System Programming Project 5

담당 교수 : 김영재

이름 : 권기윤

학번 :20161565

1. 개발 목표

- 해당 프로젝트에서 구현할 내용을 간략히 서술.
- (미니 주식 서버를 만드는 전체적인 개요에 대해서 작성하면 됨.)

Network programming과 echo 서버 프로그래밍을 배경지식으로 하여 여러 client들의 동시 접속 및 서비스를 위한 concurrent stock server를 구축하는 것을 목표로 한다. Stock server는 주식 정보를 저장하고 여러 client들과 소통하고 stock client는 각 client마다 server에 buy, sell, show등의 요청을 보내어 각 요청에 맞는 request를 server로부터 받는다. 이때, server는 각 client들의 요청에 맞는 기능을 수행하여 해당 요청에 상응하는 결과를 client들에게 socket통신을 이용하여 보낸다.

2. 개발 범위 및 내용

A. 개발 범위

- 아래 항목을 구현했을 때의 결과를 간략히 서술

1. Select

   Select 함수를 선언하면 server는 file descriptor pool을 감시하고 있다. 이때, 임의의 client들이 요청을 보내면, select 함수에서 해당하는 client들의 file descriptor에서 요청이 들어온 것을 확인하고 변화된 file descriptor를 return 해준다. 작성한 코드에서는 select 함수가 수행되고 fd_set변수 fdmask_backup에 어떤 file descriptor가 set 되었는지(어떤 client가 요청을 보냈는지) 저장이 되고 FD_ISSET()을 통해 해당 fd가 set 된 것을 감지하고 socket 통신을 시작하게 된다.

2. pthread

   stockserver에서 NTHREADS 만큼의 work thread를 생성한다. client의 첫 요청이 들어왔을 때 sbuf_insert() 함수를 사용하여 master thread에서 buffer로 descriptor를 insert하고, 선언한 work thread가 client의 요청에 따라 주식 종목 관리 테이블을 갱신하고 그 결과를 client한테 전달한다.

B. 개발 내용

- 아래 항목의 내용만 서술
- (기타 내용은 서술하지 않아도 됨. 코드 복사 붙여 넣기 금지)

- **select**

  ✓ select 함수로 구현한 부분에 대해서 간략히 설명

  ✓ stock info에 대한 file contents를 memory로 올린 방법 설명

  fd_set 변수 fdmask와 fdmashk_backup을 선언하여 fd pool중 어느 fd가 set 되어있는지 판단한다. Select를 실행하게 되면 이때 select에서 감시 대상이 되는 fd의 수는 maxfd+1로 지정해 주었는데, 이것은 다음 새로운 client의 요청이 들어왔을 때를 대비하여 설정했다. Select 함수로 fdmask_backup에 fd pool의 정보가 update(set)되면 처음 접속한 client의 경우 accept 함수를 사용하여 클라이언트의 접속 요청을 받아들이고, server-client간의 통신 전용 소켓을 생성한다. 요청을 보낸 client의 경우 connfd를 parameter로 하여 echo 함수로 넘겨주게 되고, echo서버에서 connfd에 대한 socket을 decapsulate 하여 client가 보낸 버퍼를 보고 그에 맞는 기능을 수행하여 다시 client에게 요청에 상응하는 결과를 request하게 된다.

- **pthread**

  ✓ pthread로 구현한 부분에 대해서 간략히 설명

  pthread_create로 worker thread를 선언한다. 각 worker thread는 client의 요청을 받은 Master thread에서 buffer를 전달받게 되고, 해당하는 client의 요청을 수행하게 된다.

C. 개발 방법

- **B.의 개발 내용을 구현하기 위해 어느 소스코드에 어떤 요소를 추가 또는 수정할 것인지 설명. (함수, 구조체 등의 구현이나 수정을 서술)**

- project_1

  1. stockserver.c

  fd_set 변수 fdmask 와 fdmask_backup을 선언하여 listentfd의 fd를 fdmask에 set 한다. Fdmask의 변질을 막기위해 loop 시작 마다 fdmask_backup에 fdmask를 저장하는 방식으로 구현하였다. loop에서는 select()를 수행하고, fd pool을 감시하다 요청을 받은 fd가 있으면 inner loop로 내려가게 된다. 이때, select()의 시간변수는 NULL로 하여 client의 요청이 들어올 때만 작동하게 하였다. Inner loop에서는 maxfd만큼 loop

