Test review today!

- Race condition
  - Mutex exclusion solves true.
- enforces mutual exclusion
- No deadlocks
- No Starvation
- No unnecessary delay.
- Dekkers and petersons algorithm.
  - Con: busy wait
  - hardware could cause starvation. by prioritization problems

- int semaphore
  - atomic Methods
    - wait
    - Signal

  } atomic operation cannot be interrupted.

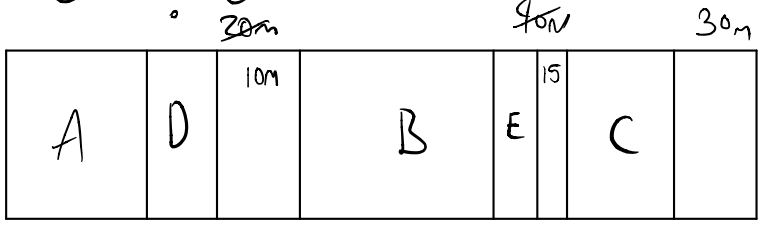
- Deadlocks - only blocked process
  - wait and hold
  - Circular wait
  - non-preemption
  - Mutual exclusion

  Live Lock - process never blocked but not making progress

- Prevention — Conservative
  - Remove one of four stipulations
  - Circular wait
    - remove by making program pick up resources in order
- Avoidance
  -

- Detect
  - Detect algorithms looking at blocked processes

Memory Management



D-10
E-25

External fragmentation - unusable memory gaps or whole in memory

internal fragmentation - too much memory allocated to program

3629
2048
1581
1024

2^{12}

Question B # 6

00001, 111000101101,

page

557
517

00110 11000 01101 - physical address

45
37

13
8

5
4

1

Segmentation fault - outof bounds of memory

- Dining philosopher know it.

- Dedicated cache

KB = 1024 or 2^{10}

page size = 2KB = 2^{11} $2 \cdot 1024$ 2048

16 bits for 3629 2KB page

$\frac{2^{16}}{2^{11}} = 2^5 = 32$ pages

3629

2048 2^{11}

1581

2^{11} 0 1 1 1 0 0 0 1 0 1 1 1

1024 2^{10} $2^{11} + 2^{10} + 256 + 16 + 4 + 2 + 1$

557

2^9

45

2^9 0 0 0 0 1 1 0 0 0 1 0 1 1 0 1

32 2^5

13 2^3

2^3

5 2^2

~~x~~

$2^0 = 1$

2098 2^{16} 2KB 2^{11}

2098

2048 2^{11}

50

2^5 2^5

18

2^4 2^4

2 2^1

0000 | 00000110010

2048 + 32 + 16 + 2

physical address = {00110, 00000110010}

10/26/18 Day 25

- Memory management

Present	Modified	Page #	frame #

page fault - program references an address in a page not in memory resident.

working set - set of pages currently used by process

thrashing - more time paging than processing!

to resolve a page fault

1. save state of process
2. find a free frame
   - if none available make one
   - invoke page replacement algorithm.
3. If page to be replaced has been modified write it to disk.
4. Read page into newly freed frame
5. restart the instruction that caused the page fault

$$test = (1-p)t_{ma} + p\,t_{pf}$$

test = effective memory access
p = probability of a page fault
t_{ma} = time memory access
tpf = time to process page fault.

Example $p = \frac{1}{1000} = 0.001$

$t_{ms} = 1\mu s$

$t_{pf} = 20ms$

$t_{eff} = (1-0.001)(1\mu s) + 0.001(20{,}000$

$= 0.999 + 20 = 20.999\ \mu s \approx 21\ \mu s$

$1.1\ \mu s = (1-p)t_{ma} + p(t_{ps})$

$1.1\mu s = (1ms - 1p + 20000p$

$.1 = 19999$

$\frac{.1}{19999} = P \approx$ 0.000005 = P

Replacement policies

Optimal page replacement algorithm

each page labeled with number of instructus that will execute from that page is referenced. Remove that page that is referenced further in the future

Suppose [1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5] = index of processer

3 frames available

1		
1	2	
1	2	3
1	2	4
1	2	5
3	2	5
3	4	5
⋮	⋮	⋮

7 page faults

First in first out (fifo) page replacement

- pick oldest page, memory reicent the longest

1		
1	2	
1	2	3
4	2	3
4	1	3
4	1	2
5	1	2
5	3	2

9 page faults

$O(n)$

May not be best approach talked here

5	3	4

4 frames FIFO

1			
1	2		
1	2	3	
1	2	3	4
5	2	3	4
5	1	3	4
5	1	2	4
5	1	2	3
4	1	2	3
4	5	2	3

10 page faults

Belady's Anomoly

Second chance page replacement
" clock page replacement"
- avoid closing heavily used pages
- add reference bit to page table.

Present	Modified	ref #	frame #

- Examine ref bit on oldest page if's set - clear it

and move on to the next oldest page.

- repeat
- select oldest page with ref not set.