An Introduction to Operating Systems

What is an operating system?

A piece of "systems software" that provides a convenient, efficient environment for the execution of user programs. It's probably the largest, and most complex, program you'll ever run!

Why do we need an operating system?

- The user's viewpoint – to provide the user interface, command interpreter, directory structure, to execute application programs (word processor, email client, web browser, MP3 player).

- The programming environment viewpoint – to enhance the bare machine, to provide utility programs (such as compilers, editors, filters), to provide high-level input and output (I/O), structure information into files, improve access to memory (size, protection, sharing).

- The efficiency viewpoint – to replace the (long departed) human operator, to schedule tasks, efficiently store and retrieve data, invoke and share programs.

- The economic viewpoint – allow simultaneous use and scheduling of resources, including disk-bound data and expensive peripherals.

Traditionally, we would summarize an operating system's goals as making "the system" *convenient* to use and to schedule its resources *efficiently*.

In addition it must support hardware and software not yet developed.

Operating System $\neq$ User/Computer Interface

An operating system is often simply seen and described as the user/computer interface.

We often (mistakenly) claim to understand, and like or dislike, an "operating system" based on its interface.

Such an interface often provides us with:

- program creation (editors, compilers, debuggers, linkers)
- program execution (character and graphical)
- access to I/O devices (both fixed and removable)
- constrained access to files of media
- constrained access to "internal" resources
- error detection, response, reporting
- accounting and monitoring

Whether or not a certain interface runs on a particular hardware or operating system platform is usually dictated by economics, marketing, and politics – not technology.

Operating System $\equiv$ Resource Manager

An operating system is better considered as being in control of its hardware and software resources.

Better still, because the "controls" are often temporal or external to the operating system itself, let's consider the operating system as a *resource manager*.

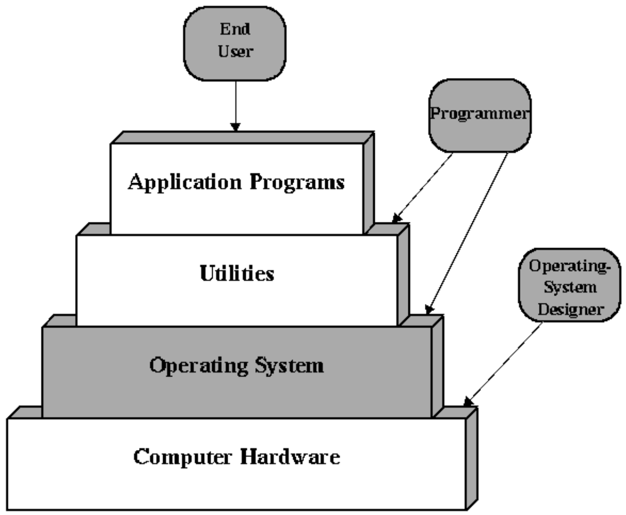
An operating system is *just another program* running on the available hardware. Most of the time, the operating system relinquishes control to the "user processes" until the hardware again dispatches the control.

Operating Systems Must Be Extensible

Of importance is an operating system's ability to *evolve* to meet new hardware and software demands:

- New hardware is constantly introduced – adding more memory presents little difficulty; new types of disks, video cards, etc, are more problematic.
- New application programs, tools, and system services are added
- Fixes and patches are released to correct operating system *deficiencies*.

All of this suggests that the operating system, as a *program*, needs to be extensible – a *modular design* seems essential.



Traditional Operating System Services

- CPU scheduling – distribute or apportion computing time among several processes (or tasks) which appear to execute simultaneously.
- Memory management – divide and share physical memory among several processes.
- Swapping – move processes and their data between main memory and disk to present the illusion of a bigger machine.
- I/O device support – provide specialized code to optimally support device requirements.
- File system – organize mass storage (on disk) into files and directories.
- Utility programs – accounting, setting/constraining system resource access, manipulating the file system.
- A command interface – textual or graphical, to enable interactive interrogation and manipulation of operating system features.
- System calls – allow constrained access to the interior of the running operating system (as a program).
- Protection – keep processes from interfering with each other, their data, and "the system", whilst permitting sharing when requested.
- Communication – allow users and their programs to communication within a single machine, and across networks.

