CS360 – Operating Systems
Assignment 2 – Multi-threaded web server

Solve this project on your own relying on materials and examples demonstrated in class.

This project was borrowed from Prof. Remzi Arpaci-Dusseau Operating Systems class at the *University of Wisconsin – Madison*.

Operating Systems: Three Easy Pieces
Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau
Arpaci-Dusseau Books
August, 2018 (Version 1.00)

Instructions

- Go to this page. This is the project you will do: change an existing single-threaded webserver to make it multi-threaded.
- Read all the sections and familiarize yourself with how web-servers work in general.
- Go to your Linux machine, open a terminal window, and cd to some temporary folder.
- Clone Remzi's ostep-projects by executing the git clone command:

  ```
  git clone https://github.com/remzi-arpacidusseau/ostep-projects.git
  ```

  The above creates a folder named "`ostep-projects`" in the directory where you executed the clone command.

- Create a folder ("`assignment2`") somewhere on your Linux file system. Then cd to `assignment2`:

  ```
  cd <path_to_assignment2_folder>
  ```

- Run the command below (change the path in blue to where you placed `ostep-projects` on your Linux - in my case ostep-projects is under `~/cs360/temp`).

  ```
  cp -r ~/cs360/temp/ostep-projects/concurrency-webserver/src/ .
  ```

  You should now have folder `src` under `assignment2`:

```
assignment2
    src
```

- You can now delete the "ostep-projects" folder because you don't need all the other projects.

  ```
  rm -rf <path_to_ostep-projects_folder>
  ```

  Example: In my case `ostep-projects` in under `~/cs360/temp`, so I run this command:
  `rm -rf ~/cs360/temp/ostep-projects/`

- cd to `assignemnt2/src`.

- To build the webserver just run: `make`

  You will see many warnings (but no errors).
  This creates executable `wserver` (this is the web-server) and other tools like `wclient`.
  You can see `wserver` by executing command `ls`.

- In the README.md "Command-line Parameters" section are instructions on how to run the web server. Read the meaning of each command-line argument.

  ```
  prompt> ./wserver [-d basedir] [-p port] [-t threads] [-b buffers] [-s schedalg]
  ```

- The `basedir` is the root directory of where your html files are. For example, if you put a few html test files in a folder called `htmlfolder` in your home directory, then `basedir` should be: ~/htmlfolder

  The parameter p is the port number the server is listening on. If you don't specify a value for -p, the server will be listening on port 10000.

- Before writing any code, test how the server works. You can start it like this (assuming you put a few HTML test files in folder `htmlfolder` located in your home directory – **enclosed are 2 html test files for you to use**):

  prompt> `./wserver -d ~/htmlfolder`

```
anehme@A109-T012992M:~/cs360/assignment2/src$ ls
Makefile     io_helper.h  request.c  request.o  spin.cgi  wclient.c  wserver    wserver.o
io_helper.c  io_helper.o  request.h  spin.c     wclient   wclient.o  wserver.c
anehme@A109-T012992M:~/cs360/assignment2/src$ ./wserver -d ~/htmlfolder/
```

I placed the 2 html test files in a folder named htmlfolder under my home directory. So I set the **-d** option to this folder's path: **~/htmlfolder/**

Then you can test the webserver by opening a browser and typing the following address:

http://localhost:10000/index.html

The content of file `index.html` should load in your browser.
You can also try:

http://localhost:10000/products.html

The content of file `products.html` should load in your browser.

If you see the contents of these 2 files, then the basic functionality of the webserver is working: Your browser is requesting a page and the web server is delivering its content.

- All changes you need to make can go in

- When you implement multithreading, the number of threads you create is configurable and is passed to the server via the `-t` flag. Same for the size of the buffer which is passed via the `-b` flag. **In other words, you should not hardcode the number of threads to create and the size of the buffer. Instead, you create those based on the values passed to main from the command line.**

- Ignore the `schedalg` (-s) argument.

- At the beginning try to manually test your webserver (using a browser or tool `wclient`). Then use the automated tests provided to you in subfolder "HowToRunTests". Read file "MultiThreadedServerTest.pdf" to learn how to run the tests.

  Your grade will be based on how your implementation does against the automated tests.

- I should be able to build your code by simply running: `make`

  Note that if you add any new source code files, then you need to update `make` so that these files are included in the build. But you don't have to (your code can be in existing source code files).

How to Test Your Implementation

1. Copy the enclosed p4a folder to the `assignment2` folder on your Linux.

   ```
   assignment2
       src
       p4a
   ```

2. In folder p4a, there is a file named **runtests**. Open it in a text editor.

3. Change the value of the **base** variable (on line 2) to be the absolute path of folder p4a on your Linux. For example, in my case p4a is located at ~/cs360/assignment2/p4a. And the base will be set to that:

```
base=~/cs360/assignment2/p4a
```

4. The testing script gives an error if used with Python3. You need to use Python2. You probably have multiple versions of Python installed (look under: `ls /usr/bin | grep python`). In my case I have version 2.7 so I am using it as shown below:

```
#!/bin/bash
base=~/cs360/assignment2/p4a
python2.7 $base/test_p4a.py --test-path $base $@ | tee -i runtests.log
exit $?
~
```

5. Save the changes you made to **runtests** and close it.
6. Make runtests executable by running this command:

```
chmod +x runtests
```

7. Now you can run the tests like this (assume you are in the **src** folder):

```
make clean                          // Removes executables
make                                // Build (verify no errors)
../p4a/runtests -c                  // Run the tests
```

If I run the tests without making any change to the code, I get this when the tests are done:

```
Passed 4 of 21 tests.
Overall 4 of 21
Points 6 of 100
```

If I run the tests by creating the requested number of threads (but without doing anything in the threads), I get this (doesn't seem to be correct – the tests aren't doing a good job testing. What I will do is check your code if you are doing something in the thread function).

```
Passed 19 of 21 tests.
Overall 19 of 21
Points 90 of 100
```

You should aim to get 21 out of 21.

Grading

- Your implementation will be graded using automated tests (p4a/runtests).

- There is a 20% penalty for an incorrect build. That is, you submit your assignment but when I run make, the make generates errors (and I have to ask you to submit again).

What to Submit

- Create folder **john_smith_hw2** (replace john smith with your name).
- Go to the src folder and run `make clean`.
  (This removes files like wserver, wclient, etc. from your src folder). Don't submit these files because I will generate them by running make on your code.

- Put folder **src** in **john_smith_hw2**.

  john_smith_hw2
  src
  wserver.c
  request.c
  …
  …
  make
  …
  …

  (Remember to first run `make clean` in src to remove all executables from src).

- Zip john_smith_hw2 to generate **john_smith_hw2.zip**.
- Upload **john_smith_hw2.zip** to Canvas before the due date.

Remember to test **make** with your added code. In other words, I will build your code by simply running **make**. And it should build without errors, and generate wserver, wclient, etc.

Do not submit your assignment if it has build errors (that is, if running **make** generates errors – warnings are OK, but not errors). It is a waste of my time and your time and adds unnecessary administrative overhead and delays my ability to grade in a reasonable time. You will have to resubmit again, send me the code by email, and then I have to find your latest code in a place other than Canvas. This is a waste of time that delays grading for other students.

Remember: If make is not working, I cannot grade the assignment.

Therefore, before you submit:

1. Run `make`
2. Verify that there are no errors (warnings OK).
    a. If you find errors, fix and check again by doing step 1.
3. Verify that wserver is created (by running `ls` and checking that the file is there).