System Programming Project 5

담당 교수 : 김 영 재

이름 : 장 수 길

학번 : 20161634

1. 개발 목표

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

본 프로젝트는 concurrent한 주식 서버를 만드는 것에 목표가 있다.
일반적인 주식 서버는 쿼리를 처리할 때 쿼리들이 결국 하나의 줄을 이루어서 serialize된 처리가 이루어지는 반면, concurrent한 주식 서버는 각 client들과 동시다동시다발 상호작용하는 것이 가능하다.

이번 과제는 3가지의 concurrent server를 구현하는 방법인 process-based, thread-based, event-based 중 thread-based와 event-based 방법을 사용하여 concurrency를 지원하는 주식 서버를 만들어보고, 각 방법에 대해서 성능을 분석하는 것을 목표한다.

2. 개발 범위 및 내용

A. 개발 범위

- 아래 항목을 구현했을 때의 결과를 간략히 서술

1. select

Select함수는 event-based concurrent 서버를 위해서 가장 중요하다고 볼 수 있는 기능이다.
select를 구현하는 것을 통해서 어떤 descriptor가 준비되어있는지 확인하는 것이 가능하므로 이와 같은 I/O multiplexing을 통해서 각각의 디스크립터를 일종의 유한 기계 장치 (finite state machine)으로 취급하여 상태에 따라서 동작을 결정해주는것이 가능했다. 즉, 연결 요청을 요구하는 descriptor와 입력에 대한 출력을 요구하는 descriptor들에 대해 효과적인 처리가 가능했다.

2. pthread

Pthread 함수를 구현한 결과는 단일 프로그램으로 구성된 서버가 여러 개의 클라이언트들의 요청을 동시다발적으로 수락할 수 있도록 만드는 것이다. 여러 개의 클라이언트로부터 요청되는 사항을 중앙 서버가 context switch를 통해서 적절한 출력값을 내보내주는 것을 통해서 concurrent 주식 서버를 구현한다.

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**

  Select 함수를 사용하여 descriptor들에 대해서 새로운 연결 요청과 존재하는 클라이언트에 대해서 읽기 가능한 상태를 검출하였다. 그부분은 main함수에 나타나 있는데, 활성화된 클라이언트들을 관리하는 pool 구조체에 select함수를 적용하는것을 통해서 앞서 서술한 두가지 활동을 감지하여 해당되는 클라이언트들을 확인해준다.
  이후 if문을 통해서 각각 해당하는 활동 (연결 요청 / 출력 신청) 을 개시해준다.

  주식 서버의 경우 세가지 입력이 존재할 수 있는데, show, buy, sell이다. Show의 경우 단순히 이진트리의 내용을 buf에 연결하여 전달해주었고, buy 와 sell의 경우 in-memory 이진트리의 내용을 수정시켜줄 필요가 존재하였다.

  ✓ **stock info에 대한 file contents를 memory로 올린 방법 설명**

  stock info에 대한 file contents는 txt파일로 저장되어 있다.
  따라서 EOF를 만나기 전까지 stock.txt를 훑으면서 해당하는 ID를 노드로 삼는 이진 탐색 트리를 구현하였다.

- **pthread**

  ✓ **pthread로 구현한 부분에 대해서 간략히 설명**

  pthread를 통해서 concurrent server을 구현하기 위해서 여러 개의 쓰레드들을 먼저 분기해준다. 이 쓰레드들은 descriptor을 통해서 sbuf라는 구조체에 의해 관리되는데, 새롭게 연결 요청을 신청하는 클라이언트 들에 대해서 하나씩 descriptor(쓰레드와 간접적으로 연결된) 들을 할당해주고 연결된 descriptor에 변동이 있을 때 감지하여 해당하는 쓰레드에서 결과값을 처리해준다.

  마찬가지로 show, buy, sell 세가지 입력 옵션에 대해서 show의 경우 단순히 buf를 전달하는 것을 통해 결과값을 나타내었고 buy, sell의 경우 이진트리 내부 값을 수정하는것이기 때문에 각 옵션에 해당하는 처리를 해주었다. 다만 이 경우 주의해야할 몇가지 concurrency issue들이 존재하였다. 이에 대해서는 C부분에서 자세히 설명하도록 하겠다.

