System Programming Project 2

담당 교수 : 김영재 교수님

이름 : 홍주표

학번 : 20181702

1. 개발 목표

주식거래를 하는 server에 여러 client가 접속을 하고 서비스를 요청하는 concurrent server를 Event-driven Approach와 Thread-based Approach로 구현하고 둘의 성능 차이를 알아보는 것이 이번 프로젝트의 목표이다.

두 방식의 주식 데이터에 대한 관리는 Binary tree를 사용한다는 공통점이 있다. 또한 stock.txt파일의 입출력을 통해 데이터를 불러오거나 입력하고, show, buy, sell, exit에 대한 기능을 정상적으로 작동하도록 한다.

2. 개발 범위 및 내용

A. 개발 범위

1. Task 1: Event-driven Approach

server를 실행시키고 여러 client가 그 서버에 접속한 후, 서비스(show, buy, sell)를 요청하면 서버가 해당 요청을 처리하여 client에 결과를 보낸다. 이 때 다른 클라이언트가 요청한 서비스와 충돌이 일어나지 않음을 확인한다.

2. Task 2: Thread-based Approach

Event-driven Approach와 유사하게 서버 실행, 여러 client의 접속, client의 서비스 요청, server의 서비스 처리의 단계를 거쳐 정상적으로 작동하는지와 concurrent하게 작동하는지를 확인한다.

3. Task 3: Performance Evaluation

기존에 주어진 multiserver.c 파일의 일부를 수정하며 test를 진행한다. 예를 들면 client가 요청하는 서비스가 buy인 경우 혹은 sell인 경우 혹은 random하게 buy, sell, show를 요청하는 경우 등으로 경우의 수를 나눈다.

B. 개발 내용

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

✓ Multi-client 요청에 따른 I/O Multiplexing 설명

Application이 단일 프로세스에서 자신만의 logical flow를 스케줄하는 concurrent 형식을 I/O multiplexing이라고 한다. 이 때 epoll 함수나 select 함수를 사용한다.

여기서 select 함수는 file descriptor를 보관하는 read_set 배열을 사용한다. 이 배열에는 connfd, listenfd가 들어간다.

select 함수를 호출하기 전에 ready_set에 read_set의 상태를 보관하고, select 함수가 호출되면 ready_set에 어떤 file descriptor가 사용되는지에 따라서 요청을 처리하기 때문에 file descriptor가 많아질 수록 loop를 여러 번 돌아야한다는 점에서 overhead가 증가한다.

- ✓ epoll과의 차이점 서술

epoll 함수는 위에서 서술한 overhead를 줄일 수 있는 방법이다. 반복문을 사용하지 않고, 운영체제에서 file descriptor를 관리한다. 따라서 select에서의 단점이 보완되는 방법이라고 할 수 있다. 하지만 다양한 운영체제에서 사용가능하지 않다는 점이 단점이다.

- Task2 (Thread-based Approach with pthread)

- ✓ Master Thread의 Connection 관리

Master Thread와 Client들은 공유 변수 sbuf를 통해 연결되고 해당 connection을 관리한다.

- ✓ Worker Thread Pool 관리하는 부분에 대해 서술

Worker Thread Pool에서 Worker Thread가 Client의 요청을 처리하고 요청이 끝나는 경우에 Pthread_detach 함수를 통해 해당 Thread를 reap하는 방식으로 관리한다.

- Task3 (Performance Evaluation)

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

$$V = \frac{10N}{S}$$

V: 동시 처리율, N: Client의 수, S: multiclient에서 10번의 order를 하고 얻은 시간(sec)

multiclient.c 파일에서 usleep 부분을 제거하고 테스트한다.

client가 buy만 하는 경우, sell만 하는 경우, show만 하는 경우, random하게 세 개의 서비스를 요청하는 경우 등 여러 경우의 수로 나누어 테스트한다.

각각의 상황마다 5번씩의 test를 진행하며 가장 적은 시간이 걸린 test의 결과값을 사용한다.

✓ Configuration 변화에 따른 예상 결과 서술

buy만 하거나 sell만 할 때는 show를 할 때와는 다르게 stock 내의 모든 id에 접근할 필요가 없기 때문에 후자보다 시간이 적게 걸리고 buy와 sell을 보면 둘의 performance는 비슷할 것으로 예상된다.

C. 개발 방법

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

먼저 stock의 정보를 담는 tree를 struct로 선언하고 leftChild, item, rightChild를 구조체의 멤버로 가진다. 또한 item은 struct로 선언되어 있고, ID, left_stock, price, readcnt를 멤버로 가진다.

Binary Tree를 구현하는 데 필요한 함수인 modifiedSearch, insertNode, binarySearch 함수는 이전에 자료구조 강의 때 배운 코드를 활용한다.

File Descriptor를 다루는 구조체 pool과 함수 init_pool, add_client, check_clients 함수는 시스템프로그래밍 강의 때 배운 코드를 활용한다.

이로써 Concurrent한 서버 구현의 틀을 잡고 client가 요청하는 서비스인 show, buy, sell 함수와 stock.txt를 최신화해주는 함수인 updateNode 함수를 추가로 구현한다.

