**System Programming Project 5**

담당 교수 : 김영재 교수님

이름 : 한찬희

학번 : 20182204

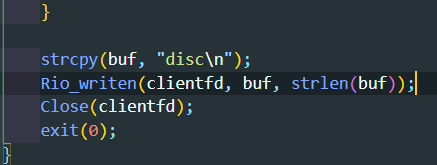
1. **개발 목표**

* **개발 목표는 여러 client 들의 동시 접속 및 서비스를 위한 Concurrent stock server을 개발함에 있다. 이번 프로젝트는 Event-Based방식과 Thread-Based 방식 두가지로 나뉘어 concurrent stock server을 개발하였다. Event-Based는 fd\_set의 array와 select함수를 사용하여 제작되었고 Thread는 PThread 계열의 함수를 사용하여 제작되었다. 또한 데드락이나 레이스 상태를 만들기 않기 위해 구조체 자료형 내부에는 sem\_t 자료형의 세마퓨어 들어가서 중간에 변수값이 바뀌는 것을 막아주었다.**

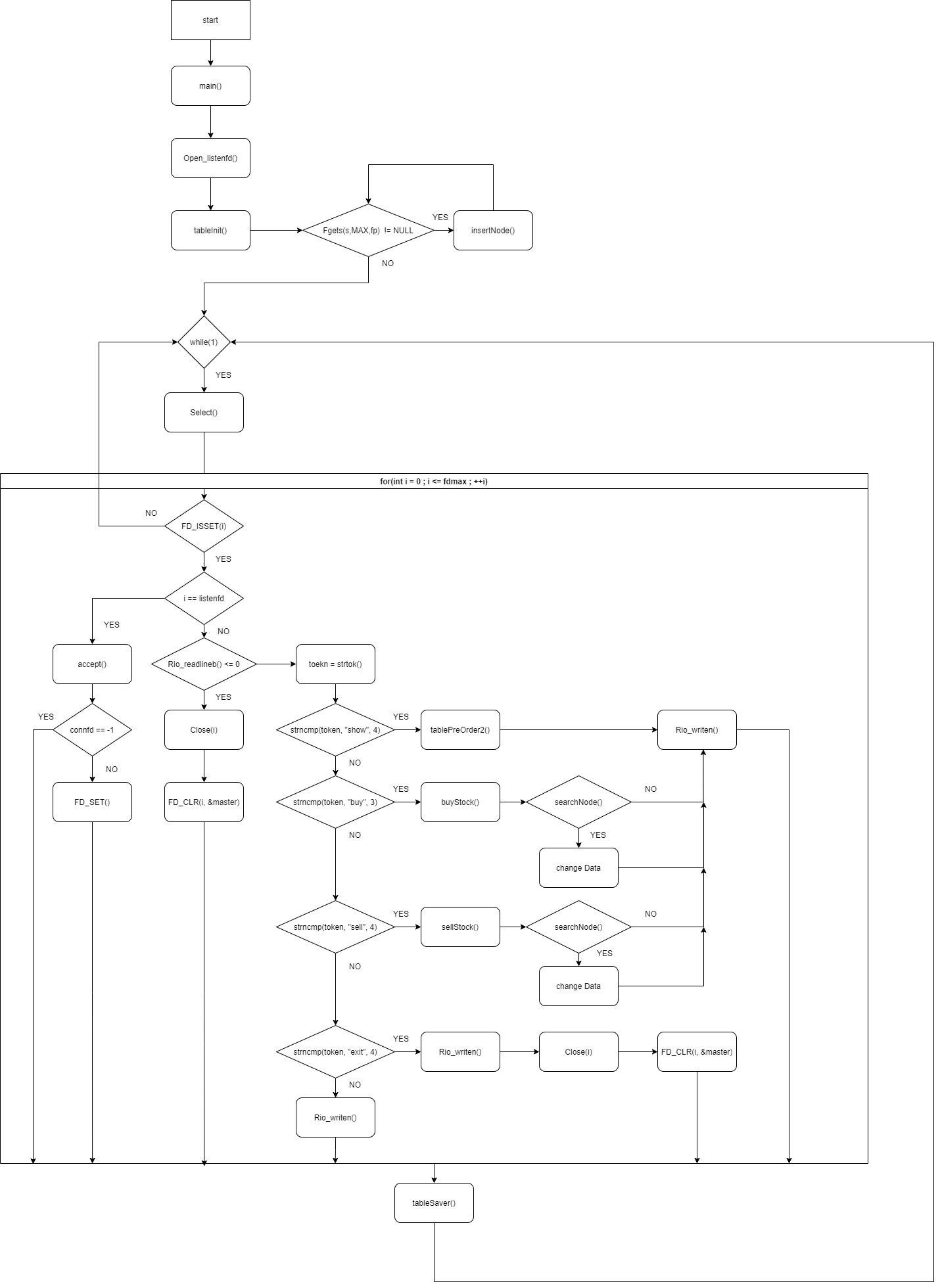
1. **개발 범위 및 내용**
   1. **개발 범위**

* **아래 항목을 구현했을 때의 결과를 간략히 서술**

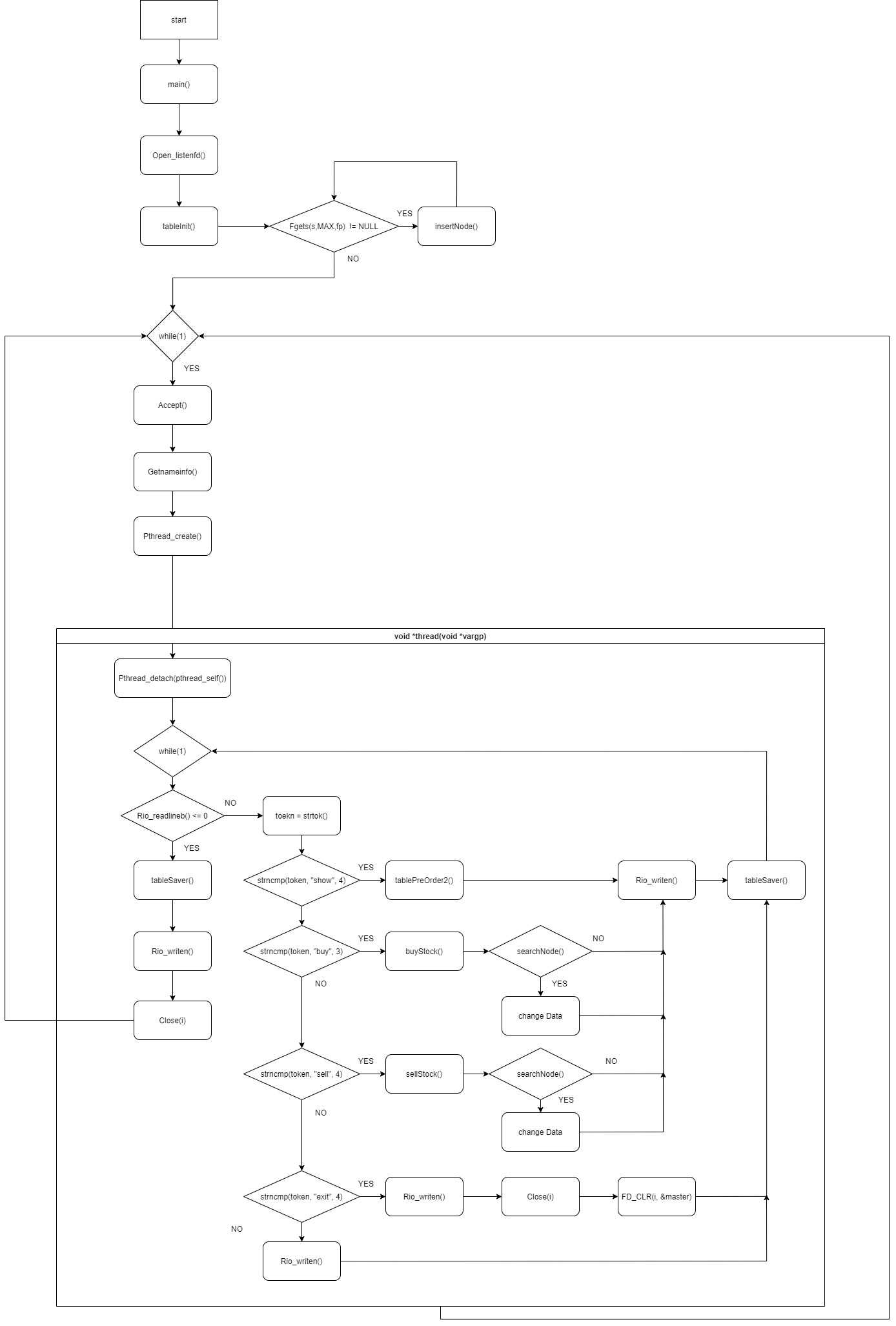
1. select
   * fd\_set 자료형의 master 소켓 집합이 event가 발생한 소켓에 대해 이벤트를 처리해준다. event가 발생하면 소켓 집합에 우선저장하고 loop가 돌면서 event 발생을 확인한다. 그리고 확인된 소켓에 대해 요청사항을 처리해준다.
2. pthread
   * client로부터 connect가 오면 PThread\_create함수로 스레드를 생성해주고 해당 스레드가 client의 요청을 처리해준다.
   1. **개발 내용**

