System Programming Project 2

담당 교수 : 김영재 교수님

이름 : 강수경

학번 : 20190340

1. 개발 목표

- 여러 client 들이 동시에 접속하는 stock server 을 만드는 것을 목표로 한다. 이번 프로젝트는 Event-based 방식과 Thread-based 방식 두가지로 stock server를 개발하였다. Client 는 stock server에 show, buy, sell, exit 요청을 전송한다. Concurrent 프로그래밍을 통해 server를 구현한다. Stock.txt 에서 관리하는 주식 content를 업데이트 하고 남은 주식보다 많은 주식을 산다고 요구하면 부족하다는 에러 메세지가 뜨도록 구현한다.

2. 개발 범위 및 내용

A. 개발 범위

1. Task 1: Event-driven Approach

   Client 의 order 에 따라 그에 해당하는 동작이 작동 및 해당 결과가 stock.txt 에 저장

2. Task 2: Thread-based Approach

   Client 의 order 에 따라 그에 해당하는 동작이 작동 및 해당 결과가 stock.txt 에 저장

3. Task 3: Performance Evaluation

   Event 기반 server 와 thread 기반 server 의 성능/ 수행속도 를 비교하여 측정 및 확인.

B. 개발 내용

- **Task1 (Event-driven Approach with select())**
  - ✓ Multi-client 이 연결을 요청할 경우 thread가 여러 개 생성되어 여러 client 들과 소통한다.
  - ✓ epoll을 사용하면 매번 fd를 체크해야해서 더욱 느리게 수행될 수 있다.
- **Task2 (Thread-based Approach with pthread)**
  - ✓ Master Thread는 각 connfd 를 select 함수를 통해 확인하며 connection들을 관리한다.
  - ✓ Worker Thread Pool은 connfd 의 정보를 저장한다

- **Task3 (Performance Evaluation)**

  ✓ Server 별로 작업을 처리하는 속도를 측정하기 위해 전체 작업을 완료하는데 걸리는 시간을 측정한다.

  ✓ Thread 는 memory 영역 자체를 copy 하는 것이기에 event based server 에 비해 overhead 가 발생할 가능성이 크다. 하지만 요즘 대부분의 컴퓨터 CPU 는 멀티 코어를 지원하기 때문에 코어가 여러 개일 경우 thread 기반의 서버는 나누어 진행되어 더 빠르게 진행될 것이다. Client 의 수를 조정해주고 client 가 내리는 order 수를 지정 해주어 비교해 볼 것이다.

  ✓ 1)모든 client가 buy 또는 sell을 요청하는 경우

  ✓ 2) 모든 client가 show만 요청하는 경우

  ✓ 3) Client가 buy, sell, show 을 모두 섞어서 요청하는 경우

  로 나누어 각 결과를 비교해본다.

C. 개발 방법

- **Task1 (Event-driven Approach with select()), Task2 (Thread-based Approach with pthread) 공통 개발 사항**

  void check_client(pool* p); , void echo_thread(int connfd);

  Rio_readlineb(&rio, buf, MAXLINE)를 불러들여 MAXLINE 으로 내용을 읽어주고 optioncheck(char* buf, int n); 을 호출하여 client 에 어떤 작업을 진행중인지 표시 할 수 있도록 하였다. 또한 server 가 몇 byte 를 받았는지 int cnt 로 세어 server 에 print 할 수 있도록 하였다. 또한 더이상의 요청이 발생하지 않을 때는 Close(connfd)로 작업을 마무리 해주었다.

  int optioncheck(char* buf, int n);

  buy, sell, show 중 client 가 어떤 request 를 했는지 check 하고 이에 따른 결과를 출력한다. 또한 이를 buf 에 저장하여 void check_client(pool* p); 로 전달하여 client 가 화면에 print 할수 있도록 한다. Show 의 경우 계속 txt file update 를 할 수 있도록 하였다. 또한 더 이상 buy 할 수 없을 때 left stock 이 없다는 메시지가 나오도록 하였다.

