Introduction to Operating Systems

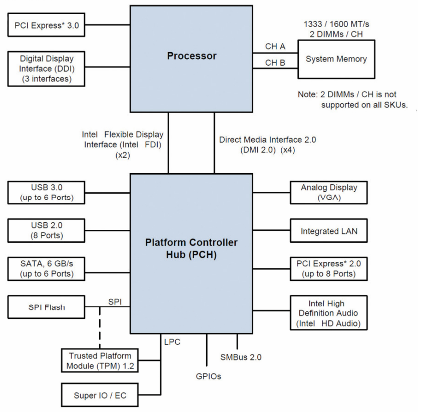
Chapter 1 – 1.3

Chapter 1.5 – 1.9

Learning Outcomes

- High-level understand what is an operating system and the role it plays
- A high-level understanding of the structure of operating systems, applications, and the relationship between them.
- Some knowledge of the services provided by operating systems.
- Exposure to some details of major OS concepts.

What is an Operating System?


PCI Express* 3.0
Digital Display Interface (DDI) (3 interfaces)
Processor
1333 / 1600 MT/s 2 DIMMs / CH
CH A
CH B
System Memory
Note: 2 DIMMs / CH is not supported on all SKUs.
Intel Flexible Display Interface (Intel FDI) (x2)
Direct Media Interface 2.0 (DMI 2.0) (x4)
USB 3.0 (up to 6 Ports)
USB 2.0 (8 Ports)
SATA, 6 GB/s (up to 6 Ports)
SPI Flash
SPI
Platform Controller Hub (PCH)
Analog Display (VGA)
Integrated LAN
PCI Express* 2.0 (up to 8 Ports)
Intel High Definition Audio (Intel HD Audio)
LPC
Trusted Platform Module (TPM) 1.2
Super IO / EC
GPIOs
SMBus 2.0

Block Diagram of Haswell Platform Architecture http://www.pcquest.com

Viewing the Operating System as an Abstract Machine

- Extends the basic hardware with added functionality
- Provides high-level abstractions
  - More programmer friendly
  - Common core for all applications
- It hides the details of the hardware
  - Makes application code portable

Disk

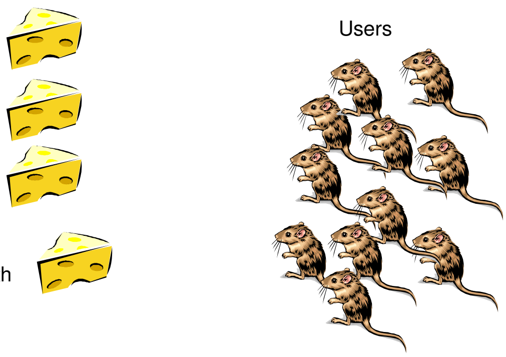
Memory

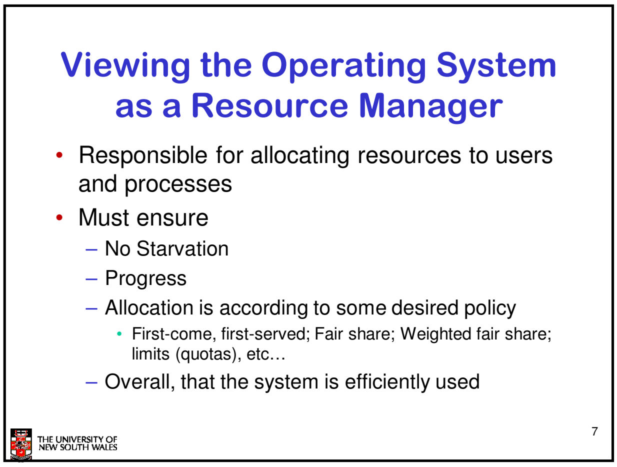
CPU

Network

Bandwidth

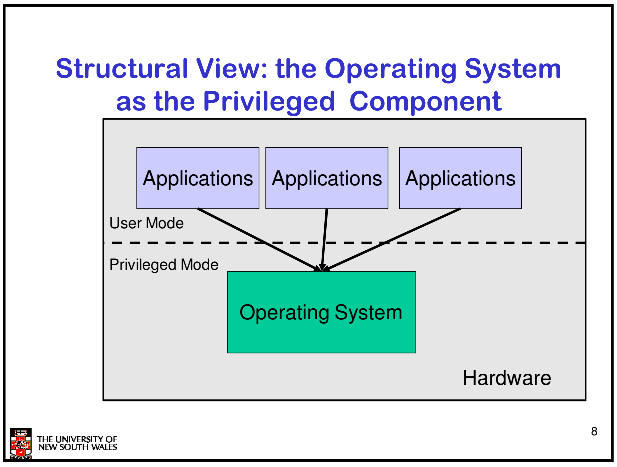
Users





Viewing the Operating System as a Resource Manager

- Responsible for allocating resources to users and processes
- Must ensure
  - No Starvation
  - Progress
  - Allocation is according to some desired policy
    - First-come, first-served; Fair share; Weighted fair share; limits (quotas), etc…
  - Overall, that the system is efficiently used



