CS 40 Final Study Guide

Ch 7 Memory Management
- 7.1 Memory Management Requirements
- 7.2 Memory Partitioning
- 7.3 Paging
- 7.4 Segmentation

Ch 8 Virtual Memory
- 8.1 Hardware and Control Structures
- 8.2 Operating System Software

Ch 12 File Management
- 12.7 Secondary Storage Management

Assignments

- 8 Address Translation
  (Linear-to-Physical Address Translation)
- 7 Page Replacement
  (Virtual Memory Page Replacement Algorithms)

Ch 4 Threads
- 4.1 Processes and Threads

Assignment 8 Address Translation

Linear-to-Physical Address Translation Steps

1. Determine page # the address falls in
2. Calculate offset from starting address of page
3. Determine frame # where the page currently resides
4. Calculate physical address by adding offset to starting address of frame

1. ① 2
   ② 12287 - 8192 = 4095
   ③ 6
   ④ 24576 + 4095 = **28671**

2. ① 3
   ② 12288 - 12288 = 0
   ③ 0
   ④ 0 + 0 = **0**

3. ① 6
   ② 25000 - 24576 = 424
   ③ **n/a**
   ④

4. ① 9
   ② 39151 - 36864 = 2287
   ③ 5
   ④ 20480 + 2287 = **22767**

5. ① 11
   ② 46000 - 45056 = 944
   ③ 7
   ④ 28672 + 944 = **29616**

Physical-to-Linear Address Translation Steps

① Determine frame # the address falls in

② Calculate offset from starting address of frame

③ Determine page # that currently resides in the frame

④ Calculate linear address by adding offset to starting address of page

6. ① 0

② 0 - 0 = 0

③ 3

④ 12288 + 0 = **12288**

7. ① 6

② 28671 - 24576 = 4095

③ 2

④ 8192 + 4095 = **12287**

8. ① 7

② 28672 - 28672 = 0

③ 11

④ 45056 + 0 = **45056**

9. ① 7

② 30000 - 28672 = 1328

③ 11

④ 45056 + 1328 = **46384**

10. ① 15

② 64535 - 61440 = 3095

③ **n/a**

④

In Class Example - Virtual Memory / Memory Exhaustion (memory3.c)

Example Output of free -m : (-m displays amount in megabytes)

	total	used	free	shared	buff/cache	available
Mem:	3951	601	1912	72	1438	2971
Swap:	4095	0	3978			

Total RAM : 3951 MB

Total Swap ("simulated RAM") : 4095 MB

Assume Used : 601 MB is memory used by the OS

∴ 3951 - 601 = 3350 MB RAM and 4095 MB Swap is available to the program

Slowdown point : **3350 MB**

Terminate point : 3350 + 4095 = **7445 MB**

Assignment 7 Page Replacement

Replacement Policy (8.1)

- Basic Algorithms
  - Optimal policy: selects for replacement that page for which the time to next reference is the longest (impossible to implement)
  - Least recently used (LRU): replaces the page in memory that has not been referenced for the longest time
  - First-in-first-out (FIFO): treats the page frames allocated to the process as a circular buffer, and pages are removed in round-robin style
  - Clock policy: requires additional bit with each frame, referred to as the use bit; tries to approximate the performance of LRU while imposing little overhead

Assignment - Virtual Memory Page Replacement Algorithms

Consider the following virtual page reference sequence: page 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3. This indicates that these particular pages need to be accessed by the computer in the order shown. Consider each of the following 4 algorithm-frame combinations:

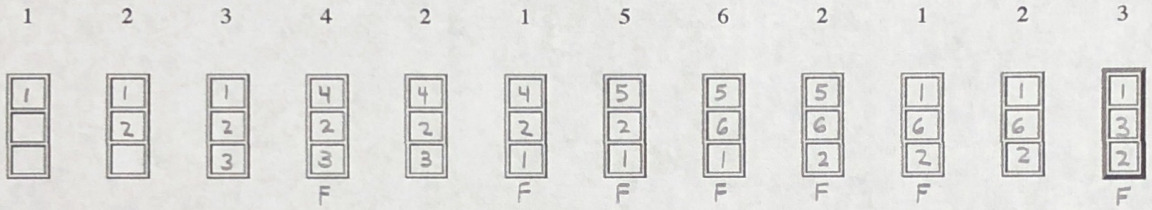
- LRU with 3 frames
- FIFO with 3 frames
- LRU with 4 frames
- FIFO with 4 frames

Print a copy of this page. For each of the 4 combinations, below, move from left to right as the virtual page numbers are requested in sequence. Put each virtual page into one of the frames by writing its number there (initially while empty frames remain, load them from top down). When all frames are already occupied by other pages, choose the right page to displace according to the applicable algorithm (LRU or FIFO) and mark the event with an F for Fault. (Do not count a fault when loading a missing page at a time when there is a frame unoccupied, in other words on the first 3 or 4 loads.) When finished, total the number of page faults and write it in where indicated.

