CS 167: Operating Systems
Staff

- Head TA
  - Isaac Davis
- UTAs
  - Jonathan Lister
  - Max Luzuriaga
  - Ben Murphy
  - Jake Saferstein
  - Di Yang Shi
Those taking 169 do the 167 Uthreads and Mthreads assignments, but not the others.
Workload: 169

- One large program with five milestones, resulting in a working operating system
  - Processes
  - Drivers
  - Virtual file system
  - S5 file system
  - Virtual memory
Skills Needed

- Ability to write and debug largeish programs in C with POSIX threads
  - CS 33
- Basic computer architecture
  - CS 33
What are Operating Systems?

- Possible definitions:
  - the code that \{Microsoft, Apple, Linus, Google\} provides
  - the code that you didn’t write
  - the code that runs in privileged mode
  - the code that makes things work
  - the code that makes things crash
  - etc.
The purpose of an operating system is to make the underlying hardware useful. Rather than forcing programmers to deal with the complicated details of such hardware, more convenient idealized abstractions of the hardware are devised, implemented, and made available, replacing the direct interface to the hardware. Thus, as discussed in upcoming slides, we deal with files rather than disks, virtual memory rather than real memory, threads rather than processors, etc.

However, in operating systems, even more so than in “ordinary” applications we must cope with the concerns listed in the slide. Performance is a huge issue. Some portions of the OS might be invoked often, say thousands of times per second. Reducing the number of instructions used is extremely important; we must also be concerned with how effectively the code uses the processor’s caches. Use of memory, both primary and secondary, is also important. If too much memory is used, the operating system might not “fit” in the available resources. Furthermore, too much memory use may require shuffling of code and data between primary and secondary storage, thus wasting time.

Security is increasingly a problem. We must protect the operating system, the applications running on it, and data maintained by it, from attack. (This is clearly not a solved problem!) An important concept is that of isolation: we want independent computations to be isolated from all others.

We must also provide various forms of sharing: not only must different applications coexist on a computer at once, but different users might be using the same computer at the same time. Our operating system must be tolerant of failures: bugs in one application must not bring down another. Problems with hardware should not cause catastrophic failure.

Finally, for an OS to be successful, people must want to use it. It’s one thing to build an operating system that does an excellent job dealing with everything mentioned above, but if none of our existing applications can be made to run on it and we have to write
everything from scratch, no one is going to want to use it.
As we've just mentioned, one of the major purposes of an operating system is to give its users an easy-to-use abstraction of a fairly complicated reality. A typical computer system consists of a number of disk drives (which the operating system accesses through specialized hardware called controllers), a network interface (also accessed via a controller), some sort of display/keyboard/mouse combination, and some amount of primary storage.
The typical user has no desire to deal with the complexities of the underlying hardware, preferring instead to deal with abstractions and depend upon the operating system to map these abstractions into reality. For example, the user may have information that needs to be accessed and perhaps updated over long periods of time. While such information may well be stored on disk, the user probably doesn’t care how.

The user’s computation will certainly use temporary storage that need not exist longer than the computation, but should be quickly accessible during the lifetime of the computation. The user’s computation will need a “compute engine” to carry out the computation; in some cases it will need multiple compute engines. Finally, the user may need to communicate with other users or computations on other computers, and may need to access information stored elsewhere.

Designing the most suitable abstractions for an operating system to present to its users is a subject of continuing research. Soon we examine the abstractions provided by a particular operating system, Unix, that is in widespread use on computer workstations and larger machines. Later in the course we discuss alternate abstractions.

Unix provides a relatively simple collection of abstractions (though many argue that Unix has become much too complicated). In particular, permanent storage is represented as files. We think of temporary storage as being built from the abstraction of primary memory, known as virtual memory. Each computation has its own private virtual memory—computations are known in Unix as processes. Traditionally, each process is embodied as a single abstract compute engine, but more recently the concept of multithreaded processes has been introduced. In this model, each process may contain multiple compute engines, or threads, each capable of independent computation. Communication is not a very well developed concept in Unix systems. The general idea is that independent processes can transfer data back and forth among one another. We explore this idea and newer approaches to handling communication in the last section of
the course.
File systems provide a simple abstraction of permanent storage, i.e., storage that doesn’t go away when your program terminates or you shut down the computer. A user accesses information, represented as files, and the operating system is responsible for retrieving and storing this information. We expect files to last a long time and we associate names (as opposed to addresses) with files.
Among the issues in implementing the notion of files are those mentioned above.
A program is an abstraction that we view as consisting of data and code. Somehow we've got to build programs from the pieces of hardware available to us, in particular memory, disks, and processors.
So that multiple programs, each with their own private libraries and each accessing shared libraries, can coexist without destroying each other, we must have some means of protecting one program’s memory from another’s. Furthermore, we must protect the operating system from being inadvertently or maliciously attacked by programs.

