CS & IT ENGINEERING

Operating System

Virtual Memory Part - 02

Revision

Lecture No.- 12

By- Dr. Khaleel Khan Sir

Recap of Previous Lecture

Topic VM Concept and Implementation

Topics to be Covered

- Topic — Page Repalcement, Thrashing, WS Model
- Topic
- Topic
- Topic
- Topic

Round Robin : To improve responsive / Interactive

TQ = 2

P.NO	A.T	B.T
1	0	7
2	2	4
3	3	2
4	9	1

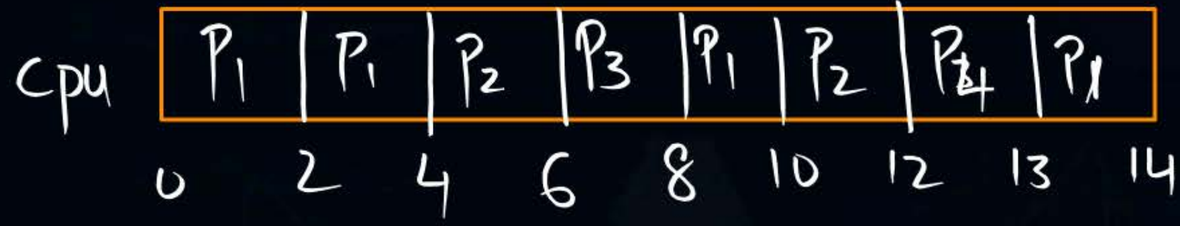
R.Q : P1 ; P2 ; P1 ; P3 ; P2 ; P1 ; P4 , P1 ✓

Favoring New Process

Av. R.T = 0+0+3+3 / 4 = 6/4 = 1.5 ✓

Avg. R.T =

R.Q : P1 ; P1 ; P2 ; P3 ; P1 ; P2 ; P4 ; P1 ;



Av. R.T = 0+2+3+3 / 4 = 8/4 = 2 ✓

Favoring Lower Pid / older Process

Topic : Demand Paging

Assuming a page size of 4 Kbytes and that a page table entry takes 4 bytes, how many levels of page tables would be required to map a 64-bit address space, if the top level page table fits into a single page?

$V.A.S = 2^S$ By

$P.S = 2^x$ By

$PTE = 2^c$ By

$P.S = 4KB = 2^{12}$

$e = 4B = 2^2$

$V.A = 64$ bits

for 'l' level Paging, to accomodate top level PT in Single page/frame

$$2^{S-l.x+l.c} = 2^x$$

$$\left[2^{64-l.12+l.2}\right] = 2^{12}$$

$64 - 10l = 12$

$10.l = 52$

$l = 5.2 = 6$

Topic : Demand Paging

Consider a paged logical address space (composed of 32 pages of 2 Kbytes each) mapped into a 1-Mbyte physical memory space.

a. What is the format of the processor's logical address?
b. What is the length and width of the page table (disregarding the "access rights" bits)?
c. What is the effect on the page table if the physical memory space is reduced by half?

$N = 32$, $P.S = 2KB$, $PA.S = 1MB$ $\Rightarrow$ $M = \frac{2^{20}}{2^{11}} = 2^9$

a) L.A: 16 bit — | p (5) | d (11) | ; $L.A.S = 64KB$

$PA.S = 512KB$

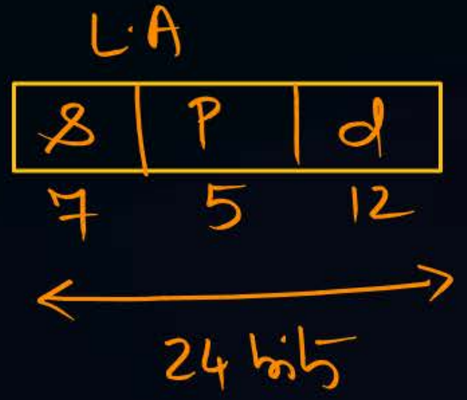
$M = \frac{512K}{2KB} = 256$ $\quad f = 8$ bits

b) $e \sim 2B$

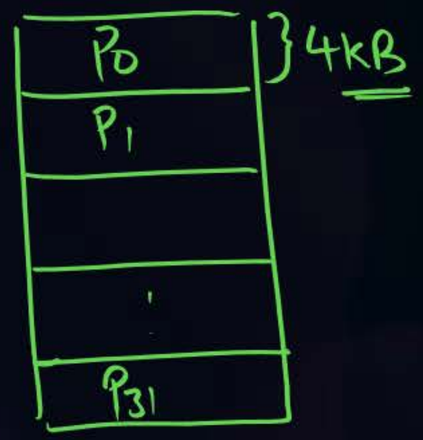
Topic : Demand Paging

The logical address space in a computer system consists of 128 segments. Each segment can have up to 32 pages of 4K words in each. Physical memory consists of 4K blocks of 4K words in each. Formulate the logical and physical address formats.

Seg-Paging



128 Segments



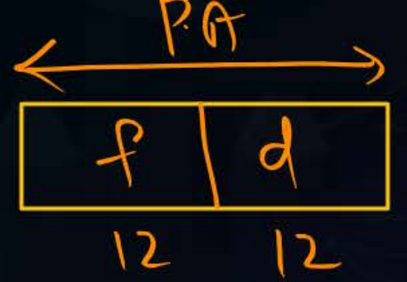
Segment

$Seg\text{-}Size = 32 * 4kB = 128kB$

$L.A.S = 128 * 128kB = 2^{24} B = 16MB$

P.A.S ; M_{frames} : 4K ;

$P.A.S = 4K * 4kB = 16MB$



Topic : Demand Paging

A process references five pages, A, B, C, D, and E, in the following order:

A; B; C; D; A; B; E; A; B; C; D; E

Assume that the replacement algorithm is first-in-first-out and find the number of page transfers during this sequence of references starting with an empty main memory with three page frames. Repeat for four page frames.

4 frames : 10 PF's

<Belady's Anomaly>

FIFO

= (9)

~~A~~	D	D	~~D~~	E	E	E		
B	~~B~~	A	A	~~A~~	C	C		
C	C	~~C~~	B	B	~~B~~	D		

Topic : Demand Paging

A virtual memory system has an address space of 8K words, a memory space of 4K words, and page and block sizes of 1K words ([redacted]). The following page reference changes occur during a given time interval. (Only page changes are listed. If the same page is referenced again, it is not listed twice.)

4 2 0 1 2 6 1 4 0 1 0 2 3 5 7

Determine the four pages that are resident in main memory after each page reference change if the replacement algorithm used is (a) FIFO; (b) LRU.