* **아래 항목의 내용만 서술**
* **(기타 내용은 서술하지 않아도 됨. 코드 복사 붙여 넣기 금지)**
* **select**
  + select 함수로 구현한 부분에 대해서 간략히 설명
  + select 함수를 구현하기 위해 fd\_set 자료형 master 소켓 배열을 만들고 03번 소켓은 server 소켓으로 사용한다. client로부터 connect 요청이 들어오면 accept하고 해당 소켓을 master에 입력해준다. 이때 소켓의 최대 개수를 저장하기 위해 fdmax 변수를 따로 저장하고 있는다. loop를 돌면서 event가 들어온 소켓에 대해 명령어를 받아들여 처리해준다.
  + stock info에 대한 file contents를 memory로 올린 방법 설명
  + 처음 서버가 켜졌을 때 파일로부터 stock.txt에 있는 내용을 읽어들인다. 읽어들인 내용은 트리의 노드들은 전역변수 root노드가 관리하고 있는다. sell, buy, exit 명령어가 들어와 처리가 된 이후 stock.txt에 내용이 저장된다. 서버서 종료될 때 stock.txt에 내용이 저장된다.
* **pthread**
  + pthread로 구현한 부분에 대해서 간략히 설명
  + client로부터 connect 요청이 들어오면 이를 accept해준다. Pthread\_create함수를 호출하여 스레드를 새로 생성해주고 client의 accept번호를 넘기어 준다. 생성된 스레드 내부에서 담당하는 client의 요청을 처리해준다.
  1. **개발 방법**
* **B.의 개발 내용을 구현하기 위해 어느 소스코드에 어떤 요소를 추가 또는 수정할 것인지 설명. (함수, 구조체 등의 구현이나 수정을 서술)**
* select
* 1) stockserver.c
  + z
    - 주식의 데이터들을 저장하기 위해서 트리의 노드들에 대한 구조체가 정의되어 있다. 내부 변수로는 ID, readcnt, price, left\_stock, right\_stock, mutex가 존재한다. ID, readcnt, price는 주식의 아이디, 개수, 가격을 이야기 한다. left\_stock, right\_stock은 트리 노드의 자식들을 가리킨다. mutex는 데드락, 레이스 상태를 피하기 위한 세마퓨어에 필요한 부분이다.
  + int insertNode(item \*\*root, int ID, int readcnt, int price)
    - 파일을 읽어들여 데이터를 트리로 바꾸는 과정에 사용하는 함수이다. root노드를 따라 BST의 형태로 데이터가 삽입된다. 이 과정에서 노드는 초기화 되며 세마퓨어를 위해 Sem\_init함수가 사용된다.
  + item \*searchNode(item \*node, int ID)
    - ID와 동일한 노드를 찾아주는 함수이다. 만약에 해당 노드를 찾았다면 노드의 주소를 반환해준다. 반대로 찾지 못했다면 NULL을 반환해준다.
  + void proOrder(item \*root)
    - 트리를 전위순회로 돌면서 모든 노드의 정보를 출력해준다.
  + void tablepreOrder(item \*root, FILE \*fp)
    - tableSaved() 함수 호출시 사용되는 함수이다. 인자로 받은 파일포인터로 전위순회로 돌면서 모든 노드들의 데이터를 파일에 저장해준다. 이 과정에서 세마퓨어를 걸고 재귀함수가 돌아오는 과정에서 해제해준다.
  + void tablepreOrder2(item \*root, char \*ans)
    - show() 함수를 호출시 사용되는 함수이다. 인자로 받은 ans에 전위순회로 돌면서 얻은 노드의 값을 출력하고자하는 형태로 저장한다. 이때 모든 노드에 세마퓨어를 걸고 재귀함수가 돌아오는 과정에서 해제해준다.
  + item \*tableInit(void)
    - stock.txt 파일을 읽기모드로 열어서 각 한줄식 읽어들인다. 읽은 데이터를 스페이스를 기준으로 나눈다. 그리고 insertNode를 호출하여서 데이터를 트리에 저장하고 root노드를 반환해준다.
  + int tableSaved(item \*root)
    - stock.txt 파일을 쓰기모드로 연고 tablePreOrder() 함수에 파일 포인터를 전달해준다. 쓰기가 종료되었다면 1을 반환해준다.
  + int buyStock(item \*root, int ID, int readcnt)
