Project #3: Concurrent Stock Server

Today: May 15, 2025
Submission Due: June 2, 2025
(Late submission: June 5, 2025, -10% per day)

1 Introduction

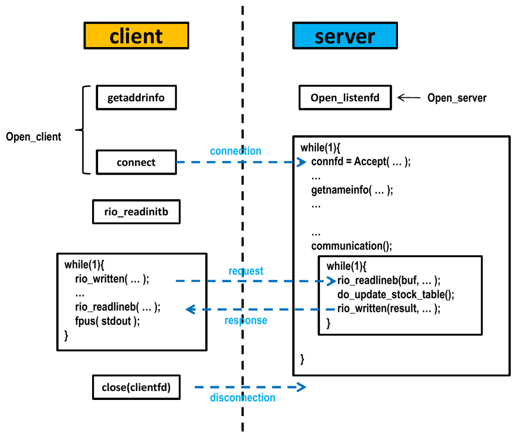
In this project, the students will apply the ideas of concurrency covered in the class. Students will learn this by programming a simple network server and host that simulates a mock stock exchange system, which is capable of serving multiple users. To represent these users, students will also program a client interface, which allows for interaction with the stock server.

Project Prerequisites. Familiarity with C/C++ programming and basic network programming concepts.

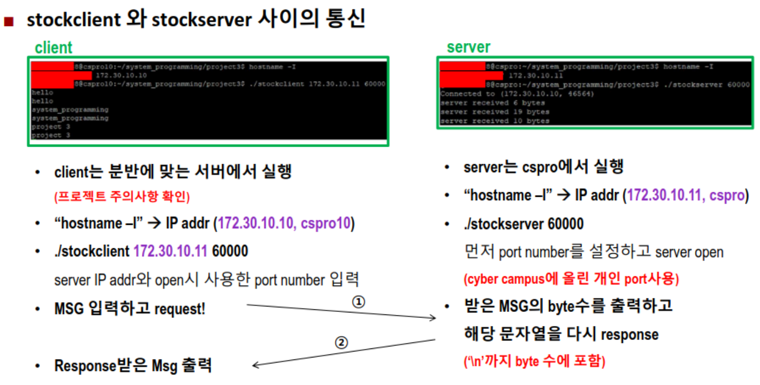
This project consists of three different tasks, with two tasks involving different approaches to the same problem, and the final task a performance analysis of both approaches.

2 Background

To provide background information about the inner workings of a server-client system, the code of a sample echoserver, along with single-client and multi-client is given. In the project files, once you compile the code with `make`, three binary executable will be created: `stockserver`, `stockclient`, and `multiclient`. `stockserver` and `stockclient` are the same as `echoserveri` and `echoclient` as seen in class. It is recommended to try running `stockserver` and `stockclient` first, before running `stockserver` and `multiclient`.



(a) Diagram of interaction between the client and server.



```
[redacted]8@cspro10:~/system_programming/project3$ hostname -I
172.30.10.10
[redacted]8@cspro10:~/system_programming/project3$ ./stockclient 172.30.10.11 60000
hello
hello
system_programming
system_programming
project 3
project 3
```

```
[redacted]8@cspro:~/system_programming/project3$ hostname -I
172.30.10.11
[redacted]8@cspro:~/system_programming/project3$ ./stockserver 60000
Connected to (172.30.10.10, 46564)
server received 6 bytes
server received 19 bytes
server received 10 bytes
```

- client는 분반에 맞는 서버에서 실행
  (프로젝트 주의사항 확인)
- "hostname –I" → IP addr (172.30.10.10, cspro10)
- ./stockclient 172.30.10.11 60000
  server IP addr와 open시 사용한 port number 입력
- MSG 입력하고 request! ①
- Response받은 Msg 출력 ②

- server는 cspro에서 실행
- "hostname –I" → IP addr (172.30.10.11, cspro)
- ./stockserver 60000
  먼저 port number를 설정하고 server open
  (cyber campus에 올린 개인 port사용)
- 받은 MSG의 byte수를 출력하고
  해당 문자열을 다시 response
  ('\n'까지 byte 수에 포함)

(b) Specfic instructions on executing `stockserver` and `stockclient`.

