

Concurrent Programming

Project II: Concurrent Stock Server

CSE4100: System Programming

Youngjae Kim (PhD)

Distributed Computing and Operating Systems Laboratory (DISCOS)

<https://discos.sogang.ac.kr>

Office: R911, E-mail: youkim@sogang.ac.kr

목차

■ 배경지식

- Network programming
- Echo 서버 프로그래밍

■ 동시 주식 서버 (Concurrent Stock Server) 설계 및 구현 (각 30점 + 보고서 10점)

- Task1: Event-driven Approach
- Task2: Thread-based Approach
- Task3: 성능 평가 및 분석

■ 제출 방법

■ 부록

프로젝트 목표

“여러 client들의 동시 접속 및 서비스를 위한 Concurrent stock server 을 구축”

■ 주식 서버

- 주식 정보를 저장하고 있고 여러 client들과 통신하여, 주식 정보 List, 판매, 구매의 동작을 수행

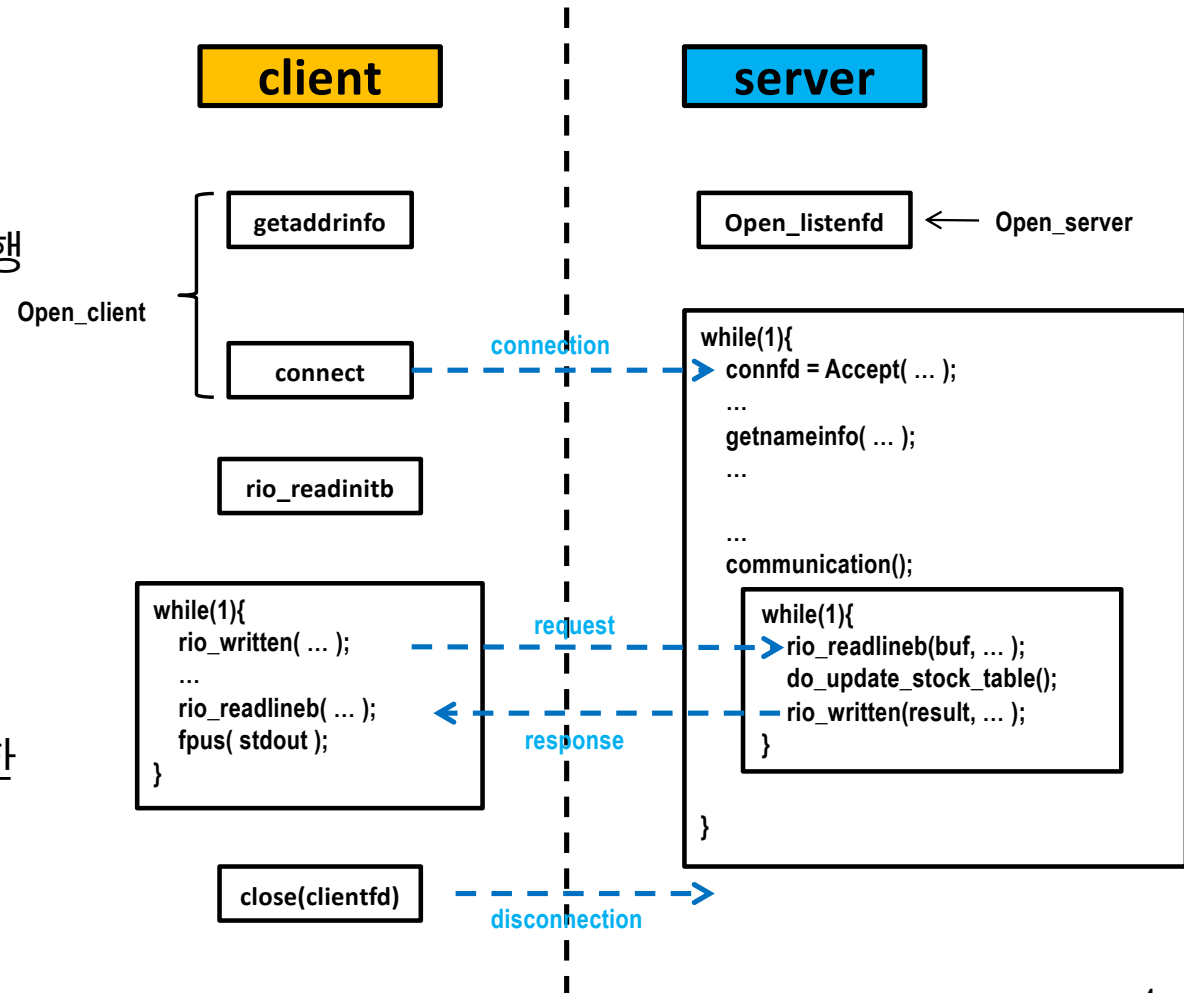
■ 주식 클라이언트

- 각 client는 server에 주식 사기, 팔기, 가격과 재고 조회 등의 요청을 함



배경지식: Network Programming

- 오른쪽 그림은 수업시간에 언급한 echoserver의 전체적인 구조임
- 배포한 압축파일을 압축해제 후 make 실행
 - stockserver, stockclient, multiclient 세 개의 실행파일 생성
 - stockserver, stockclient는 수업시간에 시연한 echoserveri, echoclient와 동일함
- stockserver, stockclient 실행 후 stockserver, multiclient 실행을 추천함
- stockserver, multiclient는 실험을 돕기 위한 multiple client 실행 프로그램임
(뒤에서 자세히 설명)



