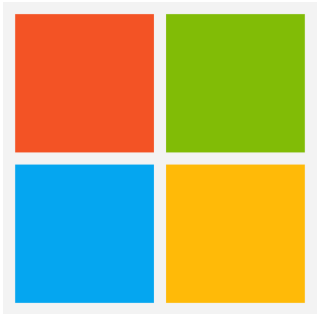
Introduction to CS211

CS211: Programming and Operating Systems

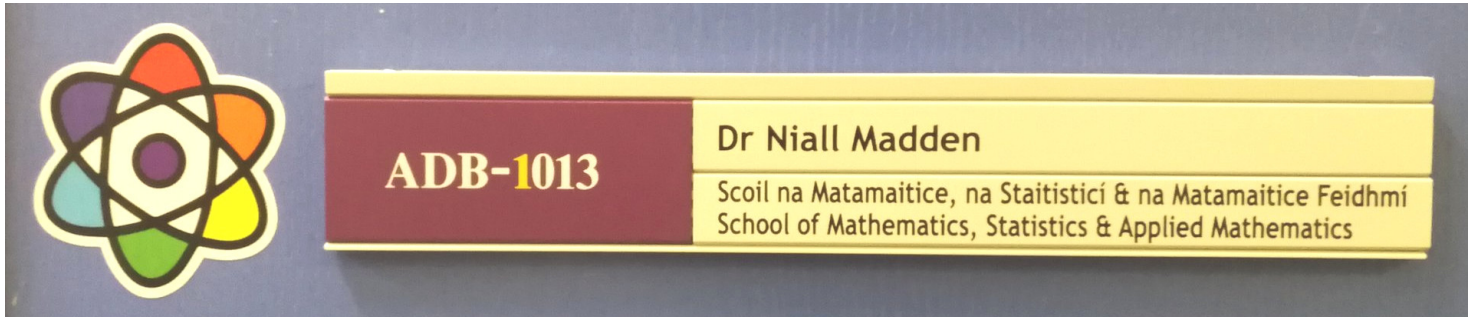
Niall Madden

Wednesday and Thursday, 15+16 Jan, 2020


Outline

PDP-11/40, a 16-bit minicomputers sold by Digital Equipment Corporation (DEC) from 1970 into the 1990s. Source: Wikipedia



Lecturer: Dr Niall Madden, School Mathematics, Statistics and Applied Mathematics (he/him).

Where I am: Office: AdB-1013, Arás de Brún.

Contact me: Email: Niall.Madden@NUIGalway.ie
Phone (091 49) 3803.

And you are...

2BMS1: Mathematical Science

2BPT1: Physics

2BS1: Science

OS Books

Operating Systems: Three Easy Pieces, by Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau. **This is our primary OS text**. It is free; individual chapters can be downloaded in PDF form from `http://pages.cs.wisc.edu/~remzi/OSTEP/` Other good OS references include ***Operating Systems*** by O'Gorman, and ***Applied Operating Systems Concepts***, by Silberschatz, Galvin and Gagne.

C Programming books

C Programming - a modern approach, by King,
A Book on C by Kelley and Pohl.

We'll use the following systems of communication in CS211:

`http://www.maths.nuigalway.ie/~niall/CS211` and
`http://blackboard.nuigalway.ie`

Blackboard will be used for announcements, emails, submitting assignments, and reporting grades.

At `http://www.maths.nuigalway.ie/~niall/CS211` you will find:

Notes These lecture slides, posted a few hours before the lecture. This means that you don't have to spend the whole class writing. However, the notes are *incomplete* and you still have to attend lectures. For example, solutions to problems are done on the white board and so are not in the on-line notes.

Problem Sets Lab assignments and exam-type questions.

Links to other material of interest (I hope).

Lectures

Wednesday	15:00–15:50	AC202
Thursday	13:00–13:50	AC204

Labs

Computer labs are a *very* important part of this course

There will (roughly) 5 assignments to be completed during the semester. All these will be graded and will contribute 40% to your final CS211 grade.

There will be a two hour supervised lab time each week, starting Week 3.