Figure 1: Overview of the interaction between the client and server.

3 The Task: Concurrent Stock Server.

A single-process/thread server cannot serve multiple client connection and their requests at the same time, as all requests from each connected clients will be serialized and processed one at a time in such a scenario. To serve multiple clients, the server needs to run make use of concurrent programming based on the lessons taught in class. To enable said concurrency, students will implement a concurrent stock server with the following two approaches in separate tasks.

1. An event-based approach using `select()`
2. A thread-based approach using the `pthread` library.

3.1 Explanation of Client

The client has four functions: `show`, `buy`, `sell`, and `exit`.

- `show`: Show the current available stocks in the market.
- `buy [stock ID] [amount of stock]`
- `sell [stock ID] [amount of stock]`
- `exit`: Leave the stock market server.

For this assignment, **only the stock server needs to be implemented**. The grading will be performed using the provided `multiclient.c` code. Also, note that for evaluation of your program, **no exception handling for unexpected inputs will be considered.** This includes typos and incompatible inputs. Information about what stocks and amount of stocks that a client has is also not considered. Thus, the `sell` command will never fail.

3.2 Implementation of Server

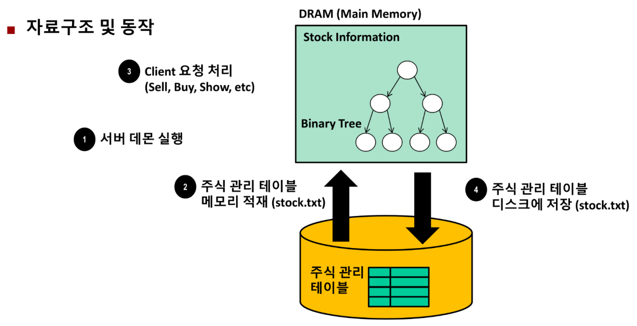
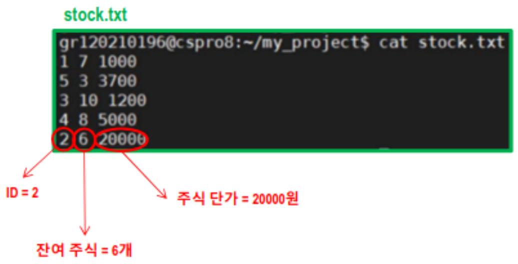

■ 자료구조 및 동작
DRAM (Main Memory)
Stock Information
Binary Tree
3 Client 요청 처리 (Sell, Buy, Show, etc)
1 서버 데몬 실행
2 주식 관리 테이블 메모리 적재 (stock.txt)
4 주식 관리 테이블 디스크에 저장 (stock.txt)
주식 관리 테이블

Figure 2: Overview of the inner workings of the stock server.

The server maintains the stocks in a text file named `stocks.txt` in a table format. Each row represents a stock entry, with three attributes:

- The ID of the stock. Each stock is given a random unique ID between 1 and `INT_MAX`.
- Remaining amount of stock.
- The price of the stock.



```
gr120210196@cspro8:~/my_project$ cat stock.txt
1 7 1000
5 3 3700
3 10 1200
4 8 5000
2 6 20000
```

Figure 3: Example of a `stock.txt` file.

The following assumptions are made involving the status of the stocks.

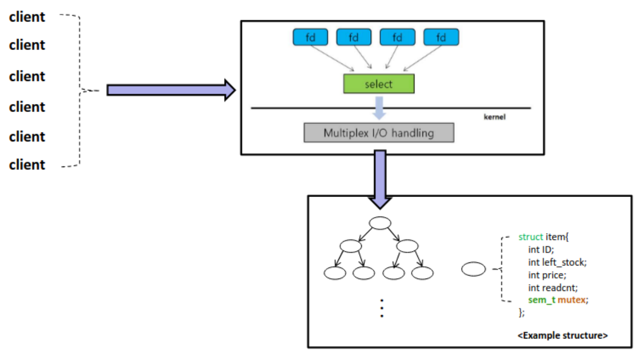
- The price of each stock stays the same. Therefore, the only change is in the amount of stock that is left.
- If a request from a client requests to buy more stock than there is currently, then only the message `Not enough left stocks` is returned, and the request is not fulfilled.