배경지식: Echo Server Program

■ echoclient 와 echoserver와 사이의 통신

client

```
gr120210196@cspro8:~/my_project$ hostname -i
172.30.10.8
gr120210196@cspro8:~/my_project$ ./echoclient 172.30.10.11 1119
hello
hello
system programming
system programming
project V
project V
```

- client는 cspro8에서 실행
- “hostname -l” → IP addr (172.30.10.8)
- ./echoclient 172.30.10.11 1119
server IP addr와 open시 사용한 port number 입력

- MSG 입력하고 request!

- Response받은 Msg 출력

server

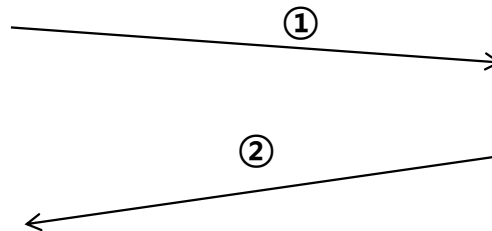
```
gr120210196@cspro:~/my_project$ hostname -i
172.30.10.11
gr120210196@cspro:~/my_project$ ./echoserveri 1119
Connected to (172.30.10.8, 34148)
server received 6 bytes
server received 19 bytes
server received 10 bytes
```

- server는 cspro에서 실행
- “hostname -l” → IP addr (172.30.10.11)
- ./echoserveri 1119

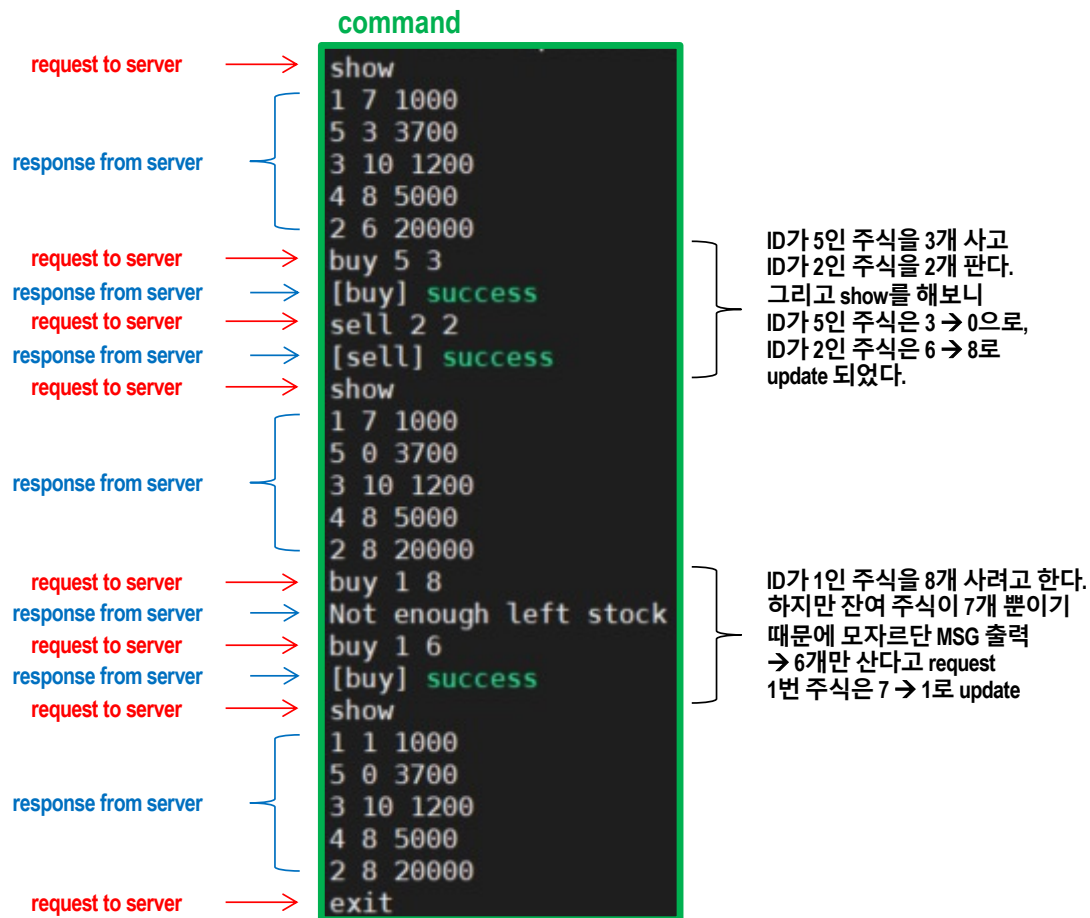
먼저 port number를 설정하고 server open

(cyber campus에 올린 개인 port사용)

