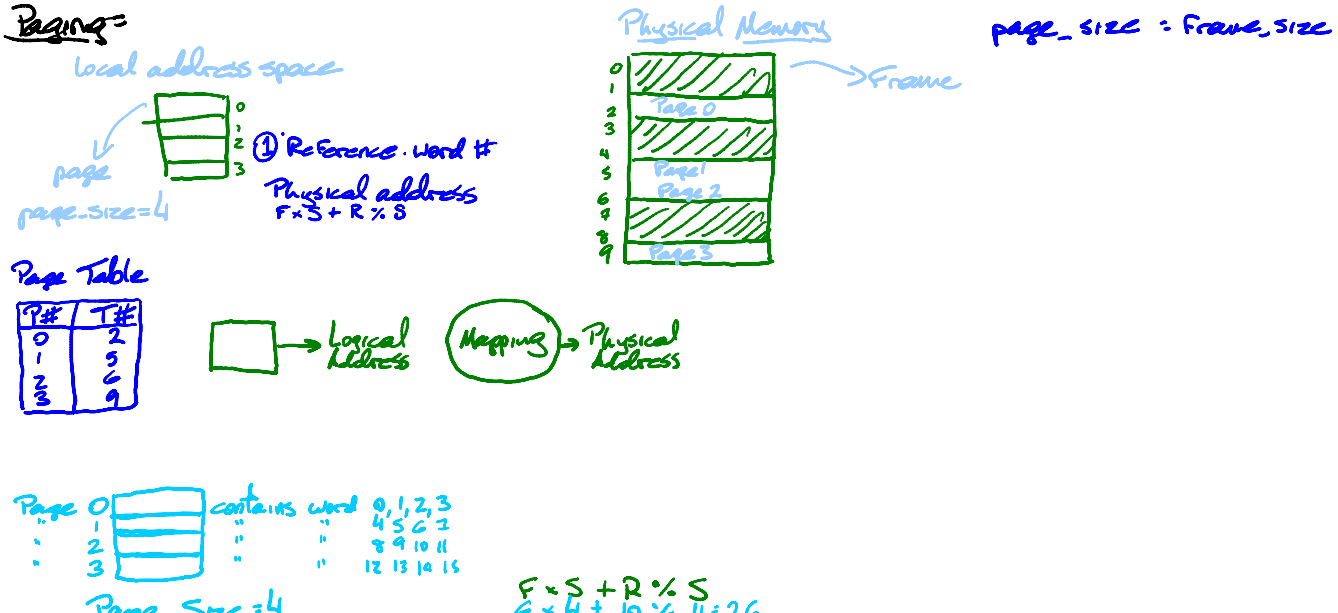
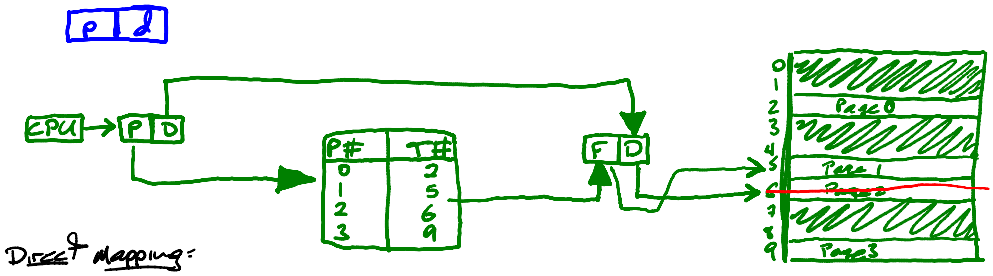
4-30-09

Thursday, April 30, 2009
5:56 PM

Paging:



② MMU Support



Direct Mapping:

cost: 1 mem access + 1 mem access = 2 mem accesses.

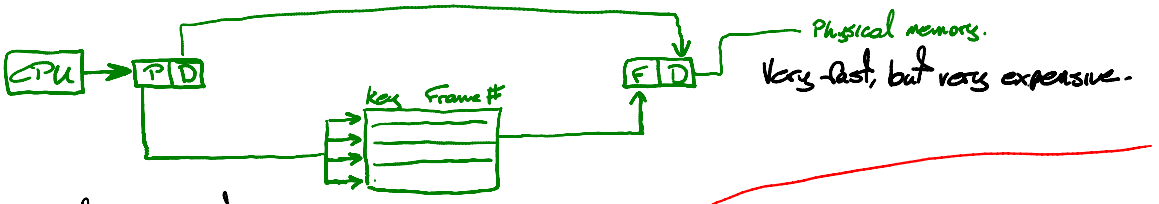
↓ for page table ↓ actual data

The time is double, which is not efficient.