먼저 show 함수는 char* show(int connfd, treePointer root, char *showbuf); 로 구성된다. 이 때 root는 전역변수로 선언된 이진트리의 최상단 노드이고, showbuf는 show 함수 실행 후 가지고 있어야 할 내용을 담고 있고 MAXLINE만큼의 크기를 가지는 변수이다. 자료구조 강의 때 배운 inorder traversal과 유사한 방법으로, 재귀를 사용하여 show 함수를 구현한다. show 함수의 return 값을 showres라는 변수에 저장한 후 이를 Rio_writen(connfd, showres, MAXLINE); 을 사용하여 원하는 결과를 출력한다. 이 때 showbuf 변수에 show 함수의 값을 넣고 이를 showres에 strcat을 이용하여 붙여넣는 것은 Rio_writen을 사용하기 위함이다.

두 번째로 buy 함수는 void buy(int connfd, treePointer root, int ID, int num, char *buybuf); 로 구성된다. 먼저 이진 트리에서 찾고자 하는 ID를 가진 주식을 binarySearch 함수를 통해 찾아낸 후 buy를 진행한다. 해당 ID를 가진 주식의 left_stock과 num을 비교하여 조건에 맞게 출력과 left_stock 관리를 한다.

sell 함수도 buy 함수와 유사하게 구현한다. void sell(int connfd, treePointer root, int ID, int num, char *sellbuf); 로 구성한다.

void updateNode(FILE *fp, treePointer root)로 구성된 updateNode 함수는 client의 요청이 모두 끝난 후 실행된다. 이전까지 buy나 sell을 통해 바뀐 left_stock을 stock.txt 파일에 반영하는 기능을 가진다. 이 때 단순 string으로 구현된 것이 아니라 이진트리로 stock을 다루기 때문에 show 함수와 비슷한 inorder 형태로, 재귀를 사용하여 updateNode 함수를 실행한다.

- Task2 (Thread-based Approach with pthread)

Task1과 유사한 구조체와 함수로 구현하였다. 대신 item 구조체에 sem_t mutex, w; 를 추가로 구현하여 semaphore를 다룬다. 그리고 sbuf_t 구조체, sbuf_t sbuf 전역변수, static sem_t mutex 전역변수를 사용한다.

client가 요청하는 서비스인 show, buy, sell 함수는 task1과 유사하게 사용한다. thread를 다루는 함수 thread, sbuf_init, sbuf_deinit, sbuf_insert, sbuf_remove 함수는 강의 시간에 배운 코드를 활용한다.

또한 show, buy, sell 함수를 사용할 때 일어날 readers-writers problem의 해결방법을 적용한다. 이 때 show가 reader에 해당하고, buy와 sell은 writer에 해당한다.

3. 구현 결과

기존에 stock.txt에 존재하는 내용을 show 함수를 통해 출력하고, buy, sell 또한 상황에 맞게 실행되는 모습을 보여준다. 또한 둘 이상의 client가 접속할 때에도 정상적으로 기능이 작동하고, client의 요청이 끝나고 stock.txt에 내용이 정상적으로 업데이트 되는 것도 볼 수 있다.

4. 성능 평가 결과 (Task 3)

본 task는 multiclient.c 파일에서 MAX_CLIENT 값을 1000, ORDER_PER_CLIENT 값은 10, STOCK_NUM을 10, BUY_SELL_MAX 값을 10으로 설정하고 원하는 출력값만을 얻기 위해 Fputs와 usleep을 주석처리한 후 진행한다.

① Show만 요청하는 경우

```
[cse20181702@cspro5:~/SP/Project2/Phase1$ ./multiclient 172.30.10.1 60154 20
elapsed time : 0.046846
[cse20181702@cspro5:~/SP/Project2/Phase1$ ./multiclient 172.30.10.1 60154 40
elapsed time : 0.090545
[cse20181702@cspro5:~/SP/Project2/Phase1$ ./multiclient 172.30.10.1 60154 60
elapsed time : 0.150226
[cse20181702@cspro5:~/SP/Project2/Phase1$ ./multiclient 172.30.10.1 60154 80
elapsed time : 0.192590
[cse20181702@cspro5:~/SP/Project2/Phase1$ ./multiclient 172.30.10.1 60154 100
elapsed time : 0.237094
[cse20181702@cspro5:~/SP/Project2/Phase2$ ./multiclient 172.30.10.1 60154 20
elapsed time : 0.016241
[cse20181702@cspro5:~/SP/Project2/Phase2$ ./multiclient 172.30.10.1 60154 40
elapsed time : 0.030221
[cse20181702@cspro5:~/SP/Project2/Phase2$ ./multiclient 172.30.10.1 60154 60
elapsed time : 0.044698
[cse20181702@cspro5:~/SP/Project2/Phase2$ ./multiclient 172.30.10.1 60154 80
elapsed time : 0.058982
[cse20181702@cspro5:~/SP/Project2/Phase2$ ./multiclient 172.30.10.1 60154 100
elapsed time : 0.073287
```

위의 사진은 Event-driven 방법 중 client가 show만 요청하는 경우이고, 아래의 사진은 Thread-based 방법 중 client가 show만 요청하는 경우이다.