Using various hardware features, each program is prevented from accessing private portions of other programs. Shared portions can be accessed, but only in a read-only fashion. When the operating system is executing, it can access the user programs. But when user programs are executing, they can access the operating system only to do prescribed requests.
Another memory-sharing issue arises when the programs that are to share memory are too big to fit in all at once (or perhaps even individually).
One popular approach for dealing with memory-protection and -fitting issues is to employ virtual memory. A program executes in a “virtual address space,” which is implemented partially on real memory and partially on disks. A facility supported as a joint effort of hardware and the operating-system kernel “maps” each program’s references of virtual memory into real memory and disk space. Since the operating system controls this mapping, it determines who can access what.
Computers are typically multiplexed among a number of users. Multiplexing is a concept that is well over thirty years old—many users share a computer by dividing its processing time among one another. It’s often known as multitasking.

Another equally old, if not older, concept is the notion of concurrency within a computation. This is the ability of one computation to utilize multiple logical “compute engines,” or threads. Concurrency means that multiple threads are in progress at one time; on a computer that employs only a single real processor, the execution of these threads is multiplexed over time.

Why do this?

• Optimize the use of computers
  — programs typically compute, perform I/O, compute, perform I/O …
  — when one program is performing I/O, another should be computing

• Optimize the use of people
  — want several applications active at once
    – interactive applications
      · editing
      · drawing
    – background applications
      · mail monitor
      · printing
      · file transfer
Parallelism means that multiple threads are executing simultaneously: parallelism requires multiple processors.

In this course we make this distinction between parallelism and concurrency; note, however, that not everyone distinguishes between these terms in this way.
1960s OS Issues

- Multiprogramming
- Time sharing
- Software complexity
- Security
2010s OS Issues

- Multiprogramming
  - not just one computer, but server farms
- Time sharing
  - voice, video, sound, etc.
- Software complexity
  - a much bigger problem than could possibly be imagined in the ’60s
- Security
  - ditto
In the Beginning …

• There was hardware
  – processor
  – storage
  – card reader
  – tape drive
  – drum
• And not much else
  – no operating system
  – no libraries
  – no compilers

See http://www-03.ibm.com/ibm/history/exhibits/701/701_intro.html for a description and photos of the IBM 701 (announced in April, 1952). This was clearly not the first computer, but it was the first computer to have something resembling an operating system (introduced somewhat later).
Early computing was roughly split between two sorts of activities: commercial data processing and scientific computing. The former was characterized by much data processing, less computation, and “production” activities such as doing a payroll. The latter involved more computation and, generally, less data processing.
The IBM 701 is, reportedly, the first computer to have an operating system (designed by General Motors (not IBM) and called the Input/Output System). The man on the right is computer pioneer Herb Grosch, in whose hotel room the notion of an operating system was reportedly hashed out in 1953. He later served as president of ACM. The man on the left is an actor who later served as president of a large country. The picture is from http://www.futunesco.org/musi/museu8.html (the link no longer works), the web site of FWT UNESCO Computer Museum, Padova, Italy, and was taken in 1956.
The photo is from http://www-03.ibm.com/ibm/history/exhibits/650/650_ph10.html and is of the IBM 650 used by the FAA for air traffic control, though it was primarily used for commercial data processing. The IBM 650 was announced in 1953 and was the most popular computer in the 1950s. It was last made in 1962 and support was terminated in 1969. It apparently had nothing resembling an operating system.
Programming Without an OS

- Assemble all software into a deck of punched cards
- Get 15-minute computer slot
  1) pay $75 ($678.18 in 2017 dollars)
  2) mount tapes containing data
  3) read cards into computer
  4) run program
     - it probably crashes
  5) output (possibly a dump) goes to printer
- Steps 1, 2, 3, and 5 take 10 minutes
  - leaving 5 minutes for step 4!

