

System Programming Project 2

담당 교수 : 김영재 교수님

이름 : 이건영

학번 : 20181662

1. 개발 목표

여러 client의 동시 접속 및 서비스가 가능한 concurrent 주식 서버를 Event 기반, Thread 기반의 두 가지 방식으로 구축하고 그 성능을 평가한다.

2. 개발 범위 및 내용

A. 개발 범위

1. Task 1: Event-driven Approach

Server의 단일 프로세스 내에서 connfd들을 array에 관리한다. Select 함수를 사용해서 pending input이 있는 지 확인한 후, 만약 있다면 event로 간주하여 해당 input에 맞는 기능을 수행한다.

2. Task 2: Thread-based Approach

Server가 여러 개의 thread들을 만들어서 client의 연결 요청을 해당 thread로 concurrent하게 처리한다. 각 client들의 connfd들은 sbuf를 통해 관리된다. Client의 요청은 thread 함수 내에서 호출되는 echo_cnt 함수에 의해 처리된다.

3. Task 3: Performance Evaluation

Multiclient 프로그램을 통해 client의 수, 요청하는 명령어의 수 등을 변화시켜가며 각 방식으로 구현된 stock server의 성능을 평가한다.

B. 개발 내용

- Task1 (Event-driven Approach with select())

✓ Multi-client 요청에 따른 I/O Multiplexing 설명

Server는 하나의 listenfd와 배열에 저장된 connfd들을 관리한다. 이후 pending input이 있는 fd들을 select 함수로 확인한 후 listenfd에 들어온 요청은 add_client 함수를 통해 connfd 배열에 새로운 fd를 추가하는 것으로 해결하고, connfd 배열 중 하나의 fd에 들어온 요청(show, buy, sell)은 check_clients 함수를 통해 그에 맞는 작업을 수행하는 것으로 해결한다. 이 때 select는 인자로 받은 pool.maxfd+1 만큼의 fd들을 관찰하여 I/O Multiplexing을 구현하는 역할을 한다.

✓ epoll과의 차이점 서술

select의 경우 대상이 되는 fd들을 모두 확인해야하는 overhead가 발생한다. Epoll 함수는 관찰의 대상이 되는 fd들을 관리하는 저장소를 OS 레벨에서 관리하며, 해당 저장소에 관한 fd를 요청하여 전체 fd를 순회하는 문제점을 해결할 수 있다.

- **Task2 (Thread-based Approach with pthread)**

✓ Master Thread의 Connection 관리

Client가 연결 요청을 보내면 master thread에서 Accept 함수를 통해 connfd를 설정한 뒤 해당 connfd를 sbuf_insert 함수를 통해 sbuf에 넣어서 관리한다.

✓ Worker Thread Pool 관리하는 부분에 대해 서술

Client에 연결된 thread들은 Pthread_detach 함수를 통해 detached mode로 실행되며, sbuf에 있던 connfd를 sbuf_remove 함수로 넘겨받아 echo_cnt(connfd)를 통해 client의 요청에 따른 결과를 출력한 후 종료된다.

- **Task3 (Performance Evaluation)**

✓ 얻고자 하는 metric 정의, 그렇게 정한 이유, 측정 방법 서술

Server가 client들의 요청을 동시에 얼마나 처리할 수 있는지 나타내기 위해

$$\text{동시처리율} = \frac{\text{Client의 수} \times \text{Client 당 명령어의 수}}{\text{Multiclient의 총 실행 시간}} \text{인 동시처리율을 구하였다.}$$

실행 시간은 time.h의 gettimeofday 함수를 이용하여 elapsed time을 측정하는 방식으로 구하였다.

✓ Configuration 변화에 따른 예상 결과 서술

실행 결과에 영향을 주는 configuration 은 최대 client 의 수, client 당 요청하는 명령의 수, 총 주식의 수로 예상되며, 세 요인이 증가하면 실행 시간도 증가할 것으로 예상된다. 그러나 Event-driven 과 Thread-driven 방식의 구현에 차이가 존재하기 때문에 증가하는 정도에 차이가 있을 것으로 예상된다.

C. 개발 방법

기본적인 구현의 틀은 강의자료를 참조하였다. (Event-driven : ch6 p.23~27, Thread-driven : ch8 p.7~18)

- 공통

주식 정보를 저장하는 구조체인 item을 선언하였다. 주식의 정보(ID, 수량, 가격) 외에 변수의 sharing에서 발생할 수 있는 문제를 방지하기 위한 readcnt와 semaphore인 mutex, w를 가지고 있다.

주식 정보를 저장하는 tree는 binary search tree로 구성하였다. Node 구조체에는 item 정보를 담은 data와 left, right pointer를 가진다.

Tree에 관련된 함수로는 새로운 노드를 삽입하는 insertNode, buy나 sell의 대상을 찾는 search를 구현하였다.

Buy, sell 함수는 입력받은 주식의 ID와 수량을 바탕으로 거래를 진행하며, 그 결과를 요청한 client의 connfd에 전달하는 방식으로 구현하였다.

Show 함수는 tree를 inorder 방식으로 순회하며 출력을 위한 showBuf에 각 node의 정보를 입력해둔 뒤 한번에 connfd에 전달하는 방식으로 구현하였다.

Stock.txt의 정보를 갱신하는 updateFile 함수는 클라이언트의 연결이 종료될 경우 호출되며, 마찬가지로 inorder 방식으로 순회하며 node별로 정보를 저장한다. 이 과정에서 ID를 기준으로 오름차순으로 정렬된다.

처음 stockserver를 실행하면 stock.txt의 정보를 읽어와서 줄 별로 item 구조체 변수에 정보를 담아 insertNode로 저장한다. 이후 각 방식에 맞게 client의 요청을 처리한다.

- Task 1: Event-driven Approach

강의자료의 기본 구조를 활용하여 stockserver.c를 작성하였다. Pool 구조체를 사용하였고, main의 while loop 내에서 select 함수를 통해 요청이 들어온 client를 관리하여 최종적으로 check_clients 함수를 통해 요청을 처리하였다.

Check_clients 함수는 현재 ready 상태의 fd들을 순회하며 들어온 입력을 처리한다. 입력의 첫 문자열을 command로 분류하여 그에 맞는 함수로 연결되도록 하였고, EOF나 "exit"을 받은 경우 해당 client와의 연결을 종료하고 pool의 정보도 수정한 후,

stock.txt 파일에 정보를 update 해 주었다.