Structural View: the Operating System as the Privileged Component

Applications

Applications

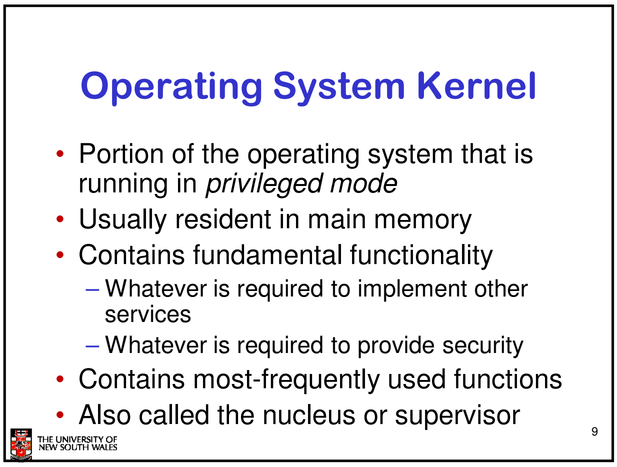
Applications

User Mode

Privileged Mode

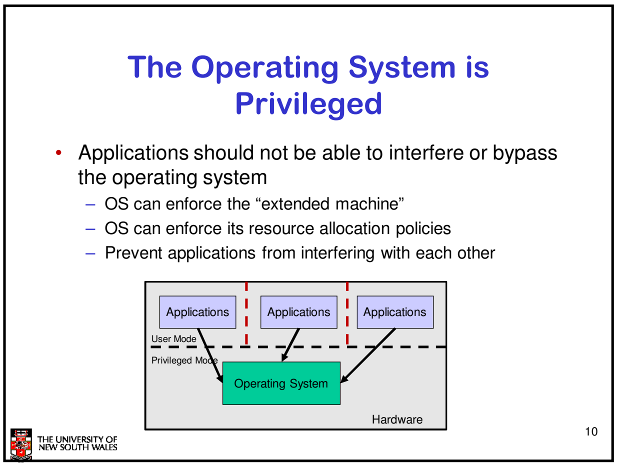
Operating System

Hardware



Operating System Kernel

- Portion of the operating system that is running in *privileged mode*
- Usually resident in main memory
- Contains fundamental functionality
  - Whatever is required to implement other services
  - Whatever is required to provide security
- Contains most-frequently used functions
- Also called the nucleus or supervisor



The Operating System is Privileged

- Applications should not be able to interfere or bypass the operating system
  - OS can enforce the "extended machine"
  - OS can enforce its resource allocation policies
  - Prevent applications from interfering with each other

