CSCI3120
Assignment 2[*]

Instructor: Alex Brodsky

Due: 12:00pm (noon), Monday, June 13, 2016

The purpose of this assignment is to get you to implement some of the scheduling algorithms and to manage client connections in manner very similar to how operating systems manage processes. In this assignment you build upon Assignment 1, to create a Scheduling Web Server. You will:

1. Implement three schedulers for your web server that schedule the service of the requests.
2. Create a test suite for the scheduling web server, and
3. Answer a couple questions.

You have a choice. You can either use your code from Assignment 1 as the starting point, or you can use the provided solution to Assignment 1. Regardless, you need to ensure that you have `assn2` checked out in your SVN repository. This directory will contain the solution to Assignment 1. Please follow the guidelines in Section 6.1.

1 Background

Our goal is to make our web server as responsive as possible. I.e, we want to minimize (on average) the amount of time a client waits for their request to complete, which depends on the web server. The amount of time that the web server takes to service a request (in our case), strictly depends on the size of the file requested: The bigger the file, the longer it will take the server to send it. By scheduling the responses to requests based on the file size, we can create a more responsive server. Not surprisingly, this problem is very similar to the problem of scheduling processes.

Recall that a web server can receive requests from multiple clients concurrently. That is, our server can get connections to all clients currently waiting by repeated calls to `network_open()`, which returns the file descriptor for a connection, or -1 if no more clients are currently waiting. For each client, the web server can also determine how long the request will take to complete: The bigger the requested file, the longer the request will take.

Since the server will not service each request in the order that they arrive, the server will need to store state for each request. Thus, it is recommended, that the server use a Request Table (similar to a Process Table), which is an array of Request Control Blocks (RCB). Each RCB, akin to a Process Control Block) will need to store several pieces of information:

- The sequence number of the request (starting at 1)
- The file descriptor of the client (returned by `network_wait()`)
- The FILE * handle (or the file descriptor) of the file being requested
- The number of bytes of the file that remain to be sent
- The quantum, which is the maximum number of bytes to be sent when the request is serviced
- Any other information that your scheduler may need

[*]Revised: May 30, 2016

2 Task 1: The Scheduling Web Server (`sws`)

Your task is to implement the scheduling web server (sws). The source file `sws.c` should contain the `main()` function. The scheduling web server should do the following:

1. Take two (2) arguments on the command line:
   (a) The first argument is a nonnegative integer denoting the port number to which the web server should bind.
   (b) The second argument is the scheduler that the server should use. Your server should implement (at minimum) the following three schedulers:
   SJF : Shortest Job First
   where Job is the size of the requested file. Once the file is being sent back, it will be sent in its entirety.
   RR : Round Robin
   The server limits the amount of data to be sent back to 8KB (8192 bytes). I.e., if the file is bigger than 8KB, the request will take multiple passes through the scheduler's queue to complete.
   MLFB : Multilevel queues with Feedback
   The server should use three queues:
   8KB : First 8KB of the response (highest priority)
   64KB : Next 64KB of the response (medium priority)
   RR : Remainder of the response (low priority)
   All requests initially end up in the 8KB queue. If the response is greater than 8KB, after the first 8KB is sent, the request is demoted to the 64KB queue. Requests in the 64KB get processed only if the 8KB queue is empty. A request from the 64KB queue, can send up to 64KB of the response. If the response is not complete, the request is demoted to the RR queue. Requests in FCFS will be completed only if the other two queues are empty. The responses from the RR queue can send 64KB of their response and then must be placed back in the RR queue until their turn comes up again.

   For example, this invocation binds to port 12345 on start up, and uses the SJF scheduler.
   ```
   ./sws 12345 SJF
   ```
2. Initialize the network module and initialize a global sequence counter to 1.
3. Enter an infinite main loop that
   (a) Check if there are clients waiting to connect or if there are requests waiting to be serviced.
   (b) If no clients are waiting and no requests are outstanding, wait for clients to connect.
   (c) Connect to each client and admit their request to the scheduler. (See Step 4.)
   (d) Get the next request to process from the scheduler
   (e) Processes the request (See Step 5.)
4. Each client's request is admitted by following steps:
   (a) Receive the client's request message.
   (b) Parse the client's request message and extracting the requested file path.
   (c) If the request is not well formed or the file is not available, send back an error response and close the connection.

