멀티 프로세스를 활용한 소켓 통신 채팅서버 구현

VEDA 1기 이영호

목차

table of contents

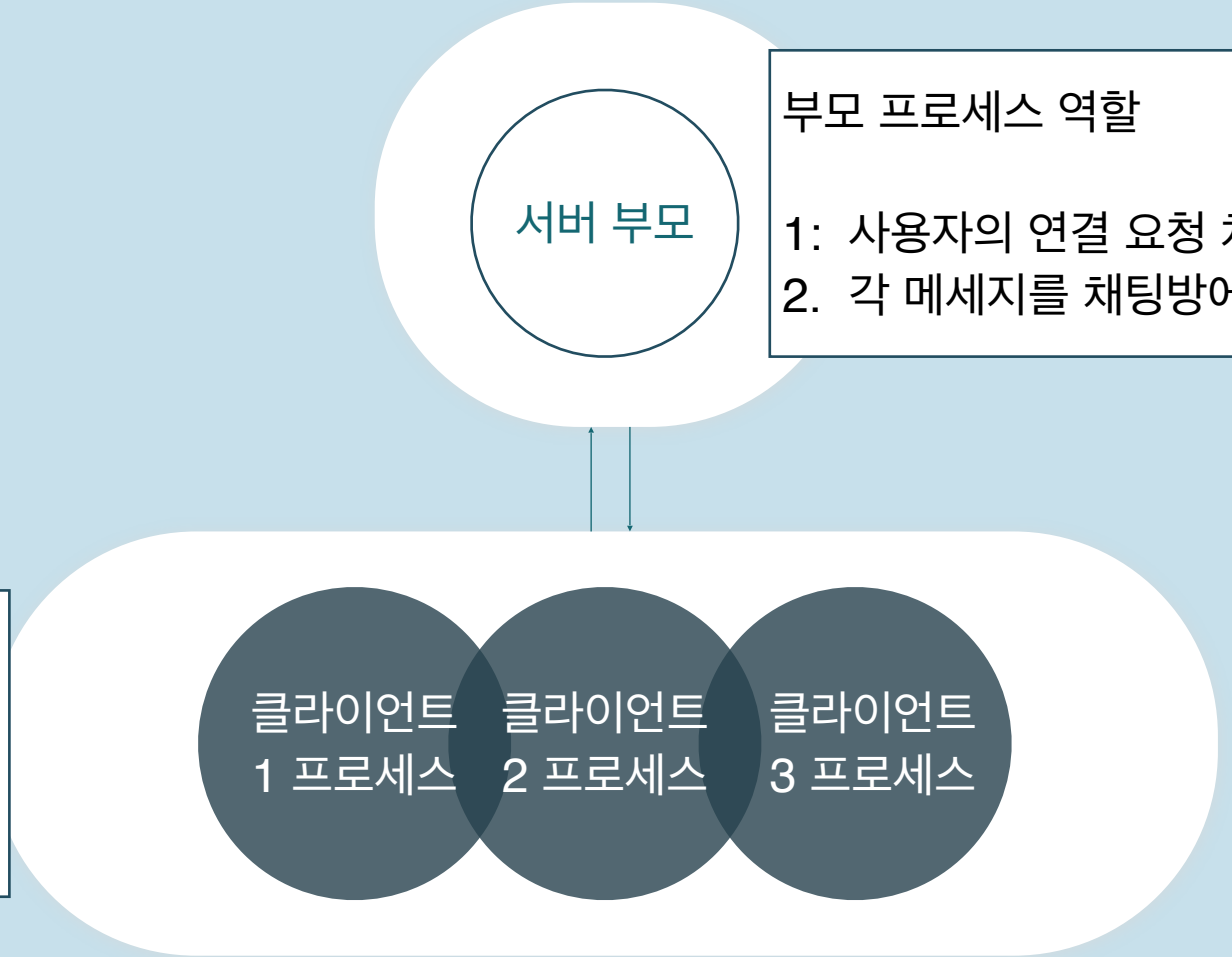
1

구현 방법

채팅 서버

서버 개요

1. 부모 프로세스에서는 자식들의 소켓 정보를 저장
2. 부모 프로세스는 연결을 수립함
3. 자식은 키보드입력(메세지)를 부모로 전달
4. 자식은 클라이언트 메세지를 수신
5. 다중 채팅방은 파일로 구현

