Introduction to Operating Systems

OPS Lecture 2, G53OPS/G52OSC

Geert De Maere
(Jason Atkin – OSC)
Geert.DeMaere@Nottingham.ac.uk

University Of Nottingham
United Kingdom

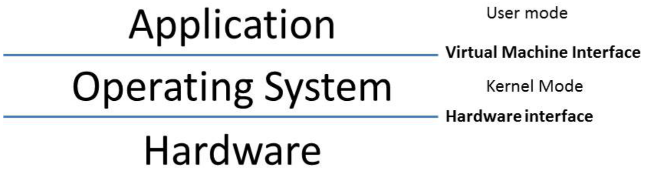
2015

Remarks
Questions after the last lecture

- A **permissions problem** on Moodle meant that the slides were inaccessible to G52OSC $\Rightarrow$ resolved
- We will make the **documentation for the G52OSC lab sessions** and **coursework** available to **G53OPS**
- For G52OSC, 3 out of 5 questions (2 CON, 3 OPS) for the exam with **at least one concurrency question** $\Rightarrow$ **at least one OPS question**

Recap
Last Lecture

1. The **code** we write heavily **uses operating system functionality**
2. The **operating systems provides abstractions** (e.g., hides away hardware details) and **manages resources**
3. **Multi-programming** results in “a few” complications, e.g. **context switches** and **deadlocks**



Goals for Today
Overview

1. **G51CSA** from an OPS point of view
2. Interrupts, address spaces and context switches
3. **OS Structures/implementation**

Computer Hardware
G51CSA in a nutshell

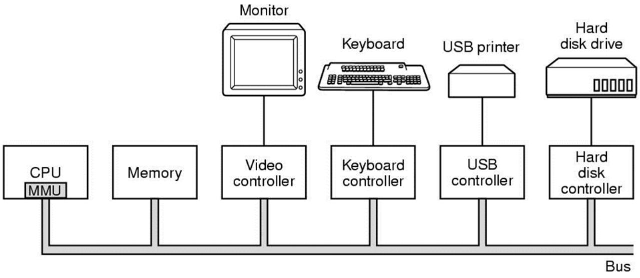


Figure: Simplified computer model (Tanenbaum, 2014)

Computer Hardware
CPU Design

- A CPU's basic cycle consist of **fetch**, **decode**, and **execute** (pipelines, or superscalar), and uses its own **instruction set**

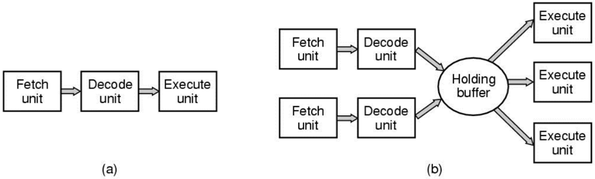


Figure: Processor Architectures (Tanenbaum, 2014)

- Each CPU has a set of **registers** used for **data** or for special functions (e.g. **program counter**, **program status word** – mode bit)
  - The compiler/programmer decides what to keep in the registers
- **Context switching** must save and restore the CPU's internal state, including its **registers**

Computer Hardware

Memory Management Unit

```
#include <stdio.h>

int iVar = 0;
main() {
    int i = 0;
    while(i < 10) {
      iVar++;
      sleep(2);
      printf("Address:%x; Value:%d\n",&iVar, iVar);
      i++;
    }
}
```

Computer Hardware
Memory Management Unit

- Modern computers use **logical** and **physical** memory addresses:
  - Every process has a logical address space – $[0, MAX_{64}]$ (theoretically)[1]
  - The machine has a physical address space – $[0, MAX]$ (MAX determined by the amount of physical memory)
- **Address translation** takes place in the memory management unit (MMU)
  - $physical = f(logical)$