- Task 2: Thread-driven Approach

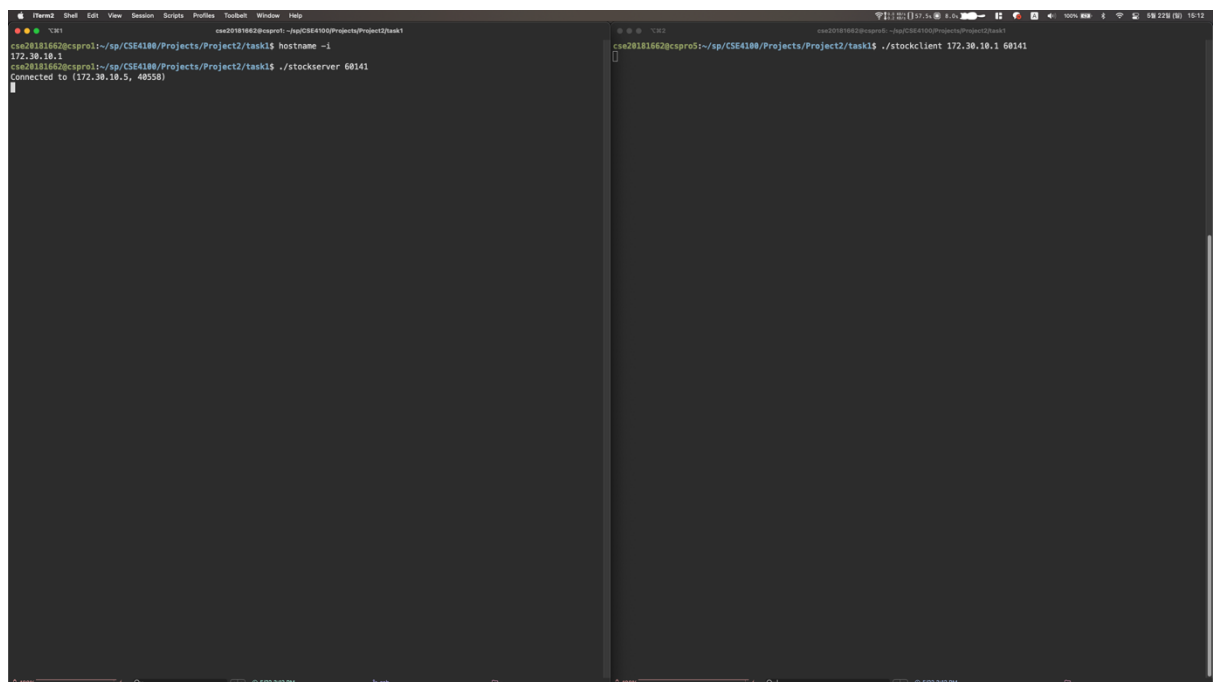
서버 실행 시 미리 지정된 NTHREAD 만큼 pthread가 생성된다. 이후 main의 while loop에서 새로운 client의 연결 요청이 오면 Accept 후 sbuf 내에 connfd를 추가하고 thread 함수로 client의 요청을 처리한다. Event-driven에서 check_clients 함수와 비슷한 역할을 하는 echo_cnt 함수에서 명령어에 대한 처리를 하고, EOF나 "exit" 명령어가 입력될 경우 updateFile을 통해 stock.txt 파일을 업데이트 한다.

Event-driven과 다른 점은 여러 thread가 concurrent하게 tree에 접근하여 생길 수 있는 문제를 해결하기 위해 semaphore를 활용하였다는 점이다. 이 때 Readers-Writers problem을 고려하여 tree를 읽는 함수인 show와 updateFile에서는 mutex와 reacnt, w를 이용하였다. Write 부분인 buy와 Sell에서는 w만을 사용하였다. (강의자료 ch8 13p 참조)

3. 구현 결과

- Task 1: Event-driven Approach

Server 실행 후 client로 server에 연결한 모습이다.



The image shows two terminal windows side-by-side. The left window is a terminal titled 'Terminal' with a dark background. It shows the execution of a stock server program. The prompt is 'cse20181862@csprail:~/sp/CSE4100/Projects/Project2/task1'. The user enters 'hostname -i', which outputs '172.30.10.1'. Then the user enters 'cse20181862@csprail:~/sp/CSE4100/Projects/Project2/task1\$./stockserver 60141', which outputs 'Connected to (172.30.10.5, 40556)'. The right window is also a terminal titled 'Terminal'. It shows the execution of a stock client program. The prompt is 'cse20181862@csprail:~/sp/CSE4100/Projects/Project2/task1'. The user enters 'cse20181862@csprail:~/sp/CSE4100/Projects/Project2/task1\$./stockclient 172.30.10.1 60141', which outputs an empty line.

Show 명령어를 통해 현재 주식의 상태를 출력한 모습이다.

```
csa20181862@cspr01:~/sp/CSE4180/Projects/Project2/task1$ hostname -i
172.30.10.1
csa20181862@cspr01:~/sp/CSE4180/Projects/Project2/task1$ ./stockserver 60141
Connected to (172.30.10.5, 40558)
received 5 bytes
[]

csa20181862@cspr05:~/sp/CSE4180/Projects/Project2/task1$ ./stockclient 172.30.10.1 60141
show
1 49 1000
2 35 20000
3 42 1200
4 38 5000
5 43 3700
6 6 3000
7 58 2500
8 27 1700
9 23 5500
10 14 1200
```

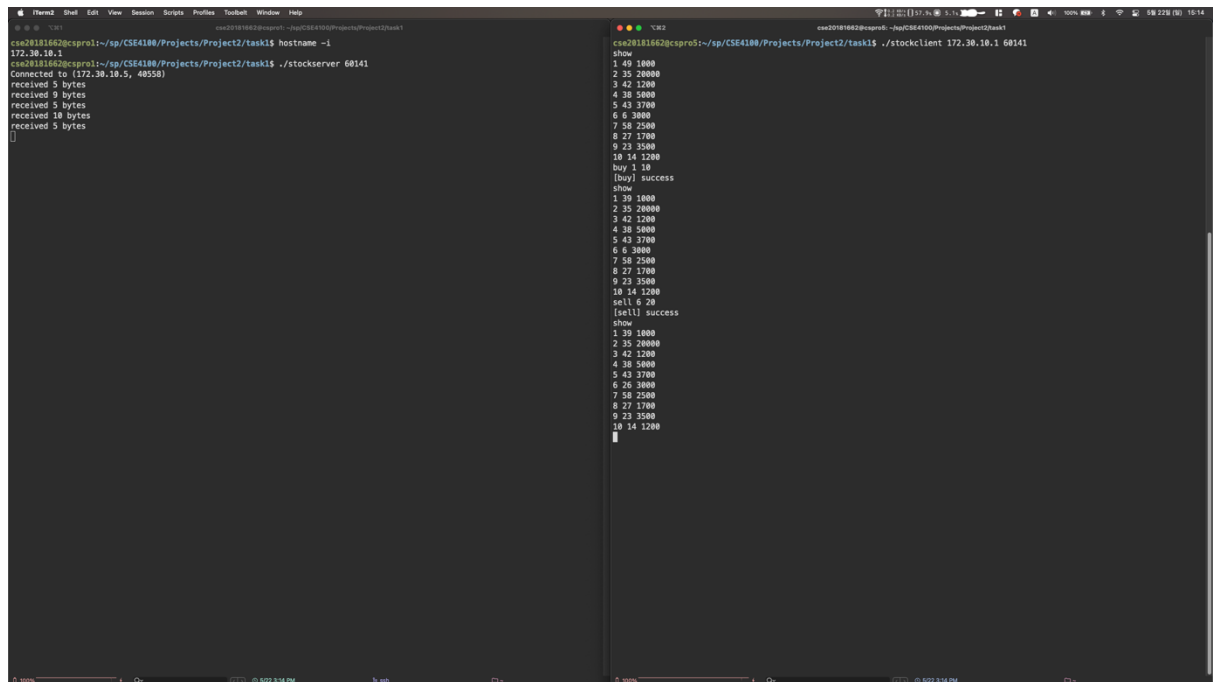
Buy 명령어로 주식을 구매한 후 show 명령어를 요청한 모습이다. 구매한 주식에 해당하는 잔여 수량이 줄어든 것을 볼 수 있다.

```
csa20181862@cspr01:~/sp/CSE4180/Projects/Project2/task1$ hostname -i
172.30.10.1
csa20181862@cspr01:~/sp/CSE4180/Projects/Project2/task1$ ./stockserver 60141
Connected to (172.30.10.5, 40558)
received 5 bytes
received 9 bytes
received 5 bytes
[]

csa20181862@cspr05:~/sp/CSE4180/Projects/Project2/task1$ ./stockclient 172.30.10.1 60141
show
1 49 1000
2 35 20000
3 42 1200
4 38 5000
5 43 3700
6 6 3000
7 58 2500
8 27 1700
9 23 5500
10 14 1200
buy 1 18
[buy] success
show
1 39 1000
2 35 20000
3 42 1200
4 38 5000
5 43 3700
6 6 3000
7 58 2500
8 27 1700
9 23 5500
10 14 1200
```

