System Programming Project 5

담당 교수 : 김영재

이름 : 함희원

학번 : 20172067

1. 개발 목표

해당 프로젝트는 여러 client들이 동시 접속하여, 주식 서비스를 제공하기 위한 concurrent stock server이다. 본 프로젝트를 위해서 Event_based approach과 thread based approach를 활용하여 구현하였고 두 방식의 차이점에 대해서 간단한 실험을 진행하였다. 우선, 전자의 경우 select 함수를 활용하여 파일 descripter의 변화를 확인하고 서비스를 진행하였다. 후자의 경우 pthread 함수를 활용하여 client 연결시 각 thread를 부여하고 처리하였다.

2. 개발 범위 및 내용

A. 개발 범위

1. Select

Show : 이진트리를 순회하여 현재 주식상태를 출력해준다.

Buy : 이진트리를 순회하여 해당 id를 가진 노드를 찾고, 요청 개수와 남은 수량을 비교하여 구매 처리해주고, 완료 메시지를 전송한다.

Sell : 이진트리를 순회하여 해당 id를 가진 노드를 찾고, 요청 수량만큼 남은 수량에 더해준다. 그리고 완료 메시지를 전송한다.

Exit : exit 입력시 클라이언트를 종료하고, 서버도 역시 클라이언트 연결 끊김을 확인하고 종료해준다.

2. Pthread

Show, buy, sell은 동일하고, exit의 경우 exit 메시지를 서버에 전송 후 종료하고, 서버는 exit를 받은 뒤 echo를 나간 후 해당 connfd를 close 해준다.

B. 개발 내용

- stock info에 대한 file contents를 memory로 올린 방법 설명 : 서버가 시작된 후 파일 포인터를 이용하여 stock.txt를 읽고 한 줄 씩 parsing하여 이진 트리에 저장하였다.
- **select**

✓ 소켓 집합을 위한 자료구조 fd_set을 만든다. Accept를 하기 전 master file description을 모두 종료하고 고객을 맞이하는 socketfd만 켜준다. 소켓 집합이 몇 개인지 알 수 없기 때문에 fdmax를 사용해서 저장하고 while 안에서 select를 호출하여 소켓에 메시지가 왔을 경우,0부터 fdmax만큼 순회하여 해당 소켓의 id를 확인한다. fd가 0일 경우 새로운 고객일 맞이하는 소켓이므로 새로운 accept를 통해 fd를 만들어준다. 그 외의 경우 기존에 소켓으로 전달했으므로 echo를 활용하여 해당 명령어를 parsing하고 결과값을 문자열에 담아서 전송해준다. Sell, buy, show의 경우 쓰레드 방법과 동일하고, exit의 경우 recv함수을 통해서 처리 가능하므로 echo안에서 별도로 처리하지 않았다. client에서 먼저 연결을 끊고 후에 select를 통해서 소켓을 확인하고 해당 내용 buf가 0일 경우 연결종료로 판단하여 fd set을 종료한다.

- **pthread**

✓ Accept 이후 thread를 하나씩 생성하였다. 쓰레드 함수 내에서 쓰레드를 detach 모드로 선언하고 에코 함수를 실행하였다. Echo 함수는 고객과의 연결이 끊길 때까지 while문을 활용하여 고객의 명령어를 수행한다. 고객의 입력이 들어오면 parsing하여 명령어, 첫번째 인자, 두번째 인자를 구분하여 처리하였다. 해당 명령어에 대한 상세한 예외처리는 하지 않았지만 잘못된 명령의 경우 고객에게 알려줄 수 있도록 메시지를 전송했다. Sell과 buy의 경우 해당 노드를 찾고 그 노드의 수량을 변경하는 과정에서 세마포를 사용하였고, show의 경우 해당 노드를 찾고 버퍼에 담는 과정에서 한 노드씩 세마포를 활용하였다. 에코 함수가 종료된 이후에는 열려있는 thread의 개수를 확인하여 마지막 thread인 경우 txt파일을 업데이트를 하였다.