Applications

Applications

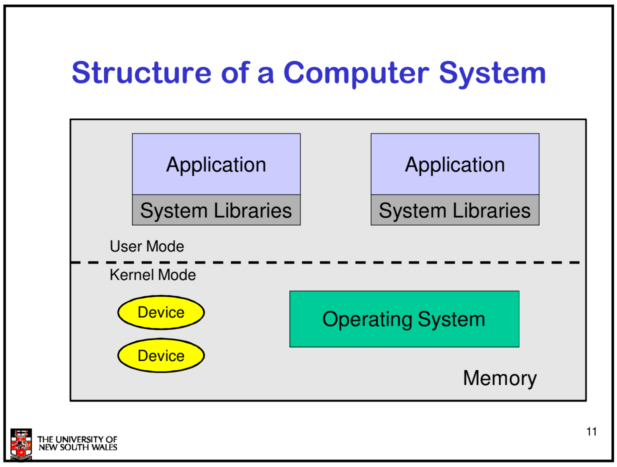
Applications

User Mode

Privileged Mode

Operating System

Hardware



Structure of a Computer System

Application

System Libraries

Application

System Libraries

User Mode

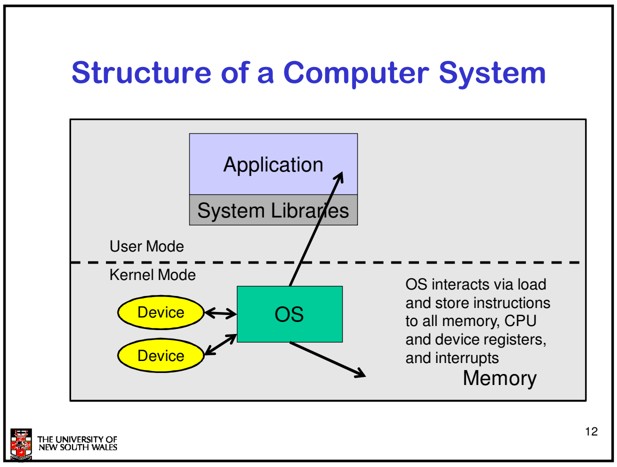
Kernel Mode

Device

Device

Operating System

Memory



Structure of a Computer System

Application

System Libraries

User Mode

Kernel Mode

Device

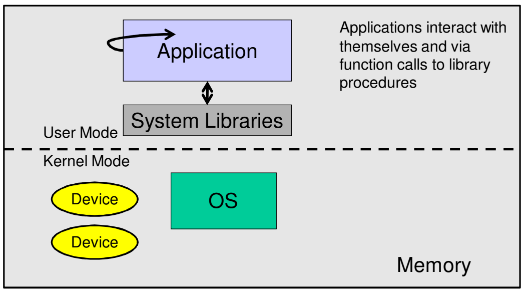
Device

OS

OS interacts via load and store instructions to all memory, CPU and device registers, and interrupts

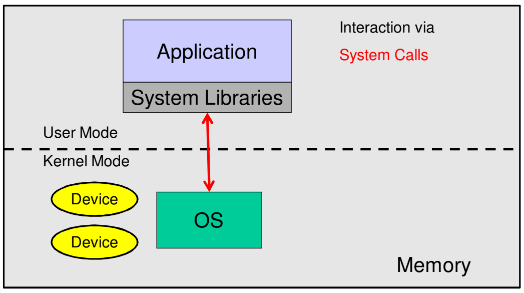
Memory

Structure of a Computer System



Applications interact with themselves and via function calls to library procedures

Structure of a Computer System

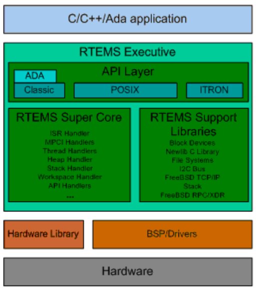


Interaction via

System Calls

Privilege-less OS

- Some Embedded OSs have no privileged component
  - e.g. PalmOS, Mac OS 9, RTEMS
  - Can implement OS functionality, but cannot enforce it.
    - All software runs together
    - No isolation
    - One fault potentially brings down entire system



A Note on System Libraries

System libraries are just that, libraries of support functions (procedures, subroutines)

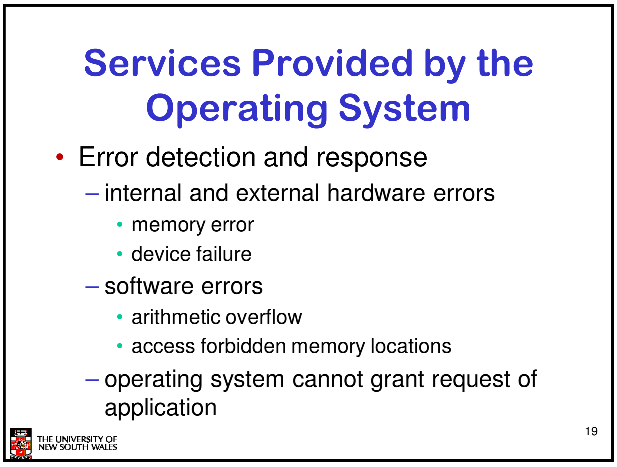
- Only a subset of library functions are actually systems calls
  - strcmp(), memcpy(), are pure library functions
    - manipulate memory within the application, or perform computation
  - open(), close(), read(), write() are system calls
    - they cross the user-kernel boundary, e.g. to read from disk device
    - Implementation mainly focused on passing request to OS and returning result to application
- System call functions are in the library for convenience
  - try `man syscalls` on Linux

Operating System Objectives

- Convenience
  - Make the computer more convenient to use
- Abstraction
  - Hardware-independent programming model
- Efficiency
  - Allows the computer system to be used in an efficient manner
- Ability to evolve
  - Permit effective development, testing, and introduction of new system functions without interfering with existing services
- Protection
  - allow only authorised access to data, computation, services, etc.