Submit the printout. The assignment will be graded on 8 items: the 4 final page configuration figures at the extreme right (correct or incorrect), and the 4 page fault totals written (correct or incorrect). Please work carefully.

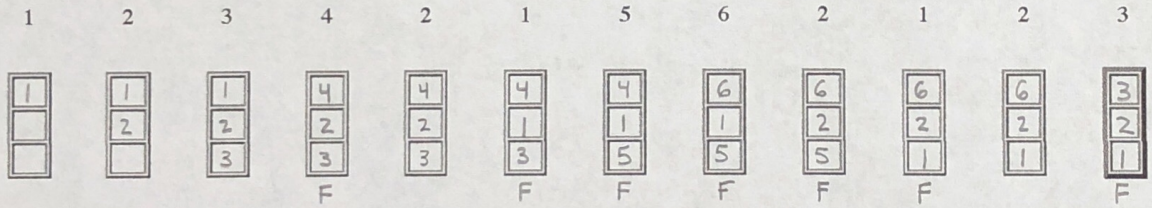
THREE Page Frames

Least-recently-used (LRU) method:



Number of page faults for LRU/3: 7

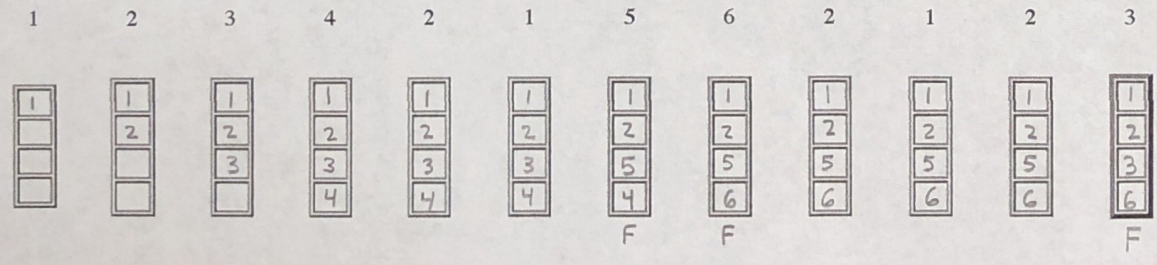
First-in-First-out (FIFO) method:



Number of page faults for FIFO/3: 7

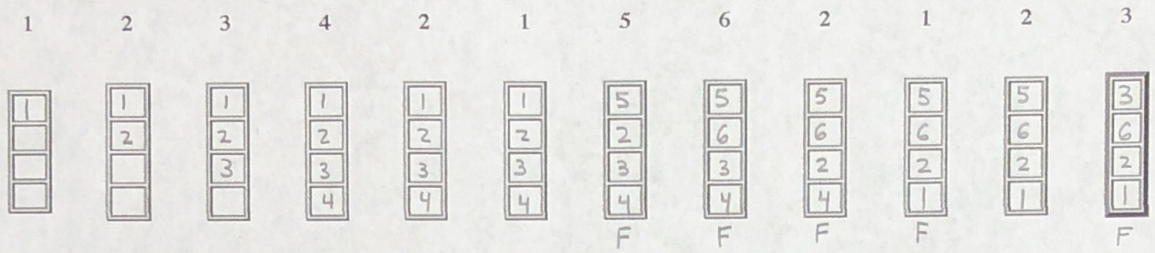
FOUR Page Frames

Least-recently-used (LRU) method:



Number of page faults for LRU/4: 3

First-in-First-out (FIFO) method:



Number of page faults for FIFO/4: 5

Program Relocation

- Ref. Code Relocation write up (home page)

Address Reference Adjustment Upon Code Relocation

Relocation of programs generally calls for a few internal adjustments. Parts of a program (certain of its instructions) typically reference addresses within the program. Obviously, if the program as a whole gets shifted from one place in memory to another (i.e., relocated), every portion of the program moves with it. So any part of the program that accounts for the address of the program, must account for the change of address of the program. (Ch 7, Appendix 7A)

Here is an example. It is a machine language program whose load point is at memory location 500. Suppose it is reloaded at 850 instead, shown in the middle column. Then the operands of the 4 instructions have to change, b/c they reference the data portion of the program and it has moved. In general, if the program is relocated to an address x, the needed adjustments in terms of x are shown in the rightmost column.