A Whirlwind History of Operating Systems

To understand the way modern operating systems are the way they are, it is useful to examine their evolution over the last 55 years.

Advances in operating systems often accompanied advances in hardware, falling prices and "exploding" capacities.

The first true digital computer was designed by English mathematician Charles Babbage (1792-1871). Although Babbage spent most of his working life and fortune building his "analytical engine", its mechanical design and the wooden technology of the day could not provide the required precision.

Needless to say, the analytical engine did not have an operating system.

"Everything that can be invented has been invented"
Charles H. Duell, Commissioner, U.S. Office of Patents, 1899

1945-55: Vacuum Tubes and Plugboards

Until World War-II, little progress was made in constructing digital computers. In the mid-1940s, four significant groups (now) laid claim to the first electrical computers:

- Howard Aitken, Harvard,
- John von Neumann, Institute of Advance Studies, Princeton,
- J.P. Eckert and W. Mauchley, University of Pennsylvania, and
- Konran Zuse, Germany.

A single group of people designed, built, programmed, operated and maintained each machine. Although filling small warehouses, with 10s of thousands of vacuum tubes, they were no match for today's cheapest home computers.

"I think there is a world market for maybe five computers."
Thomas Watson (1874-1956), Chairman of IBM, 1943

Programs were loaded manually using console switches, or more likely by direct reconfiguration of wiring; indication of a program's execution and debugging returned through console lights.

Advantages:
- Interactive, and user received immediate response.

Disadvantages:
- Expensive machine was idle most of the time, because people's reactions (and thinking) were slow.
- Programming and debugging were tedious; hardware was very unreliable.
- Each program was self contained, including its own code for mathematical functions and I/O device support.

1955-65: Transistors and Batch Systems

Programming languages and operating systems still unheard of. Collections of subroutines (procedures) to drive peripherals and to evaluate trigonometric functions were the first examples of operating systems services. The mid 1950s saw the user (still as the programmer) submitting a deck of punched Hollerith cards describing a job to be executed.

Given the high cost of the computers, ways to increase its utility were quickly sought. The general solution was the *batch system*.

Similar/related programs, perhaps each requiring the FORTRAN compiler, or a set of mathematical routines, were *batched* together so that the require routines need only be physically loaded once.

Programs were first written on paper tape, in the emerging FORTRAN language (FORula TRANslation) or in assembly language and then copied to punched cards. Such decks of cards included *job control cards*, the program itself, and often the program's data.

Jobs submitted by different users were sequenced automatically by the operating system's *resident monitor*.

Early peripherals, such as large magnetic tape drives, were used to *batch* input (jobs and data) and *spool* (from Simultaneous Peripheral Operation OnLine) output.

A generally inexpensive computer, such as an IBM 1401, was used for reading cards and printing from output tapes. The expensive machine, such as an IBM 7094, was used soley for the mathematical computations.

1955-65: Transistors and Batch Systems, *continued*

Advantages:

- The computer was kept busier.

Disadvantages:

- The computer was no longer interactive. Jobs experienced a longer *turn-around* time.
- The CPU was still idle much of the time for *I/O bound* jobs. Other jobs remained queued for execution.

The significant operating system innovation at this time was the introduction of a *command interpreter* (a job control language – JCL) to describe, order and commence execution of jobs.

The resident monitor was also protected from the user programs, and managed the automated loading of the programs after the monitor.

1965-1980: ICs and Multiprogramming

In the early 1960s, computer manufacturers typically made two types of computers – word-oriented, large scale scientific computers (such as the 7094), and character-oriented commercial computers (such as the 1401) which were really better suited for I/O. Incompatability and a lack of an upgrade path were the problems of the day.

IBM attempted to address both problems with the release of their System/360, a *family* of software compatible machines differing only in capacity, price and performance. The machines had the same architecture and instruction set.

With heavy CPU-bound scientific calculations, I/O is infrequent, so the time wasted waiting was not significant. However, commerical processing in the emeging COBOL (Computer Oriented Business Organizational Language) often spent 80-90% of its time waiting for I/O to complete.