Sell 명령어로 주식을 판매한 후 show 명령어를 요청하였다. 판매한 주식의 잔여 수

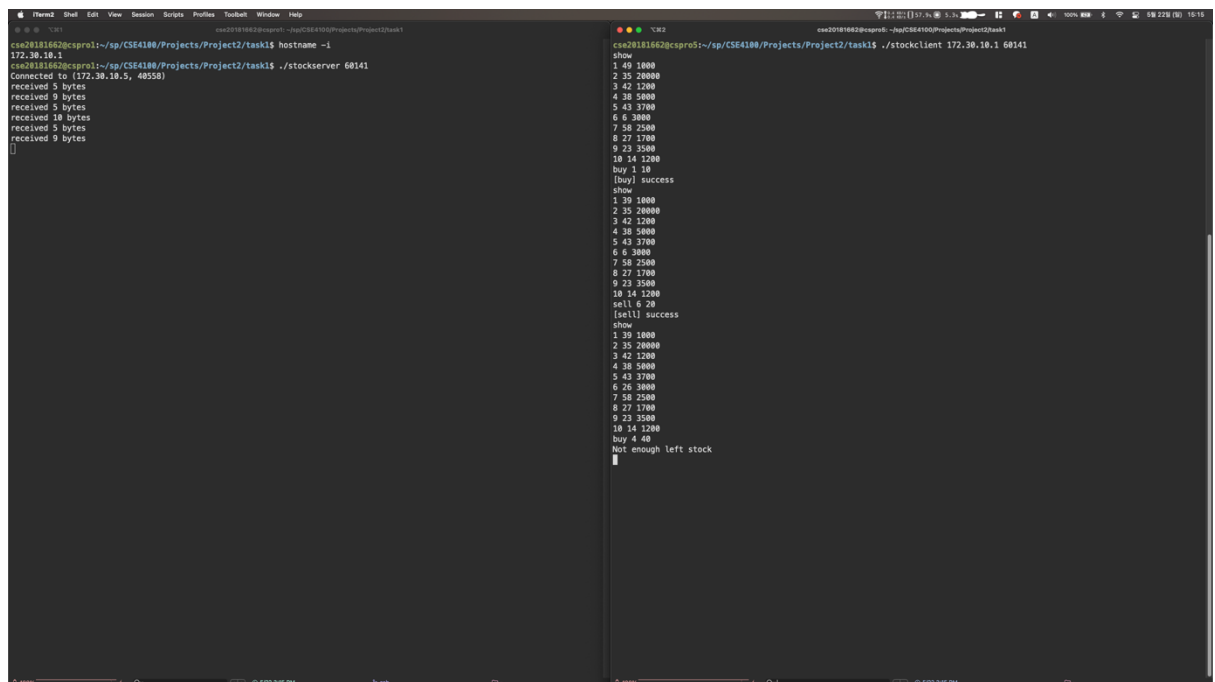
량이 늘어난 것을 확인할 수 있다.



```
csa20181662@csprail:~/sp/CS4100/Projects/Project2/task1$ hostname -i
172.30.10.1
csa20181662@csprail:~/sp/CS4100/Projects/Project2/task1$ ./stockserver 60141
Connected to (172.30.10.5, 40550)
received 5 bytes
received 9 bytes
received 5 bytes
received 10 bytes
received 5 bytes
[]

csa20181662@csprail:~/sp/CS4100/Projects/Project2/task1$ ./stockclient 172.30.10.1 60141
show
1 40 1000
2 35 20000
3 42 1200
4 38 5000
5 43 3700
6 6 3000
7 58 2500
8 27 1700
9 23 3500
10 14 1200
buy 1 10
[buy] success
show
1 39 1000
2 35 20000
3 42 1200
4 38 5000
5 43 3700
6 6 3000
7 58 2500
8 27 1700
9 23 3500
10 14 1200
sell 6 20
[sell] success
show
1 39 1000
2 35 20000
3 42 1200
4 38 5000
5 43 3700
6 6 3000
7 58 2500
8 27 1700
9 23 3500
10 14 1200
```

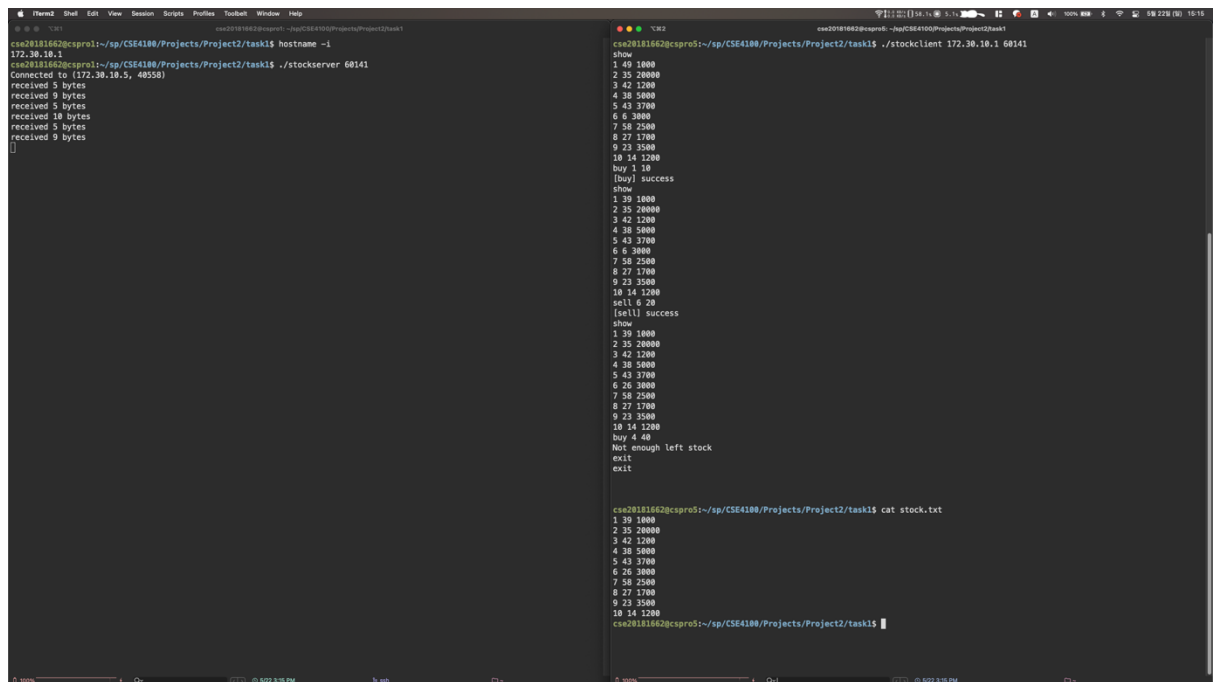
만약 구매하려는 주식의 수량이 충분하지 않을 경우 메시지를 출력하는 모습이다.



```
csa20181662@csprail:~/sp/CS4100/Projects/Project2/task1$ hostname -i
172.30.10.1
csa20181662@csprail:~/sp/CS4100/Projects/Project2/task1$ ./stockserver 60141
Connected to (172.30.10.5, 40550)
received 5 bytes
received 9 bytes
received 5 bytes
received 10 bytes
received 5 bytes
received 9 bytes
[]

csa20181662@csprail:~/sp/CS4100/Projects/Project2/task1$ ./stockclient 172.30.10.1 60141
show
1 40 1000
2 35 20000
3 42 1200
4 38 5000
5 43 3700
6 6 3000
7 58 2500
8 27 1700
9 23 3500
10 14 1200
buy 1 10
[buy] success
show
1 39 1000
2 35 20000
3 42 1200
4 38 5000
5 43 3700
6 6 3000
7 58 2500
8 27 1700
9 23 3500
10 14 1200
sell 6 20
[sell] success
show
1 39 1000
2 35 20000
3 42 1200
4 38 5000
5 43 3700
6 26 3000
7 58 2500
8 27 1700
9 23 3500
10 14 1200
buy 4 40
Not enough left stock
```