를 돌며 set 된 fd가 있는지 확인한다. 이때, fd가 set 되었고, set된 fd가 lisetenfd라면 client가 처음 요청을 보낸 것, 즉 server에 접근한 것이므로, server-client 통신전용 socket을 생성한다. Set 된 fd가 listenfd가 아니라면 client가 stock에 대한 명령을 보낸 것이므로 echo(i)를 호출한다. 위 과정에서, 전체 client가 0이라면 server가 종료되어야 하므로, 전역변수 fd_setNum을 선언하여 server 최초 접근시에 fd_setNum++, echo(i) 수행시에 client 요청이 "exit"이라면 fd_setNum—, close(i),FD_CLR(I, &fdmask)를 하여 select 함수가 수행되기 전에 fd_setNum == 0 이라면 server를 종료했다. Server 종료시에 stock.txt 업데이트를 수행하였다.

2. echo.c

Concurrent한 program 작성을 위해 while문을 if문으로 바꾸어 client의 한 요청에만 echo함수를 수행하게 하고, 각 client의 요청마다 수행될 수 있게 변경하였다. Echo() 함수에서는 connfd로 전달된 client socket을 rio_readlineb()로 읽어 들였으며, socket에 buffer를 tokenize하여 각 요청에 맞는 기능을 수행하였다. 이때, stock DB는 이진트리 구조를 가지고 있어 buy, sell, show 요청에 대해 recursive하게 탐색하여 preorder로 결과를 buffer에 저장해주었다. 기능 수행 후 결과는 rio_writen()으로 client에게 보내주었다. 이때 server에서 "exit" 요청을 감지하기 위해 void -> int 형으로 함수를 형변환하여 "exit"일 때는 0을 넘겨주어 해당 client의 접속을 끊도록 하였다.

3. stockclient.c

Rio_readlineb로 읽은 buf string이 "exit"라면 해당 client의 통신을 끊는다. 나머지 요청이 "show" 경우, stockDB의 내용을 ID element별로 개행을 하여 출력을 해야한다. 하지만 rio_readlineb()는 개행까지 buffer로 가져오기 때문에 buffer에 개행 대신 '!'을 넣어 보내주었다. 따라서 client 화면에 출력할 때에는 '!'을 '₩n'으로 변경하여 출력해 주었다.

4. multiclient.c

요청이 "show" 경우, stockDB의 내용을 ID element별로 개행을 하여 출력을 해야한다. 하지만 rio_readlineb()는 개행까지 buffer로 가져오기 때문에 buffer에 개행 대신 '!'을 넣어 보내주었다. 따라서 client 화면에 출력할 때에는 '!'을 '₩n'으로 변경하여 출력해 주었다.

- project_2

1. stockserver.c

Pthread_create() 함수를 통해 worker thread를 생성하고, client들의 요청과 접속에 따라서 while() loop를 변경했다. Client의 요청에 따라 worker thread pool에서 하나의 thread를 할당하여 echo()의 역할을 수행하게 하였다. 따라서 void* thread()함수를 선언하여 client의 요청을 echo()함수로 넘겨주어 server-client간의 통신을 가능하게 했다. 전역변수 maxfd를 선언하여 client의 접속이 있을 때 증가시켜주고, worker thread에서의 echo(connfd)가 종료되었을 때 감소시켜준다. Maxfd를 감소시켰을 때 0 이라면 모든 client가 종료했을 때를 의미하므로 변경된 stockdb를 stock.txt에 저장한다.

2. stockclient.c

Rio_readlineb로 읽은 buf string이 "exit"라면 해당 client의 통신을 끊는다. 나머지 요청이 "show" 경우, stockDB의 내용을 ID element별로 개행을 하여 출력을 해야한다. 하지만 rio_readlineb()는 개행까지 buffer로 가져오기 때문에 buffer에 개행 대신 '!'을 넣어 보내주었다. 따라서 client 화면에 출력할 때에는 '!'을 '₩n'으로 변경하여 출력해 주었다.

