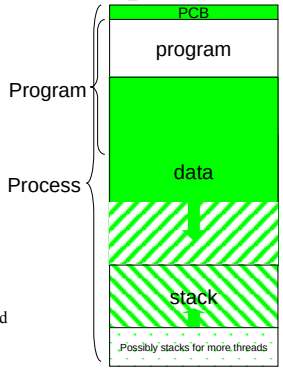
Swapping (2)



- Allocating space for growing data segment
- Allocating space for growing stack & data segment

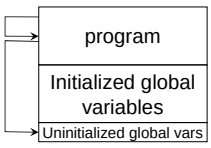
Memory use within a process

- Memory needs of known size
  - Program code
  - Global variables
- Memory needs of unknown size
  - Dynamically allocated memory
  - Stack
    - Several in multithreaded programs



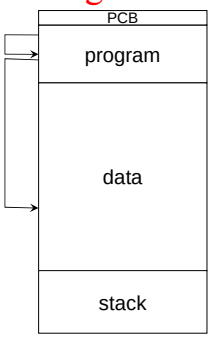
Memory Addressing

- Addressing in memory
  - Addressing needs are determined during programming
  - Must work independently of position in memory
  - Actual physical address are not known



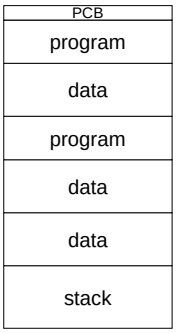
Memory Addressing

- Addressing in memory
  - Addressing needs are determined during programming
  - Must work independently of position in memory
  - Actual physical address are not known



Memory Management

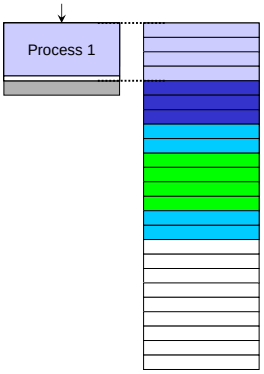
- Addressing
  - Covered address translation and virtual memory
- Important now
  - Translation is necessary
  - Therefore possible to have several parts
    - Pages
    - Segments



37

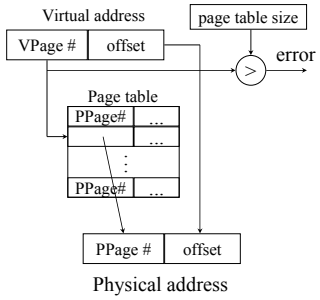
Paging

- Paging
  - Equal lengths
  - Determined by processor
  - One page moved into one memory frames
- Process is loaded into several frames
  - Not necessarily consecutive
- No external fragmentation
- Little internal fragmentation
  - Depends on frame size



38

Paging



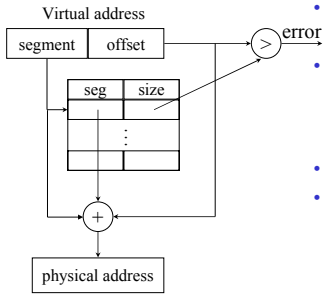
- Use a page table to translate
- Various bits in each entry
- Context switch: similar to the segmentation scheme
- What should be the page size?
- Pros: simple allocation, easy to share
- Cons: big page table and cannot deal with holes easily

Segmentation

- Segmentation
  - Different lengths
  - Determined by programmer
  - Memory frames
- Programmer (or compiler toolchain) organizes program in parts
  - Move control
  - Needs awareness of possible segment size limits
- Pros and Cons
  - Principle as in dynamic partitioning
  - No internal fragmentation
  - Less external fragmentation because on average smaller segments

40

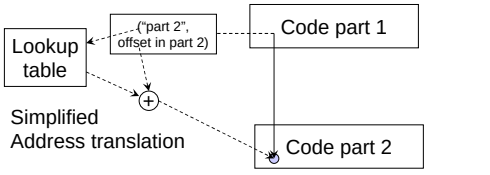
Segmentation



- Have a table of (seg, size)
- Protection: each entry has
  - (nil, read, write)
- On a context switch: save/restore the table or a pointer to the table in kernel memory
- Pros: Efficient, easy to share
- Cons: Complex management and fragmentation within a segment