이후 exit 명령어를 통해 연결을 종료한 뒤, stock.txt의 내용이 update 된 모습을 확인할 수 있다.



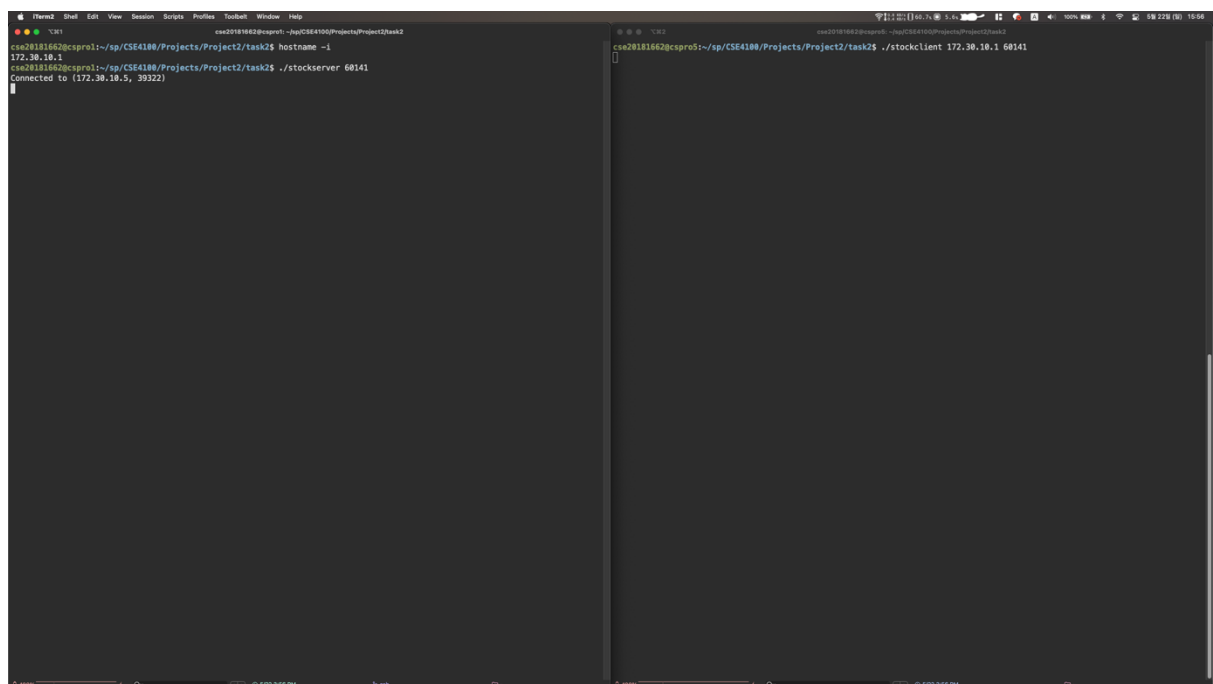
```
cse20181662@csprail:~/sp/CSE4100/Projects/Project2/task1$ hostname -i
172.30.10.1
cse20181662@csprail:~/sp/CSE4100/Projects/Project2/task1$ ./stockserver 60141
Connected to (172.30.10.5, 40550)
received 5 bytes
received 9 bytes
received 5 bytes
received 10 bytes
received 5 bytes
received 9 bytes
[]

cse20181662@csprail:~/sp/CSE4100/Projects/Project2/task2$ ./stockclient 172.30.10.1 60141
show
1 40 1000
2 35 20000
3 42 1200
4 38 5000
5 43 3700
6 6 3000
7 58 2500
8 27 1700
9 23 3500
10 14 1200
buy 1 10
[buy] success
show
1 39 1000
2 35 20000
3 42 1200
4 38 5000
5 43 3700
6 6 3000
7 58 2500
8 27 1700
9 23 3500
10 14 1200
sell 6 20
[sell] success
show
1 39 1000
2 35 20000
3 42 1200
4 38 5000
5 43 3700
6 26 3000
7 58 2500
8 27 1700
9 23 3500
10 14 1200
buy 4 40
Not enough left stock
exit
exit

cse20181662@csprail:~/sp/CSE4100/Projects/Project2/task2$ cat stock.txt
1 39 1000
2 35 20000
3 42 1200
4 38 5000
5 43 3700
6 26 3000
7 58 2500
8 27 1700
9 23 3500
10 14 1200
```