- 받은 MSG의 byte수를 출력하고
해당 문자열을 다시 response
(“\n”까지 byte 수에 포함)



주식 서버 “Client” 동작 설명



클라이언트가 **show, buy, sell, exit** 네 개의 명령어를 요청함

1. **show**
: 현재 주식의 상태를 보여준다.
2. **buy [주식 ID] [살 주식 개수]**
ex. buy 5 3 : “ID가 5인 주식을 3개 사겠다.”
3. **sell [주식 ID] [팔 주식 개수]**
ex. sell 2 2 : “ID가 2인 주식을 2개 팔겠다.”
4. **exit**
: disconnection with server (주식 장 퇴장)

※ command exception handling

명령어와 특별 상황에 대해 예외 처리는 불필요

“buy 2 @”, “slel 3 1”, “sh ow”, “buy 1 3 “

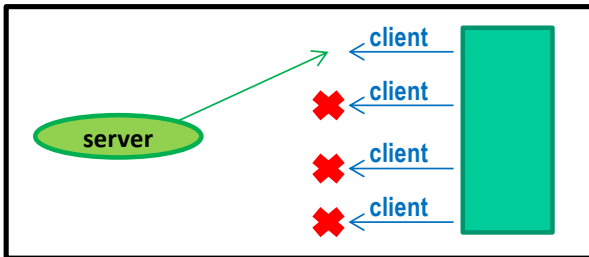
위와 같이 오타나 형식에 맞지 않는 command는 다루지 않음

Client의 보유 잔고 및 주식 정보는 고려하지 않음

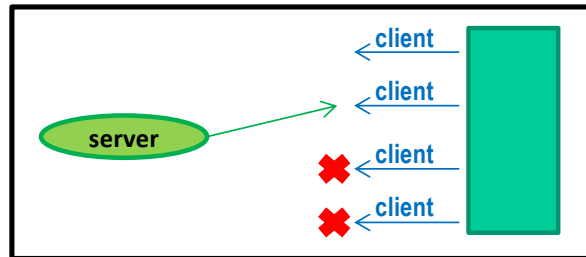
즉, Sell 명령어는 절대 실패하지 않는 것을 가정

동시 주식 서버 (Concurrent Stock Server) 필요성

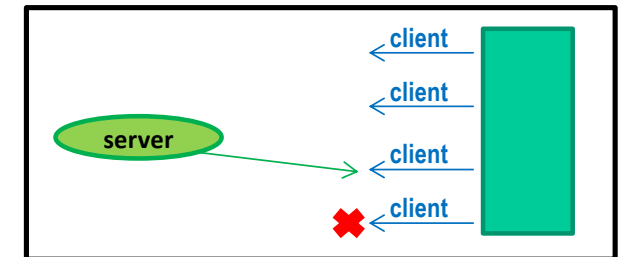
Step1



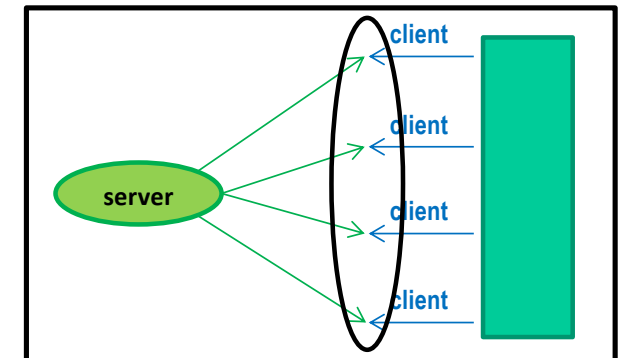
Step2



Step3



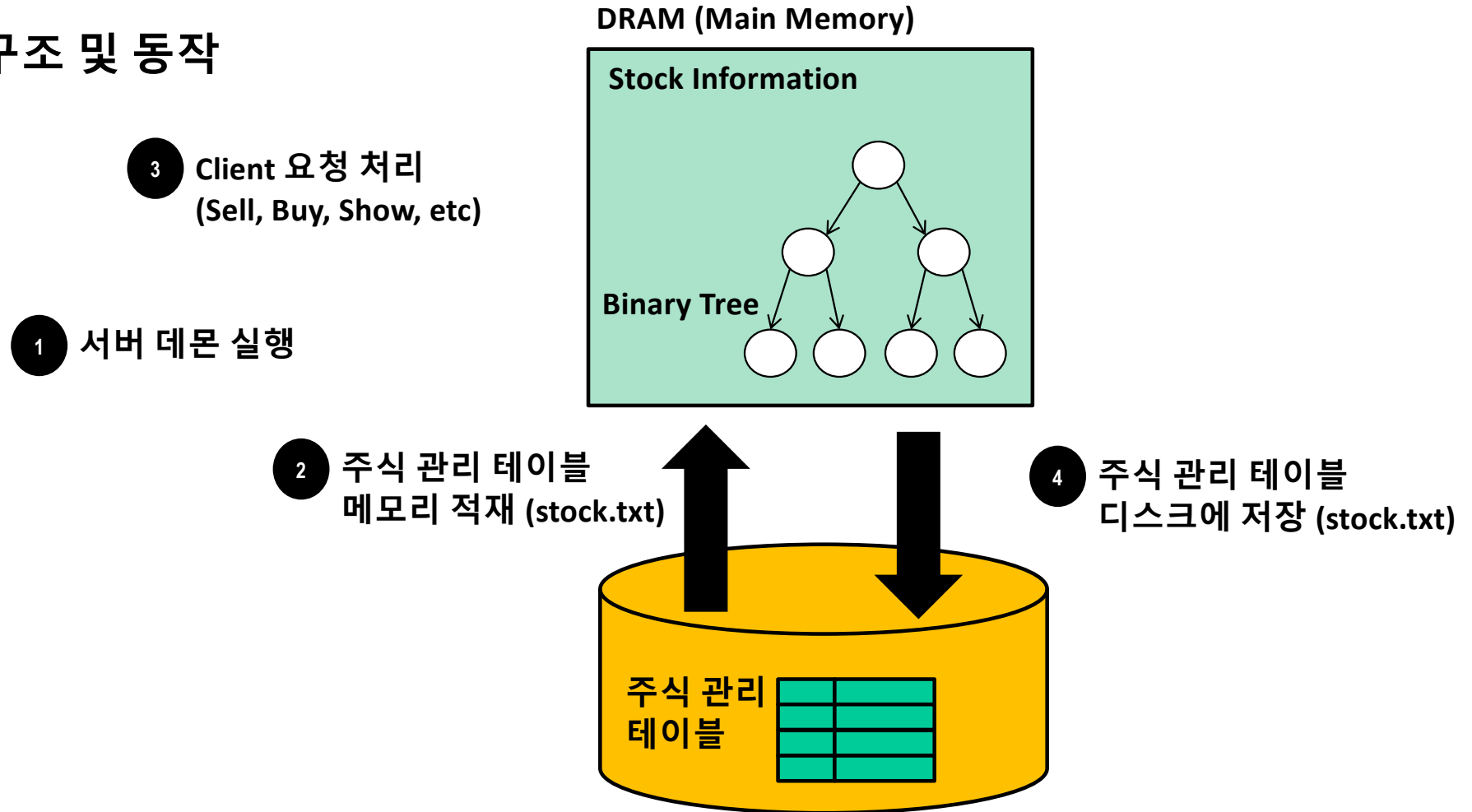
- 단일 프로세스/쓰레드 기반 stockserver는 여러 client와 connection이 불가능함