Page Table in Main Memory = Direct Mapping.

CPU Reg PTBR

Page Table in Associative Registers:

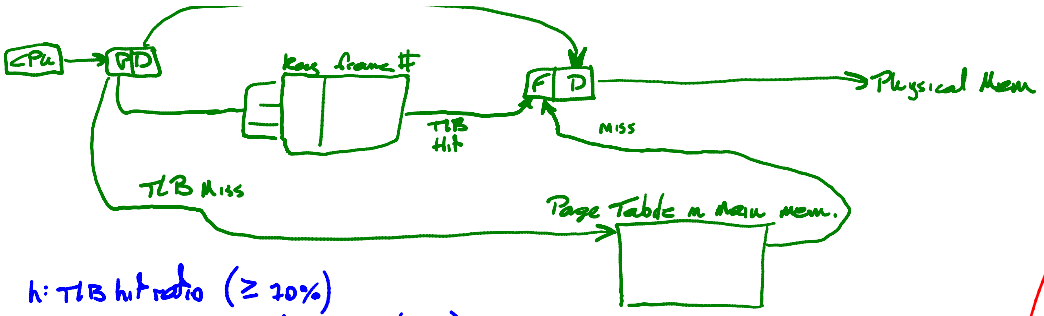


Very fast, but very expensive.

cost: 1 register search → 0 + 1 memory access → actual data

Combine the 2 mappings:

Part of the page table is in TLB
Part of the page table is in Main Memory.



h: TLB hit ratio (≥ 20%)

eat = h × TLB Hit Time + (1−h) × TLB Miss Time

TLB hit time: 1 register search + 1 mem access
→ 0 → Actual data

TLB miss time: 1 register search + 1 mem access + 1 mem access
→ 0 → P.T. → Actual Data

On website: 2 links w/ exercises.

P D

F D

Exam Q's: explain advantages & disadvantages of specific mapping.
compute effective access time.

CPU Reg = 8 bits

2	4
3 bits	5 bits

P D

010	01001

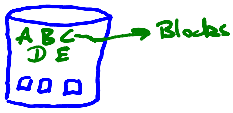
F D

Demand Paging

Page is broken in main mem only when its demanded.

If you have a reference to a page not in main mem, problem: CPU can't access disk.
∴ Interrupt → Page Fault

Page Fault: Huge overhead. OS senses Page fault, OS brings demanded page from disk to main mem



Logical addr space

0	A
1	B
2	C
3	D
4	E

page_size =
frame_size =
block_size =

V/I		Page Table
0	0	disk addr
0	1	disk addr
1	2	5
0	3	disk addr
1	4	2

Physical Memory

0	
1	
2	Page E
3	
4	
5	Page 2
6	
7	
8	
9	

load C

V/I Bit: demand paging: 1 (valid) if reference is legal and the page is in main memory.
"Valid/Invalid"

Project: 1 day late: −3
2 days: −6
3 days: −10

4-21-09

Tuesday, April 21, 2009
5:45 PM

Memory Management:

Contiguous Allocation: entire address space of process must be in main memory
address space must take a contiguous block in main memory

Paging: entire address space of process must be in main mem
address space doesn't need to be in contiguous block

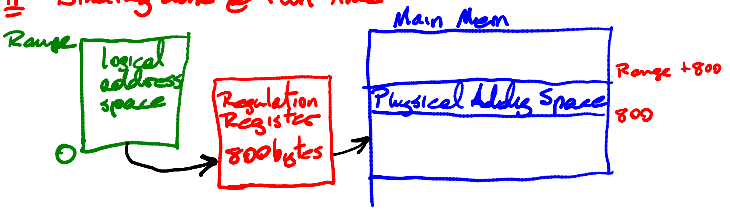
Similar to Paging: segmentation

Demand Paging: uses virtual memory concept
a process can execute with partial address space in main memory
doesn't need to be contiguous

Logical Address: CPU → logical address

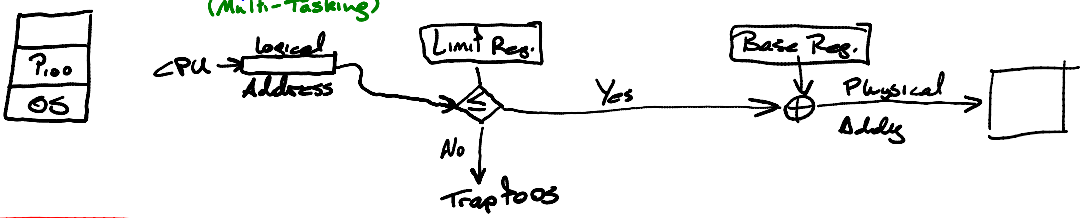
Physical Address: actual addrs in main mem

