**How and why you designed the Program**

The purpose of the program is to simulate a system that uses paging with virtual memory. The operating system has a limited amount of main memory space that it must allocate to each process. The program must ensure that space is dividedly equally amongst processes, in which each process needs to manage its own allocated frames in main memory to meet the incoming page requests. There was an integer variable inside each page object that a process had in its virtual memory which stored the last time the page was accessed. The Least Recently Used Algorithm initially checked the pages in the allocated page frames in the main memory for the process by comparing the integer variable values to find the least used page. This approach demanded a lot of searching and added unwanted complexity to the algorithm. After some research, an article suggested that instead of comparing the integer value of each page in the allocated page frames for each process, the program could utilize a linked list for the algorithm to cut down on some of the complexities that were a part of the initial design.

The linked list would remain the same size as the main memory allocated frames for the process and store the least recently used page at the tail of it. If a requested page isn't in the main memory frames for the process, then the program will remove the least recently used page from both the linked list and the allocated frames. The page requested would then be added to both the allocated main memory frames for process and to the head of the linked list. If a request page is already in the main memory then the program will find the page in the linked list and move it to the head of the linked list meaning that it is the most recently used page.

The clock algorithm also utilizes a linked list to represent the clock mechanism in the algorithm. The program includes a variable that is used as a clock pointer which is regularly checked to make sure that it remains within the allocated page frames for the process.

**S1 – Analysis**

The program was not able to execute the LRU and Clock page replacement algorithms with the input files as the fixed allocated frames for each process were never filled up by the incoming page requests. Even though the LRU and Clock algorithms were not executed, the files did allow me to check if the program handles page faults correctly, which it does. The program fulfils the assignment requirements for managing page faults, which detail that a process must be placed into the blocked queue when a page fault occurs and after 6 units of time the process is loaded back into the ready queue. The results from the input files match that of the results that were provided in the assignment files, implying that the program issues page faults and handles them correctly.

**S2 – Analysis**

The files allowed for the program to execute the LRU and Clock algorithm as the incoming page requests for each process causes the allocated page frames to be filled up. The results from the program match that of the expected output in the output file. The algorithms are used after the 94th unit of time up until the point that the allocated page frames in main memory are not filled up by the incoming page requests or a page that is already in main memory is requested. The results show that the LRU algorithm is more effective in handling incoming page requests for a process when all the page frames allocated to a process in the main memory are full. The Clock algorithm causes an extra page fault as a result of its clock mechanism which in turn increases the turnaround time for each process.

If the program was to use the Optimal Page Replacement Algorithm then only 12-page faults would’ve occurred. Since the LRU algorithm produces the closet number of the page faults to the Optimal Algorithm, it means that the LRU is better at handling incoming page demands for process.

**Relative performance of Algorithms**

The algorithm was tested with other inputs files and the examples provided in the lecture notes to which it produced the expected results. As far as I can tell, the algorithms work as expected. In saying that, more testing of the algorithms can be done to determine whether there are any bugs/errors.

**Personal Reflection**

When the algorithm worked as designed with the provided files, I tested them with some of the examples provided in the lecture notes. This revealed that the LRU algorithm worked as expected but the clock algorithm did not. The reason the algorithm did not work because in the 2nd provided file on-page request 11 and after, there was never a case where there was a page already in the main memory. This was easily fixed using the diagrams in the lecture notes to help me visualise what was supposed to happen in the clock algorithm. Another issue that arose when using other input files where two processes were ready at the same time within the blocked queue was that whatever process was added to the blocked queue first was added back into the ready queue first. This is an issue as it is stated in the assignment specs that if two processes become ready at the same time in the blocked queue, then the process with the smallest id number should be added first.