3. multiclient.c

요청이 "show" 경우, stockDB의 내용을 ID element별로 개행을 하여 출력을 해야한다. 하지만 rio_readlineb()는 개행까지 buffer로 가져오기 때문에 buffer에 개행 대신 '!'을 넣어 보내주었다. 따라서 client 화면에 출력할 때에는 '!'을 '₩n'으로 변경하여 출력해 주었다.

4. echo.c

전역변수를 사용할 때, 다른 thread의 개입을 막기위해 P(), V()함수를 사용하였다. 세부 함수의 동작방식은 event-base와 비슷하나 client의 요청을 client와의 연결이 끊어질 때까지 받기위해 while문으로 작성하였다.

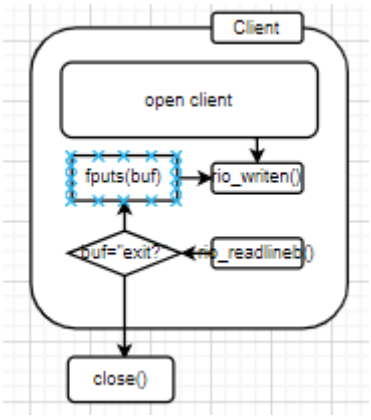
3. 구현 결과

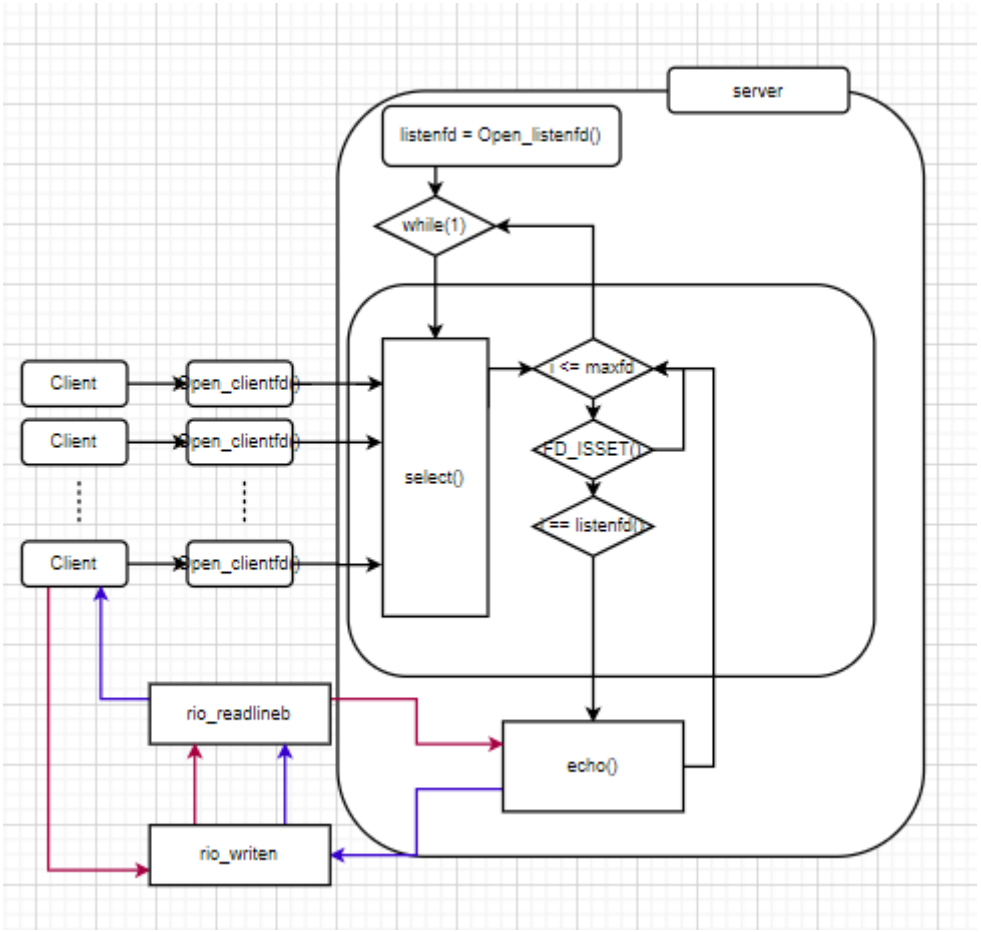
A. Flow Chart

- **2.B.개발 내용에 대한 Flow Chart를 작성.**

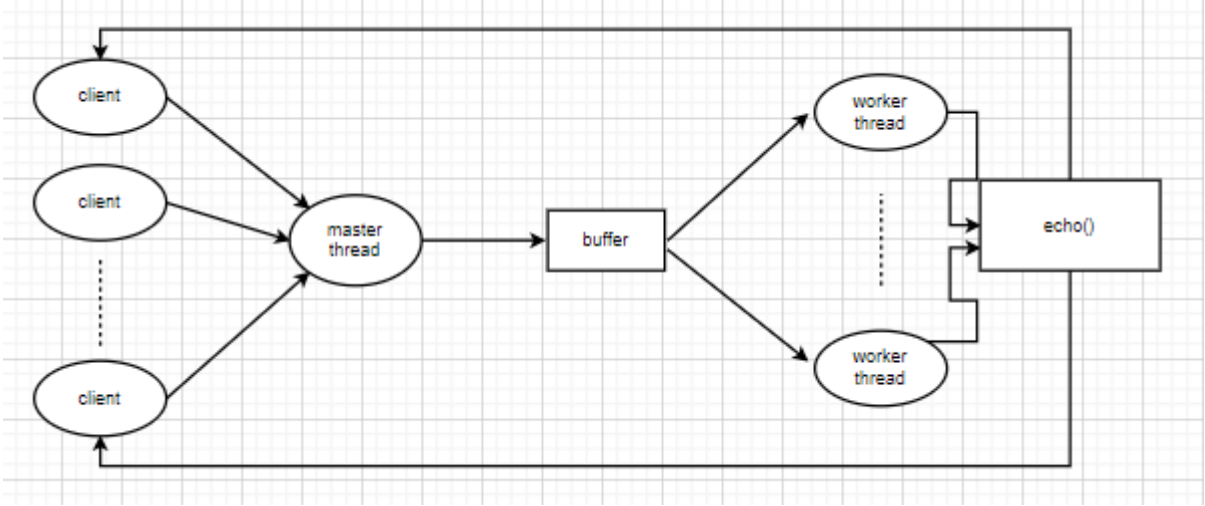
- (각각의 방법들(select, phtread)의 특성이 잘 드러나게 그리면 됨.)

1. Select





2. pthread



B. 제작 내용

- II. B. 개발 내용의 실질적인 구현에 대해 코드 관점에서 작성.
- 개발상 발생한 문제나 이슈가 있으면 이를 간략히 설명하고 해결책에 대해 설명.

1. Select