	Initial memory:	If relocated to 850:	If relocated to x:
500	1 505	850 1 855	x 1 [x+5]
501	3 506	851 3 856	x+1 3 [x+6]
502	5 507	852 5 857	x+2 5 [x+7]
503	2 504	853 2 854	x+3 2 [x+4]
504	0007	854 0007	x+4 0007
505	0008	855 0008	x+5 0008
506	0002	856 0002	x+6 0002
507	0003	857 0003	x+7 0003

Ch 7 Outline

7.1 Memory Management Requirements

- Relocation: - the programmer doesn't know where the program will be placed in memory when it is executed (it may be relocated due to swapping)
  - memory references must be translated to the actual physical memory address
- Protection: - processes should not be able to reference memory locations in another process without permission
  - impossible to check absolute addresses at compile time - must be checked at run time
- Sharing: - allow several processes to access the same portion of memory
  - better to allow each process access to the same copy of the program rather than have their own separate copy
- Logical Organization: - memory is organized linearly (usually)
  - programs are written in modules (modules can be written and compiled independently) - different degrees of protection given to modules (read-only, execute-only) - share modules among processes - segmentation helps here
- Physical Organization: - cannot leave the programmer w/ responsibility to manage memory
  - memory available for a program plus its data may be insufficient (overlaying is one solution but time consuming to program) - programmer does not know how much space will be available

7.2 Memory Partitioning (Pre-Virtual Memory)

- Fixed Partitioning
- Dynamic Partitioning
- Buddy System: - entire space treated as a single block of 2^U
  - if a request of size s where $2^{U-1} < s <= 2^U$, then entire block is allocated
  - otherwise block is split into two equal buddies
  - process continues until smallest block greater than or equal to s is generated
- Relocation:
  - when program loaded into memory the actual (absolute) memory locations are determined
  - a process may occupy different partitions which means different absolute memory locations during execution (swapping, compaction)
  - Addresses
    - Logical: reference to a memory location independent of the current assignment of data to memory
    - Relative: address expressed as a location relative to some known point
    - Physical or Absolute: the absolute address or actual location in main memory

7.3 Paging

- frame : a fixed-length block of main memory
- page : a fixed-length block of data that resides in secondary memory; a page of data may temporarily be copied into a frame of main memory
- partition memory into small equal fixed-size chunks and divide processes into the same size chunks; the chunks of a process are called pages; the chunks of memory are called frames
- operating system maintains a page table for each process
  - contains the frame location for each page in the process
  - memory address consist of a page # and offset within the page

7.4 Segmentation

- segment: a variable-length block of data that resides in secondary memory
- an entire segment may temporarily be copied into an available region of main memory (segmentation) or the segment may be divided into pages which can be individually copied into main memory (combined segmentation and paging)
- a program can be subdivided into segments
  - segments may vary in length
  - there is a maximum segment length
- addressing consists of two parts
  - a segment number and
  - an offset
- segmentation is similar to dynamic partitioning

Ch 8 Outline

8.1 Hardware and Control Structures

- 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 memory 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 assigned to a process
- real address: the address of a storage location in main memory
- Key points in Memory Management
  1) Memory references are logical addresses dynamically translated into physical addresses at run time
     - A process may be swapped in and out of main memory occupying different regions at different times during execution
  2) A process may be broken up into pieces that do not need to be located contiguously in main memory
- Breakthrough in Memory Management
  - If both of these two characteristics are present, then it is not necessary that all of the pages/segments of a process be in main memory during execution
  - If the next instruction, and the next data location are in memory then execution can proceed (at least for a time)
- Execution of a Process
  - OS brings into MM a few pieces of the program
  - Resident set: portion of process that is in MM
  - An interrupt is generated when an address is needed that is not in MM
  - OS places the process in a blocking state

- Piece of process that contains the logical address is brought into MM
  - OS issues a disk I/O read request
  - Another process is dispatched to run while the disk I/O takes place
  - An interrupt is issued when disk I/O complete which causes the OS to place the affected process in the Ready state
- Implications of this New Strategy
  - More processes may be maintained in MM
    - only load in some of the pieces of each process
    - with so many processes in MM, it is very likely a process will be in the Ready state at any particular time
  - A process may be larger than all of MM
- Real and Virtual Memory
  - Real memory: MM, the actual RAM
  - Virtual memory: memory on disk, allows for effective multiprogramming and relieves the user of tight constraints of MM
- Locality and Virtual Memory
  - Thrashing: state in which the system spends most of its time swapping pieces rather than executing instructions
    - To avoid this, the OS tries to guess which pieces are least likely to be used in the near future
      - guess is based on recent history
  - Principal 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
    - ∴ it is possible to make intelligent guesses about which pieces will be needed in the future
    - this suggests that virtual memory may work efficiently
  - A processes Performance in VM Environment
    - Note that during the lifetime of the process, references are confined to a subset of pages
  - Support Needed for Virtual Memory
    - Hardware must support paging and segmentation
    - OS must be able to manage the movement of pages/segments between secondary and MM