- 따라서, 앞의 실행 결과처럼 각각 client의 connection부터 요청된 request가 모두 직렬화 (serialize) 되어 순서대로 처리됨 (**No concurrency**)
- 본 프로젝트에서는 아래와 같은 concurrent 프로그래밍을 통해 동시 주식 서버를 두 가지 방식을 이용하여 설계 및 구현함
 - Event-based Approach using select()
 - Thread-based Approach using pthread library



Concurrent service

주식 서버 설계 (1)

■ 자료구조 및 동작



주식 서버 설계 (2)

■ 주식 관리

- 주식들은 stock.txt 파일로 Table 형태로 관리됨
- Table에서 각 행은 주식을 나타내고
주식은 세 가지 attribute을 가짐
 - ID, 잔여 주식, 주식 단가
- 오른쪽은 5개 주식 종목을 가진 stock.txt 예시임

주식 관리 테이블

stock.txt

```
gr120210196@cspro8:~/my_project$ cat stock.txt
1 7 1000
5 3 3700
3 10 1200
4 8 5000
2 6 20000
```

ID	잔여 주식	주식 단가
2	6	20000

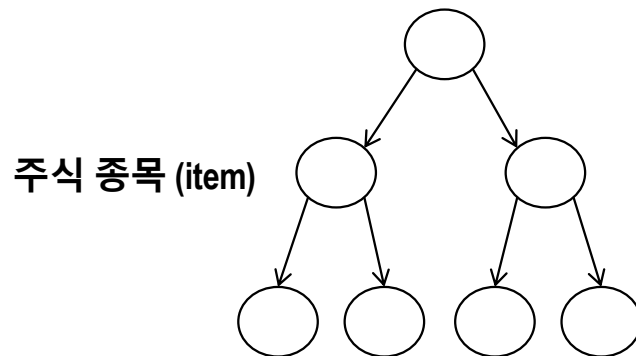
■ 가정

- 주식 단가는 변동이 없음
 - 결국 데이터에 변동되는 값은 잔여 주식 뿐!
- Client request에서 남은 주식보다 많은 주식을 요구하면 잔여 주식이 부족하다는 메시지만 출력하고 요청은 처리되지 않음
 - “Not enough left stocks”

