CSCI 360 - Lab 7 Report

Sami Al-Qusus
Nov 20, 2018

Assignment Goals:

- Simulation to Measure Page Fault Frequency

Assignment Instructions (Dr. LoPinto):

- The goal is to simulate the operation of a paging system to study the frequency of page fault under different page replacement algorithms.

- You can use this Memory Reference Simulator to generate virtual addresses. The program is believed to be correct but there are no guarantees. If you find a bug, report it to me. You can modify the simulator to experiment with making the working set of pages larger and/or to change the sizes of the simulated code and memory segments.

- The simulator calls mmu() which must convert virtual to physical addresses. Your MMU function should maintain a page table. And should record blocks that are referenced and whether the reference was for reading or writing. Assume that there are 10 page frames of physical memory but program this as an adjustable parameter so you can experiment with different size physical memories.

- Implement two page replacement algorithms - FIFO and Not Recently Used. Compare the page fault frequency (PFF) for the two algorithms. Your report should document the simulator as well as you own code and report on how you measured PFF for the different algorithms and assumptions and what you can conclude from the exercise.

Review

- Main memory (RAM) must be very carefully managed.
- Memory manager: manages memory hierarchy.
    - Keep track of which parts of memory are in use, allocate memory to processes when they need it, and deallocate it when they are done.
- RAM needed by all the processes is often much more than can fit in memory.
- Two general approaches to dealing with memory overload:
    - Swapping: bringing in each process in its entirety, running it for a while, then putting it back on the disk.
    - Virtual memory: allows pro- grams to run even when they are only partially in main memory.

- Each program has its own address space.
    - Broken up into chunks called **pages**.
    - These pages are mapped onto physical memory, but not all pages have to be in physical memory at the same time to run the program.
- MMU (Memory Management Unit) that maps the virtual addresses onto the physical memory addresses.
- Computer generates 16-bit addresses, which are the virtual addresses 0 up to 64K – 1.
- In our example we have 10x256 bytes of memory corresponding to 10 frames and 100x256 bytes of instructions of data and code combined on the disk corresponding to 100 pages.
- Because we can not have all 100x256 bytes in main memory, the mmu will put 10 in main memory at a time.
    - The mmu has a table to keep track and map each page to frame.
    - The table also keeps track of present/absent, referenced, modified, cached, protections, and frame number if page is currently mapped to a frame.
- If the program references an unmapped address, the mmu causes the cpu to trap the operating system.
    - Trap is called a **page fault**.
- The operating system picks a little-used page frame and writes its contents back to the disk (if it is not al- ready there).
- It then fetches (also from the disk) the page that was just referenced into the page frame just freed, changes the map, and restarts the trapped instruction.
- When a page fault occurs, the operating system has to choose a page to evict.
- If the page to be removed has been modified while in memory, it must be rewritten to the disk to bring the disk copy up to date.
- If, however, the page has not been changed, the disk copy is already up to date, so no rewrite is needed. The page to be read in just overwrites the page being evicted.
- While it would be possible to pick a random page to evict at each page fault, system performance is much better if a page that is not heavily used is chosen.
- If a heavily used page is removed, it will probably have to be brought back in quickly, resulting in extra overhead.
- Much work has been done on the subject of page replacement algorithms, both theoretical and experimental.

- The **NRU** (**Not Recently Used**) algorithm removes a page at random from the lowest-numbered nonempty class.
  - Class 0: not referenced, not modified.
  - Class 1: not referenced, modified.
  - Class 2: referenced, not modified.
  - Class 3: referenced, modified.
- **FIFO** (**First-In, First-Out)**

Task: Simulation to Measure Page Fault Frequency

What I did:

- I have to measure page fault frequency of nru vs fifo.
- So I started by using the provided memory reference simulator to generate 2500 instructions. Each instruction references code and data.
  - The simulator randomly decides on a page and randomly decides on an offset to produce an address.
- For consistency I stored the instructions produced into a file.
  - ./genData>data.txt
- I then created to c programs.
  - Both start by creating 4 arrays to represent the table in the mmu.
    - int frame[PAGES];
    - int absentPresent[PAGES];
    - int modified[PAGES];
    - int referenced[PAGES];
  - The four arrays represent page table entry.
  - Although the page table should also have protections, and sometimes caching disabled entry, we leave them out because that's not the purpose of this lab. We just need what's required by the algorithms.
  - I start by first zeroing the arrays to represent an empty table.
  - Then take line by line from data.txt and break the instruction into referenced, modified and address.
    - For example: Read from 1f59
    - Read means modified is 0, referenced =1 and the array or page index is 1f59/256.
  - Then we use the index and plug it in absentPresent[1f59/256]
  - If it returns 1 then the page is in main memory or a frame.
    - If that's the case we have a couple of case to check
    - If referenced and not modified before but modified now
      - Then increment page fault and update modification=1.
    - If present modified but not referenced then update to reference=1.
    - If present before but not modified or referenced and both now then update both and increment page fault.
  - If absentPresent returns 0

- If i<10 increment page fault and set referenced, modified and map to one of the empty pages.
- If i>0 increment page fault
    - For nru create a vector
        - Go through each class and pick a random page to evict from the lowest class.
    - For fifo
        - Create a linked list every time an address is first mapped. Then when space is needed evict head of linked list and remove it.
- At the end I printed the page fault.
    - Fifo gave a specific number every time because its not random.
        - Page fault was 1950
    - Nru give different numbers because its random so I ran the program on the same instructions 5 times and took average: 2760+ 2753+ 2748+ 2791+ 2803/5= 2771

Therefore comparing fifo to nru, we see at 1950 page faults for fifo compared to an average of 2771 for nru. Fifo is a better algorithm.

What I learnt:

- I learnt how simulate the memory management process.
- Create a table mapping pages to frames.
- Managing page faults.
    - Keep track of which parts of memory are in use, allocate memory to processes when they need it, and deallocate it when they are done.
- I was able to conclude that fifo is about 30% better in decreasing page faults. But of course more things decide this.

Note:

- I have provided outputs of both algorithms.
- Some comments in nru.c and fifo.c might be helpful.

Credits:

- Dr. LoPinto lab7 Memory Reference Simulator code.
- Modern Operating Systems 4th Edition--Andrew Tanenbaum
- https://linux.die.net/man/3/getline
- http://www.cplusplus.com/doc/tutorial/files/
- https://stackoverflow.com/questions/14265581/parse-split-a-string-in-c-using-string-delimiter-standard-c
- https://www.codesdope.com/blog/article/c-deletion-of-a-given-node-from-a-linked-list-in-c/