Project 2 of CS5523

Mybooks.com: An Online Book Store

Objectives:
- Familiarize you with sockets/RPCs/RMIs, processes, threads;
- Practice the usage of synchronizations
- Learn the design and internals of a distributed application
- Practice how to measure the performance of a software system

Project Requirements:
This is an INDIVIDUAL project. You may discuss with your classmates, but you have to write your own code and copying the code directly will cause you to fail this course. YOU CANNOT COPY THE CODE FROM PREVIOUS STUDENTS.

Project Descriptions:
You are tasked to design Mybooks.com - the World’s smallest book store. Mybooks.com only has the following four books for sale:
- How to be good at CS5523.
- RPCs and RMI in distributed systems.
- Why go to the graduate school.
- How to survive the graduate school.

You will have to implement a client-server program using sockets/RPCs/RMIs. There is only one server and multiple clients. The clients and servers may run on different machines. We strongly suggest that you should use multiple machines to test your program in the end.

The server is going to maintain the information about these four books. For each book, it maintains the number of available items in stock, the cost and topic of each book. Currently all books belong to one of two topics: distributed systems (first two books) and graduate school (the last two books). The server also maintains a list of all orders received for the books, thus it can calculate how many requests that it have received, and how many items are still available. More information can be seen in the following:

The server supports the following services:
- search(topic) : allows the user to specify a topic and returns all books belonging to this topic. For each book, it will show different book’s item_number (e.g. an internal id).
• \textit{lookup(item\_number)}: allows an item\_number to be specified and returns details of all items, such as number of available items in stock, cost and topic
• \textit{order(item\_number)}: allows the user to buy a book with the specified item\_number. If there are some available items on a book, it will return some successful information. Otherwise, it will return the fail information.

The server also supports some report functionalities so that we can use it to improve our service or performance in the future. For example, if we have too many requests on a book, we can increase the number of available items for the next year. Also, it can report the performance to serve each request, thus this can provide some information of how to improve the performance in the future.
  • \textit{reportRequestsNumber (service)}: allows the user to query that how many requests on each service have been received.
  • \textit{reportGoodOrders}(): tell users how many books have been sold successfully starting from the beginning.
  • \textit{reportFailedOrders}(): report how many orders are failed in total.
  • \textit{reportServicePerformance(service)}: report the average performance for serving a request for the specific service.

The server has a counter for each service -- search, lookup and buy (total \textbf{3 counters}), which records the number of requests for each service. The server also includes \textbf{3 timers} to track the accumulated processing time for each service. That is, the server needs to measure the time that it spends to serve each type of request. These counters and timers are shared across among all threads and thus needs proper synchronizations.

Each client may perform search, lookup or buy operations. The server should be able to accept multiple requests at the same time. This could be easily done using threads. Be aware of the thread synchronizing issues to avoid inconsistency or deadlock in your system. For instance, the server should be able to process multiple concurrent buy requests and decrementing the number of items in stock should be done using synchronization.

\textbf{Note: No GUIs are required. Simple command line interfaces are fine.}

\textbf{Other Notes:}
• Initial input: available items for each book can be randomly.
• System timers: you can use gettimeofday() in C/C++ or currentTimeMillies() of System class in Java to measure the processing time of serving each request. Basically, you need to read the timer before you are serving a request (beginTimer) and read the timer after sending back the reply (endTimer). You can use the value of (endTimer – beginTimer) as the processing time for a request.
• You can use either C/C++ or Java for this project.
**Project Report:**
Writing a technical report to describe this project. You should at least include the following contents in your report.

- The status of your project (completed, or partially working). If completed, you should include a complete instruction to run your program. You must provide some files to help build your programs, such as Makefile.
- How to maintain these books information in server? Whether it is scalable or not?
- How to handle a request from the client? If you are using thread-pool, how to do that?
- How to use synchronization in server?
- How to measure the performance of serving a request?
- Discuss the challenges and difficulties that you have encountered when you design this project and how you overcome them.
- How you designed some test cases to verify whether your program works correctly or not?

**Project Submission:**

- IMPORTANT: Source code should be packed in a zip file with the name NAME-ABCid-Proj1.zip, such as Tongping-OPC001-Proj1.zip. Inside this zip, you should include three directories, SERVER, CLIENT and TEST, and a README. All programs belonging to server should be put inside SERVER, and all programs belonging to client should be put inside CLIENT. TEST is used to put some test files that you have designed. You should provide Makefile for these directories. Inside each directory, you should have an additional run.sh, which includes some possible commands that we can use to verify your program.
- The technical report is also putted into the root directory of your submission, named as report.pdf or report.txt. No paper submission is accepted.

**Grading Policy:**

**Code:**
- Works correctly: 60%
- In-line documentation and scalable design: 10%
- Testing: 10%

**Technical Report:** 20%