- Task 2: Thread-driven Approach

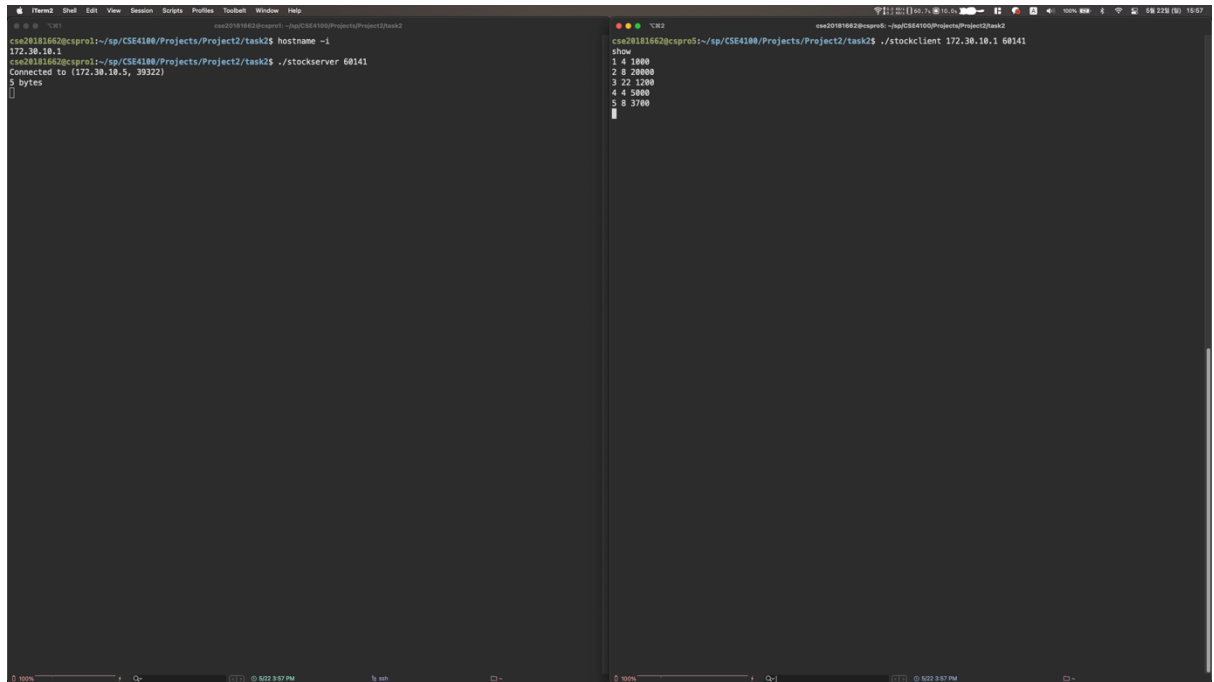
Server 실행 후 client가 연결된 모습이다.



```
cse20181662@csprail:~/sp/CSE4100/Projects/Project2/task2$ hostname -i
172.30.10.1
cse20181662@csprail:~/sp/CSE4100/Projects/Project2/task2$ ./stockserver 60141
Connected to (172.30.10.5, 39322)
[]

cse20181662@csprail:~/sp/CSE4100/Projects/Project2/task2$ ./stockclient 172.30.10.1 60141
[]
```


Show 명령어로 현재 서버의 주식 정보를 확인하는 모습이다.

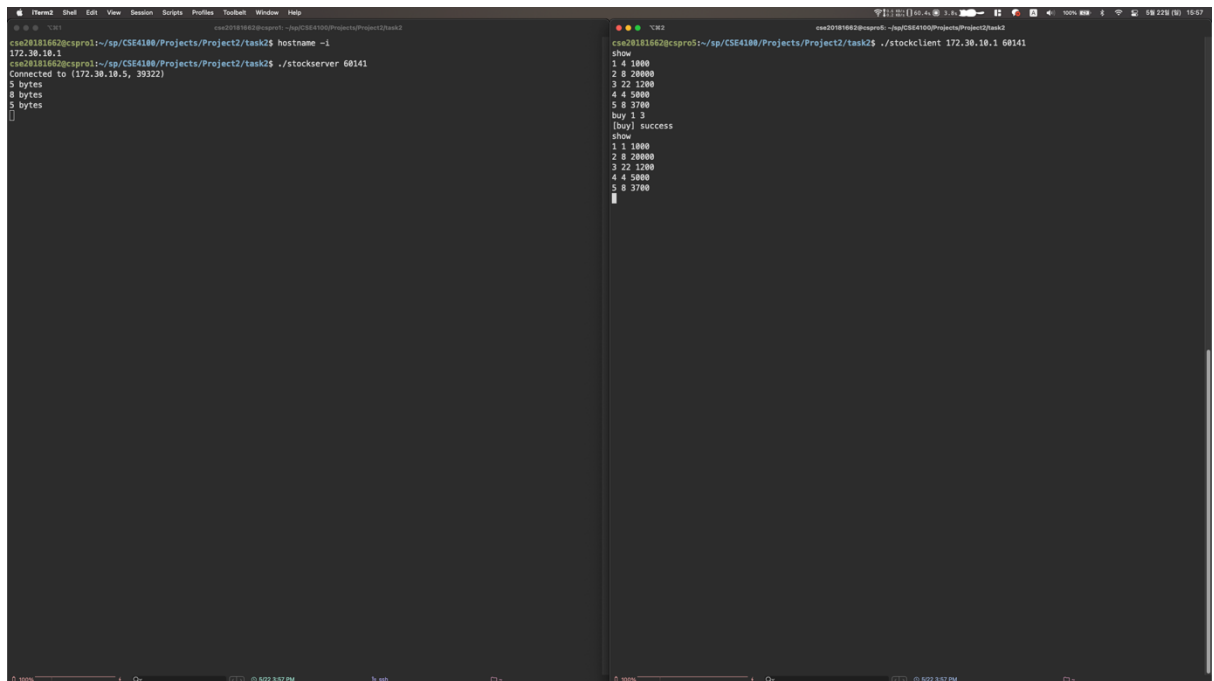


The image shows two terminal windows side-by-side. The left window is a terminal session on a host named 'cse4100' with IP '172.30.10.1'. It shows the user running 'hostname -i' and then connecting to a stock server at '172.30.10.5, 39322'. The right window is a terminal session on a host named 'cse4100' with IP '172.30.10.1'. It shows the user running './stockclient 172.30.10.1 68141' and then 'show'. The output of 'show' is a list of stock prices: 1 4 1000, 2 8 20000, 3 22 1200, 4 4 5000, 5 8 3700.

```
cs@20181162@csprail:~/sp/CSE4100/Projects/Project2/task2$ hostname -i
172.30.10.1
cs@20181162@csprail:~/sp/CSE4100/Projects/Project2/task2$ ./stockserver 68141
Connected to (172.30.10.5, 39322)
5 bytes
[]

cs@20181162@csprail:~/sp/CSE4100/Projects/Project2/task2$ ./stockclient 172.30.10.1 68141
show
1 4 1000
2 8 20000
3 22 1200
4 4 5000
5 8 3700
[]
```

Buy 명령어로 주식을 구매한 후 주식 목록을 확인하여 잔여 수량에 변화가 생긴 것을 알 수 있다.

