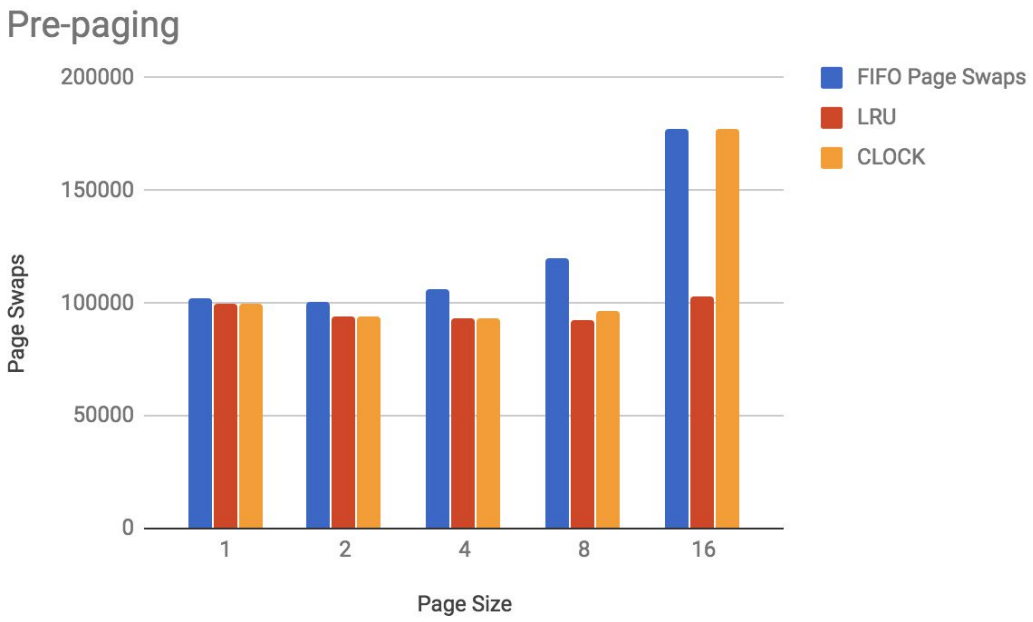
Khayyam Saleem, Faris Ibrahim
CS492 -- Operating Systems
Programming Assignment #2 Writeup
15 April 2018