이를 표로 정리하면

		20	40	60	80	100	500	1000
Event	시간(sec)	0.046846	0.090545	0.150226	0.192590	0.237094	1.306458	2.484622
	V	4269.308	4417.693	3993.982	4153.902	4217.736	3827.142	4073.947
Thread	시간(sec)	0.016241	0.030221	0.044698	0.058982	0.073287	0.399875	0.730319
	V	12314.513	13235.829	13423.419	13563.460	13644.985	12503.908	13692.647

② Buy 또는 Sell만 요청하는 경우

```
[cse20181702@cspro5:~/SP/Project2/Phase1$ ./multiclient 172.30.10.1 60154 20
elapsed time : 0.043888
[cse20181702@cspro5:~/SP/Project2/Phase1$ ./multiclient 172.30.10.1 60154 40
elapsed time : 0.088259
[cse20181702@cspro5:~/SP/Project2/Phase1$ ./multiclient 172.30.10.1 60154 60
elapsed time : 0.138967
[cse20181702@cspro5:~/SP/Project2/Phase1$ ./multiclient 172.30.10.1 60154 80
elapsed time : 0.190730
[cse20181702@cspro5:~/SP/Project2/Phase1$ ./multiclient 172.30.10.1 60154 100
elapsed time : 0.261547
```

```
[cse20181702@cspro5:~/SP/Project2/Phase2$ ./multiclient 172.30.10.1 60154 20
elapsed time : 0.016728
[cse20181702@cspro5:~/SP/Project2/Phase2$ ./multiclient 172.30.10.1 60154 40
elapsed time : 0.030012
[cse20181702@cspro5:~/SP/Project2/Phase2$ ./multiclient 172.30.10.1 60154 60
elapsed time : 0.044265
[cse20181702@cspro5:~/SP/Project2/Phase2$ ./multiclient 172.30.10.1 60154 80
elapsed time : 0.058423
[cse20181702@cspro5:~/SP/Project2/Phase2$ ./multiclient 172.30.10.1 60154 100
elapsed time : 0.072499
```

위의 사진은 Event-driven 방법 중 client가 buy 혹은 sell만 요청하는 경우이고, 아래의 사진은 Thread-based 방법 중 client가 buy 혹은 sell만 요청하는 경우이다.

이를 표로 정리하면

		20	40	60	80	100	500	1000
Event	시간(sec)	0.043888	0.088259	0.138967	0.190730	0.261547	1.273534	2.579014
	V	4557.054	4532.116	4317.572	4194.411	3823.405	3926.083	3877.451
Thread	시간(sec)	0.016728	0.030012	0.044265	0.058423	0.072499	0.363971	0.733265
	V	11956.002	13328.002	13554.727	13693.237	13793.294	13737.358	13637.634

③ Random하게 요청하는 경우

```
[cse20181702@cspro5:~/SP/Project2/Phase1$ ./multiclient 172.30.10.1 60154 20
elapsed time : 0.042651
[cse20181702@cspro5:~/SP/Project2/Phase1$ ./multiclient 172.30.10.1 60154 40
elapsed time : 0.085614
[cse20181702@cspro5:~/SP/Project2/Phase1$ ./multiclient 172.30.10.1 60154 60
elapsed time : 0.146128
[cse20181702@cspro5:~/SP/Project2/Phase1$ ./multiclient 172.30.10.1 60154 80
elapsed time : 0.183198
[cse20181702@cspro5:~/SP/Project2/Phase1$ ./multiclient 172.30.10.1 60154 100
elapsed time : 0.236702
[cse20181702@cspro5:~/SP/Project2/Phase2$ ./multiclient 172.30.10.1 60154 20
elapsed time : 0.016254
[cse20181702@cspro5:~/SP/Project2/Phase2$ ./multiclient 172.30.10.1 60154 40
elapsed time : 0.030164
[cse20181702@cspro5:~/SP/Project2/Phase2$ ./multiclient 172.30.10.1 60154 60
elapsed time : 0.044298
[cse20181702@cspro5:~/SP/Project2/Phase2$ ./multiclient 172.30.10.1 60154 80
elapsed time : 0.058906
[cse20181702@cspro5:~/SP/Project2/Phase2$ ./multiclient 172.30.10.1 60154 100
elapsed time : 0.072506
```

위의 사진은 Event-driven 방법 중 client가 random하게 요청하는 경우이고, 아래의 사진은 Thread-based 방법 중 client가 random하게 요청하는 경우이다.

이를 표로 정리하면

		20	40	60	80	100
Event	시간(sec)	0.042651	0.085614	0.146128	0.183198	0.236702
	V	4689.222	4672.133	4105.989	4366.860	4224.721
Thread	시간(sec)	0.016254	0.030164	0.044298	0.058906	0.072506
	V	12304.664	13260.841	13544.630	13580.960	13791.962