Programming Lab

Daytime *client-server* Exercises

Please note the following. All exercises to be completed on the Apollo Linux host. The IP address is "147.252.234.34". The exercises should also be completed from the command line. This will require the use of a command-line text editor such as ***nano*** and some extra commands to compile and run the program.

Exercise 1: To practice the commands use PuTTY to login into the *Apollo* server. This is remote access application that allows you login to a networked computer as if you were at the terminal. PuTTY is loaded onto the lab machines.

In the dialogue box enter the host IP address "147.252.234.34". Use SSH as your *connection type* (Port 22 is the default). Before pressing *open*, save this configuration as *apollo*. This will allow you to call it up in future sessions without having to re-enter the above configuration data.

Press *open*. You will be redirected to the *apollo* server and presented with a login screen. Your login will be your usual ***username*** and your *password*. This will give you access to your home directory (U:/>. If you are recently registered or not registered at all you will unlikely have an account.

Exercise 2: Socket Programming

Before embarking on the socket coding exercises you need to set up the environment.

Create a directory off your *apollo* home directory called ***csp*** (use the ***mkdir*** command)
Copy the following files (using the ***cp*** command) from the following location:

/Student_Distrib/DT228/DT228-3/CSP/SocketFiles

Files **Practical.h** and ***DieWithMessage.o*** go into the **csp** directory. ***DieWithMessage.c*** may not be required. Watch the case at all times.

Transcribe the daytime client program (see below) using *nano* (or another similar text editor) and save it as *daytimeclient.c* in your ***csp*** directory.

Execute the following commands from within the *csp* directory:

```
gcc -c -std=gnu99 daytimeclient.c
gcc -o daytimeclient daytimeclient.o DieWithMessage.o
```

These commands are case sensitive. Also all *o*'s (upper and lower case) are the letter *o* not zero.

The first command creates the object file from the source file. The second command links the precompiled library, **DieWithMessage.o** to your object file and creates an executable.

Run the client using the command: ***./daytimeclient 127.0.0.1 <port number>***

The second argument is the port on which the server is running (please ask during lab which port the server is running on). If you get the date and time back then your program works.

Notes:

1. The **./** in front of the above command.
2. Nano, pico and vi are all command-line Unix-specific text editors.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "Practical.h"

int main(int argc, char *argv[]) {
    char recvbuffer[BUFSIZE]; // I/O buffer
    int numBytes = 0;

    if (argc < 3) // Test for correct number of arguments
    DieWithUserMessage("Parameter(s)",
        "<Server Address> <Server Port>");

    char *servIP = argv[1];     // First arg: server IP address (dotted quad)

    in_port_t servPort = atoi(argv[2]);

    // Create a reliable, stream socket using TCP
    int sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sock < 0)
        DieWithSystemMessage("socket() failed");

    // Construct the server address structure
    struct sockaddr_in servAddr;            // Server address
    memset(&servAddr, 0, sizeof(servAddr)); // Zero out structure
    servAddr.sin_family = AF_INET;          // IPv4 address family
    // Convert address
    int rtnVal = inet_pton(AF_INET, servIP, &servAddr.sin_addr.s_addr);
```

```
34      if (rtnVal == 0)
35          DieWithUserMessage("inet_pton() failed", "invalid address string");
36      else if (rtnVal < 0)
37          DieWithSystemMessage("inet_pton() failed");
38      servAddr.sin_port = htons(servPort);    // Server port
39
40    // Establish the connection to the echo server
41      if (connect(sock, (struct sockaddr *) &servAddr, sizeof(servAddr)) < 0)
42          DieWithSystemMessage("connect() failed");
43
44      while ((numBytes = recv(sock, recvbuffer, BUFSIZE - 1, 0)) > 0) {
45          recvbuffer[numBytes] = '\0';    // Terminate the string!
46          fputs(recvbuffer, stdout);      // Print the echo buffer
47              /* Receive up to the buffer size (minus 1 to leave space for
48                  a null terminator) bytes from the sender */
49          }
50      if (numBytes < 0)
51        DieWithSystemMessage("recv() failed");
52          //    else if (numBytes == 0)
53          //    DieWithUserMessage("recv()", "connection closed prematurely");
54
55      fputc('\n', stdout); // Print a final linefeed
56
57      close(sock);
58      exit(0);
59  }
60
```