Basics of Operating Systems (1)

Dr. Qin Zhou

Coventry University
email: q.zhou@coventry.ac.uk

The desktop

- Most systems use a graphical *desktop* to allow users to interact with them
  - – mouse and/or keyboard used to activate icons representing programs, switch between windows, move and resize windows...
- Is the desktop the operating system?
  - – no, but it's the most visible part of it!

The operating system

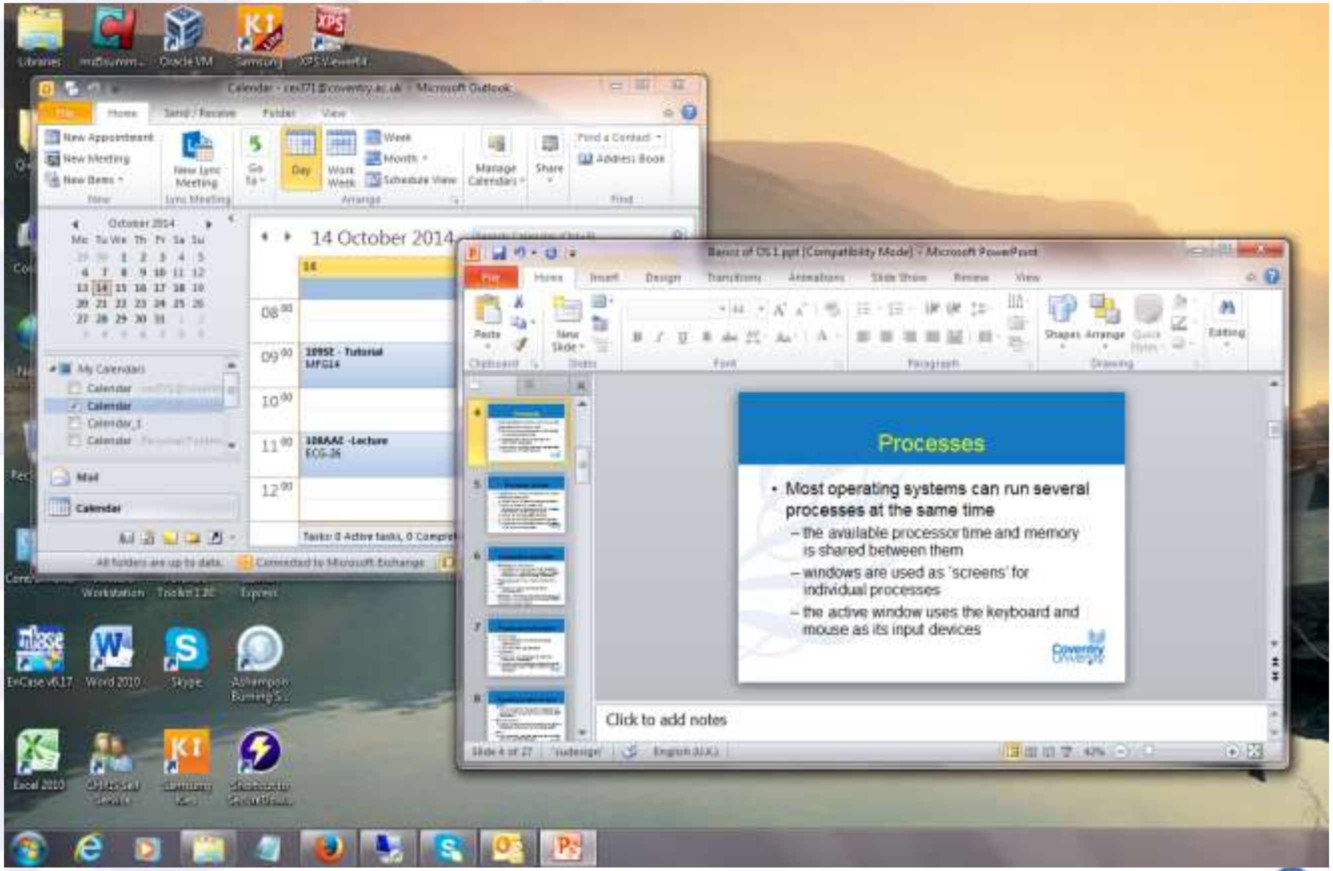
- An operating system is basically responsible for managing *processes*
- Processes execute programs
  - a process provides a program with memory, processor time and other resources it needs to execute
  - the desktop is displayed by a program which is executed by one of the first processes created when the operating system starts up