The inflation calculation is from http://www.dollartimes.com/calculators/inflation.htm and assumes the $75 was spent in 1953. The notion of 15-minute slots and the $300/hour cost are from (Ryckman 1983).
Enter the OS …

• Group jobs into batches
• Setup done for all collectively
• Software doing this called *Input/Output System*
  – the first operating system
In the 1960s, commercial data processing and scientific computing continued to be important. But the decade saw the introduction of time sharing as a means for making more productive use of an expensive computer, as well as the introduction of relatively cheap minicomputers initially used in laboratories.
This drawing of a “typical Atlas computer installation” is from http://www.chilton-computing.org.uk/acl/technology/atlas/p002.htm (where there is also a diagram labeling what each of the components in the slide are). Its operating system supported multiprogramming and seems to be the first major step in OS design and implementation after the Input/Output System of the IBM 701. It’s most famous for including the first implementation of virtual memory, which they called a “single-level store.” It was a collaboration between the University of Manchester (UK) and Ferranti Ltd. It was officially commissioned in 1962, though it seems to have been in operation starting in 1961.
This photo, of an IBM 7094, is from http://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP7094.html. It was modified at MIT and used as the hardware platform for CTSS (compatible time-sharing system), which seems to be the first time-sharing system.
The photo is of an IBM System/360 and is from http://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_intro2.html. The IBM System/360 comprised a range of computers. It was most successful in the realm of commercial data processing.
This photo, of a Honeywell 6180, is from http://www.multicians.org/multics-stories.html. This machine was the successor to the GE 645 (Honeywell had purchased GE’s computer business), on which Multics first ran. The 6180 incorporated many improvements and hosted the second generation of Multics.
The PDP-8, introduced in 1965 (the photo is from http://www.pdp8online.com/pdp8i/pdp8i.shtml), was the first “minicomputer” and was cheap enough to be used in small laboratories. Its manufacturer (DEC: Digital Equipment Corporation) as well as other companies produced numerous kinds of such minicomputers and an even greater number of operating systems for them. This OS development continued well into the 1970s.

A discussion of the operating systems written for the PDP-8 can be found at http://www.faqs.org/faqs/dec-faq/pdp8/section-10.html.
This photo is of Dennis Ritchie and Ken Thompson, the original developers of Unix, in front of a DEC PDP-11. It's from http://histoire.info.free.fr/images/pdp11-unix.jpeg, though the photo appears to be owned by Lucent Technologies. Unix was originally implemented on a DEC PDP-7, but was soon ported to the more-capable PDP-11.
Multiprogramming refers to the notion of having multiple programs active at the same time so that when the current program can no longer run (for example because it’s blocked waiting for I/O), another is available to run. Timesharing is an extension of multiprogramming in which the execution of the active programs is time-sliced: each program runs for a short period of time, then another is run. A good web site with useful information on Multics is http://www.multicians.org/. A brief description of CTSS can be found at http://www.multicians.org/thvv/7094.html.
The slide lists milestones in the history of virtual memory, from its first instance on Manchester’s Atlas computer in 1961 (when a working prototype was completed), to Apple’s announcement of VM support for the Macintosh.
The 1970s saw the continued importance of the sorts computing important in the '60s. But, what would become the most significant form of computing in later decades was introduced as personal computing and hobbyist computing.

<table>
<thead>
<tr>
<th>1970s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commercial data processing</td>
</tr>
<tr>
<td>Scientific computing</td>
</tr>
<tr>
<td>Time sharing</td>
</tr>
<tr>
<td>Laboratory computing</td>
</tr>
<tr>
<td>Personal computing</td>
</tr>
<tr>
<td>Hobbyist computing</td>
</tr>
</tbody>
</table>
The photo is of an IBM System/370 Model 168, from http://www-03.ibm.com/ibm/history/exhibits/mainframe/mainframe_2423PH3168.html. This was a high-end model and one of the first “mainstream mainframes” to support virtual memory.
The photo is of a Cray-1 supercomputer in the Computer History Museum. The photo is by Ed Toton who, according to http://en.wikipedia.org/wiki/File:Cray-1.jpg, has released it to the public domain. The first Cray-1 was installed at Los Alamos National Laboratories in 1976. Cray’s philosophy was, essentially, no compromises allowed. Virtual memory was eschewed in favor of having enough real memory to make it unnecessary. Similarly a memory cache wasn’t used — all memory was as fast as most contemporary computer’s caches were. The OS, known as COS (Cray Operating System) was strictly batch-oriented and designed to run one job at a time. Jobs (including associated data) were assembled by attached mainframe computers. In essence, COS did not do a whole lot more than the Input/Output System of the IBM 701.

