COMP124

Computer Systems

09 | Operating Systems | Multitasking | Processes

Dr Stuart Thomason

Operating Systems

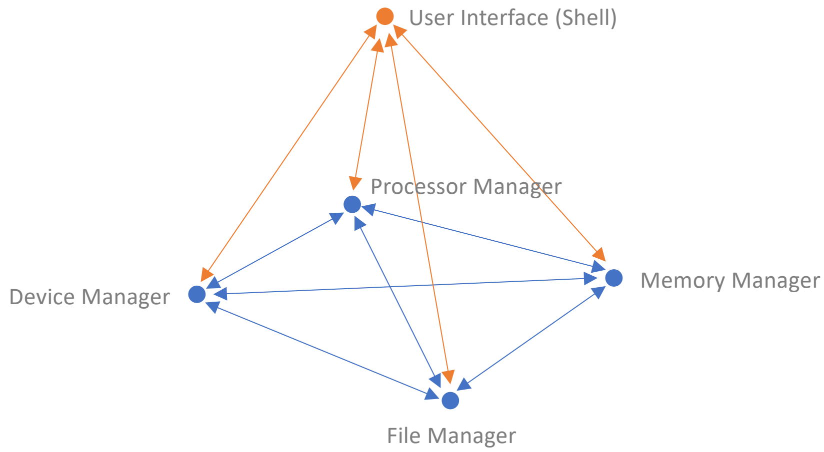
- Operating systems serve two main purposes
  - Turn hardware components into a usable device
  - Make efficient use of resources (particularly when shared between processes)

- Common general-purpose operating systems…
  - Windows
  - Unix
  - Linux
  - MacOS
  - iOS
  - Android

Based on (or heavily influenced by) Unix

- Embedded operating systems inside home appliances, TV boxes, game consoles, etc.

OS Abstract View



OS Managers

- The base of the pyramid shows the four essential managers of every operating system
  - Processor manager
  - File Manager
  - Device Manager
  - Memory manager

- Each manager must perform certain tasks
  - Continuous monitoring of resources
  - Enforcement of policies (who gets resources, when, and how much)
  - Allocation of resources when appropriate
  - Deallocation of resources when no longer needed

- Managers must work in harmony with each other to complete tasks

Processes and Programs

- There is a difference between processes and programs

- A program is the code that performs some task (algorithm, etc.)
  - Source code (Java, C++, etc.)
  - Object code (compiled executable) stored on disk somewhere
  - Program is static (doesn't change) after it's compiled

- A process is the activity that the CPU performs when it executes a program
  - Code is loaded from disk into memory
  - Instruction pointer starts at first instruction in memory
  - Process is dynamic (execution branches depending on input)

- Not necessarily a one-to-one correspondence between programs and processes

Operating System Structure

- An operating system consists of various parts
- A central kernel…
  - Resides permanently in memory
  - Performs low-level, frequently needed activities (such as context switching)
  - Operates in kernel mode (also known as privileged mode)
- A shell…
  - Provides the user interface for the operating system
  - Allows the user to run and interact with programs (processes)
  - Operates in user mode (no special privileges, restricted access)
- A set of processes…
  - Might be created by the kernel to carry out its activities
  - Or executed when the user runs software
  - Can be either privileged or non-privileged depending on how they were started

System Boot (Initialisation)

- When the device is first turned on…
  - Interrupt is sent to the CPU
  - Instruction pointer is set to the first address in ROM (read-only memory)
  - Fetch-execute cycle begins from this address

- The ROM contains a small bootstrap program
  - Performs basic system checks
  - Sets up system bus and I/O channels
  - Loads OS kernel from disk and passes control to it (via instruction pointer)

- Kernel performs further setup and system checks
  - Starts various processes to perform background tasks of the operating system
  - Starts the main shell process (from which further processes can be triggered)

Command Interpreter (Shell)

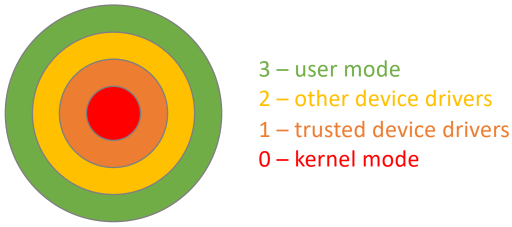
- The shell itself is a collection of processes that are spawned by the kernel
  - Allows user to interact with the operating system
  - Processes run in user mode
  - User can run more processes by typing a command or clicking an app icon