Coventry University

Processes

- Most operating systems can run several processes at the same time
  - the available processor time and memory is shared between them
  - windows are used as ‘screens’ for individual processes
  - the active window uses the keyboard and mouse as its input devices

Processes - Example



Processes

- A process is like an emulation of a self-contained computer
  - it can have a window to act as its screen
  - when a window is active, the corresponding process uses the keyboard and mouse as its input devices
  - it has its own separate memory
  - it has its own set of processor registers
  - it processes instructions from the program in its own memory area

Operating system structure

- Kernel:
  - the inner part of the system responsible for dealing with processes and allocating system resources
- Device drivers:
  - kernel components that interact with specific hardware devices (e.g. the sound card)
- Shell:
  - the ‘desktop’ program that lets you interact with the operating system

Resource management

- Resources include:
  - processor time
  - memory
  - disk space
  - network bandwidth
  - access to unshareable devices (e.g. sound card, mouse, keyboard)
  - any others?

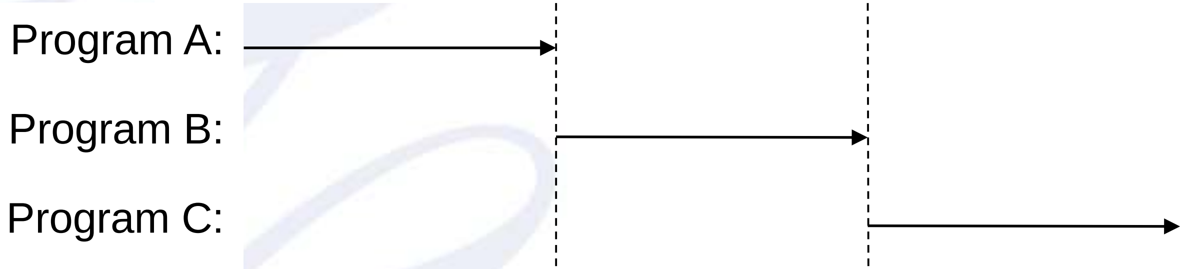
Coventry University

Processor management

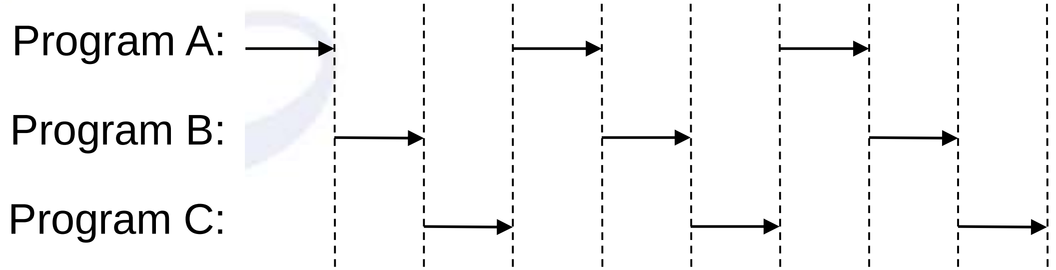
- Processes run for a short period of time (a ‘quantum’ or ‘timeslice’) and are then *interrupted*
  - the OS kernel regains control and can switch to another process
- If the timeslice is small enough, the processes appear to run in parallel with each other