To efficiently manage stock data, a binary tree can be used to hold the information of each stock. Each node will contain the previously mentioned three attributes (Stock ID, remaining amount, stock price), as well as a mutex to ensure a client exclusive access to said stock. When the server starts, the server should read the contents of `stock.txt`, store it in the binary tree, and then start accepting requests from clients. When the server shuts down, it should update `stock.txt` with the new values for each stock.

In regards to synchronization and resource contention, while one client is reading a stock node i, another client cannot be allowed to write data in said stock node i. Therefore, students need to consider the readers-writers problem solution covered in class to achieve fine-grained locking with the mutex.

4 Task I: Event-driven approach (Points: 30)

In this approach, each client triggers a file descriptor `fd`, and the server monitors the file descriptors with `select`. By monitoring multiple file descriptors in this way, synchronous I/O multiplexing and servicing of clients is achieved.



```
struct item{
    int ID;
    int left_stock;
    int price;
    int readcnt;
    sem_t mutex;
};
```

Figure 4: Overview of event-based approach of concurrent stock server.

5 Task II: Thread-based Approach (Points: 30)

In this approach, a pool of worker threads is maintained to service incoming client requests. A master thread is kept running to accept initial connection requests from clients, and to insert descriptors to a intermediary buffer. Each worker thread in the pool then takes a descriptor and provides service to the client until the client exits the server.

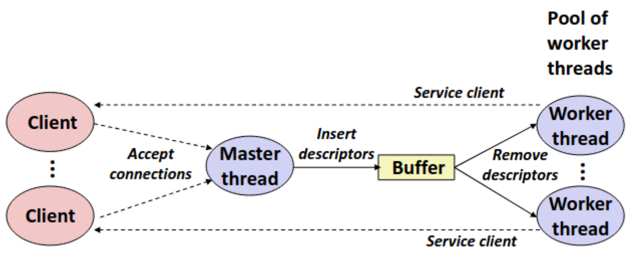


Figure 5: Overview of thread-based approach of concurrent stock server.

6 Task III: Performance evaluation and analysis (Points: 30)

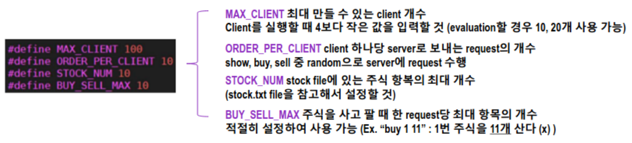
The final task of the project is to evaluate the performance of both approaches. The analysis should be done by changing the configuration of the client count in `multiclient.c`, and compare the elapsed time between the two approaches based on change in client count. For this task, you can use more clients than in testing for previous tasks, such as 10 20 clients. (Task I and II are restricted to a maximum of 4 clients.) You can use the `gettimeofday` function to make time measurements for elapsed time.
Some criteria/ideas to perform analysis on are as follows:

- Expandability: Analyze the effect of increasing client count has for each approach.
- Workload type: Analyze how the ratio of client requests types (buy, show, sell) affects performance.
- An analysis of whether or not your implementation of the two approaches match what you've learned in class. If the performance results don't match, why?

7 Appendix

7.1 More information on `multiclient`

`multiclient.c` allows for the evaluation of your concurrent stock server by creating a specified amount of client processes, and issue a specified amount of show, buy, and sell commands for each client.



```
#define MAX_CLIENT 100
#define ORDER_PER_CLIENT 10
#define STOCK_NUM 10
#define BUY_SELL_MAX 10
```

MAX_CLIENT 최대 만들 수 있는 client 개수
Client를 실행할 때 4보다 작은 값을 입력할 것 (evaluation할 경우 10, 20개 사용 가능)

ORDER_PER_CLIENT client 하나당 server로 보내는 request의 개수
show, buy, sell 중 random으로 server에 request 수행

STOCK_NUM stock file에 있는 주식 항목의 최대 개수
(stock.txt file을 참고해서 설정할 것)

BUY_SELL_MAX 주식을 사고 팔 때 한 request당 최대 항목의 개수
적절히 설정하여 사용 가능 (Ex. "buy 1 11" : 1번 주식을 11개 산다 (x))

Figure 6: `#define` settings for `multiclient.c`.