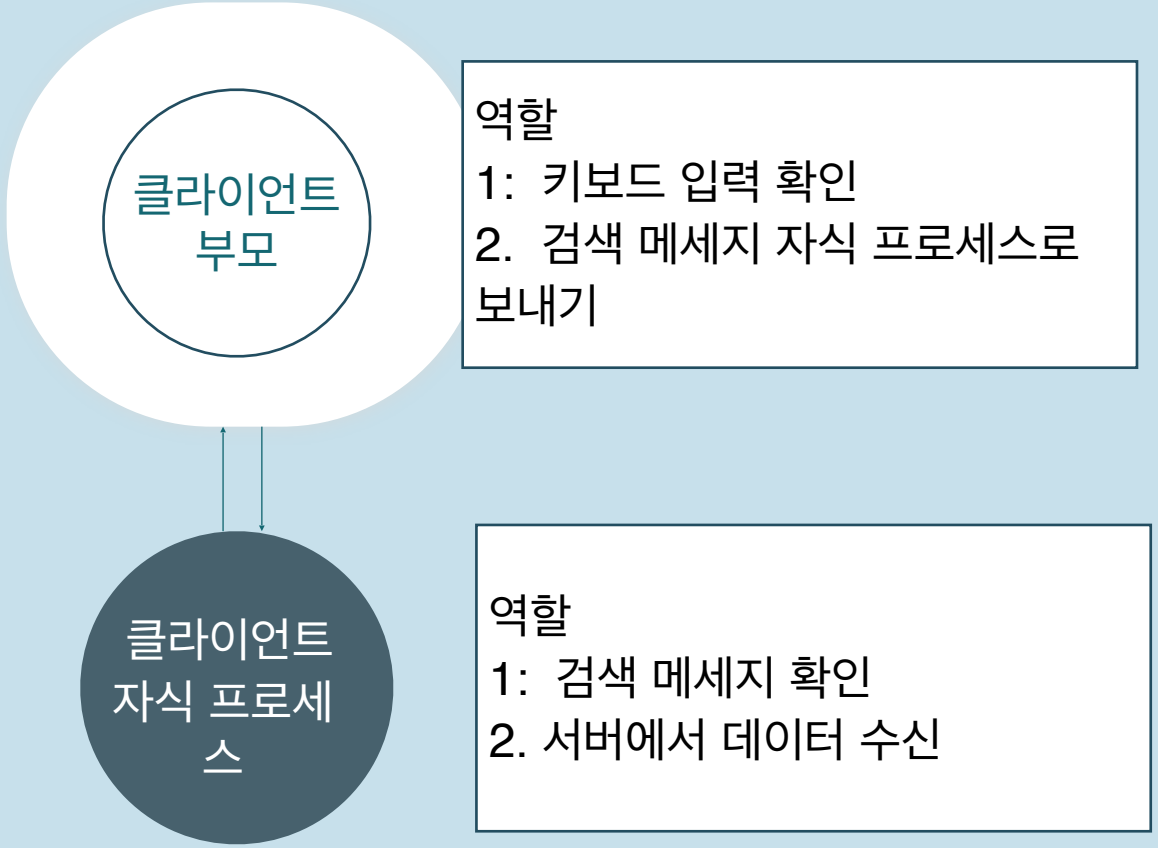


클라이언트 서버

클라이언트 개요

1. 부모 프로세스에서는 키보드 입력을 저장
2. 자식 프로세스는 서버의 응답을 받음

4. 부모 프로세스는 저장한 자식들 소켓 정보로 메세지 발송 처리



전체 구현사항

1. 기존 코드를 Test Double 로 사용
2. 파일 입출력을 이용한 로그인 로그아웃 구현
3. 채팅 시 상대방 아이디 표시
4. cmake를 이용한 빌드 시스템
5. 채팅방 생성 기능
6. 채팅 메세지 검색 기능
7. Fork 를 사용해 구현
8. 서버는 데몬을 사용해 터미널과 무관하게 작동