The advent of separate I/O processors made simultaneous I/O and CPU execution possible.

The CPU was *multiplexed* (shared), or employed *multiprogramming*, amongst a number of jobs – while one job was waiting for I/O from comparatively slow I/O devices (such as a keyboard or tape), another job could use the CPU.

Jobs would run until their completion or until they made an I/O request.

(*The world's first commercially available time-sharing computer, the DEC PDP-6, was installed in UWA's Physics Building in 1965 – c.f. www.general.uwa.edu.au/u/alex/warcc/CyberHistory-ch4-UWA.html)*

1965-1980: ICs and Multiprogramming, *continued*

Advantages:

- Inter-activeness was restored.
- The CPU was kept busy if enough jobs were ready to run.

Disadvantages:

- The computer hardware and the operating system software became significantly more complex (and there has been no looking back since).

Still, the desire for quicker response times developed a variant of multiprogramming in which each user communicated directly with one of a multitude of I/O devices.

The introduction of *timesharing* introduced *pre-emptive scheduling*. Jobs would execute for at most pre-defined time interval, after which it was another job's turn to use the CPU.

The first serious timesharing system (CTSS, from MIT 1962) lacked adequate memory protection.

Most (modern) operating system complexity was first introduced with the support of multiprogramming – scheduling algorithms, deadlock prevention, memory protection, and memory management.

"There are only 2 things to come out of Berkley, Unix and LSD, and I don't think this is a coincidence"

Jeremy S. Anderson

1970s – Minicomputers and Microcomputers

Another major development occuring in parallel was the phenomenal growth in minicomputers, starting with the DEC (Digital Equipment Corporation) PDP-1 (Programmed Data Processor) in 1961. The DEC PDP family of minicomputers The PDP-1, with 4K of 18bit words cost only US$120,000 – 5% of the IBM 7094.

The trend was towards many small mid-range personal computers, rather than a single mainframe.

Early minicomputers and microcomputers were simple in their hardware architectures, and so there was some regression to earlier operating system ideas (single user, no pre-emption, no multiprogramming).

> *For example, MS-DOS on an IBM-PC (circa. 1975) was essentially introduced as a batch system, similar to those of the 1960s, with a few modern ideas, such as a hierarchical file system, thrown in.*

With some notable exceptions, the trend quickly moved towards support of all modern operating system facilities on microcomputers.

"There is no reason anyone would want a computer in their home."
Ken Olsen, DEC Founder and Chairman, 1977

Perhaps most significant has been the evolution, and importance, of operating systems' user interfaces.

In particular, the graphical *desk metaphor* has remained for some time.

1980-90 – Personal Computers and Networking

"640K ought to be enough for anybody"
Bill Gates (1955-), in 1981

The decentralization of computing resource, now *data* and not the hardware, required more support for inter-operating system communication – both physical support and application program support.

As minicomputers shrunk in size, but exploded in capacity, the powerful *computer workstation* was born. Companies such as Sun Microsystems and Silicon Graphics rode this wave of success.

Local-area networks (primarily Ethernet and token-ring) connected workstations, while wide-area networks connected minicomputers.

Operating system developments included the development of fast and efficient network communication protocols, data encryption (of networks and file systems), security, reliability, and consistency of distributed data.

1990s and Beyond

- Hardware prices drop, dramatically, and capacities explode.
- High speed, long distance communication links encourage graphical and audio communication (surpassing text).
- The *desktop metaphor* for computer interfaces becomes only an instance of the wider *Web metaphor*.
- Computers become portable and increasingly ubiquitous.
- Computers again become idle; artificial uses keep them busy.

> *"For years, we thought that a million monkeys sitting at a million keyboards would produce the complete works of Shakespeare. Today, thanks to the Internet, we know that's not true."*

What to do this week

- Make your laboratory and tutorial bookings via *csbookings*, from 12noon Wednesday 24th.

- Reading Stallings 4th edition, sections 2.1-2.4, for a quick introduction to operating system development.

- Read Stallings, sections 2.5 $\rightarrow$ 2.7, as introductions to the Windows-NT and UNIX/LINUX operating systems.

- Prepare your answers for, and attend the 1st tutorial (commencing Monday 29th).