OS Program Exercise I

* 上傳壓縮檔(以學號為檔案名)，壓縮檔內含有文件檔與程式檔。
* 文件內容:

1. 姓名、學號 2. 實作題目 3. 實作之程式語言與平台
4. 實作之程式**檔案** 5. 程式主流程 6. 重要的程式碼或副程式
7. 執行結果 (截圖) 8. 心得

** 以下任選一題實作

- 選 2.26 最高分 = 70
- 選 3.21 最高分 = 90

2.26. In Section2.3, we described a program that copies the contents of one file to a destination file. This program works by first prompting the user for the name of the source and destination files. **Write this program using either the Windows or POSIX API.** Be sure to **include** all necessary **error checking**, including ensuring that the source file exists. Once you have correctly designed and tested the program, if you used a system that supports it, **run the program using a utility that traces system calls.** Linux systems provide the **strace** utility, and Solaris and Mac OS X systems use the **dtrace** command. As Windows systems do not provide such features, you will have to trace through the Windows version of this program **using a debugger.**

3.21 The Collatz conjecture concerns what happens when we take any positive integer n and apply the following **algorithm: n= n/2, if n is even; n= 3×n+1, if n is odd.** The conjecture states that when this algorithm is **continually applied,** all positive integers will eventually reach 1. For example, if n = 35, the sequence is 35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1. Write a C program using the **fork()** system call that **generates this sequence in the child process.** The **starting number** will be provided from the **command line**. For example, if 8 is passed as a parameter on the command line, the child process will output 8,4,2,1. Because the parent and child processes have their own copies of the data, it will be necessary for the **child to output the sequence**. Have the **parent invoke the wait() call** to wait for the child process to complete before exiting the program. Perform necessary **error checking** to ensure that a **positive integer** is passed on the command line.

3.25 An **echo server** echoes back whatever it receives from a client. For example, if a client sends the server the string Hello there!, the server will respond with Hello there! **Write an echo server using the Java networking API** described in Section 3.6.1. This **server will wait for a client connection using the accept()** method. When

a client connection is received, the server will loop, performing the following steps:

- Read data from the socket into a buffer.
- Write the contents of the buffer back to the client. The server will break out of the loop only when it has determined that the client has closed the connection.

The date server shown in Figure 3.21 uses the **java.io.BufferedReader** class. **BufferedReader** extends the **java.io.Reader** class, which is used for reading character streams. However, the echo server **cannot guarantee** that it will read **characters** from clients; it may **receive binary** data as well. The class **java.io.InputStream** deals with data at the **byte level** rather than the character level. Thus, your echo server must use an **object that extends java.io.InputStream**. The **read()** method in the java.io.InputStream class **returns −1** when the **client** has **closed its end of the socket** connection.