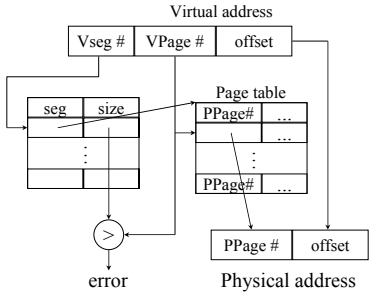
Paging and Segmentation

- Typical for paging and swapping
  - Address translation
  - At execution time
  - With processor support
- Simple paging and segmentation
  - Without virtual memory and protection
  - Can be implemented
    - by address rewriting at load time
    - by jump tables setup at load time



42

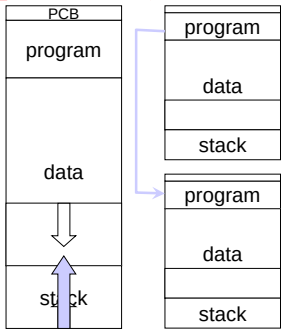
Segmentation with Paging



Other needs (protection)

- Protection of process from itself
  - (stack grows into heap)
- Protection of processes from each other
  - (write to other process)

Solutions to protection with Address Translation



44

Summary: Memory Management

- Algorithms
  - Paging and segmentation
    - Extended in address translation and virtual memory lectures
  - Placement algorithms for partitioning strategies
    - Mostly obsolete for system memory management
      - since hardware address translation is available
    - But still necessary for managing
      - kernel memory
      - memory within a process
      - memory of specialized systems (esp. database systems)
- Address translation solves
  - Solves addressing in a loaded program
- Hardware address translation
  - Supports protection from data access
  - Supports new physical memory position after swapping in
- Virtual memory provides
  - Provide larger logical (virtual) than physical memory
  - Selects process, page or segment for removal from physical memory

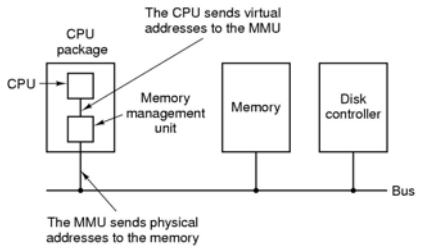
45

Why Virtual Memory?

- Use secondary storage
  - Extend expensive DRAM with reasonable performance
- Protection
  - Programs do not step over each other and communicate with each other require explicit IPC operations
- Convenience
  - Flat address space and programs have the same view of the world

46

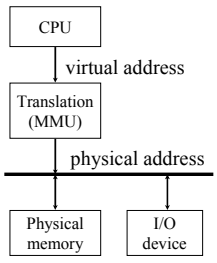
Virtual Memory
Paging (1)



The position and function of the MMU

47

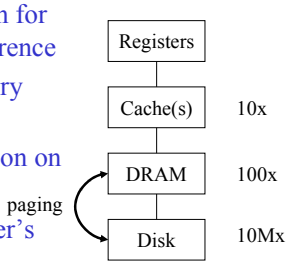
Translation Overview



- Actual translation is in hardware (MMU)
- Controlled in software
- CPU view
  - what program sees, virtual memory
- Memory view
  - physical memory

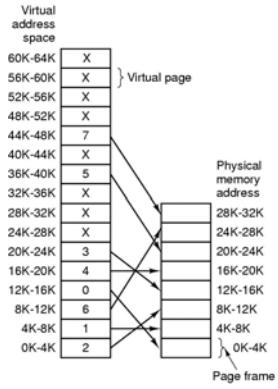
Goals of Translation

- Implicit translation for each memory reference
- A hit should be very fast
- Trigger an exception on a miss
- Protected from user's faults