C. 개발 방법

- **Select**

- 쓰레드를 먼저 완료 후 select함수를 구현하여, 전반적인 이진트리 관련 함수는 복사 붙여넣기를 진행하였다. 그리고 main 함수내에서 select를 사용하기 위해서 네트워크 수업에서 배웠던 select 함수를 구현하였고, select를 이용하여 파일 디스크립터에 변경된 fd가 sockfd인 경우 새로운 클라이언트로 accept하고 fd_set을 활성화시키고, 기존의 클라이언트에게 온 경우 echo함수를 수행하였다. 쓰레드와 다르게 echo은 while이 아니며 echo에서 처리하는 내용을 문자열로 저장하여 main함

수 내에서 send 처리를 진행하였다. echo내에 기존의 내용은 쓰레드와 동일하다.

- **Pthread**

✓ 기존 main 함수 while문에 connfd를 포인터형식으로 변경하여 새 client가 올 때마다 연결해주고 thread를 생성한다. 그리고 전역변수인 count(thread 개수)를 따로 관리하여 추후에 count가 0일 때 txt 파일을 새로 갱신할 수 있도록 하였다.

✓ Ppt에 있는 thread 함수를 참조하여 추가하고 Close(connfp) 밑에 추가적인 내용을 작성하였다. 전역변수 count를 하나씩 감소하기 위해서 세마포로 감싸 처리하고, 만약 count가 0일 경우우, 파일포인터를 이용해서 stock.txt 파일을 업데이트 하였다.

✓ echo함수를 따로 파일로 놓지 않고 main내 함수로 이동했다. 그리고 기존 echo를 내용을 수정하여 클라이언트로 오는 내용을 받아서 parsing하고 해당 command가 정해진 형식을 만족하는 경우, 해당 명령어를 처리하였다. 예를 들어, show의 경우 Stock_show라는 함수에 문자열을 매개변수로 받아서 그 매개변수 문자열에 이진트리를 순회하며 문자를 strcat하였다. 해당 함수가 끝난 뒤 문자열을 클라이언트에게 전송하였다. 그리고 sell, buy의 경우 Stock_search를 통해서 해당 노드를 찾고 그 노드의 수량을 변경하기 위해서 세마포로 막은 뒤 처리하였다. 그리고 처리가 완료한 뒤 고객에게 완료 메시지를 보내주었다. 마지막 exit는 해당 고객에게 내용을 그대로 보내주고 break를 통해서 echo를 끝내고 thread 함수로 돌아가 Close로 해당 connfd를 닫았다.

