Multicore Programming Project 2

담당 교수: 최재승

이름: 김상현

학번: 20181605

1. 개발 목표

서버에 저장된 주식 데이터를 기반으로 client들이 보내는 구매, 매도, 주식 데이터 나열, 주식 장 퇴장 request들을 서비스하는 주식 서버를 개발하는 것이 이번 프로젝트의 목표이다.

Client들을 서비스하는 서버를 구현하는 방식에는 크게 event-driven approach와 thread-based approach가 있다. 이 방식들을 구현한 후 둘의 성능을 비교하는 것으로 프로젝트가 마무리된다.

2. 개발 범위 및 내용

A. 개발 범위

1. Task 1: Event-driven Approach

   Select를 이용한 I/O multiplexing을 통해 여러 client들의 connection request를 동시에 처리할 수 있다. 하지만 새로운 I/O 이벤트 외에는 반응하지 않기 때문에 fine-grained locking이 불가능하다는 단점이 있다. 각 connection들을 통해 들어온 request에 대한 실행은 sequential하게 일어나기 때문에 순서에 따른 stock 정보에 대한 변화의 차이(race)는 있어도 corrupt 되는 일을 없을 것으로 보인다.

2. Task 2: Thread-based Approach

   Shared buffer와 threading을 이용하여 task 1과 동일한 기능을 수행한다. 다만 어떤 thread에서 service가 수행되는 중에 다른 service를 수행하고 있는 thread로 control flow가 진행될 수 있기 때문에 fine-grained locking이 필수적이다.

3. Task 3: Performance Evaluation

   지금까지 배운 내용에 따르면 thread-based approach가 event-driven approach보다 성능이 좋아야 한다. 이를 확인하기 위해 multiclient.c를 이용하여 많은 client들의 명령을 잘 수행하는지 동시 처리율과 실행 시간 등의 metric으로 두 implementation을 비교한다.

B. 개발 내용

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

connfd array를 통해 단일 프로세스로 실행되는 서버에서 활성화된 모든 connection들을 관리한다. 이 때 프로세스 안에서는 active한 connection들의 모

든 request에 대한 서비스를 수행하고 나서 select나 epoll을 활용하여 새로운 connection에 대한 connfd를 세팅한다. 구현의 용이함을 위해 pool이라는 구조체 안에서 select에 필요한 변수들을 관리한다. listenfd를 통해 새로운 connection request가 들어올 때마다 새로운 connfd가 설정되고 pool 내의 clientfd와 read set에 추가된다. 서버는 모든 clientfd들을 iterate하며 request를 처리한다.

epoll은 select 방식에서 사용되는 iteration에 따른 overhead를 줄이는데 사용된다. clientfd를 iterate하지 않고 OS에서 file descriptor들을 모두 관리한다. 하지만 모든 OS에서 제공되는 서비스는 아니기 때문에 portable하지 않다.

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

Master Thread에서 새로운 connection request를 받고 모든 connection에서 들어오는 request나 입력을 각자의 connection에 대응하는 sbuf를 통해 관리한다. 각 thread에서 sbuf를 돌며 client의 request를 수행한다.

Pthread_detach 함수를 사용하여 사전에 생선된 thread들의 수행이 종료될 경우 별도의 join 없이 시스템에 반환된다. 구현된 프로그램에서는 server 프로그램이 종료되기 전까지 thread가 돌아간다.

- **Task3 (Performance Evaluation)**

구현된 서버의 성능을 볼드체로 표시된 두개의 metric을 통해 확인할 수 있다.

$$\boldsymbol{W} = \frac{R \times N}{\boldsymbol{T}}$$

W: 동시 처리율

R: Client 당 request 개수

N: Client들의 개수

T: multiclient 프로그램 구동 시간(모든 client들의 request가 처리되기까지 걸린 시간)

동시 처리율을 사용하여 서버의 시간 당 client의 request 처리율을 확인할 수 있고 총 수행시간을 통해 처리율에 대응하는 처리시간을 확인할 수 있다.

정확한 측정과 편리한 결과 확인을 위해 server reply에 대한 출력과 usleep을 통한 delay를 mute하고 오로지 출력 결과로 동시 처리율과 총 수행 시간만을 출력하도록 코드를 수정했다.