Logical addrs → Physical addrs (mapping)
If Binding done @ run time



Contiguous Allocation → Single-Partition Allocation (Single-Tasking)
→ Multiple-Partition Allocation (Multi-Tasking)

S.P.A.:



M.P.A.: Job Queue

MPA → Fixed Partitioning (IBM OS/360) → Equal Size
→ Unequal Size
→ Dynamic Partitioning

Fixed Partitioning: Partition can contain @ most 1 proc.
1 proc. can take @ most 1 partition

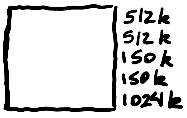
→ Equal Size:

Creates some problems: ① 

412k are wasted
(aka = "internal fragmentation")

② If proc > part. size, proc. can't be loaded

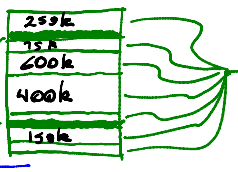
→ Unequal Size:



Dynamic Partitioning: Eliminates internal fragmentation

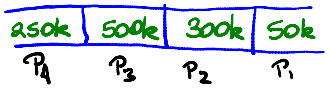
If P300 is done, then we have block of free mem
P30 comes in, half size of P300, P30 can use ½ the block and the other half becomes a new partition.

We have varying blocks of mem



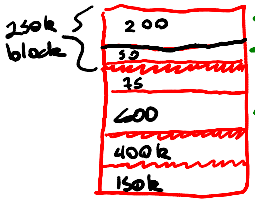
→ Free blocks of memory
Processes ←
OS keeps track of all free blocks

Job Queue:



1ST Proc can go in any free block - How can it decide?

First Fit: Pick 1ST block that fits the process. (50k proc can go into 250k block)



← 200k block
← 1ST proc
← 2nd proc, was 600k block, now 300k
3rd proc can't fit anywhere, can't run

Best Fit: Pick smallest block that fits proc

Worst Fit: largest process

Same Job Queue

250k
75k ← 1ST Proc (50k)
600k ← Proc (-0k)
150k

300k proc → 250k 550k ← 600k ← 50k proc

250k
75k
600k
400k
150k

External fragmentation: in Tolaas's PDF

Know internal vs. external fragmentation

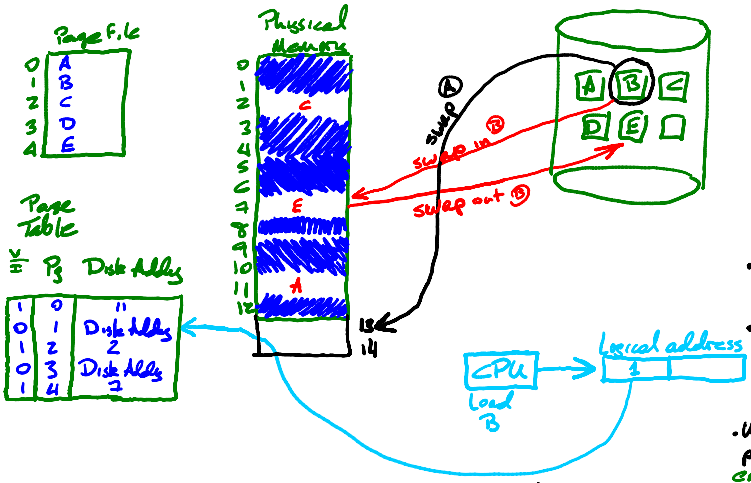
Paging: mem partitioned into pages, non-contiguous.

Thursday: Review on 3 examples

5-05-09

Tuesday, May 05, 2009
5:59 PM

Demand Paging



In case there is a page fault;
OS checks if ∃ a free frame

A) Yes
- Choose a free frame (8, 13, 14)
- Pick 13
- Swap in demanded page
- Update page table
  entry = 1
  V/I = 1
  F#: Disk Addy
  13
- Restart execution

B) No
- Choose victim frame
  Global Allocation: Proc. takes frame from another proc. (0, 1, 3, 4...)
  Local Allocation: Proc picks frame of executing proc. (2, 7, 10)
  (FIFO, LRU, Optimal)
- Based on Local Allocation, pick (7)
- Swap-out (save) victim frame so it can be overwritten.
- Update page table
  entry = 4
  V/I = 1 → 0
  F# = 7 → Disk Address
- Swap in demanded page
- Update Page Table
  entry = 1
  V/I = 0 → 1
  F# = Disk address → F
- Restart execution.

