Program 1
HTML Tag and Byte Counter

Prof. Kredo

Due: By 23:59 Sunday, September 22

Introduction

In this program you will accomplish several goals:

- Study the socket API client interface
- Develop a simple network client program
- Practice robust programming techniques

All programming assignments must be done in pairs.

Requirements

You will start this assignment during the "Introduction to the Socket API" lab using file provided to you. Even though you have permission to use the starter program, it is good practice to indicate you started with this code in your comments.

Your objective for this program is to write a program that requests a file from a web server and parses the data received to count the instances of the string '`<h1>`' and the total number of bytes you receive from the server. In HTML files, values within angled brackets are called tags, so `<h1>` is a `h1` (1st level header) tag. You should ignore any tag that is not the exact four character sequence '`<h1>`'. Your program must then display the total number of `h1` tags and total number of bytes to the user. Your program requests the file by sending the string `"GET /~kkredo/file.html HTTP/1.0\r\n\r\n"` to port 80 on the host `www.ecst.csuchico.edu`. The HTML file for this program contains only text characters.

While processing the file, you must do so in chunks whose size in Bytes is determined by the user through a command line argument. The chunk size will always be a positive integer larger than the tag and no greater than 1000.

The type of sockets you are using for this assignment, `SOCK_STREAM`, can arbitrarily break your data into pieces, so your program will need to account for this situation. For example, if you need 30 B of data and request that amount with `recv`, it is possible you only get 20 B. When this happens, you need to perform another `recv` for the remaining data (10 B in this example). This can happen an arbitrary number of times. For this assignment, `send` **and** `recv` **will not give you all the data you requested**, but they will always give you at least 1 B when there is data available. See 'Handling Partial send()s'[1] and other sections of Beej's Guide for details about partial data with `send`. A similar solution applies to `recv`.

Do not count `h1` tags that span chunks. If `<` is at the end of one chunk and `h1>` is at the start of the next chunk, then you do not count that tag. Thus, different chunk sizes will yield a different count of `h1` tags, as shown in Figure 1. To get full points you must use a library function or system call, *not a `for` loop that iterates through each array index* to compare or search for tags within a chunk.

Many students find it very tempting to reset the contents of a data array with a loop or a function similar to `memset`. `memset(array, 0, sizeof(array))` is a common example. This is not necessary and wastes resources. Your submission should passes all tests without using an operation equivalent to `memset` on more than one byte of an array. You can, and may need to, set one array location to a specific value.

[1] https://beej.us/guide/bgnet/html/#sendall

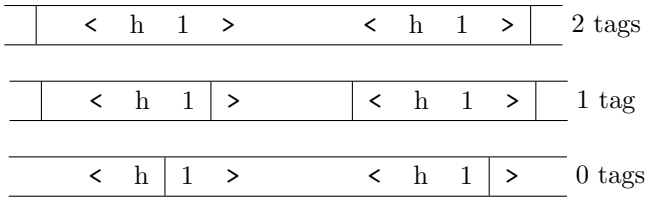

Figure 1: Example Tag Counts with Different Chunk Sizes

These are the general steps required for your program:

1. Read the chunk size from the command line arguments
2. Connect to `www.ecst.csuchico.edu` on port 80
3. Send the string `"GET /~kkredo/file.html HTTP/1.0\r\n\r\n"`
4. Receive and process the file
   (a) Receive a chunk of data
   (b) Count the `h1` tags in the chunk
   (c) Count the bytes received
   (d) Repeat until the file is done
5. Print the number of `h1` tags and total bytes

Additional requirements for this program include:

- Do not make assumptions about the total amount of data. Your program must work for data of arbitrary size.
- Ensure you use the `AF_UNSPEC` address family, not the `AF_INET` family. The starter program does this.
- You may not use any form of sleep function in this assignment and you may not use any socket flags (i.e., no `MSG_WAITALL`). You may not use `ioctl(2)` or equivalent functions.
- All submissions must pass compilation checks before they will be graded. Run the script `/user/home/kkredo/public_bin/program1_check` on `ecc-linux` to check your submission. Run the script without arguments or with the argument `help` for directions.
- Submit your program files through Canvas. Do not submit archive (zip/tar) files.
- You must include a Makefile with your submission, which will build the program by default and remove all generated files with the target `clean`. For Program 1, an example Makefile is provided on Canvas.
- The executable must be called `h1-counter` and accept one command line argument, the size in bytes, of the chunk used for counting tags.
- Your program must compile cleanly on the lab computers in OCNL 340 or `ecc-linux` and you must use the gcc/g++ argument `-Wall`. You may not use `-w` to disable warnings.
- Check all function calls for errors. Points will be deducted for poor code.
- Put both group member names, the class, and the semester in the comments of all files you submit.

Input/Output

Below is an input and output example (with incorrect values). Make your programs match the style and formatting of these examples, but you are free to write your own error messages.

```
$ ./h1-counter 10
Number of <h1> tags: 20
Number of bytes: 511
$ ./h1-counter 20
Number of <h1> tags: 22
Number of bytes: 511
```

Testing Chunk Size

Testing your program on `ecc-linux` without taking extra steps can lead to misleading results due to the very low latency experienced by two programs communicating on the same machine. To better simulate a real network, and better evaluate your program's ability to handle partial `send`s and `recv`s, you need to introduce some variability in how your program receives data. You can use a library written for this assignment to accomplish this task. When testing your program on `ecc-linux`, use the command below:

```
LD_PRELOAD=/user/home/kkredo/public_bin/libtesting_lib.so ./h1-counter
```

If you want to run your program with `gdb`, use the command below:

```
LD_PRELOAD=/user/home/kkredo/public_bin/libtesting_lib.so gdb ./h1-counter
```

Evaluation

Your program will be evaluated based on:

- 10 points: Command line argument used for chunk size
- 5 points: Connection made to server
- 20 points: Request sent correctly
- 30 points: Correct count of `<h1>` tags
- 10 points: Correct count of bytes received
- 10 points: Partial `recv` handled correctly
- 10 points: Partial `send` handled correctly
- 5 points: Tag count using library function or system call and no `memset` or equivalent operation used

Hints

- You can run your program with partial sends and receives by forcing the OS to use the testing library for Program 1. When running your program manually, use the command
  `LD_PRELOAD=/user/home/kkredo/public_bin/libtesting_lib.so ./h1-counter`
  To use the library with gdb, use the command
  `LD_PRELOAD=/user/home/kkredo/public_bin/libtesting_lib.so gdb ./h1-counter`
- In order to debug your program with partial `send`s and `recv`s, you will need to include the testing_lib file in your compilation and likely provide it to your debugger. The process to do this varies with each debugger, so get help in office hours if you are not able to find the necessary steps.
- Beej's Guide to Network Programming[2] is a valuable resource. **Especially section 7.4, which also applies to recv!**
- Test and debug in steps. Start with a subset of the program requirements, implement it, test it, and then add other requirements. Performing the testing and debugging in steps will ease your efforts. In general, the Grading requirements are ordered in a way to ease this implementation process.
- Your program should close after processing the entire file. The server closes the connection when the file is transferred, so `recv(2)` may be helpful. (HINT:"orderly shutdown" means to close a connection).
- You may have an optional command line argument that enables debugging output. The debug argument must be optional (meaning that your program must run when it is not specified) and the debug output in your code must be cleanly organized.
- For the declarations `char buff[256]; char *bp = buff;`, ensure you know the difference between `sizeof(buff)` and `sizeof(bp)`. **They are not equal!**
- An easy way to find an HTML tag in an array is to perform a string search. The data sent from the HTML server are all ASCII characters.

[2] https://beej.us/guide/bgnet/