Processor management

Sequential execution of three programs



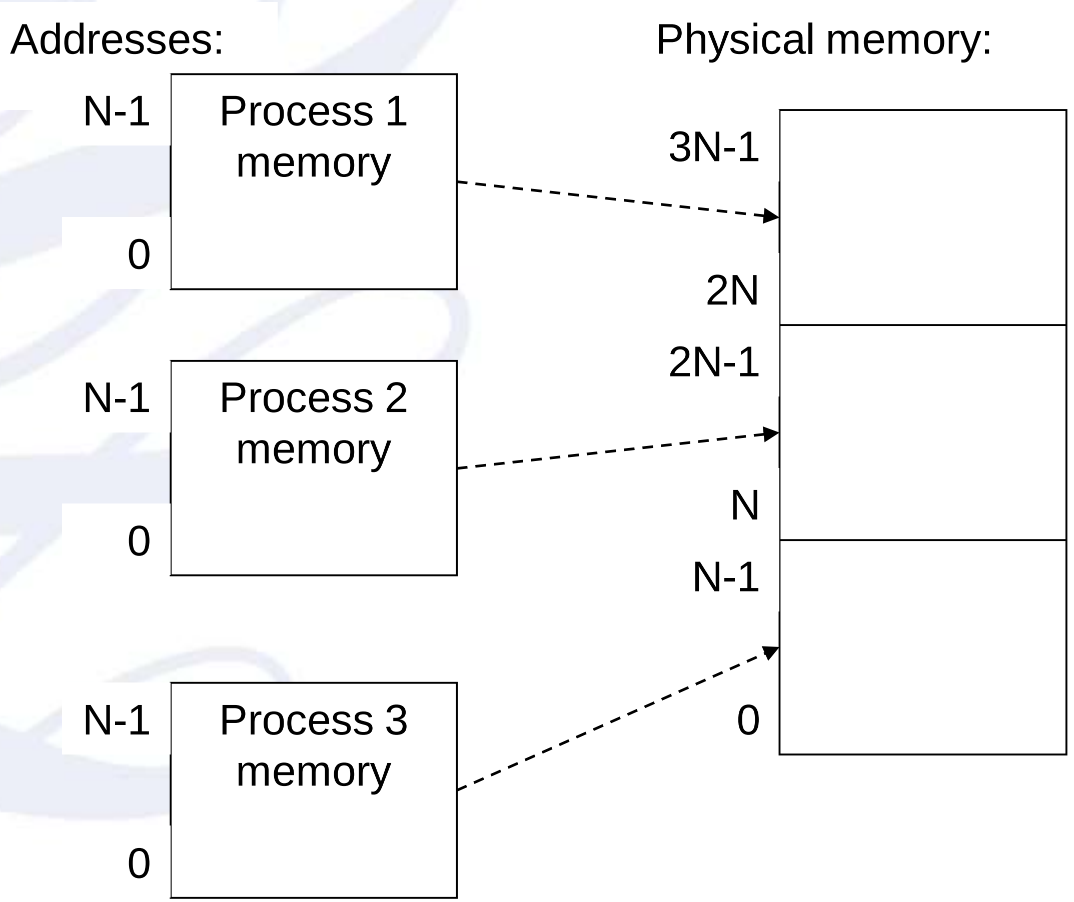
Concurrent execution of three programs



Memory management

- Each process has its own address space
  - range of addresses from 0 upwards that it can refer to
  - memory management hardware maps (translates) process addresses into corresponding addresses in physical memory

