A Report in which Various Memory Management Algorithms versus Page Faults in Virtual Memory is Discussed

Jesse Lieberg

CSCI460

4/29/17

Overview

I ran into some issues in the assignment and had to push fixing some issues to try and get the assignment finished. I will review the issues and how they pertain to the issues.

Implementation

For my version of the capstone project I opted to write it in Java, as I find the language easier to work with and enjoy the safety. I used an object-oriented approach. My files consisted of the following:

- Process.java
    - A class that emulates a process. It has basic data including pid, process size, what page the process starts on, and various functions to get the page size based on a relative address.
- Frame.java
    - A class that keeps track of frames, including their address and reference bit used in the second chance algorithms.
- FrameTable.java
    - A class that keeps track of various elements of the memory management system. It knows the buffer size of frames and keeps track of a linked list representing a stack of frames and an array of all available processes.
- Driver.java
    - The class that checks input and reads in files for the simulation

For the implementation of the frame table I opted for a linked list. This allowed me to easily remove the last value for the FIFO and LRU algorithms by keeping the most recently added/used frames at the beginning of the table.

I ran into some issues with second chance LRU algorithm. My algorithm is as follows:

```
if current_frame_to_add is not in frame_table{
    if no room to insert current_frame_to_add {
        traverse frame_table in reverse order
            if current_frame can be replaced {
                remove current_frame
                insert current_frame_to_add to beginning of frame_table
                increase fault_count
                return
            }
            else {
                give current_frame second chance
            }
        }
        remove last frame from frame_table
        insert current_frame_to_add to beginning of frame_table
    }
    increase fault count
    return
}
else {
    remove current_frame_to_add
    insert current_frame_to_add to beginning of frame_table
}
```

The page fault value was always the same as regular LRU. After testing the algorithm and having other students look at it, I had to put it off as a loss as I simply didn't have time to fix the issue. Because of this issue, the LRU and

second chance LRU demand algorithms have the same values, as well as the LRU and second chance LRU for pre-paging having the same lines.

Results

The data I obtained from the simulations are shown in table 1. I highlighted the effected columns to more easily highlight where the duplicate values were.

page size	FIFOD	LRUD	SCLRUD	FIFOP	LRUP	SCLRUP
1	117902	116916	116916	124006	122758	122758
2	92889	87568	87568	101170	94838	94838
4	80103	72496	72496	105080	91150	91150
8	77220	65134	65134	142148	99407	99407
16	83830	61322	61322	168734	191331	191331

Table 1. Results

By putting the data into Excel I obtained the following graph:

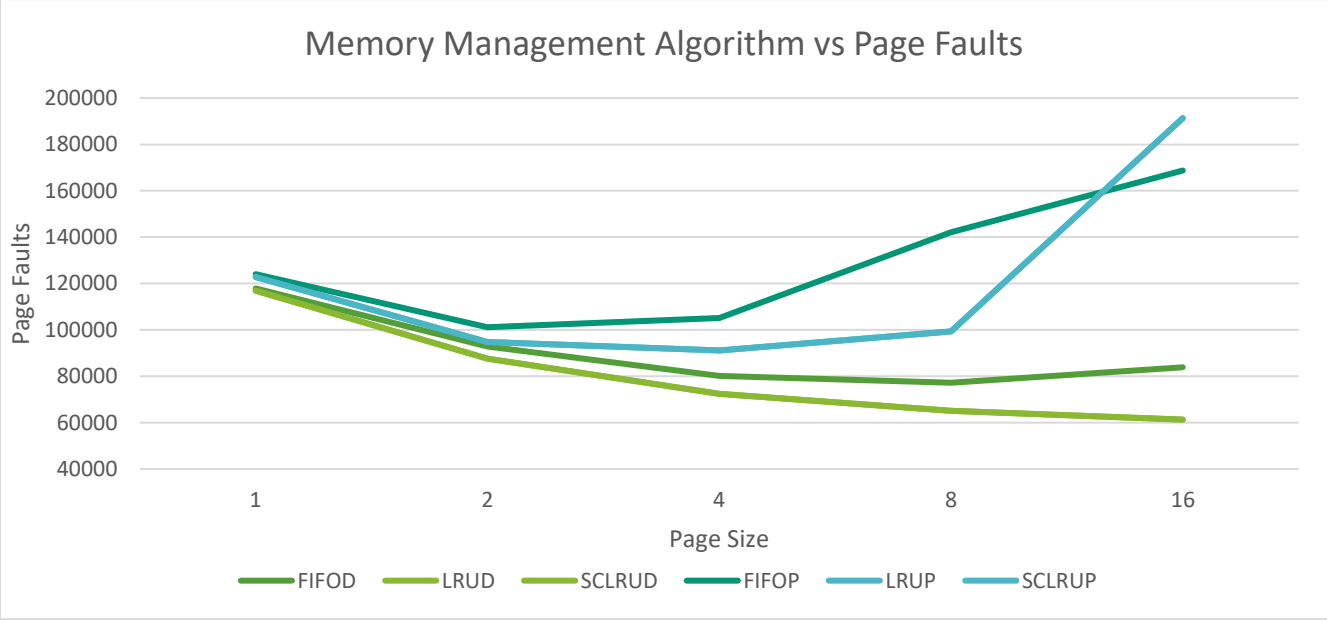

Figure 1. Memory management Algorithm vs Page Faults

As expected, there were more page faults as the page size was increased. My results showed LRU as the optimal algorithm, with FIFO following close but showing a slight raise at the end. The demand paging was far more optimal in all algorithms. Pre-paging seemed to negatively affect performance. I account this phenomenon to the algorithm not considering the common used commands, and always just inserting the next page (if needed). This potentially caused double the number of faults for many of the additions, while demand paging would only have at most one fault per addition.

Conclusion

As you can see, my results were less than I had expected to have done. Overall I am happy with the design of the project as it made finding bugs in general easier. If I had more time to work on the project I would gladly try to fix the issues, but time is a limited asset and I seem to have run out, despite working on the project off and on for the few weeks leading up and more-so the final week.