System Programming Project 5

담당 교수 :

이름 :김세영

학번 :20191571

1. 개발 목표

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

여러 client가 동시에 접속할 수 있는 주식 서버를 구축한다. 여러 client가 주식을 사고 팔거나 주식을 확인할 수 있고, concurrent 하게 이루어진다.

2. 개발 범위 및 내용

A. 개발 범위

- 아래 항목을 구현했을 때의 결과를 간략히 서술

1. select

   새로 client가 접속하면 server에서 새로운 client를 pool에 추가한다. 그 후 client가 server와 연결되어 sell,buy 명령어를 입력하면 해당 주식을 사거나 팔고, show 명령어를 입력하면 현재 server에 저장된 주식을 출력하고 exit을 입력하면 client는 접속을 종료한다. 주식이 update된 경우에는 text file도 update한다.

2. pthread

   새로 client가 접속하면 server가 새로 thread를 띄워서 client와 연결한다. 그 후 sell,buy 명령어를 입력하면 해당 주식을 사거나 팔고, show 명령어를 입력하면 현재 server에 저장된 주식을 출력하고 exit을 입력하면 client는 접속을 종료한다. 주식이 update된 경우에는 text file도 update한다.

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

  structure pool 을 구현하여 connected descriptor에 필요한 정보들을 저장한

다. main에서 pool을 초기화하고 새로운 client에 connection 요청이 오면 Select를 사용하여 server과 연결한다.

✓ stock info에 대한 file contents를 memory로 올린 방법 설명

binary search tree를 사용하였다. struct item에 mutex(thread에서 사용), price, left_stock,ID, readcnt를 저장하고 client의 connection request를 기다리기 전에 파일을 읽어 tree를 초기화한다. client의 명령어를 수행하는 과정에서 tree가 업데이트 되는데, 명령어를 한번 수행할 때마다 file을 update한다.

- **pthread**

✓ pthread로 구현한 부분에 대해서 간략히 설명

client로부터 connection 요청이 들어오면 Pthread_create()를 사용하여 새로운 thread를 생성한다. thread()에서는 pthread_detach()를 사용하여 별도의 reaping 과정이 필요없도록 하였다. thread()에서 echo()를 호출하여 client가 보낸 명령어를 수행하도록 하고, 접속이 종료되면 Close(connfd)를 한다.

C. 개발 방법

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

select: pool이라는 구조체를 만들어 connected descriptor에 필요한 정보들을 저장한다. init_pool()로 struct pool pool 의 변수들을 초기화하고, client에서 sever로 connection request가 들어온 경우 add_client()로 client를 pool에 추가하여 client와 server를 연결한다. check_clients()에서는 client에서 입력을 받아서 해당 명령어를 수행한다.

stock info: binary search tree 를 구현하기 위해 struct item이라는 구조체를 추가하였다. init_tree()에서 파일을 열어 insert_tree()를 호출하여 트리를 만들고, buy나 sell 명령어를 수행할 때 update_stock()에서 search_tree()를 호출하여 해당하는 노드를 찾는다. text file 을 수정할 때는 update_file()을 호출한다.

insert_tree()에서는 해당 노드에 왼쪽에는 더 작은 값이 , 오른쪽에는 더 큰 값이 오도록하고 recursion을 이용하여 구현한다.

show()에서는 tree를 탐색하는데, preorder traversal 방법을 사용하였다.

