System Programming Project 3

담당 교수 : 박성용

이름 : 이윤기

학번 : 20192136

1. 개발 목표

- 해당 프로젝트에서 구현할 내용을 간략히 서술.
- (주식 서버를 만드는 전체적인 개요에 대해서 작성하면 됨.)

이번 프로젝트의 목표는 서버에 여러 client들이 동시접속 가능한 주식 서버를 만드는 것이 목표이다. 여러 client가 서버에 concurrent하게 접속 가능하게 server를 Event-based 방식과 thread-based방식 두가지 방식으로 구현한다. Event 기반의 서버의 경우 select를 사용하여 thread기반은 스레드 pool을 미리 만들어 두어 접속이 들어오면 pool에 할당하도록 한다. 서버의 기능은 show, buy, sell이 있고 stock.txt파일에 내용을 바이너리 트리 형태로 저장 관리한다.

2. 개발 범위 및 내용

A. 개발 범위

- 아래 항목을 구현했을 때의 결과를 간략히 서술

1. Task 1: Event-driven Approach

   주식 서버를 열고 client가 서버에 연결을 요청하여 연결이 된다. 그 후 select명령어를 통해 event가 발생한 fd를 찾아 해당 client로부터 받은 명령들을 서버에 그대로 출력하고 명령에 맞는 기능을 수행한 후 적절한 결과를 출력한다.

2. Task 2: Thread-based Approach

   Event기반 서버와 유사하게 작동하지만 client를 처리하는 과정에서 미리 스레드를 만들어 두고 master thread에서 적절하게 분배하여 분배 받은 스레드가 기능을 수행한다.

3. Task 3: Performance Evaluation

   스레드 기반의 서버가 이벤트 기반서버에 비해 좋은 기능을 보임을 알 수 있었다. Client의 개수가 늘어날수록 이벤트 기반에 비해 스레드 기반의 서버가 좋은 성능을 보였다.

B. 개발 내용

- 아래 항목의 내용만 서술
- (기타 내용은 서술하지 않아도 됨. 코드 복사 붙여 넣기 금지)
- **Task1 (Event-driven Approach with select())**
  - ✓ Multi-client 요청에 따른 I/O Multiplexing 설명

    서버는 select함수를 사용하여 event 발생 유무를 확인하는데 이때 read_set 대신에 ready_set이라는 복사본을 이용한다. 그 후 event가 발생한 fd와 accept로 연결하고 add_client를 이용하여 client목록에 넣는다. 그 후 check_client함수 내부에서 명령에 맞는 기능들을 수행한다.
  - ✓ epoll과의 차이점 서술

    select방식의 경우 반복문을 사용하여 계속해서 event가 발생한 fd들을 탐색해야하는 overhead가 있다. 그러나 epoll의 경우 리눅스 환경에서 사용가능하며 os가 직접 fd들을 관리해주기 때문에 반복해서 fd들을 확인할 필요가 없어 속도가 빠르다는 차이가 있다.
- **Task2 (Thread-based Approach with pthread)**
  - ✓ Master Thread의 Connection 관리

    thread기반 서버의 경우 NTHREAD 값에 맞게 미리 thread들을 만들어 둔다. 이후 반복문을 통해 accpet함수로 연결을 기다린다. 연결이 되면 sbuf_insert 함수를 사용하여 삽입하고 기능을 수행한다.
  - ✓ Worker Thread Pool 관리하는 부분에 대해 서술

    Client의 명령에 맞는 기능을 수행하기 위해 미리 만들어 두었던 thread들 중에 하나가 thread함수를 통해 기능을 수행한다. 이때 Pthread_detach를 사용하여 연결을 끊고 sbuf_remove로 fd목록중에서 하나를 가져와 수행한다.
- **Task3 (Performance Evaluation)**
  - ✓ 얻고자 하는 metric 정의, 그렇게 정한 이유, 측정 방법 서술

    시간당 처리량을 비교하기 위해 CLIENT_NUM * ORDER_PER_CLIENT / eplapsed_time을 계산할 것이다. 우선 CLIENT_NUM에 변화를 주면서 접속

수에 따른 처리량을 비교하고 workload에 따라 show,buy,sell로 명령어 별로 하나씩 고정시켜 처리량을 비교할 것이다.

✓ Configuration 변화에 따른 예상 결과 서술

C. 개발 방법

- **B.의 개발 내용을 구현하기 위해 어느 소스코드에 어떤 요소를 추가 또는 수정할 것인지 설명. (함수, 구조체 등의 구현이나 수정을 서술)**

우선 정보들을 프로그램 시작하고 나서 stock.txt에서 파일을 읽어 binary tree형태로 저장을 한다. 그러기 위해 item구조체에 struct * item left,right를 추가한다. 트리를 이용할 때 사용하는 함수로는 show,insert, search, printStock이 있다. Insert는 이진트리의 삽입 기능을 search는 특정 노드 탐색, show와printStock은 traversal기능을 갖는다.

