Multicore Programming Project 2

담당 교수 : 최재승 교수님

이름 : 이승연

학번 : 20211569

1. 개발 목표

여러 client들의 동시 접속 및 서비스를 위한 Concurrent stock sever을 구축하는 것이 이번 프로젝트의 목표이다. 주식 서버는 주식 정보를 저장하고 있고 여러 client들과 통신하여, 주식 정보 List, 판매, 구매의 동작을 수행한다. 각 주식 클라이언트는 server에 주식 사기, 팔기, 재고 조회 등의 요청을 한다.

Event-driven Approach, Thread-based Approach 두 가지 방식으로 각 Concurrent stock server를 구축했다. 효율적인 데이터 관리를 위해 Binary Tree를 이용해 각 주식 종목의 정보를 저장하는 방법을 사용했다.

2. 개발 범위 및 내용

A. 개발 범위

1. Task 1: Event-driven Approach

   - 각 client는 서버에 연결하기 위한 client socket을 생성하고 각 server은 Select 함수를 통해 이벤트를 감지하고 새로운 client 연결을 받는다.

   - server가 시작되면 stock.txt 파일에서 주식 정보를 읽어 메모리에 load하고 Binary Tree를 구현하여 주식 정보를 저장한다. 이후 buy, sell 명령어의 구현은 이 Binary Tree로 처리한다.

   - 한 줄에 대한 명령어 처리가 끝날 때마다 Binary Tree에 업데이트 된 변경사항을 stock.txt에 다시 저장한다.

2. Task 2: Thread-based Approach

   - server가 실행되면 여러 개의 worker thread를 생성해 놓는다. 이후 client의 연결 요청을 worker thread에 할당하여 Concurrent하게 처리한다.

   - client들의 connfd는 구조체인 shared buffer로 관리된다. master thread가 fd들 buffer에 넣으면 worker thread가 그 fd를 buffer에서 가져와 fd에 해당하는 client에 대해 서비스를 실행시킨다.

   - server가 시작되면 stock.txt 파일에서 주식 정보를 읽어 메모리에 load하고 Binary Tree를 구현하여 주식 정보를 저장한다. 이후 buy, sell 명령어의 구현은 이 Binary Tree로 처리한다.

- 한 줄에 대한 명령어 처리가 끝날 때마다 Binary Tree에 업데이트 된 변경사항을 stock.txt에 다시 저장한다.

3. Task 3: Performance Evaluation

   - <sys/time.h>라이브러리의 gettimeofday 함수를 사용하여 두 가지 동작 방식(Event-driven, thread)의 elapse time을 분석한다.

   - 확장성 분석을 수행한다. 즉, 각 방법에 대한 Client 개수 변화에 따른 동시 처리율 변화를 분석한다.

   - 워크로드에 따른 분석을 수행한다.

     1) 모든 client가 buy 또는 sell을 요청하는 경우

     2) 모든 client가 show만 요청하는 경우

     3) Client가 buy, sell, show를 랜덤하게 섞어서 요청하는 경우

B. 개발 내용

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

  ✓ Multi-client 요청에 따른 I/O Multiplexing 설명

  Server는 하나의 listenfd와 여러 개의 connfd를 가진다. Select 함수를 사용해 fd들를 모니터링하고 pending input을 지정해준다. 해당 fd 중 pending input이 있는 fd에 대해 add_client 함수에서 pool 구조체에서 빈 slot을 찾아 connfd를 추가한다. 이렇게 할당된 connfd는 client와 server의 연결을 저장한다. 이후 check_client 함수에서 fd에 들어온 요청을 처리한다. buy, show, sell, exit 이 4가지의 요청을 구현했다.

  ✓ epoll과의 차이점 서술

  select은 오래 전에 개발된 multiplexing 기법으로 동시에 접속할 수 있는 최대 인원이 epoll에 비해 적다. 또한 select 호출 시 관찰 대상 fd에 대한 정보들을 매번 운영체제로 전달해야 한다는 번거로움이 있다. 반면 epoll은 동시 접속자수의 한도가 높으며 관찰 대상 fd에 대한 정보를 매번 전달하지 않아도 된다. fd의 저장소를 운영체제 수준에서 관리하기 때문이다.

