CMPSCI 377 Operating Systems **Fall 2013**

Lecture 1: September 4

Lecturer: Prashant Shenoy *Scribe: Armand Halbert*

1.1 General information

This class is taught by professor Prashant Shenoy, at CMPSCI 142, every Mondays and Wednesdays from 9.00am to 10.20am. For all kind of information related to the course (syllabus, projects, etc), please visit `http://lass.cs.umass.edu/~shenoy/courses/377/`.

Although the class name *is* Operating Systems, this course will deal mostly with large scale computer systems. The main question to be studied is *how to build correct, reliable and high performance computer systems*. In order to answer this question, problems such as memory management, concurrency, scheduling, etc, will be studied.

1.1.1 Getting help

Prof. Shenoy has office hours on Mondays and Wednesdays from 9.00am to 10.20am., at his office (CS336). The TA has office hours on Tuesdays and Thursdays from 11.00am to 12:00pm, at his office (Room 224, for the moment).

In addition to office hours, Discussion Sections will be lead by the TA, on Fridays from 1.25pm to 2.15pm at i142. Attendance is very important since sample test problems will be discussed, and the main concepts presented during the last couple of classes will be reinforced. Lecture notes will also be available (on Prof. Shenoy's course website). The textbook for this class is *Operating System Concepts* (Silberschatz et all), 7th Edition OR 8th Edition. There will be a couple of copies of the text book available in the library. Other useful textbooks include *Operating Systems In Depth* (T. Doeppner, Wiler) and *Modern Operating Systems* (Tannenbaum), but they are not necessary to purchase.

1.1.2 Grading

Student's grades will depend both on exams (3 exams, 40% of the final grade in total for two midterms and a final) and on programming assignments (45% of the final grade). Homeworks and some in-class assignments will constitute the remaining 15%. Prof. Shenoy adopts a very strict late policy, so please keep that in mind when deciding when to start working on your assignments. All the projects can be done in groups of 2. You are welcome to do the projects alone too. See the group policies in the syllabus. One of the projects may use Nachos (a java based toy operating system). The other programming assignments will be more standalone and not based on pre-written code. Assignments will be submitted electronically using Moodle `moodle.umass.edu`.

Do not cheat! An automatic system for finding cheaters will be used, and you *will* be caught.

1.2 Introduction to Operating Systems

The term **Operating System** (OS) is often misused. It is common, for example, for people to speak of an OS when they are in fact referring to an OS *and* to a set of additional applications (eg: on Windows, Notepad, Windows Explorer, etc).

The traditional view of the area, however, defines an OS in a different way. The OS can be seen as the layer that stands between the user-level applications and the hardware. Its main goal is to hide the complexity of the hardware from the applications. The important concept here is *abstraction*: an OS abstracts architectural details, giving programs the illusion of existing in a "homogeneous" environment. The OS effectively makes programs believe that they live in a virtual machine with large amounts of memory, a dedicated processor, and so on. This results in an environment that is easier to program for. It is also the OS's function to deal with managing the computer's resourcers (eg: the OS decides which process to run, when, for how long, etc). The OS also provides services such as the file system, networking, and the coordination of multiple applications, and help protect a program's memory from others'. The downside to what the OS provides is an overhead that is added on top of the raw hardware. The operating system sits between the hardware and the user-level applications meaning that all interactions have to go through the OS. These interactions will run slower than if there was no OS at all. The more functionality the OS has, the more overhead that is introduced. As whole, the goal of designing OS is make computer a convenient system to use for the user while not sacrificing too much efficiency.

1.2.1 A brief history of Operating Systems

Operating Systems evolved tremendously in the last few decades. In fact, it would probably be impossible for someone coming directly from the 60s to perceive any resemblance between modern OSs and the OSs from those days.

The first approach for building Operating Systems, taken during the 40s and 60s, was to allow *only one user and one task at a time*. Users had to wait for a task to be finished before they could specify another task, or even interact with the computer. In other words, not only OSs were monouser and monotask, there was no overlapping between computation and IO.

The next step in the development of OSs was to allow batch processing. Now, multiple "jobs" could be executed in a batch mode, such that a program was loaded, executed, output was generated, and then the cycle restarted with the next job. Although in this type of processing there was still no interference/communication between programs, some type of protection (from poorly or maliciously written programs, for instance) was clearly needed.

Overlap between IO and computation was the next obvious problem to be addressed. Of course, this new feature brought with itself a series of new challenges, such as the need for buffers, interrupt handling, etc.

Although the OSs from this time allowed users to interact with the computer while jobs were being processed, only one task at a time was permited. Multiprogramming solved this, and it was a task of Operating System to manage the interactions between the programs (eg: which jobs to run at each time, how to protect a program's memory from others, etc). All these were complex issues that effectively lead to OS failures in the old days (such as Multics). Eventually, these problems brought up the attention for the need to design OSs in a scientific manner.

During the 70s, hardware became cheap, but humans (operators, users) were expensive. During this decade, interaction was done via terminals, in which a user could send commands to a mainframe. This was the Unix era. Response time and thrashing became problems to be dealt with; OSs started to treat programs and data in a more homogeneous way.

During the 80s, hardware became even cheaper. It was then that PCs became widespread, and simple OSs, such as DOS and MacOS, were used. DOS, for example, was so simple that it didn't have any multiprogramming features. The first GUIs were added to the OS.

¿From the 90s on (until today), hardware became *even* cheaper. Processing demands keep increasing since then, and "real" OSs, such as WindowsNT, MacOSX and Linux, finally became available for PCs. New areas currently developing include cloud computing, embedded systems (how small can the kernel get), and large distributed computing systems.

If there is a lesson to be learnt from all this history, it is that it's very hard to outline trends for the future. After all, computers advanced *9 orders of magnitude* (in terms of speed, size, price) in the last 50 years. Moore's law also seem to be running out of steam, mainly due to fundamental physics limits. However, we can do some guesses on what to expect in the next few years: more cores, unreliable memory, serious power/heat constraints, trading off computer power for reliability and new application spaces like *Web*, *Grid*, *Cloud*, etc.