CS 5523: Operating Systems
Fall 2014

Project 1: Threaded Server and Performance Measurement

(Due: **Nov. 4, 2014**, Tuesday before class)

- **Objectives**
  - Practice network and socket programming
  - Practice system calls for using threads and multi-thread programming
  - Practice the usage of synchronization mechanisms
  - Learn and practice the usage of system timers;
  - Learn and practice performance measurement with system timers
  - Practice writing technical project report;

- **Project Description and Requirements** (this is an **INDIVIDUAL** project; you may **discuss** with your classmates on the usage of system calls, but should write your own codes)

You will implement a client-server program using socket programming. There is only one server but **multiple** clients. The clients and server may run on different physical machines.

Serve side: The server provides the three magic arithmetic services and one magic sorting service, which are defined as follows:

- ***magicAdd, magicSub,*** and ***magicMul***: here, each operation takes **two floating numbers (dataOne and dataTwo), and one integer (k);** it will accumulate the results of k iterations of the specified operation. For example, for magicAdd, the i^{th} iteration should be result[i]=(dataOne/i + dataTwo/i), (i = 1, …, k). Then, the accumulated **returnValue = sum (result[i])** will be returned as a float number;
- ***magicSort***: this operating takes **one integer (k)**, and a **sequence of k integers** as input. It will sort the integers with increasing order, and return the sorted sequence of integers.

The server accepts and processes service requests from different clients, and should have **one new thread to serve each request**.

The server has a counter for each service, which records the number of requests performed for each service since the server start (that is, the server will have **4 counters**, and their initial values are 0). In addition, the server has **4 timers** to track the accumulated processing time for each kind of service. That is, the server needs to measure how much time it takes to processing a request (including message exchange and actual data operations). See below for more information about system timers. [**Note that, the counters and timers are shared global variables in the server, and the service threads need to have proper synchronization mechanism to access/modify them.**]

Client side: For **each client**, it supposes to have **5 threads** to concurrently send out service requests, and the **total number of requests** send out by **a client should be 1000**. Here, each request is randomly selected from the **4 services** provided by the server, as well as the

required data parameters. For the 3 magic math operations, the floating data should be a random value between the maximum and minimum floating values, and k should be in the range of [500, 1000]. For the magicSort, k should also be in the range of [100, 1000], the integers in the sequence should be in the range of [1, 1000000].

Note that, when a client needs to send a sequence of values to the server, it may not be able to send all of them in one round. In that, it can send it one at a time, or multiple values a time (as the agreement between clients and the server).

Other requirements and notes:

Server output: After processing each 1000 total service requests, the server should output (print out and report) the current values of counters and timers, as well as the average processing time for each kind of service requests.
System timers: you may use *gettimeofday*( ) in C/C++ or *currentTimeMillis*() of System class in Java, to measure the processing time of each service request. Basically, you need to read the timer to get the *beginTime* at the beginning of processing the request, and then read the timer again to get the *endTime* after sending back the results. The processing time will be (endTime – beginTime) with the time unit of the functions used.

Program Languages: you can use either C/C++ or Java for this project.

Extra credit [20 points]: if you implement thread-pool instead of using a new thread to process each request.

- **Project report:** Write a technical report for this project. You should include at least the following materials and discussions.
    - The status of your project (completed, partially working, or not working etc.); if completed, instructions to compile/run your program;
    - The protocols used to exchange a large sequence of values;
    - The synchronization mechanism used among the service threads;
    - The system timer used and the machine information (such as OS and CPU etc);
    - Discuss what you find out about the system timer;
    - Discuss the challenges and difficulties that you encountered for this project and how do you overcome them;
    - Comments and suggestions for this project: which part you like or dislike; what's is your opinion to make it easier/interesting/useful etc;
    - List the references (such as books and links) that help you finish this project;

- **Project submission (where and how):**

    - Have your program codes and report in a zip file with name as XXX-proj-1.zip (for example, Dakai-proj-1.zip) and **submit/upload to Blackboard**. No hardcopy is needed for the program codes.
    - Print a **hardcopy** of your report and hand in before the class **Tuesday Nov. 4, 2014.**