Reflective Report

By: Alasdair Smith
MMU ID: 14061121

BSc Computer Science

Computer Networks and Operating Systems

Upon reflection of this semester's teachings in Computer Networks and Operating Systems I have had the opportunity to learn about some very fundamental, yet critical, aspects of the of operating system design. The topics that were covered included theory on Files, Processes, Concurrency, Deadlock, Scheduling, Memory/Virtual Memory, I/O and Security. Whilst studying operating systems this semester has taught me much about the internal theory supporting the way different subsystems operate in a modern OS, it was clear to me that a modern OS's greatest struggle is with the diversity of devices available, concurrency and multi-programming.

Design choices to enable multi-programming and concurrency can be seen in many areas of an OS. The most obvious example of this are through an OS's scheduling of processes. Modern systems make extensive use of multiple processes to improve responsiveness/speed of execution/efficiency/availability of the system. Less obvious examples of design choices to enable multiprogramming might be seen in subsystems such as the Memory Management Subsystem. In order to enable multiple processes to operate in the same address space, the MMS makes each process perceive the entirety of the address space as solely in use by itself. This is done through the use of a Virtualized Address Space. All of these features (and more) have been more and more frequently incorporated into common everyday state of computing since the advent of multi-core processors.

Today's diverse range of processors capable of different levels of concurrent threads of execution points towards an increasing demand for operating systems to be capable of handling multi-everything. With multiple CPUs capable of servicing multiple interrupts from multiple devices, today's OSs are being pushed further and further towards the limits of our ability to manage concurrent execution in a multi-programmable environment.

Whilst multiprogramming and concurrency is a commonplace feature of present day computing I believe it is likely to be a primary area of development in the future too. At least, until a newer way to compute arises in the public domain (e.g Quantum Computation). Until, this newer form of computation takes centre stage, I believe the future of faster and more efficient computing still lies in the miniaturisation and clustering of more processors on a single chip or the distribution of additional independent processors to hardware devices. For this reason, understanding the fundamental principles of how multiprogramming/concurrency can be managed and enabled is critical. However, just because multiprogramming has become part of the norm, it doesn't mean that fair and efficient concurrency is an easy task to achieve. This is an additional reason that I believe there is still progress to be made in this field.

Deadlock and resource starvation is still an unavoidable feature of concurrent execution of processes/hardware. Ultimately, in any system that is attempting to operate concurrently (at either a hardware or software level) there must be a feature of the system that manages this concurrency to ensure fair use of resources. From a processes' perspective this 'feature' is the Scheduler. With any Scheduler comes the potential for resource starvation and deadlock. In an attempt to avoid deadlock it becomes more likely for resource starvation to occur. Conversely, without any attempt to avoid deadlock the system runs a guessing game as to whether deadlock will occur in the current iteration.
It is my personal opinion that the root of the problem lies in the strictly algorithmic nature of current scheduling methods. I believe that a system capable of near-perfect concurrency/multiprogramming would be managed by an AI (Artificial Intelligence) based solution. An OS which is capable of reacting intuitively to formulate a plan of how to fairly and effectively distribute resources, would be far more dynamic in its response to (for example) the submission of new process. Regardless, a truly perfect multiprogramming capable operating system would require a management solution capable of near precognition. A solution, perhaps, best left to Quantum Computation.