Request 타입에 따른 서버의 성능을 확인하기 위해 각 implementation에 대해 3가지의 경우 (buy, sell, show), (buy, sell), (show)에 대한 request만을 처리하는 환경을 구성했다.

결과를 예측하자면 모든 경우에 대해 thread-based approach가 event-driven approach보다 빠를 것으로 예상한다. 다만 client의 수가 적다면 두 implementation 사이의 성능 차이가 드러나지 않을 것이다. 그리고 show request만을 수행하는 configuration이 동일한 approach에서 모든 경우 중 가장 느릴 것이다. 왜냐하면 show는 reader 타입으로 모든 노드에 대해 buffer에 write하고 이를 위해 각 노드에 reader locking을 하기 때문이다. 이와 반대로 buy와 sell은 수정되는 노드에 대한 reader-writer locking만 수행하면 된다.

C. 개발 방법

우선 주식의 정보들을 각 노드에 저장하고 있는 binary tree를 구성해야 한다. Binary tree는 stock ID에 따라 순서가 정해져 있다. 각 노드에서 왼쪽 branch로 내려가면 현재 노드보다 낮은 stock ID를 가지고 있는 주식 노드, 오른쪽 branch로 가면 현재 노드보다 높은 stock ID를 가지고 있는 주식 노드를 찾을 수 있게 설계한다.

Select를 구현하기 위해 pool이라는 구조체를 선언한다. 해당 구조체 안에는 최대 fd 번호, read set, ready set, nready, maxi, clientfd array, clientrio array가 들어있다. 최대 fd 번호는 Select 함수가 최대 fd 번호까지 iterate 할 수 있도록 설정된다. 그리고 ready set은 Select 함수가 read set을 새로운 정보로 덮어씌우지 않도록 일종의 copy를 제공한다. nready는 현재 연결된 connection들 중 request가 대기 중인 connection의 수를 보여준다. clientfd와 clientrio 배열은 각 connection들의 buffer를 가리키는 기능을 수행한다.

Client의 request를 수행하기 위해 buy, sell, show, exit 함수를 만든다. Buy와 sell 함수는 binary tree에서 client가 요구한 stock ID에 대응하는 노드를 찾아야 한다. 따라서 tree에서 주어진 stock ID에 대응하는 노드를 찾는 binary tree search 함수인 find 함수를 만든다.

show 함수는 binary tree에서 inorder print를 하는 것과 동일한 logic으로 설계한다. 다만 request를 받은 connection의 buffer에 write 할 수 있도록 buffer에 대한 포인터도 함수의 인자로 전달한다. 그리고 buffer에 inorder로 write할 수 있도록 offset도 계산한다.

Stock 정보를 저장하는 binary tree를 관리하고 수정하는 기능은 별도의 stock_data_structure 파일에 추가한다.

마지막으로 thread-based approach를 구현하기 위해 fine-grained locking을 추가해야 한다. 이를 통해 threading으로 인한 readers-writers 문제를 해결할 수 있다. 주식 자료 구조 파일에 구현된 함수 중 node를 수정을 하는 부분에는 node의 writer mutex를 lock하고 unlock하는 기능을 구현하고 buffer에 write하거나 출력하는 부분은 reader mutex를 lock하고 unlock하는 기능을 구현한다. Locking 외에도 각 connection에 대응하는 buffer를 관리하는 sbuf 기능을 csapp 파일에 추가한다.

3. 구현 결과

Client 입장에서 buy, sell, show, exit과 같은 client의 요청은 서버의 세부적인 구현 방법과는 무관하게 정확하게 처리된다. Event-driven과 thread-based approach 모두 client들의 연결이 전부 종료되더라도 서버는 종료되지 않는다. Ctrl-C를 통해 server 프로그램을 종료해야 종료되기 전까지의 수행 결과가 stock.txt에 저장된다.

추가적으로 모든 client가 종료되면 자동으로 server 프로그램도 종료되도록 설계하고 싶었으나 해당 기능까지는 구현할 수 없었다. 하지만 logically하게 보자면 server는 현재 수행 중인 모든 client가 종료되더라도 추후에 새로운 client가 연결될 수도 있기 때문에 server를 종료하는 것은 옳은 설계가 아닐 수도 있다.