Services Provided by the Operating System

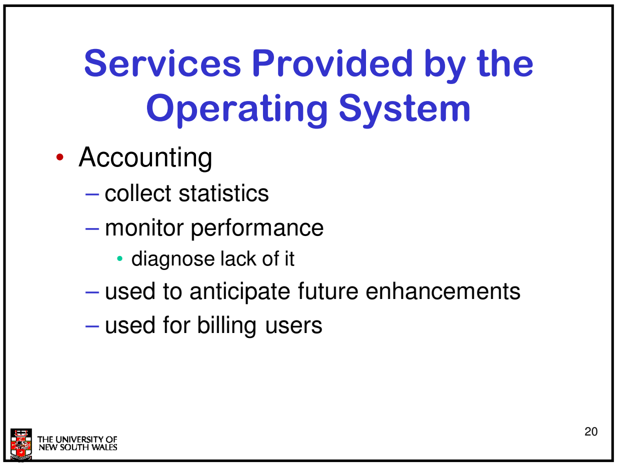
- Program execution
  - Load a program and its data
- Access to I/O devices
  - Display, disk, network, printer, keyboard, camera, etc.
- Controlled access to files
  - Access protection
- System access
  - User authentication



Services Provided by the Operating System

- Error detection and response
  - internal and external hardware errors
    - memory error
    - device failure
  - software errors
    - arithmetic overflow
    - access forbidden memory locations
  - operating system cannot grant request of application

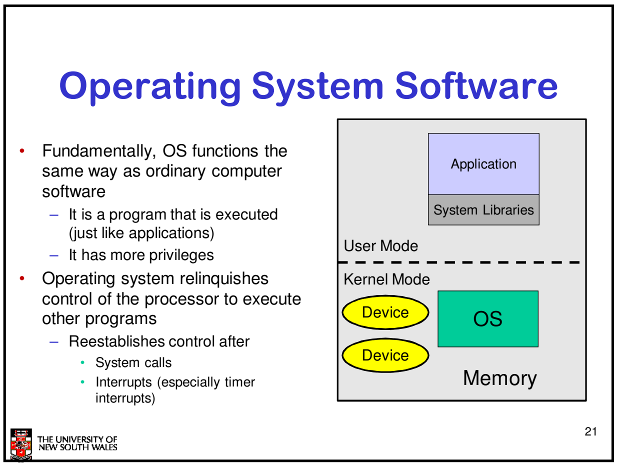
THE UNIVERSITY OF NEW SOUTH WALES



Services Provided by the Operating System

- Accounting
  - collect statistics
  - monitor performance
    - diagnose lack of it
  - used to anticipate future enhancements
  - used for billing users

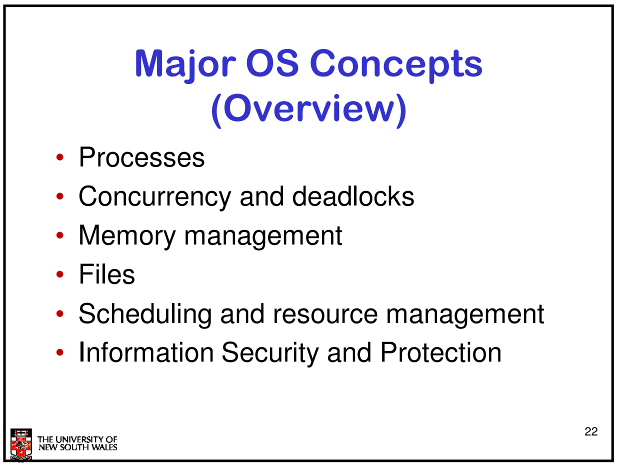
THE UNIVERSITY OF NEW SOUTH WALES



Operating System Software

- Fundamentally, OS functions the same way as ordinary computer software
  - It is a program that is executed (just like applications)
  - It has more privileges
- Operating system relinquishes control of the processor to execute other programs
  - Reestablishes control after
    - System calls
    - Interrupts (especially timer interrupts)

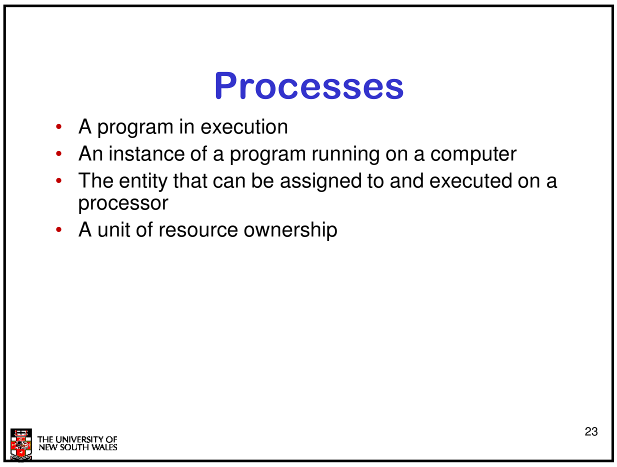
THE UNIVERSITY OF NEW SOUTH WALES



Major OS Concepts (Overview)

