OPERATING SYSTEMS

Memory Management - 12

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

OPERATING SYSTEMS

Course Syllabus - Unit 3

Unit-3:Unit 3: Memory Management: Main Memory

Hardware and control structures, OS support, Address translation, Swapping, Memory Allocation (Partitioning, relocation), Fragmentation, Segmentation, Paging, TLBs context switches Virtual Memory - Demand Paging, Copy-on-Write, Page replacement policy - LRU (in comparison with FIFO & Optimal), Thrashing, design alternatives - inverted page tables, bigger pages. Case Study: Linux/Windows Memory

OPERATING SYSTEMS

Course Outline

25	Main Memory: Hardware and control structures, OS support, Address translation	8.1	64.2
26	Dynamic linking, Swapping	8.2	
27	Memory Allocation (Partitioning, relocation), Fragmentation	8.3	
28	Segmentation	8.4	
29	Paging: OS Support, TLBs, Address Translation	8.5	
30	Structure of the Page Table	8.6	
31	Design Alternatives - Inverted Page Tables, Bigger Pages	8.7-8.8	
32	Virtual Memory: Demand Paging, Copy-OnWrite	9.1-9.3	
33	Page replacement policy - LRU	9.4	
34	FIFO & Optimal	9.5	
35	Thrashing	9.6	
36	Case Study: Linux/ Windows Memory Management	9.10	

Topic Outline

- Virtual Memory - Page replacement
- What happens if there is no free Frame ?
- Basic Page Replacement
- Page and Frame Replacement Algorithms
- Graph of Page Faults versus the number of Frames

Topic Outline

- First-In-First-Out (FIFO) Algorithm
- FIFO illustrating Belady's Anomaly
- Optimal Page Replacement Algorithm
- Least Recently Used (LRU) Algorithm
- Use of a Stack to Record Most Recent Page References

Topic Outline

- LRU Algorithm Implementation
- LRU Approximation Algorithm
- Second-Chance (clock) Page-Replacement Algorithm
- Enhanced Second-Chance Algorithm
- Counting Algorithms
- Page-Buffering Algorithms

Topic Outline

- Applications and Page Replacement
- Allocation of Frames
- Fixed Allocation
- Priority Allocation
- Global vs. Local Allocation
- Non-Uniform Memory Access

Topic Outline

- Virtual Memory - Thrashing
- Demand Paging and Thrashing
- Working-Set Model
- Keeping Track of the Working Set
- Page-Fault Frequency

