CS 370: OPERATING SYSTEMS
[VIRTUAL MEMORY]

Shrideep Pallickara
Computer Science
Colorado State University

COMPUTER SCIENCE DEPARTMENT   COLORADO STATE UNIVERSITY

1

Frequently asked questions from the previous class survey

- Belady's anomaly
  - Does it still occur? Do current OS rely on stack algorithms?
- Are pages removed from the backing store?
  - Why block transfers?
- How does demand paging "guess" which pages to bring in? Can you have mix of demand and non-demand paging in the same system?
- Page faults:
  - Is the user made aware of them? Or is handled transparently?
  - Can the OS pivot to some other process when this happens?
- Is there a way to maintain a set of victim frames?
- Page table entries?

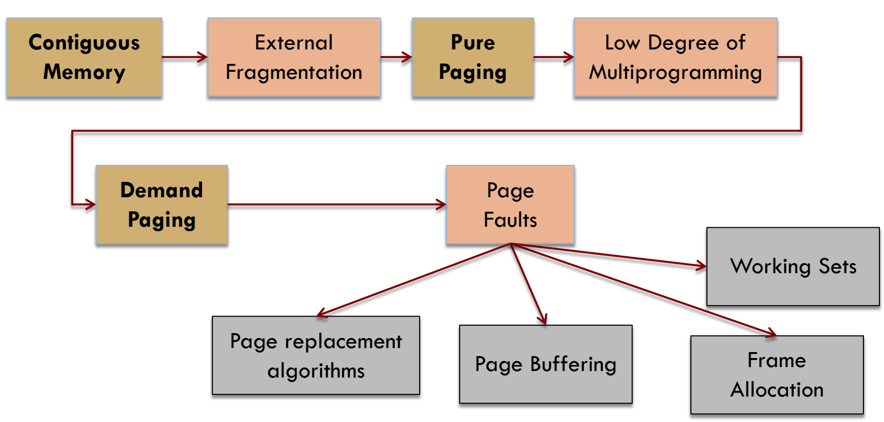
COLORADO STATE UNIVERSITY   Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT   VIRTUAL MEMORY   L24.2

2

Topics covered in this lecture

- Page replacement algorithms
- Page Buffering
- Frame Allocations
- Working Sets
- TLB Reach

How we got here …



THE OPTIMAL PAGE REPLACEMENT ALGORITHM

COMPUTER SCIENCE DEPARTMENT COLORADO STATE UNIVERSITY

5

The optimal page replacement algorithm

- The best possible algorithm
- Easy to describe but **impossible to implement**
- **Crux:**
  Put off unpleasant stuff for as long as possible

COLORADO STATE UNIVERSITY Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT VIRTUAL MEMORY L24.6

6

The optimal page replacement algorithm description

- When a page fault occurs some set of pages are in memory
- One of these pages will be referenced next
  - Other pages may be not be referenced until 10, 100 or 1000 instructions later
- **Label** each page with the number of instructions to be executed *before* it will be referenced
  - When there is a page-fault, the page with the highest label should be removed

Problem with the optimal page replacement algorithm

- It is **unrealizable**
- During a page fault, the OS has no way of knowing ***when*** each of the pages will be referenced next

So why are we looking at it?

- Run a program
  - Track all page references
- Implement optimal page replacement on the second run
  - Based on reference information from the first run
- **Compare** performance of **realizable** algorithms with the best possible one

LRU PAGE REPLACEMENTS

The Least Recently Used (LRU) page replacement algorithm

- Approximation of the optimal algorithm
- Observation
  - Pages used heavily in the last few instructions
    - Probably will be used heavily in the next few
  - Pages that have not been used
    - Will probably remain unused for a long time
- When a page fault occurs?
  - Throw out page that has been **unused the longest**

LRU example: 3 memory frames

Reference String	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
Recent	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
		7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0
Least Used			7	0	1	2	2	3	0	4	2	2	0	3	3	1	2	0	1	7

Implementing LRU

- Logical clock
- Stacks

Using Logical clocks to implement LRU

- Each page table entry has a **time-of-use** field
  - Entry updated when page is referenced
    - Contents of clock register are copied
- Replace the page with the smallest value
  - Time increases monotonically
    - **Overflows** must be accounted for
- Requires search of page table to find LRU page

Stack based approach

- Keep stack of page numbers
- When page is referenced
  - Move to the top of the stack
- Implemented as a doubly linked list
- No search done for replacement
  - Bottom of the stack is the LRU page