Check Valid/Invalid Bit. If 0 → Interrupt
OS is invoked → checks cause
→ Page not in main memory
→ Page fault
OS must serve the Page File.

Modify Bit: → 1 if page is dirty (at least 1 bit has been changed)

Page Fault Rate: P (want it to be low.)

20% = 80 were in main mem

effective access time: (1-p) * No. PF Time + p * PF Time (page fault)

e.a.t.
- 1-p: No page fault { P: 2 mem accesses; A: 1 reg. read + 1 mem access; D/Δ: h * TLB Hit + (1-b) TLB Miss }
- p: Yes, page fault
  - β: Yes FF ≈ 1 I/O (swap-in demanded frame)
  - 1-β: No FF
    - Δ: Not modified ≈ 1 I/O (swap in demanded frame)
    - (1-Δ): Yes modified ≈ 2 I/O → Swap-out V.F. → Swap-in D.P.

Principle of Locality, Working Set, & Thrashing

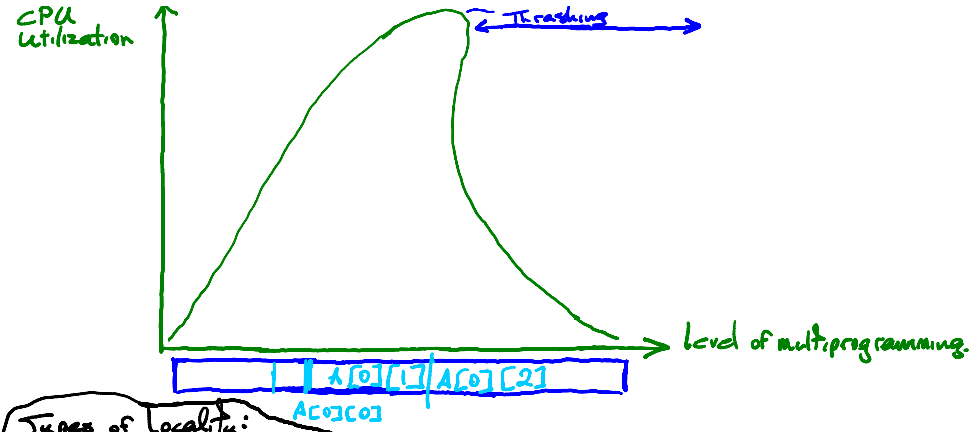
Matrix A in Page 2 (40 elements: 4x10)

init A: 2, 2, ... 2 (40 references in pg 2) 1 cluster
init B: 7, 7, ... 7 (40 references in pg 7) 1 cluster
Compute C: 2, 7, 2, 7, 2, 7, ... 7 (in pg 9)
2, 7, 2, 7, 2, 7, ... 7 (in pg 9)
] → 1 cluster
Size of working set is 1, contains pg 2
size of working set = 3
contains pgs 2, 7, 9

Matrix B in Page 7 (40 elements: 4x10)

AxB in page 9

Ideal situation: have enough frames to handle working sets, otherwise too many page faults.
If too many P.F.'s: Thrashing



Types of Locality:
- Spatial:
- Temporal:

② No deadlock because there don't have a right stick to pick up.

④ Reader/Writer: mention violated condition.

① Use counting sem. for machines.

Student: P(machine) ← Supervisor: P(Done) V(Machine)

```
V(Done)
bin sem mutex = 1;
counting sem machine=5, done=0;
Student:
    P(machine);
    V(Done);
    P(mutex);
    waitingStudents++;
    if(waitingStudents == groupSize)
    {
      for (int i=0; i< groupSize; i++
      {
        V(waiting);
      }
      waitingStudents=0;
      P(waiting);
      V(mutex);
    }
    else
    {
      P(waiting);
      V(mutex);
    }

Supervisor:
    while(true)
    {
      P(Done);
      V(machine);
    }
```

5-07-09

Thursday, May 07, 2009
3:01 PM

Memory Management

- Continuous Allocation:
  - Single and Multitasking
  - Fixed partitioning
  - Dynamic partitioning: First Fit, Best Fit, Worst Fit
  - Internal fragmentation
  - External fragmentation
- Paging:
  1. Word Number (number in binary)
  2. | P | D |
     | F | D |
  - Problems on website
  - 8 pages, 0 to 7 ⇒ need at least 3 bits.
    1024 words (page size): 1st word has offset of 0.
    0-1023 = need 10 bits. 10+3 = 13 bits needed.
- Mapping:
  - Direct Mapping — 2 mem accesses
  - Associative Mapping — 1 register search + 1 mem access.
  - D/A Mapping.
  - h * TLB hit + (1-h) * TLB Miss
    - 1 reg search + 1 mem access
    - 1 reg search + 1 mem access + 1 mem access (PT, Data)
  - 9.10: Consider a paging system w/ page table (direct mapping) stored in mem
    Mem access = 200 ns
    Find eat = 1 mem access + 1 mem access = 2 mem access = 400 millisec.
    (for page table, for data)
  - h = 75% find e.a.t. if TLB/reg search = 0 msec

