G51CSF SOCKETS COURSEWORK
ASSESSED

Steven R. Bagley

Introduction

For this coursework, you are to implement a 'Daytime Protocol' server and client. This server returns the current date and time to any client that connects to it. There is a standard Internet protocol for this which is defined in RFC[1] 867 (a copy of which is attached to the end of this coursework). The RFC defines both a TCP and UDP version of the protocol but we are only implementing the TCP version. RFC 867 states that the server should listen on port 13 but since this is a privileged port **your client and server should use port 1313**.

The coursework is split into two parts. In the first part, you will create a server and a client that implement the standard Daytime protocol defined in RFC 867 using C and the Berkeley sockets API. In the second part, you will implement a client and server for a modified version of the protocol which allows the user to fetch the date and time in more than one timezone during the session. More detailed descriptions of all four programs can be found below.

For both parts of the coursework, you can assume that only one client will be connected to the server at a time. An implementation of both servers is available that you can use to test your clients independently of your server, see the section on testing below for details…

Skeleton C files for all four programs are supplied on Moodle, please ensure you use these for your implementation. It is expected that you are developing and testing this coursework on the school's Linux systems.

Daytime Client

This program should connect to a daytime server on port 1313, read the date and time sent by the server, and print it out to the standard output. The address of the server should be specified as the first parameter on the command line, e.g. like this:

```
$ ./daytimeclient some.server.name
```

You can assume that the server will not send a date and time longer than 512 bytes. Once the date and time has been read, you should print it out and close the connection to the server.

Hint: Remember that you might have to read from the network connection multiple times to obtain the complete time.

Daytime Server

This program should implements a daytime server. It should listen on port 1313 for incoming connections from a client. When it accepts the connection it should then return the current date and time to the client. This should be presented on a single line, that is terminated with a carriage return (ASCII code 13) and line feed (ASCII code 10).

---

[1] Most Internet protocols are defined in documents called Requests for Comments (or RFC, for short)

You should ensure the entire string (including `CRLF`) is not longer than 512 bytes. Once the date and time have been sent the server should close its end of the connection and return to waiting for further connections. See the section below for details on how to easily generate the date and time string, but it should look like this:

```
Tue Feb 22 17:37:43 1982-GMT
```

Multi-timezone Daytime Protocol

The second part of the coursework revolves around a modified version of the Daytime protocol that allows the client to retrieve the time in multiple timezones from the server. Instead of running on port 1313, this **multi-timezone daytime protocol runs on port 1414**, so you will need to adjust your client and server accordingly.

The server should respond with the current time as usual (in GMT). However, instead of closing the connection it should wait for a command from the client. All communication between the client and server is sent as complete lines that are terminated by a `CRLF`.

The command can either be a line containing the word '`CLOSE`' (in capitals), in which case the server should send the response 'BYE' and then close the connection to the client. Alternatively, the client might respond with a line specifying a timezone (from the list below — in capitals) in which case the server should send the date and time in that timezone and await further commands from the client.

If the server receives any string from the client other than a valid timezone or '`CLOSE`', it should send the response '`ERROR`' and wait for a proper response from the client.

An example session between the server (**`S:`**) and client (**`C:`**) might look like this…

```
[C connects]
S: Tue Feb 22 17:37:43 1982-GMT
C: CET
S: Tue Feb 22 18:37:43 1982-CET
C: PEST
S: ERROR
C: PST
S: Tue Feb 22 09:37:43 1982-PST
C: CLOSE
S: BYE
[S closes connection]
[C closes connection]
```

Multi-timezone server

The multi-timezone server is very similar to the server you implemented before except that it should implement the revised protocol described above. The table below lists the offsets that should be applied to the current time for the timezones that are supported by your server. You should applied these to the time returned by `time()`.

Multi-timezone client

The multi-timezone client should take several parameters on the command line. The first parameter is the address of the server to connect to, as before. The remaining parameters is a list of timezones to request from the specified server, e.g. like this:

```
$ ./mtz-client some.server.name GMT PST CET EST
```

Your client should connect to the server and read the current time from the server, before printing it to the standard output, as before. It should then send the necessary commands to the server, to cause the server to send the time in the requested timezones, printing each of them to the standard output, separated by a single blank line.

Hint: Again, remember that you might have to read from the network connection multiple times to obtain a complete time.

