Project CSE325 Spring 2018(Part-1)

Multithreaded Web Server

Due date: Feb 26-27, Corresponding Lab Class

Objectives: Understand and expose the basic activities of a simple web server, and learn how to create and synchronize multiple threads in it.

Problem Description:

In this project, you are going to develop a multithreaded real, working web server. To simplify your project work, the code of a simple web server is going to provide you. The simple web server operates with only a single thread environment. Your job is to understand the code and working strategies of the simple web server and make the web server multi-threaded so that it is capable to take multi-client request and work more efficiently.

Background:

The project works are formulated with the hypothesis that the students enrolled in CSE325 do not have pre-knowledge on network activities as well as internet protocols. A socket programming activities will be explained in your lab works and a client-server code of an echo server will be provided. You will need to learn socket (), listen (), accept (), read (), and write () system call C APIs. Find more details about socket programming in https://www.tutorialspoint.com/unix_sockets/socket_server_example.htm

The code for the web server is available in your course Google drive directory. It is a simple web server and limited to HTTP Get request only. Copy all of the files from there into your own working directory. Compile the files by simply typing "make" command. Compile and run this basic web server before making any changes. You can use "make clean" to remove all object files (.o) to do a clean build. Learn more details about the web server from WIKI link: https://en.wikipedia.org/wiki/Web_server

Multithreaded Web Server Implementation:

The given single-threaded web server suffers from a fundamental performance problem, serves HTTP request synchronously, and performs multi-tasking approach to serve client requests. Handling request of one process creates control problem for the kernel and increases context switch time, waiting time for other clients to access this web server. Thus, cooperating threads approach rather than multi-tasking is required, where scheduling will take place in user space, and thus your task is to implement multi-threaded by extending the simple web server to multi-threaded web server.

The simplest approach to building a multi-threaded server is to spawn a new thread for every new http request. The OS will then schedule these threads according to its own thread handle policies. You will learn the thread handling with POSIX C thread libraries in your thread laboratory works (lab4).

In your implementation, you must have a master thread that is responsible to create worker thread(s). Master thread accepts new http requests over the network and placing the socket descriptor for new thread connection. You should investigate the POSIX thread APIs (https://computing.llnl.gov/tutorials/pthreads/) related to how to create and manage threads with pthread_create, pthread_detach and thread_join.

Each worker thread is able to handle request. A worker thread is created when an http request arrives. When there are multiple http requests, which request is handled depends upon the scheduling policy. It will be implemented in your next assignment. Once the worker thread is created, it performs the read on the socket descriptor, obtains the specified request and then returns the content to the client by writing to the descriptor. The worker thread then waits for another http request.

Note that the master thread and the worker threads are in a producer-consumer relationship and they require share accesses to the synchronized shared buffer. Specifically, the master thread must be blocked and wait if the buffer (thread pool) is empty; a worker thread must wait if the buffer is full. Such locking mechanism will be implemented in your second assignment.

You need to understand the following code snippet from the simple web server:

```
for (hit=1;  ;hit++)
{
        length = sizeof(cli_addr);
        if((socketfd = accept(listenfd, (struct sockaddr *)&cli_addr, &length)) < 0)
                log(ERROR,"system call","accept",0);

        if((pid = fork()) < 0) {
                log(ERROR,"system call","fork",0);
        }
        else {
                if(pid == 0) {
                        (void)close(listenfd);
                        web(socketfd,hit);
                }
               else {
                        (void)close(socketfd);
                }
        }
}
```

Try to find answer for the following question:

1. How does fork() systemcall work and what is pid==0 represent?
2. What are parent process tasks and what are child process tasks?
3. How does the socketfd work in web function and what does the hit represent?

Now replace this code snippet from multi-process (parent, childs) to use multi-threaded (main thread, child threads).

Mark Distribution

The project mark (10) will be measured in **psychomotor domain** including physical movement, coordination, and use of the motor-skill areas.

Level	Category	Meaning	Marks
P2	Manipulation	Reproduce activity from instruction or memory	3
P3	Precision	Execute skills reliably; independent of help.	3
P4	Articulation	Adapt and integrate expertise to satisfy a non-standard objective.	2
P5	Naturalization	Automated, unconscious mastery of activity and related skills at strategic level.	2