- Processes
- Concurrency and deadlocks
- Memory management
- Files
- Scheduling and resource management
- Information Security and Protection

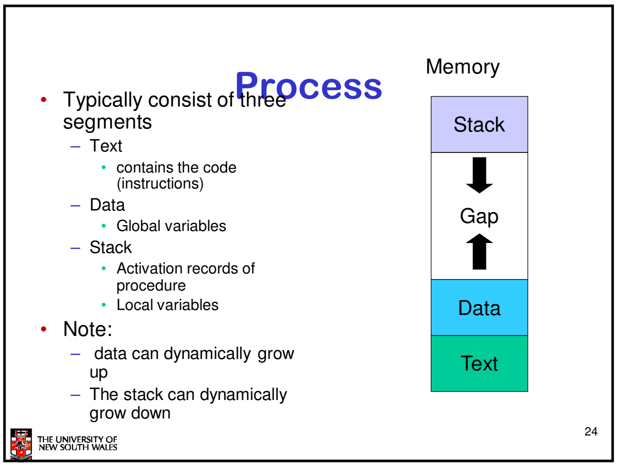
THE UNIVERSITY OF NEW SOUTH WALES



Processes

- A program in execution
- An instance of a program running on a computer
- The entity that can be assigned to and executed on a processor
- A unit of resource ownership

THE UNIVERSITY OF NEW SOUTH WALES



Process

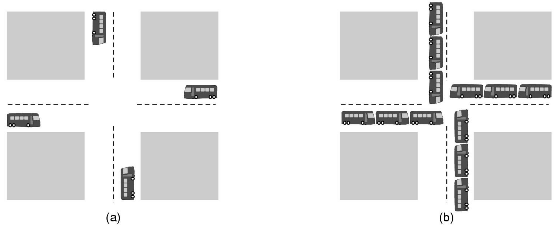
- Typically consist of three segments
  - Text
    - contains the code (instructions)
  - Data
    - Global variables
  - Stack
    - Activation records of procedure
    - Local variables
- Note:
  - data can dynamically grow up
  - The stack can dynamically grow down

THE UNIVERSITY OF NEW SOUTH WALES

Process

- Consists of three components
  - An executable program
    - text
  - Associated data needed by the program
    - Data and stack
  - Execution context of the program
    - All information the operating system needs to manage the process
      - Registers, program counter, stack pointer, OS-internal bookkeeping, etc…

Multiple processes creates concurrency issues



(a) A potential deadlock. (b) an actual deadlock.

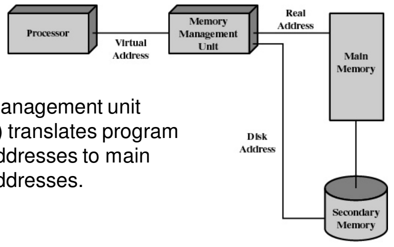
Memory Management

- The view from thirty thousand feet
  - Process isolation
    - Prevent processes from accessing each others data
  - Automatic allocation and management
    - Don't want users to deal with physical memory directly
  - Protection and access control
    - Still want controlled sharing
  - Long-term storage
  - OS services
    - Virtual memory
    - File system

Virtual Memory

- Allows programmers to address memory from a logical point of view
  - Gives apps the illusion of having RAM to themselves
  - Logical addresses are independent of other processes
  - Provides isolation of processes from each other
- Can overlap execution of one process while swapping in/out others to disk.

Virtual Memory Addressing



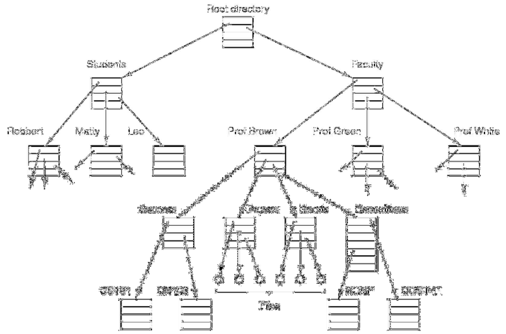
Memory management unit (hardware) translates program memory addresses to main memory addresses.