주식 서버 설계 (3)

■ 효율적인 데이터 관리를 위해서 Binary Tree 사용

- 트리의 각 노드는 <ID, 잔여 주식, 주식 단가, mutex lock 변수> 등을 선언 (뒤 페이지 그림 참고)
- 주식 ID는 1~MAX에 random unique id로 발급



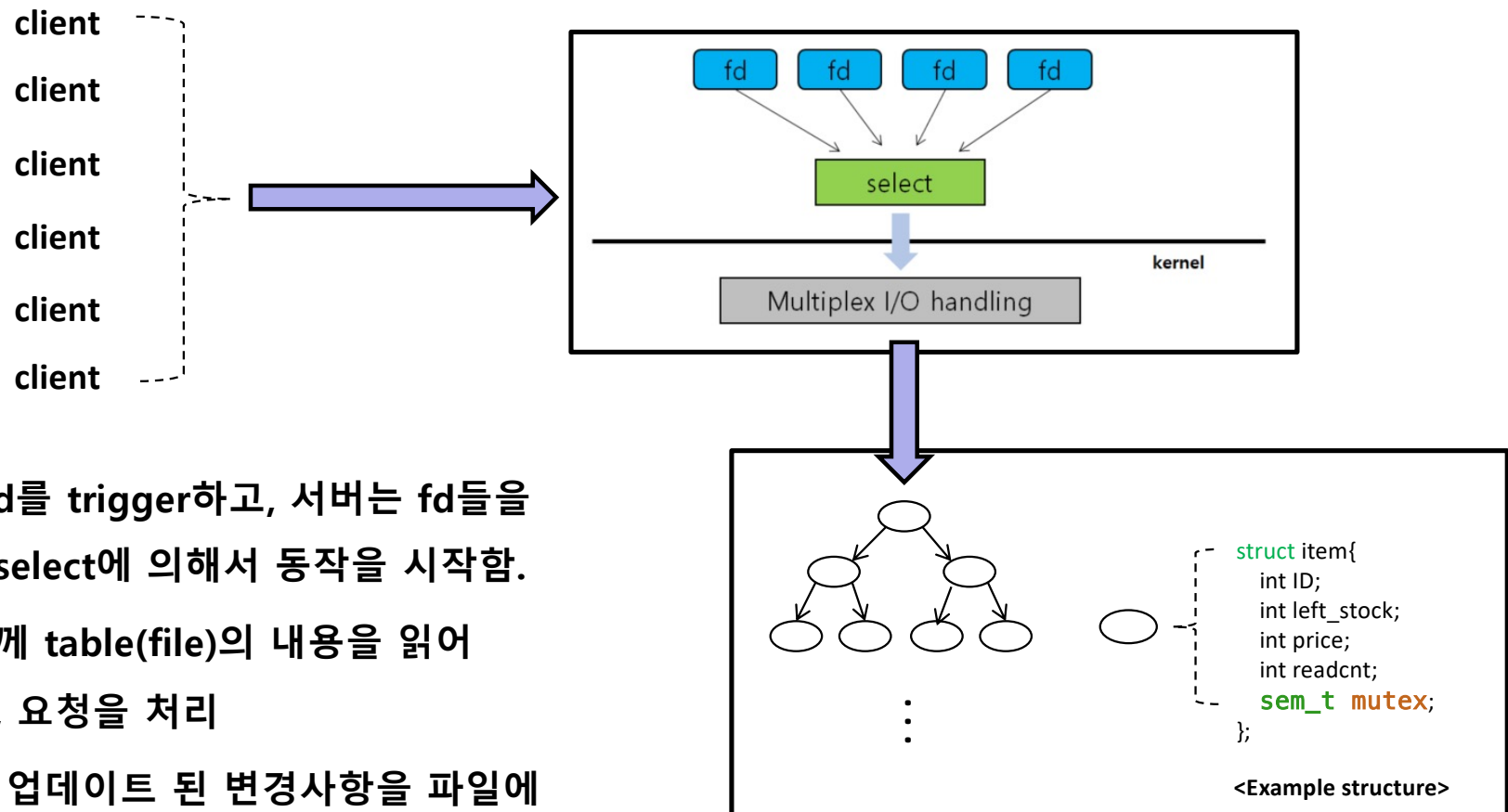
```

struct item{
    int ID;
    int left_stock;
    int price;
    int readcnt;
    sem_t mutex;
};
<예: 노드 구조>
  
```

■ Readers-Writers Problem

- 어떤 client가 i 종목 주식 읽을 때, 다른 client가 i 값을 update하게 되면 올바르게 동작 x
- Readers-writers problem solution을 고려한 노드 단위의 관리 (fine-grained locking) 필요

