CAB403 Study Guide | 2023 Semester 1

Timothy Chappell | Notes for CAB403 at the Queensland University of Technology

Unit Description

Disclaimer

Everything written here is based off the QUT course content. However, there are at times parts of text that are taken from the QUT slides and most of the examples are directly from the course slides (these are referenced when done). This content is designed only for those currently studying an IT degree at QUT, do not share these resources with anyone outside of this community.

If any member of the QUT staff or a representative of such finds any issue with these guides please contact me at jeynesbrook@gmail.com and I will take these down without an argument. The last thing I want to do is cause any issues or damages to the QUT name or QUT resources. I am simply just trying to help students out with presenting the content in an easy to digest manor.

Week 1

Operating Systems

What is an Operating System

An operating system is a program that acts as an intermediary between a user of a computer and the computer hardware. It's acts as a resource allocator managing all resources and decides between conflicting requests for efficient and fair resource use. An OS also controls the execution of programs to prevent errors and improper use of the computer.

The operating system is responsible for:

- Executing programs
- Make solving user problems easier
- Make the computer system convenient to use
- Use the computer hardware in an efficient manner

Computer System Structure

Computer systems can be divided into four main components

1. **Hardware**: These items provide basic computing resources, i.e. CPU, memory, I/O devices.
2. **Operating system**: Controls and coordinates the use of hardware among various applications and users.
3. **Application programs**: These items define the ways in which the system resources are used to solve the computing problems of the use, i.e. word processors, compilers, web browsers, database systems, video games.
4. **Users**: People, machines, or other computers.

Computer Startup

A bootstrap program is loaded at power-up or reboot. This program is typically stored in ROM or EPROM and is generally know as firmware. This bootstrap program is responsible for initialising all aspects of the system, loading the operating system kernal, and starting execution.

Computer System Organisation

- I/O devices and the CPU can execute concurrently.
- Each device controller is in charge of a particular device type and has a local buffer.
- The CPU moves data from/to the main memory to/from local buffers.
- I/O is from the device to the local buffer of a particular controller.
- The device controller informs the CPU that it has finished its operation by causing an interrupt.

Common Functions of Interrupts

Operating systems are interrupt driven. Interrupts transfer control to the interrupt service routine. This generally happens through the interrupt vector which contains the addresses of all the service routines. The interrupt architecture must save the address of the interrupted instruction.

A trap or exception is a software-generated interrupt caused by either an error or a user request.

Interrupt Handling

The operating systems preserves the state of the CPU by storing registers and the program counter. It then determines which type of interrupt occured,

polling or vectored interrupt system.

Once determined what caused the interrupt, separate segments of code determine what action should be taken for each type of interrupt.

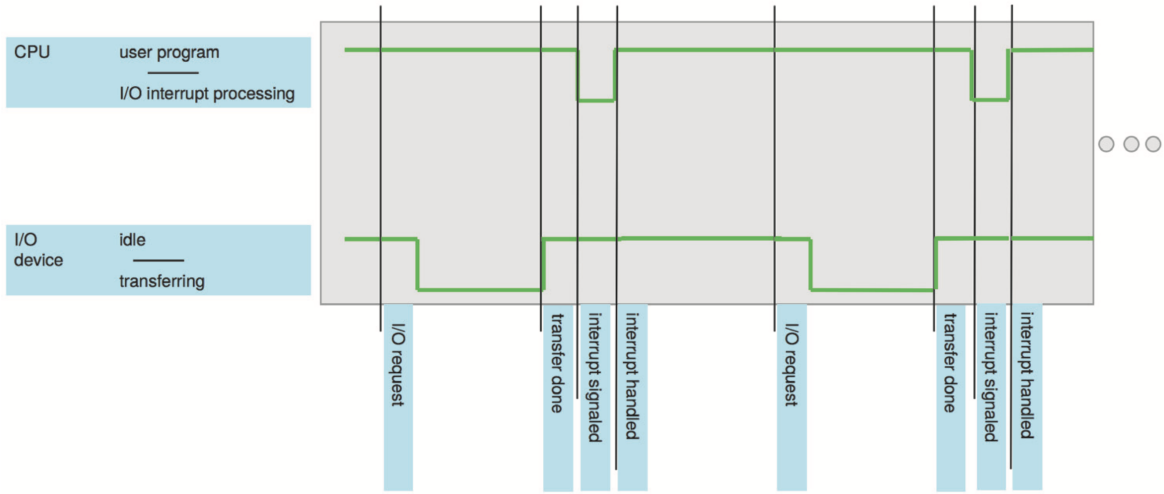

Figure: Interrupt timeline for a single program doing output.

