CIT 595 Spring 2020 - Final Project

Introduction

Your final project in this course seeks to combine the various technologies and concepts that you have learned this semester, including threads, socket programming, HTTP, and JavaScript.

The goal of this project is to develop a Remote CPU Monitor which keeps track of how much time the CPUs on a host computer are spending executing processes. This would allow a system administrator to get information about how busy a computer's CPUs are and to decide, for instance, if some processes need to be terminated or if additional machines/CPUs are needed in order to complete some task.

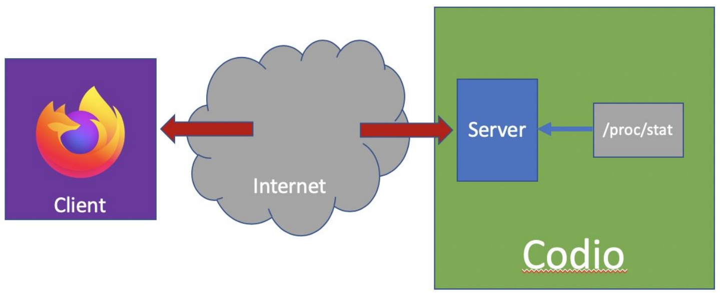
You may complete this project alone or with one other student; groups of three or more will not be permitted. The instructions below detail the additional work that a group of two would be expected to complete.

Overview

Certain implementations of the UNIX operating system make information about the underlying hardware and the operating system itself available via files in the "/proc" directory. These are not actual files on the disk, but are rather like "device files" in the "/dev" directory in that they are a way of providing information using standard file I/O mechanisms.

One such file is "/proc/stat", which provides statistics about the amount of time the CPUs on the host machine spend doing various things like executing user processes, handling interaction with external devices, or doing nothing, which we call being "idle".

In your final project for this course, you will implement a C program that reads from the "/proc/stat" file, computes some basic statistics, and makes that data available to a user who is accessing your program via a web browser.



Getting Started

As with the other C programming assignments in this class, we will use Codio as the development environment and host machine for the project.

A brief video describing how to get started with this project is available in Canvas in Files -> Project -> GettingStarted.mp4; text instructions are below:

Before starting on the project, be sure you can compile and run the starter code we have provided, which implements a very simple HTTP server, and that you can access the HTTP server running on Codio through your web browser.

Log into Codio and open the project named “CIT 595 S20 Project”.

Open a Terminal and compile the file named server.c and name the output “server” like this:

```
cc -o server server.c
```

This program configures a socket to listen for incoming connections on a specified port, then upon receiving a connection, prints the contents (which we assume are from an HTTP Request), and then sends back an HTTP Response reading “It works!” along with the number of requests handled since the program began.

Run the program; the starter code has hard-coded 3000 as the port number on which to listen:

```
./server
```

It should print out “Server configured to listen on port 3000” and it is now awaiting incoming connections.

Now you will use your web browser to access the HTTP server that is running on Codio. In Codio, click the button that reads “Open browser (port 3000)” in the top menu bar.

This should open a new tab (you may need to configure your browser to allow a webpage to open new tabs) and you should see “It works!” and “count=1”.

Then, in that same browser tab, click refresh or reload to make a new HTTP Request, and it should now say “It works!” and “count=2” or “count=3”.

If it still says “count=1”, or if you don’t see anything at all, try using a different web browser to access your Codio project. We have run into problems with this using Google Chrome on Mac, but Firefox on Mac seems to work okay.

If you try different browsers and none seem to work, please notify the Instructor, otherwise it will be very hard to proceed with your project!

This starter code runs in an infinite loop to await incoming connections; you can use Ctrl-C in Codio to stop this program.

Part 1. Calculating CPU Usage

The first step in developing your solution is to write a program that calculates and displays the average usage of the CPUs on the host computer.

In the same Codio project, create a new C file (you can call it whatever you'd like) that does the following:

First, open and read the file named "/proc/stat", which contains statistics for all the CPUs on the host machine and looks something like this:

```
cpu  4182512 24070992 10977154 202353854 9059845 0 274918 11399 0 0
cpu0 1041758 6069007 2745614 50592318 2189755 0 82347 10643 0 0
cpu1 1047584 6249123 2729366 50449288 2177479 0 79310 214 0 0
cpu2 1046249 5953982 2739203 50606454 2329140 0 56504 312 0 0
cpu3 1046921 5798880 2762971 50705794 2363471 0 56757 230 0 0
```

followed by lots of other information.

There is much more detail about this file online at websites like this one, but for our purposes we only care about the first line, which provides aggregate information for all CPUs on the host machine, and specifically the fourth number (which is "`202353854`" in the example above) which represents the total amount of idle time, measured in hundredths of seconds, for all CPUs since the computer started.

So in this case, "`202353854`" means that since the computer started, the combined amount of time for which all the CPUs were idle (not executing any processes) was 2023538.54 seconds.

Next, your program should sleep for one second (review your Homework #2 solution if you don't recall how to do this) and then read that same value from "/proc/stat" again.

Now subtract the previous/initial reading from the current/most recent reading, which gives the (approximate) aggregate idle time for the CPUs in the past second. Then divide it by four to get the average, since the Codio host has 4 CPUs. This number gives the average amount of idle time for all the CPUs, and also the average percent of idle time, as it should be between 0 and 100 since we waited one second and the value is in hundredths of seconds. Last, subtract that number from 100 to get the average usage time, i.e. the time the CPUs are not idle.

For instance, if on the second reading, your program read “`202354176`”, then we would do the following:

1. Subtract the previous reading from the current: `202354176 - 202353854 = 322`
2. Divide the number by four: `322 / 4 = 80.5`
3. Subtract it from 100: `100 - 80.5 = 19.5`

This means that, in the past second, the CPUs had an average usage of 19.5%.