(i) FIFO : 10 (4 frames)

(ii) LRU : 9

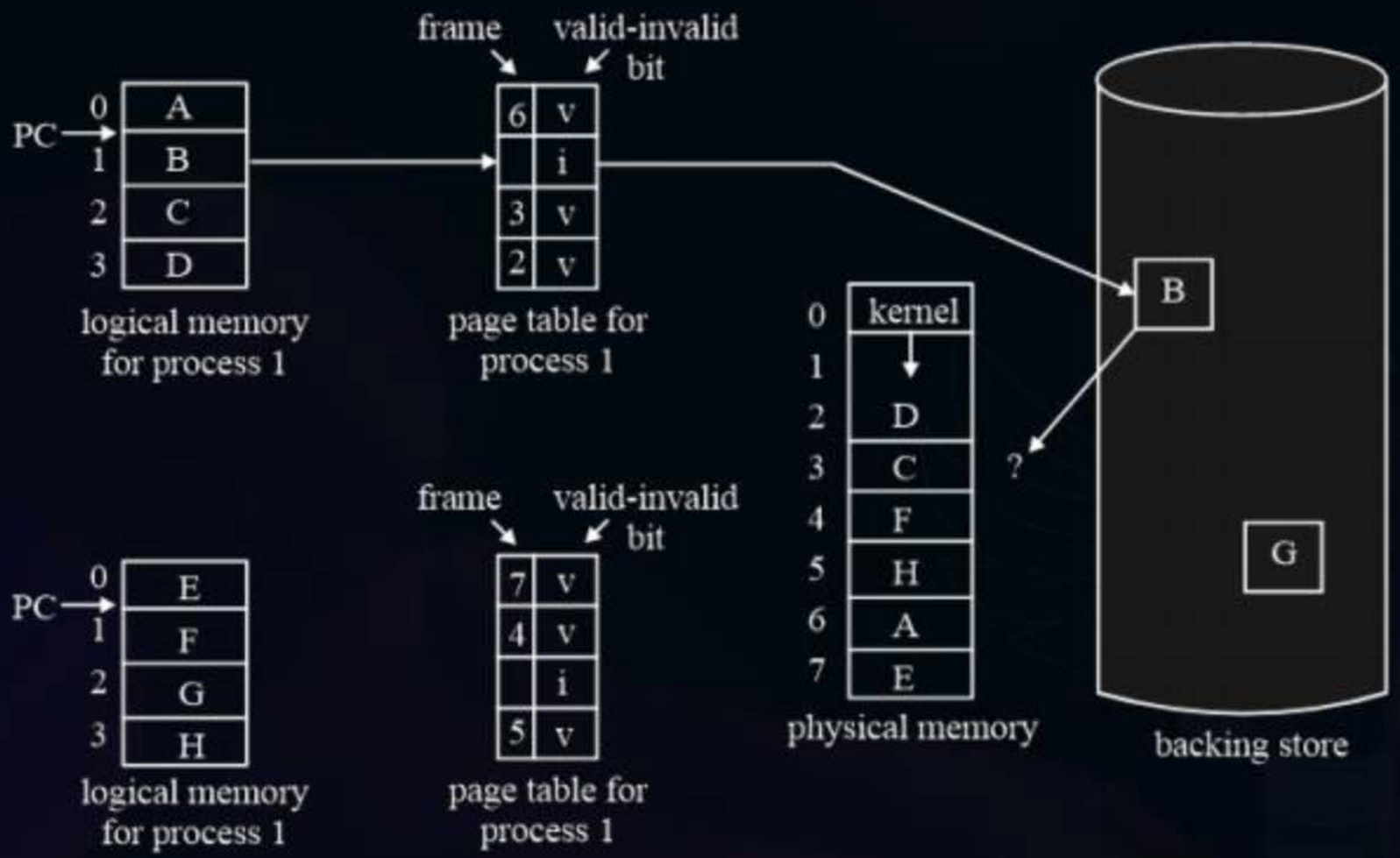
(iii) optimal : 9

Topic : Page Replacement

: There are no Empty frames

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement
- Use modify (dirty) bit to reduce overhead of page transfers – only modified pages are written to disk
- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory