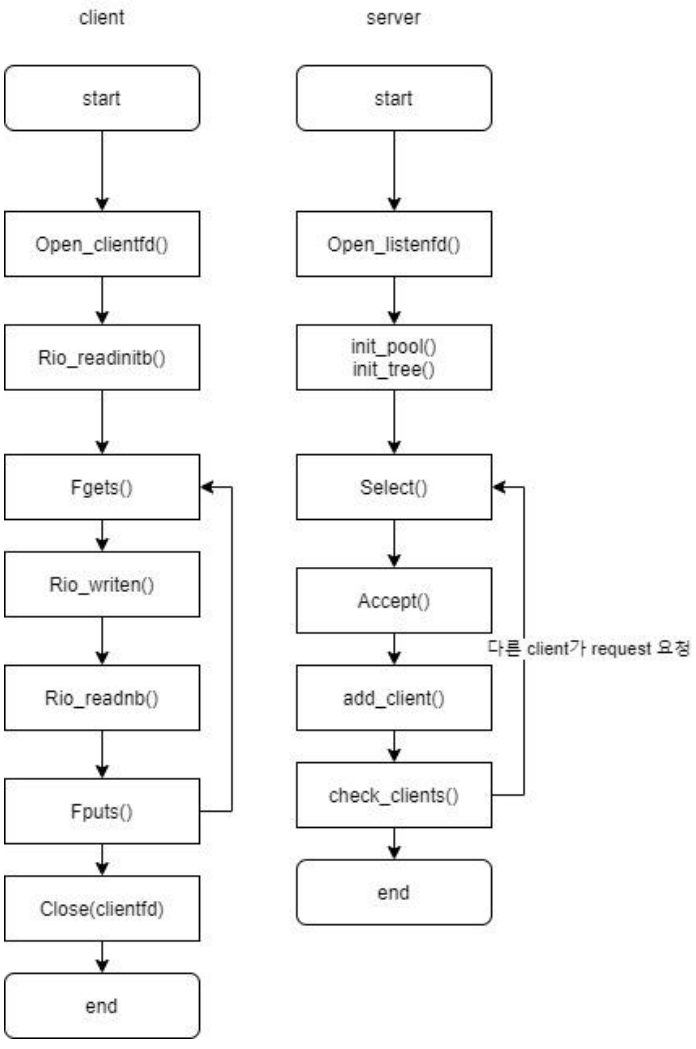
pthread: Pthread_create()에서 thread를 생성한다. 이때 Pthread_detach()를 사용하여 별도의 reaping과정이 필요없도록 한다. thread()에서 echo()를 호출하여 명령어를 수행한다.