Figure 2.10 Virtual Memory Addressing

File System

- Implements long-term store
- Information stored in named objects called files

Example File System



Information Protection and Security

- Access control
  - regulate user access to the system
  - Involves authentication
- Information flow control
  - regulate flow of data within the system and its delivery to users

Scheduling and Resource Management

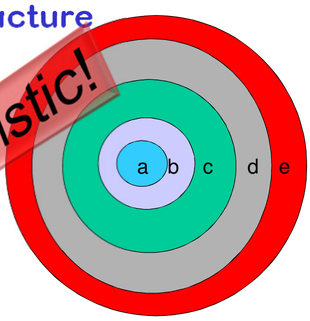
- Fairness
  - give equal and fair access to all processes
- Differential responsiveness
  - discriminate between different classes of jobs
- Efficiency
  - maximize throughput, minimize response time, and accommodate as many uses as possible

Operating System Internal Structure?

Classic Operating System Structure

- The layered approach
  a) Processor allocation and multiprogramming
  b) Memory Management
  c) Devices
  d) File system
  e) Users
- Each layer depends on the inner layers

Unrealistic!

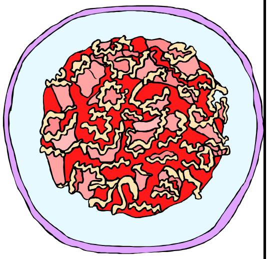


Operating System Structure

- In practice, layering is only a guide
  - Operating Systems have many interdependencies
    - Scheduling on virtual memory
    - Virtual memory on I/O to disk
    - VM on files (page to file)
    - Files on VM (memory mapped files)
    - And many more…

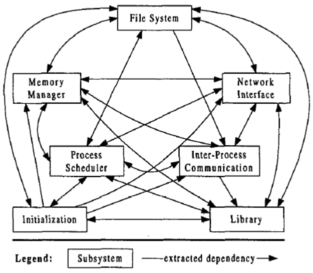
The Monolithic Operating System Structure

- Also called the “spaghetti nest” approach
  - Everything is tangled up with everything else.
- Linux, Windows, ….

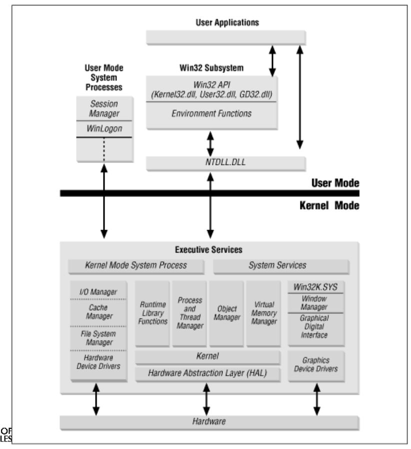


The Monolithic Operating System Structure

- However, some reasonable structure usually prevails



Bowman, I. T., Holt, R. C., and Brewster, N. V. 1999. Linux as a case study: its extracted software architecture. In *Proceedings of the 21st international Conference on Software Engineering* (Los Angeles, California, United States, May 16 - 22, 1999). ICSE '99. ACM, New York, NY, 555-563. DOI= http://doi.acm.org/10.1145/302405.302691



Computer Hardware Review

Chapter 1.4

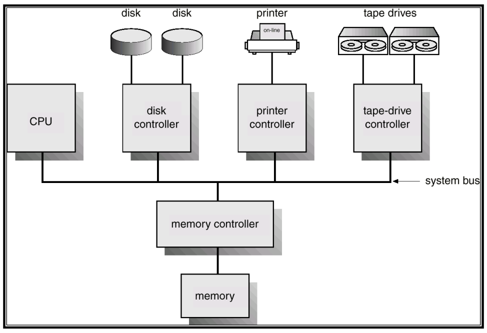
Learning Outcomes

- Understand the basic components of computer hardware
  - CPU, buses, memory, devices controllers, DMA, Interrupts, hard disks
- Understand the concepts of memory hierarchy and caching, and how they affect performance.

Operating Systems

- Exploit the hardware available
- Provide a set of high-level services that represent or are implemented by the hardware.
- Manages the hardware reliably and efficiently
- *Understanding operating systems requires a basic understanding of the underlying hardware*

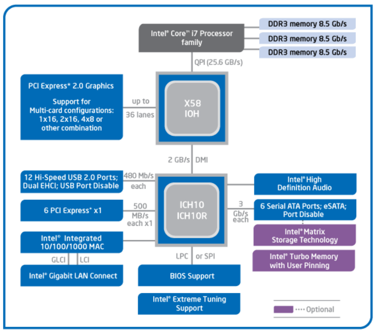
Basic Computer Elements



Basic Computer Elements

- CPU
  - Performs computations
  - Load data to/from memory via system bus
- Device controllers
  - Control operation of their particular device
  - Operate in parallel with CPU
  - Can also load/store to memory (Direct Memory Access, DMA)
  - Control register appear as memory locations to CPU
    - Or I/O ports
  - Signal the CPU with “interrupts”
- Memory Controller
  - Responsible for refreshing dynamic RAM
  - Arbitrating access between different devices and CPU