Topic : Need For Page Replacement

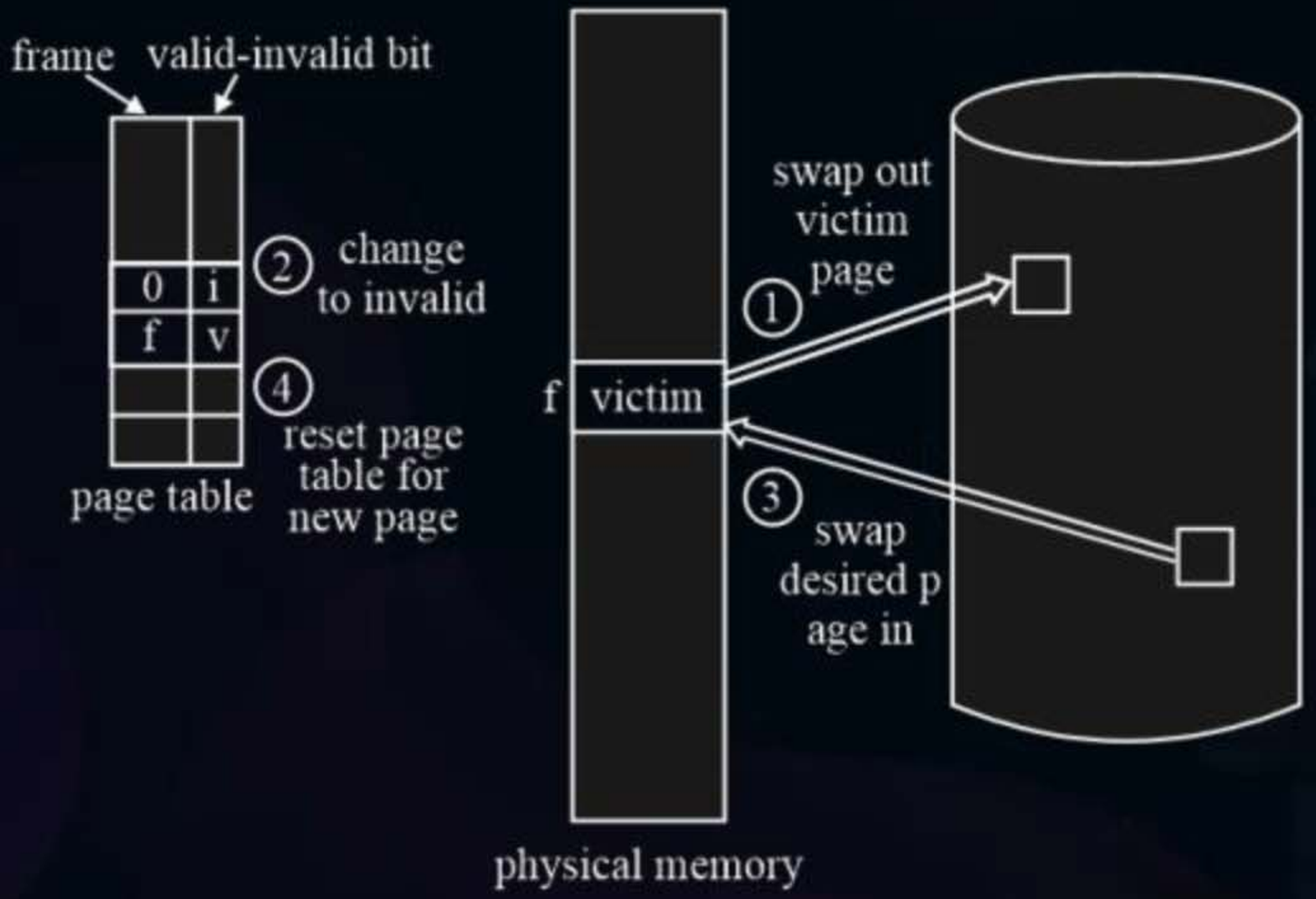


Topic : Basic Page Replacement

1. Find the location of the desired page on disk
2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a victim frame
   - Write victim frame to disk if dirty
1. Bring the desired page into the (newly) free frame; update the page and frame tables
2. Continue the process by restarting the instruction that caused the trap

Note now potentially 2 page transfers for page fault – increasing EAT

Topic : Page Replacement



Topic : Page and Frame Replacement Algorithms

$a_i = \frac{M}{n}$ (i) Equal Allocation

$a_i = \left(\frac{s_i}{S}\right) M$ (ii) Proportionate "

(iii) 50% rule

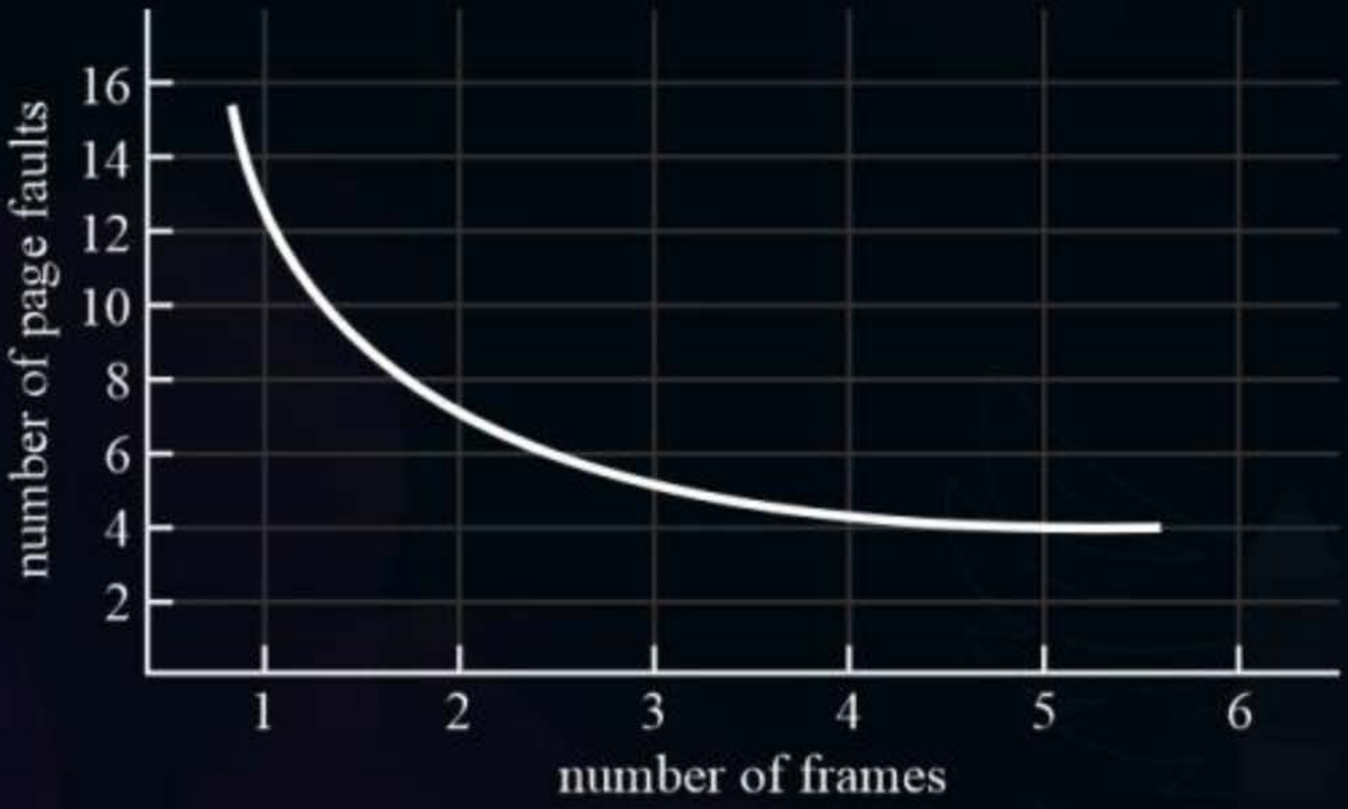
$a_i = \left(s_i / 2\right)$

- Frame-allocation algorithm determines
  - How many frames to give each process
  - Which frames to replace
- Page-replacement algorithm
  - Want lowest page-fault rate on both first access and re-access
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
  - String is just page numbers, not full addresses
  - Repeated access to the same page does not cause a page fault
  - Results depend on number of frames available
- In all our examples, the reference string of referenced page numbers is
  7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

Min no. of frames to be alloc. to the Process

JA

Topic : Graph of Page Faults Versus the Number of Frames



Topic : First-In-First-Out (FIFO) Algorithm

- Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1
- 3 frames (3 pages can be in memory at a time per process)