C. 개발 방법

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

공통 이진 트리

두 방법 모드 in-memory 자료구조를 유지해야 했기 때문에 이진 탐색트리를 활용하였다.
각 노드에 ID, 가격, 남은 주식 개수를 보유하도록 구성한 구조체를 선언하였다.
각 노드의 ID, 즉 주식의 ID에 대한 정보를 저장하고 좌측 자식 노드가 부모 노드보다 작은 값을 갖고 우측 자식 노드가 부모 노드보다 큰 값을 갖는 조건을 만족하도록 이진 트리를 구성하였다.

Select

Pool 구조체를 이용하여 descriptor들을 관리해주었다.
준비된 descriptor와 전체 descriptor, 활성화된 descriptor들을 관리해주는 복합적 구조체를 사용하는 것을 통해서 Select함수를 적용하는 것을 통해 변화된 descriptor값들에 대해서 연결 요청의 경우 새로운 클라이언트를 연결시켜주고, 출력 요청에 대해서는 pool 내부의 활성화된 클라이언트 중 준비된 descriptor를 찾아서 출력시켜주었다.

Pthread

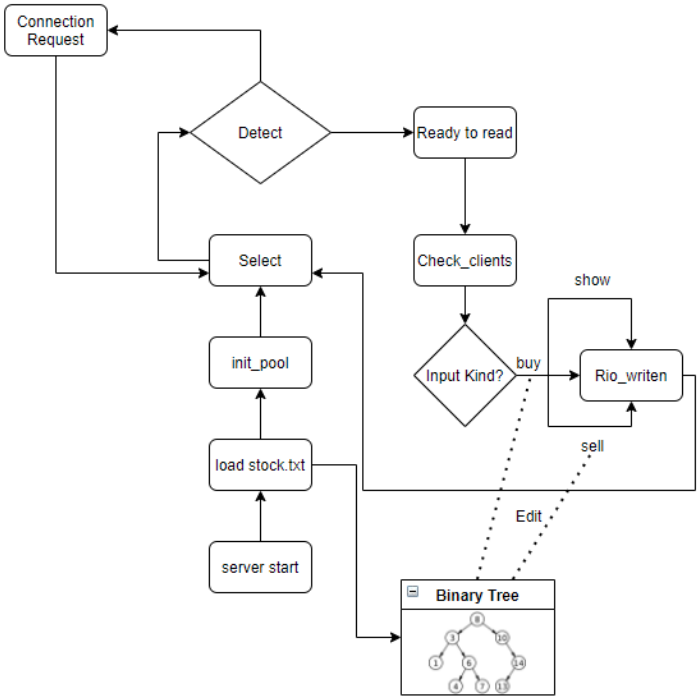
Concurrency의 결과로 일어날 수 있는 공유 변수에 대한 문제들은 대표적으로 Producer-Consumer problem과 Readers Writers problem이 존재한다. 이 문제들을 해결해주기 위해서 sbuf package를 이용한 간접적 descriptor 통제를 사용하였다. 사용 가능한 슬롯들을 producer들은 생성해주고, consumer은 생성된 슬롯을 사용하는 것을 통해서 동일한 변수 또는 자원에 대해서 잘못된 접근을 회피할 수 있었다. Readers-Writers problem의 경우, 읽기 또는 쓰기 둘 중하나에 우선권을 주는 방식을 통해서 해결이 가능하였다.