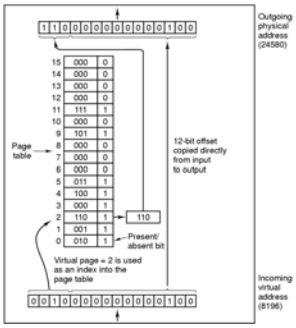


Paging (2)

The relation between virtual addresses and physical memory addresses given by page table

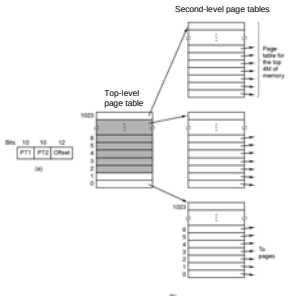


Page Tables (1)



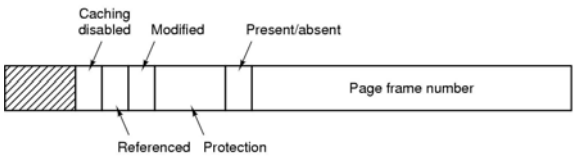
Internal operation of MMU with 16 4 KB pages

Page Tables (2)



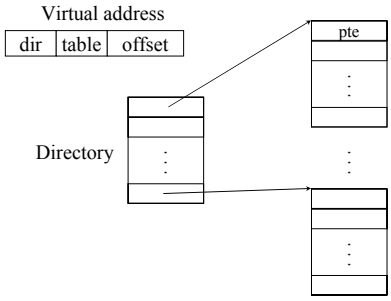
- 32 bit address with 2 page table fields
- Two-level page tables

Page Tables (3)



Typical page table entry

Multiple-Level Page Tables



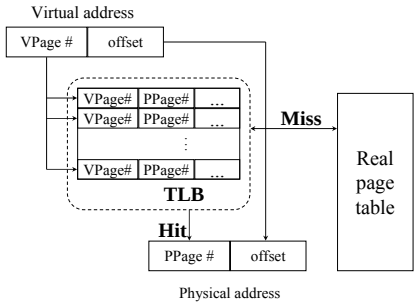
TLBs – Translation Lookaside Buffers

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

A TLB to speed up paging

55

Translation Look-aside Buffer (TLB)



Bits in A TLB Entry

- Common (necessary) bits
  - Virtual page number: match with the virtual address
  - Physical page number: translated address
  - Valid
  - Access bits: kernel and user (nil, read, write)
- Optional (useful) bits
  - Process tag
  - Reference
  - Modify
  - Cacheable

Hardware-Controlled TLB

- On a TLB miss
  - Hardware loads the PTE into the TLB
    - Need to write back if there is no free entry
  - Generate a fault if the page containing the PTE is invalid
  - VM software performs fault handling
  - Restart the CPU
- On a TLB hit, hardware checks the valid bit
  - If valid, pointer to page frame in memory
  - If invalid, the hardware generates a page fault
    - Perform page fault handling
    - Restart the faulting instruction

Software-Controlled TLB

- On a miss in TLB
  - Write back if there is no free entry
  - Check if the page containing the PTE is in memory
  - If no, perform page fault handling
  - Load the PTE into the TLB
  - Restart the faulting instruction
- On a hit in TLB, the hardware checks valid bit
  - If valid, pointer to page frame in memory
  - If invalid, the hardware generates a page fault
    - Perform page fault handling
    - Restart the faulting instruction

Hardware vs. Software Controlled

- Hardware approach
  - Efficient
  - Inflexible
  - Need more space for page table
- Software approach
  - Flexible
  - Software can do mappings by hashing
    - PP# → (Pid, VP#)
    - (Pid, VP#) → PP#
  - Can deal with large virtual address space

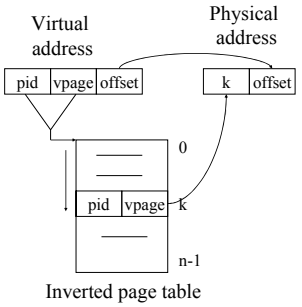
60

How Many PTEs Do We Need?

