**Mid-Term Reflection**

During this half-semester as part of my degree in Computer Science, I have already learnt so much about the principles of Operating Systems and their importance whilst making design decisions when creating an application.

I believe it is important to understand the background and history of any topic as has been discussed during the lectures in week 0 on the Overview of Operating Systems. We have discussed the different types of operating systems such as layered, microkernel and hybrid. It is very important to know and understand the differences in structure and the benefits and drawbacks of each type of operating systems, especially if you want to work with low-level languages, such as C, or want to work for a company that designs and maintains operating systems.

I believe that two of the most important and valuable topics that we have discussed so far are on Processes and Threads, discussed during week 1, and Process Scheduling, discussed during week 2. These two topics underpin my understanding of concurrent programming techniques. To be able to build an efficient operating system you need to be able to understand the way threads interact in a multi-threaded program and the different scheduling algorithms, such as FCFS, RR and SRT, to efficiently complete as many processes as possible in the shortest possible time.

As computers have become more powerful and the number of tasks that can be run at any one time has increased, so has the importance and relevance of understanding and being able to implement concurrent solutions. In addition, concurrent programs are of paramount importance in the financial sector as, for example, in the London Stock Exchange alone there are a million trades per day amounting to an average of 11 trades per second. If the software that handles these transactions was not concurrent, fraudulent transactions could easily be completed and slip under the radar due to the sheer volume of money transfer.

While benefits are in the plentiful when it comes to concurrency and multi-programming, deadlocks are a big problem. An example of a deadlock is when you try and rename a file, for example, a Word file, that is open from File Explorer on Windows. You will get an error saying that the file is in use. This deadlock could easily have made the application crash, but, a ‘try catch’ block could be used to catch this exception and handle it so the program doesn’t crash.