3. 구현 결과

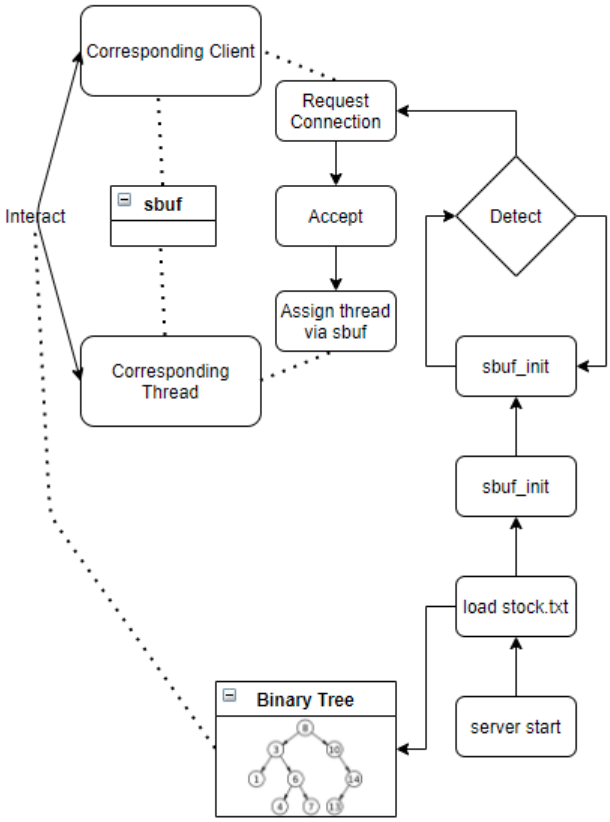
A. Flow Chart

- **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**

   구현 중 발생한 몇가지 문제로는 show 명령어 처리에 대한 문제였다. Rio_readlineb함수가 개행을 만날때까지 문자를 읽어들이기 때문에 실제로 개행을 출력해야하는 show명령어의 경우 중간에 끊어지는 등 예상하지 못한 일들이 발생하였다. 이에 대한 해결방안으로 show를 요청했을 경우, 서버에서 돌려주는 return message에는 앞에 show, 그리고 주식의 총 개수를 넣어주고 뒤에 주식정보들을 개행 없이 넣어주었다.
   클라이언트에서는 먼저 sscanf 함수를 통해서 앞에 show 라는 문자열이 존재하는지 선제적으로 파악한 후, 만약 존재할 경우 주식의 총 개수를 읽어들이고, 그만큼 3개씩 끊어서 출력하는 것을 반복하여 show 쿼리에 대한 적절한 결과를 출력하였다.
   추가적인 문제로는 Rio_readlineb함수가 계속해서 end of connection을 발생시켜 서버가 segmentation fault를 일으키는 문제가 존재하였다. 이경우는 디버깅 결과 multiclient 파일에서 define된 주식 ID의 범위를 정확히 설정하지 않았기 때문에 일어나는 문제로, 이진트리에 존재하지 않는 ID에 대한 값을 요청하는 것으로 인해 발생한 에러로 확인되었다.

2. **pthread**

   Pthread의 경우 또한 마찬가지로 show 쿼리에 대한 문제가 존재하였다. event-based 에서 언급한 내용과 같은 방식으로 해결하였다.
   두번째 문제는 여러 개의 클라이언트가 접근할 경우 값이 정확하게 출력되지 않거나 프로그램이 터지는 문제가 있었는데, 이는 다수의 쓰레드가 같은 값에 대해서 동시에 접근할 경우 critical 영역을 침범하게 되어 발생하는 문제인것을 확인하였다. 즉, 여러 쓰레드들이 공유하는 resource인 이진트리의 값을 동시에 읽는 것 (show)는 문제없지만, read, sell 과 같은 값을 직접적으로 변경시키는 경우에는 에러를 발생시켰던 것이었다.
   이에 대한 해결방안으로 P(&mutex), V(&mutex)를 활용해 데이터를 변화시키는 부분만 한정시키는 방법을 통해 최저한의 대기시간과 효과적인 데이터 관리를 완성시켰다.

마지막으로 multiclient를 실행하던 중 간혈적으로 장문의 에러 메시지 (stack drop)과 함께 프로그램이 종료되는 경우가 존재했는데, 이경우는 동일한 파일을 여러 번 close하는것으로 인해 발생한것으로 확인하였다. txt파일에 대한 업데이트를 진행하는 과정에서 발생한 문제로, fclose()를 하기 전 다시 한번 fopen을 한번씩 해주도록 설정하는 것을 통해 해결할 수 있었다.