- Worst case for 32-bit address machine
  - # of processes × 2^{20} (if page size is 4096 bytes)
- What about 64-bit address machine?
  - # of processes × 2^{52}

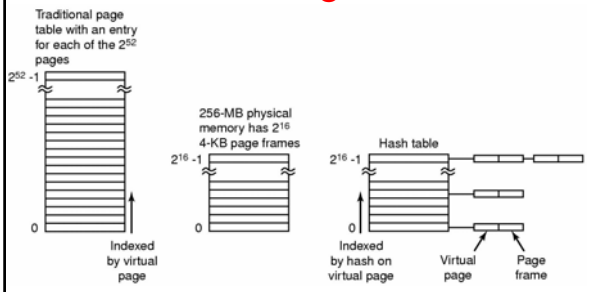
61

Inverted Page Tables



- Main idea
  - One PTE for each physical page frame
  - Hash (Vpage, pid) to Ppage#
- Pros
  - Small page table for large address space
- Cons
  - Lookup is difficult
  - Overhead of managing hash chains, etc

Inverted Page Tables



Comparison of a traditional page table with an inverted page table

63

Page Replacement Algorithms

- Page fault forces choice
  - which page must be removed
  - make room for incoming page
- Modified page must first be saved
  - unmodified just overwritten
- Better not to choose an often used page
  - will probably need to be brought back in soon

64

Optimal Page Replacement Algorithm

- Replace page needed at the farthest point in future
  - Optimal but unrealizable
- Estimate by …
  - logging page use on previous runs of process
  - although this is impractical

65

Not Recently Used Page Replacement Algorithm

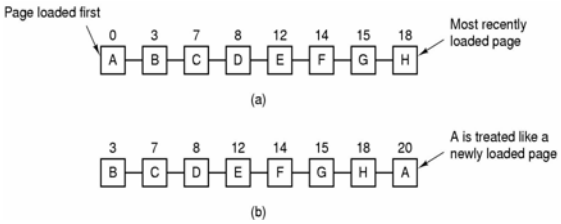
- Each page has Reference bit, Modified bit
  - bits are set when page is referenced, modified
- Pages are classified
  1. not referenced, not modified
  2. not referenced, modified
  3. referenced, not modified
  4. referenced, modified
- NRU removes page at random
  - from lowest numbered non empty class

66

FIFO Page Replacement Algorithm

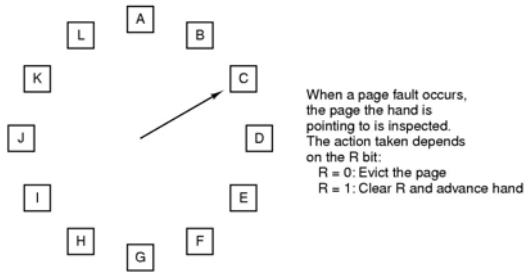
- Maintain a linked list of all pages
  - in order they came into memory
- Page at beginning of list replaced
- Disadvantage
  - page in memory the longest may be often used

Second Chance Page Replacement Algorithm



- Operation of a second chance
  - pages sorted in FIFO order
  - Page list if fault occurs at time 20, A has *R* bit set (numbers above pages are loading times)

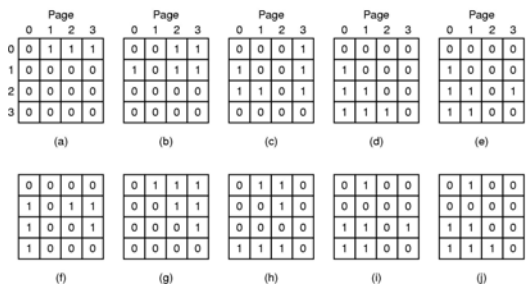
The Clock Page Replacement Algorithm



Least Recently Used (LRU)