    - root로부터 입력 받은 ID와 동일한 노드를 searchNode 함수를 호출하여 반환받는다. 해당 노드의 세마포어를 걸고 readcnt 요청된 주식 수보다 남은 주식의 수가 적으면 세마포어를 풀고 0을 반환한다. 충분한 주식 수가 있다면 노드의 주식수를 감소하고 세마포어를 풀고 1을 반환한다.
  + int sekkStock(item \*root, int ID< int readcnt)
    - root로부터 입력 받은 ID와 동일한 노드를 searchNode 함수를 호출하여 반환받는다. 해당 노드의 세마포어를 걸고 readcnt만큼 주식 수를 증가시킨다. 세마포어를 풀고 1을 반환해준다.
  + int main(int argc, char \*\*argv)
    - Open\_listenfd(argv[1])을 호출하여 입력받은 포트로 소켓을 제작하고 소켓의 번호를 int listenfd로 넘기어 준다.
    - tableInit() 함수를 호출하여 stock.txt 파일을 읽어들이고 item \*root 포인터 변수로 제작된 트리의 루트 노트를 건네준다.
    - FD\_SET(listenfd, &master)함수를 호출하여 fd\_set master array에 listenfd 를 셋팅해준다. 0..2번은 STDIN,STDOUT,STDERR가 이미 사용하기 때문에 3번에 셋팅된다.
    - 반복문을 돌면서 다음의 역할을 수행한다.
    - fd\_set read\_fds = master하여 master fd\_set을 복사해 놓는다. Select(fdmax+1, &read\_fds, NULL, NULL, NULL) 함수를 호출하여 이벤트가 발생한 array를 찾아낸다. 이때 master가 아닌 read\_fds를 사용한 이유는 Select함수가 fd\_set array의 상태를 바꾸기 때문이다.
    - fdmax까지 fd\_set array를 모두 탐색하며 다음의 역할을 수행한다.
    - FD\_ISSET(I, &read\_fds) read\_fds에서 i번째 array에 event가 발생한지 체크한다. 이벤트가 있는 경우에만 처리해주고 그 외의 경우는 그 다음 루프를 돈다.
    - 만약 I == listenfd인 경우 3번 서버 소켓으로 connect가 들어온 것이기 때문에 해당 요청을 connfd = accept(listenfd, (struct sockaddr \*)&clientaddr, &clientlen) 으로 요청을 받아들이다. connfd가 -1이라는 것은 에러가 발생한 것이므로 perror을 띄어준다. connfd가 -1이 아니라면 정상 connect이기 때문에 FD\_SET(connfd, &master)로 해당 요청을 master fd\_set에 저장한다. 그리고 connfd가 fdmax 값보다 큰 경우 이를 fdmax에 저장한다. server에는 해당 호스트의 이름과 포트번호를 출력해준다.
    - I != listenfd인 경우 rio\_t rio를 Rio\_readinitb(&rio, i)로 초기화해준다. 그리고 numbytes = Rio\_readlineb(&rio, buf, MAXLINE)로 클라이언트의 요청을 읽어들인다.
    - 읽어들인 요청의 결과 그 길이가 0이거나 음수인 경우 클라이언트 접속이 종료된 것이기에 Close(i)로 해당 소켓을 닫아주고 FD\_CLR(I, &master)로 fd\_set에서 해당 클라이언트를 지워준다.
    - 사용자의 입력이 있는 경우 문자열을 스페이스바를 기준으로 strtok를 이용해 잘라준다.
    - 사용자의 입력이 show인 경우 tablePreOrder2()함수를 호출하여 전체 노드의 정보를 ans로 받아오고 client에게 보낸다.
    - 사용자의 입력이 buy인 경우 buyStock함수로 ID, readcnt를 보내어 주식을 산다. 만약에 결과가 양수이면 [buy] success를 음수이면 Not enough left stock을 client에게 보낸다.
    - 사용자의 입력이 sell인 경우 sellStock함수로 ID, readcnt를 보내어 주식을 판다. 만약에 결과가 양수이면 [sell] success를 client에게 보낸다. 그외의 경우는 client에게 아무것도 보내지 않는다.
    - 사용자의 입력이 exit인 경우 사용자에게 exit무자열을 보낸다. 그리고 Close(i), FD\_CLR(I, &master)로 사용자 정보를 없앤다.
    - 가장 중요하게 위의 조건을 모두 마무리하고 client에게 EOF를 보낸다. 클라이언트에서는 Rio\_readlineb를 while문으로 받고 있기 때문에 EOF입력이 오기 전까지 반복문을 돌리기 때문이다. EOF를 보내고 나서 내용드을 tableSaved()함수를 사용해서 stock.txt에 저장한다.