Problems with clock/stack based approaches to LRU replacements

- Inconceivable without hardware support
  - Few systems provide requisite support for true LRU implementations
- Updates of clock fields or stack needed at **every** memory reference
- If we use interrupts and do software updates of data structures things would be very slow
  - Would slow down every memory reference
    - At least 10 times slower

LRU APPROXIMATION PAGE REPLACEMENTS

LRU Approximation: Reference bit

- **Reference bit** associated with page table entries
- Reference bit is set by hardware when page is referenced
  - Read/write access of the page
- Determine which page has been used and which has not
  - No way of knowing the *order of references* though

LRU Approximation: Additional reference bits

- Maintain 8-bit byte for each page in memory
- OS **shifts** the reference bit for page into the highest order bit of the 8-bit byte
  - Operation performed at *regular intervals*
  - The reference bit is then *cleared*

LRU approximation: Reference bits

Shift Register	Reference bit for the page	Shift Register after the OS timer interrupt
00000000	1	10000000
10010001	1	11001000
01100011	0	00110001

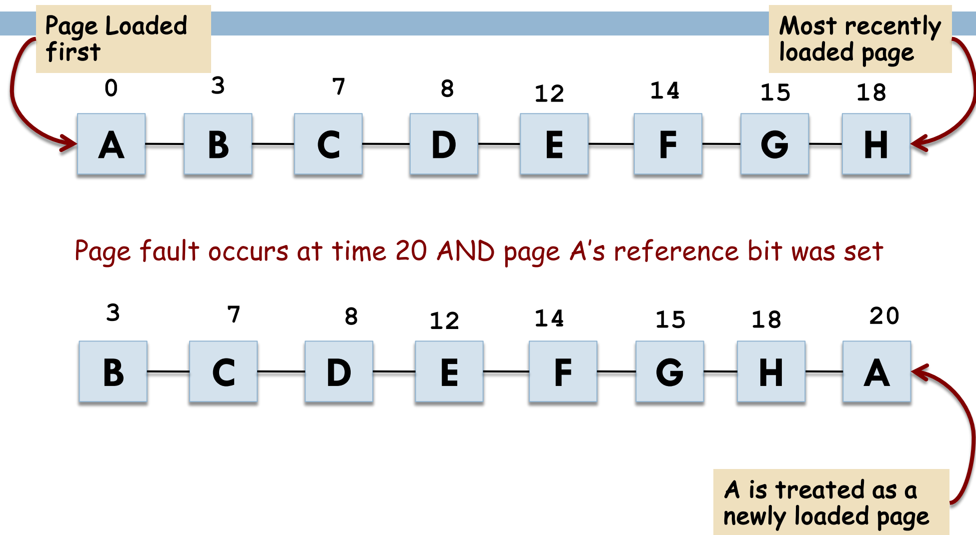
LRU Approximation: Interpreting the reference bits

- Interpret 8-bit bytes as **unsigned integers**
- Page with the lowest number is the LRU page
- **00000000** : Not used in last 8 periods
- **01100101** : Used 4 times in the last 8 periods
- **11000100** used ***more recently*** than **01110111**

The Second Chance Algorithm

- Simple modification of FIFO
- Avoids throwing out a heavily used page
- Inspect the reference bit of a page
  - If it is **0**: Page is old and unused
    - **Evict**
  - If it is **1**: Page is given a second chance
    - Move page to the end of the list

The Operation of second chance



Second chance

- Reasonable algorithm, but unnecessarily **inefficient**
  - Constantly moving pages around on its list
- Better to keep pages in a circular list
  - In the form of a clock …

Clock Page Replacement

- Keep all frames on a circular list in the form of a clock
  - Hand points to the oldest page
- When a page fault occurs, page being pointed to by the hand is inspected
  - If its R bit is 0: the page is evicted
    - New page is inserted into the clock in its place
    - Hand is advanced one position
  - If its R bit is 1
    - It is cleared and the hand is advanced one position until a page is found with R =0

25

Counting based page replacements Most Frequently Used (MFU)

- **Argument**:
  Page with the smallest count was probably just brought in

26

Summary of Page Replacement Algorithms

Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
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
NFU (Not Frequently Used)	Fairly crude approximate to LRU
Aging [Multiple reference bits]	Efficient algorithm that approximates LRU well

PAGE BUFFERING ALGORITHMS

Page Buffering

1. Maintain a buffer of free frames
2. When a page-fault occurs
   - Victim frame chosen as before
   - Desired page read into free-frame **from buffer**
     - **Before** victim frame is written out
   - Process that page-faulted can restart much faster

