By Thomas Flanagan and David Mattia

Section 1: Overall Description of Experiments

Project 5 dealt with page faults and how to hand virtual memory in a computer. Below the user level in a computer this is implemented in the OS and hardware. For this project, we simulated a page/frame table with page faults to help us understand the basics. With the project complete and functional we tested it against several different ratios, with different page fault algorithms, against different types of programs.

The change in the page table to frame table ratio was simply changed how much memory our function was able to work with. It represents the actual memory we have to store data in versus how much data we have to keep track of. This affect the replacement rate of our page fault algorithm heavily.

The next section is the page replacement algorithm. This occurs when the method the handles page faults executes. It means we need to swap in and out of virtual memory. It depends heavily on the aforementioned ratio and what type of program is running. Generally, a good page replacement algorithm will keep heavily used entries in virtual memory as long as possible, but no one replacement algorithm is best. Our project involved using three different algorithms to demonstrate both these facts.

The final section is what type of program was run against our page fault test cases. A program that accesses some set of data in a uniform fashion is going to have better results with a quicker page fault algorithm. But a program that heavily uses a small subsection of data will have better results with a program that keep heavily used data in virtual memory the longest.

Combinations of page: frame ratios, page fault algorithms, and program specifics will affect the running and efficient ratings of the overall system. Our experiments were designed to test these conditions and draw conclusions from the results.

Section 2: Custom Algorithm

Our implemented algorithm was rather easy. It involves first using a FIFO like approach to narrow down the ‘oldest’ entry in the frame table. However, there is a secondary check. If the data is being used to both read and write, there is statically a higher change that it will be used again relative to the other data. So, after we narrow our results, if it has both PROT\_READ and PROT\_WRITE then we select random data instead. This algorithm, commonly called second chance, is a hybrid of random and FIFO that generally produces better results.

Section 3: Nature of Program Types and Results

Random:

Random’s performance sort of categorizes it as a jack of all trades, master of none. Random has the advantage of being quick, not relying on state related material, and having uniform chance of eviction. These traits ensure that it will never lead to a look of replacements, and generally make it fairly fit for use. So why random doesn’t really fall into in pitfalls that certain orders of memory access might cause, it doesn’t really take advantage of any conditions either. So, its fitness for an algorithm doesn’t depend on the data access order. It will never really be the best, but will also likely not be the worst.

FIFO:

First in, First out. This is the methodology behind a queue. This ensures that the oldest frame entry will be evicted. It is a pretty sound algorithm. It doesn’t require much to function and doesn’t really fall into pitfalls much like random. However, when you have ordered accesses to uniform amounts of data, FIFO performs terribly. It faults nearly every time because it forces the oldest out. This creates a look in the data. So, any program that uniformly(ish) access its data reservoirs will perform poorly with FIFO. In summary: FIFO fails to prioritizes heavily used data, and fails to circumvent rotating access problems. Pretty good algorithm, but we can do better. FIFO performs best when new frame additions are used the most often.

Second Chance:

Second chance is a series use of the first two. It takes FIFO use case, which is actually depending on the state of the table. Before it evicts, it checks to see if the data in the frame is heavily used. We can check this by looking at its permission bits. If it is being written to there is a higher chance of reuse. This is really noticeable during focus. So, if these bits are set, instead of returning our frame we return a random frame. If the bits are not set, we return our initial choice. This has a higher chance of retaining heavily used memory while still keeping a general oldest out algorithm. Second chance has almost the same best performance case as FIFO, but it exceeds FIFO.

Section 4: Results