서버의 경우 두 방식 모두 ctrl + c를 입력하면 종료 되도록 만들었다.
Sigint_handler를 만들어서 강제종료 되는경우에 최신 상태의 트리를 printStock을 사용하여 출력하도록 만들었다.

Show, printStock

Show와 printStock모두 inorder traversal의 기능을 갖도록 구현하였다. Show의 경우 temp문자열을 선언하여 노드를 돌아다니면서 sprintf로 지정된 형태로 str에 연결해주었다. printStock의 경우는 파일 출력을 수행하기 위해 인자로 전달받은 fd에 출력을 수행한다.

Buy,sell

Buy와 sell모두 입력으로 받은 id에 맞는 노드를 찾기 위해 search함수를 사용했다. 이진트리의 특성에 맞게 크기 비교를 해가면서 아래 노드로 내려가며 노드를 찾고 없는 경우와 찾아도 수량이 맞지 않는 경우 조건문을 사용하여 알맞은 출력을 수행하도록 하였다. 찾고자 하는 노드가 있는 경우 left_stock을 update해주었다.

Event_based

이벤트 기반의 경우 check_clients함수에서 명령어에 따른 기능들을 수행하도록 구현하였다. 반복문을 사용하여 pool의 fd들을 순차적으로 확인하면서 event가 발생한 fd의 경우 작업을 수행했다. 명령어들을 rio_readlineb로 buf에 받아 명령어 별로 지정된 위치(잘못된 입력은 고려하지 않기 때문)에 id와 개수를 atoi함수를 사용해 정수형으로 바꿔 위에서 설명한 함수들에 인자들로 전달을 해주었다.

Thread_based

Event-based와 전체적인 기능은 유사하지만 연결들을 처리하는 방식에 차이가 있다. 해당 서버는 func함수를 thread함수 내부에서 수행하도록 하였다. Thread의 경우 전역변수를 공유하기 때문에 critical section이 되기 때문에root, byte_cnt 를 사용하는 경우 semaphore를 사용하여 P,V함수로 thread들이 배타적으로 작동할 수 있도록 하였다.

3. 구현 결과

- **2번의 구현 결과를 간략하게 작성**
- **미처 구현하지 못한 부분에 대해선 디자인에 대한 내용도 추가**

이 여러 client가 들어와 connect요청을 하면 서버가 요청을 받아 연결을 수행한다. 그 후 client가 명령을 전달하면 그 명령을 받아 stock.txt로부터 받아 tree형태로 저장한 데이터를 기반으로 명령을 수행한다. Show 명령을 입력한다면 stock의 목록들이 id오름차순으로 출력된다. Buy(sell) id count를 입력한다면 id에 맞는 주식을 찾아 없다면 오류 메시지를 있는데 수량이 적절하지 않다면 또 다른 오류 메세지를 출력하고 적절한 입력의 경우 buy(sell) success를 출력한다. CLIENT_NUM과 ORDER_PER_CLIENT의 개수에 맞게 명령을 수행하고 끝나면 client는 종료되고 서버의 경우 ctrl + c를 입력해주면 stock.txt파일을 update한 후에 종료된다.

디자인 적으로 구현하지 못한 부분으로는 예시 pdf에서 buy와 sell의 경우 성공적으로 수행한 경우에 success가 초록색으로 출력된다. 이는 ₩033[0;30m을 사용하여 출력을 초록색으로 변경하고 ₩033[0m을 사용하여 초기화 하면 success부분

이 초록색으로 출력된다. Printf("₩033[0;30msuccess₩033[0m₩n")와 같이 작성하면 된다.

4. 성능 평가 결과 (Task 3)

- **강의자료 슬라이드의 내용 참고하여 작성 (측정 시점, 출력 결과 값 캡처 포함)**

시간의 측정은 multiclient에서 반복문이 실행되기 전부터 모든 자식 프로세스들을 종료하고 끝나기 까지의 시간을 측정했다. 시간의 측정은 두가지 방식으로 진행할 예정이다. 한가지는 모든 명령어를 랜덤하게 받는 경우와 하나는 show,buy,sell중 한가지 명령어만을 받는 경우를 선택하여 CLIENT_NUM에 변화를 주면서 elapsedtime과 throughput을 측정할 것이다.

