FIT1047 - Week 7

Operating Systems (Part 1)

MONASH University

Important People

Important People

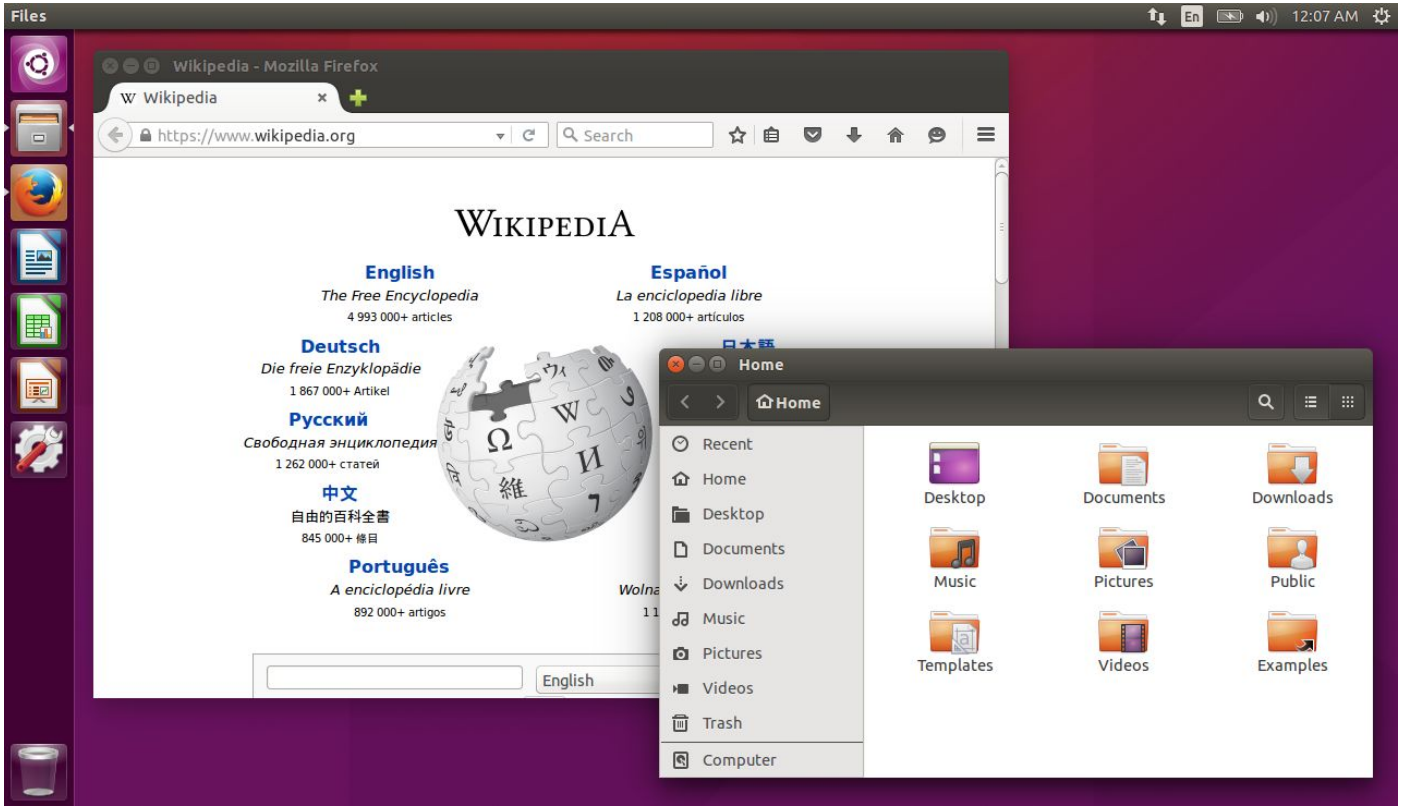
Important People

Important People

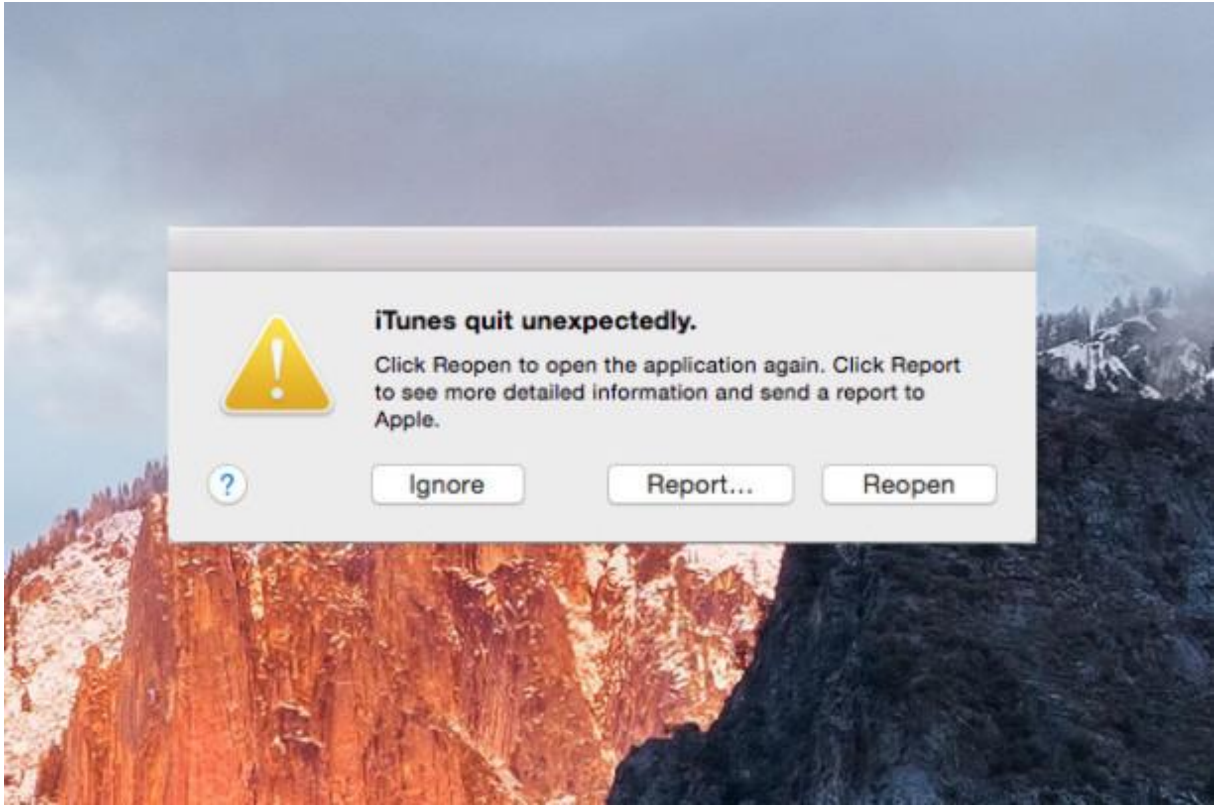
What does an OS do?

What does an OS do?

What does an OS do?

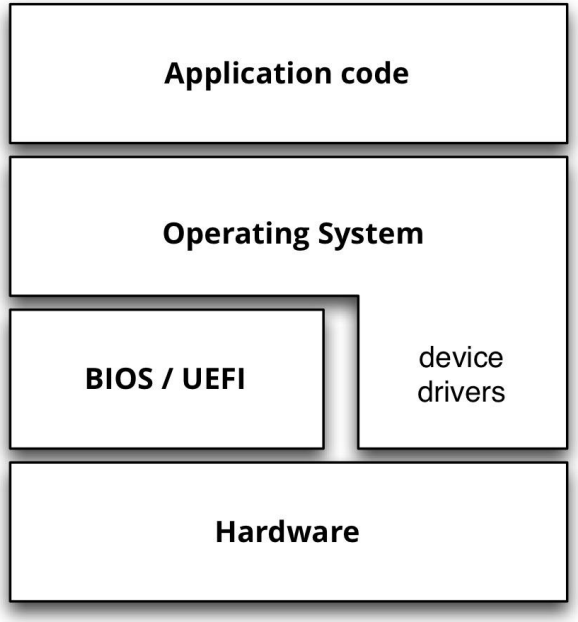


What does an OS do?



What does an OS do?

An OS is a **level of abstraction** between hardware and user software:



What does an OS do?

- Process management
  - A process is a running program
- Memory management
- I/O

(it does more, but that’s what we will cover)

A bit of history

First Operating Systems

- Just a **library**
  - A library is a collection of subroutines that programmers can include in their programs
- No support for **multiprogramming**
  - Only a single program could run at a time
- No **protection**
  - Running program could read/write entire memory and disk

Multiprogramming

- OS runs multiple processes “simultaneously”
- Improves CPU utilisation
- Important when computers were very expensive

This requires **protection:**

- Protect memory of other processes
- Protect files of other users

Gave rise to the **UNIX** family of operating systems

Modern Operating Systems

- Hundreds of processes running at any point in time
- Provide access to diverse hardware
- Full network support built-in
- Are somewhat related to UNIX
  - Linux: free re-implementation of UNIX
  - macOS: based on BSD UNIX
  - Windows: not directly related but heavily influenced

Goals of an Operating System

Main Goal: Ease of Use

For end users:

- Provide consistent **user interface**
- Manage **multiple applications** simultaneously
- **Protect** users from malicious or buggy code

For programmers:

- Provide **programming interface** (library subroutines)
- Enable access to **hardware and I/O**
- Manage **system resources** (memory, disk, network)

Introduce a **level of abstraction** between hardware and software.