Overview of CS211

Lectures and Labs

	Mon	Tue	Wed	Thu	Fri
9 – 10					
10 – 11					
11 – 12					
12 – 1					
1 – 2					
2 – 3					
3 – 4					
4 – 5					
5 – 6					

Course Content

The two most important things any student of computing must learn are:

(i) How computers work and (ii) How to program

CS211 addresses both of these.

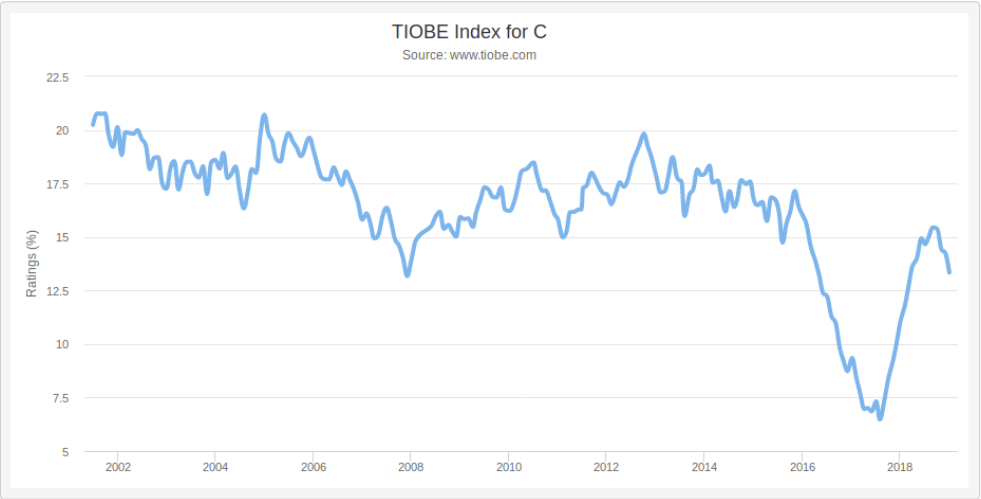
A computer's ***Operating System*** is the fundamental piece of software that allows it to function. Understanding computers requires in knowledge of OS. So we'll

- Study the theory of operating systems.
- Investigates the similarities of and differences between certain operating systems (old and new).

- Introduction: ***What is an operating system***?
- Computer and Operating System structure.
- The three “easy” pieces:
  1. **Virtualization**,
  2. **Concurrency** and
  3. **Persistence**.
- **Processes** (i.e., *a program in execution*), threads, and CPU scheduling.
- Memory management and Virtual Memory
- Process synchronisation and deadlock
- Mass Storage and File Systems

The most important development in (computer?) science during the last century was the development of high-level programming languages, and compilers for them. Among the most important/popular are (as rated by TIOBE)

1. Java
2. C
3. Python
4. C++
5. Visual BASIC .NET



Also (in order) Javascript (6); C# (7); PHP (8); SQL (9); Objective-C (10); MATLAB (11); R (12); Perl (13); Assembly Language (14); Swift (15)

The particular importance/popularity of C is due to the fact that most mainstream operating systems (desktop and mobile) are written in C.

This also means that it's a very language to complement a course on Operating Systems.

What is an OS?

We can divide a computer's software into two categories:

User software: Applications such as media players, email clients, word processors, games, etc.

System software: e.g., memory managers, disk managers, network daemons, etc.

In this scheme, the **Operating System** (OS) is the most fundamental piece of system software.

The OS is dedicated to make the computer ***efficient*** and ***usable***.

By ***usable*** we don't just mean that it has a reasonable useful interface but also that it provides

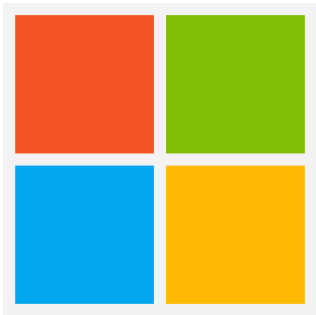
- Libraries and routines so that programmers don't need to write elementary functions or need to know assembly language.
- Security systems so that data is not destroyed intentionally or by accident.

