**COMPSCI 3SH3 Winter, 2021**
**Student Name:** Jatin Chowdhary
**Mac ID:** Chowdhaj
**Student #:** 400033011
**Date:** February 14th, 2021

# Assignment #1

## 1. a)

Copying code, and information in general, is unethical. However, if you seek out the permission of the author, and get his approval, then copying code is acceptable. In addition, you should give full credits to the author, and write it somewhere noticeable. You should also notify your team that you obtained the code from someone else, but with their permission. Furthermore, you should review the code, ensure that there are no bugs, it works nicely with your code, and that you thoroughly understand it.

## 1. b)

Implementing code that you didn't write into your system is not good engineering practice. However, if you take the time to understand the code, how it works, and its pitfalls, then you can incorporate it into your system after making some adjustments. These adjustments should modify the code to adhere to good engineering practices. Furthermore, you should make sure that the code does not conflict with the standards agreed upon by your team, and you should make sure that it does not conflict with the design specification. (i.e. Team wants to do X, but the code does Y)

## 2.1)

An interrupt transfers control to the interrupt service routine. Generally, this is done through the interrupt vector. When an interrupt occurs, the flow of execution and hardware resources are transferred to another process. Operating systems are interrupt driven! The purpose of interrupts is to signal to the operating system that an action needs to be taken on an event. This event could be an exception generated by a process, a process requesting I/O data, a process signalling its completion, etc.

Traps, not to be confused with Mach traps, are software generated interrupts caused by an error or a user request. For example: dividing by zero and accessing the 11th element in an array with a size of 2 will cause an exception (in Java). Traps can be intentionally generated by a user program for I/O devices. For example: when a program says "press any key to continue" or when your PS3 controller dies and the system pauses your game so you can connect a charger or another controller.

## 2.2 a)

In direct memory access, the CPU is only involved at the beginning and end of the transfer. This initial step is done with the help of DMA registers, which are setup by the CPU, and contain relevant information to the memory transfer. Once this has been setup, memory is copied from the I/O device into main memory without going through the processor.

## 2.2 b)

The CPU knows that the memory operations are complete through interrupts. Remember, operating systems are interrupt driven. For example: the DMA controller can modify the values in the DMA register to signal to the processor that the memory transfer is complete. And when the CPU periodically checks this value, it will know when the operation has completed. This is a programmed I/O interrupt. The DMA can directly send an interrupt to the CPU and signal that the transfer is complete.

## 2.2 c)

Normally, this process does not interfere with the execution of other user programs. However, the processor executes more slowly during a transfer when processor access to the bus is required. Another interference/problem can be caused if the CPU and DMA controller try to access the same bit of memory.

## 3.1)

The similarities between iOS and Android are summarized below:

Similarities:

=> Android and iOS have a similar stack

=> Both operating systems were built for mobile phones, and that's their main use

=> Android and iOS have an app store where developers can upload their programs for others to download. Plus, Apple and Google take a gargantuan 30% cut of your profits.

=> Both operating systems contain power management software (i.e. Deep sleep and low power mode)

=> Well crafted exploits for each operating system fetchs a high price in the exploit black market

=> Android and iOS have spin-offs to work on other devices (i.e. iPadOS for iPads, watchOS for watches, etc.)

=> The main functionalities and features across both operating systems is largely the same (i.e. Call, text, take pictures, listen to music, browse the web, download apps, play games, etc.)

Differences:

=> The base version of android is fully open source and other people can use it to create their own version of Android (i.e. LineageOS). On the other hand, iOS is a mix of open source components, proprietary software, and you cannot modify it to create your own version of iOS.

=> Android can be installed on just about any device (i.e. Phones, computers, tablets, thermostats, cars, fridges, watches, etc). Apple's iOS is strictly for their iPhones. This is why multiple companies (i.e. Samsung (shitsung), LG, Sony, etc.) ship their phones/devices with Android, and the only phones with iOS are iPhones.

=> iOS is a walled-garden and you cannot install apps without Apple's approval; side-loading is (kind of) not allowed, at least for the average user. Android allows you to install 3rd party apps, quite effortlessly. Plus, most vendors allow you to root your Android device and obtain root permissions. If you want to do the same on an iPhone, you need to jailbreak it (i.e. unc0ver).

=> Once you update your iPhone, you cannot go back to a previous version of iOS, unless it is still signed by Apple. With Android, you can easily downgrade your phone and goto any version you want.

=> iOS was developed and is maintained by Apple. On the other hand, Android was developed by the Open Handset Alliance, and is now maintained by Google

=> Android is based on the linux kernel. iOS is based on Darwin (BSD), and the kernel is a hybrid

=> Android apps are written in Java plus the Android API. Conversely, iOS apps are written in Objective-C or Swift.

=> iPhones (iOS) get better support and updates from Apple than Android devices get from their vendors (i.e. shitsung).

## 3.2 a)

System calls provide an interface to the services made available by an operating system. System calls are mostly accessed by programs via a high-level application programming interface (API). The purpose of system calls is so user programs can interface with the operating system. For example, a user program requests the OS/kernel for more RAM. The intricacies of memory management are handled by the kernel/OS and the user program gets the memory it needs without having to worry about the granular details or knowing how the system call is implemented.

## 3.2 b)

System programs are essential functions in every operating system. They sit ideally until invoked. These programs are used to carry out common tasks that the user and other processes need. For example, deleting files is very common and useful in an operating system. The system program ‘rm’ accomplishes this. Examples of other system programs include, but is not limited to: cd, ls, rm, format, etc.

## 3.2 c)

When you type words into a terminal, the command interpreter takes your input and converts it into commands/arguments before executing it or sending it off to be executed. The main function of the command interpreter is to get and execute the next user-specified command.

The command interpreter is usually separated from the kernel because the command interpreter only needs to be programmed once, and does not require subsequent updates. On the other hand, system programs and the kernel need to be constantly updated due to bugs, new features, etc. Furthermore, isolating the command interpreter from the kernel is good software engineering practice, because it incorporates the “design for change” and “separation of concerns” motto. New system programs can be added to the operating system without needing to change the command interpreter.

Note: The command interpreter may need to be updated if adding support for a new language.

## 4. a)

A = 0

B = 2603

C = 2603

D = 2600

## 4. b)

Deadlock; a major problem introduced by concurrent processing is deadlock. Deadlock occurs when tasks are waiting on each other. For instance, we have 3 tasks, A, B, and C. If A is waiting on B, and B is waiting on C, and C is waiting on A, then there is deadlock and nothing gets done. The program freezes/crashes; this is bad. Deadlock can also occur if a concurrent system is badly programmed (i.e. Not releasing the mutex lock).

Starvation; another problem introduced by concurrency is starvation. Starvation occurs when a process does not get the resources it needs to complete its job because other processes are hogging CPU time, or because its priority is too low.

Complex; the added complexity of concurrency, makes it much (much) harder to (properly) design, program, test, and debug a concurrent system; when compared to non-concurrent systems. It takes more time, and money, to create a successful concurrent system.

Security; concurrency introduces new security holes to a system that can be maliciously exploited by an attacker. If concurrency isn’t properly implemented it can lead to race condition bugs, memory corruption, buffer/stack overflow, etc.