Virtual Memory - 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

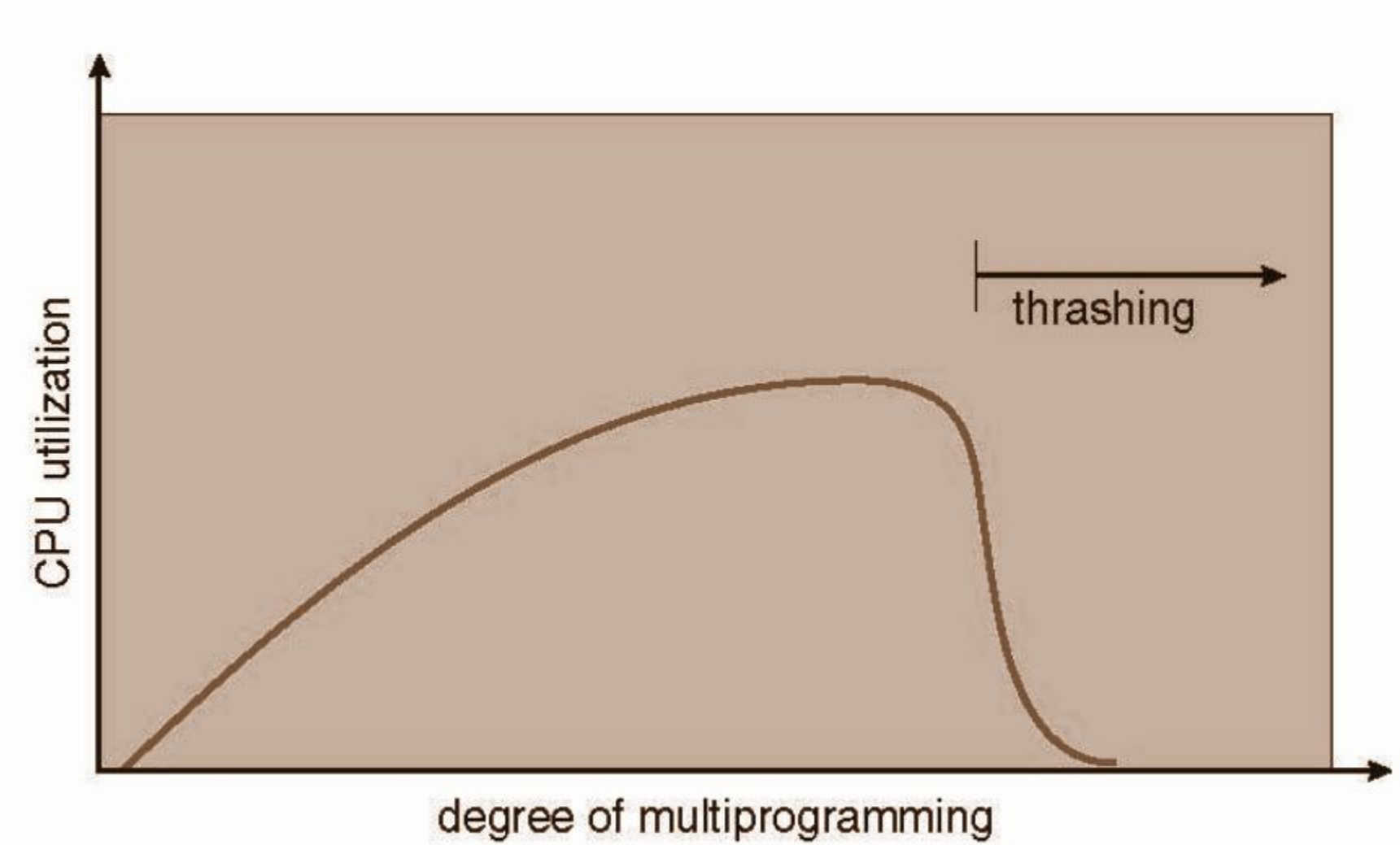
Virtual Memory - Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high
  - This leads to:
    - Low CPU utilization
    - Operating system thinking that it needs to increase the degree of multiprogramming
    - Another process added to the system

Virtual Memory - Thrashing

Thrashing is a process of keeping system busy with swapping pages **in** and **out**

Virtual Memory - Thrashing



Demand Paging and Thrashing

- Why does demand paging work ?
  - Locality model
    - Process migrates from one locality to another

Demand Paging and Thrashing

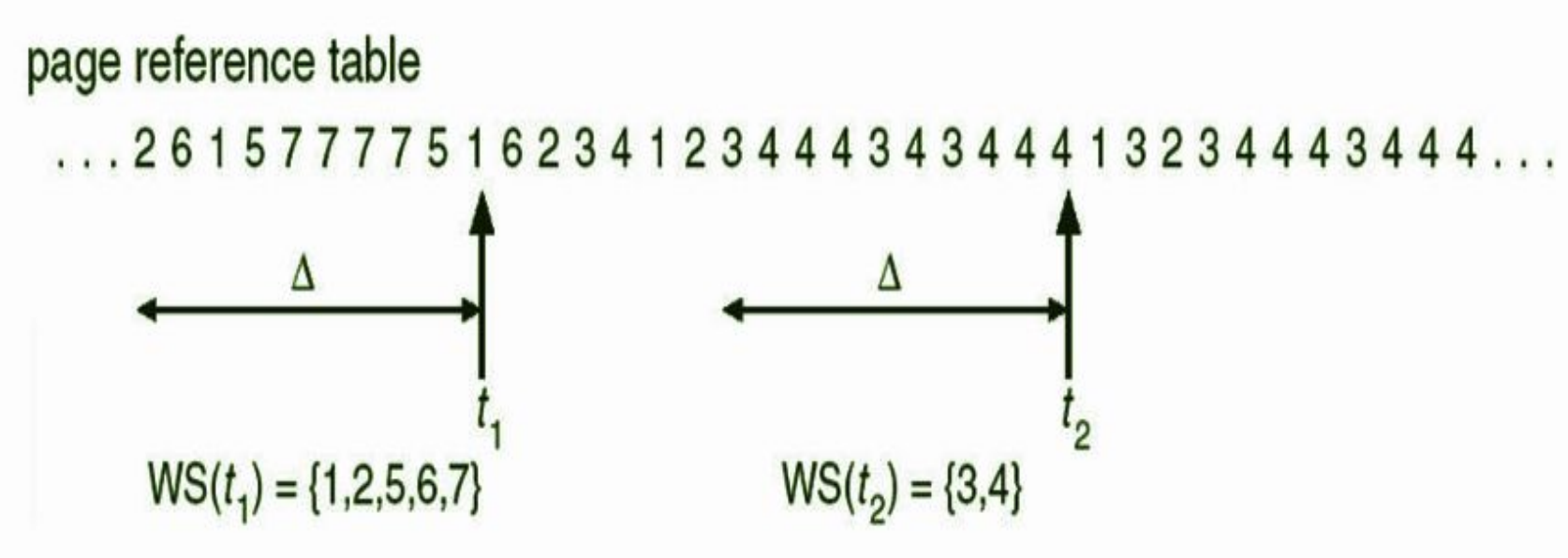
- Why does thrashing occur ?
  - **Size of Locality** > Total Memory Size
    - Limit effects by using local or priority page replacement