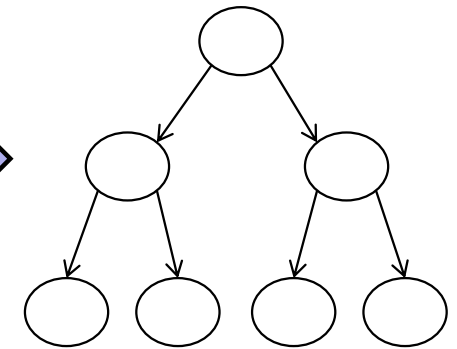
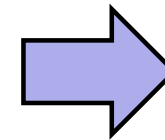
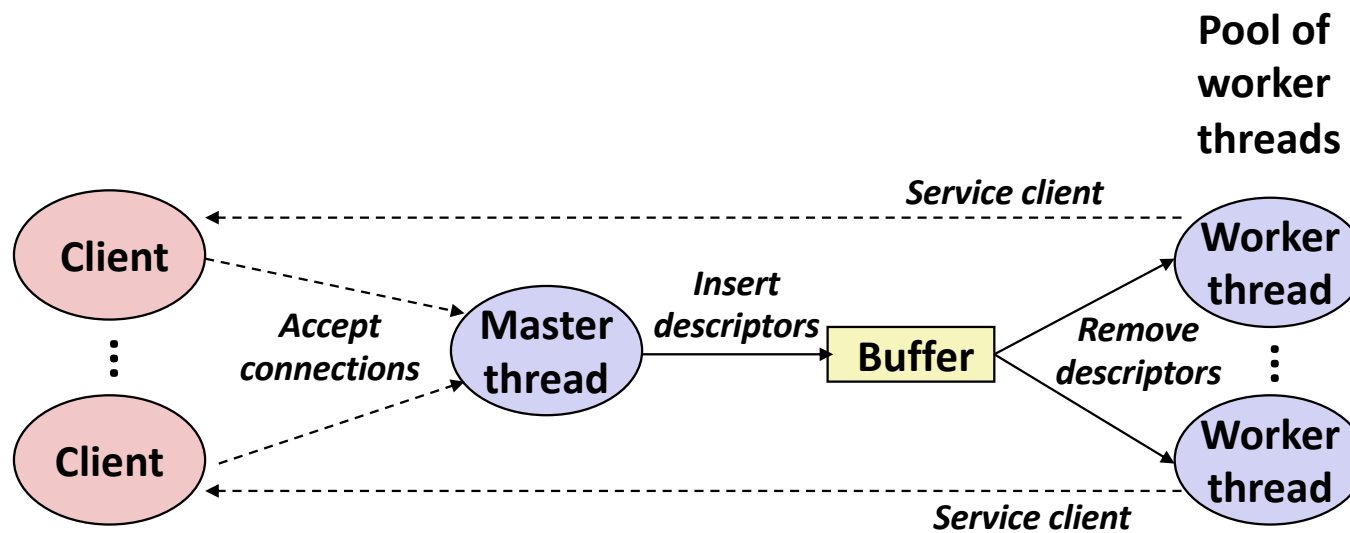
Task1: Event-driven Approach (30점)



- 각각의 client는 fd를 trigger하고, 서버는 fd들을 monitoring하던 select에 의해서 동작을 시작함.
- Server 시작과 함께 table(file)의 내용을 읽어 메모리에 적재 후, 요청을 처리
- Server의 종료 시 업데이트 된 변경사항을 파일에 정해진 형식에 따라 저장

Task2: Thread-based Approach (30점)

■ Thread-based Concurrent Stock Server



주식 종목 관리 테이블

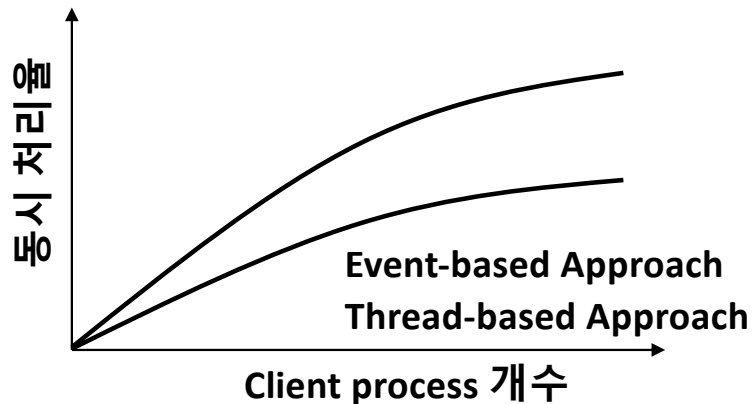
Task3: 성능 평가 및 분석 (30점)

■ 성능 평가

- Client 실행파일 내에 configuration들을 바꿔가면서 두 가지 동작 방식(event-driven, thread)의 elapse time을 측정하여 분석
 - 이 경우에 한해서는 client process의 개수를 제한하지 않고 10, 20개 이상 띄우면서 실험 가능 (Task 1-2 수행 시에는 4개 이하로 진행할 것!)
- 분석 내용은 보고서에 최대한 상세히 기재

