COMP4621 Lab 6

Name: CHAN, Chun Hin

Student ID: 20853893

Email: chchanec@connect.ust.hk

1. *Can the UDP server in Tutorial2 serves multiple UDP clients concurrently? Why?*

No, the UDP server in Tutorial2 is not able to serves multiple UDP clients concurrently.

Since UDP is a connectionless protocol, this means the UDP server in Tutorial2 can only receive and send message to only one client at one time. It cannot establish continuous connection with multiple clients at the same time.

2. *What will happen if we do not close the server_socket/client_socket in client/server process?*

If we do not close the server_socker in client process, the child process will not be able to access the port that it would use to connect with the server_socker since the port would be occupied by the parent process.
In addition, the server_socket will consume computer resources if we do not close it. If we leave it idle, it may caused some computer resources not able to be released. This may eventually crash your machine at the end if more and more resources not being released.

3. What's the problem in non-blocking I/O?

The problem is that non-blocking I/O will receive whatever the client socket has sent to it. As it is non-blocking, this means it would receive all the messages based on the time it receives. This will likely to cause the problem that the received messages are placed in the incorrect order, because it does not guarantee to preserve the order of the original message sent by each clients.

4. *Find the problem of sigchld_handler function and give the right one.*

Recall the tutorial slide for this lab:

Regarding the return value of waitpid it said:
waitpid (): on success, returns the process ID of the child whose state has changed; if WNOHANG was specified and one or more child(ren) specified by pid exist, but have not yet changed state, then 0 is returned. On error, 1 is returned.

Hence, the problem is that the while loop continues to run when there is no terminated child process we need to handle. It is because when the child process is still running and none of the child is being terminated, PID is 0. However, we should jump out of the while loop when (i) no child process is terminated; or (ii) errors occurred.

We should change the while condition from "PID != -1" to "PID != 0 && PID != -1".

Therefore, below is the correct code:

```
staticvoid

sigchld_handler(intsigno){

   pid_t PID;

   int status;

   do{

      PID = waitpid(-1, &status, WNOHANG);

   } while(PID != 0 && PID != -1);

   /* Re-instate handler*/
```

```
    signal(SIGCHLD, sigchld_handler);
}
```