Submission and Assessment

You should submit your four C programs to Moodle (just the `.c` source files) **by the 16th December**.

This coursework is worth 20% of the final mark, that 20% will be broken down as follows:

Daytime Client	2.5%
Daytime Server	2.5%
Multi-timezone Client	5%
Multi-timezone Server	10%

The marks will be awarded for each program based on whether it works, whether it implements the required features, and also the quality of that implementation.

Testing

To enable you to test your clients independently of your servers, there are test implementations of both the daytime server and the multi-timezone server running on the machine `arrow.cs.nott.ac.uk` on ports 1313 and 1414 respectively. You can point your client at this machine to test your implementation before you have implemented your server.

Also, you might want to try testing the 'interoperability' of your client and server with your friends' implementation, but remember your program should be your own work and so if you discover a bug it is up to you to work out what's wrong with your program alone. You can find the address of your Linux session by using the `hostname` command and then connect to this from another Linux session.

Time Zones

Your multi-timezone server should support the following timezones:

Timezone Command	Offset from GMT (in hours)
PST	-8
MST	-7
CST	-6
EST	-5
GMT	0
CET	+1
MSK	+3
JST	+9
AEST	+10

Obtaining Time

To obtain the current time, you can make use of the C library function, `time()`. This takes a pointer to a time_t variable (a synonym type for a large integer) that is filled in with the number of seconds since January 1, 1970. You can then pass this value into the library function `ctime()` to get an ASCII representation of the string (although you'll manually need to append the correct timezone string). For example, the following C program will print out the current time:

```
#include <stdio.h>
#include <time.h>

int main(int argc, char *argv[])
{
      time_t currentTime;
      char *timeString;

      time(&currentTime);
      timeString = ctime(&currentTime);

      printf("Time is %s", timeString);
}
```

Note, you'll almost certainly find it easier to do the timezone conversion when the time is in the integer `time_t` format, rather than trying to manipulate the string

Hint There are 3600 seconds in an hour.

Extracts from the UNIX **`man`** pages for **`time()`** and **`ctime()`**

```
time_t time(time_t *tloc);
```

The `time()` function returns the value of time in seconds since 0 hours, 0 minutes, 0 seconds, January 1, 1970, Coordinated Universal Time, without including leap seconds. If an error occurs, `time()` returns the value `(time_t)-1`.

The return value is also stored in `*tloc`, provided that `tloc` is non-null.

```
char *ctime(const time_t *clock);
```

The function `ctime()` takes as an argument a time value representing the time in seconds since the Epoch (00:00:00 UTC, January 1, 1970; see `time()`). When encountering an error, this function returns `NULL` and set `errno` to an appropriate value. The `ctime()` function returns a pointer to a 26-character string of the form:

```
Thu Nov 24 18:22:48 1986\n
```

All of the fields have constant width.

Hint You'll almost certainly need to remove that trailing newline (`\n`)…

Network Working Group
Request for Comments: 867

J. Postel
ISI

May 1983

Daytime Protocol

This RFC specifies a standard for the ARPA Internet community. Hosts on the ARPA Internet that choose to implement a Daytime Protocol are expected to adopt and implement this standard.

A useful debugging and measurement tool is a daytime service. A daytime service simply sends a the current date and time as a character string without regard to the input.

TCP Based Daytime Service

One daytime service is defined as a connection based application on TCP. A server listens for TCP connections on TCP port 13. Once a connection is established the current date and time is sent out the connection as a ascii character string (and any data received is thrown away). The service closes the connection after sending the quote.

UDP Based Daytime Service

Another daytime service service is defined as a datagram based application on UDP. A server listens for UDP datagrams on UDP port 13. When a datagram is received, an answering datagram is sent containing the current date and time as a ASCII character string (the data in the received datagram is ignored).

Daytime Syntax

There is no specific syntax for the daytime. It is recommended that it be limited to the ASCII printing characters, space, carriage return, and line feed. The daytime should be just one line.

One popular syntax is:

Weekday, Month Day, Year Time-Zone

Example:

Tuesday, February 22, 1982 17:37:43-PST

Another popular syntax is that used in SMTP:

dd mmm yy hh:mm:ss zzz

Example:

02 FEB 82 07:59:01 PST

NOTE: For machine useful time use the Time Protocol (RFC-868).