15 page faults

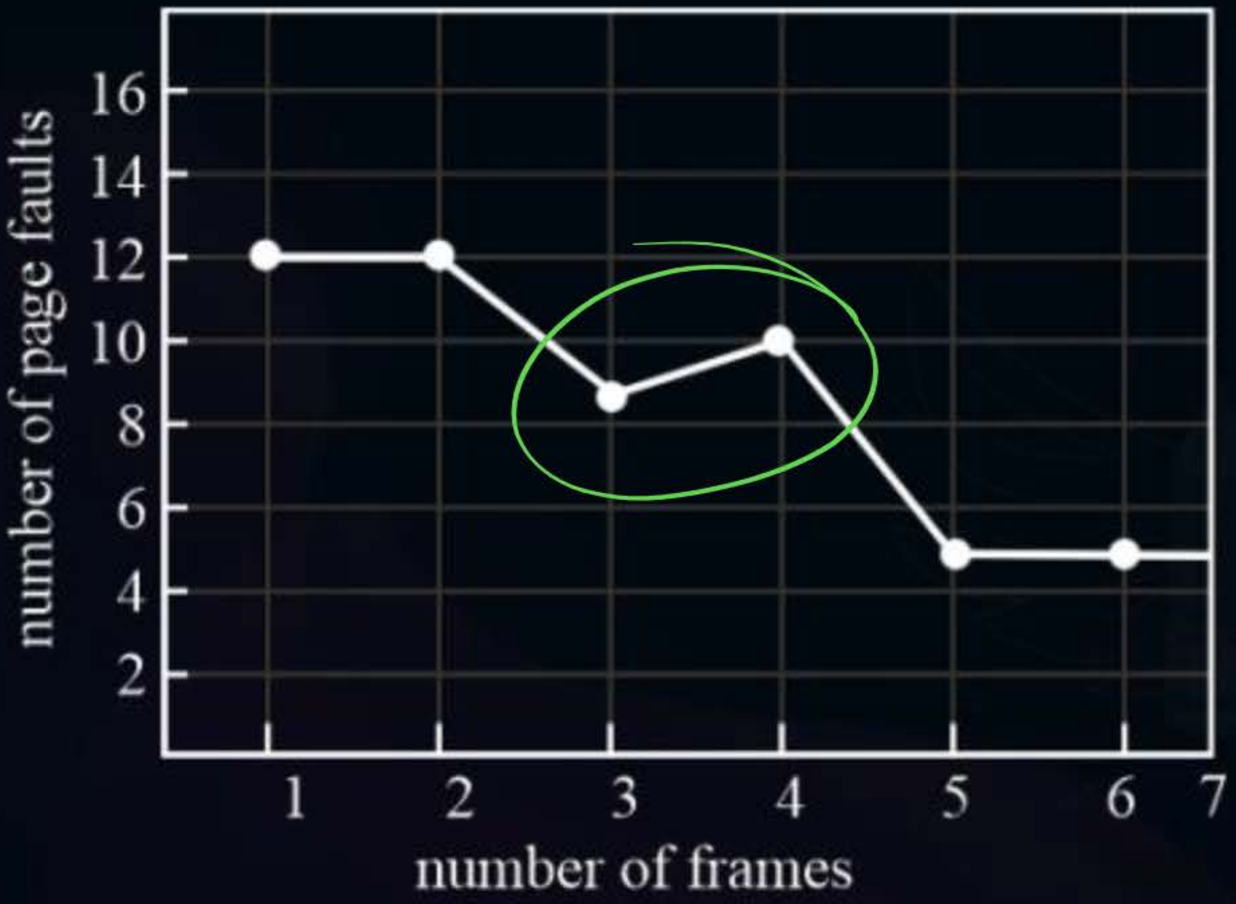
- Can vary by reference string: consider 1,2,3,4,1,2,5,1,2,3,4,5
  - Adding more frames can cause more page faults!
    - Belady's Anomaly
- How to track ages of pages?
  - Just use a FIFO queue

Belady's Anomaly

< FIFO
+FIFO Based
+Randomly >

3F : 9
4F : 10

Topic : FIFO Illustrating Belady's Anomaly



Topic : Optimal Algorithm : Ideal / optimal / Non-Implementable

- Replace page that will not be used for longest period of time
  - 9 is optimal for the example
- How do you know this?
  - Can't read the future
- Used for measuring how well your algorithm performs

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

page frames

Topic : Least Recently Used (LRU) Algorithm

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

page frames

- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used
- But how to implement?

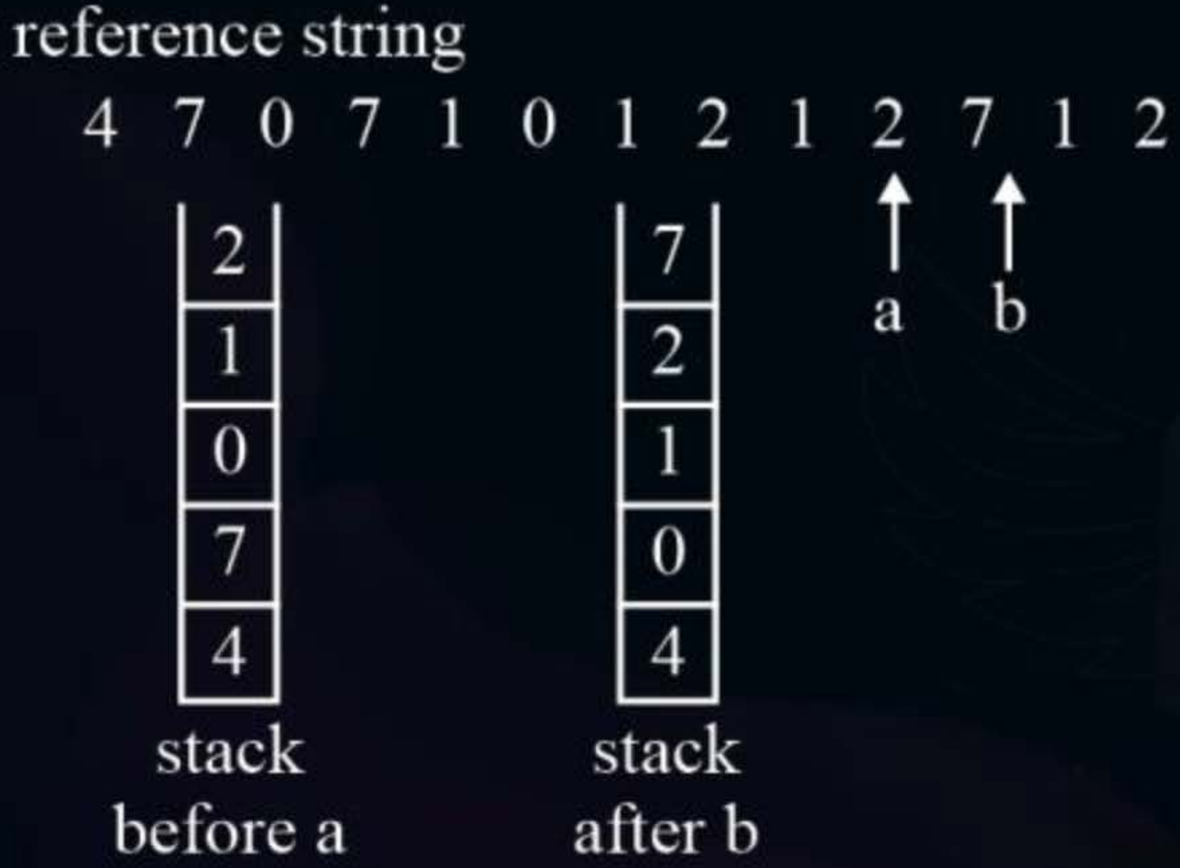
Topic : LRU Algorithm (Cont.)