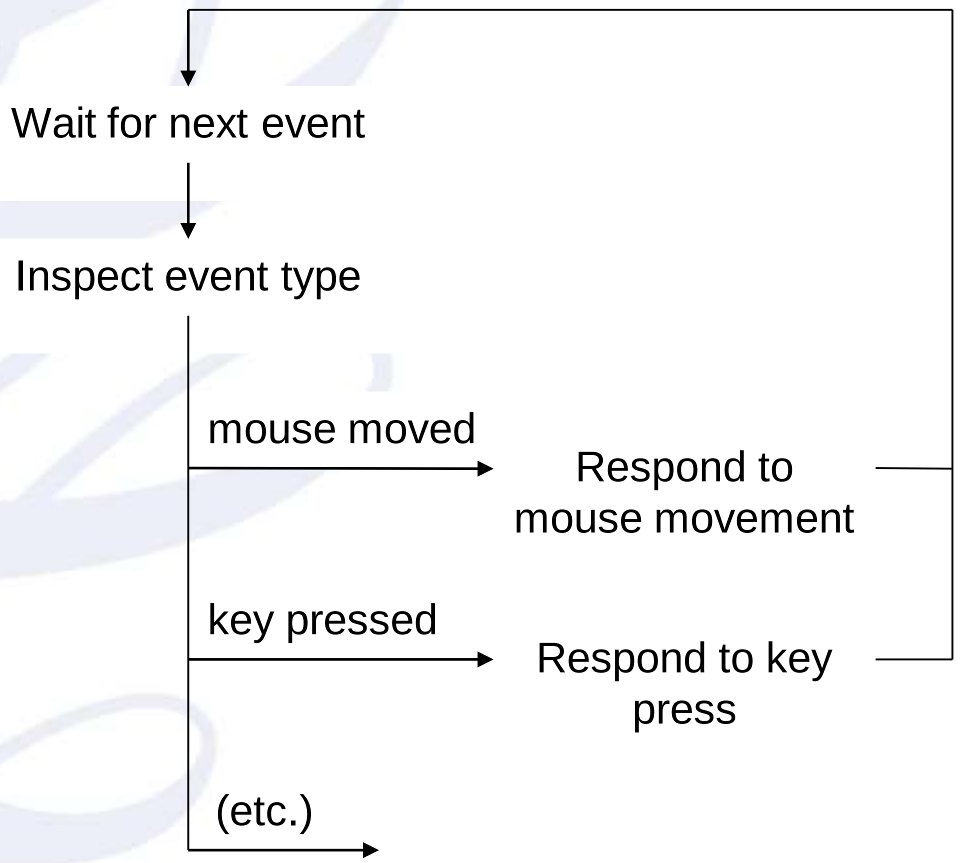
Memory management



Event-driven systems

- GUI programs are event driven
  - events arrive in an unpredictable sequence from various sources (mouse, keyboard, ...)
  - programs must be prepared to deal with events in any sequence

Event-driven systems



Event-driven systems

- Events are produced by hardware generating *interrupt* signals
  - these interrupt the current process and enter the kernel
  - the kernel calls the appropriate *device driver* to deal with each interrupt
  - an event notification message is added to a message queue associated with the process which needs to deal with the event

Event-driven systems

- After processing an interrupt, the kernel needs to resume a process
  - the kernel provides a mechanism to resume a process in the same state as it was when interrupted
- The decision about which process to resume depends on system’s *scheduling policy*
  - policies are kept separate from mechanisms

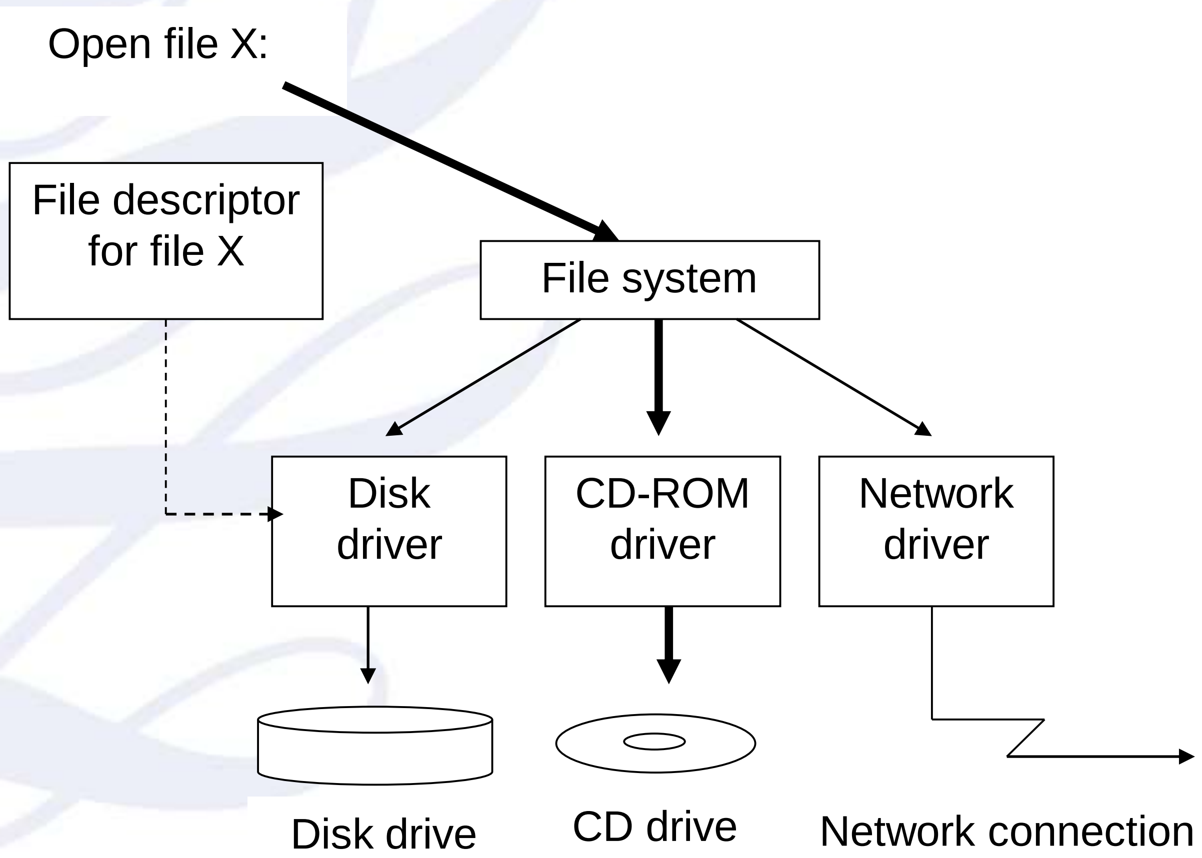
Protection

- Memory management hardware prevents access outside the allocated address space of a process
- ‘Privileged instructions’ can only be executed by the system kernel
  - e.g. halt processor, disable interrupts, access I/O devices, switch to supervisor (kernel) mode
- These precautions prevent processes from interfering with each other