Part 1

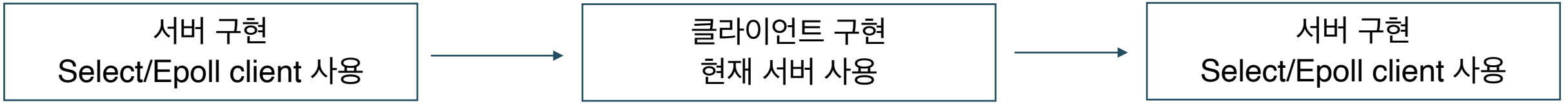
1.기존 코드를 Test Double 로 사용

1.이전에 Select/Epoll 로 만들어진 프로그램을 사용

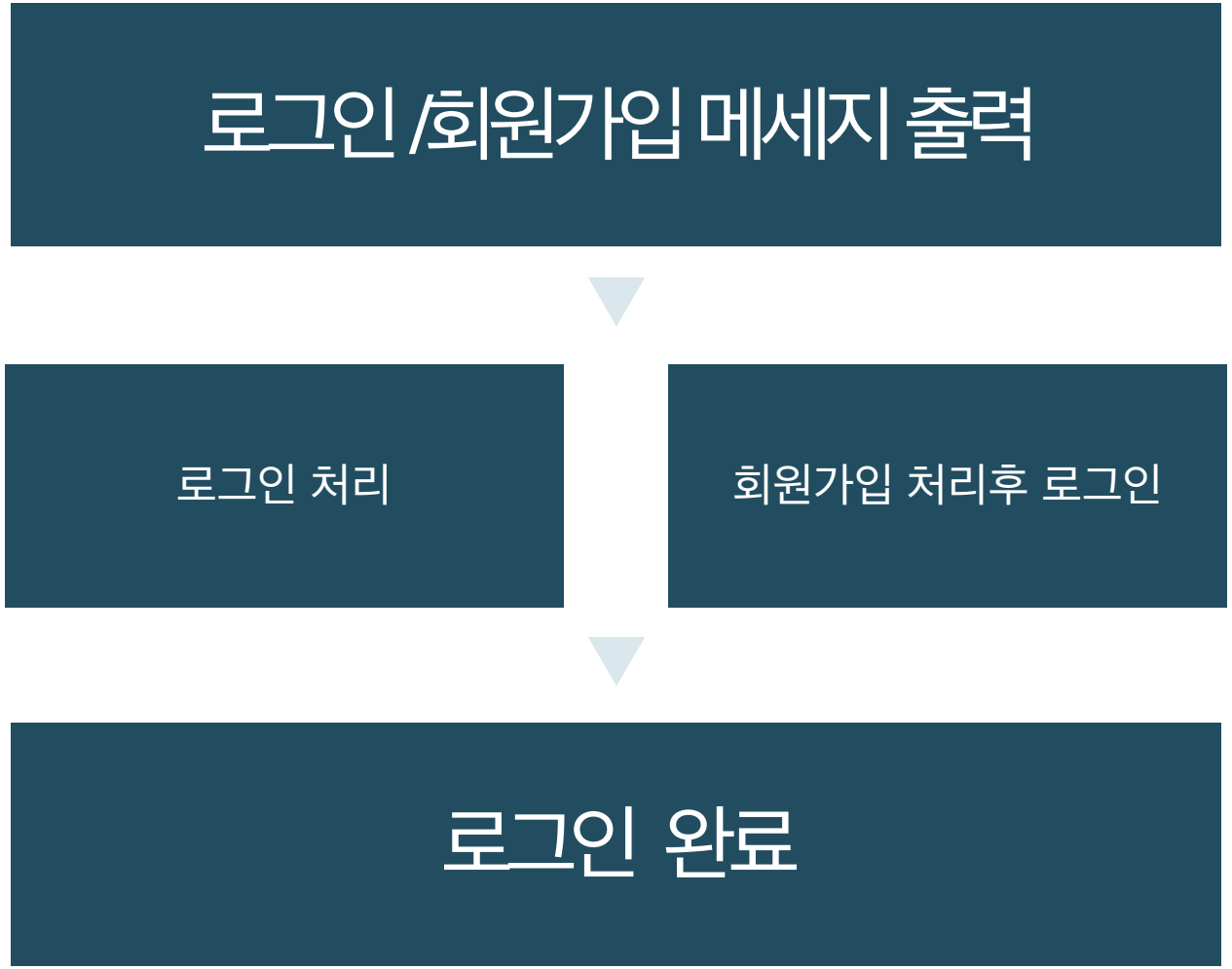
2. 서버 개발시 Select클라이언트를 사용해 테스트에 용이하게 구현

3. 클라이언트 개발 시 서버 구현에 맞추어 구현

진행 방법



2. 로그인 구현



2-1 로그인 처리

1. 올바른 입력

```
This is a fork-based client
input type (login/signup):
login
ID:
lee
PW:
1234
SUCCESS LOGIN
```

2. 올바르지 않은 입력

```
input type (login/signup):
ID:
lee
PW:
ewii
INCORRECT UserName PassWord
```

3. 올바르지 않은 로그인 옵션

```
input type (login/signup):
login
WRONG OPTION QUIT CONNECTION
```

2-2 로그아웃 처리

… 입력시 로그아웃(종료) 처리

```
ID:
lee
PW:
1234