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

  -int main(int argc, char** argv)

  -void init_pool(int listenfd, pool* p);

  -void add_client(int connfd, pool* p);

  기존 제공된 src 코드에 주식정보가 저장된 stock.txt을 읽어 올 수 있도록 추가 하였으며 읽은 정보를 tree에 저장하여 조작할 수 있도록 하였다. 또한 node 를 구성하는 구조체를 선언하여 클라이언트의 요구사항을 구조체형식으로 저장하여 올바르게 server 의 기능을 갖출 수 있도록 구현하였다. 강의자료 6장의 event based concurrent echo server 를 참고하여 이를 토대로 구현하였다.

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

  -int main(int argc, char** argv)

  -void sbuf_init(sbuf_t *sp, int n);

  -void sbuf_deinit(sbuf_t *sp);

  -void sbuf_insert(sbuf_t *sp, int item);

  -int sbuf_remove(sbuf_t *sp);

  -void *thread(void *vargp);

  -static void init_echo_cnt(void);

  기존 제공된 src 코드에 주식정보가 저장된 stock.txt을 읽어 올 수 있도록 추가 하였으며 읽은 정보를 tree에 저장하여 조작할 수 있도록 하였다. 강의자료 6장의 thread based concurrent echo server 를 참고하여 이를 토대로 구현하였다 client 의 연결 요청을 master thread 에서 받고 connectioco 을 관리하며 client 의 연결요청을 확인한다. 쓰레드의 개수와 sbuf slot의 개수가 너무 적으면 성능이 좋지 않아서 적당한 개수를 설정하는 것이 이슈였다. Client 수가 너무 많으면 error message 가 발생하는 문제가 발생하였다. 연결을 시도하려는데 Close 가 되버렸을 때 이 문제 발생하였다. 이를 해결하기위해 NTHREADS와 SBUFSIZE를 증가시켜주었다.

3. 구현 결과

위 사항을 모두 구현한 결과 client 의 request 인 buy. sell, show 모두를 처리하고 이 결과를 기존에 stock의 내용이 저장 되어있는 txt 파일에 update 해 저장하는 프로그램을 만들었다.

4. 성능 평가 결과 (Task 3)

-예상

1.CPU는 멀티코어를 지원하기 때문에 멀티코어를 사용하지 않는 event based 보다 멀티코어를 사용하는 thread based가 더 빠를 것이다. 또한 thread 는 완만하게 실행시간이 증가하는 것에 비해 event based 는 더 가파르게 실행 시간이 증가할 것이다.

2.show 명령만 실행할 경우 node 내용을 변경하지 않고 그대로 show 결과만 print 하기에 다른 명령을 혼합해서 실행할 때보다 빠를 것이다.

3. 모든 명령(buy, sell, show)을 사용할 때> 두가지 명령만 사용할 때> show 만 사용 할 때 순으로 실행 시간이 차이 날 것이다.

-성능 평가 실험

gettimeofday 함수를 이용하여 작동 시간을 측정한다.

```
gettimeofday(&end, 0);
   e_usec = ((end.tv_sec * 1000000) + end.tv_usec) - ((start.tv_sec * 1000000) + start.tv_usec);
   printf("elapsed time: %lu microseconds₩n", e_usec);
```

A. 1)모든 client가 buy 또는 sell을 요청하는 경우

Client 수를 조정하며 event based 와 thread based 를 비교.

(단위:microsecond)