   (d) Otherwise,
      i. Determine size of file.
      ii. Allocate and initialize a Request Control Block (RCB). Be sure to set the sequence number and increment the global sequence counter.
      iii. Pass it to the scheduler to add to its queue.
      iv. Send back the response status, i.e., `HTTP/1.1 200 OK\n\n`

5. A request returned from the scheduler is to be serviced in the following way:
   (a) Send to the client the next part of the response that is the minimum of *remaining number of bytes* and *the quantum*.
   (b) Update the number of bytes left to send.
   (c) Notify the scheduler. The scheduler does the following:
      i. If the number of bytes left is 0 (after sending), print out (to `stdout` the line:

         `Request` *seq* `completed.`

         where *seq* is the sequence number of the request. Lastly, close the connection, close the file, and free the RCB.
      ii. Otherwise, insert the request back into the queue.

You can use the provided `makefile` to build your web server simply by running `make`. You can use your `sws.c` from Assignment 1, or the solution to Assignment 1 as a starting point. Please see Section 3 on how you can test your implementation.

3 Task 2: Test Your Code

You will need to test your scheduling web server implementation. A tool `hydra.py` is provided to help you test your code. This program, written in Python, acts as a multiheaded client that can send concurrent requests to your server. You can schedule when the requests are sent, and time how long the requests take, as well as the response time for each request. You can also compute the average response time for all requests. Please read `hydra.py` to see how to use it.

To test your implementation, you first create script files for `hydra.py` and the set of files to be requested. Second, you run your web server either in the background or in another terminal window. Suppose your test script was contained in the file `test1.in`. To view the performance of your server, use the command line:

```
./hydra.py < test1.in
```

and to save the response from `hydra.py` in a file called `test1.out` you, you would use the command:

```
./hydra.py < test1.in  > test1.out
```

You can then examine the output to see if it is as it should be.

Develop 15 test cases to test your scheduling web server, 5 per scheduler. Create three (3) files for each test:

`test.##.in` is the script file for `hydra.py`
`test.##.txt` is a brief description of the test and the expected result
`test.##.out` is the expected output of the test

where ## represents a number 00 . . . 14. The `.out` file can be created from the `.in` file. E.g.,

```
./hydra.py < test.01.in  > test.01.out
```

The tests should test all the functionality of the scheduling web server. The test description files **must** contain: a title, author (your name) purpose of the test, how the test works, and a description of the expected result.

4 Task 3: Written Work

Answer the following questions. Your answers should provide sufficient detail and justification but need not be more than a paragraph long for each part.

1. Which of the three schedulers should perform best? Why? (Be sure to explicitly state your assumptions about the workload and why the chosen scheduler is best suited for it.)
2. How does your server perform? Note: You should develop a simple performance suite of tests (using `hydra.py`), run your server on performance suite (multiple times), and do some data gathering and analysis to describe the performance of your server. A table of your performance results would be helpful. You will also likely want to run the same performance tests on each of your three schedulers.

5 Hints and Suggestions

1. You will need to use the `stat()` or `fstat()` functions to get the size of a file.
2. You can allocate a static request table with 64 RCBs.
3. Start early, the testing and questions will also take some time.
4. Even if you do not finish the code, submit the test files, they are worth many marks.

6 What to Hand In

You must submit an electronic copy of your assignment. Submissions must be done by 12 noon of the due date. The submission should be done using SVN. See the course web-page (Assignments or Resources) on how to submit using SVN.

6.1 Programming Guidelines

1. Use the file and function names specified in the assignment.
2. Your programs will be compiled and tested on the `bluenose.cs.dal.ca` Unix server. In order to receive credit, your programs must compile and run on this server.
3. All submitted programs must compile. **Programs that do not compile will receive a 0.**
4. All submitted programs must be built with the provided makefile. **Programs that do not build with the makefile will receive a 0.**
5. The code must be well commented, properly indented, and clearly written. Please see the style guidelines under Assignments or Resources on the course website.

6.2 Required Files

Use SVN to submit the following files.

1. The source code files that implement the scheduling web server.
2. The makefile used to build your web server
3. The test files specified in Section 3
4. The file `questions.pdf` or `questions.txt` containing the answers to the questions in Section 4

7 Grading

	Mark	
Simple Web Server		/50
Structure and Design of Web Server	/10	
Shortest Job First Scheduler	/5	
Round Robin Scheduler	/5	
Multilevel Queues w/ Feedback	/5	
Functionality (automated tests)	/25	
Test Suite		/25
Quality (effectiveness of the tests)	/10	
Number of tests	/15	
Written Work		/15
Question 1	/5	
Question 2	/10	
Code Clarity		/10
Total		**/100**