The real world is logically similar, but more complex


Intel® Core™ i7 Processor family
DDR3 memory 8.5 Gb/s
DDR3 memory 8.5 Gb/s
DDR3 memory 8.5 Gb/s
QPI (25.6 GB/s)
PCI Express* 2.0 Graphics
Support for Multi-card configurations: 1x16, 2x16, 4x8 or other combination
up to 36 lanes
X58 IOH
2 GB/s DMI
12 Hi-Speed USB 2.0 Ports; Dual EHCI; USB Port Disable
480 Mb/s each
6 PCI Express* x1
500 MB/s each x1
Intel® Integrated 10/100/1000 MAC
GLCI
LCI
Intel® Gigabit LAN Connect
ICH10 ICH10R
LPC or SPI
BIOS Support
Intel® Extreme Tuning Support
Intel® High Definition Audio
3 Gb/s each
6 Serial ATA Ports; eSATA; Port Disable
Intel® Matrix Storage Technology
Intel® Turbo Memory with User Pinning
Optional

Intel® X58 Express Chipset Block Diagram

A Simple Model of CPU Computation

- The fetch-execute cycle

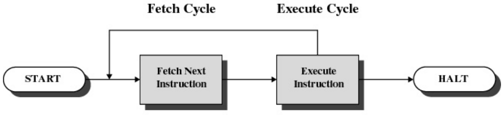


Figure 1.2 Basic Instruction Cycle

A Simple Model of CPU Computation

- The fetch-execute cycle
  - Load memory contents from address in program counter (PC)
    - The instruction
  - Execute the instruction
  - Increment PC
  - Repeat

CPU Registers
PC: 0x0300
SP: 0xcbf3
Status
R1
↕
Rn

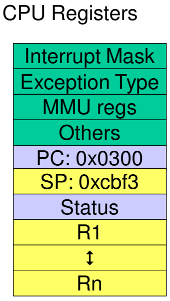
A Simple Model of CPU Computation

- Stack Pointer
- Status Register
  - Condition codes
    - Positive result
    - Zero result
    - Negative result
- General Purpose Registers
  - Holds operands of most instructions
  - Enables programmers (compiler) to minimise memory references.

CPU Registers
PC: 0x0300
SP: 0xcbf3
Status
R1
↕
Rn

Privileged-mode Operation

- To protect operating system execution, two or more CPU modes of operation exist
  - Privileged mode (system-, kernel-mode)
    - All instructions and registers are available
  - User-mode
    - Uses 'safe' subset of the instruction set
      - E.g. no disable interrupts instruction
    - Only 'safe' registers are accessible



'Safe' registers and instructions

- Registers and instructions are safe if
  - Only affect the state of the application itself
  - They cannot be used to uncontrollably interfere with
    - The operating system
    - Other applications
  - They cannot be used to violate a correctly implemented operating system.

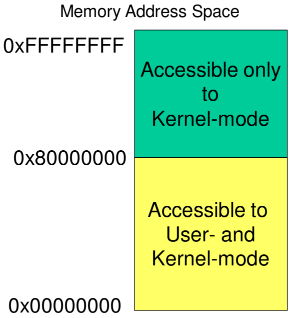
Example Unsafe Instruction

- "cli" instruction on x86 architecture
  - Disables interrupts
- Example exploit

```
cli /* disable interrupts */
while (true)
   /* loop forever */;
```

Privileged-mode Operation

- The accessibility of addresses within an address space changes depending on operating mode
  - To protect kernel code and data
- Note: The exact memory ranges are usually configurable, and vary between CPU architectures and/or operating systems.


Memory Address Space
0xFFFFFFFF
Accessible only to Kernel-mode
0x80000000
Accessible to User- and Kernel-mode
0x00000000

I/O and Interrupts

- I/O events (keyboard, mouse, incoming network packets) happen at unpredictable times
- How does the CPU know when to service an I/O event?