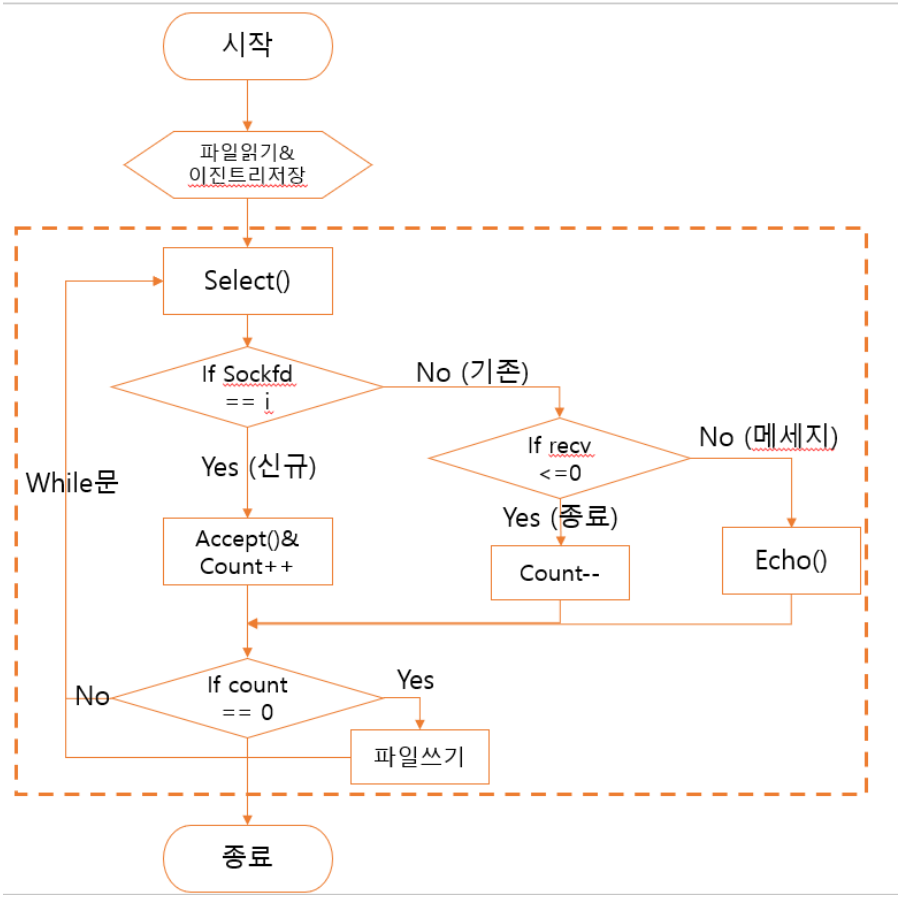
Client

➢ multiclient, stockclient 모두 show를 구현하기 위해서 `Rio_readlinenb` 함수를 `Rio_readnb`함수로 변경하였다.