- In the early days…
  - Computers were big and expensive (as big as a room)
  - Users connected to a central mainframe via a text-based terminal
  - Much of the ASCII notation dates back to this era (character codes)

- Nowadays…
  - Linux, MacOS and Windows all provide a text-based view (terminal or DOS window)
  - But they also have graphical shells (desktop metaphor)

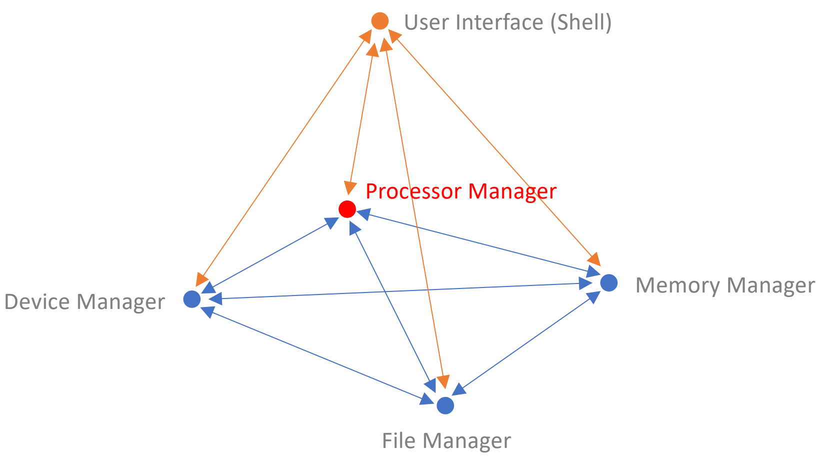
Protection Levels

- The kernel has access to all parts of the CPU, every component, and every I/O device
  - Must be protected from unauthorised access by users (hackers, malware, etc.)
  - Kernel mode is enforced via a protection ring (Intel x86 processors have four rings)
  - The CPU flags register stores the privilege level of each process



- Certain registers, instructions and memory addresses are protected
  - Can only be used by processes with the correct privilege level
  - Interrupt generated if privilege level doesn't match (general protection fault)

Processor Manager



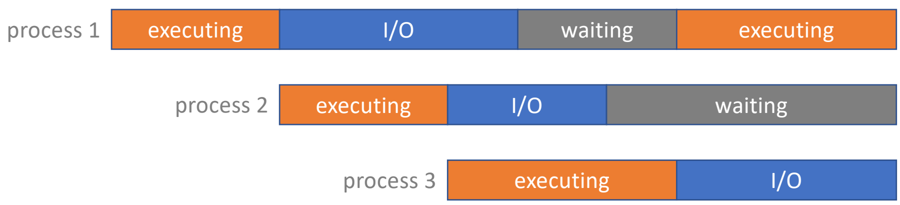
Processor Manager

- Decides how to allocate the CPU to waiting processes
- Driven by a desire to do something useful when a process cannot continue (ie. maximise throughput)

- Processor manager performs various tasks for the operating system…
  - Creates processes when a program is executed
  - Initialises memory and stack for new processes
  - Keeps track of the status of processes
  - Assigns processes to the CPU when available (context switch)
  - Changes process states as events occur
  - Handles termination of processes on completion or abort
  - Handles inter-process communication
  - Manages process queues and prioritisation (scheduling)

Multiprogramming

- Early operating systems facilitated multiprogramming
  - Load several processes into memory simultaneously (sharing the CPU)
  - When running process can't continue (eg. waiting for I/O), switch to another
  - Hence I/O and computation can overlap

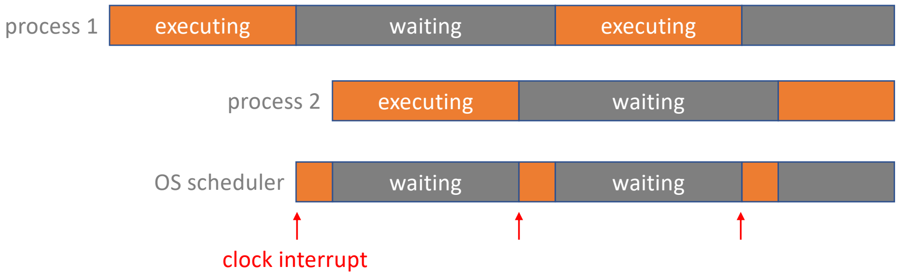


  - This means the CPU is always in use (executing something)

- Problems arise with compute-bound and I/O-bound processes

Multitasking (Time-Sharing)