-실험 모습과 elapsed time 캡쳐 화면

1)event based server

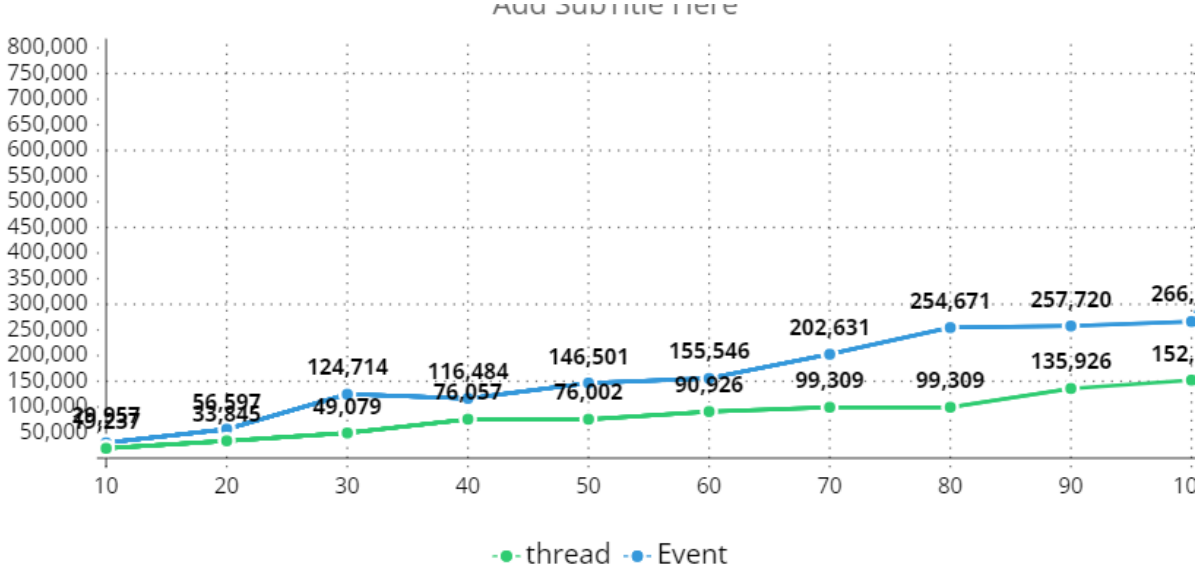
```
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 10
elapsed time: 29957 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 20
elapsed time: 56597 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 30
elapsed time: 124714 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 40
elapsed time: 116484 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 50
elapsed time: 146501 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 60
elapsed time: 155546 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 70
elapsed time: 202631 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 80
elapsed time: 254671 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 90
elapsed time: 257720 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 100
elapsed time: 266569 microseconds
```

2)thread based server

```
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 10
elapsed time: 19237 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 20
elapsed time: 33845 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 30
elapsed time: 49079 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 40
elapsed time: 76057 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 50
elapsed time: 76002 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 60
elapsed time: 90926 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 70
elapsed time: 99309 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 80
elapsed time: 99309 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 90
elapsed time: 135926 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 100
elapsed time: 152213 microseconds
```

-실험결과

Client수	10	20	30	40	50	60	70	80	90	100
Event	29957	56597	124714	116484	146501	155546	202631	254671	257720	266569
thread	19237	33845	49079	76057	76002	90926	99309	99309	135926	152213



B. 2) 모든 client가 show만 요청하는 경우

Client 수를 조정하며 event based 와 thread based 를 비교.

(단위:microsecond)