Virtual resources

- Operating system can provide a simplified, idealised view of a resource
  - example: file system provides named files
  - they may be located on disks, CDs, remote machines...
  - access is the same regardless of how or where the data is actually stored

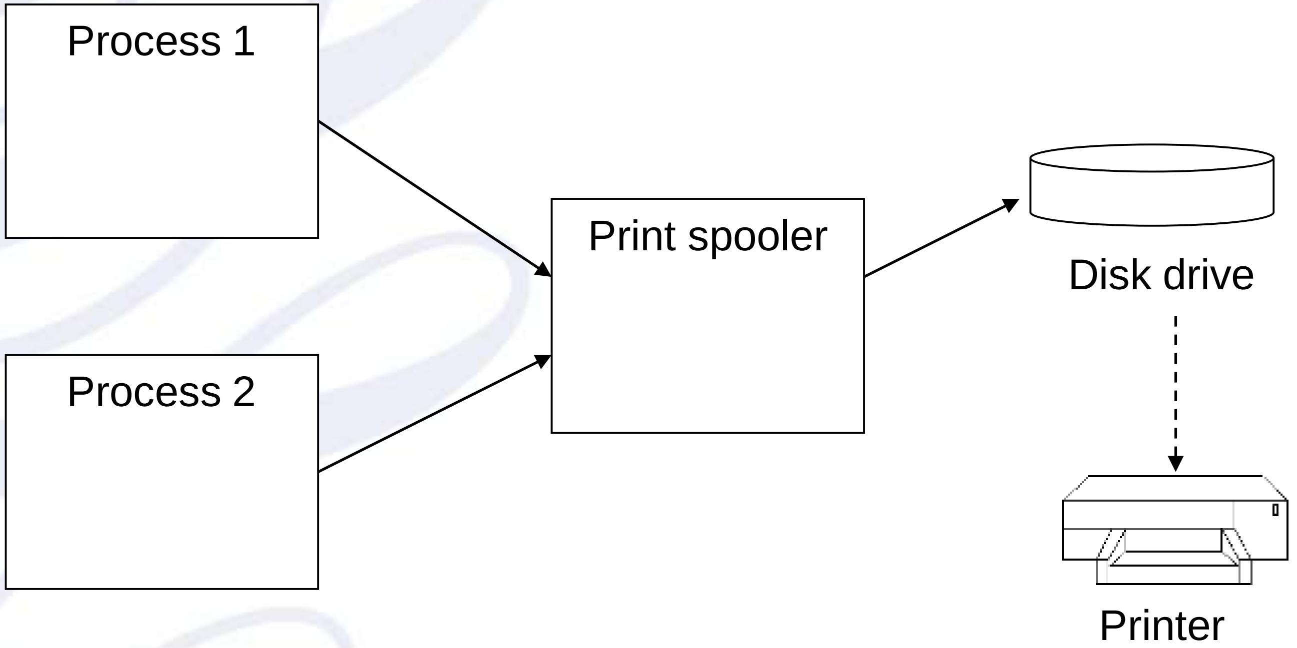
Virtual resources



Virtual resources

- Another example: print spooling
- Printers are unshareable devices (a printer can only be used by one process at a time)
  - solution: `print' documents into a file on disk and have a `spooler' process which prints pending documents one after another
  - behaves like a perfect printer: always available, always online...