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

  ✓ Master Thread의 Connection 관리

  Pthread_create함수를 통해 설정한 worker thread 개수(NTHREADS)만큼 worker threads를 생성한다. server가 client의 연결 요청을 받으면 Master Thread는 Accept함수를 이용해 connfd를 성공적으로 반환하고 sbuf_insert함수를 이용해 shared buffer 구조체인 sbuf에 insert하여 관리한다.

  ✓ Worker Thread Pool 관리하는 부분에 대해 서술

  client들의 connfd는 구조체인 shared buffer로 관리된다. master thread가 fd들 buffer에 넣으면 worker thread가 그 fd를 buffer에서 가져와 fd에 해당하는 client에 대해 서비스를 실행시킨다.

  현재 client들과 연결된 thread에서는 Pthread_detach 함수를 호출해 현재 thread를 분리하여 후에 thread 종료 시 자동으로 reapping되도록 한다. 이후 while문으로 sbuf_remove 함수를 반복해서 호출하며 데이터를 읽고 처리한 후에 buffer에서 connfd값을 제거한다.

- **Task3 (Performance Evaluation)**

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

  <sys/time.h>라이브러리의 gettimeofday 함수를 사용하여 두 가지 동작 방식(Event-driven, thread)의 elapse time을 측정한다. multiclient의 main 함수의 while loop 직전 시점에서 start, 모든 client process가 종료된 시점에서 end로 시간을 측정할 것이다.

  server가 각 client들에 대해 얼마나 동시에 처리를 잘 하는지 분석하기 위해 동시처리율 = (총 실행 시간) / (client 개수 * client당 command 수) 으로 구했다. 동시처리율을 구해 1초당 몇 개의 명령어를 처리하는지 구할 것이다.

  ✓ Configuration 변화에 따른 예상 결과 서술

  결과에 영향을 미칠 것이라 예상되는 변수들은 client개수, client 당 command 개수, 주식 개수이다. 다른 조건은 같다는 가정 하에 client 개수가 많아질수록, client 당 command개수가 많아질수록, 주식 개수가 많아질수록 실행 시간이 늘어날 것

이라고 예상된다. 또한 Ecent-based Approach는 실제 Multi-core를 사용하지 않는 반면에 Thread-based Approach는 실제로 여러 Threads를 사용해 context switching 하는 방식으로 Multi-core의 효과를 내기 때문에 Thread-based Approach의 동시처리율이 더 높을 것이라고 예상된다.