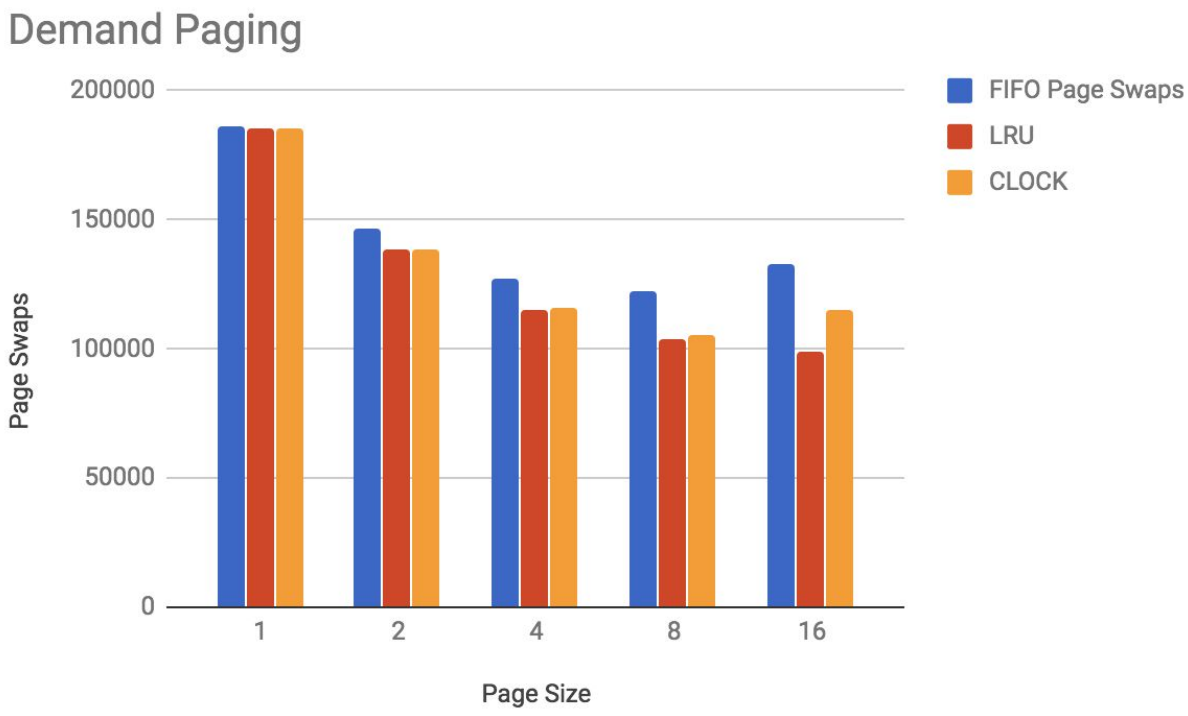
With Pre-paging

| Pre-paging | | |
|---|---|---|
| **Page Size** | **Algorithm** | **Page Swaps** |
| 1 | FIFO | 102416 |
| 2 | FIFO | 100214 |
| 4 | FIFO | 106486 |
| 8 | FIFO | 120172 |
| 16 | FIFO | 177156 |
| 1 | LRU | 99702 |
| 2 | LRU | 94318 |
| 4 | LRU | 93190 |
| 8 | LRU | 92505 |
| 16 | LRU | 103298 |
| 1 | CLOCK | 99817 |
| 2 | CLOCK | 94393 |
| 4 | CLOCK | 93599 |
| 8 | CLOCK | 96758 |
| 16 | CLOCK | 177151 |



With Demand Paging

| Demand paging | | |
|---|---|---|
| **Page Size** | **Algorithm** | **Page Swaps** |
| 1 | FIFO | 186384 |
| 2 | FIFO | 146739 |
| 4 | FIFO | 127322 |
| 8 | FIFO | 122718 |
| 16 | FIFO | 133157 |
| 1 | LRU | 185063 |
| 2 | LRU | 138449 |
| 4 | LRU | 115484 |
| 8 | LRU | 103543 |
| 16 | LRU | 98930 |
| 1 | CLOCK | 185098 |
| 2 | CLOCK | 138501 |
| 4 | CLOCK | 115730 |
| 8 | CLOCK | 105376 |
| 16 | CLOCK | 115238 |



With the number of page swaps as a metric, LRU is the most efficient algorithm for virtual page management, with the CLOCK algorithm being a close second and FIFO the least.

**LRU** imposes significant overhead and is difficult to implement, but is based on the principle of locality, in that the page selected for replacement is the one that has not been referenced for the longest time.

The **CLOCK** algorithm takes advantage of a circular implementation for finding pages to replace. This implementation has less overhead, but could take longer to find a page to replace, and is hence beat out by LRU closely.

The **FIFO** algorithm had the simplest implementation by far, and required very little overhead. It replaces the page that had been in memory the longest every time. The greater the amount of page frames, the more page swaps it has to do and hence the more poorly it performs.

With **demand paging**, pages are brought into memory only when a location on the page is referenced during execution. With **pre-paging**, other pages are also brought into memory based on common access patterns, locality, or time last referenced with the hope that it will be needed in the near future and will hence not require another access to main memory.

Prepaging was more efficient than demand paging for page-sizes less than 8. They performed similarly for a page size of 8, and then demand paging took the lead for page sizes of 16 and up. This behavior matches our expectations given the descriptions of the differences in the algorithms above.

Random memory access traces would have significantly reduced performance, as the principle of locality states that, with pre-paging, we can reduce the number of memory accesses by taking advantage of memory locations being adjacent to one another. Using larger page sizes would not help to reduce page faults, as demonstrated by the above graphs.