Least Recently used

- Counter implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
  - When a page needs to be changed, look at the counters to find smallest value
    - Search through table needed
- Stack implementation
  - Keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - But each update more expensive
  - No search for replacement

Topic : LRU Algorithm (Cont.)

- LRU and OPT are cases of stack algorithms that don't have Belady's Anomaly
- Use Of A Stack to Record Most Recent Page References



Topic : LRU Approximation Algorithms

- LRU needs special hardware and still slow
- Reference bit
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Replace any with reference bit = 0 (if one exists)
    - We do not know the order, however

(i) Reference Bit
(R) < 0 / 1

(ii) Additional Reference bits

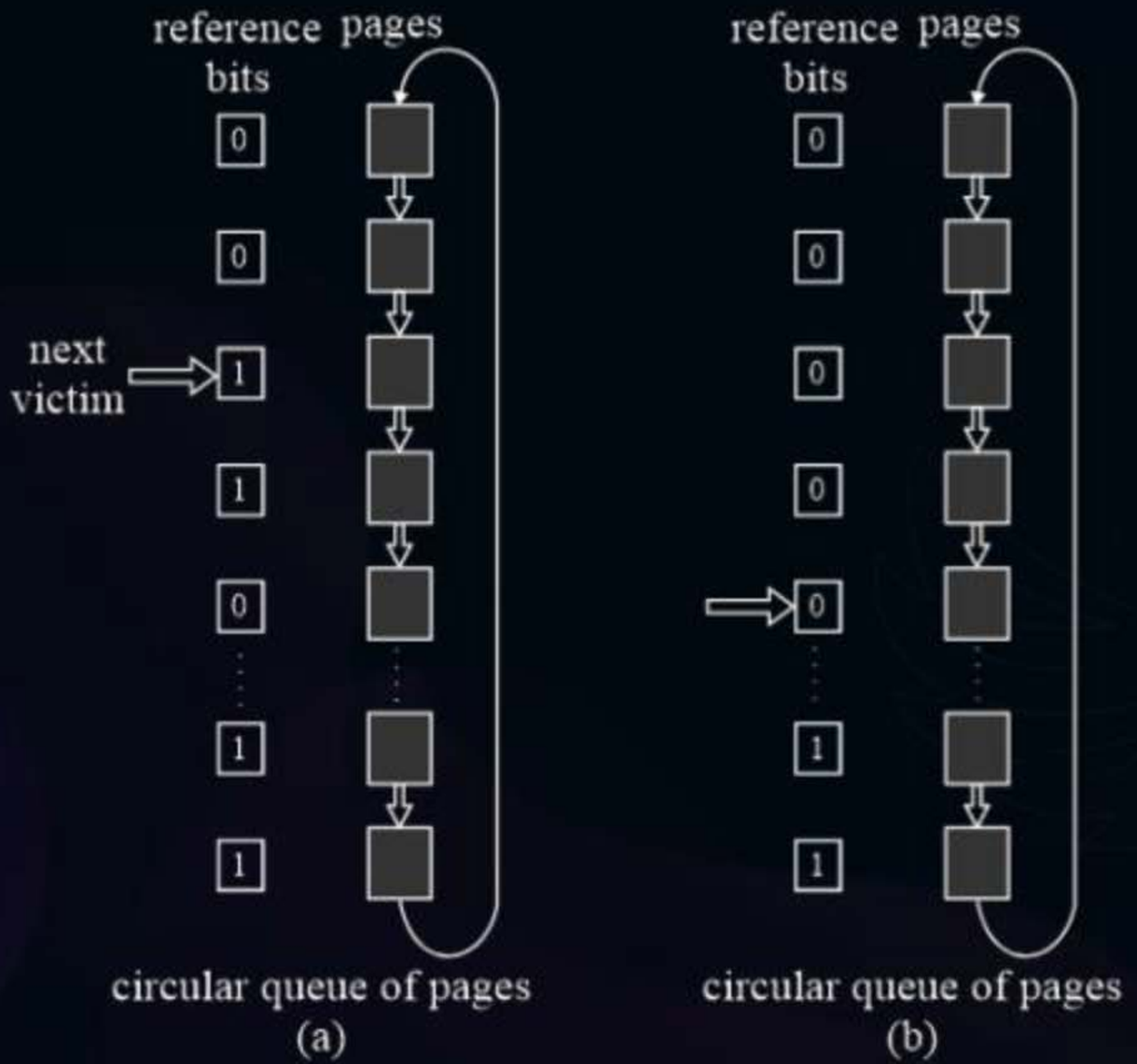
(iii) Second chance
: (1.0.L+R)

(iv) Enhanced Second chance (NRU)
: (R+M)

Topic : LRU Approximation Algorithms (cont.)

- Second-chance algorithm
  - Generally FIFO, plus hardware-provided reference bit
  - Clock replacement
  - If page to be replaced has
    - Reference bit = 0 -> replace it
    - reference bit = 1 then:
      - ✓ set reference bit 0, leave page in memory
      - ✓ replace next page, subject to same rules

Topic : Second-chance Algorithm



Topic : Enhanced Second-Chance Algorithm

- Improve algorithm by using reference bit and modify bit (if available) in concert
- Take ordered pair (reference, modify):
  - (0, 0) neither recently used not modified – best page to replace
  - (0, 1) not recently used but modified – not quite as good, must write out before replacement
  - (1, 0) recently used but clean – probably will be used again soon
  - (1, 1) recently used and modified – probably will be used again soon and need to write out before replacement
- When page replacement called for, use the clock scheme but use the four classes replace page in lowest non-empty class
  - Might need to search circular queue several times

Topic : Counting Algorithms

Count of Reference (COR)

Ex: FIFO

- Keep a counter of the number of references that have been made to each page
  - Not common
- Lease Frequently Used (LFU) Algorithm:
- Replaces page with smallest count
  - Most Frequently Used (MFU) Algorithm:
  - Based on the argument that the page with the smallest count was probably just brought in and has yet to be used

Topic : Page-Buffering Algorithms

- Keep a pool of free frames, always
  - Then frame available when needed, not found at fault time
  - Read page into free frame and select victim to evict and add to free pool
  - When convenient, evict victim
- Possibly, keep list of modified pages
  - When backing store otherwise idle, write pages there and set to non-dirty
- Possibly, keep free frame contents intact and note what is in them
  - If referenced again before reused, no need to load contents again from disk
  - Generally useful to reduce penalty if wrong victim frame selected