In the 1980s, Cray adapted Unix for their machines, calling their version Unicos.
This is a photo of a Xerox Alto from http://commons.wikimedia.org/wiki/File:Xerox_Alto_full.jpg. Produced in 1973 at Xerox’s Palo Alto Research Center (PARC), it’s generally regarded as the first computer to demonstrate the utility of a window manager and mouse. Though it was never a product, it introduced the notion of a serious personal computer.

Though its historical importance is primarily in its pioneering use of bit-mapped graphics to implement a window-managed display, it had an interesting OS as well. It was strictly a single-user system and provided no protection whatsoever. However, it had a layered approach to OS design and implementation. Through its notion of “juntas,” one could remove layers of the OS, one at a time, just in case one needed the storage for a large program (and one could read them back from disk).

A great deal of Alto documentation can be found at http://www.bitsavers.org/pdf/xerox/alto/.
This photo, of a MITS ALTAIR 8800, is from http://www.vintage-computer.com/altair8800.shtml. Introduced in 1975, it was an early, if not the first, “hobbyist computer.” It was also the platform on which Microsoft’s first product, a BASIC interpreter, ran. It had no operating system, just standalone programs (in particular, a BASIC interpreter supplied by Microsoft — their very first product and very first platform).
CP/M was the first OS affordable to individual hobbyists.
This photo, of an Apple II, is from http://commons.wikimedia.org/wiki/File:Apple_II.jpg. It was introduced in 1977 with no OS. Later (in 1978) a simple OS was released with functionality similar to that of CP/M.
Microsoft Enters the OS Business: Late 1970s

- It’s called …
  - Xenix
    - a version of Unix
  - predominant version of Unix in the 1980s
  - used by MS internally into the 1990s
The VAX-11, of which the 780 was the first model, was introduced in 1978 and was noted for two operating systems. The first, VMS, was the product of its manufacturer, DEC. It is still supported by HP, the company that purchased Compaq, which was the company that purchased DEC. VMS was very much the predecessor of modern Windows.

The second operating system is Unix, most notably BSD Unix. Seventh-Edition Unix was ported to the VAX-11 by Bell Labs and called Unix 32V. It did not support virtual memory. Researchers at the University of California at Berkeley used it as the basis of 3BSD Unix, which did support virtual memory. Later came 4.0BSD, 4.1BSD, up through 4.4BSD. Along the way DEC produced its own version of Unix for the VAX-11, based on 4.2 BSD, called Ultrix. It was the implementation of the TCP/IP network protocols on 4.2BSD that did much to grow the Internet, making it accessible to a large number of academic computer science departments and industrial research organizations.
The introduction of the IBM PC co-opted the term “personal computing,” which previously had been applied to work done at Xerox PARC. High-end “computer workstations” were introduced for professional computing, running more-or-less state-of-the-art operating systems. More affordable computers, such as IBM PCs (and clones) and Apple Macintoshes, had much less capable operating systems.

The advent of the PC pretty much eliminated the mini-computer market and separate computers (and operating systems) for laboratory computing.
The 1980s saw the rise of two operating systems: Unix via BSD Unix on VAXes and Xenix on higher-end PCs, and MS-DOS on PCs.
This photo of an early IBM PC is from http://www-03.ibm.com/ibm/history/history/history_intro.html.
An Apollo workstation, introduced in 1982. (Note the “C” on its front — this was its serial number (in hexadecimal)). It had a fairly sophisticated OS for its day.
The 1990s saw the convergence of the low- and high-end personal computing: hardware powerful enough to run the operating systems of high-end personal computing became affordable at the low-end.
Toy Operating Systems

- 1987: Andrew Tanenbaum of Vrije Universiteit, Amsterdam, publishes *Operating Systems: Design and Implementation*
  - included is source code for a complete, though toy, operating system: Minix, sort of based on Unix