워크로드 별 비교

Show

```
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 5
elapsed time: 0.010293 seconds
throughput : 4857.670261 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 10
elapsed time: 0.017491 seconds
throughput : 5717.226002 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 20
elapsed time: 0.030364 seconds
throughput : 6586.747464 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 50
elapsed time: 0.064778 seconds
throughput : 7718.669919 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 100
elapsed time: 0.118775 seconds
throughput : 8419.280152 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 200
elapsed time: 1.076983 seconds
throughput : 1857.039526 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 300
elapsed time: 1.146179 seconds
throughput : 2617.392222 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 400
elapsed time: 1.187359 seconds
throughput : 3368.821056 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 500
elapsed time: 1.305971 seconds
throughput : 3828.568935 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$
```

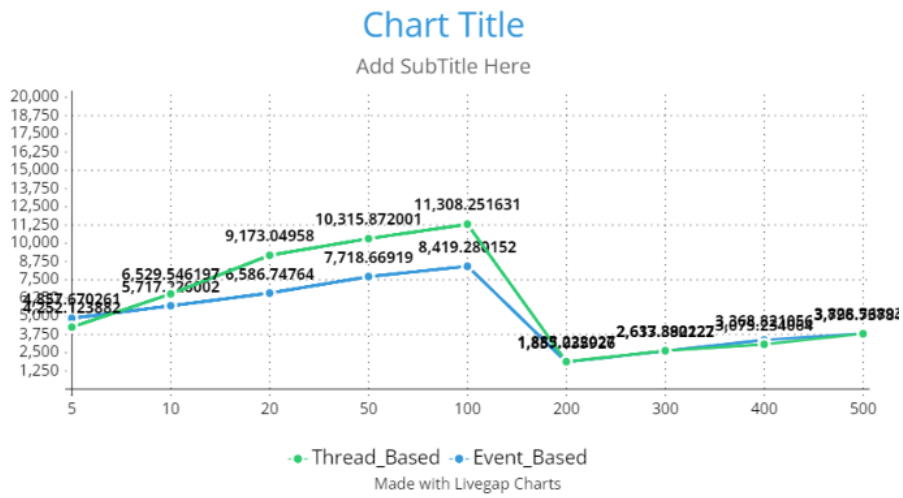
```
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 5
elapsed time: 0.011758 seconds
throughput : 4252.423882 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 10
elapsed time: 0.015315 seconds
throughput : 6529.546197 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 20
elapsed time: 0.021803 seconds
throughput : 9173.049580 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 50
elapsed time: 0.048469 seconds
throughput : 10315.872001 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 100
elapsed time: 0.088431 seconds
throughput : 11308.251631 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 200
^[[Aelapsed time: 1.060883 seconds
throughput : 1885.222027 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 300
elapsed time: 1.139004 seconds
throughput : 2633.880127 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 400
elapsed time: 1.300714 seconds
throughput : 3075.234064 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 500
elapsed time: 1.316920 seconds
throughput : 3796.737843 requests per second
```

위에서 사진에서 첫번째가 event기반 서버이고 아래가 thread기반 서버이다. 클

라이언트 숫자에 변화를 주면서 시간이랑 시간에 따른 처리량을 비교하였다.

아래는 비교하기 쉽게 그래프로 나타냈다.

아래는 클라이언트 수에 따른 출력량을 비교한 그래프이다.



.

Show 명령어만을 수행하는 경우에 적은 수의 클라이언트 구간에서는 thread기반이 전반적으로 좋은 효율을 보여주었다. 200을 넘어가는 많은 수의 클라이언트를 수행하는 경우부터는 두 서버가 유사한 효율을 보여주었다.