Ch 8 Outline cont'd

- Paging
  - Each process has its own page table
  - Each page table entry contains the frame number of the corresping page in MM
  - Two extra bits are needed to indicate:
    - whether the page is in MM or not - P (present)
    - whether the contents of the page has been altered since it was last loaded - M (modified)
  - Page Tables
    - Page tables are also stored in virtual memory
    - When a process is running, part of it's page table is in MM
  - Page tables grow proportionately
    - a drawback of page tables is that their size is proportional to the virtual address space
    - An alternative is inverted page tables
- Inverted Page Table
  - Used on PowerPC, UltraSPARC, and IA-64 architecture
  - Page # 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
  - Each entry in the page table includes:
    - Page #
    - Process identifier
    - Control Bits
    - Chain Pointer
- Translation Lookaside Buffer
  - Each virtual memory reference can cause two physical memory accesses
    - one to fetch the page table
    - one to fetch the data
  - To overcome this problem a high-speed cache is set up for page table entries
    - Called a TLB
    - Contains page table entries that have been most recently used

- TLB Operation
  - Given a virtual address, processor examines the TLB
  - If page table entry is present (TLB hit)
    - the frame # is retrieved and the real address is formed
  - If page table entry is not found in the TLB (TLB miss)
    - the page # is used to index the process page table

