README.pdf
CS570 Operating Systems Summer 2017
Ryan Ragasa (cssc1144)
Paul Nguyen (cssc1143)
Assignment #2
Dr. Guy Leonard
6/17/2017
Test Account: cssc1143

File List:
a2.c
a2.h
clock.c
opt.c
secondchance.c
Makefile
pages.txt
README.pdf

How to run. Type commands as follows:

```
make run      (to compile executable)
bots          (to execute code)
```

User will get prompted with:
"Please enter desired number of frames: "

To which the user should enter a number

To clear executable file:
make clean

Code Decisions and Comments:

function: main
Main file that will initiate all other c files to test our three algorithms; OPT, Clock, and Second Chance. a2.c contains the main function, promptUser(), and getPagesInfo(). Prompt user is a simple function which reads user input and passes it onto the global variable, frameNumber. There is no error checking on the user
input as there was no need/ not in the requirements. We assumed the user would input a reasonable frame number that would not ultimately destroy our hard work =(. getPagesInfo() has two primary goals; parse the pages.txt file into an array that we could use and fill all empty pages in the array with -1. As C does not differentiate

between 0 and null, -1 is being used to signify a null/non-existant term. This will allow us to us 0 in as a possible page number. The main function executes these two functions, as well as call functions for our three algorithms as stated previously, before gracefully exiting out with a simple message.

function: opt
Note: To build the opt we made the design decision to make 2 separate counters when figuring out whether or not the frame was actually in the remaining pages of the sequence. We also made a while loop to cycle through the sequence array itself. The most dificult part was figuring out the logic of when and where to set the counters as well as utilizing the "furthest away" frame as well as whether or not it was in the sequence at all. No extra features were implemented that seemed to be "not required" nor were there any bugs that appeared to have manifested given that the arguments were a valid int for the frameNumber and int[] for the sequence. We learned that while setting one variable to be the furthest frame down the sequence was a good option, using two was just as effective.

function: secondchance
Note: To build the Second Chance Algorithm, we set a loop to check if the page number exists in a frame,setting or resetting the reference bit accordingly if frames exist or do not exist, etc. We also loop through to reset the position of pages. The most difficult part of this was understanding how to cycle through the frames and capture the correct frame to remove with the reference bit.

function: clock
Note: To build the clock we made the design decision to go with an int counter to keep track of the pointer cycling through the frames. We also made a while loop to cycle through the sequence array. The most difficult part was figuring out the logic of when and where to set the reference bits. No extra features were implemented that seemed to be "not required" nor were there any bugs that appeared to have manifested given that the arguments were a valid int for the frameNumber and int[] for the sequence. We learned that while an actual "address pointer" could have been useful it wasn't the only way to approach this problem.

Assignment 2 Analysis

For this Assignment we had to study, code and assess three page replacement algorithms: Clock, OPT(Optimal Page Replacement), and Second Chance.

The way the Clock algorithm works is by approximating how old each page in the frame is and looks for the one that has been in the stack the longest and has not been referenced recently. To accomplish this we needed to add a reference bit that would indicate whether or not it had been recently referenced. The rules for the clock algorithm are as follows. When the page is first loaded into memory, it’s used bit is set to one. When the page is referenced, the used bit is set to one as well. When the memory is full a pointer is used and dereferences the frames one by one until it finds a page on the stack whose used bit is zero.

Upon testing this algorithm we first intentionally used three frames. For this test using the first sequence of pages (in comparison to the others), the Clock page algorithm returned the most number of page faults, totalling a grand total of 8. The second sequence of pages yielded event more surprising results. Both clock and second chance resulted in the highest number of faults, at 12. This shows us that the sequence of pages being called plays a significant role in the algorithms' performances. Keep in mind that during the implementation of our program we took into account "empty frames" and set those to be page faults as well. For the programming implementation of this algorithm we used three main functions(the other two follow the same structure).

The second page replacement algorithm we analyzed was the OPT(Optimal) replacement algorithm. What this algorithm does is that it replaces the page for which the next reference is furthest along in the future. In a run time system it is impossible to know what the future events are but it is important to analyze because it creates less faults than the previous algorithms allows us to benchmark our tests. However, given a big enough frame size to work with, the OPT algorithm can "level" in terms of the number of faults as compared to the other algorithms.

For the analysis of the OPT page replacement algorithm, there was several key differences. When we assigned less frames to our tests, the OPT algorithms was always the clear winner with less page faults. For example with a frame stack of 3, the OPT returned a total of 6 page frames. For the second test case, using a different page sequence and the same frame stack, OPT again returned less page faults than the other two algorithms respectively. As we increased the frame stack to 5, OPT still returned less page faults than the other two. Same with 7, however the margin of difference was only by a factor of "1."

The way the Second Chance page algorithm works is by simply modifying FIFO so that it avoids the problem of throwing out a heavily used page by inspecting a reference bit of the oldest page. If it is 0, the page is both old and unused, so it can be replaced immediately. If the reference bit is 1, the bit is cleared, the page is put onto the end of the list of pages, and its load time is updated as though it had just arrived in memory. Then the search continues.

Upon testing the Second Chance page algorithm, we followed the same methodology as previously stated and noticed something peculiar. Despite being a FIFO variant with expected improved results. we notice that the Second Chance page algorithm would underperform. Our perception was that it would have comparable performance to Clock as they both use a way to refer to heavily used pages. But as noted, the sequence being called plays a significant role in the performance of any of the page algorithms.

To conclude, the following tests that we conducted for these three page replacement algorithms demonstrate that there are in fact key differences that need to be noted.

When it came down to using smaller page frames, the OPT algorithm usually resulted with significantly less page faults than Clock or Second Chance. However, as we increased the frame stack size, they all became more prevalent in their page faults. Below are the following test cases that we used in order to form this analysis. We used two different types of Page sequences, as well as different lengths in frames within those page sequences.

TEST#1: Pages:2 3 2 1 5 2 4 5 3 2 5 2
Frames:3
    Clock Page Faults: 8
    Second Chance Page Faults: 6
    OPT Page Faults: 6

TEST#2: Pages:2 3 2 1 5 2 4 5 3 2 5 2
Frames:5
    Clock Page Faults: 5
    Second Chance Page Faults: 5
    OPT Page Faults: 5

TEST#1: Pages: 3 2 3 0 8 4 2 5 0 9 8 3 2
Frames:3
    Clock Page Faults: 12
    Second Chance Page Faults: 12
    OPT Page Faults: 9

TEST#2: Pages: 3 2 3 0 8 4 2 5 0 9 8 3 2
Frames:5
    Clock Page Faults: 9
    Second Chance Page Faults: 11
    OPT Page Faults: 7