Interrupts

- An interruption of the normal sequence of execution
- A suspension of processing caused by an event external to that processing, and performed in such a way that the processing can be resumed.
- Improves processing efficiency
  - Allows the processor to execute other instructions while an I/O operation is in progress
  - Avoids unnecessary completion checking (polling)

Interrupt Cycle

- Processor checks for interrupts
- If no interrupts, fetch the next instruction
- If an interrupt is pending, divert to the interrupt handler

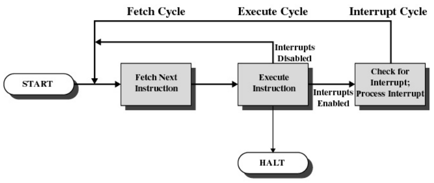


Figure 1.7 Instruction Cycle with Interrupts

Interrupt Terminology

- Program *exceptions*

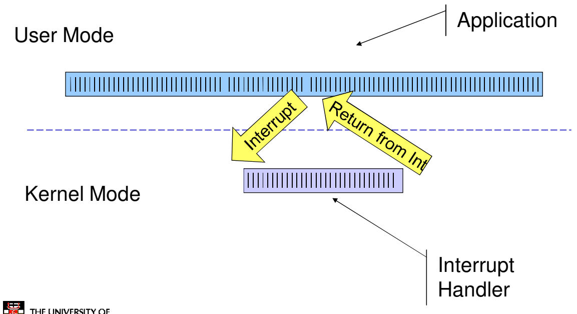
  (sometimes called *synchronous interrupts*)
  - Arithmetic overflow
  - Division by zero
  - Executing an illegal/privileged instruction
  - Reference outside user's memory space.
- Asynchronous (external) interrupts

  (usually just called *interrupts*)
  - Timer
  - I/O
  - Hardware or power failure

Interrupt Handler

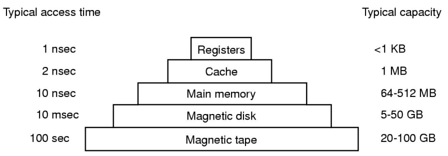
- A software routine that determines the nature of the interrupt and performs whatever actions are needed.
- Control is transferred to the handler by *hardware*.
- The handler is generally part of the operating system.

Simple Interrupt

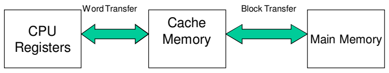


Memory Hierarchy

- Going down the hierarchy
  - Decreasing cost per bit
  - Increasing capacity
  - Increasing access time
  - Decreasing frequency of access to the memory by the processor
    - Hopefully
    - Principle of locality!!!!!

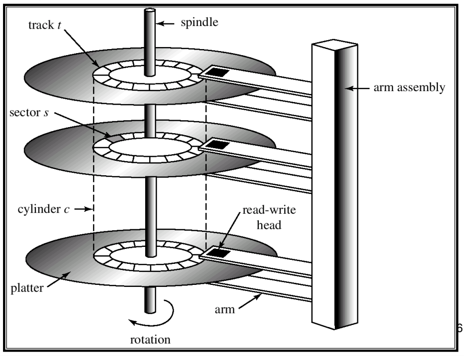


CPU Cache



- CPU cache is fast memory placed between the CPU and main memory
  - 1 to a few cycles access time compared to RAM access time of tens – hundreds of cycles
- Holds recently used data or instructions to save memory accesses.
- Matches slow RAM access time to CPU speed if high hit rate
- Is hardware maintained and (mostly) transparent to software
- Sizes range from few kB to several MB.
- Usually a hierarchy of caches (2–5 levels), on- and off-chip.
- Block transfers can achieve higher transfer bandwidth than single words.
  - Also assumes probability of using newly fetch data is higher than the probability of reusing ejected data.

Moving-Head Disk Mechanism



Example Disk Access Times

- Disk can read/write data relatively fast
  - 15,000 rpm drive - 80 MB/sec
  - 1 KB block is read in 12 microseconds
- Access time dominated by time to locate the head over data
  - Rotational latency
    - Half one rotation is 2 milliseconds
  - Seek time
    - Full inside to outside is 8 milliseconds
    - Track to track .5 milliseconds
- 2 milliseconds is 164KB in “lost bandwidth”

A Strategy: Avoid Waiting for Disk Access

- Keep a subset of the disk’s data in main memory

⇒ Main memory acts as a *cache* of disk contents

Caching as a general technique

- Given a two-levels data storage: small and fast, versus large and slow,
  - cache memory and main memory (RAM)
  - main memory and disk
  - Local disk versus the cloud.
- Can speed access to slower data by using faster memory as a cache.
- What is the effective access time?
- Answer: It depends on the hit rate in the first level.

Effective Access Time

$$T_{eff} = H \times T_1 + (1 - H) \times T_2$$

T_1 = access time of memory 1

T_2 = access time of memory 2

H = hit rate in memory 1

T_{eff} = effective access time of system

Example

- Cache memory access time 1ns
- Main memory access time 10ns
- Hit rate of 95%

$$T_{eff} = 0.95 \times 10^{-9} +$$
$$(1 - 0.95) \times (10^{-9} + 10 \times 10^{-9})$$
$$= 1.5 \times 10^{-9}$$