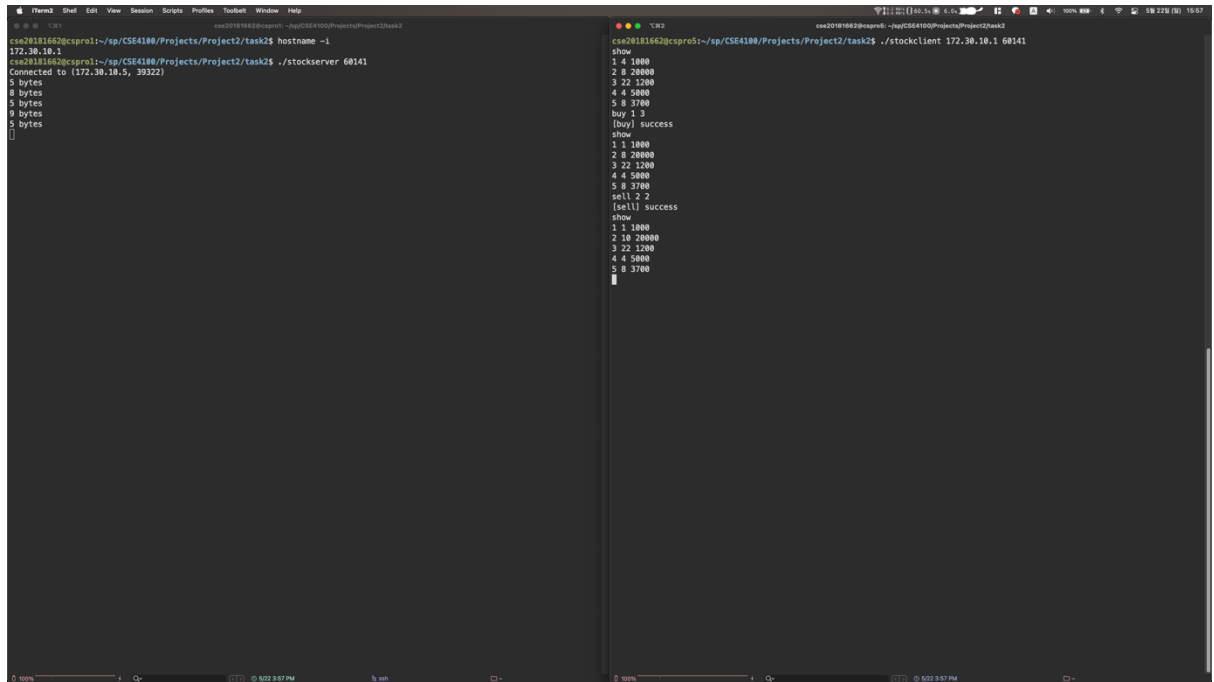


The image shows two terminal windows side-by-side. The left window is a terminal session on a host named 'cse4100' with IP '172.30.10.1'. It shows the user running 'hostname -i' and then connecting to a stock server at '172.30.10.5, 39322'. The right window is a terminal session on a host named 'cse4100' with IP '172.30.10.1'. It shows the user running './stockclient 172.30.10.1 68141' and then 'show'. The output of 'show' is a list of stock prices: 1 1 1000, 2 8 20000, 3 22 1200, 4 4 5000, 5 8 3700. Then the user runs 'buy 1 3' and the output is '[buy] success'. Finally, the user runs 'show' and the output is a list of stock prices: 1 1 1000, 2 8 20000, 3 22 1200, 4 4 5000, 5 8 3700.

```
cs@20181162@csprail:~/sp/CSE4100/Projects/Project2/task2$ hostname -i
172.30.10.1
cs@20181162@csprail:~/sp/CSE4100/Projects/Project2/task2$ ./stockserver 68141
Connected to (172.30.10.5, 39322)
5 bytes
[]

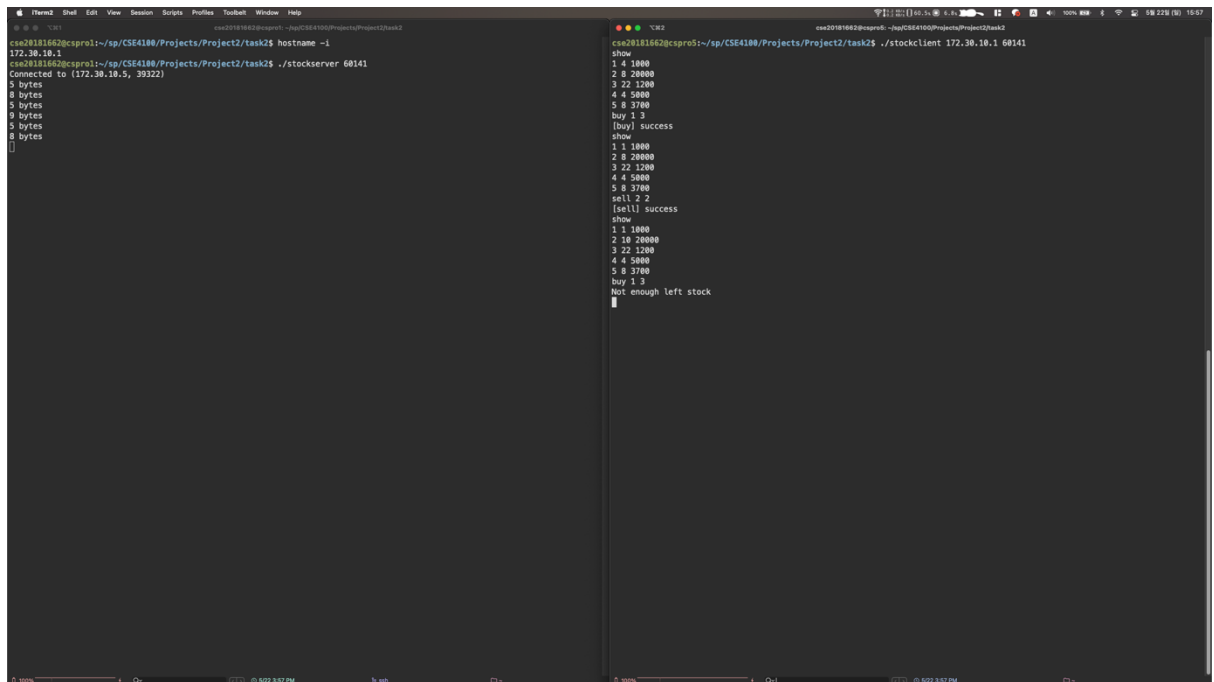
cs@20181162@csprail:~/sp/CSE4100/Projects/Project2/task2$ ./stockclient 172.30.10.1 68141
show
1 1 1000
2 8 20000
3 22 1200
4 4 5000
5 8 3700
buy 1 3
[buy] success
show
1 1 1000
2 8 20000
3 22 1200
4 4 5000
5 8 3700
[]
```

Sell 명령어로 주식 판매 시에도 잔여 수량의 변동이 확인되는 모습이다.



```
csa20181662@csprail:~/sp/CS4100/Projects/Project2/task2$ hostname -i
172.30.10.1
csa20181662@csprail:~/sp/CS4100/Projects/Project2/task2$ ./stockserver 60141
Connected to (172.30.10.5, 39322)
5 bytes
8 bytes
5 bytes
9 bytes
5 bytes
csa20181662@csprail:~/sp/CS4100/Projects/Project2/task2$ ./stockClient 172.30.10.1 60141
show
1 4 1000
2 8 20000
3 22 1200
4 4 5000
5 8 3700
buy 1 3
[buy] success
show
1 1 1000
2 8 20000
3 22 1200
4 4 5000
5 8 3700
sell 2 2
[sell] success
show
1 1 1000
2 10 20000
3 22 1200
4 4 5000
5 8 3700
```

잔여 수량보다 많은 주식은 구매하지 못하는 모습이다.



```
csa20181662@csprail:~/sp/CS4100/Projects/Project2/task2$ hostname -i
172.30.10.1
csa20181662@csprail:~/sp/CS4100/Projects/Project2/task2$ ./stockserver 60141
Connected to (172.30.10.5, 39322)
5 bytes
8 bytes
5 bytes
9 bytes
5 bytes
csa20181662@csprail:~/sp/CS4100/Projects/Project2/task2$ ./stockClient 172.30.10.1 60141
show
1 4 1000
2 8 20000
3 22 1200
4 4 5000
5 8 3700
buy 1 3
[buy] success
show
1 1 1000
2 8 20000
3 22 1200
4 4 5000
5 8 3700
sell 2 2
[sell] success
show
1 1 1000
2 10 20000
3 22 1200
4 4 5000
5 8 3700
buy 1 3
Not enough left stock
```

The image displays two terminal windows side-by-side, illustrating a stock server simulation.