- Extends the concept of multiprogramming by providing fair access to CPU
  - OS switches rapidly between processes to give illusion of uninterrupted execution in parallel (multitasking)
  - Each running process is given a fixed time slice (quantum) on the CPU



  - The OS itself needs time on the CPU to perform its own scheduling tasks

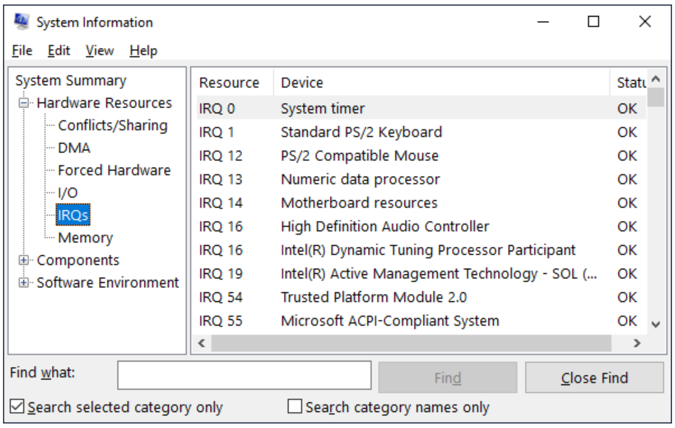
Interrupt Handling

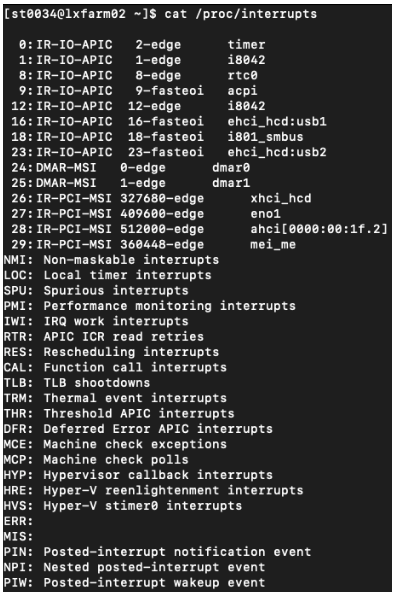
- Multitasking depends on the ability to interrupt the CPU at regular intervals
  - An interrupt request (IRQ) is a hardware signal
  - Usually occurs because something happened outside normal program execution
  - Can happen at any time, regardless of what the CPU is doing
  - Tells CPU to stop current process execution and load an interrupt handler

- CPU has an interrupt vector
  - Stores the memory address of handler for each type of interrupt
  - Populated by operating system when it first boots up
  - So the OS is responsible for handling and managing each interrupt
  - Interrupt handlers also known as interrupt service routines (ISR)

Inspecting Interrupts

- View interrupt vector easily in Windows or Linux
  - See the IRQ number and description
  - IRQ0 is the clock interrupt (quantum tick)





Context Switch

- The clock interrupt is triggered at the end of each time slice (quantum)
  - Operating system runs its scheduling algorithm to choose next process
  - This is known as a context switch

- During the context switch, the current state of the CPU (registers) is saved in a special data structure called the process control block (PCB)
  - Operating system stores a PCB in memory for each process
  - When process is placed onto CPU, the state of its registers is restored from its PCB

- The context switch needs time on the CPU
  - Wastes a few CPU cycles to perform the switch
  - But gives us the benefit of multitasking and maximum CPU usage
  - Without the problems associated with multiprogramming

Process Control Block

- The kernel maintains a PCB for every process
  - Usually stored in memory
  - Can also be represented as a file (see Linux lectures)

- Contains information such as…
  - Unique process ID
  - User ID of the process owner
  - Process state
  - Memory address of process
  - Accounting statistics (time used, etc.)
  - Resources allocated to process (open files, network connections, devices, etc.)
  - Register values from context switch (so CPU can be restored exactly)

Process States

- A process goes through possibly many state changes during its lifetime
- The operating system must keep track of these and update the PCB accordingly
  - Running – Currently being executed by the CPU (interrupted at end of quantum)
  - Ready – Able to run, but waiting for CPU to become available
  - Blocked – Waiting for I/O to complete
- There could be several (or many) processes in each of the ready and blocked states
  - Blocked processes are unavailable for despatch to the CPU
  - Ready processes selected for despatch according to a scheduling algorithm
- Processor manager is responsible for creating and terminating processes
  - Creation – Reserve memory for the process and its stack, set up process control block, initialise I/O channels, place process into the ready state
  - Termination – Close any open I/O channels, remove process control block, deallocate memory

State Changes