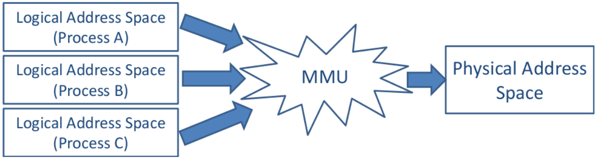

Logical Address Space (Process A)
Logical Address Space (Process B)
Logical Address Space (Process C)
MMU
Physical Address Space

Figure: Address Translation

[1] MAX_{64} = 16 exbibyte, 1 exbibyte = 260 bytes = 1152921504606846976bytes = 1024pebibytes

Computer Hardware
Memory Management Unit (Cont'ed)

- **Context switching** invalidates the MMU (and registers, and cache) $\rightarrow$ they must **save** and **restore** the CPU's internal state

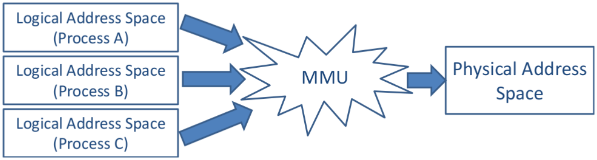
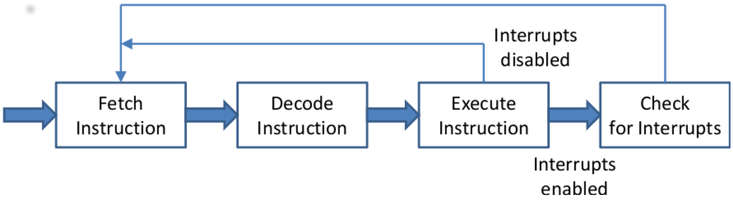

Logical Address Space (Process A)
Logical Address Space (Process B)
Logical Address Space (Process C)
MMU
Physical Address Space

Figure: Address Translation

Computer Hardware
Context Switching: Timer Interrupts

- Interrupts are responsible for **temporarily pausing** a process's normal operation
- Different types of interrupts exist, including:
  - Timer interrupts by **CPU clock**
  - **I/O interrupts** for **I/O completion** or error codes
  - **Software generated**, e.g. errors and exceptions
- **Context switches** can be initiated by **timer interrupts** after a **"set time"**



Defining Operating Systems

A Resource Manager (Timer Interrupts)

```
1. Timer generates an interrupt
2. CPU finishes current instruction and tests
      for interrupt
4. Transfer to interrupt service routine
  - Hardware saves current process state
      (PSW, program counter)
  - Set program counter to interrupt service routine
  - Save registers and other state information
6. Carry out interrupt service routine (scheduler)
7. Restore next process to run
```

Computer Hardware

Moore's "law"

Moore's law

"The number of transistors on an integrated circuit (chip) doubles roughly every two years"

- Closely linked, but **not necessarily related to performance** (performance roughly doubles until 2003)
- Moore's still continuing, but the **"power wall"** slows performance improvements of single core/single processor systems
  - A few cores for multiple "programs" is easy to justify
  - How to use **massively parallel** computers/CPUs/many core machines
  - Can we **extract parallelism automatically**, can we implement **parallelism at the lowest level** (similar to multiprogramming)

Computer Hardware

General Motors vs. Microsoft

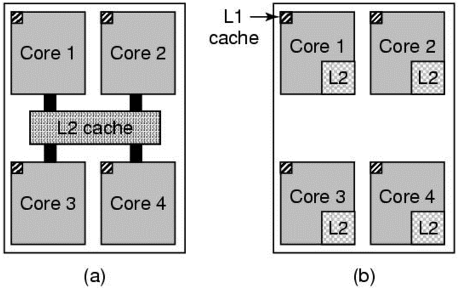
- Bill Gates:
  *“If GM had kept up with technology like the computer industry has, we would all be **driving $25.00*** cars that got **1,000 miles to the gallon**”
- General Motors:
  - *“For no reason whatsoever, your car would **crash twice a day**”*
  - *“The airbag system would ask "**Are you sure**?" before deploying. ”*
  - *“Occasionally, for no reason whatsoever, your **car would lock you** out and refuse to let you in until you simultaneously lifted the door handle, turned the key and grabbed hold of the radio antenna”*
  - *“You’d have **to press the "Start" button to turn the engine off.**”*
  - More at: http://mistupid.com/jokes/msvgm.htm

