ENCE360 Assignment 2021

Task

Your task is to write (some parts of) a multi-threaded HTTP downloader program. There are two parts:

- Part 1 – a minimal HTTP client
- Part 2 - a concurrent queue using semaphores

A web downloader has been provided that uses both of both these parts and you will do some analysis and write a short report. All the programs can be compiled using a Makefile provided, by going to the base directory where you unpacked the assignment and typing '**make**'.

Guidelines

- No more than 3 levels of nesting in any function and less than 40 lines of code
- Use minimum amount of code required
- Do not modify any code outside of **http.c** and **queue.c** (or the test programs which you may modify to improve – but they will not be marked)**,** otherwise your submission will fail to compile and you will receive zero marks
- All memory allocated during the programs execution should be free()d and all resources, files and sockets should be closed before the program finishes.
  Check your program with valgrind –leak-check=all to be sure!
- Comment code as necessary: excessive commenting is not required but anything not obvious should be documented
- Marks will be awarded for clean code and documentation

Submission

Submit files (zipped) in the same directory structure as provided, with an extra file called **report.pdf** or **report.doc{x}** in the base directory, and with the code files **queue.c** and **http.c** completed.

Do NOT MODIFY any header files (.h files): your code will be compiled using the original .h files and any modifications will be ignored. Your submission will be automatically marked, so any submission which does not fit the correct template *won't receive any marks*.

Provided files

src/

downloader.c: a multi-threaded downloader based on your solutions to Part 1 and Part 2.
http.c http.h: Http client implementation, Part 1.
queue.c queue.h: A skeleton for the concurrent queue, Part 2.

test/

http_test.c queue_test.c: These are provided to test your implementations for Part 1 and Part 2.

Part 1 – http client (30%)

Write an implementation of an http client in **http.c** which performs an HTTP 1.0 query to a website and returns the response string (including header and page content).

Make sure to test your implementation against a selection of binary files, and text files – small files and big. Be very careful using string manipulation functions, e.g. **strcpy** – they will not copy any binary data containing a '\0' – so you will want to use the binary counterparts such as **memcpy**.

For downloading bigger files you will need to think about how to allocate memory as you go – functions such as **realloc** allow you to dynamically resize a buffer without copying the contents each time.

Functions which you will need to implement this include: **memset, getaddrinfo, socket, connect, realloc, memcpy, perror and free.** Also, possibly useful are: **fdopen** and **dup.** Your lecture notes may be helpful.

HTTP GET request

To retrieve the file http://infolab.stanford.edu/~ullman/dragon/w06/lectures/gc.pdf your code needs to send a GET request in the following format:

"GET /page HTTP/1.0\r\n Host: host\r\n User-Agent: getter\r\n\r\n"
Where host = "infolab.stanford.edu " and page = "~ullman/dragon/w06/lectures/gc.pdf ".

Example output

Test programs **http_test**, and **http_download** are provided for you to test your code. An example output of the **http_test** is shown below. It is implemented in **test/http_test.c** and is built by the Makefile by default. The response consists of the HTTP header (lines up to "Content-Length") and the returned contents, in this case XML data.

```
./http_test www.thomas-bayer.com restnames/countries.groovy
Response:
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/xml;charset=ISO-8859-1
Date: Wed, 25 Aug 2021 01:22:21 GMT
Connection: close
Content-Length: 6273

<?xml version="1.0" ?><restnames>
  <countries>
    <country href="http://www.thomas-
bayer.com:80/restnames/namesincountry.groovy?country=Great+Britain">Great
Britain</country>
    <country href="http://www.thomas-
bayer.com:80/restnames/namesincountry.groovy?country=Ireland">Ireland</country>
    <country href="http://www.thomas-
bayer.com:80/restnames/namesincountry.groovy?country=U.S.A.">U.S.A.</country>
    <country href="http://www.thomas-
bayer.com:80/restnames/namesincountry.groovy?country=France">France</country>
        ...
```

`http_download` does a similar thing but instead writes the downloaded file to disk using a filename you give it. A script is provided to test your implementation against **wget:**

```
./test_download.sh infolab.stanford.edu/~ullman/dragon/w06/lectures/gc.pdf
downloaded 258064 bytes from
infolab.stanford.edu/~ullman/dragon/w06/lectures/gc.pdf
Files __template and __output are identical
```

Part 2 – concurrent queue (30%)

Write a classic implementation of a concurrent FIFO queue in **queue.c** which allows multiple producers and consumers to communicate in a thread-safe way. Before studying the requirements, it is advised to study the test program **queue_test** for an example of how such a queue is intended to be used.

Hints

Use semaphores (sem_init, sem_wait, sem_post etc.) and mutex(s) for thread synchronization. Use a minimum number of synchronization primitives while still maintaining correctness and maximum performance.

Testing

A test program **queue_test** is provided for you to test your code and illustrate how to use the concurrent queue. Note that it is not a completely comprehensive test – and the test used for marking will be much more stringent on correctness, it may be possible to run the queue_test program yet receive low marks. So you may wish to write your own tests and/or test with the provided **downloader** program.

```
./queue_test
total sum: 1783293664, expected sum: 1783293664
```

(This should complete within two seconds on the lab machines)

Part 3 – report (40%)

Algorithm analysis

Describe the algorithm found in **downloader.c** (the **main** routine, lines 175 onward) step by step. How is this similar to algorithms found in your notes, which is it most like – and are there any improvements which could be made?

Performance analysis

Provide a small analysis of performance of the downloader program showing how performance changes as the number of worker threads increases. What is the optimal number of worker threads? What are some of the issues when the number of threads is increased?
Run the downloader several times over different urls pointing to small and large files to download. The files **small.txt** and **large.txt** contain some sample test urls. For example, to download files in **small.txt** with 12 threads and save to directory '`download`'):

```
./downloader small.txt 12 download
```

How does the file size impact performance? Are there any parts of your http downloader which can be modified to download files faster?

A pre-compiled version of the downloader exists in '**bin/downloader**'. How does your implementation compare? If there's a large discrepancy when the number of threads increases, it's likely there's something wrong with your concurrent queue!

Submit this as report.pdf with your submission