4. 성능 평가 결과 (Task 3)

Multiclient 총 실행시간(elapsed time)과 동시처리율(workload)로 성능평가를 실행.

- Multiclient 상수 조건

MAX_CLIENT	ORDER_PER_CLIENT	STOCK_NUM	BUY_SELL_MAX
1000	10	10	10

- 성능평가 계산과 출력

```
printf("Elapsed Time: %lf\nWorkload: %lf\n",
    ((end.tv_sec - start.tv_sec) + (double)(end.tv_usec - start.tv_usec) / 1000000),
    ((double)(num_client * ORDER_PER_CLIENT)) / ((end.tv_sec - start.tv_sec)
    + (double)(end.tv_usec - start.tv_usec) / 1000000)
);
```

- start & end 시간 측정 위치

```
    gettimeofday(&start, NULL);
/*  fork for each client process    */
    while(runprocess < num_client){
        //wait(&state);

        pids[runprocess] = fork();

        if(pids[runprocess] < 0)
            return -1;
```

```
        runprocess++;
    }
    for(i=0;i<num_client;i++){
        waitpid(pids[i], &status, 0);
    }

    gettimeofday(&end, NULL);
    printf("Elapsed Time: %lf\nWorkload: %lf\n",
        ((end.tv_sec - start.tv_sec) + (double)(end.tv_usec - star
        ((float)(num_client * ORDER_PER_CLIENT)) / ((end.tv_sec -
        + (double)(end.tv_usec - start.tv_usec) / 1000000)
    );

    return 0;
```

1. 워크로드: buy, show, sell 모두 service

   a. Event-based Approach

#Clients	10	20	50	100	200	500	1000
Time	0.0331	0.0376	0.0570	0.1289	0.2316	0.6318	1.0223
Workload	3018.5	5308.0	8765.8	7758.1	8634.1	7913.9	9781.7

```
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 10
Elapsed Time: 0.029109
Workload: 3435.363633
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 20
Elapsed Time: 0.045653
Workload: 4380.873108
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 50
Elapsed Time: 0.054373
Workload: 9195.740533
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 100
Elapsed Time: 0.124253
Workload: 8048.095418
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 200
Elapsed Time: 0.203241
Workload: 9840.534144
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 500
Elapsed Time: 0.508598
Workload: 9830.947035
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 1000
Elapsed Time: 0.883880
Workload: 11313.752998
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$
```

   b. Thread-based Approach

#Clients	10	20	50	100	200	500	1000
Time	0.0283	0.0263	0.0473	0.0825	0.1545	0.3678	0.7215
Workload	3536.0	7610.4	10573.5	12126.7	12941.5	13592.8	13859.7

```
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 10
Elapsed Time: 0.033653
Workload: 2971.503284
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 20
Elapsed Time: 0.032163
Workload: 6218.325405
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 50
Elapsed Time: 0.045180
Workload: 11066.843736
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 100
Elapsed Time: 0.086711
Workload: 11532.562189
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 200
Elapsed Time: 0.153398
Workload: 13037.979635
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 500
Elapsed Time: 0.364862
Workload: 13703.811304
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 1000
Elapsed Time: 0.794328
Workload: 12589.257838
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$
```

2. 워크로드: buy, sell만 service

a. Event-based Approach

#Clients	10	20	50	100	200	500	1000
Time	0.0291	0.0284	0.0519	0.1059	0.2032	0.5151	0.8676
Workload	3441.4	7030.4	9631.5	9443.0	9844.3	9706.6	11525.8

```
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 10
Elapsed Time: 0.029058
Workload: 3441.393076
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 20
Elapsed Time: 0.028448
Workload: 7030.371204
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 50
Elapsed Time: 0.051913
Workload: 9631.498854
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 100
Elapsed Time: 0.105898
Workload: 9443.048972
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 200
Elapsed Time: 0.203164
Workload: 9844.263748
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 500
Elapsed Time: 0.515115
Workload: 9706.570377
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 1000
Elapsed Time: 0.867620
Workload: 11525.783177
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$
```

