Lab05 – Concurrent Server Design

Sun, Wen-Lin

Since the client may send an indefinite amount of data to the server (and thus servicing the client may take an indefinite amount of time), a concurrent server design is appropriate, so that multiple clients can be simultaneously served. In this lab, we introduce two common designs of concurrent server: “Multi-tasking Server Design” and “Asynchronous Server Design”.

- ## Multi-tasking Server Design

In this design, the main task is responsible to listen to and accept socket connections. Whenever the main task gets a new connection, it will create a new task to interact with the connection and then keep listening to the binding port. This design concept can be implemented with both processes and threads.

Here we provide a segment of code to show the main progress we have introduced above. (Reference: *The Linux Programming Interface*, Chapter 60.3)

```
while(true) {
        /* Wait for connection */
        if(new_client_socket = accept(server_socket, NULL, NULL)) {
                syslog(LOG_ERR, "Failure in accept().");
                exit(EXIT_FAILURE);
        }

        /* Handle each client request in a new child process */

        switch(fork()) {
                case -1:        /* Fork Error */
                        syslog(LOG_ERR, "Can't create child process");
                        close(new_client_socket);
                        break;
                case 0:         /* Child Process */
                        close(server_socket);
                        handleRequest(new_client_socket);
                        exit(EXIT_SUCCESS);
                default:        /* Parent Process */
                        close(new_client_socket);
                        break;
        }
}
```

- **Asynchronous Server Design**

  In this design, we can design a single server process to handle multiple clients. To do this, the process simultaneously monitors multiple file descriptors for I/O events. It is easy to handle multiple file descriptors with the *select()* system call. You may reference to *The Linux Programming Interface, Ch63.2.1* for more detail about the usage of the *select()* system call.

There are still many different approaches of concurrent server. Here we only provide two simplest designs for you, and you can do more survey on it to find out an appropriate design for your homework.

Lab5 Task

Use multi-tasking server design to implement two different concurrent echo servers with processes and thread, respectively. Each client will send multiple commands. Both of the echo servers should receive and reply every commands from the client until the client close the connection. All the clients should be served simultaneously.

The server should print out some messages:

1. When it accept a client:
   Accept Client [no. of the client]
2. When it receives a message from a client:
   Received "[received message]" from Client [no. of the client]

Reference: Michael Kerrisk (2010). *The Linux Programming Interface*.