3. 구현 결과

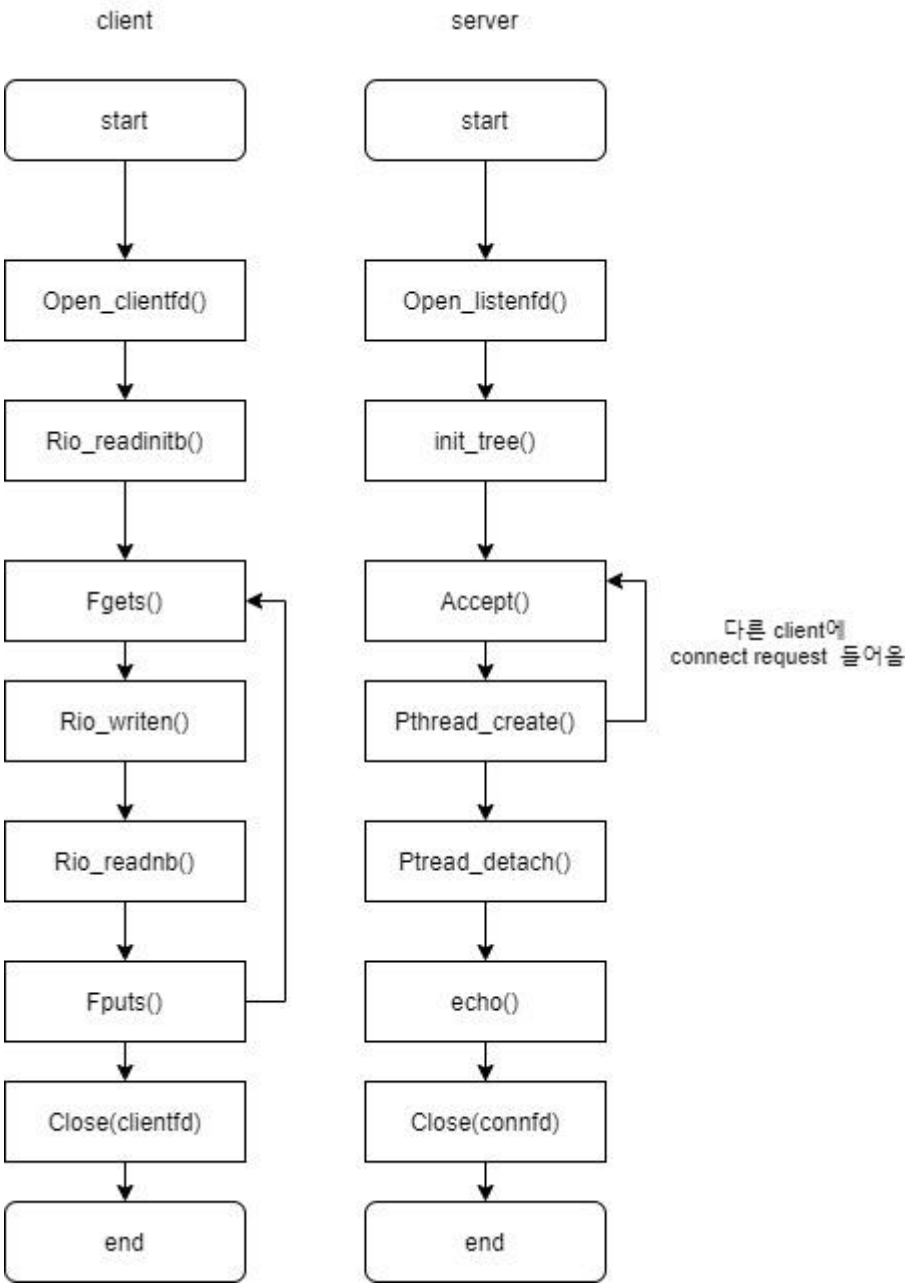
A. Flow Chart

- 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

server,client간의 read/write는 Rio 함수를 사용하여 구현하였고, 문자열 하나에 띄어쓰기가 포함되어 있기 때문에 Rio_readnb를 사용하였다. linked list를

사용하여 binary search tree의 left,right를 구현했다.

server에서 client로 Rio_writen()를 사용하여 출력할 메시지를 보내고 이를 Rio_readnb()으로 읽는데, 이때 readnb의 parameter n(몇바이트를 읽을지 결정)과 writen의 n이 일치하지 않으면 읽고 쓰기가 제대로 이루어지지 않는다. stock.txt에서 각 숫자가 공백으로 구분되어있으므로 strtok()를 사용하였고, 숫자를 문자열로 바꿀때는 itoa()함수 (별도로 구현)를 사용하였다.

server에서 client로 출력할 문자열을 printline이라는 변수를 사용하여 보내는데, 매 명령어 수행마다 이 문자열을 초기화해야하고, memset()을 사용하여 구현하였다.

main에서 while문을 돌며 client에서 connection request가 올때마다 Select()를 호출하고, listening descriptor 가 ready이면 Accept(),add_client()를 호출한다. check_clients()를 호출하여 client로부터 명령어를 받고 실행한다.

2. **pthread**

tree가 공유변수이기 때문에 P() ,V()를 사용하여 같은 node는 한번에 하나씩만 접근할수 있도록한다. 파일을 업데이트 하는 것도 겹칠 수 있기 때문에 파일도 세마포어를 사용하여 한번에 하나씩만 접근할 수 있도록 한다. stock.txt에 update된 내용을 쓸 때 Write()를 사용하였다. thread에서 echo서버가 종료한 후에 connfd를 close해야한다.

printline을 전역변수로 선언할 경우 세마포어를 사용해야 하므로 이를 local variable로 선언하고, printline이 사용되는 함수에 parameter passing하였다.

main에서 while문을 돌면서 client에 요청이 올때마다 Accept()를 호출하고 Pthread_create()로 새로운 thread를 생성하고 thread()를 호출한다. thread()에서 echo()를 호출하여 client로부터 명령어를 입력받는다.

