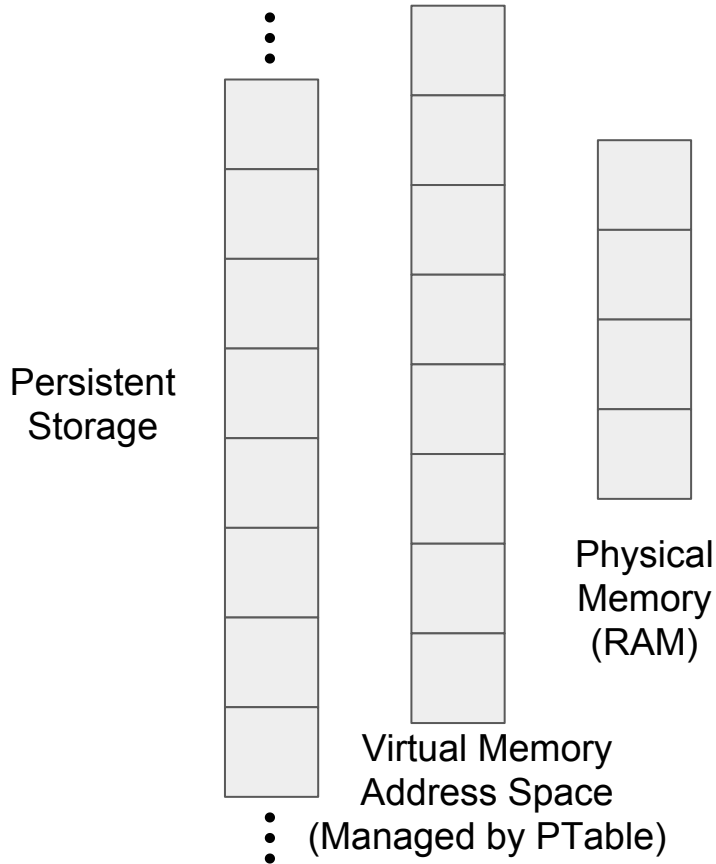


Discussion 3/29/19

Physical and Virtual memory

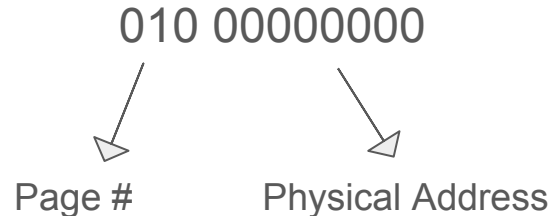
- Physical memory has a limited size (i.e. my laptop has 16GB of RAM)
- We want to simulate an infinite amount of physical memory.
- Virtual memory attempts to simulate more memory than exists on the system through automatic “overlays”.
- We do this by offering a “virtual address space” that allows the user to access any address within the system (i.e. 32 bit or 64 bit).
- The virtual address space of a 32 bit system can hold a maximum of 4GB of data.

Physical and Virtual memory



- The virtual memory address space is managed by a Page Table.
- A page table entry contains:
 - If page is valid
 - If frame is clean or dirty
 - Corresponding frame #
 - More!
- Virtual Addresses have this format:

Virtual Address:



Physical and Virtual memory

Persistent
Storage

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

Page Table
F# D/C Val

| | | |
|--|---|---|
| | C | F |
| | C | F |
| | C | F |
| | C | F |
| | C | F |
| | C | F |
| | C | F |
| | C | F |

Physical
Memory
(RAM)

| |
|--|
| |
| |
| |
| |

- Virtual Address format: first 4 bits = physical address, last 3 = page number.
- Consider the following instructions:

R 0100101

Physical and Virtual memory

Persistent
Storage

Page Table
F# D/C Val

| | | | |
|---|----------|---|----------|
| 0 | | C | F |
| 1 | | C | F |
| 2 | 0 | C | T |
| 3 | | C | F |
| 4 | | C | F |
| 5 | | C | F |
| 6 | | C | F |
| 7 | | C | F |

Physical
Memory
(RAM)

| |
|----------|
| 2 |
| |
| |
| |

- Virtual Address format: first 4 bits = physical address, last 3 = page number.
- Consider the following instructions:

R 0100101

W 0001111

Physical and Virtual memory

Persistent
Storage

Page Table
F# D/C Val

| | | | |
|---|---|---|---|
| 0 | 1 | D | T |
| 1 | | C | F |
| 2 | 0 | C | T |
| 3 | | C | F |
| 4 | | C | F |
| 5 | | C | F |
| 6 | | C | F |
| 7 | | C | F |

Physical
Memory
(RAM)

| |
|---|
| 2 |
| 0 |
| |
| |

- Virtual Address format: first 4 bits = physical address, last 3 = page number.
- Consider the following instructions:

R 0100101

W 0001111

W 1101010

Physical and Virtual memory

Persistent
Storage

Page Table
F# D/C Val

| | | | |
|---|---|---|---|
| 0 | 1 | D | T |
| 1 | | C | F |
| 2 | 0 | C | T |
| 3 | | C | F |
| 4 | | C | F |
| 5 | | C | F |
| 6 | 2 | D | T |
| 7 | | C | F |

Physical
Memory
(RAM)

| |
|---|
| 2 |
| 0 |
| 6 |
| |

- Virtual Address format: first 4 bits = physical address, last 3 = page number.
- Consider the following instructions:

R 0100101

W 0001111

W 1101010

R 0001111

Physical and Virtual memory

Persistent
Storage

Page Table
F# D/C Val

| | | | |
|---|---|---|---|
| 0 | 1 | D | T |
| 1 | | C | F |
| 2 | 0 | C | T |
| 3 | | C | F |
| 4 | | C | F |
| 5 | | C | F |
| 6 | 2 | D | T |
| 7 | | C | F |

Physical
Memory
(RAM)

| |
|---|
| 2 |
| 0 |
| 6 |
| |

- Virtual Address format: first 4 bits = physical address, last 3 = page number.
- Consider the following instructions:

R 0100101

W 0001111

W 1101010

R 0001111 // **Nothing!**

R 1000000

Physical and Virtual memory

Persistent
Storage

Page Table
F# D/C Val

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

| | | |
|---|---|---|
| 1 | D | T |
| | C | F |
| 0 | C | T |
| | C | F |
| 3 | C | T |
| | C | F |
| 2 | D | T |
| | C | F |

Physical
Memory
(RAM)

| |
|---|
| 2 |
| 0 |
| 6 |
| 4 |

- Virtual Address format: first 4 bits = physical address, last 3 = page number.
- Consider the following instructions:

R 0100101

W 0001111

W 1101010

R 0001111 // **Nothing!**

R 1000000

W 10010101

Physical and Virtual memory

Persistent
Storage

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

Page Table

F# D/C Val

| | | |
|---|---|---|
| 1 | D | T |
| | C | F |
| 0 | C | T |
| | C | F |
| 3 | D | T |
| | C | F |
| 2 | D | T |
| | C | F |

Physical
Memory
(RAM)

| |
|---|
| 2 |
| 0 |
| 6 |
| 4 |

- Virtual Address format: first 4 bits = physical address, last 3 = page number.
- Consider the following instructions:

R 0100101

W 0001111

W 1101010

R 0001111 // **Nothing!**

R 1000000

W 10010101

Page Replacement Algorithms

- When our physical memory is full, we must eject a page from a frame in order to create space to load.
- The method in which we choose what page to eject is a **Page Replacement Algorithm**.
- Consider **FIFO**
 1. Create a queue
 2. When loading a page into a frame, add it into the queue
 3. When ejecting a page from a frame, choose the head of the queue

Physical and Virtual memory

Persistent
Storage

Page Table
F# D/C Val

| | | | |
|---|---|---|---|
| 0 | 1 | D | T |
| 1 | | C | F |
| 2 | 0 | C | T |
| 3 | | C | F |
| 4 | 3 | D | T |
| 5 | | C | F |
| 6 | 2 | D | T |
| 7 | | C | F |

Physical
Memory
(RAM)

| |
|---|
| 2 |
| 0 |
| 6 |
| 4 |

- Instructions:

R 0100101

W 0001111

W 1101010

R 0001111 // **Nothing!**

R 1000000

W 10010101

- Our queue is: **2 -> 0 -> 6 -> 4**
- Now consider:
R 1110000

Physical and Virtual memory

Persistent
Storage

Page Table
F# D/C Val

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

| | | |
|----------|---|----------|
| 1 | D | T |
| | C | F |
| | C | F |
| | C | F |
| 3 | D | T |
| | C | F |
| 2 | D | T |
| 0 | C | T |

Physical
Memory
(RAM)

| |
|----------|
| 7 |
| 0 |
| 6 |
| 4 |

- Instructions:

R 0100101

W 0001111

W 1101010

R 0001111 // **Nothing!**

R 1000000

W 10010101

- Our queue is: **2 -> 0 -> 6 -> 4**

- Now consider:

R 1110000 // **0 -> 6 -> 4 -> 7**

R 0100101

Physical and Virtual memory

Persistent
Storage

Page Table
F# D/C Val

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

| | | |
|---|---|---|
| | D | T |
| | C | F |
| 1 | C | T |
| | C | F |
| 3 | D | T |
| | C | F |
| 2 | D | T |
| 0 | C | T |

Physical
Memory
(RAM)

| |
|---|
| 7 |
| 2 |
| 6 |
| 4 |

- Instructions:

R 0100101

W 0001111

W 1101010

R 0001111 // **Nothing!**

R 1000000

W 10010101

- Our queue is: **2 -> 0 -> 6 -> 4**
- Now consider:

R 1110000 // **0 -> 6 -> 4 -> 7**

R 0100101 // **6 -> 4 -> 7 -> 2**

Project 4

- You will:
 - Implement a page table and page replacement algorithms.
 - Simulate memory access' based on execution traces.
 - Collect statistics about each algorithms performance.
 - Visualize and report your findings.
- You will be making a single-level page table for a 32-bit address space.
- Each page will be 2KB. The number of frames will be passed as a parameter.
- You may use any of C/C++, Java, or Python as long as it runs on lectura!
- Write a program that takes:

`vmsim -n <numframes> -a <opt|clock|fifo|rand> <tracefile>`

Page Replacement Algorithms

- **OPT**
 - What the optimal page replacement would choose if it had perfect knowledge (Which you do!)
- **Clock**
 - The better implementation of the second chance algorithm.
- **FIFO**
 - Evict oldest page in memory
- **Random Eviction**
 - Randomly select page to evict

Note: Implementing OPT in a naïve fashion will lead to unacceptable performance. It should not take more than 5 minutes to run your program.

How it works

- During execution, you will print out whats happening in the algorithm:
 - Hit
 - Page fault - no eviction
 - Page fault - evict clean
 - Page fault - evict dirty
- Print statistics:

Algorithm: %s

Number of frames: %d

Total memory accesses: %d

Total page faults: %d

Total writes to disk: %d

Total number of page table leaves: %d

Total size of page table: %d bytes

Traces

- You are given two example traces (found on [lectura](#) at [~jmisurda/original](#))
- They have the following format:

I 0023C790,2 # instruction read at 0x0023C790 of size 2

I 0023C792,5

S BE80199C,4 # data store at 0xBE80199C of size 4

I 0025242B,3

L BE801950,4 # data load at 0xBE801950 of size 4

I 0023D476,7

M 0025747C,1 # data modify at 0x0025747C of size 1

Trace Instructions

- Instruction formats:
 - **Modify:** M addr,size (This is both a load and a store)
 - **Read:** I addr,size
 - **Store:** S addr,size
 - **Load:** L addr,size
- Ignore lines that do not follow the above formats

Write up

- For each of the four algorithms, describe in a document the resulting page fault statistics for 8, 16, 32, and 64 frames. Plot the number of page faults versus the number of frames
- Use this information to determine which algorithm you think might be most appropriate for use in an actual operating system. (Make sure to give your reasoning)
- Use OPT as the baseline for your comparisons
- Additionally, include if you find any occurrences of Beladys anomaly for FIFO.
- You will turn in:
 - Source code
 - A .doc or .pdf of your report
 - **DO NOT SUBMIT TRACES**