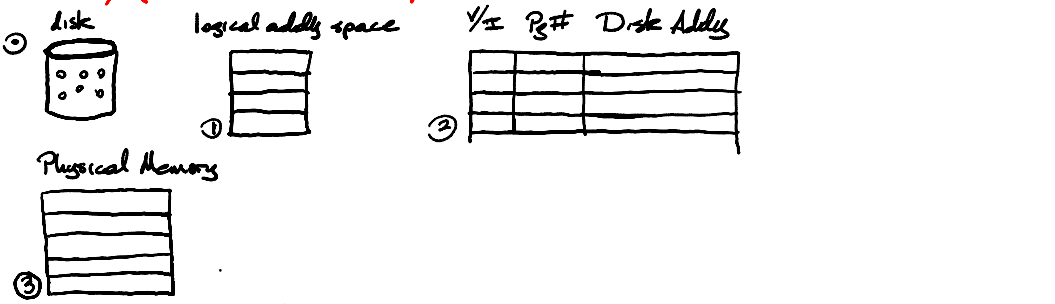
    h * TLB Hit + (1-h) * TLB Miss

    0.75 → 1 reg search + 1 mem access (0 + Data)

    0.25 → 1 reg search + 1 mem access + 1 mem access (0 + PT + Data)

    = (0.75 * (0+200)) + ((0.25) * (0+200+200))

- Demand Paging:



- Page Fault Steps: Load B, Global Allocation
- E.A.T. for Demand Paging:

  eat = (1-p) * No P.Fault + p * Yes P.Fault

  - 1-p → No PF { D, I, D/I }
  - p → Yes PF
    - Yes Free Frame ≈ 1 I/O
    - No Free Frame (replacement)
      - Not modified ≈ 1 I/O
      - Yes Modified ≈ 2 I/O
- Associative Mapping, 8ms for 1 I/O, 20ms if replaced page is modified
  - 1-p → No PF: 1 reg search + 1 mem access = 0
  - p → Yes PF
    - 50% Yes FF or (No FF & not mod) 8ms
    - 50% No FF and yes mod 20ms

  $$p = \frac{eat - \text{No PF}}{\text{Yes PF} - \text{No PF}}$$

  Yes PF = 30% * {Yes PF or (No FF and Not Mod)} + 70% * {No FF and Yes Mod}

YesPF = 30% * (YesPF or (No FF and Not Mod)) + 70% * (No FF and Yes Mod)

(YesPF or (No FF and Not Mod)): 8 ms; (No FF and Yes Mod): 20 ms

Problem

@ 10-10 ⇒ array[100][100] starts @ loc 200

Size of page 200. # pages = 50 pages needed ($\frac{\#\text{elements}}{\text{page\_size}}$)

every instruction fetch will be from page 0

Page Table

every page has a full 2 rows of the array

0	instructions	0-199
1	array[0][0]	200-399 → has array[0][0] → array[1][99] (rows 0 and 1)
2		400-599 → rows 2 and 3
3		600-799 → rows 4 and 5
4		
5		
6		
7		
8		
9		

Physical Memory

Page 0
Page 1 / 3
Page 2

ⓐ By column

Fetch instruction (in page 0), A[0][0] in page 1 → page fault. (page 1 is not in main memory).

Fetch instruction (pg 0), A[1][0] → no page fault because previous loaded pg!

" " " " A[2][0] → Page fault! (page 2)

" " " " A[4][0] → Page fault! (page 3)

Implement Least Recently Used → page 3 replaces page 1

50 page faults in 1 row:

Page:	0	1 PF	0	1	0	2 PF	0	2	0	3 PF
Action:	instruction	A00	inst.	A10	inst.	A20	inst	A30	inst	A40

ⓑ By Rows:

0 inst. A[0,0] (1 PF) 0 inst A[0,1] (1) ..... A[0,99] (1)

0 inst A[1,0] (1) 0 inst A[1,99] (1)

1 page fault in 2 rows: $\frac{100}{2} = 50$ PF

Trashing & Working Set

Principal of Locality

No lecture on Tuesday!