**Testing the Allocation Methods Comparer**

*by Sebastian Hadley*

*14/10/2021*  
  
**Introduction**

The purpose of this report is to explain the methods of testing the program underwent when being developed along with the problems encountered during the process and the ways they were fixed.  
  
**Testing Overview**

There were two main things used in the testing of the program to ensure that the results were not just correct but consistent with other inputs. The first of these two was having the program output information on everything it was doing by saying which process was running what the page they were printing was, when a fault occurred etc. The other was Sleeping the thread in the program within the main while loop to allow the output to the console to come up step by step which provided easier management and helped step through any issues as they came.

**Memory Structure**

To ensure the Memory Structure with the program was working correctly, I first checked that the information was being gotten from the files correctly. To allocate the frames for the fixed local allocation, I simply divided the total frames between them and didn’t include the remainder, and to ensure that a process wouldn’t be able to complete a page without it being in main memory I used a lot of if statements that ensured every specific condition was met.

For the Variable-global allocation the amount offrames were simply given to the process whenever it tried to page, if the number of frames available was 0 it had to remove the oldest of the frames being allocated by its pages.

**Paging Design**

The assignment specification specificies that there may be a maximum of 50 pages, to enforce this I made the program exit if a file with more than 50 pages was input, with a message explaining what happened.

To implement the FIFO policy I used the Queue data structure as it enforces FIFO by design, however the round robin scheduling had to be done manually.

**Fault Handling**

To debug the fault handling I used a lot of output to the command line, and an array of booleans with a size of the number of processes detailing if a specific index of the processes was currently blocked. This array was given information whenever a fault occurred and a function to check whether the current time was equal to or greater than the time in which its last page fault would re open. Which determined if the index of that array should be made open again.

**Scheduling**

To ensure the scheduling was accurate I would output the index of the processes that the main loop was currently on to see if the processes were being done in the right order. To test that the timing was all right and that the results werent just the same in finishing, I had the program output specific details about everything it did and at what times which allowed me to follow the logic of the program and sure accurate results.

**Algorithm Comparison**

The two algorithms were implemented very similarly with only a few details changing and those being related to the processes paging and memory, however the Variable-Global Replacement algorithm was far easier to implement, which may have just been because it was the latter of the two I did. The Variable-Global Replacement algorithm aslo performed noticeably better around the bored with the worst case scenario being the same results as the fixed, however when there were more pages in each process the variable algorithm reduced waste from other processes and allowed a much quicker result.

**Conclusion**

Overall the program did not have many major hurdles in design meaning there was not an excessive amount of debugging or testing performed, however the methods used provided easier comprehension of the problems the program had and made it much easier to target them for repair.