해당 task 설계 시에 1대1 통신을 기준으로 먼저 client의 응답을 처리하고, 그 다음 multiclient에 대한 통신을 구현하려 했습니다. Client의 응답에 대한 처리를 모두 마친 후에 multiclient 통신 구현 파트에서 크나큰 어려움을 겪었습니다. 결국 먼저 구현을 마쳤던 응답 파트를 다시 작성하고, multiclient 까지 구현을 완료하였습니다. 다음 설계시에는 전체적인 flow를 먼저 작성한 뒤에 세부 request detail등을 작성할 예정입니다.

2. pthread

해당 task 설계 시에 1개의 thread 통신을 위해서 설계를 했습니다. 허나 thread에 대한 개념이 이해가 가지 않아 설계하는데 굉장한 어려움이 있었습니다. P(),V() 함수를 사용할 때 미처 발견하지 못한 부분이 있어 multiclient 동작시에 client가 닫히지 않는 일이 일어났는데 함수 return 을 재배치하여 해결하였습니다.

Select에서는 client가 모두 접속을 종료하였을 때 server도 종료 했다면, pthread에서는 모든 client가 접속을 종료하더라도 server는 종료하지 않도록 구현하였습니다.

C. 시험 및 평가 내용

- select, pthread에 대해서 각각 구현상 차이점과 성능상에 예측되는 부분에 대해서 작성. (ex. select는 ~~한 점에 있어서 pthread보다 좋을 것이다.)
- 실제 실험을 통한 결과 분석 (그래프 삽입)
-