- 1991: Linus Torvalds buys an Intel 386 PC
  - MS-DOS doesn’t support all its features (e.g., memory protection, multi-tasking)
  - “soups up” Minix to support all this

- January 1992: Torvalds releases Linux 0.12
- January 1992: Tanenbaum declares Linux obsolete
Late ’80s/Early ’90s

- 1988: Most major Unix vendors get together and form OSF to produce a common Unix: OSF/1, based on IBM’s AIX
- 1989: Microsoft begins work on NT
- 1990: OSF abandons AIX, restarts with Mach
- 1991: OSF releases OSF/1
- 1992: Sun releases Solaris 2
  - many SunOS (Solaris 1) programs are broken
- 1993: All major players but DEC have abandoned OSF/1
- 1993: Microsoft releases Windows NT 3.1
- 1994: Linux 1.0 released

Note that the notion of a completely free (and useful) operating system didn’t really exist in the early 1990s. Unix was licensed by AT&T, who charged a hefty fee for its commercial use.
Late ’90s

- IBM has three different versions of Unix, all called “AIX”
- 1996: DEC renames its OSF/1 “Digital Unix”
- 1996: Microsoft releases Windows NT 4
- 1996: Linux 2.0 released
- 1998: DEC is purchased by Compaq; “Digital Unix” is renamed “Tru64 Unix”
- 1999: Sun’s follow-on to Solaris 2.6 is called Solaris 7
The 2000s have brought on the “gadget” as an important computing device, one deserving of an OS.
The ’00s Part 1

- 2000: Linux 2.2 is released
- 2000: IBM “commits” to Linux (on servers)
- 2001: Apple releases Mac OS X, based on Unix
- 2001: Linux 2.4 is released
- 2001: Microsoft releases Windows XP
- 2002: Compaq is purchased by HP
- 2003: SCO claims their code is in Linux, sues IBM; IBM countersues
  - August 10, 2007: judge rules that SCO is not the rightful owner of the Unix copyright, Novell is
  - Novell says there is no Unix in Linux
  - September 2007: SCO files for Chapter 11 bankruptcy protection
The ’00s Part 2

- 2004: Linux 2.6 is released
- 2005: IBM sells PC business to Lenovo
- July 2005: Microsoft announces Windows Vista
- January 2007: Microsoft releases Windows Vista
- Later in 2007: Microsoft starts hinting at Windows 7
- June 2007: Apple releases iOS for iPhone
- April 2009: Oracle announces purchase of Sun Microsystems
- July 2009: Google announces Chrome OS
- October 2009: Microsoft releases Windows 7
The ’10s Part 1

- January 2011: Microsoft announces Windows 8
- June 2011: first products shipped running Chrome OS
- April 2011: Linux 3.0 released
- July 2011: Apple drops “Mac” prefix for its OS: it’s now “OS X”
  - one million copies of OS X Lion sold on first day of release
- October 2011: deaths of both Dennis Ritchie and Steve Jobs
  - “Dennis Ritchie: The Shoulders Steve Jobs Stood On”
    - Wired Magazine, 10/13/2011
- October 2012: Microsoft releases Windows 8
- October 2013: Microsoft releases Windows 8.1
The ‘10s Part 2

• September 2014: Microsoft Announces Windows 10
  – Windows 9 skipped because it might be confused with Windows 95
• October 2014: Apple releases OS X 10.10 Yosemite
  – doesn’t work with Brown University wi-fi
  – November 2014 release of 10.10.1 doesn’t fix it ...
• April 2015: Linux 4.0 released
• July 2015: Microsoft Releases Windows 10
  – Brown’s CIS says “don’t switch yet”
  – August 2015: Brown’s card-access system switched to Windows 10
  – August 2015: Brown’s card access fails university-wide
• September 2015: Apple releases OS X 10.11 El Capitan
  – Brown’s CIS says “don’t switch yet”
The ’10s Part 3

• September 2016: Apple releases OS X 10.12 Sierra
  – Brown CIS says don’t switch
  – Brown CS tstaff says printing is broken
• December 2016: Apple releases OS X 10.12.2
  – twd switches
  – his D-Link router no longer supports printing
  – his Logitech mouse’s scroll wheel scrolls window in random directions
• January 2018: Meltdown and Spectre vulnerabilities disclosed
  – end of computing (if not civilization) as we know it?
Friday

- Implementing threads
  - now is a good time to review POSIX threads