-실험 모습과 elapsed time 캡쳐 화면

1)event based server

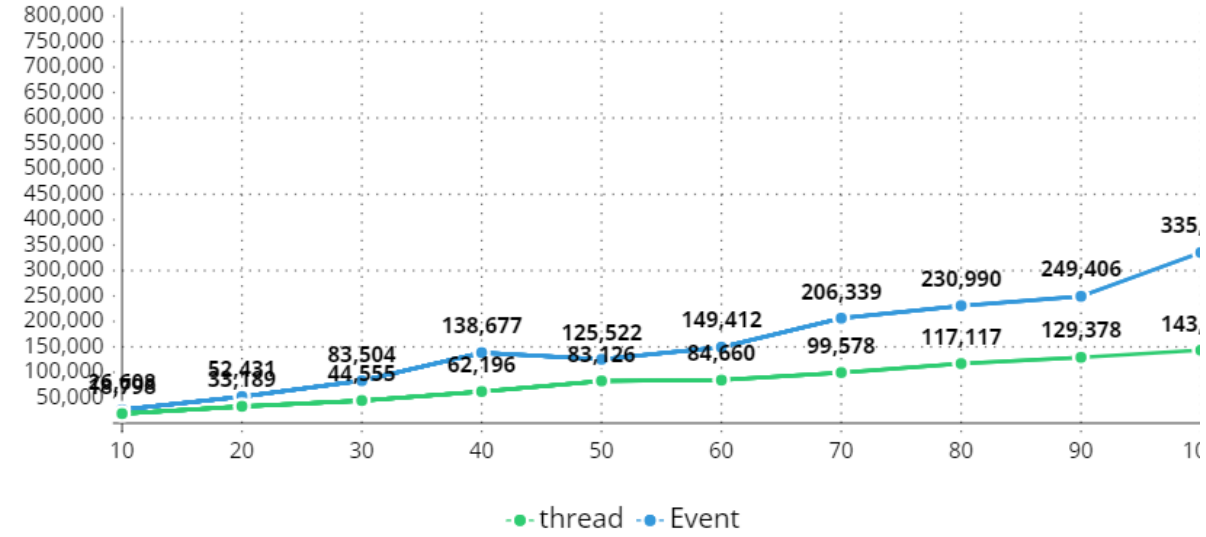
```
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 10
elapsed time: 26608 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 20
elapsed time: 52431 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 30
elapsed time: 83504 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 40
elapsed time: 138677 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 50
elapsed time: 125522 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 60
elapsed time: 149412 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 70
elapsed time: 206339 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 80
elapsed time: 230990 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 90
elapsed time: 249406 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 100
elapsed time: 335543 microseconds
```

2)thread based server

```
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 10
elapsed time: 18798 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 20
elapsed time: 33189 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 30
elapsed time: 44555 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 40
elapsed time: 62196 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 50
elapsed time: 83126 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 60
elapsed time: 84660 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 70
elapsed time: 99578 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 80
elapsed time: 117177 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 90
elapsed time: 129378 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 100
elapsed time: 143479 microseconds
```

-실험결과

Client수	10	20	30	40	50	60	70	80	90	100
Event	26608	52431	83504	138677	125522	149412	206339	230990	249406	335543
thread	18798	33189	44555	62196	83126	84660	99578	117117	129378	143479



C. 3) Client가 buy, sell, show 을 모두 섞어서 요청하는 경우

Client 수를 조정하며 event based 와 thread based 를 비교.

(단위:microsecond)

-실험 모습과 elapsed time 캡쳐 화면

1)event based server

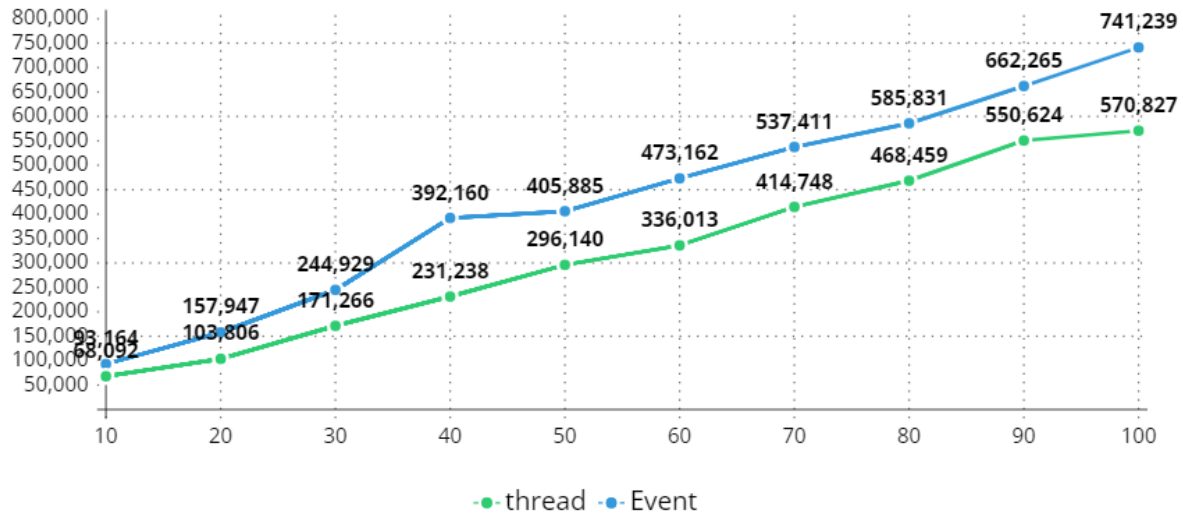
```
cse20190340@cspro1:~$ cd event
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 10
elapsed time: 93164 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 20
elapsed time: 157947 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 30
elapsed time: 244929 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 40
elapsed time: 392160 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 50
elapsed time: 405885 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 60
elapsed time: 473162 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 70
elapsed time: 537411 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 80
elapsed time: 585831 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 90
elapsed time: 662265 microseconds
cse20190340@cspro1:~/event$ ./multiclient 172.30.10.11 60133 100
elapsed time: 741239 microseconds
```

