CMPUT 379 Assignment 2b, Winter 2015

Due Friday, Mar 21 (submit by 11:55 pm)

Worth 40 marks (however, 10% of total course weight)
(processes, IPC, sockets, synchronization)

Objective

In this assignment you will implement an unreliable simplified file server. This assignment will expose you to many issues related to writing daemons and network servers.

The file server is started with three command line arguments. The first argument is a UDP port on which it will listen to service requests made by clients. The second argument is a directory from where the server serves files to the clients. The last argument is a log file in which the server logs all transactions with all clients.

You will also implement the file client. The client is started with three command line arguments. The first argument is the IP address of the machine the server is running on, the second argument is the UDP port the server listens to client requests, and the third argument is the name of the file the client is trying to get from the server.

Server and client may run on different machines connected by a network.

The Server

You must implement **`file_server`** which is invoked in the following way:

```
file_server  9090   /some/where/documents   /some/where/logfile
```

This would invoke **`file_server`** listening on the local machine on port 9090, serving up files from the directory `/some/where/documents`, and logging transactions to `/some/where/logfile`.

- The server must properly *daemonize* on startup.
- The server place no limit on the size of files it can send to the clients.
- If the supplied `logfile` does not exist, the server should create it.
- **`file_server`** must be implemented using processes and `fork()`. That is, the server must fork a new child process to serve each new request it receives from a client.

The server accepts a request from a client that only sends the file name to be sent. The server splits the intended file into chunks of 1KB and sends each chunk as a separate message to the client. If the file has a size that is a multiple 1KB, the server should send an additional message containing only “$” after sending all of the chunks.

The server logs all transactions in the log file, one line for each request. Each line has the following format:

```
<client IP> <client port> <name of the file to be sent> <time when request received> <time when file transmission completed>

Or

<client IP> <client port> <name of the file to be sent> <time when request received> <file not found>

Or

<client IP> <client port> <name of the file to be sent> <time when request received> <transmission not completed>
```

The Client

You must implement **`file_client`** which is invoked in the following way:

```
file_client   192.168.10.1   9090    filename
```

This would invoke **`file_client`** sending a transfer request of `filename` to the server running on `192.168.10.1` and listening to UDP port `9090`.

The client accepts chunks of messages from the server. If a chunk contains less than 1KB of data or only the message “$”, the desired file transmission is completed. In the latter case, the client should ignore the message “$”. We assume that if a file has just one more byte beyond the multiple of 1KB, the last byte is not a ‘$’. For an active file transfer, if 5 seconds have been passed since the last chunk received, the client assumes that the transmission had been aborted, and the client prints the error message on the screen.

Marking

This assignment will be marked out of 40 marks.

- `file_server` implementation will be worth 30 marks
- `file_client` implementation will be worth 10 marks

Deliverables

You must deliver the code for server and client, as well as a `Makefile` to build them, where the command "`make all`" will build both server and client.

Mandatory Coding Style

Consistent and readable C coding style is *very* important to spotting bugs and having your code readable by others. You **MUST** comply with following coding style:

- Proper indentation should be included
- TAB should be used for indentation
- NO line may be longer than 80 chars
- All functions should be prototyped
- All variables are declared at the start of a block
- C style comments must be used (not C++ style)

Important: please always remember to remove from your machine the running server daemon(s) before logging off. Generally, make sure there are no stray processes left on your machine before leaving.