Topic : Allocation of Frames

- Each process needs minimum number of frames
- Example: IBM 370 – 6 pages to handle SS MOVE instruction:
  - instruction is 6 bytes, might span 2 pages
  - 2 pages to handle from
  - 2 pages to handle to
- Maximum of course is total frames in the system
- Two major allocation schemes
  - fixed allocation
  - priority allocation
- Many variations

Topic : Fixed Allocation

- Equal allocation – For example, if there are 100 frames (after allocating frames for the OS) and 5 processes, give each process 20 frames
  - Keep some as free frame buffer pool
- Proportional allocation – Allocate according to the size of process
  - Dynamic as degree of multiprogramming, process sizes change

s_i = size of process p_i

$S = \sum s_i$

m = total number of frames

a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$m = 64$

$s_1 = 10$

$s_2 = 127$

$a_1 = \frac{10}{137} \times 62 \approx 4$

$a_2 = \frac{127}{137} \times 62 \approx 57$

Topic : Global vs. Local Allocation

- Global replacement – process selects a replacement frame from the set of all frames; one process can take a frame from another
  - But then process execution time can vary greatly
  - But greater throughput so more common
- Local replacement – each process selects from only its own set of allocated frames
  - More consistent per-process performance
  - But possibly underutilized memory

Topic : Reclaiming Pages

- A strategy to implement global page-replacement policy
- All memory requests are satisfied from the free-frame list, rather than waiting for the list to drop to zero before we begin selecting pages for replacement,
- Page replacement is triggered when the list falls below a certain threshold.
- This strategy attempts to ensure there is always sufficient free memory to satisfy new requests.

Thrashing :

↳ excessive / High Paging activity (Page fault Rate)

↓

cpu utiliz. drops

Reasons :

(i) Lack of frames (Memory)

(ii) High degree of M.Pr

Control Thrashing :

(i) Controlling degree of M.Pr

(ii) Suspension

Secondary :

1) Page Replacement Technique

2) Page Size

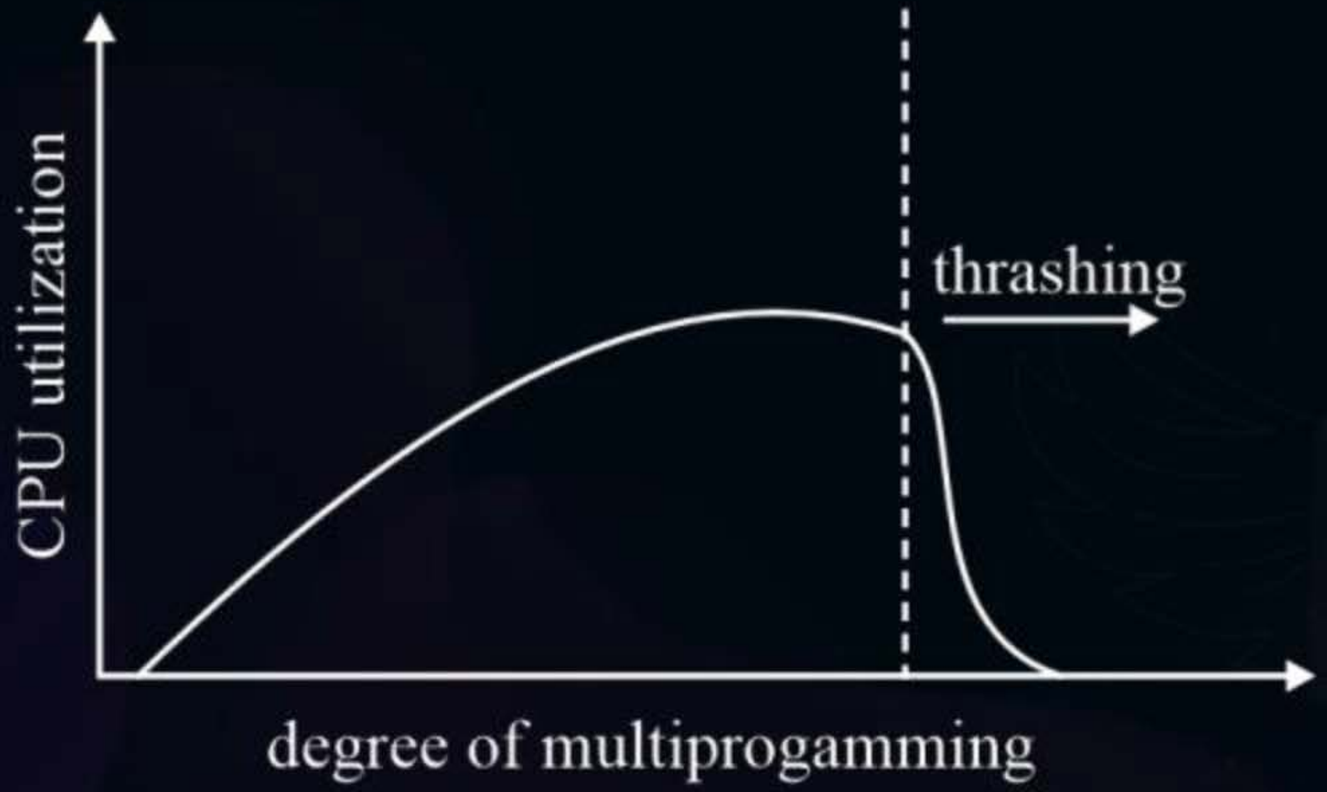
3) Prog. Techniques & D.S ⭐

Topic : Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high
  - Page fault to get page
  - Replace existing frame
  - But quickly need replaced frame back
  - This leads to:
    - Low CPU utilization
    - Operating system thinking that it needs to increase the degree of multiprogramming
    - Another process added to the system

Topic : Thrashing (Cont.)

- Thrashing. A process is busy swapping pages in and out



Topic : Demand Paging and Thrashing

- Why does demand paging work?
- **Locality model**
  - Process migrates from one locality to another
  - Localities may overlap
- Why does thrashing occur?

  $\sum$ size of locality > total memory size
- Limit effects by using local or priority page replacement

Topic : Working-Set Model

- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  Example: 10,000 instructions
- WSS_i (working set of Process P_i) = total number of pages referenced in the most recent Δ (varies in time)
  - if Δ too small will not encompass entire locality
  - if Δ too large will encompass several localities
  - if $\Delta = \infty \Rightarrow$ will encompass entire program
- $D = \Sigma\ WSS_i \equiv$ total demand frames
  - Approximation of locality

Topic : Working-Set Model (Cont.)

- if D > m ⇒ Thrashing
- Policy if D > m, then suspend or swap out one of the processes

page reference table

... 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 ...