b. Thread-based Approach

#Clients	10	20	50	100	200	500	1000
Time	0.0283	0.0263	0.0473	0.0825	0.1545	0.3678	0.7215
Workload	3535.9	7610.4	10573.5	12126.7	12941.5	13592.8	13859.7

```
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 10
Elapsed Time: 0.028281
Workload: 3535.942859
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 20
Elapsed Time: 0.026280
Workload: 7610.350076
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 50
Elapsed Time: 0.047288
Workload: 10573.507021
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 100
Elapsed Time: 0.082463
Workload: 12126.650740
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 200
Elapsed Time: 0.154542
Workload: 12941.465750
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 500
Elapsed Time: 0.367842
Workload: 13592.792558
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 1000
Elapsed Time: 0.721514
Workload: 13859.744925
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$
```

3. 워크로드: show만 service

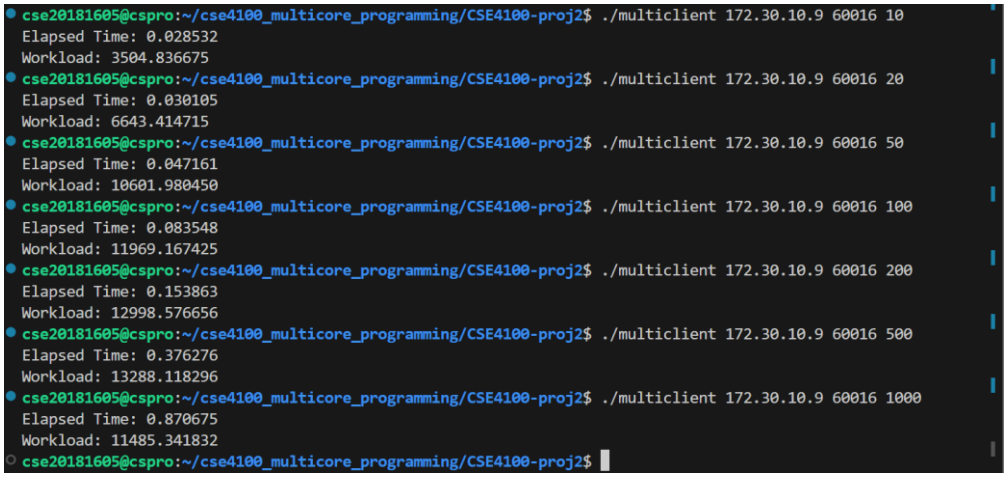
   a. Event-based Approach

#Clients	10	20	50	100	200	500	1000
Time	0.0331	0.0377	0.0570	0.1289	0.2316	0.6318	1.0223
Workload	3018.5	5308.0	8765.8	7758.1	8634.1	7913.9	9781.7

```
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 10
Elapsed Time: 0.033129
Workload: 3018.503426
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 20
Elapsed Time: 0.037679
Workload: 5307.996497
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 50
Elapsed Time: 0.057040
Workload: 8765.778401
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 100
Elapsed Time: 0.128898
Workload: 7758.072274
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 200
Elapsed Time: 0.231641
Workload: 8634.050103
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 500
Elapsed Time: 0.631796
Workload: 7913.946907
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$ ./multiclient 172.30.10.9 60016 1000
Elapsed Time: 1.022316
Workload: 9781.711330
cse20181605@cspro:~/cse4100_multicore_programming/CSE4100-proj2$
```

   b. Thread-based Approach

#Clients	10	20	50	100	200	500	1000
Time	0.0285	0.0301	0.0472	0.0835	0.1539	0.3763	0.8707
Workload	3504.8	6643.4	10602.0	11969.2	12998.6	13288.1	11485.3



4. Analysis

예측한 바와 동일하게 모든 경우에 대해 event-driven approach보다 thread-based approach가 실행시간과 처리율 면에서 client의 수가 늘어날수록 성능이 좋은 것을 확인할 수 있다. 그리고 show만을 request하는 configuration이 가장 오린 시간이 걸렸고 처리율도 상대적으로 낮은 것을 확인할 수 있다.