ProgC Programming Exercise 2021/22

Getting started

In this exercise you are required to understand, modify and write code in C and C++ and to link the two languages together. You will also gain a little exposure to low-level networking primitives, which will be a useful foundation for the networking course. To start this exercise, you will need your own personal *starter pack*, which will be emailed to you shortly, as explained at the top of `http://www.cl.cam.ac.uk/teaching/current/ProgC`

As well as this exercise sheet, the ZIP file should contain the following files:

```
server.c   client.c   rfc0791.txt   rfc0793.txt
message1   message2   message3      message4
```

Important: You should avoid the use of any compiler-specific features in the exercise. Code written for this exercise should conform to ISO9899:2011 ("C11") or ISO14882:2011 ("C++11") and use only standard libraries. The code should compile and run correctly using `gcc` or `g++` as available on MCS Linux. Your code should not generate any warnings, even with all warnings turned on; in other words

```
gcc -std=c11 -Wall --pedantic sourcefile.c
```
or
```
g++ -std=c++11 -Wall --pedantic sourcefile.cc
```

should produce no warnings for all source files you submit.

Exercise – Part 1

The exercise starter pack contains two computer programs written in C. The first program, `server.c` transmits data and the second program, `client.c` receives data. There are a few mistakes in these programs which prevent the code from compiling or functioning correctly. Do the following:

1. Describe in fewer than 100 words the intended functionality of the two programs `server.c` and `client.c`. Write this description into the file `answers.txt`. The format of this file should be plain ASCII text.

2. Find and list three programming errors in the code. (There could in fact be more than three errors, but you only need to document three.) Modify the code to correct the errors. Compile and execute the code with suitable test data. (Hint: Use a port number above 1024 to avoid running as root and the IP address `127.0.0.1` (called localhost) can be used as the IP address if you want to run both the client and server components on the same machine.) A list of the three errors should be appended to the file `answers.txt`. Your corrected code should be saved into `server1.c` and `client1.c`.

3. While `server1.c` is running on port 2080, type *localhost:2080/djg11* into the address bar of a browser running on the same computer. Add a final, short paragraph in `answers.txt` that explains how the request is being served and how the browser is interpreting the response.

Exercise – Part 2

A computer scientist writes a packet inspector to record every bit of data sent by (a corrected version of) `server.c` on machine A and received by (a corrected version of) `client.c` on machine B. The recorded data contains *all* the TCP/IP packets and includes the TCP and IP headers in *network byte order* together with the actual message payload.

The files `rfc0791.txt` and `rfc0793.txt` contain the specification of IP and TCP headers respectively. Your attention is drawn in particular to the layout, in bytes, of the contents of the header files as expressed in these documents. For convenience, the layouts are also shown in the Appendix.

The computer scientist invokes the server program on machine A and the client program on machine B once, and generates a log file of all the packets sent between the two machines. The log file contains the raw packet data in the order in which the packets were transmitted on machine A. Therefore the first packet is at the start of the file, and this is immediately followed by data from the second packet, and so on.

An example log file is `message1`. Your next task is to write a C or C++ program to read in *any* log file in this format and determine the following facts:

- The IP addresses of machines A and B (`source` and `destination`)
- The value of the first IP packet's header size field (`IHL` – IP header length)
- The length of the first IP packet in bytes (`total length`)
- The value of the first TCP packet's header size field (`data offset`)
- The total number of IP packets in the trace

Your program can initially be written as one C file. But before submission it should be split into two separately compiled files called `pcolparse.c` and `summary.c`. The former must be reused, in .o form, without recompilation, in Part 3 of this exercise. You must also write `pcolparse.h`, which contain declarations but no definitions, to be `#include`'d by both C files.

Your program should output the above details in the same order as shown above by printing to `stdout` on a single line, using spaces to separate the fields. For example, if the IP addresses of machines A and B are `192.168.1.1` and `192.168.1.2`, the header length field value is `6`, the length of the IP packet is `52` bytes, the value in the TCP header length is `5`, and the total number of packets in the trace is `20`, your program should output:

```
192.168.1.1 192.168.1.2 6 52 5 20
```

Exercise – Part 3

Your final task is to write a second program, this time in C++, that removes the IP and TCP headers from *any* log file conforming to the format described above and extracts the data transmitted from machine A to machine B. The source code for this program should be saved in `extract.cc`.

Your program must use your `pcolparse` as a library (i.e. `#include` your `pcolparse.h` and be linked with a `pcolparse.o` file resulting from C compilation of `pcolparse.c`). Note that `pcolparse.h` will need to be written carefully to be usable both by `summary.c` and `extract.cc` as the `extern "C"` form of C++ is not valid C.

Your `extract` program should take two arguments on the command line, and open and read log data from the filename passed as the first argument, and write data into a file whose name will be passed as the second argument. Your program should print an error message if a file with the name of the first argument does not exist. Your program should overwrite the data in any file with the same name as the second argument, or create a file of this name if it does not exist. For example, if an executable form of your program was called `extract` then the following execution

```
./extract message1 message1.txt
```

will extract the data held in `message1` and write it into the file `message1.txt`.