$WS(t_1) = \{1, 2, 5, 6, 7\}$

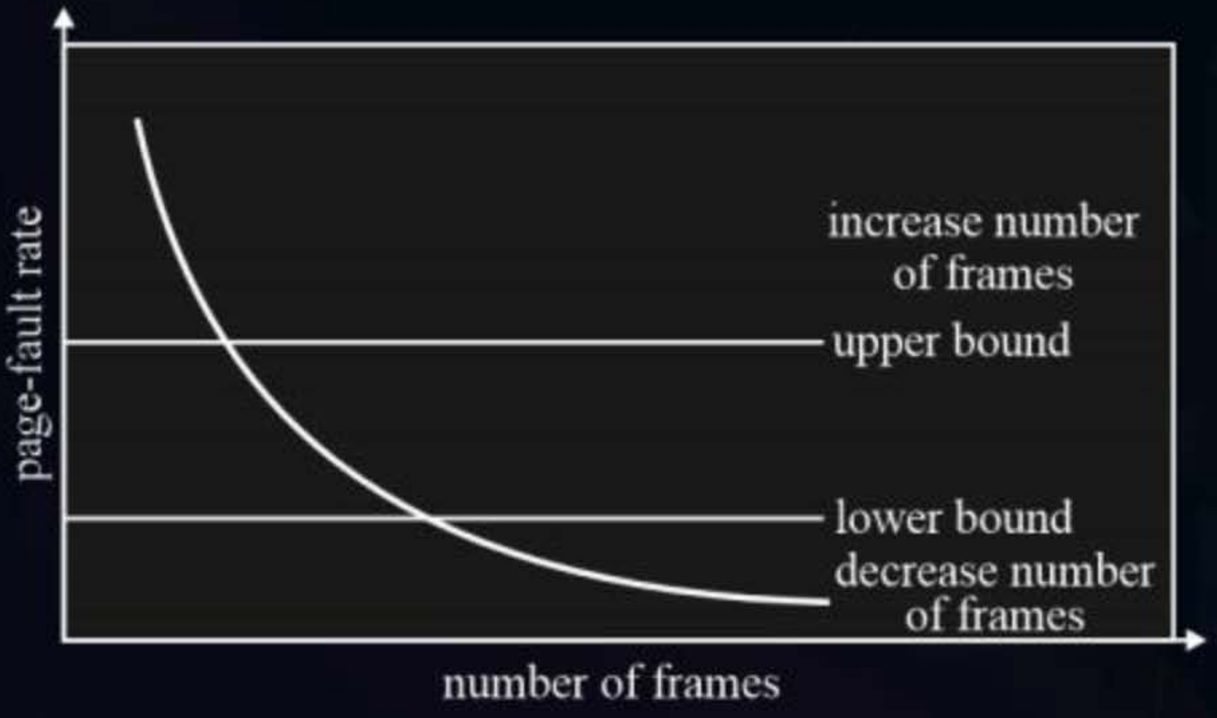
$WS(t_2) = \{3, 4\}$

Topic : Keeping Track of the Working Set

- Approximate with interval timer + a reference bit
- Example: $\Delta = 10{,}000$
  - Timer interrupts after every 5000 time units
  - Keep in memory 2 bits for each page
  - Whenever a timer interrupts copy and sets the values of all reference bits to 0
  - If one of the bits in memory = 1 $\Rightarrow$ page in working set
- Why is this not completely accurate?
- Improvement = 10 bits and interrupt every 1000 time units

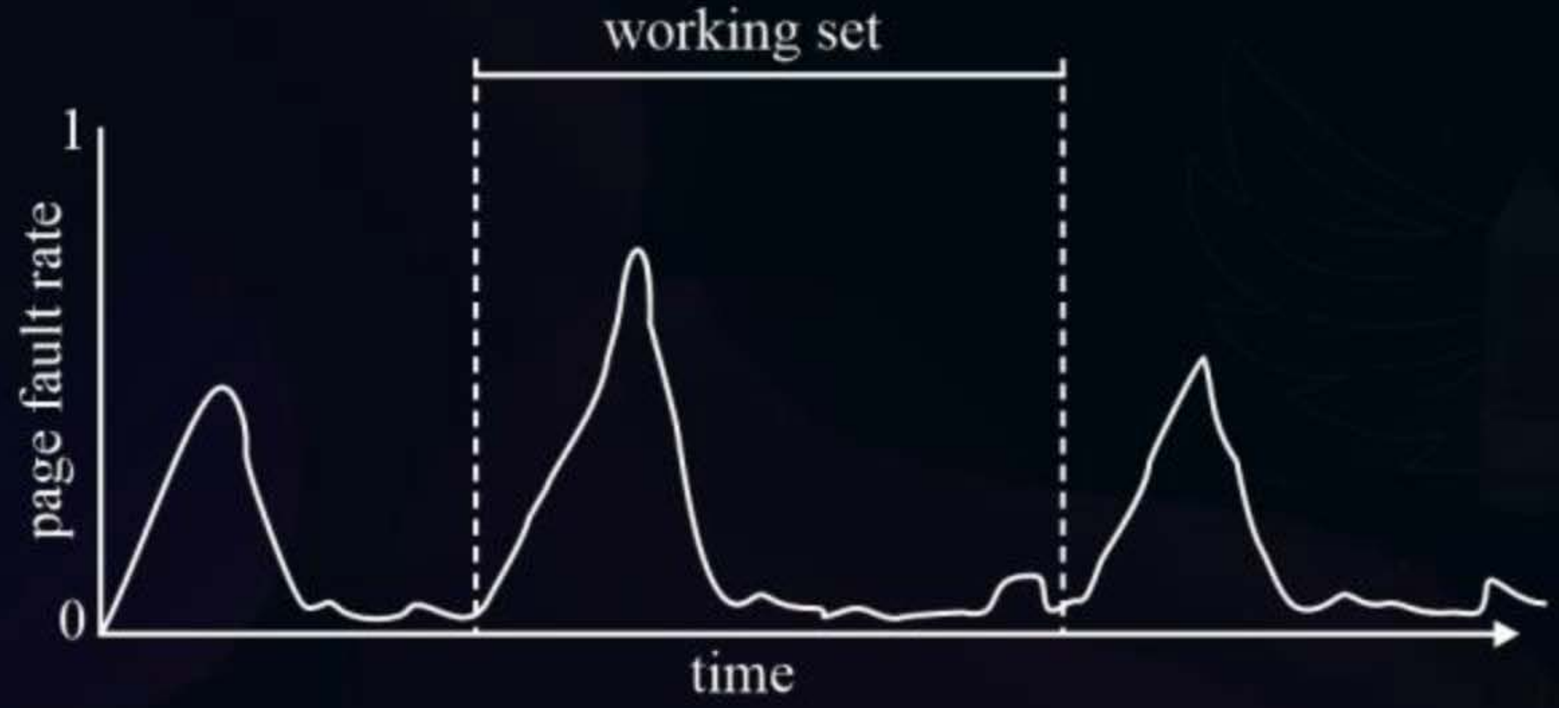
Topic : Page-Fault Frequency

- More direct approach than WSS
- Establish “acceptable” page-fault frequency (PFF) rate and use local replacement policy
  - If actual rate too low, process loses frame
  - If actual rate too high, process gains frame