Virtual resources

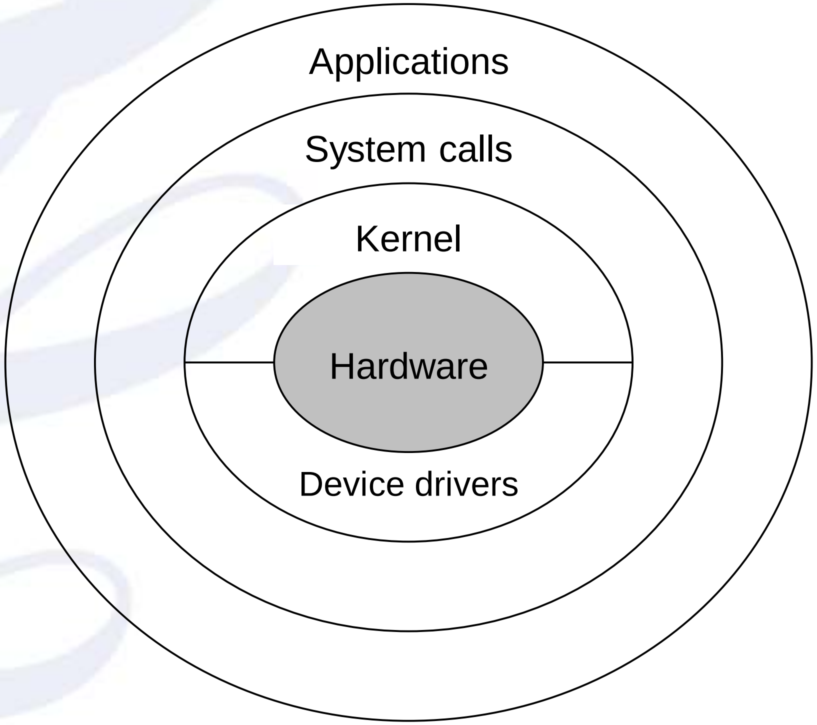


OS designs

- These reflect the evolution of programming paradigms:
  - early OSs: unstructured, hard to maintain
  - structured programming: layered designs
  - object-oriented programming: object-oriented designs, microkernel designs

OS designs

- Most OSs use a layered (possibly O-O) design:



OS requirements

- Main requirements:
  - efficiency
  - reliability
  - ease of use (configurability)
  - security
- Other requirements:
  - maintainability (modularity)
  - flexibility (mechanisms vs. policies)

Coventry University

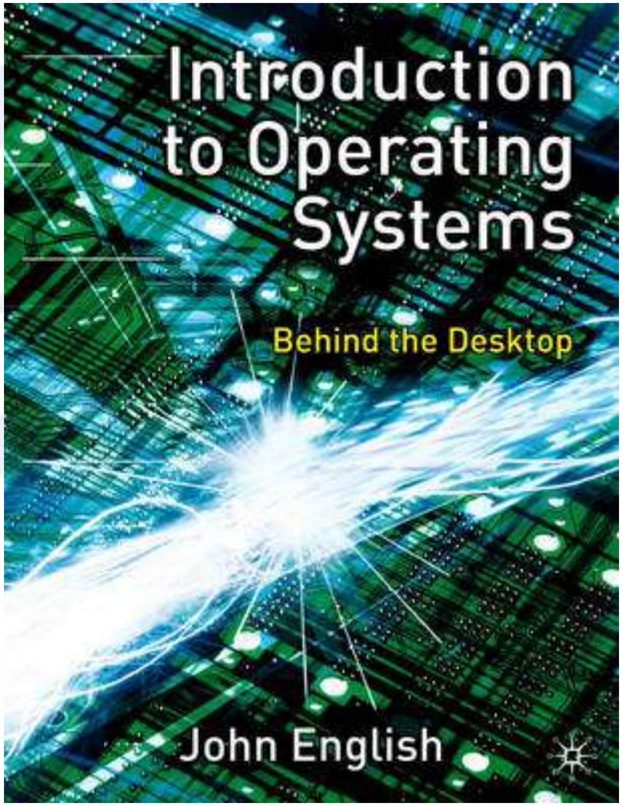
OS examples

- Unix
  - developed by Kernighan & Ritchie at Bell Labs in the early 1970s
  - written in C, so easily ported to different hardware platforms
  - many versions (Apple’s A/UX, OS/X; IBM’s AIX; SGI’s IRIX; Sun’s SunOS, Solaris)
  - PC versions: Xenix, SCO Unix, Solaris, Linux, FreeBSD
  - POSIX standard (IEEE 1003.1 etc.)

OS examples

- Microsoft Windows
  - a commercial product
  - versions include 95, 98, ME, NT, 2000, XP, Vista, 7/8/10
  - common functionality (the Win32 API)
  - some incompatibilities between versions
  - POSIX subsystem (required for all US government systems)

Reference


Introduction to Operating Systems
Behind the Desktop
John English