C. 개발 방법

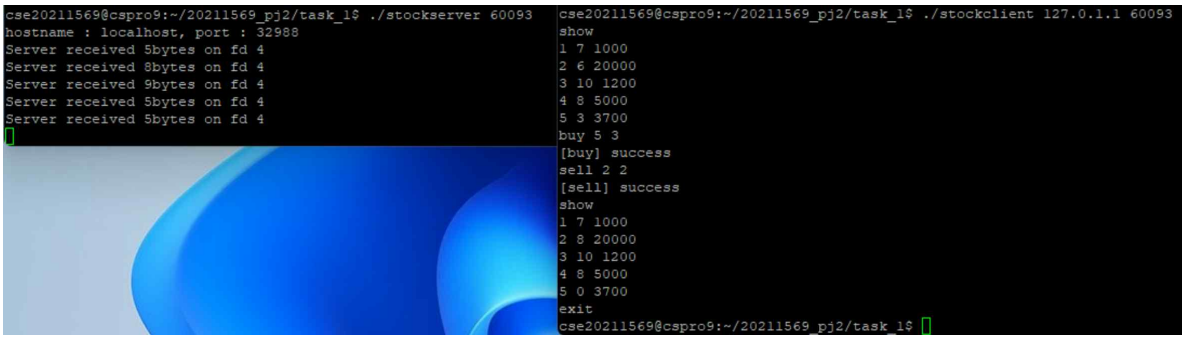
- ✓ **Task1 (Event-drived Approach)**
- ✓ 주식 정보를 저장하기 위한 item 구조체를 선언하였다. 주식의 정보(ID, left_stock, price)와 readcnt, 그리고 세마포어인 mutex를 가진다. 이 주식 정보들은 Binary Tree 형태로 저장된다. node 구조체로 구현했다. node는 item 구조체인 data, node 구조체인 left, right pointer를 가진다. main 함수가 실행되면 stock.txt 파일을 열어 주식정보를 한 줄 한 줄 읽어와 tree에 저장한다. 이 때 tree_insert 함수를 이용했다.
- ✓ main함수에서 init_pool을 통해 pool을 초기화 한 후 while문에 들어가 Select 함수를 사용해 fd들를 모니터링하고 pending input을 지정해준다. 해당 fd 중 pending input이 있는 fd에 대해 add_client 함수에서 pool 구조체에서 빈 slot을 찾아 connfd를 추가한다. 이후 check_client를 통해 주식서버 서비스를 진행한다. check_client에서는 크게 4가지 기능을 수행한다. buy, sell, show, exit이다.
- ✓ buy, sell 요청은 client가 입력한 정보를 바탕으로 주식 정보를 변경한다. buy에 관련된 함수는 search_node, buy_tree이다. search_node 함수로 입력받은 ID가 가리키는 노드를 찾아서 리턴한다. 이후 buy_tree 함수로 해당 노드에 대한 left_stock 값을 감소시킨다. 이 때 남아있는 left_stock값이 입력받은 stock값보다 작다면 에러 메세지를 출력하도록 핸들링 했다. 마찬가지로 sell에 관련된 함수는 search_node, sell_tree이다. search_node 함수로 입력받은 ID가 가리키는 노드를 찾아서 리턴하고 sell_tree 함수로 해당 노드에 대한 left_stock값을 증가시킨다.
- ✓ show 요청은 현재 tree에서 inorder 순서로 주식 정보들을 출력한다. exit 요청은 connfd를 Close하여 해당 client를 닫아준다.
- ✓ **Task2 (Thread-drived Approach)**
- ✓ Binary Tree 구현방법은 Task1과 유사하다. 다만 item 구조체에 세마포어인 writer

변수가 추가된 점이 다르다. 한 client가 buy혹은 sell 명령어를 사용해 노드를 업데이트 하는 도중에 다른 client가 show 명령어를 수행한다면 reader-writer 문제가 발생해 잘못된 동작을 수행할 수 있기 때문이다.

✓ buy_tree, sell_tree, show_tree 함수에 모두 P, V함수를 사용해 한 task에 writer가 한 명만 존재할 수 있도록 처리해주었다.

3. 구현 결과

- Task1 (Event-driven Approach)

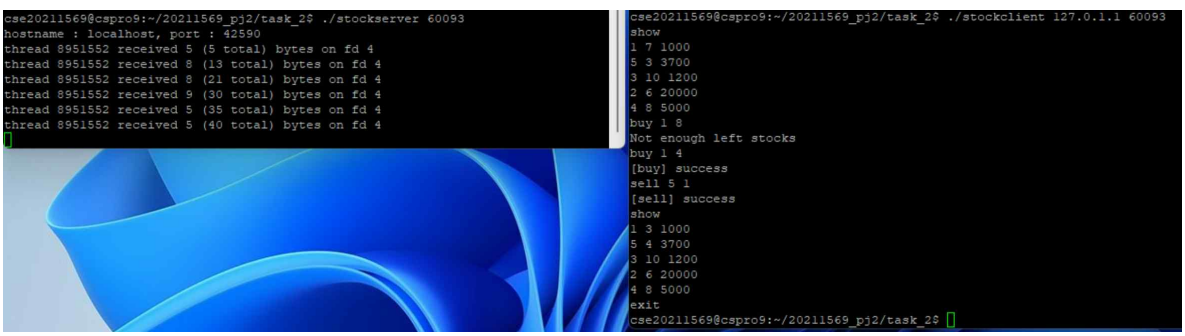


왼쪽이 server, 오른쪽이 client이다. server와 client를 연결한 후, show명령어로 현재 tree에 존재하는 모든 주식 정보를 출력한다.

이후 buy 5 3 명령어를 입력해 ID가 5인 주식을 구매하고 sell 2 2 명령어를 입력해 ID가 2인 주식을 판다. 다시 show 명령어를 입력해 출력해보면 의도한 대로 수량이 변경된 것을 볼 수 있다.

exit 명령어를 이용해 주식장을 나온다.