Left Terminal Window:

```
cse20181662@csp01: ~/sp/CSE4100/Projects/Project2/task2
$?22.30.10.1
cse20181662@csp01:~/sp/CSE4100/Projects/Project2/task2$ hostname -i
Connected to (172.30.10.5, 39322)
5 bytes
0 bytes
5 bytes
0 bytes
5 bytes
0 bytes
5 bytes
0

```

Right Terminal Window:

```
cse20181662@csp05:~/sp/CSE4100/Projects/Project2/task2$ ./stockclient 172.30.10.1 6041
show
1 1 1000
2 10 20000
3 22 1200
4 4 5000
5 8 3700
buy 1 3
[buy] success
show
1 1 1000
2 10 20000
3 22 1200
4 4 5000
5 8 3700
sell 2 2
[sell] success
show
1 1 1000
2 10 20000
3 22 1200
4 4 5000
5 8 3700
Not enough left stock
exit

cse20181662@csp05:~/sp/CSE4100/Projects/Project2/task2$ cat stock.txt
1 1 1000
2 10 20000
3 22 1200
4 4 5000
5 8 3700
cse20181662@csp05:~/sp/CSE4100/Projects/Project2/task2$
```

시간은 $end - start$ 로 계산하였으며, $start$ 는 multicient의 main의 while loop 시작 직전에 측정하였고 end 는 multicient가 종료되기 직전으로 측정하였다.

```

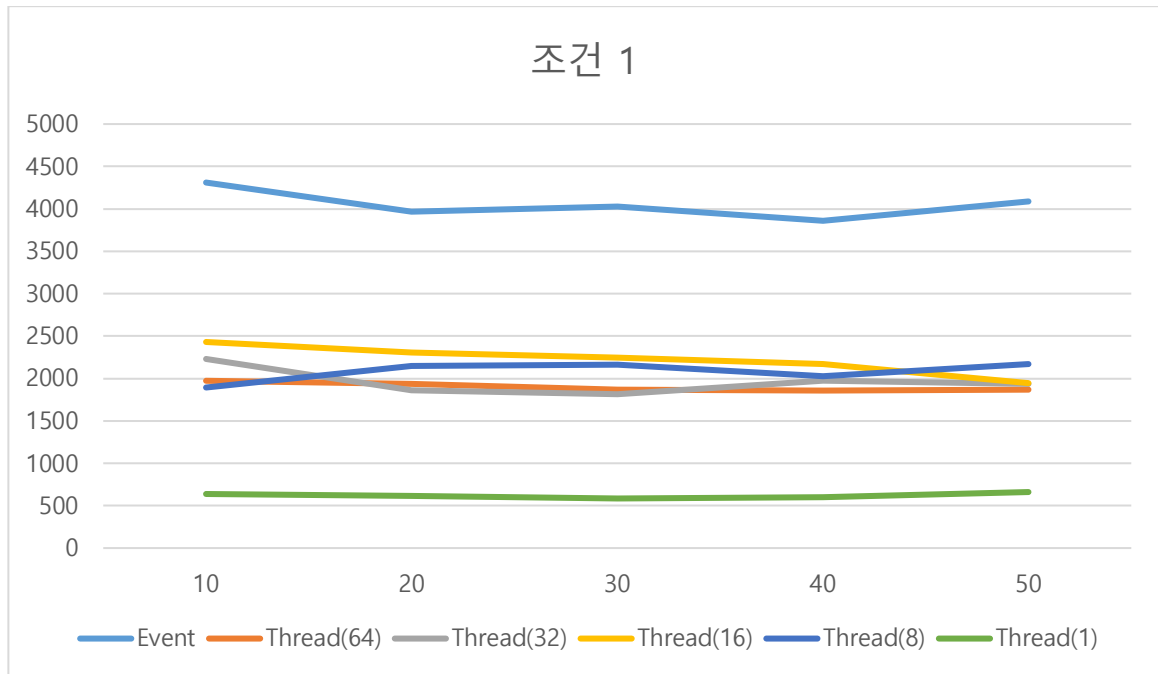
root@8091882cgsprol:~# sshCSE4100/Projects/Project2/task25 -u multient 172.30.16.1 68414 2
ch1g 173000
ch1g 173010
[hoj] success
1 46 1000
2 131 20000
3 16 1200
4 306 5000
5 359 3700
1 46 1000
2 131 20000
3 16 1200
4 306 5000
5 359 3700
[hoj] success
[hoj] success
[hoj] success
1 49 1000
2 123 20000
3 11 1200
4 306 5000
5 359 3700
1 49 1000
2 123 20000
3 11 1200
4 306 5000
5 359 3700
[hoj] success
[hoj] success
[hoj] success
1 49 1000
2 123 20000
3 28 1200
4 306 5000
5 364 3700
[hoj] success
[hoj] success
[hoj] success
1 49 1000
2 128 20000
3 28 1200
4 306 5000
5 362 3700
1 49 1000
2 128 20000
3 26 1200
4 306 5000
5 362 3700
[hoj] success
[hoj] success
[hoj] success
1 54 1000
2 178 20000
3 26 1200
4 306 5000
5 362 3700
elapsed time: 14628 microseconds
root@8091882cgsprol:~# sshCSE4100/Projects/Project2/task25

```

그래프의 가로축은 client의 수, 세로축은 동시처리율(number of commands / sec) 이

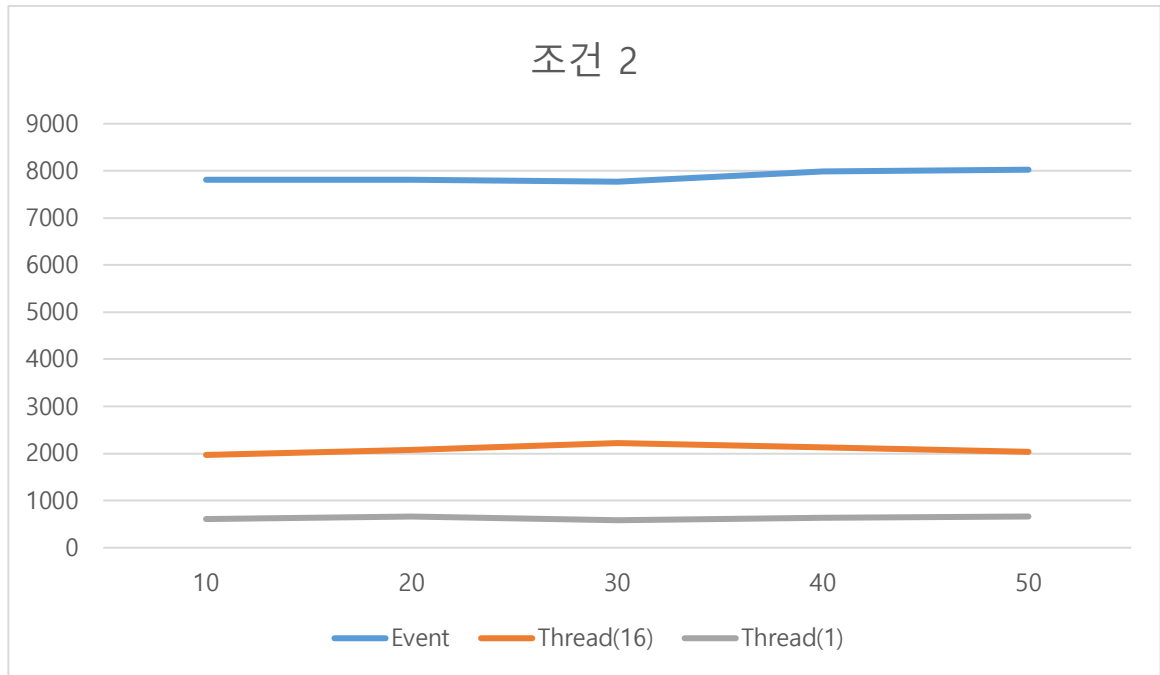
다.