2)thread based server

```
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 10
elapsed time: 63092 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 20
elapsed time: 103806 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 30
elapsed time: 171266 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 40
elapsed time: 231238 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 50
elapsed time: 296140 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 60
elapsed time: 336013 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 70
elapsed time: 414748 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 80
elapsed time: 468459 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 90
elapsed time: 550624 microseconds
cse20190340@cspro1:~/thread$ ./multiclient 172.30.10.11 60133 100
elapsed time: 570827 microseconds
```

-실험결과

Client수	10	20	30	40	50	60	70	80	90	100
Event	93164	157947	244929	392160	405885	473162	537411	585831	662265	741239
thread	68092	103806	171266	231238	296140	336013	414748	468459	550624	570827



- 실험 결과 분석 및 예측과 비교

예상 1에서 예상한 것과 같이 A, B, C 실험 모두 동일하게 client 수가 증가함에 따라 event based server보다 thread based server 가 더 짧은 실행시간을 가짐과 완만한 증가폭을 가지는 것을 관찰할 수 있었다.

이를 통해 단일 thread 구조인 event based server 는 오버헤드는 적을 수 있으나 multi-core 환경에서는 multicore 의 혜택을 보지 못하고 thread based server 는 multicore 의 혜택을 보아 좀더 빠른 수행 시간과 적은 증가 폭을 가짐을 알 수 있었다.

예상 2 에서 생각 한 바와 결과는 다르게 나타났다. Buy, sell 두가지 명령(node 내부를 건들이는 명령)을 실행할 때와 단순 출력인 show 두 결과값의 차이를 구분할만한 큰 차이를 찾기 어려웠다. 하지만 변경하며 실험 했던 client 값 대부분에서 sell 과 buy 두 명령을 요청했던 것이 show 명령만 실행했을 때보다 실행시간이 오래 걸린 횟수가 많았다. (아래 표에서 실행시간이 더 오래걸린 자료만 하이라이트 하였다.)

buy, sell 두 명령만

Client수	10	20	30	40	50	60	70	80	90	100
Event	29957	56597	124714	116484	146501	155546	202631	254671	257720	266569
thread	19237	33845	49079	76057	76002	90926	99309	99309	135926	152213

Show 명령만

Client수	10	20	30	40	50	60	70	80	90	100
Event	26608	52431	83504	138677	125522	149412	206339	230990	249406	335543
thread	18798	33189	44555	62196	83126	84660	99578	117117	129378	143479

유의미한 차이가 발생하지 않은 것은 1개의 단순한 명령과 2개의 복잡한 명령어를 실행할 때의 차이는 간소할 것으로 생각되고 더 숫자를 확연히 차이가 나도록 한다면 유의미한 결과를 도출 할 것으로 예상해본다.

예상 3에서 Client 가 요구하는 명령의 수가 증가할수록 처리 속도가 증가할 것이라고 생각했다. 이도 1개 명령과 2개의 명령은 아주 약간의 차이가 났으나 3개 명령을 실행 했을 때는 확연히 1개 와 2개 명령을 실행 했을 때보다 오랜 시간이 소요 되었다.