Different ways one may characterise an operating system:

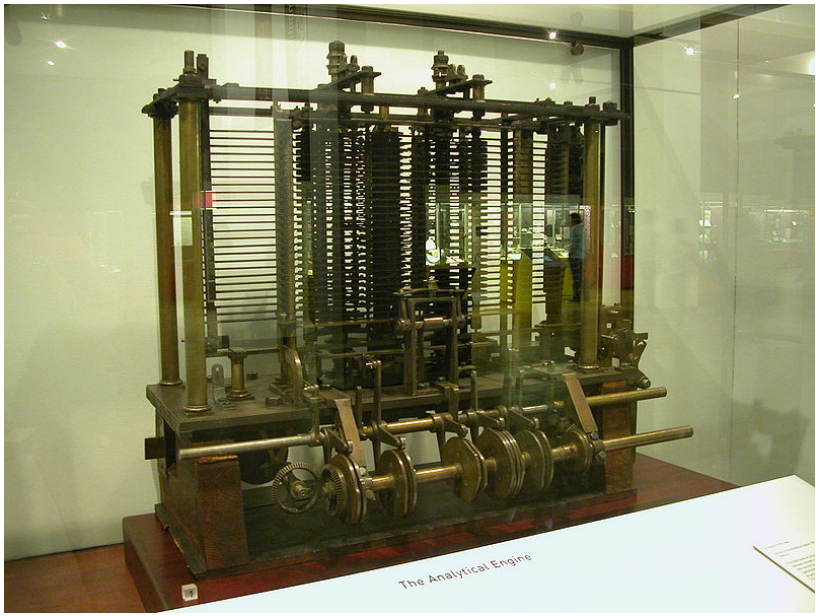
- After the computer is booted, it is the first program to run. All other programs are run by it.
- It provides an interface between users and the computer's hardware.
- It is a resource allocator – it gives access to the CPU, RAM, mass storage, etc., and makes efficient use of resources.
- It controls the execution of (all) other programs.
- **It is running at all times** – this part of the Operating System is often called the *kernel*.

Examples of Operating Systems include:

- Microsoft: Windows 10 (prior to that: Windows 7 and 8; Vista; XP, NT), etc.
- Apple: MacOS, iOS, etc.
- GNU/Linux, e.g., Ubuntu, Fedora, etc.
- Google: Chrome OS, Andriod (which uses a Linux Kernel)
- Other UNIX-type operating systems

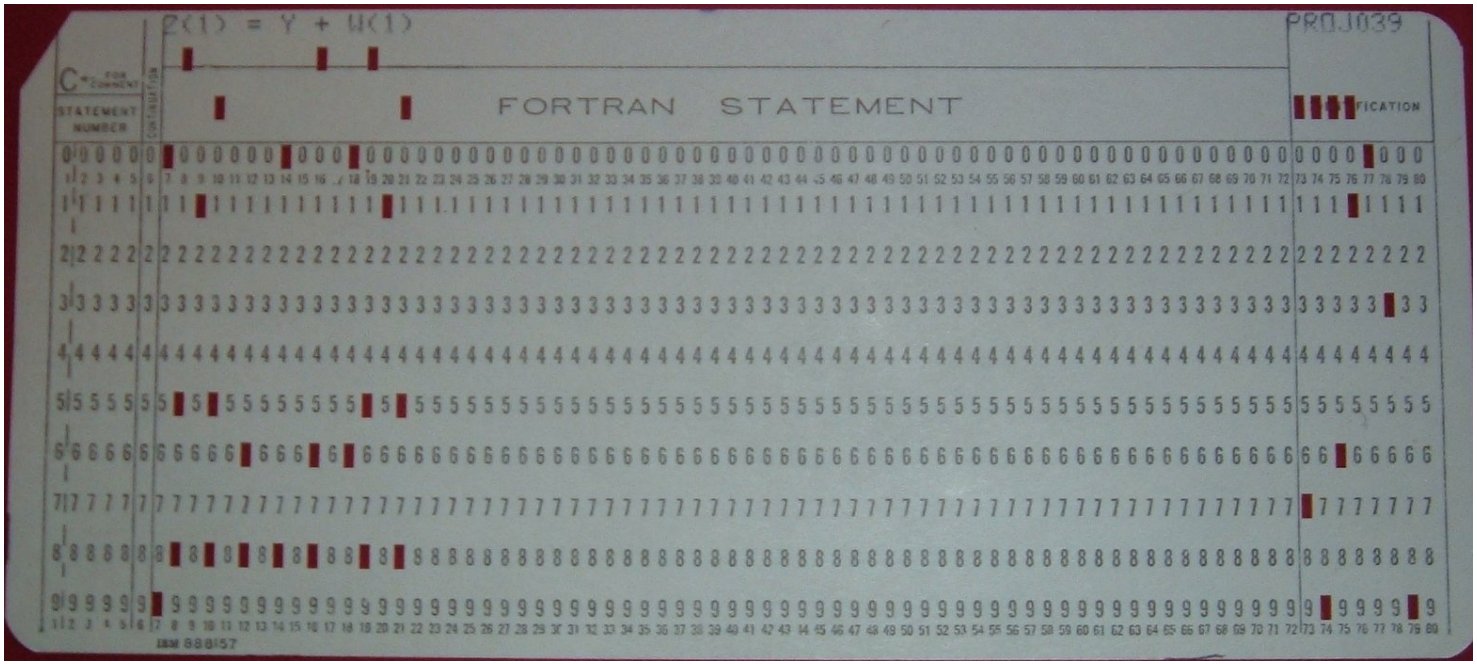
See Section 2.6 of OSTEP:
http://pages.cs.wisc.edu/~remzi/OSTEP/intro.pdf