- Task2 (Thread-driven Approach)



왼쪽이 server, 오른쪽이 client이다. task1과 같이 server와 client를 연결한 후, show 명령어로 현재 tree에 존재하는 모든 주식 정보를 출력한다.

이후 buy 1 8 명령어를 입력해 남아있는 주식이 충분하지 않을 때 에러 메세지를 출력하는 것을 확인하고 buy 1 4 명령어를 입력해 ID가 1인 주식 4개를 구매한다. sell 5 1 명령어를 입력해 ID가 5인 주식 1개를 팔고 마지막으로 show 명령어를 입력해 출력해보면 의도한 대로 수량이 변경된 것을 볼 수 있다.

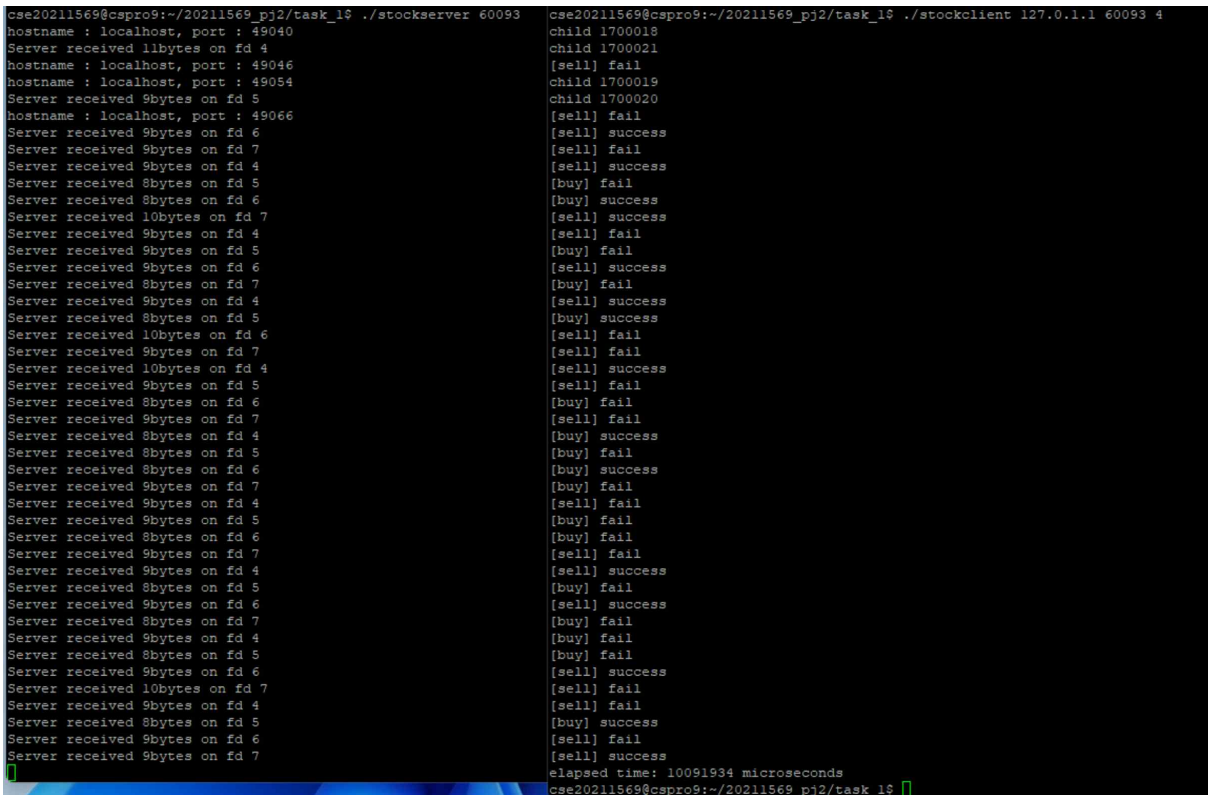
exit 명령어를 이용해 주식장을 나온다.