Abstraction

The most important concept in IT!

- Hide complexity from users
- Provide clean, well-defined **interface** to functionality

Simple example: MARIE multiplication subroutine

- User does not need to know how it's implemented
- Provides a simple interface

Abstraction in OSs

Virtualisation

- Provide **virtual** form of each **physical** resource for each process

This means you can code as if your program

- Has the entire CPU to itself
- Has a large, contiguous block of memory just for itself
- Can use system resources through library functions
    - E.g. keyboard, graphics, disk, network

The OS kernel

Modern operating systems have many different functions.

We are only looking at the *core* functionality.

The part of the OS that implements this is called the **kernel.**

Virtualising the CPU

A CPU can only execute one instruction at a time.

How can we make it run several programs simultaneously?

Timesharing!

- OS kernel **switches** periodically between processes
- If switching is fast and occurs often, it creates the **illusion of concurrency**
- Illusion works both for **programmers and end users**

Managing processes

Mechanisms:

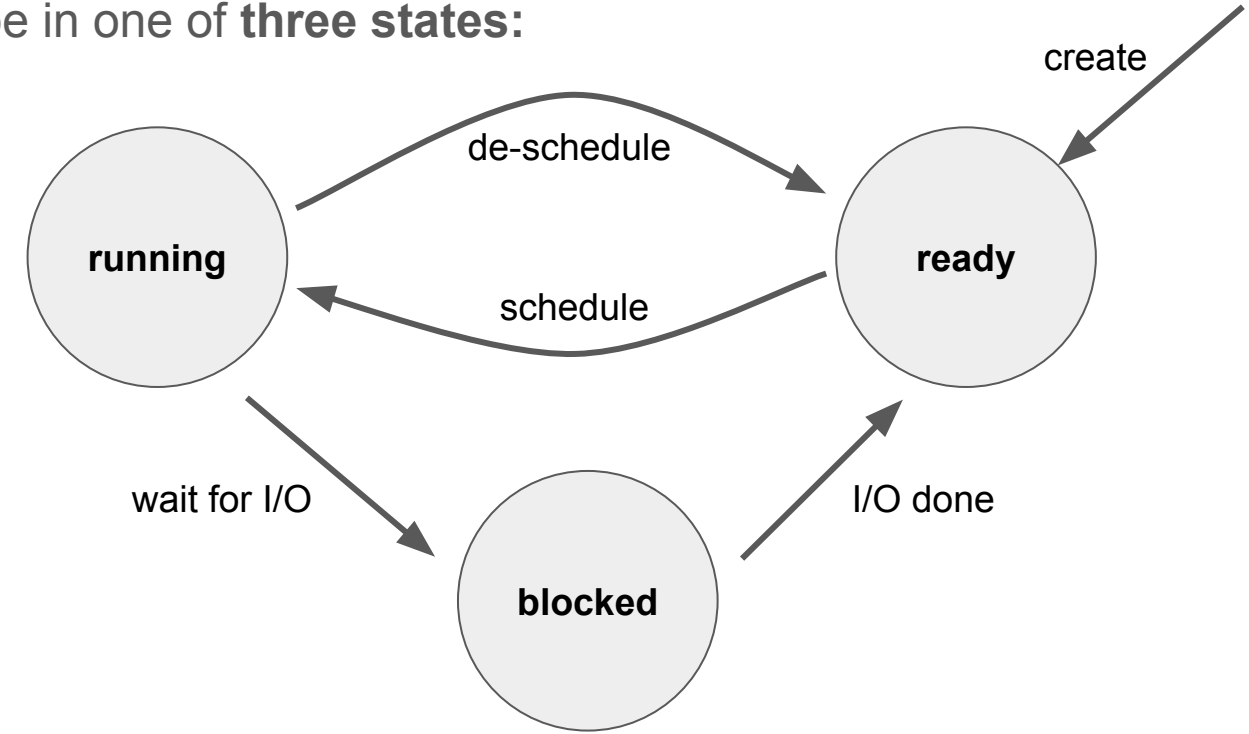
- **How** to virtualise the CPU
- Rest of this lecture

Policies:

- **When** to switch between processes
- Next lecture

Processes

- Created by loading code into memory
- Can be in one of **three states:**



Process states

Time	Media Player (MP)	Web browser (WB)	Description
1	Running	Ready	
2			
3			
4			
5			
6			
7			
8			
9			
10			

Process states

Time	Media Player (MP)	Web browser (WB)	Description
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	MP initiates I/O
4			
5			
6			
7			
8			
9			
10			

Process states

Time	Media Player (MP)	Web browser (WB)	Description
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	MP initiates I/O
4	Blocked	Running	Switch to WB
5	Blocked	Running	
6			
7			
8			
9			
10			

Process states

Time	Media Player (MP)	Web browser (WB)	Description
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	MP initiates I/O
4	Blocked	Running	Switch to WB
5	Blocked	Running	
6	Ready	Running	I/O finished
7	Ready	Running	
8			
9			
10			

Process states

Time	Media Player (MP)	Web browser (WB)	Description
1	Running	Ready	
2	Running	Ready	
3	Running	Ready	MP initiates I/O
4	Blocked	Running	Switch to WB
5	Blocked	Running	
6	Ready	Running	I/O finished
7	Ready	Running	
8	Running	Ready	Switch to MP
9	Running	Ready	
10	Ready	Running	Switch to WB

Challenges

Performance

- CPU virtualisation should not create huge overhead

Control

- OS must **stay in control**
- Enable **fair scheduling** (each process gets fair amount of time)
- **Protect** against malicious and buggy code

This requires hardware support!

Limited Direct Execution

Limited Direct Execution

Application code runs **directly on the CPU**

So while application is running, clearly the OS is not!

Two problems:

- How can the OS restrict what the program can do without affecting performance?
- How can the OS stop a process and switch to another one?

Restricting what programs can do

Solution: CPUs have **different modes!**

Kernel mode:

- Code runs without any restrictions
- The OS runs in kernel mode
- Any interrupt triggers a switch into kernel mode

User mode:

- Only a limited subset of instructions is allowed
- E.g. no I/O instructions
- Normal applications run in user mode

Remember:

Kernel mode: no restrictions

User mode: limited

Problem: how can a user application do I/O?

System calls

Special instructions that expose OS functionality to user programs.

Examples:

- Perform file I/O
- Access the network interface
- Communicate with other processes
- Allocate memory (we'll talk about that later)

System calls

OS has a table of system call handlers.

When a process makes a system call with number *n*,

- Save process context (registers etc) into memory
- Switch CPU to kernel mode
- Jump to handler *n* and execute it
- Restore process context
- Switch CPU to user mode
- Return to calling process

Sounds familiar?

Software Interrupts

Recap of interrupts:

- Hardware triggers flag in the CPU
- CPU jumps to special code and then returns to running program
- Context switch makes sure program continues as if no interrupt had occurred

To implement systems calls:

- Add an **instruction that causes an interrupt**
- Also called *software interrupt*

Summary: Limited Direct Execution

- Application code runs directly on the CPU
- But we need to **restrict what it can do**
- CPU has two modes
  - Kernel mode (no restrictions)
  - User mode (some instructions are not allowed)
- **System calls** enable user code to call OS functions

Process Switching

Process switching

Remember: User code runs directly on CPU, so while it's running, no OS code is running.

How can the OS regain control over a running process?

Solution: any **interrupt** switches from user code to kernel.

Recap from week 4: An interrupt performs a **context switch** and executes an **interrupt handler**.

Only difference with process switching: **return to a different process.**

Process switching

Two variants:

- Cooperative:
  - Switch into kernel when user code makes a **system call**, or when a **hardware interrupt** happens
- Preemptive:
  - Wait for a **timer interrupt**

Cooperative timesharing

OS regains control when user mode process makes a system call.

- OS then checks whether to switch processes
- If no, just handle the system call and return to running process
- If yes, put running process into *ready* state, switch some other process from *ready* state into *running* state

Properties of Cooperative Timesharing

- Easy to implement
- But what if processes don’t cooperate?
- E.g. infinite loop without system calls

Preemptive timesharing

OS sets up **timer interrupt**

- Timer is implemented in special hardware
- “Fires” e.g. every 10 milliseconds
- Interrupt switches into kernel mode and executes interrupt handler
- Interrupt handler is part of OS, so OS can then switch to different process

Properties of Preemptive Timesharing

- Needs hardware for timer interrupt
- Now OS always regains control in regular intervals!

Summary

- OS makes hardware easier and safer to use
- Virtualisation:
  - Makes it look as if each process has exclusive access to the hardware
  - CPU has user and kernel mode
- System calls give user code access to OS (file system, network, memory, …)
- Cooperative vs. preemptive timesharing

Next lecture

When to switch processes (process scheduling)

Virtual memory