SUCCESS LOGIN
Input ChatRoom Number
If you want to create use 'create:{channelName} to create channel
default:0

create:fusifj
Channel Added

You Selected channel:1
Start chat

...
exit.
```

3 상대 아이디 표시

상대방의 ID를 터미널에 같이 표시함

```
You Selected channel:0
Start chat

kim: hello
```

4. Cmake를 사용한 빌드

Server

```
cmake_minimum_required(VERSION 3.25)
project(server C)

set(CMAKE_C_STANDARD 11)

add_executable(server
        tcpServer.c)
```

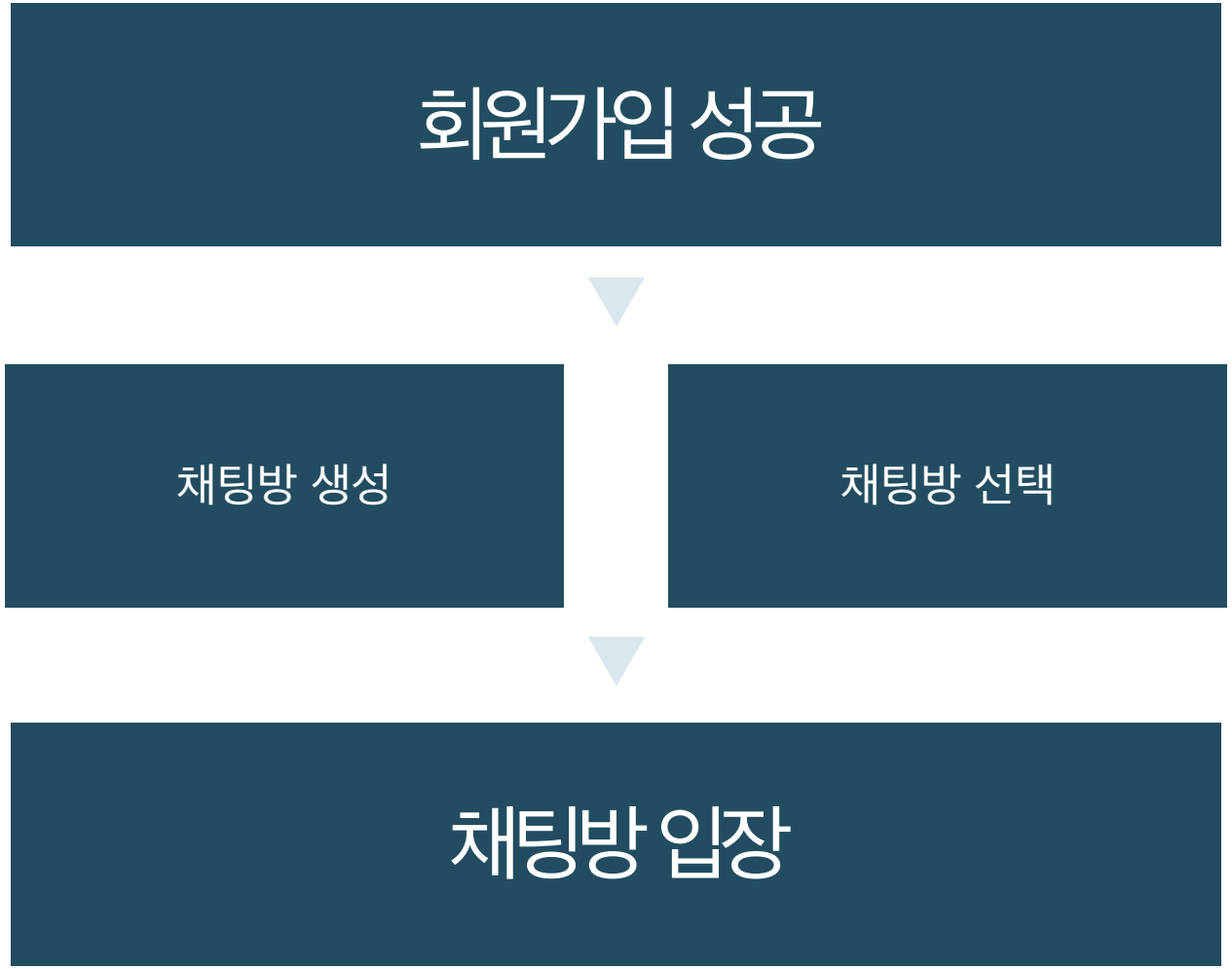
Client

```
cmake_minimum_required(VERSION 3.28)
project(client C)

set(CMAKE_C_STANDARD 11)

add_executable(client tcpClient.c)
```