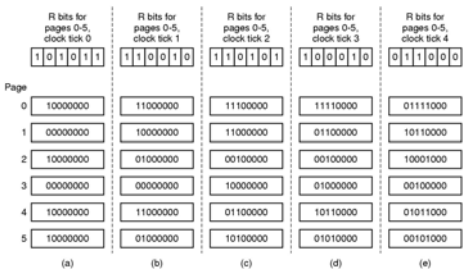
- Assume pages used recently will used again soon
  - throw out page that has been unused for longest time
- Must keep a linked list of pages
  - most recently used at front, least at rear
  - update this list every memory reference !!
- Alternatively keep counter in each page table entry
  - choose page with lowest value counter
  - periodically zero the counter

Simulating LRU in Software (1)



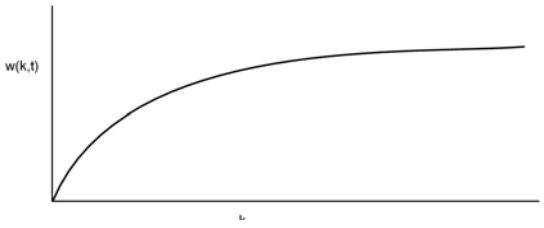
LRU using a matrix – pages referenced in order 0,1,2,3,2,1,0,3,2,3

Simulating LRU in Software (2)



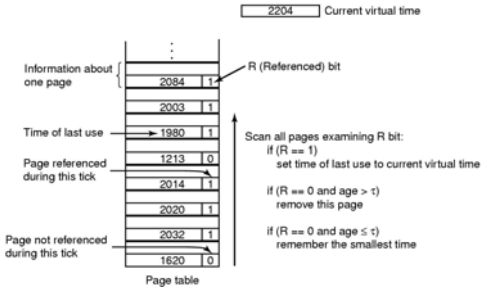
- The aging algorithm simulates LRU in software
- Note 6 pages for 5 clock ticks, (a) – (e)

The Working Set Page Replacement Algorithm (1)



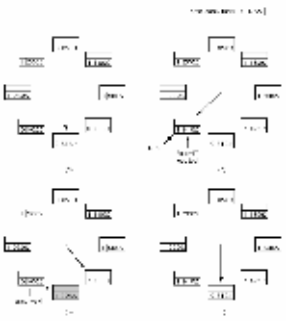
- The working set is the set of pages used by the *k* most recent memory references
- w(k,t) is the size of the working set at time, *t*

The Working Set Page Replacement Algorithm (2)



The working set algorithm

The WSClock Page Replacement Algorithm

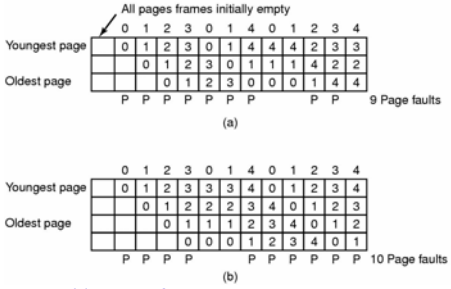


Operation of the WSClock algorithm

Review of Page Replacement Algorithms

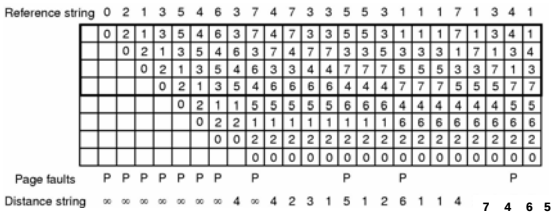
Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not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

Modeling Page Replacement Algorithms Belady's Anomaly



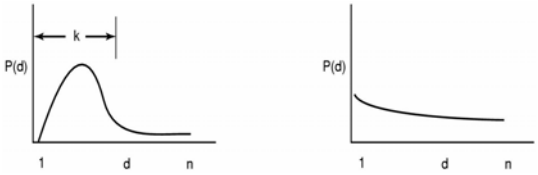
- FIFO with 3 page frames
- FIFO with 4 page frames
- *P*'s show which page references show page faults