조건 1 : 주식의 개수 = 5, client 당 요청 = 10, 명령어 = show, buy, sell



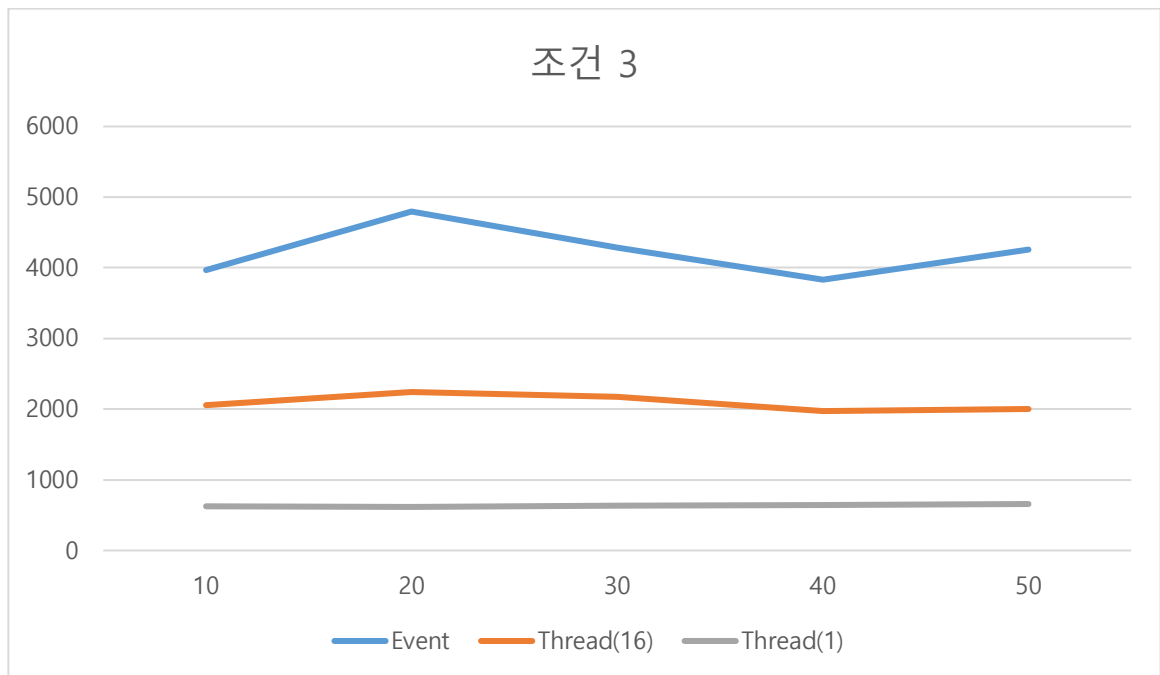
측정 결과 Event 기반 방식의 서버가 Thread 기반 방식의 서버보다 더 좋은 동시처리율을 보여주는 것으로 나타났다. Thread 기반의 방식에서 NTHREAD 값을 변경하여 측정한 결과는 단일 Thread일 경우 동시처리율이 현저하게 떨어졌으며, 평균적으로 가장 좋은 동시처리율을 보인 경우는 16 Thread인 경우였다. 이외에 client 수와 동시처리율 간에는 큰 상관관계가 없는 것을 확인할 수 있었다.

조건 2 : 주식의 개수 = 5, client 당 요청 = 30, 명령어 = show, buy, sell



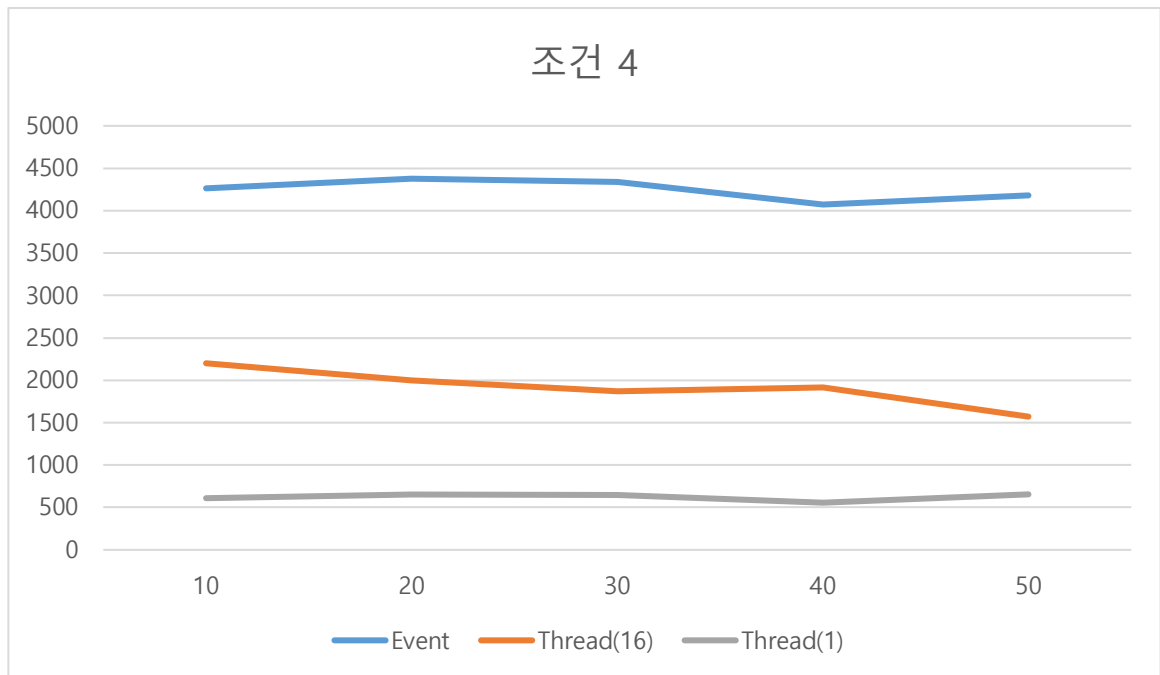
조건 1에서 client당 보내는 요청의 수를 늘렸을 때의 결과이다. Event 기반의 경우 조건 1보다 더욱 높은 동시처리율을 보여주었다. Thread 기반의 경우 조건 1보다 동시처리율이 저하되었는데 이는 Event 기반에 비해 Semaphore, pthread의 관리 등 추가적인 overhead가 많기 때문인 것으로 추정된다.

조건 3 : 주식의 개수 = 5, client 당 요청 = 10, 명령어 = buy, sell



조건 3에서는 명령어 중 buy와 sell만 입력으로 들어오도록 설정한 후 측정하였다. Event 기반과 Thread 기반 모두 이전과 큰 차이가 없는 것을 확인할 수 있었다.

조건 4 : 주식의 개수 = 5, client 당 요청 = 10, 명령어 = show



이번에는 명령어로 show 명령만 들어오는 경우를 측정하였다. Event 기반에서는 조건 3과 큰 차이가 없었으나 Thread 기반에서는 조건 3보다 낮아진 동시처리율을 확인할 수 있었다. 이는 show 명령어는 Readers-Writers 문제를 해결하기 위해 semaphore를 이중으로 lock-unlock하는 방식을 적용하여 overhead가 다른 명령어보다 크기 때문으로 추정된다.