Page Buffering: Being proactive

- Maintain a list of **modified** pages
- When the paging device is **idle**?
  - Write modified pages to disk
- Implications
  - If a page is selected for replacement *increase likelihood* of that page being clean

Page Buffering: Reuse what you can

- Keep pool of free frames as before
  - BUT **remember** which pages they held
- Frame contents are not modified when page is written to disk
- If page needs to come back in?
  - **Reuse** the same frame, if it was not used to hold some other page

Buffering and applications

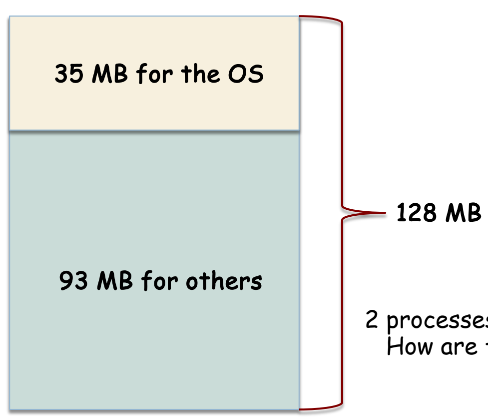
- Applications often understand their memory/disk usage <u>better</u> than the OS
  - Provide their own buffering schemes
- If both the OS and the application were to buffer
  - Twice the I/O is being utilized for a given I/O

ALLOCATION OF FRAMES

33

Frame allocation: How do you divvy up free memory among processes?

Frame size = 1 MB; Total Size = 128 MB



2 processes at T_0
How are frames allocated?

With demand paging all 93 frames would be in the free frame pool

34

Constraints on frame allocation

- **Max:** Total number of frames in the system
  - Available physical memory
- **Min:** We need to allocate at least a minimum number of frames
  - Defined by the architecture of the underlying system

Minimum number of frames

- As you decrease the number of frames for a process
  - Page fault increases
  - Execution time increases too
- Defined by the **architecture**
  - In some cases, instructions and operands (*indirect references*) straddle page boundaries
    - With 2 operands at least 6 frames needed

FRAME ALLOCATION POLICIES

COMPUTER SCIENCE DEPARTMENT   COLORADO STATE UNIVERSITY

37

Global vs Local Allocation

- Global replacement
  - One process can **take** a memory frame from another process
- Local replacement
  - Process can only choose from the set of frames that was allocated to it

COLORADO STATE UNIVERSITY   Professor: SHRIDEEP PALLICKARA COMPUTER SCIENCE DEPARTMENT   VIRTUAL MEMORY   L24.38

38

Local vs Global replacement: Based on how often a page is referenced

Processes A, B and C

Pages	Usage Count
A1	10
A2	7
A3	5
A4	3
B1	9
B2	4
B3	2
B4	6
C1	3
C2	5
C3	6

Local Replacement

Pages
A1
A2
A3
A5
B1
B2
B3
B4
C1
C2
C3

Global Replacement

Pages
A1
A2
A3
A4
B1
B2
A5
B4
C1
C2
C3

Process **A** has page faulted and needs to bring in a page

Global vs Local Replacement

	Local	Global
Number of frames allocated to process	Fixed	Varies dynamically
Can process control its own fault rate?	YES	NO
Can it use free frames that are available?	NO	YES
Increases system throughput?	NO	YES

WORKING SETS & THRASHING

Locality of References

- During any phase of execution, a process references a relatively small **fraction** of its pages
- Set of pages that a process is currently using
  - **Working set**
- Working set **evolves** during process execution

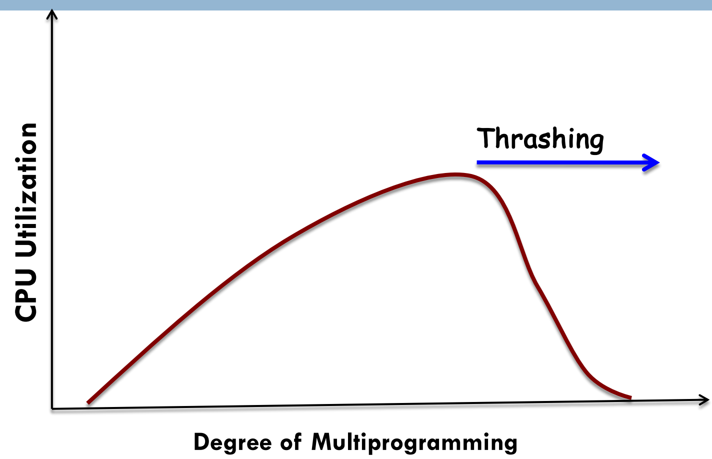
Implications of the working set

- If the entire working set is in memory
  - Process will execute without causing many faults
    - Until it moves to *another phase* of execution
- If the available memory is too small to hold the working set?
  1. Process will cause many faults
  2. Run very slowly

