CS 361: Systems Programming **Lab Session 9 Worksheet**
Fall 2019

Threading

1. In this lab, we will go over what you will need to know about multi-threading for homework 5. We will also be going over multithreading a great deal in class, but the basics of getting it to work for homework 5 are much simpler. Additionally, it may be helpful for you to read Section 12.3 of the textbook.

2. Open the file thread_example.c (in the code you checked out for homework 5) and read through it. This file demonstrates creating and using posix threads. Compile and run this file, and see what it does.

3. This file uses two posix threads functions, pthread_create and pthread_join.

4. Read the man pages for these functions with "man pthread_create" and "man pthread_join."

5. Below is the definition for pthread_create

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                           void *(*start_routine) (void *), void *arg);
```

6. pthread_create takes a pointer to an (empty) thread struct, a pointer to a thread attribute structure, a pointer to a function, and a pointer to an argument to be passed to the function. When it is run, it will create the new thread using the attributes, populate the thread struct with information about the thread, and start the thread running the function on the passed in argument.

7. In our program, we create our threads like so:

```
int retval = pthread_create(&threads[i], NULL,
                                      thread_function, (void *) &arguments[i]);
```

8. Answer the questions about thread_example.c on gradescope.

9. Notice we also call pthread_join on all our threads. Just like calling wait on a process, calling join on a thread pauses the calling thread until the thread it called join on has exited. Here, we use it to make sure the main process does not exit before our created threads.

10. Try commenting out the code that calls thread_join. What happens?

Multi-threaded Webservers

1. A common web server structure uses a different thread to take care of each of its client connections. That is what we will be doing here.

2. Below is the code from the webserver's main method (I have removed some comments and error handling for space.)

```
int main(int argc, char** argv) {
    int retval;

    int port = atoi(argv[1]);

   int server_sock = socket(AF_INET6, SOCK_STREAM, 0);

    int reuse_true = 1;
    retval = setsockopt(server_sock, SOL_SOCKET, SO_REUSEADDR, &reuse_true,
                        sizeof(reuse_true));

    struct sockaddr_in6 addr;   // internet socket address data structure
    addr.sin6_family = AF_INET6;
    addr.sin6_port = htons(port); // byte order is significant
    addr.sin6_addr = in6addr_any; // listen to all interfaces

    retval = bind(server_sock, (struct sockaddr*)&addr, sizeof(addr));

    retval = listen(server_sock, BACKLOG);

    while(1) {
        int sock;
        char buffer[256];
        struct sockaddr_in remote_addr;
        unsigned int socklen = sizeof(remote_addr);

        sock = accept(server_sock, (struct sockaddr*) &remote_addr, &socklen);

        serve_request(sock);

        close(sock);
 }

    close(server_sock);
}
```

3. Answer the questions on gradescope about the webserver.

Testing

1. The easiest way to test whether your web server is basically working or not is to simply point a web browser at it, as described in the previous lab. However, web browsers are built to be extremely forgiving - if you send a web browser anything that seems vaguely like a webpage, it will do its best to display it, even if your server is sending an invalid response. We here in CS 361 are much, much less forgiving than web browsers.

2. You will need to test both your response string, and the files you send, to make sure both are perfect.

3. To test your files, use wget to download the file. (For example, to download mono2.jpg from your webserver, try `wget http://systems1.cs.uic.edu:PORTNO/WWW/monorail.jpg`.)

4. First, try opening the file and make sure it opens properly in an image display program

5. Next, make sure it is bit-for-bit identical to the original file. To test this, you can run the `diff` program on both files to make sure they're the same. For instance, after downloading monorail.jpg, you can test your downloaded version against the version in the WWW subdirectory by running: `diff monorail.jpg WWW/monorail.jpg`
   Try this on two images that should be identical, and two files that should not be. Answer the question on gradescope.

6. To print out the response string your program is sending, try using curl. Typing `curl -v URL > /dev/null` will download the object at URL, redirect standard out (the file contents) to /dev/null (which discards the bytes), and print out the request and response bodies.

7. Answer the questions about testing on gradescope.