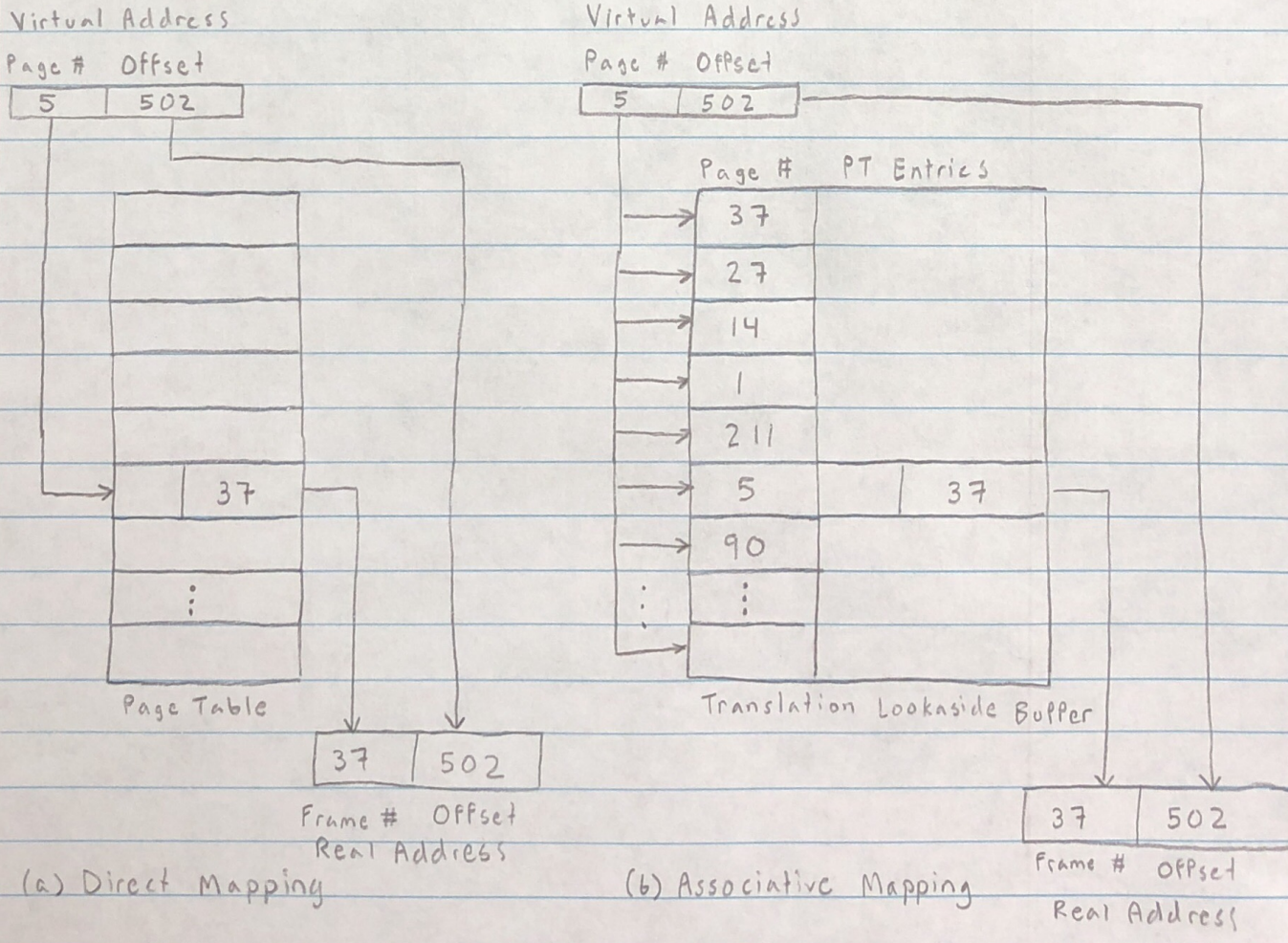
⋮

Associative vs. Indexed Mapping

Associative Mapping: b/c the TLB contains only some of the entries in the full page table, we cannot simply index into the TLB based on page #; instead, each entry in the TLB must include the page # 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 #

Indexing aka Direct Mapping: the page # is used to index into the page table; that is, each page table entry does not include the page # like it does in associative mapping, rather, it is implicit b/c the entries are in order

Fig 8.8 Direct versus Associative Lookup for Page Table Entries



Process C page table

Free frame list

Fig 7.10

Page Tables

7.3 Paging

- The OS maintains a page table for each process
- The page table shows the frame location for each page of the process
- Memory address consists of a page # and offset within the page

8.1 Hardware and Control Structures

- Paging
  - A page table is also needed for a virtual memory scheme based on paging, but in this case the page table entries become more complex

Page table entry

P	M	Other control bits	Frame number

P = present bit
M = modified bit

Fig 8.1

↳ Note: the page # is implicit in the PTE by the ordering of the entries

- B/c only some of the pages of a process may be in main memory, a bit is needed in each page table entry to indicate whether the corresponding page is present (P) in MM or not
  - If the bit indicates the page is in memory, then the entry also includes the frame number of that page
- The PTE includes a modify (M) bit, indicating whether the contents of the corresponding page have been altered since the page was last loaded into MM
  - If there has been no change, then it is not necessary to write the page out when it comes time to replace the page in the frame that it currently occupies
- Page Table Structure
  - Fig 8.2 Address Translation in a Paging System
    - When a process is running, a register holds the starting address of the page table for that process
    - The page # of a virtual address is used to index that table and look up the corresponding frame #
    - This is combined with the offset portion of the virtual address to produce the desired real address

Virtual Address

Page Number	Offset

↳ Note: the page # is used to index into the page table (Indexing aka Direct Mapping)

- Most virtual memory schemes store page tables in virtual memory rather than main memory due to their size
- When a process is running, part of its page table is in MM
- Some processors make use of a two-level scheme to organize large page tables

- Inverted Page Table
  - A drawback of the type of page tables just discussed is that their size is proportional to that of the virtual address space
  - An alternative is Inverted Page Tables
  - Page # portion of a virtual address is mapped into a hash value
  - Hash value points to inverted page table
  - Fixed portion of real memory is required for the tables regardless of the # of processes