4. 성능 평가 결과 (Task 3)

(1) 확장성 분석

- 측정 시점 : multiclient의 main 함수의 while loop 직전 시점에서 start, 모든 client process가 종료된 시점에서 end로 시간을 측정할 것이다. 총 시간을 command 수로 나누어 시간당 client 처리 요청 개수를 분석한다.

- 출력 결과 캡처 :

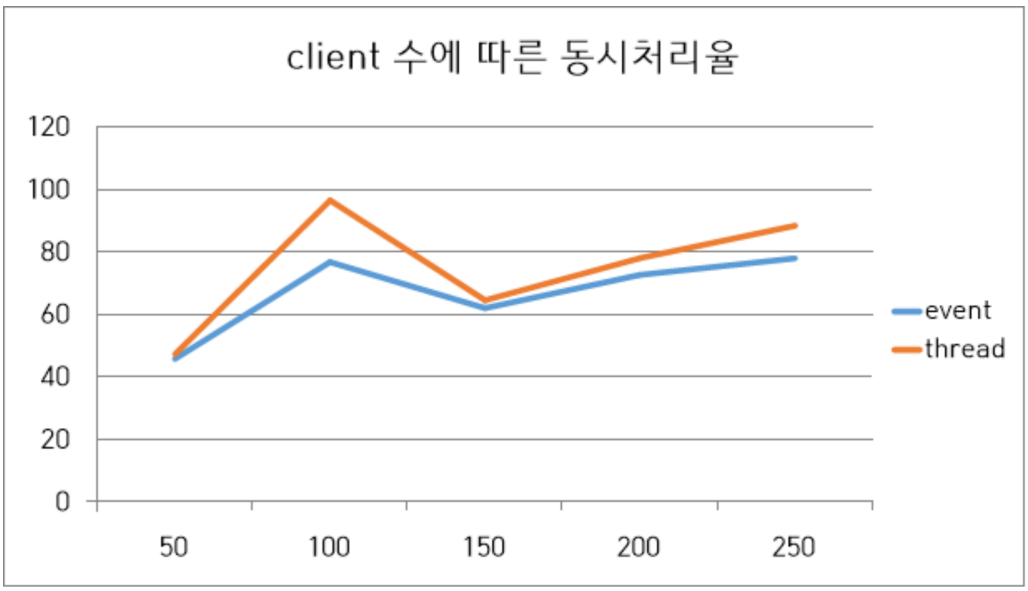


위의 그림은 buy 또는 sell만 요청하는 경우이고, client수는 4, 주식 개수는 5, client 당 command 수는 10으로 총 40개의 command를 수행한 것을 알 수 있다. 이 경우에 걸린 시간은 10.091934초이다. (한 화면에 호출부터 종료까지 보여주기 위해 이 사진을 넣었다.)

- 표, 그래프 분석

client 수	총 command 수	총 걸린 시간		총 걸린 시간(1초당)	
		event	thread	event	thread
50	500	10.98211	10.57189	45.5285915	47.29523292
100	1000	13.069454	10.362323	76.5142905	96.50345777
150	1500	24.282843	23.237186	61.77200915	64.55170605
200	2000	27.613112	25.631118	72.42935892	78.03015069
250	2500	32.16634	28.280487	77.72099654	88.400175

확장성 분석에서는 show, buy, sell은 랜덤으로, client 당 command 수는 10으로 고정, stock 개수는 5로 고정하고 client 수만 변수로 두어 총 command수를 달리 해가며 실행시간을 측정했다.



가로축은 client수, 세로축은 동시처리율이다. 그래프를 보면 thread가 event-driven에 비해 항상 동시처리율이 높다는 것을 알 수 있다. multi-core효과를 내는 thread의 특성 때문이다.

(2) 워크로드 분석

- 측정 시점 : multiclient의 main 함수의 while loop 직전 시점에서 start, 모든 client process가 종료된 시점에서 end로 시간을 측정할 것이다. 총 시간을 command 수로 나누어 시간당 client 처리 요청 개수를 분석한다.