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)**

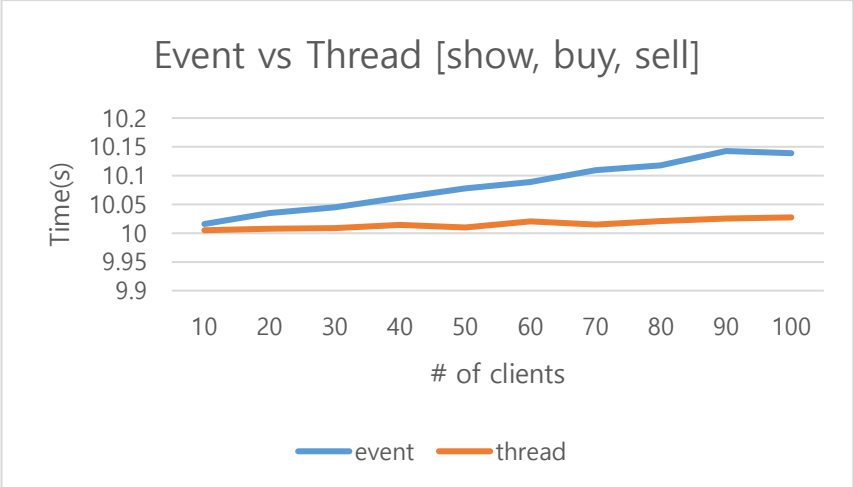
초기 예상은 다음과 같았다.
현재 사용되는 대부분의 CPU는 멀티코어를 지원하기 때문에 멀티코어를 지원하지 않는 select 보다 멀티코어를 지원하는 pthread가 더 대체적으로 빠를것이라고 생각했다.

1. [show, buy, sell 모두 혼합되어있는 명령의 경우]

실험 결과는 다음과 같았다.

	Number of Clients									
	10	20	30	40	50	60	70	80	90	100
event	10.01586	10.03488	10.04498	10.06145	10.07793	10.08856	10.10916	10.11749	10.14266	10.13863
thread	10.00498	10.00763	10.00893	10.01444	0.009692	10.02023	10.01457	10.02091	10.02567	10.02737

<실험결과>



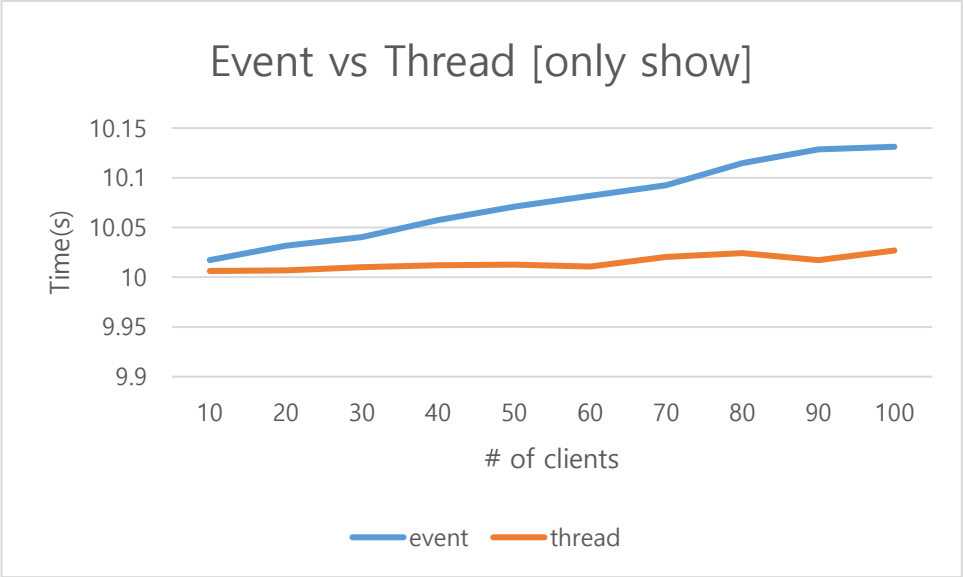
먼저 주목해야할 것은 예상과 동일하게 event-based server가 더 오래 걸리는 추세였다는 것이.
그리고 한가지 관찰한 점은 둘다 상승폭은 느리지만 클라이언트의 개수에 따라서 확실히 수행시간이 늘어나고있다는 점이고, thread-based가 더 효율적인 처리를 하고있다는 것을 볼 수 있다. 이유는 완만한 상승폭에 있다.
또 한가지는 둘 다 상승폭이 굉장히 적다는 점을 시사할 수 있다.
그래프의 y축을 더 큰 스케일로 변환할 경우, 굉장히 느린 속도로 증가한다는것을 확인할 수 있다.