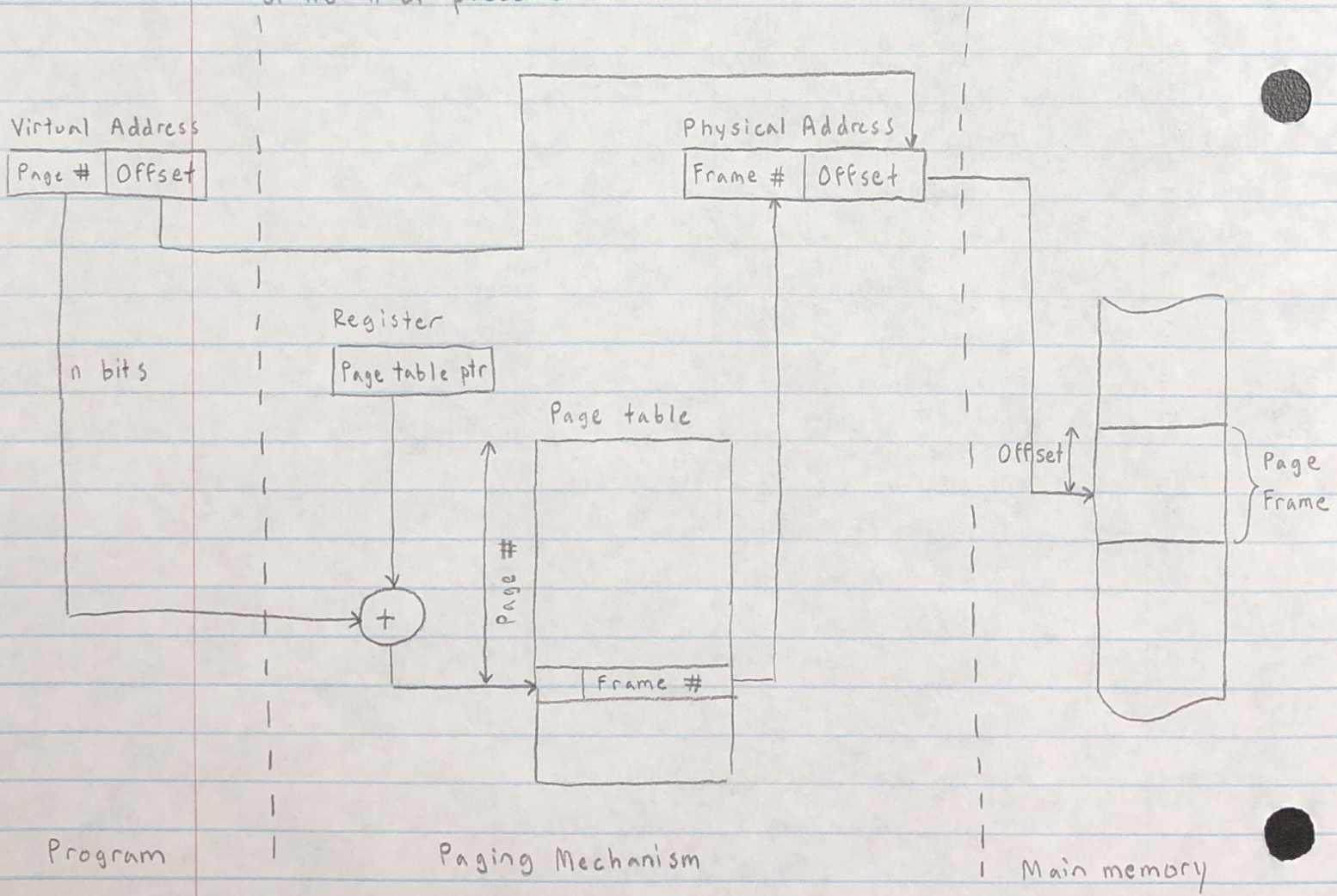


Fig 8.2 Address Translation in a Paging System

Dynamic Partitioning

Description - Partitions are created dynamically, so that each process is loaded into a partition of exactly the same size as that process.

Strengths
- No internal fragmentation
- More efficient use of MM

Weaknesses - Inefficient use of processor due to the need for compaction to counter external fragmentation