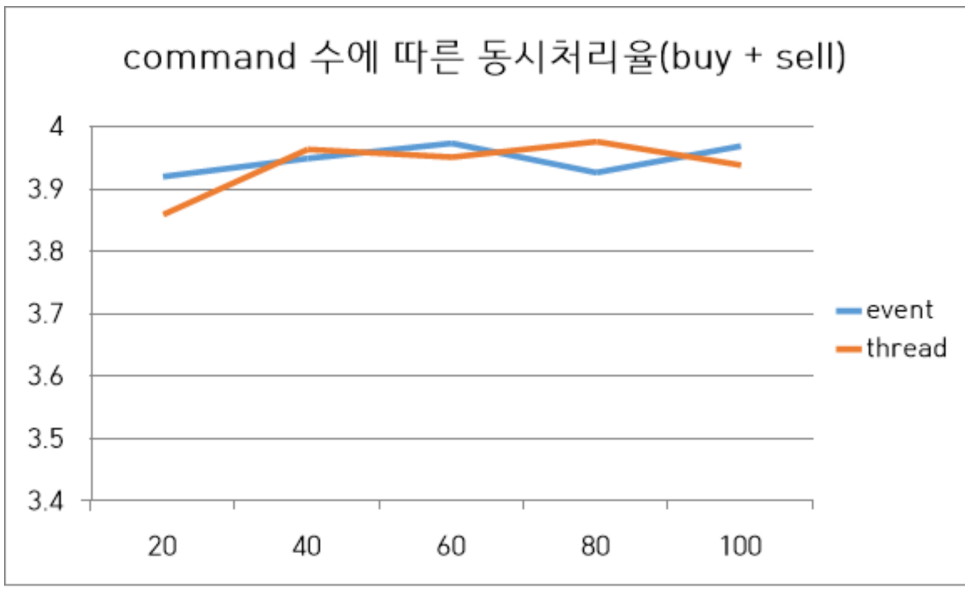
- 모든 조건에 대해 client 수를 4개로, stock 개수를 5개로 고정하고 측정했다.

1) 모든 client가 buy 또는 sell을 요청하는 경우

- 표, 그래프 분석

client 수	command 수	총 command 수	총 걸린 시간		총 걸린 시간(1초당)	
			event	thread	event	thread
4	5	20	5.102305	5.182152	3.919797033	3.859400496
4	10	40	10.129623	10.091212	3.948814285	3.963844977
4	15	60	15.10133	15.184264	3.97315998	3.951459221
4	20	80	20.375871	20.119328	3.926212529	3.976275947
4	25	100	25.197154	25.389735	3.96870218	3.938599595

buy 또는 sell 명령어만 요청하는 경우 command수를 5, 10, 15, 20, 25로 늘려가며 실행시간을 측정했고 총 command수를 실행 시간으로 나누어 1초당 동시처리율을 구했다.