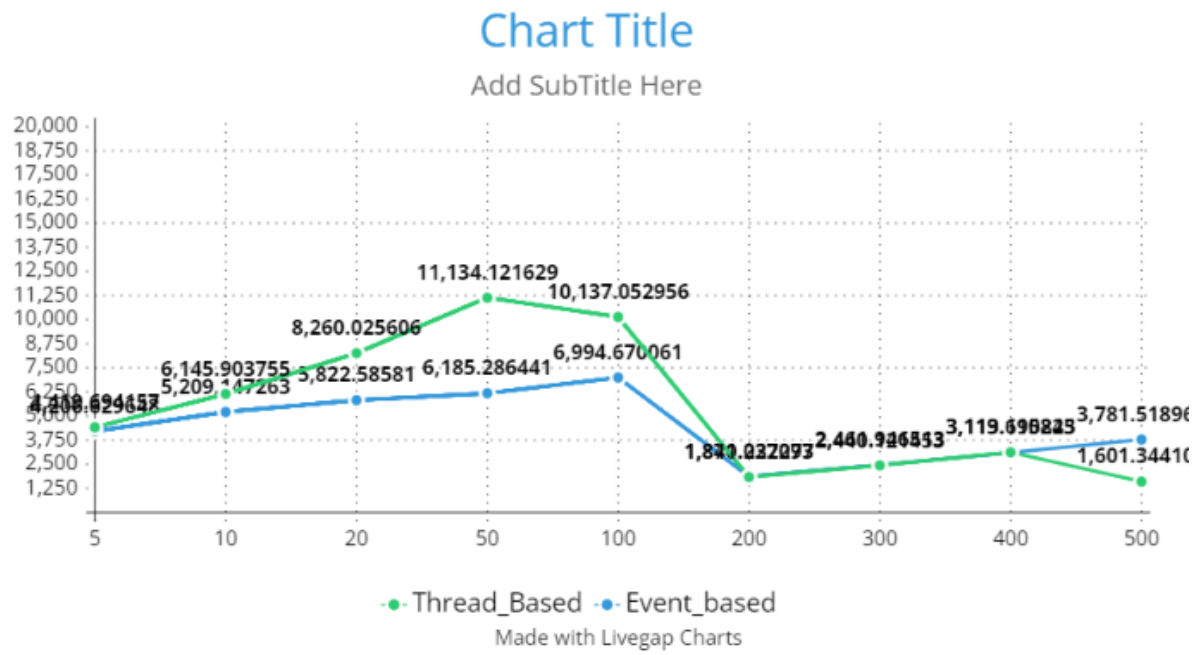
이론적으로는 스레드 기반이 더 높은 효율을 갖는 것이 맞지만 스레드 기반에서 critical section을 얼마나 효율적으로 보호하는지에 따라 효율성의 차이가 난다는 것을 알 수 있었다. P와 V함수를 실행하는 간격에 대기하고 있는 동안에 발생하는 시간에서 효율성의 차이가 발생한 것으로 보인다.

Buy

```
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 5
elapsed time: 0.011961 seconds
throughput : 4180.252487 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 10
elapsed time: 0.017003 seconds
throughput : 5881.315062 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 20
elapsed time: 0.032146 seconds
throughput : 6221.613887 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 50
elapsed time: 0.064265 seconds
throughput : 7780.284758 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 100
elapsed time: 0.112275 seconds
throughput : 8906.702293 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 200
elapsed time: 1.067567 seconds
throughput : 1873.418718 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 300
elapsed time: 1.154807 seconds
throughput : 2597.836695 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 400
elapsed time: 1.194544 seconds
throughput : 3348.558111 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 500
elapsed time: 1.341159 seconds
throughput : 3728.118739 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$
```

```
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 5
elapsed time: 0.011653 seconds
throughput : 4290.740582 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 10
elapsed time: 0.016582 seconds
throughput : 6030.635629 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 20
elapsed time: 0.060524 seconds
throughput : 3304.474258 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 50
elapsed time: 0.065455 seconds
throughput : 7638.835841 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 100
elapsed time: 0.112212 seconds
throughput : 8911.702848 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 200
elapsed time: 1.078266 seconds
throughput : 1854.829884 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 300
elapsed time: 1.150732 seconds
throughput : 2607.036217 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 400
elapsed time: 1.179971 seconds
throughput : 3389.913820 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 500
elapsed time: 1.319536 seconds
throughput : 3789.210753 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$
```

위의 경우와 마찬가지로 클라이언트 수에 따른 출력량과 경과 시간을 출력하였고 아래는 그래프 형태로 나타냈다.



인원이 적은 경우에는 show와 유사한 형태를 보이지만 인원이 많은 경우에서 thread가 적은 효율을 갖는것으로 보인다. 500에 도달하는 경우에는 절반까지로 떨어진다. 스레드 기반이 search를 하고 데이터를 수정하는 과정에서 P와 V함수에 의해 대기하는 시간이 길어지는 것이 이벤트 서버보다 낮은 효율을 갖는 원인이라고 보여진다. 더 많은 인원에서 thread가 낮은 효율을 갖는 것으로 보아 랜덤하게 출력하는 경우 더 큰 영향을 갖을 것으로 예산된다.