Topic : Working Sets and Page Fault Rates

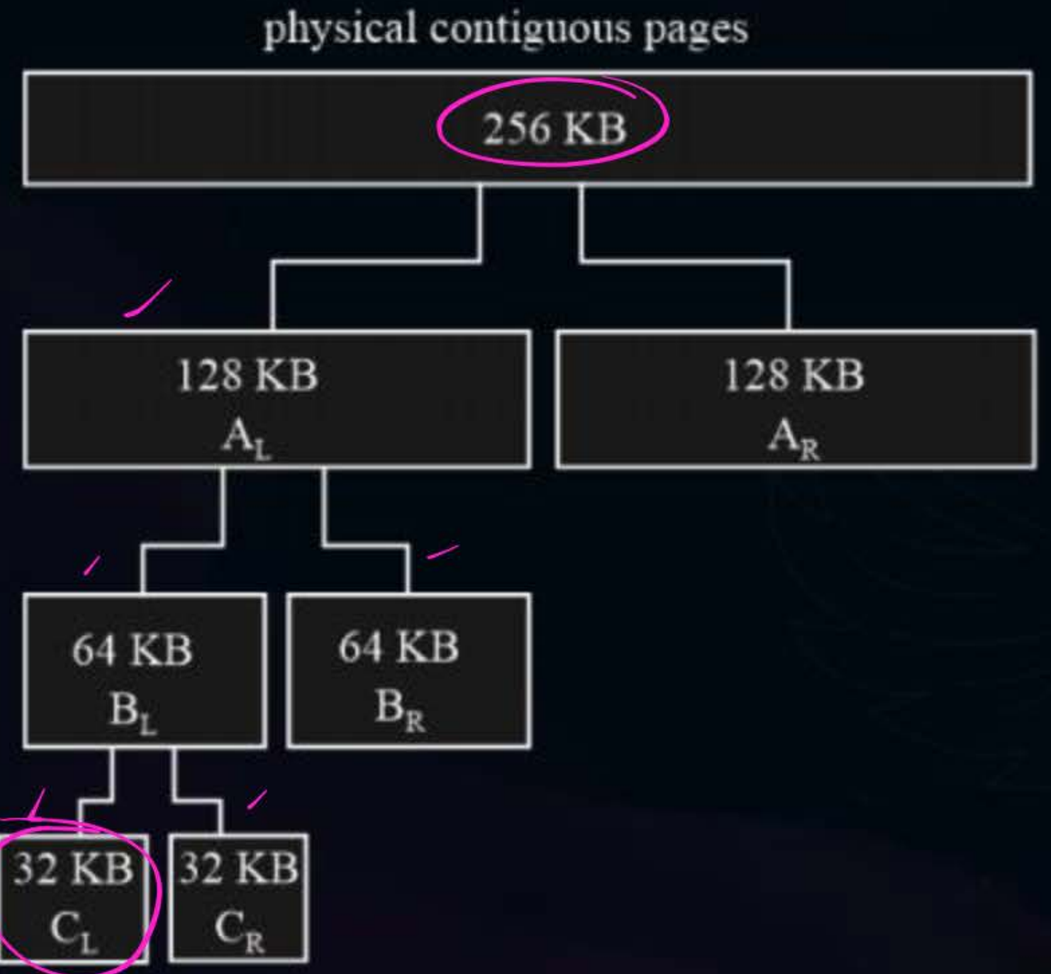
- Direct relationship between working set of a process and its page-fault rate
- Working set changes over time
- Peaks and valleys over time



Topic : Buddy System

- Allocates memory from fixed-size segment consisting of physically-contiguous pages
- Memory allocated using power-of-2 allocator
  - Satisfies requests in units sized as power of 2
  - Request rounded up to next highest power of 2
  - When smaller allocation needed than is available, current chunk split into two buddies of next-lower power of 2
    - Continue until appropriate sized chunk available
- For example, assume 256KB chunk available, kernel requests 21KB
  - Split into AL and AR of 128KB each
    - One further divided into BL and BR of 64KB
      - ✓ One further into CL and CR of 32KB each – one used to satisfy request
- Advantage – quickly coalesce unused chunks into larger chunk
- Disadvantage - fragmentation

Topic : Buddy System Allocator



P_{size} : 25 KB

Topic : Other Considerations

- Prepaging
- Page size
- TLB reach
- Inverted page table
- Program structure
- I/O interlock and page locking

Topic : Prepaging

- To reduce the large number of page faults that occurs at process startup
- Prepage all or some of the pages a process will need, before they are referenced
- But if prepaged pages are unused, I/O and memory was wasted
- Assume s pages are prepaged and α of the pages is used
  - Is cost of s * α save pages faults > or < than the cost of prepaging s * (1- α) unnecessary pages?
  - α near zero prepaging loses

Topic : Page Size

- Sometimes OS designers have a choice
  - Especially if running on custom-built CPU
- Page size selection must take into consideration:
  - Fragmentation
  - Page table size
  - Resolution
  - I/O overhead
  - Number of page faults
  - Locality
  - TLB size and effectiveness
- Always power of 2, usually in the range 212 (4,096 bytes) to 222 (4,194,304 bytes)
- On average, growing over time

Topic : TLB Reach

- TLB Reach - The amount of memory accessible from the TLB
- TLB Reach = (TLB Size) X (Page Size)
- Ideally, the working set of each process is stored in the TLB
  - Otherwise there is a high degree of page faults
- Increase the Page Size
  - This may lead to an increase in fragmentation as not all applications require a large page size
- Provide Multiple Page Sizes
  - This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation

Topic : Program Structure

- Program structure
  - int[128,128] data;
  - Each row is stored in one page
  - Program 1

```
for (j = 0; j <128; j++)
        for (i = 0; i < 128; i++)
                data[i,j] = 0;
```

128 x 128 = 16,384 page faults

  - Program 2

```
for (i = 0; i < 128; i++)
        for (j = 0; j < 128; j++)
                data[i,j] = 0;
```

128 page faults

Topic : I/O interlock

- I/O Interlock – Pages must sometimes be locked into memory
- Consider I/O - Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm
- Pinning of pages to lock into memory

2 mins Summary

Topic One — I/O interlock

Topic Two — Program Structure

Topic Three — TLB Reach

Topic Four — Demand Paging

Topic Five

THANK - YOU