To evaluate your server, do the following:

1. Set the `#define` values.
2. Start your stock server (`./stockserver 1119`)
3. Run `multiclient`. (`./multiclient [host server IP_addr] [port #] [client_num]`) During normal testing, remember to keep the `client_num` value to 4 or less.

7.2 Server output format

- `[buy] success`: When a buy request from a client is successful.
- `[sell] success`: When a sell request from a client is successful.
- `Not enough left stocks`: When there is not enough stock to fulfill a buy request.

8 Submission Guideline

Process Management (IMPORTANT): Make sure that no background processes that are not used continue to run while working on the projext. Use the `ps -aux` or `top` commands to do so. If background processes do not terminate or if the `cspro` server does not work, please contact the TAs as soon as possible (sogang_sp2025@googlegroups.com).

CSPRO server assignments (IMPORTANT): Ensure that you are using the correct servers for both the client and server that was assigned to your class, as well as the individual port that you have been assigned. Refer to the assignment page for more details on ports.

- Class 1 (CSE4100-01, AIE4050-01): Use `cspro9` for client, `cspro` for server.
- Class 2 (CSE4100-02, AIE4050-02): Use `cspro10` for client, `cspro` for server.

Hand-In Specifications: The submission should contain only source code file(s), `include` file(s), Makefile(s), the `stock.txt` file, and the performance report `.pdf` file. No executable program should be included. The TAs will automatically rebuild your shell program from the provided source code.

Please strictly follow the submission instruction below. If you not, you may get a penalty on your overall project score.

1. Only submit one archive file. The submitted archive file (`.tar`) should be named `prj3_`*your_student_id*`.tar`.
2. When the `tar` file is extracted, the name of the root directory of the tar file should be your student id.
3. You should create directories for each phase with the name of `task1/2`.
4. Each directory for each phase should include a Makefile, and source codes. You also have to submit one document for the performance evaluation and analysis task.

Attachment File:

- Task I folder (source code(csapp.c/h, echo.c, multiclient.c, stockclient.c, stockserver.c), Makefile, stock.txt) (30 points)
- Task II folder (source code(csapp.c/h, echo.c, multiclient.c, stockclient.c, stockserver.c), Makefile, stock.txt) (30 points)
- Performance evaluation and performance report (30 points)

Following is an example of a submission.

```
$~> ls
prj3_20221234.tar
$~> tar -xvf prj3_20221234.tar
$~> ls
20221234  prj3_20221234.tar
$~> tree 20221234
20221234
├── document.pdf
├── task1
│   ├── Makefile
│   ├── csapp.c
│   ├── csapp.h
│   ├── echo.c
│   ├── multiclient.c
│   ├── stock.txt
│   ├── stockclient.c
│   └── stockserver.c
└── task2
    ├── Makefile
    ├── csapp.c
    ├── csapp.h
    ├── echo.c
├── multiclient.c
├── stock.txt
├── stockclient.c
└── stockserver.c
```

All students are requested to upload the archived source files on Cyber Campus.

Note: Please make sure to compile your source code on CSPRO server, as the TA will build and compile your submitted project on that server. If the submitted code does not compile it cannot be scored!!!

9 Miscellaneous

9.1 Best practices the system programmers must comply to

WHAT YOU MUST DO:

1. You must read Chapter 12 (Concurrent Programming) of the textbook in its entirety, as well as the class slides related to said chapter, to fully understand, learn and complete this project.
2. Read the complete project specifications before programming your shell project.
3. The `Makefile` needs to include all dependencies (libraries etc) required to build your program. If your code does not compile after running `make` inside each task directory (i.e a compile error), you will not receive any marks for that task.
4. Source code commenting is a professional programmer's practice, and a must-do in this project as well.

WHAT YOU MUST NOT DO:

1. Do not hand-in any binary or object code files in your source code directory.
2. Do not include any hardcoded paths in your makefile.
3. Do not copy or share your source code, it is strictly prohibited otherwise you will be given penalty for such an action.