I/O Structure

There are two ways I/O is usually structured:

1. After I/O starts, control returns to the user program only upon I/O completion.
    - Wait instructions idle the CPU until the next interrupt.
    - At most, one I/O request is outstanding at a time. This means no simultaneous I/O processing.
2. After I/O starts, control returns to the user program without waiting for I/O completion.
    - **System call**: Request to the OS to allow users to wait for I/O completion.
    - A **device-status table** containes entries for each I/O device indicating its type, address, and state.
    - The OS indexes into the I/O device table to determine the device status and to modify a table entry to include an interrupt.

Storage Definitions and Notation Review

The basic unit of computer storage is a bit. A bit contains one of two values, 0 and 1. A byte is 8 bits, and on most computers is the smallest convenient chunck of storage.

- A kilobyte, or KB, is $1,024$ bytes
- A megabyte, or MB, is $1,024^2$ bytes
- A gigabyte, or GB, is $1,024^3$ bytes
- A terabyte, or TB, is $1,024^4$ bytes
- A petabyte, or PB, is $1,024^5$ bytes

Direct Memory Access Structure

This method is used for high-speed I/O devices able to transmit information at close to memory speeds. Device controllers transfer blocks of data from buffer storage directly to main memory without CPU intervention. This means only one interrupt is generated per block rather than the one interrupt per byte.

Storage Structure

- **Main memory**: Only large storage media that the CPU can access directly.
    - Random access
    - Typically volatile
- **Secondary storage**: An extension of main memory that provides large non-volatile storage capacity.
- **Magnetic discs**: Rigid metal or glass platters covered with magnetic recording material. The disk surface is logically divided into tracks which are sub-diveded into sectors. The disk controller determines the logical interaction between the device and the computer.
- **Solid-state disks**: Achieves faster speeds than magnetic disks and non-volatile storage capacity through various technologies.

Storage Hierarchy

Storages systems are organised into a hierarchy:

- Speeds
- Cost
- Volatility.

There is a device driver for each device controller used to manage I/O. They provide uniform interfaces between controllers and the kernal.

Caching

Caching allows information to be copied into a faster storage system. The main memory can be viewed as a cache for the secondary storage.

Faster storage (cache) is checked first to determine if the information is there:

- If so, information is used directly from the cache
- If not, data is copied to the cache and used there

The cache is usually smaller and more expensive that the storage being cached. This means cache management is an important design problem.

Computer-System Architecture

Most systems use a single general-purpose processor. However, most systems have special-purpose processors as well.

Multi-processor systems, also known as parallel systems or tightly-coupled systems, usually come in two types; Asymmetric Multi-processing or Symmetric Multi-processor. Multi-processor systems have a few advantages over a single general-purpose processor:

- Increase throughput

- Economy of scale
- Increased reliability, i.e. graceful degradation or fault tolerance

Clustered Systems

Clustered systems are like Multi-processor systems, they have multiple systems working together.

- These systems typically share storage via a storage-area network (SAN).
- Provide a high-availability service which survices failures:
    - Asymmetric clustering have one machine in hot-standby mode.
    - Symmetric clustering have multiple nodes running applications, monitoring each other.
- Some clusters are for high-performance computing (HPC). Applications running on these clusters must be written to use parallelisation.
- Some have a distributed lock manager (DLM) to avoid conflicting operations.

Operating System Structure

Multi-programming organises jobs (code and data) so the CPU always has one to execute. This is needed for efficiency as a single user cannot keep a CPU and I/O devices busy at all times. Multi-programming works by keeping a subset of total jobs in the system, in memory. One job is selected and run via job scheduling. When it has to wait (for I/O for example), the OS will switch to another job.

Timesharing is a logical extension in which the CPU switches jobs so frequently that users can interact with each job while it is running.

- The response time should be less than one second.
- Each user has at least one program executing in memory (process).
- If processes don't fit in memory, swapping moves them in and out to run.
- Virtual memory allows execution of processes not completely in memory.