Trial model of a part of the Analytical Engine, built by Charles Babbage, as displayed at the Science Museum, London (source: Wikipedia).

- Batch Systems
- Multiprogramming Systems
- Multitasking Systems
- Personal Computers
- Multiprocessor Systems
- Distributed Systems

Processor speeds measured in kHz.
Programs were stored on cards – a fast card reader could handle up to 20 cards per second!



Programs submitted by hand to the (human!) operator who put complementary programs into separate "***batches***" and loaded them.

The OS took the form of the ***resident monitor*** – consisting of

Card Interpreter – read and carry out instructions on cards.

Loader – place system and application programs into memory.

Device Drivers – manage the Input and Output (I/O) devices.

Biggest problem with Batch Systems was slow performance: I/O and CPU operations could not overlap; the CPU was much faster than the card reader but had to wait for instructions to be read in.

The solution is to over-lap job execution with I/O. The OS would simultaneously

- Read the next jobs from card reader onto a disk
- Execute the current job
- Output the results of the previous job to the printer.

With the advancement of disk technology, ***job scheduling*** became possible. In particular, the concept of **Multiprogramming** was introduced.

Key IDEA: Processors run faster than I/O devices.
Primary GOAL: Keep the processor in use at all times.

- Several jobs are selected from a job pool and are loaded into memory.
- The OS selects one and starts to executing it.
- If that job is delayed (waiting for I/O) the CPU selects a second one and runs that instead.
  When that one is delayed, a third is selected and that is executed.
- Eventually, by the time the OS returns to the first job it should have completed the I/O by then.

Multiprogramming requires the OS have certain components:

Job scheduling: must decide which jobs are selected from the pool and loaded into memory together.

CPU scheduling: if job *X* is delayed, must have a mechanism for selecting which job replaces it.

Memory management: programmers did not have to use physical locations in code.

Memory and File protection: make sure that different jobs don't try to write to the same location in memory or the on disk drive

In order to make key-board driven, multi-user computers feasible the idea had to be extended further, if only because typing is much slower that even card reading. This leads to **Multitasking**.

Key IDEA: for interactive computing the CPU must be able to switch between jobs quickly

Primary GOAL: to give the impression to users/programs that they have exclusive use of the computer

The CPU gives a small amount of attention to each available job. This is called ***Time Sharing***. This gives the impression that all the jobs are running at the same time.

Instead of submitting programs on cards to the operator, several users (2, 10, 100, . . .) interact with the computer at the same time, using key-board/mouse/whatever.

Typing is slow: about 1 key-press every 0.1s. Between key-presses, the CPU has time to perform other tasks such as reading another user's key-strokes.

