//Jacob Darabaris
//P4
//2/11/18

Project is compiled using “clang++ -Wall -o main main.cpp” or similar.

Discussion:

The program is designed to test and compare the usefulness of the three major page replacement algorithms, First In First Out, Least Recently Used, and Optimal, for various data sets. The program is designed to run with two arguments specifying the input file and number of frames in that order. The design of the program itself is composed of six major functions, aside from the main, and one lesser function.

The program flow is as follows: Input arguments are passed into container variables and enter a function called ReferenceHandler() that takes the arguments and produces a unique page reference name greater than 0. This is done because of the global replacement scheme, and the fact that we are considering containers that hold 0 to be empty. The modified page reference is then stored in new container, and the container is passed into three functions representing the three replacement algorithms. Each function is composed of three conditional sections contained in an overarching for-loop: the first section determines whether or not the current page reference is in the frame buffer, and if it is it increments the loop. The second section determines if the frame buffer is empty, and if it is, inserts the page reference into the first opening and increments the loop and the number of faults. The third section occurs when the frame is full and the page reference is not found in any of the frames. The actual algorithm is needed here, and a separate function has been composed for each. After the algorithm completes, the frame buffer now contains the new page reference and the number of faults and loop increment.

All algorithms are nearly identical in code, and essentially just compare each index of the frames vector with each index of the pages vector and then do something to notate relevant indices in an auxiliary vector. This auxiliary vector, named counter, holds valuable information regarding how long each page has survived each pass of the overarching for-loop. Each algorithm selects its victim page from this counter vector appropriately.

Questions:

1. Which replacement strategy would you choose and why? You should consider both the end results and the effort it took to implement each strategy. Discuss what your results show about the relative merits of FIFO, LRU, and OPT for the different input files.
   LRU seems to be the obvious choice. It took about as much work as FIFO, yet is practical unlike OPT. The output table suggests that LRU is typically better as well.
2. What aspect of memory management did you find most difficult to implement?
   Determining the method of realizing the replacement algorithm was very difficult. I settled upon an auxiliary array after much trial and error.
3. What aspect of memory management did you find least difficult to implement?
   I took no issue in breaking down each algorithm into the three cases of frames empty, frame contains page, and frame does not contain page
4. What, if anything, would you change in your current design?

//Jacob Darabaris
//P4
//2/11/18

I feel the run time of my program is not efficient. I was drawn to uniformity rather than efficiency across the three algorithms, and decided against using pointers, the find() function, and general iterators. I also would have liked to combine the three algorithm function's overarching for-loops in to one, but I found it to be too confusing to design around

5. What, if anything, did you find interesting or surprising about page replacement algorithms that you did not know before doing this project?

I was surprised at its simplicity of design, although it was not so simple to do. However inefficient, I found it very doable.

Chart on next page

//Jacob Darabaris
//P4
//2/11/18

10 frames:

# processes	algorithm	s=98,o=98	s=98,o=95	s=95,o=98	s=95,o=95
1	FIFO	46	56	81	133
	LRU	46	54	73	128
	OPT	39	47	66	118
10	FIFO	175	202	321	320
	LRU	167	200	318	313
	OPT	115	134	233	230
50	FIFO	234	246	325	362
	LRU	234	246	324	362
	OPT	190	201	292	316
100	FIFO	235	250	331	372
	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=95
1	FIFO	39	47	61	111
	LRU	39	47	61	111
	OPT	39	47	61	111
10	FIFO	64	97	165	202
	LRU	61	94	154	199
	OPT	59	87	108	145
50	FIFO	185	198	299	320
	LRU	181	195	295	316
	OPT	133	154	228	254
100	FIFO	206	235	302	346
	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=95
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