Nhan Tran

OS

11/30/12

Lab 3

1. Write up your analysis and design of the program describing the functionality of the major components of the program and how they interact with each other, and a specific functional analysis and design for the data structures and algorithms used in the Memory management portion of the program.

The program, outside of main(), consist of three important functions, two critical helper functions and two minor helper functions.

The basic breakdown is:

* main() will take in the user input file and the selected number of frames
* main() will then call three functions: fifo(), lru(), and opt()
  + each of these functions will individually run their replacement policy algorithm and then return a page fault value to main()

The main data structure used in the program is a **List** from the Standard Template Library. Each replacement policy function will create its own list of a declared struct type called **frame**. A frame type is a type that is meant to represent a frame, which holds a pid and page reference number, there are two other field the frame type holds which is meant to be used for when computing the OPT, the fields are 1, to indicate if the frame is holding a future page reference and 2, how far away is that future page. The **List** structure is beneficial because it allows you to easily keep track of the order the pages come in from the input file. The order is very important to FIFO and also in LRU and OPT.

Algorithms:

FIFO

Page comes in, does page exist?

* No – need to place page in physical memory
  + Is physical memory full?
    - No – put page into the back of the list
    - Yes
      * remove the front of list (oldest)
      * put new page into the ‘frame’ in the back of the list
* Yes – it’s a hit, don’t need to do anything

LRU

Page comes in, does page exist?

* No – need to place page in physical memory
  + Is physical memory full?
    - No – put page into the back of the list
    - Yes
      * remove front frame, which is the least recently used
      * put new page into frame in the back of the list
* Yes – it’s a hit
  + move the frame location of this page in the list to the bottom of the list because though it’s a hit this page is also most recently used so putting in the back of the list will indicate so

OPT

Page comes in, does page exist?

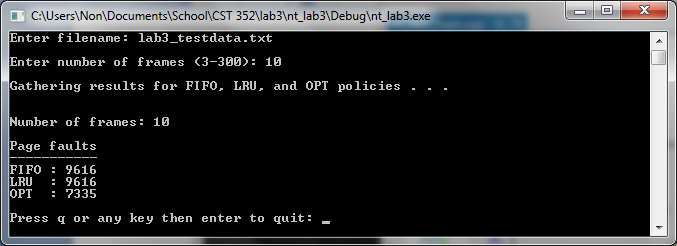
* No – need to place page in physical memory
  + Is physical memory full?
    - No – put page into the back of the list
    - Yes
      * for every frame in physical memory
        + check all the future pages to see if the memory will be needed to load again in the future
        + the first frame from the top of the list that is NOT needed again in the future is the frame that will be removed
        + if every frame in physical memory will be needed again

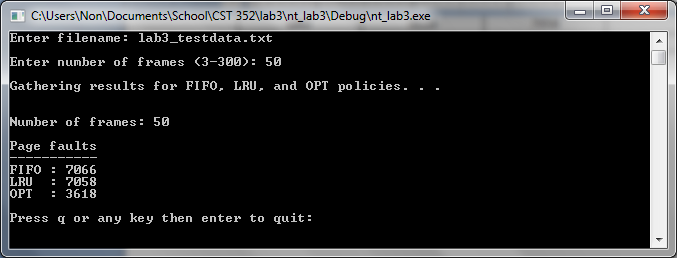
the frame with the page information that is farthest away from the current page will be removed

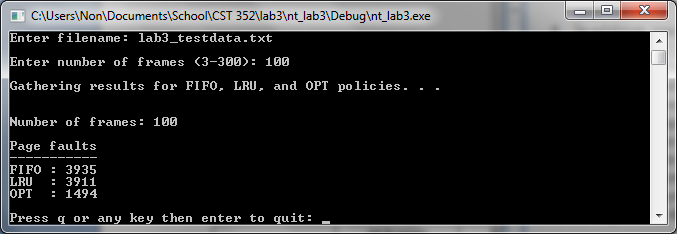
* + - * + if there aren’t any more future pages then removed the oldest frame, which is the one at the front of the list
      * put new page into frame in the back of the list
* Yes – it’s a hit, no need to do anything

**2. Simulation Results of Number of Page Faults:**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **10 frames** | **50 frames** | **100 frames** |
| **FIFO** | 9616 | 7066 | 3935 |
| **LRU** | 9616 | 7058 | 3911 |
| **OPT** | 7335 | 3618 | 1494 |

****

****

****

1. In addition, you should answer the following questions:
2. Which replacement strategy would you choose and why? You should consider both the end results and the effort it took to implement each strategy. Discuss what your results show about the relative merits of FIFO, LRU, and OPT for the different input files.

According to the page fault results I would have to say OPT might be the best way to go. The reason why I say ‘might’ is because the overhead of computing the ‘optimal’ way to do it was great. Even so, the performance savings with less page faults and having the memory loaded might outweigh the overhead of calculating the OPT. It is a tough trade off in either case. From the results it looks like the more frames there are, the more efficient LRU is to FIFO, and the more efficient OPT is to both FIFO and LRU.

1. What aspect of memory management did you find most difficult to implement?

No doubt the OPT replacement policy was the most difficult because it does so much checking on a few different things on every single page replacement.

1. What aspect of memory management did you find least difficult to implement?

The LRU was easier to implement then I originally thought. With my use of a linked list and sorting the order of oldest (front) to newest (bottom), the LRU replacement is exactly the same as FIFO with only three extra lines of code.

1. What, if anything, would you change in your current design?

The FIFO and LRU is very similar and I could have included both fifo() and lru() into one function but choose to keep them separated for better readability. For FIFO and LRU I read the input file as I was doing the calculations but with OPT this couldn’t be done so I read the input file once and stored all the data in frame objects in a List. I could have done this for FIFO and LRU and not mess around with re-reading the input file a few times.

1. What, if anything, did you find interesting or surprising about page replacement algorithms that you did not know before doing this project?

The most interesting and surprising thing I saw was the result of LRU compared to FIFO. I was shocked that LRU only provided a very small improvement over FIFO. I thought the improvement would be a lot more than 0, 8, and 24 across 10, 50, and 100 frames respectively. The fact that OPT does so much just to prevent page faults was also a little interesting but after seeing the results and the dramatic improvement in page faults OPT offers I can see why some would go this route.