C. 시험 및 평가 내용

- **select, pthread에 대해서 각각 구현상 차이점과 성능상에 예측되는 부분에 대해서 작성. (ex. select는 ~~한 점에 있어서 pthread보다 좋을 것이다.)**

select는 fd를 array형식으로(fd_set)저장하여 관리하고, pthread는 connection request가 들어올 때마다 새로운 thread 를 생성한다. 또한 thread는 공유변수가 있기 때문에 P() V()를 사용하여 공유변수를 한번에 하나씩만 접근할 수 있도록 하였다.

1) 확장성(client 개수): select는 control overhead가 없기 때문에 client process의 개수가 늘어날수록 select가 pthread에 비해 성능이 좋을 것이다.

2) 워크로드에 따른 분석

예1) 모든 client가 buy 또는 sell을 요청하는 경우

예2) 모든 client가 show만 요청하는 경우

예3) Client가 buy, show 등을 섞어서 요청하는 경우

select : 예1은 tree의 일부분을 탐색하고, show는 tree의 전체를 탐색하기 때문에 예3<예2<예1순으로 시간이 걸릴 것이다.

thread: 예1에서는 node의 data를 변경하기 때문에 세마포어를 사용하기 때문에 이 overhead로 인해 시간이 제일 많이 걸릴것이다. 즉 예3>예2>예1 순으로 시간이 걸릴 것이다.

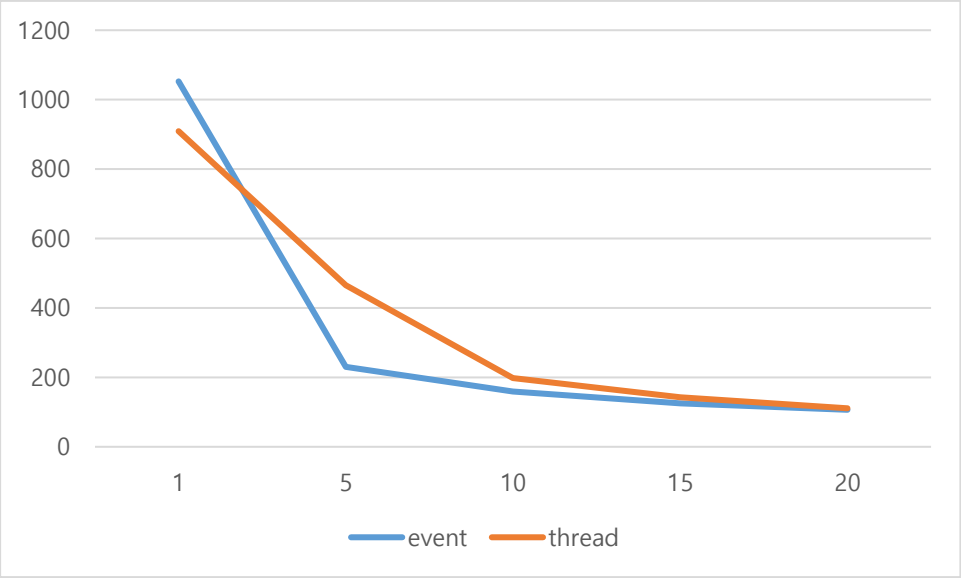
3) stock 항목 개수에 따른 분석

stock 항목이 늘어날수록 tree의 depth가 커지므로 실행시간이 늘어나는데, select는 공유변수를 사용하지 않아 thread보다 시간이 적게 걸릴것이다.