Stack Algorithms



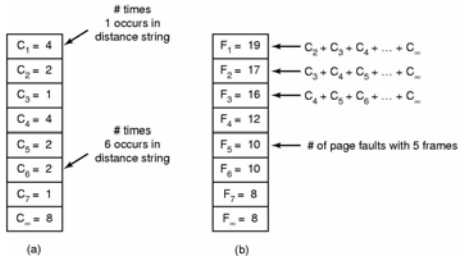
State of memory array, *M*, after each item in reference string is processed

The Distance String



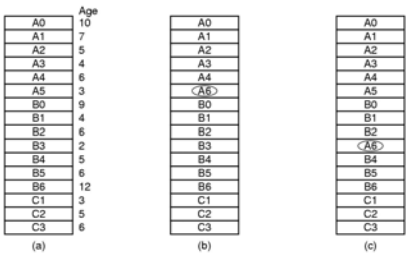
Probability density functions for two hypothetical distance strings

The Distance String



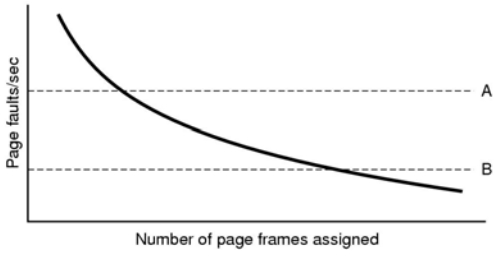
- Computation of page fault rate from distance string
  - the *C* vector
  - the *F* vector

Design Issues for Paging Systems
Local versus Global Allocation Policies (1)



- Original configuration
- Local page replacement
- Global page replacement

Local versus Global Allocation Policies (2)



Page fault rate as a function of the number of page frames assigned

Load Control

- Despite good designs, system may still thrash
- When PFF algorithm indicates
  - some processes need more memory
  - but no processes need less
- Solution :
  Reduce number of processes competing for memory
  - swap one or more to disk, divide up pages they held
  - reconsider degree of multiprogramming

Page Size (1)

Small page size

- Advantages
  - less internal fragmentation
  - better fit for various data structures, code sections
  - less unused program in memory
- Disadvantages
  - programs need many pages, larger page tables

Page Size (2)

- Overhead due to page table and internal fragmentation

$$overhead = \frac{s \cdot e}{p} + \frac{p}{2}$$

page table space

internal fragmentation

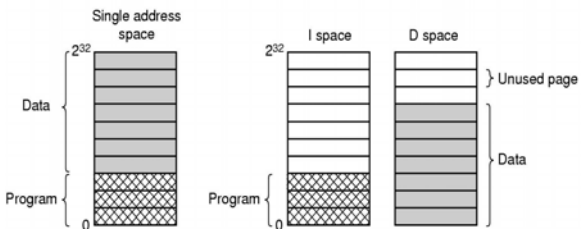
- Where
  - s = average process size in bytes
  - p = page size in bytes
  - e = page entry

Optimized when

$$p = \sqrt{2se}$$

85

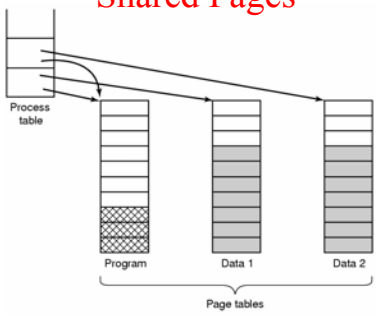
Separate Instruction and Data Spaces



- One address space
- Separate I and D spaces

86

Shared Pages



Two processes sharing same program sharing its page table

87

Cleaning Policy

- Need for a background process, paging daemon
  - periodically inspects state of memory
- When too few frames are free
  - selects pages to evict using a replacement algorithm
- It can use same circular list (clock)
  - as regular page replacement algorithmbut with diff ptr

88

Implementation Issues

Operating System Involvement with Paging

Four times when OS involved with paging

1. Process creation
   - determine program size
   - create page table
2. Process execution
   - MMU reset for new process
   - TLB flushed
3. Page fault time
   - determine virtual address causing fault
   - swap target page out, needed page in
4. Process termination time
   - release page table, pages