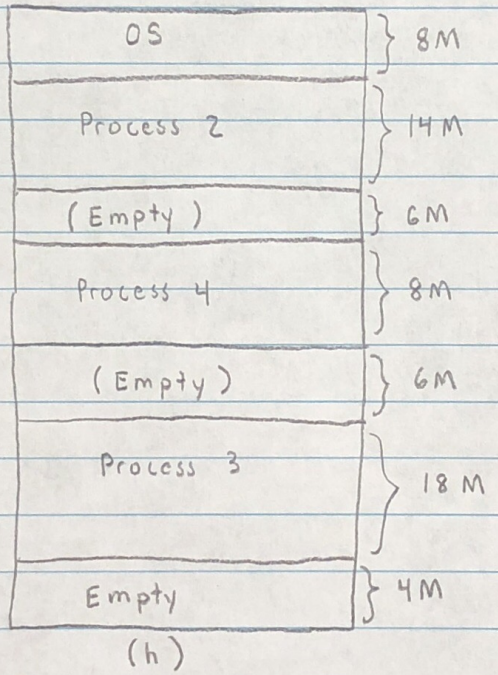


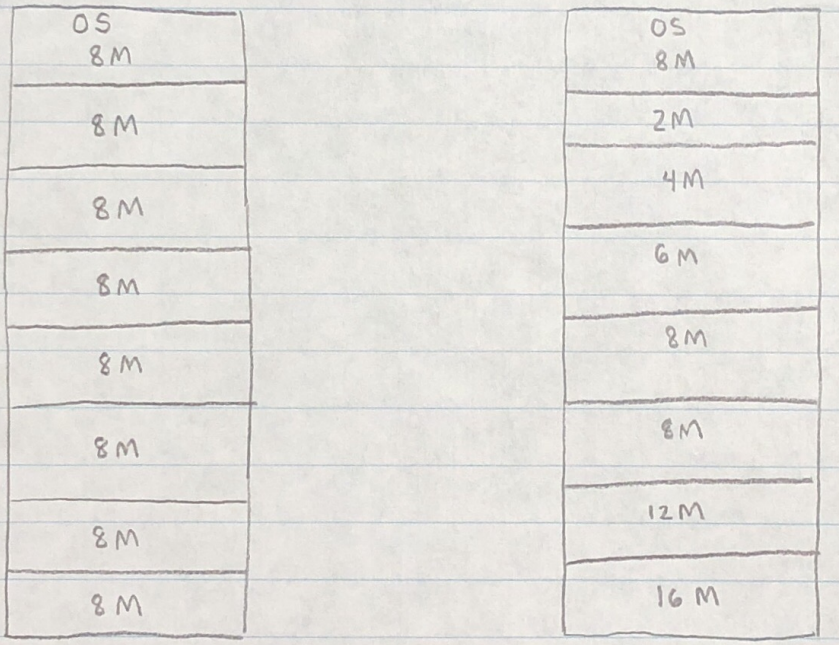
Fig 7.4 The Effect of Dynamic Partitioning

Fixed Partitioning

Description
- MM is divided into a number of static partitions at system generation time
- A process may be loaded into a partition of equal or greater size

Strengths
- Simple to implement
- Little OS overhead

Weaknesses
- Inefficient use of memory due to internal fragmentation
- Maximum number of active processes is fixed



(a) Equal-size partitions (b) Unequal-size partitions

Fig 7.2 Fixed Partitioning of 64-Mbyte Memory

Fragmentation

Technique	Strengths	Weaknesses
Fixed Partitioning		- Internal fragmentation
Dynamic Partitioning	- No internal fragmentation	- External fragmentation
Simple Paging	- No external fragmentation	- A small amount of internal fragmentation
Simple Segmentation	- No internal fragmentation	- External fragmentation
Virtual Memory Paging	- No external fragmentation	
Virtual Memory Segmentation	- No internal fragmentation	

- internal fragmentation: when there is wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition

- external fragmentation: when memory external to all partitions becomes increasingly fragmented; one technique to overcome this is compaction

Extra Credit

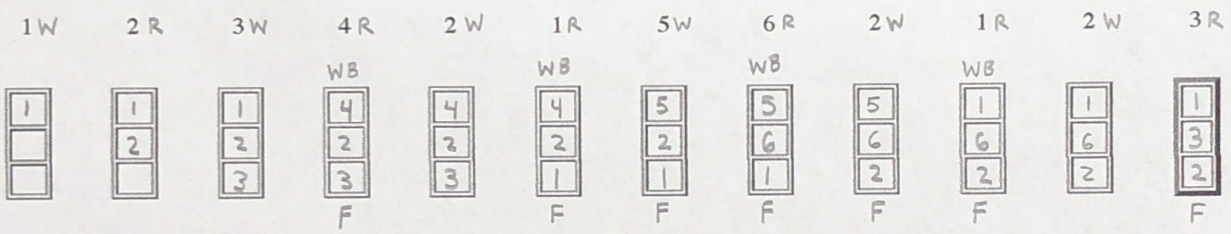
Under which of the following 4 scenarios is operating the memory going to be the least expensive? (Least # of writes out to disk)

- Determine and count the # of any writes out to disk
- Recall that when replacing a page, you need to write it out to disk if the previous page references include a Write (W)

THREE Page Frames

Least-recently-used (LRU) method:

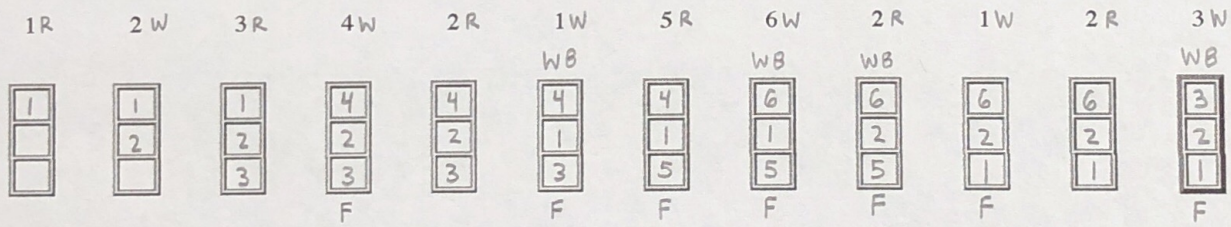
(1)