Store this value in a global variable (you’ll see why later!) as a double, and then print it to the console.

If the number is less than 0, print 0; if it is greater than 100, print 100. These may happen because of inaccuracies in the implementation of the “sleep” function or in measuring CPU usage, but should not happen frequently.

Now, repeat the previous step so that, once a second, the program prints the average CPU usage percentage for the past second, using the previous reading and the most recent reading.

Your program should keep looping and printing the average value; you can use Ctrl-C in Codio to stop the program.

Be sure this program works correctly before moving on to the next part of the project!

Part 2. Remote Access via HTTP

In this next part of the project, you will make it possible for a user to access the latest CPU usage percentage, as well as some basic statistics about CPU usage, via a web browser.

Modify the code in server.c so that, whenever it receives an HTTP request, the contents of the HTTP response are dynamically generated and include:

- The latest CPU usage percentage as determined by your solution to Part 1
- The maximum CPU usage percentage value that has been observed since the program started
- The average of the CPU usage percentage values over the past hour, or since the program started if it has been running for less than an hour

To do this, you will have to run your solution from Part 1 in one thread, and then have the start_server function from server.c running in another, since the program needs to simultaneously wait for incoming network connections but also continuously read the /proc/stat file.

As hinted at above, you should use global variables so that the code from Part 1 can write the data to them and the start_server function can read them, just like you did in Homework #2.

Part 3. Stopping the Server

Modify your server program so that there is an additional thread that waits for the user of the program (in the Codio terminal, *not* via the web browser) to enter the letter ‘q’ and hit Enter/Return. Once the user does that, the server should print the message “Shutting down” and terminate.

Part 4. Additional Features

In addition to allowing the user to see the current, maximum, and average CPU usage percentages using a web browser, your system must also implement some subset of the features described below, which are separated into Category 1 and Category 2.

If you are working alone, then you must implement **one** of the additional features; the feature can come from either category. Category 1 features are probably less time consuming, but you may want to try a Category 2 feature if they seem more interesting or you want to give yourself a challenge.

If you are working in a pair, then your team must implement **both** Category 1 features and **one** Category 2 feature.

Although you are encouraged to implement as many of these features as you like, you will not receive extra credit for doing more than what is specified here.

Category 1

- **Auto-Update:** Instead of requiring the user to refresh the browser in order to get updated readings, it should be possible for the user to click a button in the HTML page that puts it into “auto-update mode” such that, once a second, it automatically makes an HTTP request to the server to get the latest data. However, this should not require reloading the entire HTML page, but rather only retrieving the data from the server and using JavaScript to update the HTML content. Information about the JavaScript you need to use to implement this feature will be made available in Piazza.
- **Threshold Alert:** The user should be able to specify a threshold by entering a number into an HTML input field and clicking a button, such that if the next CPU usage percentage that is displayed in the browser exceeds that threshold, the value is displayed in red and a JavaScript alert is shown informing the user that the value has exceeded the threshold; if the value does not exceed the threshold, then no alert should be shown.

Note that a team of two is expected to implement **both** of these features and cannot use an extra Category 2 feature as a replacement for either of these.

Category 2

- **Display Usage Data in a Chart:** Use a JavaScript library such as D3.js to produce a chart showing a graphical representation of the CPU usage data. The x-axis should represent time and the y-axis should show the usage percentage at that time. You may combine this with the "Auto-Update" feature if you would like to have the chart automatically update itself, or you can display only the data that was retrieved when the browser made an HTTP request.
- **Changing Frequency:** By default, the server reads /proc/stat once a second. It should be possible for the user to change the frequency with which the server reads /proc/stat by typing a number into an HTML field and clicking a button so that the number is sent to the server and is used to specify the amount of time that the server "sleeps" before re-reading /proc/stat. This number should be a positive integer; you can handle errors however you like. Keep in mind that, in addition to changing the amount of time for which the server sleeps, you'll also need to use this number for performing the calculation of the current, max, and average usage percentages.
- **Multi-threaded HTTP Server:** The HTTP server that we provided runs in a single thread, so that if the server is handling an HTTP request and another comes in, the second has to wait until the first one finishes before it can be handled. Modify the "start_server" function so that, after accepting an incoming connection, it creates a new thread to handle the request and send back the response, so that the first thread can go back to waiting for incoming connections.

Additional resources for completing these features are provided in Canvas in Files -> Project, including videos on HTML forms (which you will need for both Category 1 features) and D3.js, which you can use for the first Category 2 feature.

Schedule

You will need to do a remote demo of your system (e.g. via Zoom) to a member of the Instruction Staff no later than **11:59pm EDT on Friday, May 1**.

Information about signing up for a demo will be posted in Piazza as we get closer to the end of the semester. Late demos may be permitted with prior permission from the Instructor.

To ensure that you are staying on track with the project and not waiting too long to get started, it is **strongly** recommended that you try to accomplish the following by Friday, April 17, which is the approximate midway point of the project:

- A student working alone should have finished Part 1 and at least be able to see the latest CPU usage percentage via the web browser in Part 2
- A team of two should have finished Parts 1, 2 and 3, and one of the Category 1 features

Although no formal demo is required by April 17, members of the Instruction Staff would be happy to set up a brief meeting with you around that time if you think it would help to have a goal to work toward.

Additionally, if by April 17 you are not nearly completed with the tasks suggested above, it is highly recommended that you reach out to the Instructor to figure out a plan for getting you back on track.

Grading

Information about grading will be released shortly.

Keep in mind that a student working alone is expected to complete **one** additional feature (from either category), and a team of two is expected to complete **both** Category 1 features **and** one Category 2 feature.