89

Page Fault Handling (1)

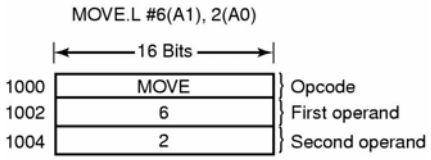
1. Hardware traps to kernel
2. General registers saved
3. OS determines which virtual page needed
4. OS checks validity of address, seeks page frame
5. If selected frame is dirty, write it to disk

90

Page Fault Handling (2)

6. OS brings schedules new page in from disk
7. Page tables updated
- Faulting instruction backed up to when it began
6. Faulting process scheduled
7. Registers restored
- Program continues

Instruction Backup

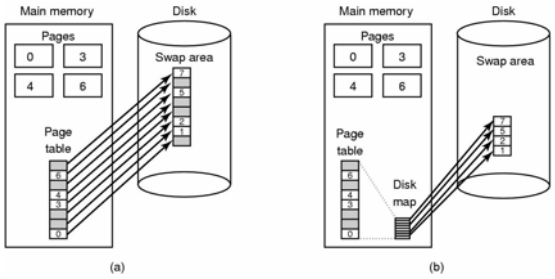


An instruction causing a page fault

Locking Pages in Memory

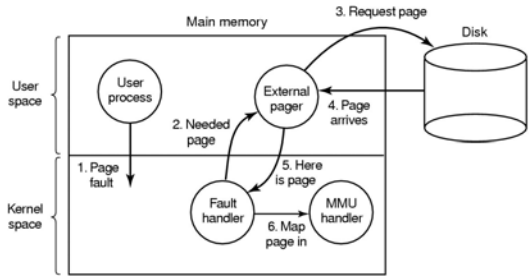
- Virtual memory and I/O occasionally interact
- Proc issues call for read from device into buffer
  - while waiting for I/O, another processes starts up
  - has a page fault
  - buffer for the first proc may be chosen to be paged out
- Need to specify some pages locked
  - exempted from being target pages

Backing Store



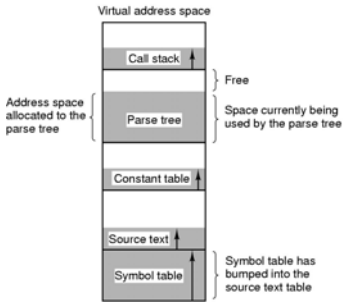
(a) Paging to static swap area
(b) Backing up pages dynamically

Separation of Policy and Mechanism



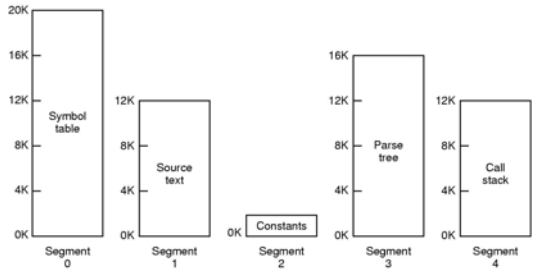
Page fault handling with an external pager

Segmentation (1)



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

Segmentation (2)



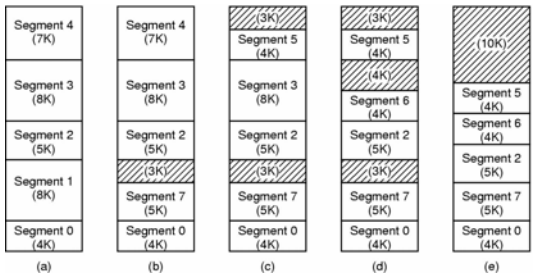
Allows each table to grow or shrink, independently

Segmentation (3)