* 2) stockclient.c
  + stockclient.c에서 유일하게 바뀐 점은 Rio\_readlineb를 하는 부분이 while문의 조건문으로 들어갔다는 점 그리고 받은 문자열에 EOF, exit가 있는가로 while문 탈출이 결정된다는 점이다. 이는 Pthread에서도 동일하다.
  + 
* Pthread
* 1) stockserver.c
  + Pthread에서 사용되는 함수는 모두 Event방식과 모두 동일하다. 유일한 차이점은 Thread함수가 추가되었다는 점과 트리 root변수가 volatile의 전역변수로 선언되었다는 점이다. 따라서 Event함수와 다른점만 서술한다.
  + int main(int argc, char \*\*argv)
    - listenfd = Open\_listenfd(argv[1])로 서버 소켓을 받아온다.
    - root = tableInit()으로 stock.txt파일을 읽어들여 트리를 만든다.
    - while(1)문 안에서 사용자의 connect 요청을 기다린다.
    - 사용자로부터 요청이 들어오면 connfd = Accept(listenfd, (SA \*)&clientaddr, &clientlen) 함수를 호출해 accept한다.
    - Getnameinfo함수롤 호출하여 client의 정보를 받아오고 server에 띄어준다.
    - Pthread\_create(&tid, NULL, thread, &connfd)함수를 호출하여 스레드를 만들고 client의 소켓 번호를 thread함수로 넘겨준다. 이 시점에서 서버에는 새로운 스레드가 생성된다.
  + void \*thread(void \*vargp)
    - Pthread\_detach(pthread\_self()) 함수로 스레드가 해제될 때 리소스를 할당 해제해준다.
    - int fd = \*((int \*)vargp)함수로 연결된 client 소켓 정보를 넘겨 받는다.
    - while(1) 문 안에서 함수가 작동한다.
    - rio\_t rio를 Rio\_readinitb(&rio, fd)로 초기화해준다. 그리고 numbytes = Rio\_readlineb(&rio, buf, MAXLINE)로 클라이언트의 요청을 읽어들인다.
    - 읽어들인 요청의 결과 그 길이가 0이거나 음수인 경우 클라이언트 접속이 종료된 것이기에 Close(fd)로 해당 소켓을 닫아주고 FD\_CLR(fd, &master)로 fd\_set에서 해당 클라이언트를 지워준다. 그리고 tableSaved(root) 함수를 호출하여 데이터를 stock.txt에 저장한다.
    - 사용자의 입력이 있는 경우 문자열을 스페이스바를 기준으로 strtok를 이용해 잘라준다.
    - 사용자의 입력이 show인 경우 tablePreOrder2()함수를 호출하여 전체 노드의 정보를 ans로 받아오고 client에게 보낸다.
    - 사용자의 입력이 buy인 경우 buyStock함수로 ID, readcnt를 보내어 주식을 산다. 만약에 결과가 양수이면 [buy] success를 음수이면 Not enough left stock을 client에게 보낸다.
    - 사용자의 입력이 sell인 경우 sellStock함수로 ID, readcnt를 보내어 주식을 판다. 만약에 결과가 양수이면 [sell] success를 client에게 보낸다. 그외의 경우는 client에게 아무것도 보내지 않는다.
    - 사용자의 입력이 exit인 경우 사용자에게 exit문자열을 보낸다. 그리고 Close(fd), FD\_CLR(fd, &master)로 사용자 정보를 없앤다. tableSaved(root) 함수를 호출해서 stock.txt에 데이터들을 저장한다.
    - 사용자의 입력이 disc인 경우 tableSaved한다음 while문을 break한다.
    - 가장 중요하게 위의 조건을 모두 마무리하고 client에게 EOF를 보낸다. 클라이언트에서는 Rio\_readlineb를 while문으로 받고 있기 때문에 EOF입력이 오기 전까지 반복문을 돌리기 때문이다. EOF를 보내고 나서 내용드을 tableSaved()함수를 사용해서 stock.txt에 저장한다.
    - 종료된 스레드는 main으로 돌아간다.
  + 2) stockclient.c
    - client 프로세스가 종료 바로직전에 disc를 명령어를 서버로 보낸다.
    - 