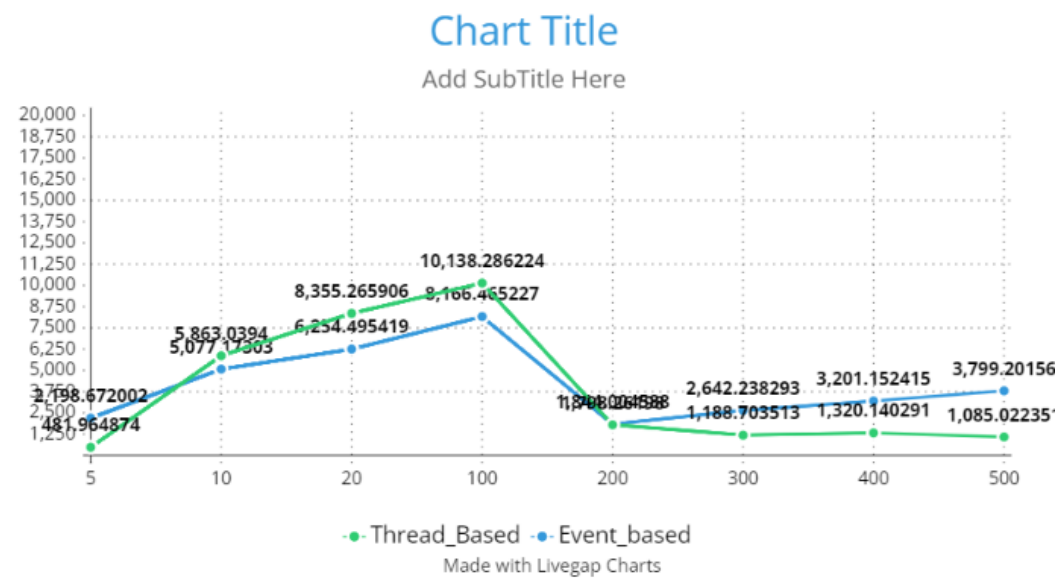
전체 랜덤기능 비교

```
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 5
elapsed time: 0.010958 seconds
throughput : 4562.876437 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 10
elapsed time: 0.017005 seconds
throughput : 5880.623346 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 20
elapsed time: 0.031080 seconds
throughput : 6435.006435 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 50
elapsed time: 0.057139 seconds
throughput : 8750.590665 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 100
elapsed time: 0.115418 seconds
throughput : 8664.159836 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 200
elapsed time: 1.072521 seconds
throughput : 1864.765352 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 300
elapsed time: 1.148527 seconds
throughput : 2612.041336 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 400
elapsed time: 1.191347 seconds
throughput : 3357.544024 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$ ./multiclient 172.30.10.11 60018 500
elapsed time: 1.318480 seconds
throughput : 3792.245616 requests per second
cse20192136@cspro9:~/system_programming/prj3/task1$
```

```
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 5
elapsed time: 0.010405 seconds
throughput : 4805.382028 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 10
elapsed time: 0.015060 seconds
throughput : 6640.106242 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 20
elapsed time: 0.025541 seconds
throughput : 7830.546964 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 50
elapsed time: 0.049867 seconds
throughput : 10026.670945 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 100
elapsed time: 0.115585 seconds
throughput : 8651.641649 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 200
elapsed time: 1.071637 seconds
throughput : 1866.303608 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 300
elapsed time: 1.122263 seconds
throughput : 2673.170193 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 400
elapsed time: 1.280056 seconds
throughput : 3124.863287 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$ ./multiclient 172.30.10.11 60018 500
elapsed time: 1.314904 seconds
throughput : 3802.558970 requests per second
cse20192136@cspro9:~/system_programming/prj3/task2$
```

마찬가지로 클라이언트 숫자에 변화를 주면서 시간이랑 시간에 따른 처리량을 비교하였다.

아래는 클라이언트 수에 따른 처리량을 그래프로 표현하였다.



Show,buy,sell 모든 기능들을 랜덤하게 수행하는 경우 두 경우 모두 100에서 200으로 넘어가는 시점에 효율의 하락이 있었다. 적은 클라이언트 수에서는 스레드 기반이 200을 넘는 기점으로 이벤트 기반이 더 높은 효율성을 갖는 것으로 나타났다.

Buy와 show 모두 적은 구간에서 더 나은 성능을 보여줬기에 적은 구간에서는 두 경우와 동일하게 thread기반 서버가 더 조은 효율을 보여주었으나, 클라이언트가 많은 구간에서는 buy의 경우 thread가 더 낮은 효율을 보여줬기에 더 많은 클라

이언트가 랜덤하게 출력을 하는 경우에도 더 큰 영향을 주어 클라이언트가 많은 경우에 Thread기반의 서버가 낮은 효율을 갖는 것으로 보인다.

이를 통해 이번 프로젝트에서 구현한 서버에서는 적은 수에서는 스레드 기반 서버가 많은 경우에서는 이벤트 기반 서버가 효율적임을 알 수 있었다.