- If several jobs are ready to run at the same time, the CPU scheduler handles which to run.

Operating-System Operations

Dual-mode operations (user mode and kernal mode) allow the OS to protect itself and other system components. A mode bit provided by the hardware provides the ability to distinguish when a system is running user code or kernel code. Some instructions are designated as privileged and are only executable in kernal mode. System calls are used to change the mode to kernal, a return from call resets the mode back to user.

Most CPUs also support multi-mode operations, i.e. virtual machine manages (VMM) mode for guest VMs.

Input and Output

`printf()`

`printf()` is an output function included in `stdio.h`. It outputs a character stream to the standard output file, also known as `stdout`, which is normally connected to the screen.

It takes 1 or more arguments with the first being called the control string.

Format specifications can be used to interpolate values within the string. A format specification is a string that begins with `%` and ends with a conversion character. In the above example, the format specifications `%s` and `%d` were used. Characters in the control string that are not part of a format specification are placed directly in the output stream; characters in the control string that are format specifications are replaced with the value of the corresponding argument.

Example 1: Output with `printf()`

```
printf("name: %s, age: %d\n", "John", 24); // "name: John, age: 24"
```

`scanf()`

`scanf()` is an input function included in `stdio.h`. It reads a series of characters from the standard input file, also known as `stdin`, which is normally connected to the keyboard.

It takes 1 or more arguments with the first being called the control string.

Example 2: Reading input with `scanf()`

```
char a, b, c, s[100];
int n;
double x;

scanf("%c%c%c%d%s%lf", &a, &b, &c, &n, n, &x);
```

Relevant Links

- cppreference - printf
- cppreference - scanf

Pointers

A pointer is a variable used to store a memory address. They can be used to access memory and manipulate an address.

Example 1: Various ways of declaring a pointer

```c
// type *variable;

int *a;
int *b = 0;
int *c = NULL;
int *d = (int *) 1307;

int e = 3;
int *f = &e; // `f` is a pointer to the memory address of `e`
```

Example 2: Dereferencing pointers

```c
int a = 3;
int *b = &a;

printf("Values: %d == %d\nAddresses: %p == %p\n", *b, a, b, &a);
```

Functions

A function construct in C is used to write code that solves a (small) problem. A procedural C program is made up of one or more functions, one of them being `main()`. A C program will always begin execution with `main()`.

Function parameters can be passed into a function in one of two ways; pass by value and pass by reference. When a parameter is passed in via value, the data for the parameters are copied. This means any changes to said variables within the function will not affect the original values passed in. Pass by reference on the other hand passes in the memory address of each variable into the function. This means that changes to the variables within the function will affect the original variables.

Example 1: Function control

```c
#include <stdio.h>

void prn_message(const int k);

int main(void) {
    int n;

    printf("There is a message for you.\n");
    printf("How many times do you want to see it?\n");

    scanf("%d", &n);

    prn_message(n);

    return 0;
}

void prn_message(const int k) {
    printf("Here is the message:\n");

    for (size_t i = 0; i < k; i++) {
        printf("Have a nice day!\n");
    }
}
```

Example 2: Pass by values

```c
#include <stdio.h>

void swapx(int a, int b);

int main(void) {
    int a = 10;
    int b = 20;

    // Pass by value
    swapx(a, b);

    printf("within caller - a: %d, b: %b\n", a, b); // "within
caller - a: 10, b: 20"

    return 0;
}

void swapx(int a, int b) {
    int temp;

    temp = a;
    a = b;
    b = temp;

    printf("within function - a: %d, b: %b\n", a, b); // "within
function - a: 20, b: 10"
}
```

Example 3: Pass by value

```c
#include <stdio.h>

void swapx(int *a, int *b);

int main(void) {
    int a = 10;
    int b = 20;

    // Pass by reference
    swapx(&a, &b);

    printf("within caller - a: %d, b: %b\n", a, b); // "within
caller - a: 20, b: 10"

    return 0;
}

void swapx(int *a, int *b) {
    int temp;

    temp = *a;
    *a = *b;
    *b = temp;

    printf("within function - a: %d, b: %b\n", *a, *b); // "within
function - a: 20, b: 10"
}
```