2. [show만 수행하는 경우]

show만 수행하는 경우는 다음과 같은 결과를 얻을 수 있었다.

	Number of Clients									
	10	20	30	40	50	60	70	80	90	100
event	10.01722	10.03157	10.04021	10.05751	10.07093	10.08181	10.09262	10.11476	10.12839	10.13115
thread	10.00625	10.00687	10.01013	10.01186	10.0126	10.01078	10.02041	10.02423	10.01708	10.02686

<only show>



마찬가지로 유사한 양상이 나오는 것을 확인할 수 있었다. Client의 수가 늘어날수록 thread와의 처리율의 차이가 나는 것을 볼 수 있다.

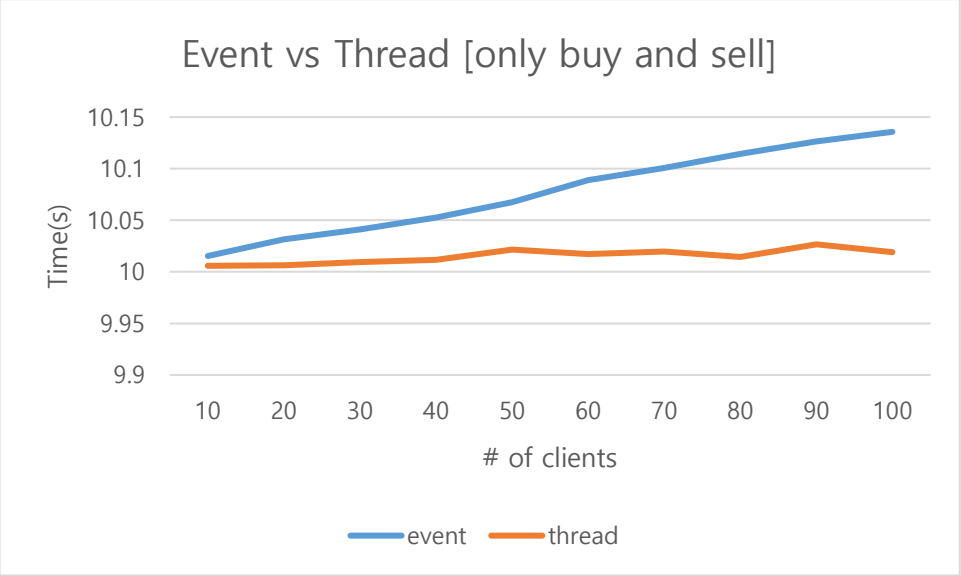
전반적으로 노드에 대해서 값의 변경이 일어나지 않기 때문에 buy 와 sell 이 들어가는 쿼리보다 적은 시간이 걸리는 것을 확인할 수 있었다.

3. [buy, sell만 수행하는경우]

buy와 sell만 수행하는 경우는 다음과 같은 결과를 얻을 수 있었다.

	Number of Clients									
	10	20	30	40	50	60	70	80	90	100
event	10.01534	10.03159	10.04112	10.0526	10.06737	10.08879	10.10077	10.11435	10.12638	10.13577
thread	10.00583	10.00638	10.0093	10.01172	10.02155	10.01708	10.0196	10.01449	10.02666	10.01916

<only buy and sell>



이 경우 또한 기조를그대로 따라가는 것을 볼 수 있다.

예상은 buy와 sell만으로 이루어진 쿼리의 경우 mutex와 binary search tree에서 값을 검색하는 오버헤드로 인해서 더 오랜 시간이 걸릴것이라고 예상했는데 실제 실험결과는 사뭇 다르게 나왔다.

Pthread함수의 경우 구현 방식을 새로운 클라이언트가 연결 신청을 할때마다 새롭게 쓰레드를 할당하는 방식으로 구현한다면 더 오랜 시간이 걸릴것으로 예상되었다.