Use your program to do the following:

- Extract the data from `message1` and save this data into a new file called `message1.txt`. View this image Does the message make sense? (If not, you probably need to debug your code!)
- Extract the data from `message2` and save this data into a new file called `message2.jpg`. By using a web-browser or otherwise, view this image. What does the image show? (If you cannot view the image you probably need to debug your code!)
- *(Optional)* Decipher the contents of `message3` and `message4` to claim a prize (if there are any left or newly donated).

Submission

By the end of this tick you should have generated the following files: (No core files, temporary files, or Makefiles. All of these files and only these files to be submitted, please.)

```
answers.txt
client1.c AND server1.c
pcolparse.c AND pcolparse.h AND summary.c
extract.cc
message1.txt
message2.jpg
```

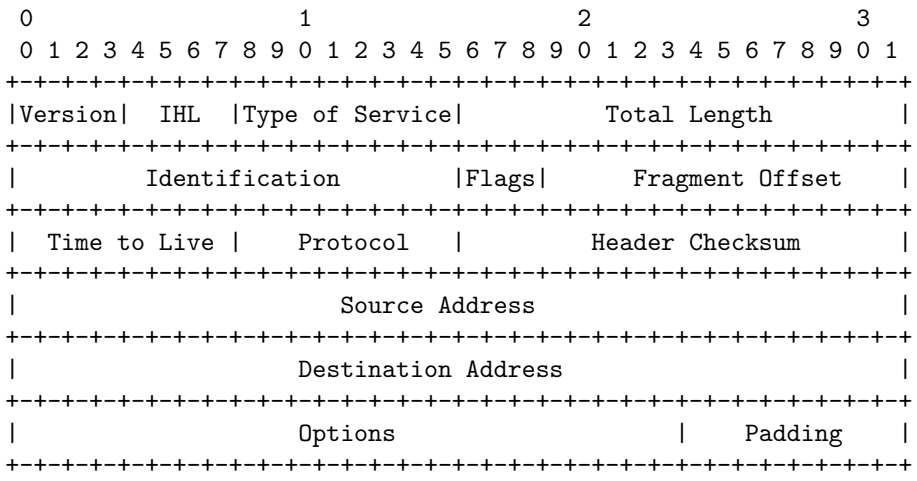
Once you have correctly completed the exercise you should make a new ZIP file of a folder called 'tick' that contains these files. This ZIP file must be named `CC++<year>-<crsid>.zip`, e.g. `CC++2022-djg11.zip`. To submit your tick, send email having 'subject' field identical to the ZIP file name, and containing the ZIP file as an attachment, to:

```
c-tick@cl.cam.ac.uk
```

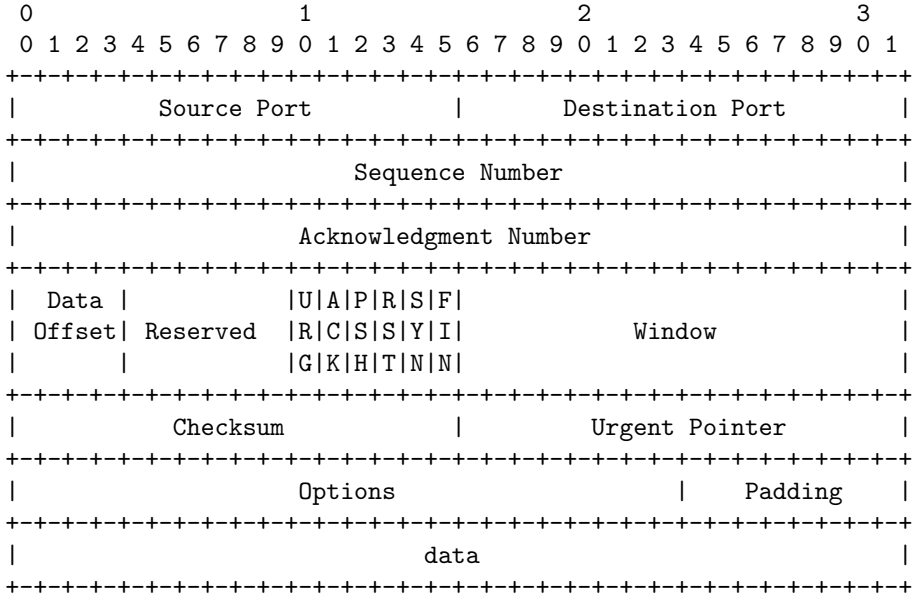
You must send this email before the tick deadline. A randomly selected group of candidates will be required to attend a Zoom viva voce examination of about 5 minutes duration. Do not copy files from someone else's starter pack since your files may be watermarked.

Appendix

RFC 791 (`http://www.ietf.org/rfc/rfc0791.txt`) specifies the layout of packets in the IP Protocol. You may need to read portions of this document in order to complete this exercise. Of particular interest is the position and size of the fields in the IP header:



RFC 793 (`http://www.ietf.org/rfc/rfc793.html`) specifies the layout of packets using the TCP protocol. You may need to read portions of this document in order to complete this exercise. Of particular interest is the position and size of the key fields in the TCP header:



Note: it is possible to model these structures in C either as a byte array or as a `struct`. Both representations have advantages and disadvantages.