Figure: Volkswagen XL1 (Wikipedia)

Computer Hardware

Multi-core, hyperthreaded processors

- Modern CPUs contain **multiple cores** and are often **hyper-threaded**
- **Evolution in hardware** has implications on **operating system design**
    - XP did not support **multi processor architectures**
    - **Process scheduling** needs to account for **load balancing** and **CPU affinity**[2]
    - Cache **coherency** becomes important



[2] Last year's exam: Describe how, in your opinion, recent developments in computer architecture and computer design have influenced operating system design

Computer Hardware
Memory

- **Memory hierarchies** used to balance **cost** and **performance**
  - Fast and expensive memory is used for **caching**
  - Slow and inexpensive memory is used for **long term storage**
  - Memory includes, registers, L1/L2 cache, **main/core memory**, **disk**, etc.
- **L2 Cache** can be **shared** or **dedicated** to individual cores
- **Cache management** is mainly done by **hardware**
- The CPU can only **access main memory directly** (i.e. files have to be brought into memory first) **context switches**

Computer Hardware
I/O Devices

- **Device driver** interacts with the **controller**, controller interacts with the **device** (e.g., disk controller)
- The operating system/device driver typically **communicates with the controller through registers**
- I/O can take place through:
  - **Busy waiting**
  - **Interrupt** based
  - Direct memory access (using **DMA** chip)

Micro Kernels
Operating System Structure

- Operating Systems contain **a lot of functionality**, including:
  - Processes, process management, process scheduling, context switching, etc.
  - Memory management, virtual memory, etc.
  - File Systems
  - I/O Management
- $\Rightarrow$ How are they **logically organised**?
  - Micro kernels
  - Monolithic

Micro Kernels

Minimal Kernel Functionality

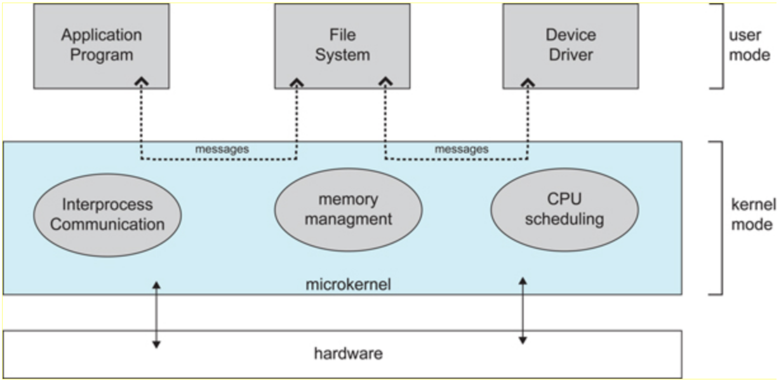


Figure: Structure of a Micro Kernel (Silberschatz, 2008)

Micro Kernels

Minimal Kernel Functionality

- All **non-essential functionality** are **extracted** from the kernel
  - **Communication**, **memory management** and **CPU scheduling** are likely to be included
  - The **file system**, **GUI**, **device drivers** are likely to be user processes
- Micro kernels are more **easy to extend**, more **portable**, and usually more **reliable**
- Frequent **system calls** and **kernel traps** cause significant **overhead**
- Some Unix version, Mac OS X, Minix, and early versions of Windows (NT4.0) were (partially) micro kernels

Monolithic Systems

The big mess

- All procedures are **linked together** into one **single executable** running in **kernel mode**
- Monolithic kernels are **difficult to maintain**
- Current versions of Windows, Linux are implemented as monolithic Kernels

Recap
Take-Home Message

- Operating Systems are heavily linked to **computer architecture**
- **Interrupts**, **address translation**
- **OS structures**

Next Lecture
Content

- Processes!