Number of page faults for LRU/3: 7 Number of write backs 4

First-in-First-out (FIFO) method:

(2)

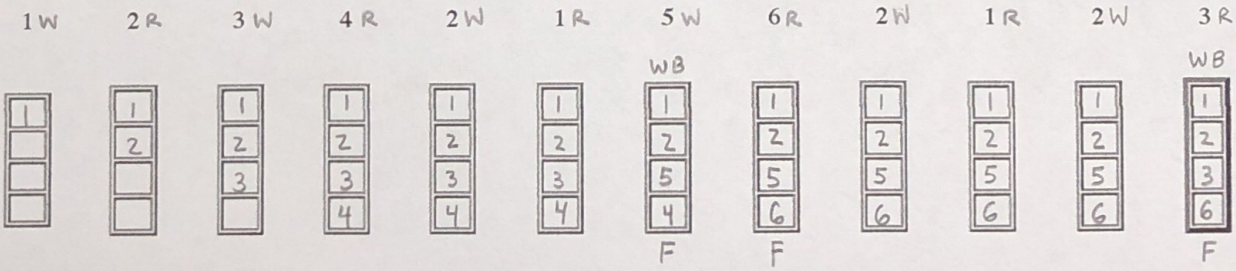


Number of page faults for FIFO/3: 7 Number of write backs 4

FOUR Page Frames

Least-recently-used (LRU) method:

(3)



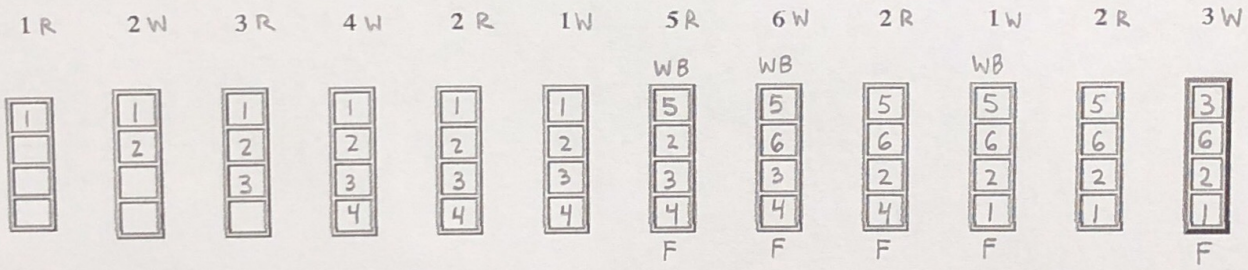
5/26/2018

Virtual Memory - Page Replacement

Number of page faults for LRU/4: 3 Number of write backs 2

First-in-First-out (FIFO) method:

(4)



Number of page faults for FIFO/4: 5 Number of write backs 3

∴ #3 is the least expensive in terms of # of writes out to disk (2)

Ch 12 Outline

12.7 Secondary Storage Management

- File Allocation Method
  - three in common use: contiguous, chained, and indexed
- Contiguous Allocation
  - Single set of blocks is allocated to a file at the time of creation
  - Only a single entry in the file allocation table
    - starting block and length of the file
  - External fragmentation will occur
    - need to perform compaction
- Chained Allocation
  - Allocation on basis of individual block
  - Each block contains a pointer to the next block in the chain
  - Only single entry in the file allocation table
    - starting block and length of file
  - No external fragmentation
  - Best for sequential files
- Indexed Allocation
  - File allocation table contains a separate one-level index for each file
  - The index has one entry for each portion allocated to the file
  - The file allocation table contains block number for the index
  - Allocation may be either
    - fixed size blocks or
    - variable size blocks
  - Allocating by blocks eliminates external fragmentation
  - Varinble sized blocks improves locality
  - Both cases require occasional consolidation

Context - put the different types of memory management we are talking about into this context

- fixed partition mem. mgmt
  - equal partition size
  - unequal partition size
- dynamic (= variable partition size) mgmt
- simple segmentation (variable size pieces)
- simple paging (fixed size pieces)

always contiguous

entire program loaded into memory

- overlay (large user-defined pieces)
- virtual memory
  - v.m. segmentation
  - v.m. paging

program partially loaded