A program causing page faults every few instructions is said to be thrashing

- System throughput **plunges**
  - Processes spend all their time paging
- Increasing the degree of multiprogramming can cause this
  - New process may **steal** frames from another process {*Global Replacement*}
    - Overall page-faults in the system increases

Characterizing the affect of multiprogramming on thrashing



45

Mitigating the effects of thrashing

- Using a local page replacement algorithm
  - One process thrashing does not cause **cascading thrashing** among other processes
  - BUT if a process is thrashing?
    - Average service time for a page fault increases
- Best approach
  1. Track a process' working set
  2. Make sure the working set is in memory **before** you let it run

46

WORKING SETS & THRASHING

Working set is an approximation of the program's locality

- Most important property of the working set is its **size**

```
Page reference table
...2 6 1 5 7 7 7 7 5 1 6 2 3 4 4 4 3 4 3 4 4 4 1 2 3 4 8
   <------ Δ ------>       <------ Δ ------>
   WS = {1,2,5,6,7}         WS = {3,4}
```

- WSS_i = Working set size for process p_i
- If total demand exceeds available frames
  - Thrashing will occur

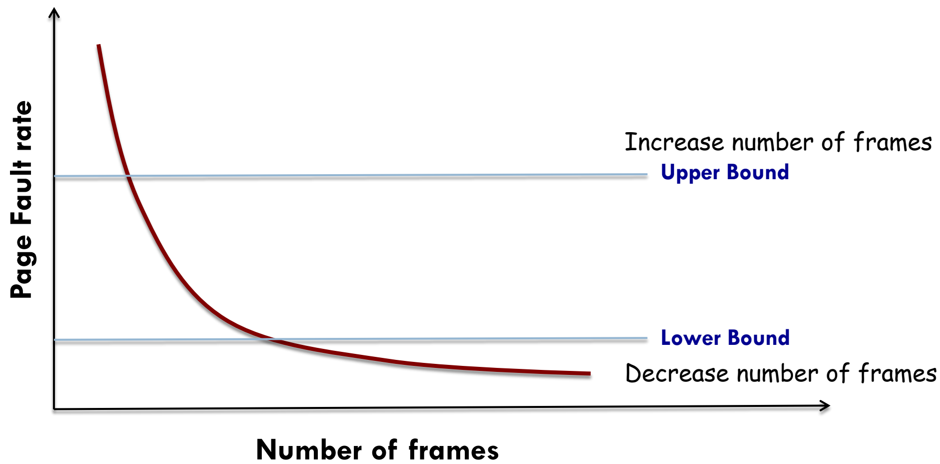
Working sets and page fault rates

- The peak in page-fault rate happens when a *new locality* is being demand-paged
- Once working set is in memory
  - Page fault rate falls
- When process moves towards a new working set window?
  - Fault rate rises again

The page fault frequency approach to reducing thrashing

- When the page fault rate is high
  - Process needs **more** frames
- When the page fault rate is too low
  - Process may have **too many** frames

Using page fault frequencies to control thrashing: Establish bounds



51

OTHER CONSIDERATIONS

52

Prepaging: Loading pages BEFORE letting a process run

- Bring into memory -- at one time -- all the pages that will be needed
  - Prepage frames for small files
- With the working set model
  - Ensure that the **entire working set** *is in memory* before the process is resumed

TLB Reach is the amount of memory accessible from the TLB

- TLB-Reach = Number of TLB entries x Page Size
- Approaches to increasing TLB reach
  - Double the entries
    - Expensive
  - Increase page size
    - Increases (internal) fragmentation
  - Support multiple page sizes
    - OS not hardware manages the TLB
    - Increase reach and hit ratio

  } Current trend

Select data structures and program structures efficiently

- Increase locality
  - Reduce page fault rates
- Loops
  - If data is stored in *row-major* format, but program reads it as *column-major* format
- Loader should avoiding placing routines across page boundaries

The contents of this slide-set are based on the following references

- *Avi Silberschatz, Peter Galvin, Greg Gagne. Operating Systems Concepts, 9th edition. John Wiley & Sons, Inc. ISBN-13: 978-1118063330.* [Chapter 9]
- *Andrew S Tanenbaum and Herbert Bos. Modern Operating Systems. 4th Edition, 2014. Prentice Hall. ISBN: 013359162X/ 978-0133591620. [Chapter 3]*