Working Set Model

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

Working Set Model

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



Working Set Model - Additional Input

- Working Sets: Conceptual model proposed by Peter Denning to prevent thrashing.

- **Informal definition:** the collection of pages a process is using actively, and which must thus be memory-resident to prevent this process from thrashing.

- If the sum of all working sets of all runnable threads or active process exceeds the size of memory, then stop running some of the threads or process for a while.

Working Set Model - Additional Input

- Divide processes into two groups: **active** and **inactive**:
  - When a process is **active** its entire working set must always be in memory: never execute a thread or a process whose working set is not resident.
  - When a process becomes **inactive**, its working set can migrate to disk.
  - Threads or child processes from inactive processes are never scheduled for execution.
  - The collection of **active** processes is called the **balance set**.
  - The system must have a mechanism for gradually moving processes into and out of **the balance set**.
  - As working sets change, **the balance set** must be adjusted.

Working Set Model - Additional Input

- How to compute working sets ?
    - Denning proposed a working set parameter T: all pages referenced in the last T seconds comprise the working set.
    - Can extend the clock algorithm to keep an idle time for each page.
    - Pages with idle times less than T are in the working set.
- Difficult questions for the working set approach:
    - How long should T be (typically minutes) ?
    - How to handle changes in working sets ?
    - How to manage the balance set ?
    - How to account for memory shared between processes ?

Keeping track of the Working Set

- Approximate with interval timer + a reference bit
- Example: Δ = 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 page in working set
- Why is this not completely accurate ?
- Improvement = 10 bits and interrupt every 1000 time units

Current Situation - Additional Input

In practice, today's operating systems don't worry much about thrashing:

- With personal computers, users can notice thrashing and handle it themselves:
  - Typically, just buy more memory
  - manage balance set by hand
- Thrashing was a bigger issue for timesharing machines with dozens or hundreds of users:
  - **For a typical question like why should I stop my processes just so you can make progress ?**
  - **Answered by making the System to handle thrashing automatically.**
- Technology changes make it unreasonable to operate machines in a range where memory is even slightly overcommitted; better to just buy more memory.