- **실제 실험을 통한 결과 분석 (그래프 삽입)**

1) 확장성(client 개수)

동시처리율은 client처리 요청개수/시간(초)이다.



x축은 process개수, y축은 동시처리율이다. stock의 가짓수는 10로 동일하고, 각 stock의 개수는 10이다.

process의 개수가 증가할 수록 동시처리율이 낮아진다. 또한 process개수가 1개일 때는 event가 더 빠르지만, process의 개수가 증가할수록 thread가 더 동시처리율이 높아진다.

event는 매번 명령어를 수행할 때마다 stock.txt 파일을 update하고, thread는 한 프로세스가 끝날 때만 stock.txt 파일을 update하도록 구현했기 때문에 process의 개수가 5이상인 구간에서 thread의 동시처리율이 event보다 높아지는 것이다. 또한, 이 결과는 각각 3번씩 수행한 평균 수치인데, 각 수행별로 show, buy, sell에 걸리는 시간이 다르기 때문에 수행당 시간 편차가 크게 나타난다. thread에서 세마포어를 사용하는 과정에서 overhead 가 크기 때문에 process 개수가 증가할수록 동시처리율이 감소한다.

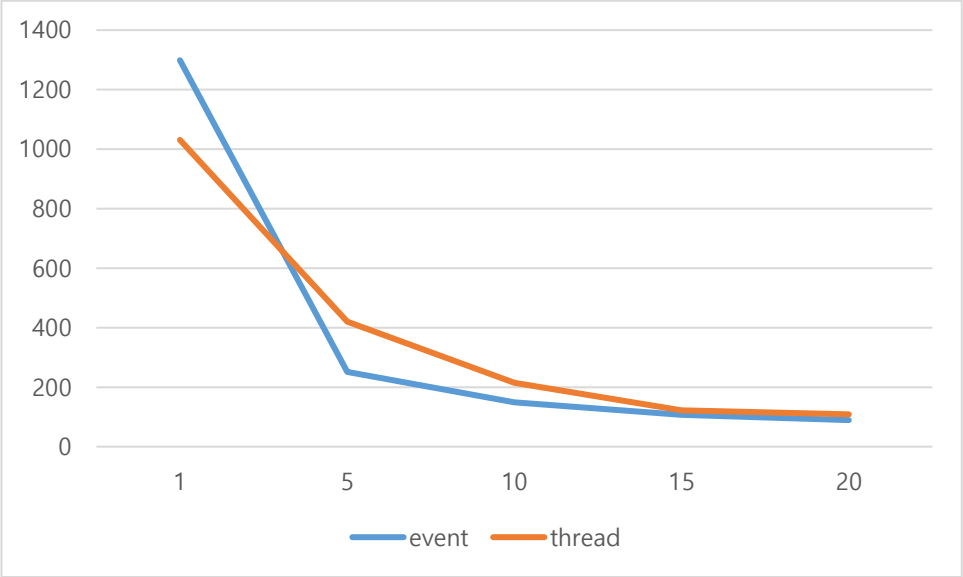
2) 워크로드에 따른 분석

1) buy or sell:

초기 stock 10000개씩 (stock 가짓수는 5개)

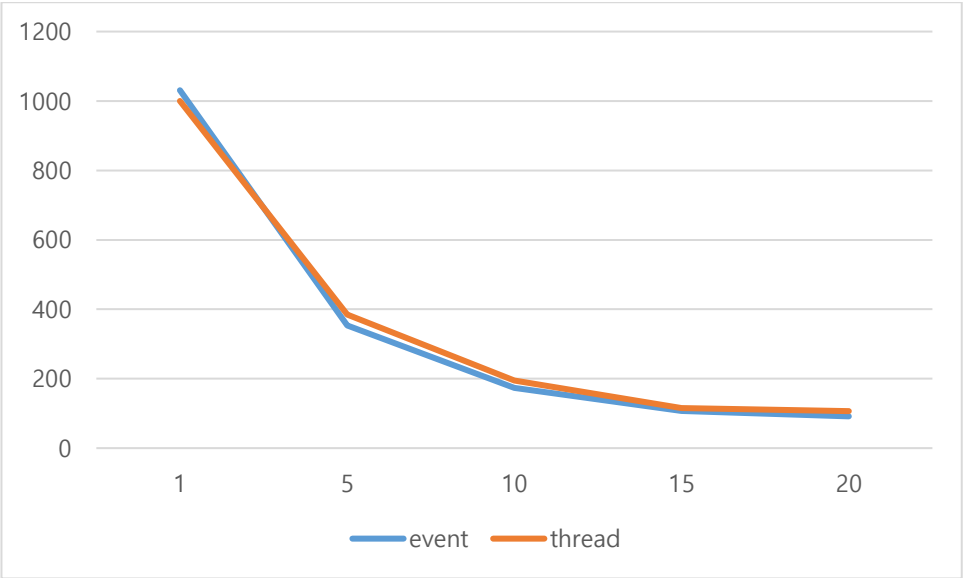
이는 buy 에서 개수가 모자라서 사지 않는 경우에는 thread의 세마포