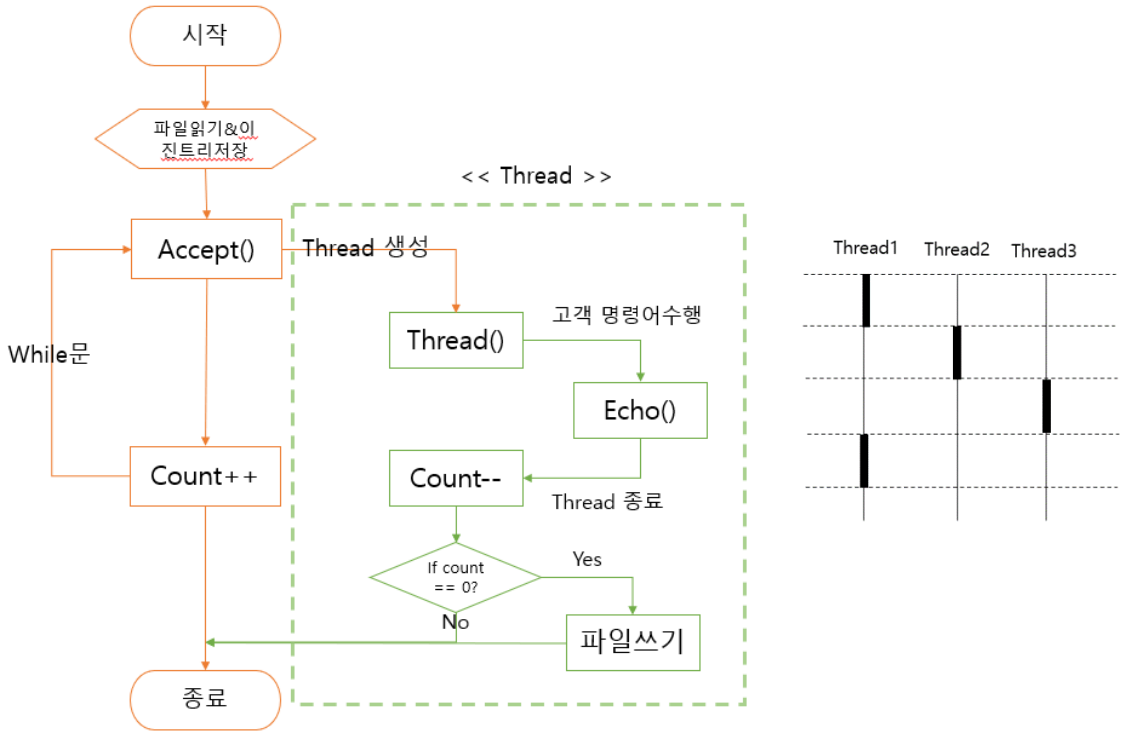
3. 구현 결과

A. Flow Chart

1. Select



2. pthread



B. 제작 내용

1. **Select**

✓ 쓰레드를 먼저 완료 후 진행하여, 전반적인 내용을 복사하였으나 event 기반 프로그래밍의 경우 echo에서 connf를 매개변수로 받아서 클라이언트로 전송하는 과정이 문제가 발생하였다. 이런 저런 방법을 사용하다가 쓰레드에서 해결했던 방법과 유사하게 이번에는 echo에서 처리하는 내용을 문자열로 저장하여 main내에서 클라이언트로 전송하는 방법으로 수정하였고 다행이 정상적으로 전송되었다.

2. **Pthread**

✓ Show : 클라이언트와 서버가 '주고 받기'의 과정의 연결을 가지고 있어서 show을 실행할 경우 한 줄만 전달되었다. 그래서 show 순회하는 내용을 한 문자열에 담아서 한번에 전송하여 해결하였다. 물론 주식의 내용이 많을 경우 문자열을 초과하는 문제가 발생할 것으로 예상된다. 만약 주식의 내용이 많을 경우, stockclient를 write , read의 주고 받는 echo형식이 아닌 서버에 오는 내용을 계속해서 받을 수 있도록 수정할 필요가 있어 보인다.

✓ 앞서 show의 내용을 한 문자열에 넣는 과정에서 기존의 client 서버는 readline으로 읽게 되어 ₩n 부분을 연결하지 못하였다. 이 부분을 해결하기 위해서 client에서 `Rio_readnb`함수로 변경하였다.

✓ 쓰레드 함수에서 가장 중요한 것은 공유변수의 처리 방법이었기 때문에 오랜 시간동안 고민을 했다. Reader, Writer problem처럼 세마포 w, readcnt를 추가적으로 사용할까 고민하다 그럴 경우 sequential한 처리가 될 것으로 판단되어 구조체에 있는 mutex만 사용하기로 결정했다. buy, sell에서는 해당노드의 내용을 변경할 경우 세마포를 사용하였고, show에서는 strcat하는 작업에서 세마포를 사용하였다.

-

C. 시험 및 평가 내용

Event based server의 경우, 하나의 논리적인 control flow를 가지고 있으므로 속도가 현저히 낮을 것으로 예상했다. 그러나 thread와 속도에서 차이가 나지 않아서 당황스러웠다. 기존 2, 4, 8명의 클라이언트의 실행속도를 3회측정하여 평균을 내어 처리하려고 했으나 속도 비슷하여 16명의 고객까지 추가적으로 실험을 진행하게 되었다. 그 결과 16명의 경우 thread가 약간의 속도가 더 빠름을 알 수

있었다. 아마 더 많은 고객을 진행하게 된다면 thread가 더 빠른 속도를 가지고 있을 것으로 생각되어진다. 수업의 내용처럼 event based는 코딩면에서 더 복잡하였으나 쓰레드를 하나만 사용하였고, Thread는 공유변수에 주의하여 처리한다면 더 효율적인 결과를 낼 수 있다는 것을 확인하게 되었다.

Client 수	2	4	8	16
Event-based server	10.49 10.69 10.35	10.86 10.88 10.54	10.70 10.94 10.88	11.59 11.54 11.21
Thread-based server	10.92 11.14 11.11	10.82 10.70 10.62	10.88 10.49 10.69	10.82 10.62 10.70
이벤트 평균	10.51	10.76	10.84	11.44
쓰레드 평균	11.05	10.71	10.68	10.71