Page-Fault Frequency

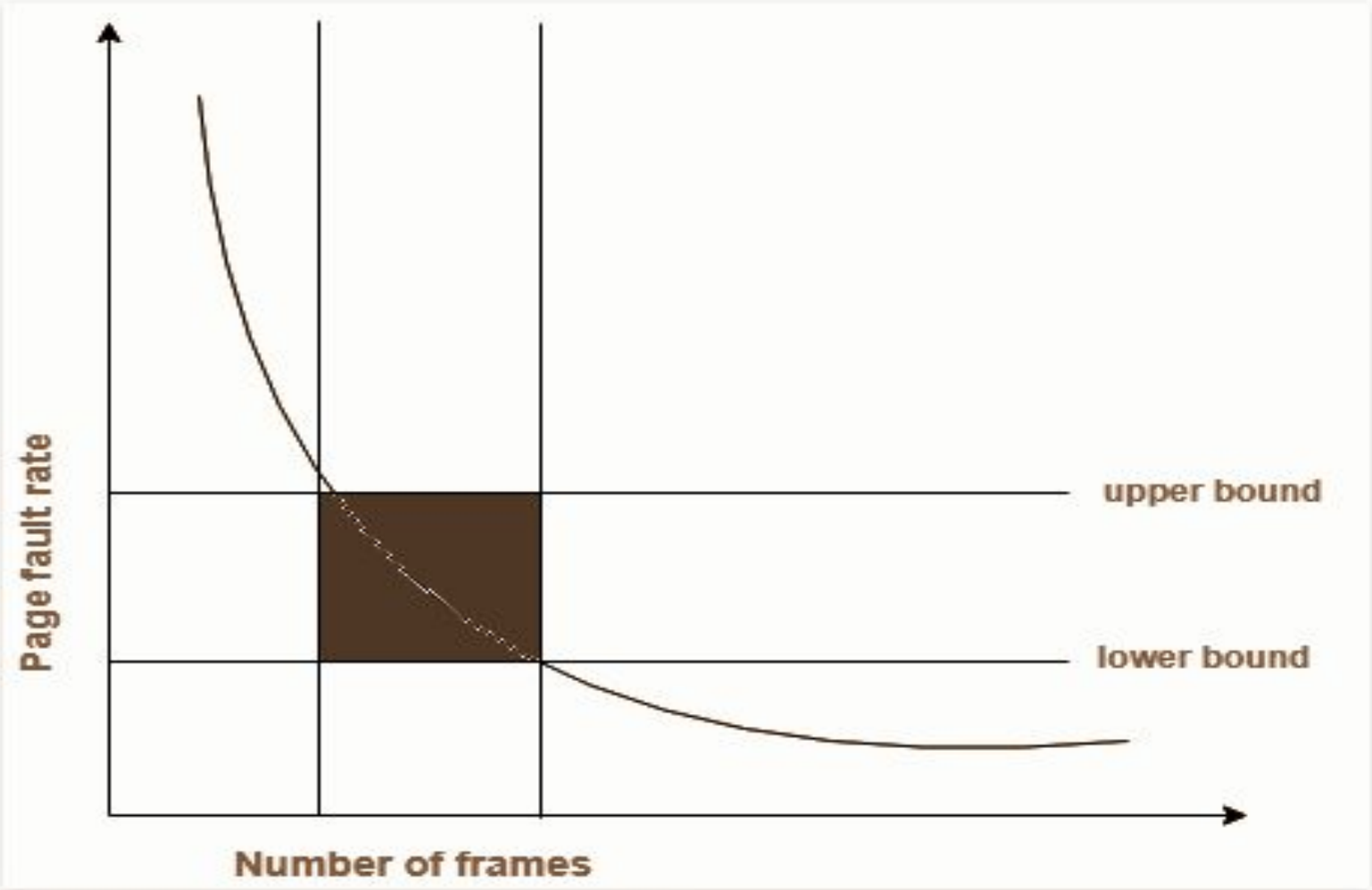
- More direct approach than Working set of process S
- Establish “acceptable” page-fault frequency (PFF) rate for a process and use local replacement policy
  - If actual rate too **low**, process **loses** frame
    - A low page fault rate indicates that the process has **too many frames**.
  - If actual rate too **high**, process **gains** frame
    - If the page fault rate is too high, it indicates that the process **has too few frames** allocated to it.

Page-Fault Frequency - Additional Input

Page Fault Frequency: another approach to preventing thrashing.

- Per-process replacement; at any given time, each process is allocated a fixed number of physical page frames.
- Monitor the rate at which page faults are occurring for each process.
- If the rate gets too high for a process, assume that its memory is overcommitted; increase the size of its memory pool.
- If the rate gets too low for a process, assume that its memory pool can be reduced in size.
- If the sum of all memory pools don't fit in memory, deactivate some processes.

Page-Fault Frequency - Additional Input



Topic Outline

- Virtual Memory - Thrashing
- Demand Paging and Thrashing
- Working-Set Model
- Keeping Track of the Working Set
- Page-Fault Frequency

THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

nitin.pujari@pes.edu

For Course Deliverables by the Anchor Faculty click on www.pesuacademy.com