■ 분석 방법

동시 처리율: 시간당 client 처리 요청 개수



➤ 분석 포인트

- 확장성: 각 방법에 대한 Client 개수 변화에 따른 동시 처리율 변화 분석
- 워크로드에 따른 분석: Client 요청 타입 (buy, show, sell 등)에 따른 동시 처리율 변화 분석
 - 워크로드 예제
 - 예1) 모든 client가 buy 또는 sell을 요청하는 경우
 - 예2) 모든 client가 show만 요청하는 경우
 - 예3) Client가 buy, show 등을 섞어서 요청하는 경우
- 가지 방법의 성능 또는 다양한 관점에서 비교 분석 (수업시간에 배운 내용과 일치하는지 또는 불일치하는지)
- 기타: 자유로운 분석

제출 방법

■ Documentation

- 첨부된 document에 해당 내용 삽입 (코드 붙여넣기 금지)
- 위 실험에 대한 결과 그래프와 분석 내용을 자세하게 작성 (높은 배점)
- Document는 pdf file로 변환해서 제출 (ex. document.pdf)

■ Submission

- 제출일 : ~ 5/24 23:59pm (late : 5/27 23:59pm 하루 10% 감점)

■ 주의사항

- 실행파일이나 필요 없는 파일은 제거 (감점 요인)
- Makefile에서 output executable file name은 그대로 유지 (수정 시, 감점 요인)
- [Mandatory] TA가 cspro 환경에서 채점하기 때문에 꼭 cspro에서 실행해 볼 것! (compile 오류 시 0점)
- 본인 학번 폴더(ex. 20191234)를 생성하고 그 안에 총 2개의 프로젝트 폴더와 1개의 document (pdf파일로 변환)를 포함하여 압축할 것
- 각 프로젝트 파일에는 실행하는데 필요한 모든 코드가 포함시켜야 합니다.
(execution file, binary file 제외)
- 학번 파일 상위 폴더에서 tar 압축을 해서 cyber campus 과제란에 제출
(tar -cvzf [HW2]20201234.tar.gz 20201234/)

프로젝트 주의 사항

■ Process Management에 유의해주세요

- Shell 프로젝트 수행하며 Zombie process와 background process를 남겨 놓으신 분들이 너무 많아 서버 관리에 애를 많이 먹었습니다
- `ps -aux` 명령어 혹은 `top` 명령어를 통해 백그라운드로 남아 있는 프로세스를 수시로 확인해주세요
- 백그라운드 프로세스가 죽지 않거나 서버가 이상할 경우 조교 이메일로 곧장 연락주세요 (seungjinn@sogang.ac.kr)

■ 제출 파일 양식 우측 이미지 참조

- task_3의 경우에는 보고서에 해당 내용 실행 결과 캡처와 함께 자세히 기재

```
$ pwd
~/20201234

$ cd ..
$ tree 20201234
20201234
├── document.pdf
├── task_1
│   ├── Makefile
│   ├── csapp.c
│   ├── csapp.h
│   ├── echo.c
│   ├── multiclient.c
│   ├── stock.txt
│   ├── stockclient.c
│   └── stockserver.c
└── task_2
    ├── Makefile
    ├── csapp.c
    ├── csapp.h
    ├── echo.c
    ├── multiclient.c
    ├── stock.txt
    ├── stockclient.c
    └── stockserver.c

$ tar -cvzf [HW2]20201234.tar.gz ./20201234
```

부록: 추가 설명

Multiclient 실행 파일(1)

- 터미널은 10개씩 띄우고 일일이 명령어를 작성하기는 어려움
- Multiclient 실행 파일은 이를 편하게 해 줌
 - 원하는 client 개수를 입력해주면 해당 개수만큼의 client 프로세스들을 생성하고, 각 client 프로세스는 앞에 정의된 show, buy, sell 등의 명령어를 주어진 횟수만큼 임의 순서로 실행하고 종료함
 - multiclient.c를 보면 아래와 같이 변수들이 정의되어 있음
(이 변수들은 얼마든지 수정해가면서 실험 진행 가능!)

```
#define MAX_CLIENT 100
#define ORDER_PER_CLIENT 10
#define STOCK_NUM 10
#define BUY_SELL_MAX 10
```

MAX_CLIENT 최대 만들 수 있는 client 개수
 Client를 실행할 때 4보다 작은 값을 입력할 것 (evaluation할 경우 10, 20개 사용 가능)

ORDER_PER_CLIENT client 하나당 server로 보내는 request의 개수
 show, buy, sell 중 random으로 server에 request 수행

STOCK_NUM stock file에 있는 주식 항목의 최대 개수
 (stock.txt file을 참고해서 설정할 것)

BUY_SELL_MAX 주식을 사고 팔 때 한 request당 최대 항목의 개수
 적절히 설정하여 사용 가능 (Ex. "buy 1 11" : 1번 주식을 11개 산다 (x))

Multiclient 실행 파일(2)