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

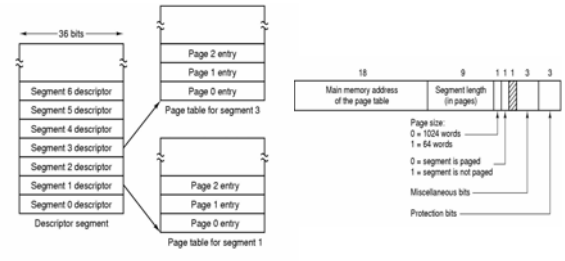
Comparison of paging and segmentation

Implementation of Pure Segmentation



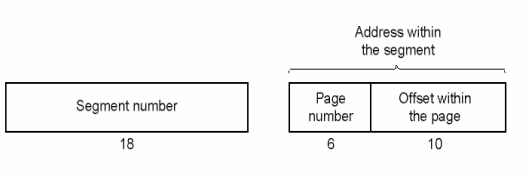
(a)-(d) Development of checkerboarding
(e) Removal of the checkerboarding by compaction

Segmentation with Paging: MULTICS (1)



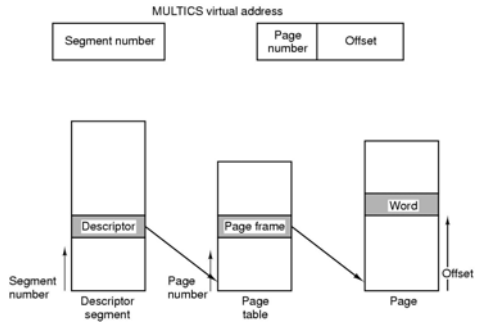
- Descriptor segment points to page tables
- Segment descriptor – numbers are field lengths

Segmentation with Paging: MULTICS (2)



A 34-bit MULTICS virtual address

Segmentation with Paging: MULTICS (3)



Conversion of a 2-part MULTICS address into a main memory address

Segmentation with Paging: MULTICS (4)

Comparison field: Segment number	Virtual page	Page frame	Protection	Age	Is this entry used?
4	1	7	Read/write	13	1
6	0	2	Read only	10	1
12	3	1	Read/write	2	1
					0
2	1	0	Execute only	7	1
2	2	12	Execute only	9	1

- Simplified version of the MULTICS TLB
- Existence of 2 page sizes makes actual TLB more complicated

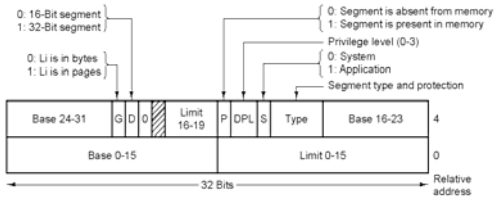
103

Segmentation with Paging: Pentium (1)

A Pentium selector

104

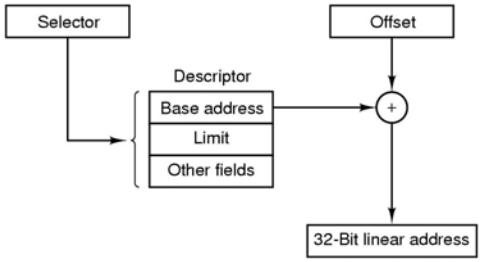
Segmentation with Paging: Pentium (2)



- Pentium code segment descriptor
- Data segments differ slightly

105

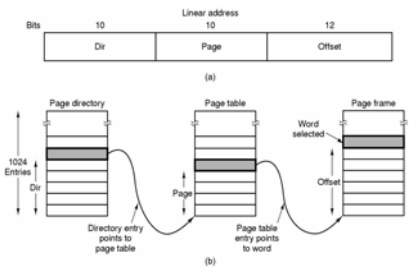
Segmentation with Paging: Pentium (3)



Conversion of a (selector, offset) pair to a linear address

106

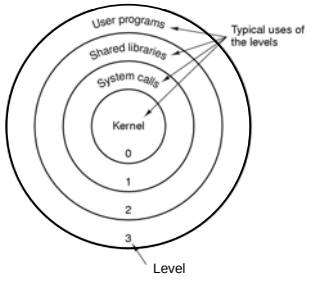
Segmentation with Paging: Pentium (4)



Mapping of a linear address onto a physical address

107

Segmentation with Paging: Pentium (5)



Protection on the Pentium

108