Clayton Brutus
Jacob Ball
CS470
Dr. Hwang
Memory Management Project
4/12/17

The program can be built using the provided Makefile, and is run using:
```
./mMan <#frames> <inputFilename>
```

<u>High Level Discussion:</u>

The program allows the user to input the number of frames to use and the path to a text file which contains the reference string. A struct is used for representing the pages which contains the PID and reference number. An unordered_set is used for storing the current pages in all three replacement algorithms since it allows for O(1) lookup time. For FIFO, a linked list is used as a FIFO queue to keep track of which page to replace when a page fault occurs. For LRU, a linked list used as a dequeue which has the most recently used page in the front and the least recently used at the back (which will be replaced if a page fault occurs). For OPT, the page to be replaced is the one that will be used furthest into the future, or not at all. This is done by searching the reference string and finding the distance to the next use of each page in the current set of pages, and the page with the max distance (or won't be used at all) is that one that will be replaced. The program meets all the specifications of the project.

Simulation Results:

10 Frames:

# Processes	Algorithm	s = 98, o = 98	s = 98, o = 95	s = 95, o = 98	s = 95, o = 98
	FIFO	46	56	81	133
1	LRU	46	54	73	128
	OPT	39	47	66	118
	FIFO	175	202	321	320
10	LRU	167	200	318	313
	OPT	115	134	233	230
	FIFO	234	246	325	362
50	LRU	234	246	324	362
	OPT	190	201	292	316
	FIFO	235	250	331	372
100	LRU	235	250	330	370
	OPT	206	221	294	335

50 Frames:

# Processes	Algorithm	s = 98, o = 98	s = 98, o = 95	s = 95, o = 98	s = 95, o = 98
	FIFO	39	47	61	111
1	LRU	39	47	61	111
	OPT	39	47	61	111
	FIFO	64	97	165	202
10	LRU	61	94	154	199
	OPT	59	87	108	145
	FIFO	185	198	299	320
50	LRU	181	195	295	316
	OPT	133	154	228	254
	FIFO	206	235	302	346
100	LRU	204	235	301	345
	OPT	186	200	258	285

100 Frames:

# Processes	Algorithm	s = 98, o = 98	s = 98, o = 95	s = 95, o = 98	s = 95, o = 98
1	FIFO	39	47	61	111
	LRU	39	47	61	111
	OPT	39	47	61	111
10	FIFO	59	87	110	146
	LRU	59	87	107	146
	OPT	59	87	107	142
50	FIFO	144	163	261	289
	LRU	139	163	254	286
	OPT	133	152	223	254
100	FIFO	193	215	292	309
	LRU	193	214	289	309
	OPT	186	200	258	285

Clayton's Answers:

1. Since in reality we don't know the reference string (required for OPT), given these three options, I would use LRU since it is always at least as good as FIFO, usually better. For this hypothetical situation, where the reference string is known, I would use the OPT algorithm since it always results in the least number of page faults, which can be a significant difference compared to FIFO and LRU, and I did not find it particularly hard to implement. The most prominent example of this in the simulation results is for 10 frames and 10 processes where the OPT algorithm results in almost 100 less page faults than the other algorithms. For most of the scenarios OPT only results in a difference of 20-50 page faults, but it is still the best option.
2. I found that the most difficult part was in implementing the LRU algorithm. Since I had to use a queue but still needed to search through and manipulate objects in the queue in an efficient way.
3. I found that implementing the three algorithms was easier than I expected, but FIFO was the easiest and most straightforward since it was very easy to understand and think about what it should be doing.

4. If I had more time to do this project, I would find a way to make the OPT algorithm more efficient by improving its algorithm for finding the page furthest into the future. I would probably implement this by finding the distances to the next use of all the pages, then maintain that distance throughout the process instead of finding it every time there is a page fault.
5. I found it most surprising that, in most cases, FIFO and LRU algorithms resulted in similar results in most cases. Even when compared to OPT they were relatively close. This surprised me because I thought that OPT would be vastly superior to the others since it is the only one that looks into the future and not the past. I realized that it is not very hard to predict the future based on the past with these two simple algorithms.

<u>Jacob's Answers:</u>

1. In a realistic environment, I would choose LRU because it has typically an even amount or fewer page faults as FIFO. However, in this simulation the obvious candidate is OPT because the process string is known and so the fewest number of page faults is achieved. With a low number of processes, the algorithms do a good job at keeping the page faults close to the optimal. When there is a high probability of the next page being the same, both LRU and FIFO have a very close performance, but if the next page is more erratic, the LRU begins to have a more desirable output than FIFO. With an increased number of frames, the gap between outputs starts to decrease again.
2. The most difficult aspect would have been the data structure to which we save the references to. In our simulation we were able to put it into a vector which is pretty dynamic. In reality, the references would come in one at a time rather than in a list so our simulation is a bit inaccurate in that sense.
3. The lease difficult aspect would have been the actual replacement algorithms. After defining the structure, the algorithm can be written without thinking about what is happening after modifying the frames.
4. The simulation could be made more accurate by loading in the lines of the file one at a time, but that would prevent us from being able to compare to the Optimal page replacement.
5. I found it surprising how close the LRU and FIFO algorithms are with such a varying number of frames or processes. Part of me was expecting there to be a larger spread when load testing them.