- 앞서 말한 #define configuration을 적절히 설정 후 실행
- stockserver를 먼저 실행하고 (./stockserver 1119), 아래와 같이 multiclient를 실행
- ./multiclient를 입력하면 사용법이 나옴 (stockclient 실행 방법과 비슷함)

```
gr120210196@cspro:~/project5$ ./multiclient
usage: ./multiclient <host> <port> <client#>
gr120210196@cspro:~/project5$
```

- ./multiclient [IP_addr] [port #] [client_num] (ex. ./multiclient 172.30.10.11 1119 4)
아래와 같이 4개의 client process가 configuration에 따라 적절한 동작을 수행
 - 실험 시에는 client process를 4개 이하로 띄워서 실험할 것!
 - zombie나 orphan이 발생하지 않도록 주의할 것!

child process
4개 생성

```
gr120210196@cspro:~/project5$ ./multiclient 172.30.10.11 1119 4
child 45130
child 45131
child 45132
sell 6 4
child 45133
show
show
show
buy 1 4
sell 6 10
```

Multiclient 실행 파일(3)

client 1 request	<pre> gri20210196@cspro:~/project5\$./stockserver 1119 Connected to (cspro.sogang.ac.kr, 54594) server received 9 bytes server received 5 bytes server received 5 bytes server received 5 bytes server received 8 bytes server received 10 bytes server received 5 bytes server received 8 bytes server received 5 bytes server received 5 bytes </pre>	<pre> gri20210196@cspro:~/project5\$./multiclient 172.30.10.11 1119 4 child 45130 child 45131 child 45132 sell 6 4 child 45133 show show show buy 1 4 sell 6 10 show </pre>
client 2 request	<pre> Connected to (cspro.sogang.ac.kr, 54596) server received 9 bytes server received 9 bytes server received 5 bytes server received 9 bytes server received 9 bytes server received 9 bytes server received 9 bytes server received 8 bytes server received 9 bytes server received 9 bytes </pre>	<pre> buy 1 1 show show sell 6 9 sell 4 8 show sell 7 7 buy 10 5 sell 6 6 sell 2 7 buy 7 9 </pre>
client 3 request	<pre> Connected to (cspro.sogang.ac.kr, 54598) server received 5 bytes server received 9 bytes server received 9 bytes server received 5 bytes server received 8 bytes server received 5 bytes server received 9 bytes server received 8 bytes server received 10 bytes server received 5 bytes </pre>	<pre> sell 5 5 buy 10 8 show sell 1 3 sell 4 4 show buy 8 5 show sell 9 8 buy 4 5 sell 10 9 </pre>
client 4 request	<pre> Connected to (cspro.sogang.ac.kr, 54600) server received 5 bytes server received 5 bytes server received 8 bytes server received 10 bytes server received 8 bytes server received 9 bytes server received 10 bytes server received 9 bytes server received 5 bytes server received 8 bytes </pre>	<pre> show show show buy 7 9 sell 10 4 buy 6 1 sell 4 5 sell 10 7 sell 2 6 show buy 6 5 </pre>

서버 구현 상세 (1)

- 직접 구현해야 하는 부분은 stock server부분 뿐!
(client 부분은 multiclient.c 그대로 사용하여 실험)
- stockserver.c에서 multiple client request를 처리 할 수 있도록 select와 pthread를 사용하여 구현
- **Recommendation** : 먼저 아래와 같이 command string을 server가 concurrency하게 판독할 수 있게 구현

```
gr120210196@cspro:~/my_project$ ./echoservt 1119
server received 8 bytes
server received 5 bytes
server received 5 bytes
server received 8 bytes
server received 10 bytes
server received 8 bytes
server received 10 bytes
```

```
gr120210196@cspro:~/project5$ ./multiclient 172.30.10.11 1119 4
child 45130
child 45131
child 45132
sell 6 4
child 45133
show
show
show
buy 1 4
sell 6 10
show
buy 1 1
```

서버 구현 상세 (2)

■ Response 양식 (오른쪽 그림 참고)

- **[buy] success** : buy request가 정상적으로 처리되었을 경우
- **[sell] success** : sell request가 정상적으로 처리되었을 경우
- **Not enough left stocks** : buy request에 대해서 잔여 주식이 부족한 경우
Recommendation : 이때는 client가 아닌 stockclient를 실행시키고 command를 하나씩 입력하고 제대로 처리되는지 확인하면서 진행

■ Persistency를 위해서 connection이 모두 종료되면 stock.txt 에 update 내용이 반영되어야 함

- stock.txt에 update 내용이 제대로 반영되었는지 확인함으로써 채점을 진행할 예정

```
show
1 7 1000
5 3 3700
3 10 1200
4 8 5000
2 6 20000
buy 5 3
[buy] success
sell 2 2
[sell] success
show
1 7 1000
5 0 3700
3 10 1200
4 8 5000
2 8 20000
buy 1 8
Not enough left stock
buy 1 6
[buy] success
show
1 1 1000
5 0 3700
3 10 1200
4 8 5000
2 8 20000
exit
```