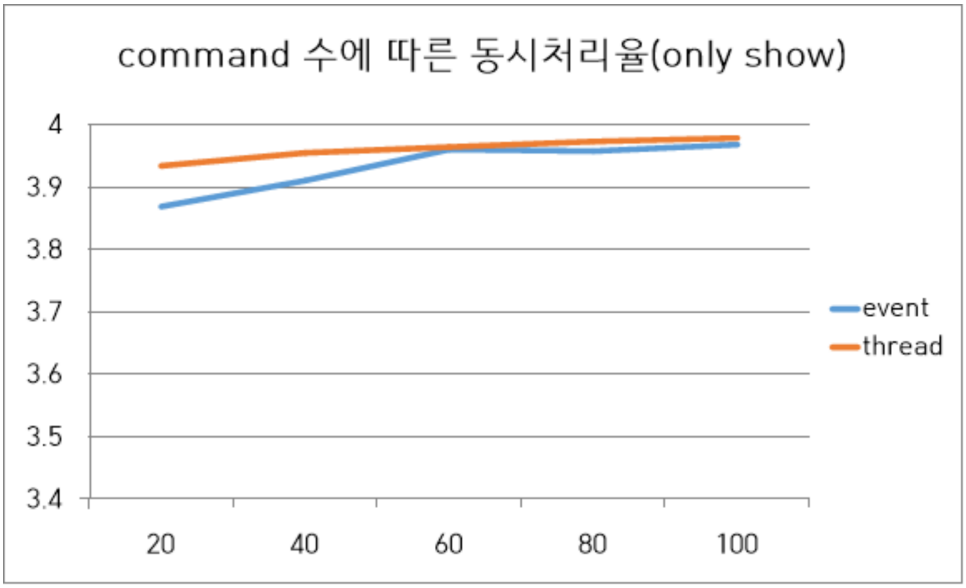


2) 모든 client가 show만 요청하는 경우

- 표, 그래프 분석

client 수	command 수	총 command 수	총 걸린 시간		총 걸린 시간(1초당)	
			event	thread	event	thread
4	5	20	5.169649	5.283926	3.868734608	3.933967568
4	10	40	10.229035	10.114356	3.910437299	3.954774778
4	15	60	15.150503	15.135905	3.960264554	3.964084077
4	20	80	20.21446	20.135719	3.957563051	3.973039155
4	25	100	25.201179	25.134028	3.968068319	3.978669873

show 명령어만 요청하는 경우 command수를 5, 10, 15, 20, 25로 늘려가며 실행시간을 측정했고 총 command수를 실행 시간으로 나누어 1초당 동시처리율을 구했다.



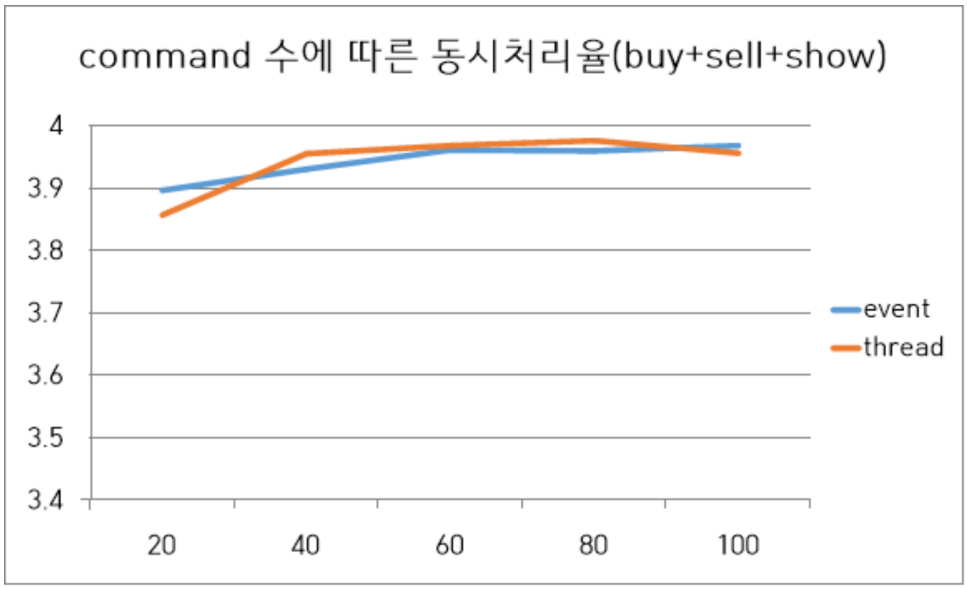
그래프의 가로축은 총 command 개수이고 세로축은 동시 처리율이다. show만 요청하는 경우가 buy+sell을 요청하는 것보다 실행시간이 오래 걸리는 것을 알 수 있다.

3) Client가 buy, sell, show를 랜덤하게 섞어서 요청하는 경우

- 표, 그래프 분석

client 수	command 수	총 command 수	총 걸린 시간		총 걸린 시간(1초당)	
			event	thread	event	thread
4	5	20	5.132468	5.184997	3.896760779	3.857282849
4	10	40	10.176127	10.111872	3.930768553	3.955746275
4	15	60	15.145671	15.118614	3.961528017	3.968617758
4	20	80	20.201977	20.117492	3.960008468	3.976638837
4	25	100	25.19784	25.274981	3.968594133	3.956481708

buy, sell, show 명령어 모두 요청하는 경우 command수를 5, 10, 15, 20, 25로 늘려가며 실행시간을 측정했고 총 command수를 실행 시간으로 나누어 1초당 동시처리율을 구했다.



그래프의 가로축은 총 command 개수이고 세로축은 동시 처리율이다. 모든 명령어를 섞어서 요청하는 경우 대체로 show와 buy+sell 동시처리율 사이의 동시처리율을 나타내는 것을 볼 수 있다.