1. **구현 결과**
   1. **Flow Chart**
2. **select**



1. **pthread**



* 1. **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**
   1. typedef struct s\_item 구조체는 각 주식 노드들을 관리한다. 이때 내부에 sem\_t mutex를 넣어 데이터 접근시 데드락이나 레이스 상태가 되는 것을 막는다. 트리는 이진트리 구조를 띄며 모든 노드들은 동적할당 받는다.
   2. event-based방식이기 때문에 fd\_set master 변수가 이벤트들을 관리한다. client로부터 온 connect연결부터 명령어 입력까지 이벤트가 발생하였음을 fd\_set에 저장하여 알게된다. 그리고 현재 입력된 최대 값만큼 돌면서 이벤트가 발생한 곳을 확인하고 로직에 따라 이벤트를 처리한다.
   3. 관련 모든 함수들은 반드시 세마포어를 걸고 시작한다. 동시에 끝나는 시점에서는 세마포어를 해제한다. 이를 통하여 임계영역에 들어가는 것을 막을 수 있다.
   4. 이슈 : Rio\_readlineb error: Connection reset by peer. 이미 끊어진 소켓으로부터 데이터를 읽어들이는 과정에서 문제가 생기었다. 이 문제를 해결하기 위해서 Close함수가 나올 수 있는 경우를 rio\_readlineb <= 0인 경우와 disc명령어, exit 명령어가 온 경우로 한정하였다.
2. **pthread**
   1. typedef struct s\_item 구조체는 각 주식 노드들을 관리한다. 이때 내부에 sem\_t mutex를 넣어 데이터 접근시 데드락이나 레이스 상태가 되는 것을 막는다. 트리는 이진트리 구조를 띄며 모든 노드들은 동적할당 받는다. 이 부분은 event-based와 동일하다
   2. 사용자로부터 accept가 온 경우 이에 대한 반응으로 스레드를 만든다. 만들때는 Pthread\_create함수를 사용하여서 스레드를 제작할 수 있다. 이렇게 되면 해당 client에 대해서는 하나의 thread가 모든 것을 관리하게 된다. 또한 thread가 종료시 리소스를 해제해주지 않으면 메모리 누수가 나는 문제가 발생하였다. 이를 해결하고자 Pthread\_detach(pthread\_self())함수를 호출하여 해결하였다.
   3. 이슈 Bad file descriptor : 이미 종료된 스레드를 다시 종료해서 생기는 문제였다. client가 한번에 오면 시간차 문제로 인하여 스레드를 생성하지 못하는 경우가 생기었다. 이는 기존에 accept함수 호출 후 출력 한 번 하고 thread를 생성하는 방식으로 생기는 문제였다. 이를 해결하기 위해 accept함수 바로 뒤에 Pthread\_create 함수를 호출하여 해결할 수 있었다.
   4. 이슈 : Rio\_readlineb error: Connection reset by peer. 이미 끊어진 소켓으로부터 데이터를 읽어들이는 과정에서 문제가 생기었다. 이 문제를 해결하기 위해서 Close함수가 나올 수 있는 경우를 rio\_readlineb <= 0인 경우와 disc명령어, exit 명령어가 온 경우로 한정하였다.
   5. **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)**
* **테스트 환경 : n(100~1000)명의 사람이 10개의 명령을 보내고 결과를 받는다.**
* **실혐결과 : Event는 선형으로 증가하였다. 반면 Thread는 정체되는 구간도 있고 극적인 변화는 자주 나오지 않아 보다 좋은 그래프를 보여주었다. 이는 select가 세마포어를 풀자마자 바로 데이터를 변화시킬 수 있는 pthread기반으로 생긴 점이라고 생각한다.**
* **client가 buy또는 sell을 요청하는 경우**
* **client가 show만 요청하는 경우**
* **show, buy, sell을 요청하는 경우**