어가 사용되지 않으므로 overhead가 달라지는데, 이를 막고 최대한 여러 번 수행에서 동일한 결과를 내기 위함이다.



process의 개수가 1인 경우에는 event가 thread보다 동시 처리율이 높지만, 그 이외의 경우에는 thread가 더 높다. 이는 1)에서 언급했던 text file update가 이유가 될 수 있다.

2) show



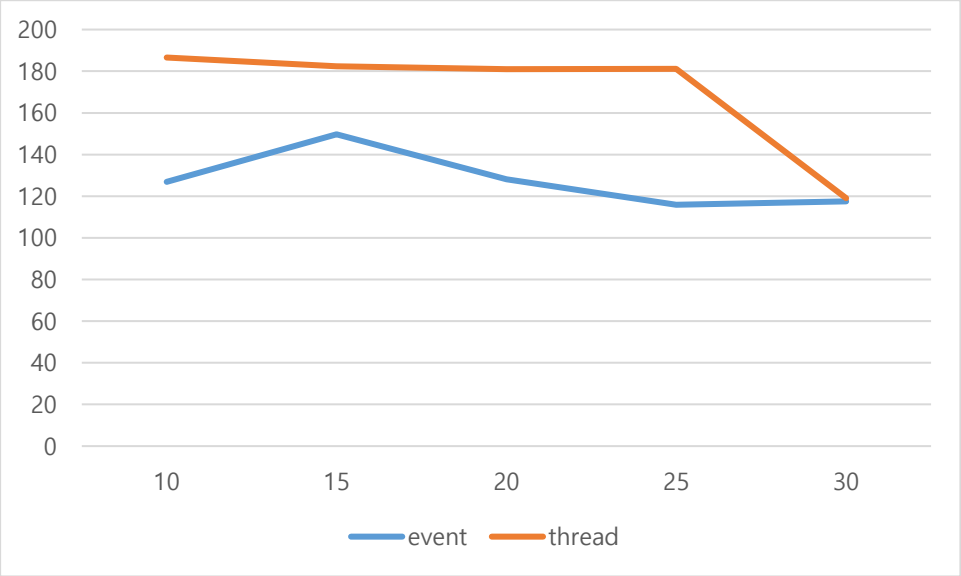
x축은 child process개수, y축은 동시처리율이다. stock의 가짓수는 10개이다. process의 개수가 증가할 수록 동시처리율이 낮아진다. event와 thread가 둘다 유사하게 나타난다.

process가 증가할수록 동시처리율이 낮아지는 이유는 1)확장성 에서 설명한 것과 같다.

3) 섞어서 요청: 확장성(1)과 같다

3) stock 개수에 따른 분석

process 개수는 10개로 고정, stock 가짓수를 변화시켰다. 각 stock의 개수는 10000개이다. x축은 stock의 가짓수, y축은 동시처리율이다.



thread의 동시 처리율이 더 높고, event에서는 stock의 가짓수에 따라 큰 변화가 나타나지 않는 것을 확인할 수 있다. 따라서 깊이에 따라 stock을 탐색하는 과정에서의 시간소모는 변화가 크지 않다는 것을 유추할 수 있다. thread는 탐색하는 과정에서 세마포어가 사용되기 때문에 동시처리율이 감소한다.