A Time-sharing/multitasking system must have the following facilities:

- multiprogramming and CPU scheduling
- more that one job executing at a given time – ***processes***
- Memory management and virtual memory
- File systems and Disk management
- process synchronisation and communication

Most modern Operating Systems are multitasking.

The systems mentioned above had many concepts in common. One crucial one was that ***Computer time was expensive and People's time was cheap***.

Also known as ***micro computers***, PCs were cheap compared to mainframes/minicomputers – could afford to have a ratio of

One user to one Computer

Because user's time became more valuable than the computers, the operating systems were designed primarily to be easy to use.

Early PC makers included Apple and IBM.

Because only one user was using a computer at any given time, the OS could be much simpler than on the big systems:

- no multiprogramming
- no security
- no file/memory protection

This was to change because

- small computers became faster – comparable to speed on mini computers.
- use of networking requires security.
- Multitasking Operating Systems were ported to micro-computers. In particular several Unix systems were re-written for Intel (x086) architecture. E.g., Free BSD, SCO Unix, Linux.

So other OSs were developed for personal computer but with features of multitasking operating systems.

BASIC IDEA: to increase a computer's speed, increase the number of CPUs that it has access to

As limits of processor speeds are reached, high performance computing (where emphasis is on performing as many calculations as possible in a short amount of time) looked to increasing the number of CPUs in a computer.

Multiprocessor Systems vary from dual-processor desk-top systems to ***massively parallel computers*** with 1000's of CPUs.

Advantages:

- Ten 1 Ghz chips might be cheaper that one 10 Ghz chip.
- Faster absolute speeds can be achieved
- Reliability – if one processor fails, others will keep running.

There are two basic types of multiprocessing:

Symmetric multiprocessing (SMP): common on smaller systems, most operating systems have a facility for SMP. All processors are equal (peers) and run a copy of the OS.

Asymmetric multiprocessing: master-slave model.

Note: in general if you can get w units of work done on one processor and you have n processors, you will get less than $w \times n$ units done with them.

Real-time systems are employed when there are very rigid time constraints – when the system *must* complete its designated task in a given time frame. Examples include

- Robotics Control
- Medical Equipment
- Domestic Appliances
- Fuel injection systems

The primary design goal is no to maximise efficiency (through-put), but to guarantee time-to-completion.

There are two basic types of real-time system:

1. **Hard real-time:** no secondary storage (eg., hard-drive), no time-sharing or multiprogramming, data is usually stored in ROM.
   Operating System lacks components that we study in this course.
2. **Soft real-time:** Used in real-time systems that require some advanced OS features, e.g., multimedia systems, space exploration, etc.
   When a time-critical task must be completed, it gets priority over other operations until it completes.

(See section 48 of the text book)

Distributed Systems are familiar to us, particularly as services (Facebook, Netflix, Amazon, Dropbox, Blackboard, and just about anything else that you can think of).

From OSTEP

"When you use one of these, you are not just interacting with a single machine, however; behind the scenes, these complex services are built from a large collection (i.e., thousands) of machines, each of which cooperate to provide the particular service of the site. Thus, it should be clear what makes studying distributed systems interesting".

Distributed systems, and, thus, their operating systems, must provide

- Shared Resources (data-bases, file systems, ...)
- Load sharing – work can be distributed over the different computers so that no one is over-loaded.
- Reliability – one computer fails, the system will continue to function.

Examples of distributed systems:

1. Large information servers, e.g., Google. Need to be fast and reliable.
2. Intensive Computational Work. E.g., rendering for animated films are often done using on clusters of PCs running (Beowolf) Linux.
3. Remote computational Systems, e.g., *SETI*: though this is more an example of distributed computer rather than a distributed operating system.

KEY IDEA: Distributed Systems are similar to Parallel Systems, but use several/many computers on one network rather than several/many processors in one computer

Exercises

Exercise (1.1)

Name 5 operating systems. Who developed them, and when? What are their primary uses (e.g., on mobile devices, or desktop computer systems, or clusters, etc)?

Exercise (1.2)

State the primary goals of multiprogramming systems and multitasking systems. What features of an operating system are common to both multiprogramming or multitasking systems? Which are features of just one of these?