5. 채팅방 생성/선택 기능



5. 채팅방 생성/선택 기능

1. 채팅방을 선택하는 경우

```
Input ChatRoom Number
If you want to create use 'create:{channelName} to create channel
default:0 fuisfu:1

1

You Selected channel:1
Start chat
```

채팅방 번호를 입력하면 해당 번호로 입장

2. 채팅방을 생성하는 경우

```
Input ChatRoom Number
If you want to create use 'create:{channelName} to create channel
default:0 fuisfu:1

create:diiio
Channel Added

You Selected channel:2
Start chat
```

create: 를 사용해 채팅방 생성
생성된 채팅방의 채널 이름이 표시됨

6. 채팅 메세지 검색 기능

```
Start chat

hi
kim: hi

kim: good morining

oh my god
kim: project is too hard

search:pro
Chat Log
Scan chat log ... please wait
kim: project is too hard

Search Finished
```

1. 검색방법: Search:키워드

2. KMP 알고리즘을 사용해 문자열 매칭

7. Fork 사용

```
pid_t  pid = fork();
if ( pid == 0) { // 자식 프로세스
    close(ssock);
    handleClient(csock,  client_index: client_count, pipe_fd, cliaddr);
}
process_info[client_count].pid = pid;
process_info[client_count].csock = csock;
client_csock_info[client_count].csock = csock;
close(pipe_fd[client_count][1]); // 부모는 쓰기 디스크립터 닫기
client_count++;
```

8. 서버 프로그램을 데몬으로 사용

```
pi@raspberrypi:~/miniserver/server/build $ ps aux | grep server
pi          1739  0.0  0.0   2512  1536 ?        Ss   10:50   0:00 /usr/lib/openssh/sftp-server
pi          3375  0.0  0.0   7584  1536 pts/0    S+   14:45   0:00 grep --color=auto server
pi@raspberrypi:~/miniserver/server/build $ ./server
currentDirectory: /home/pi/miniserver/server/build
pi@raspberrypi:~/miniserver/server/build $ ps aux | grep server
pi          1739  0.0  0.0   2512  1536 ?        Ss   10:50   0:00 /usr/lib/openssh/sftp-server
pi          3378 99.0  0.0   2320  1024 ?        R    14:45   0:01 ./server
pi          3380  0.0  0.0   7584  1536 pts/0    S+   14:45   0:00 grep --color=auto server
pi@raspberrypi:~/miniserver/server/build $
```

2

설계 상세

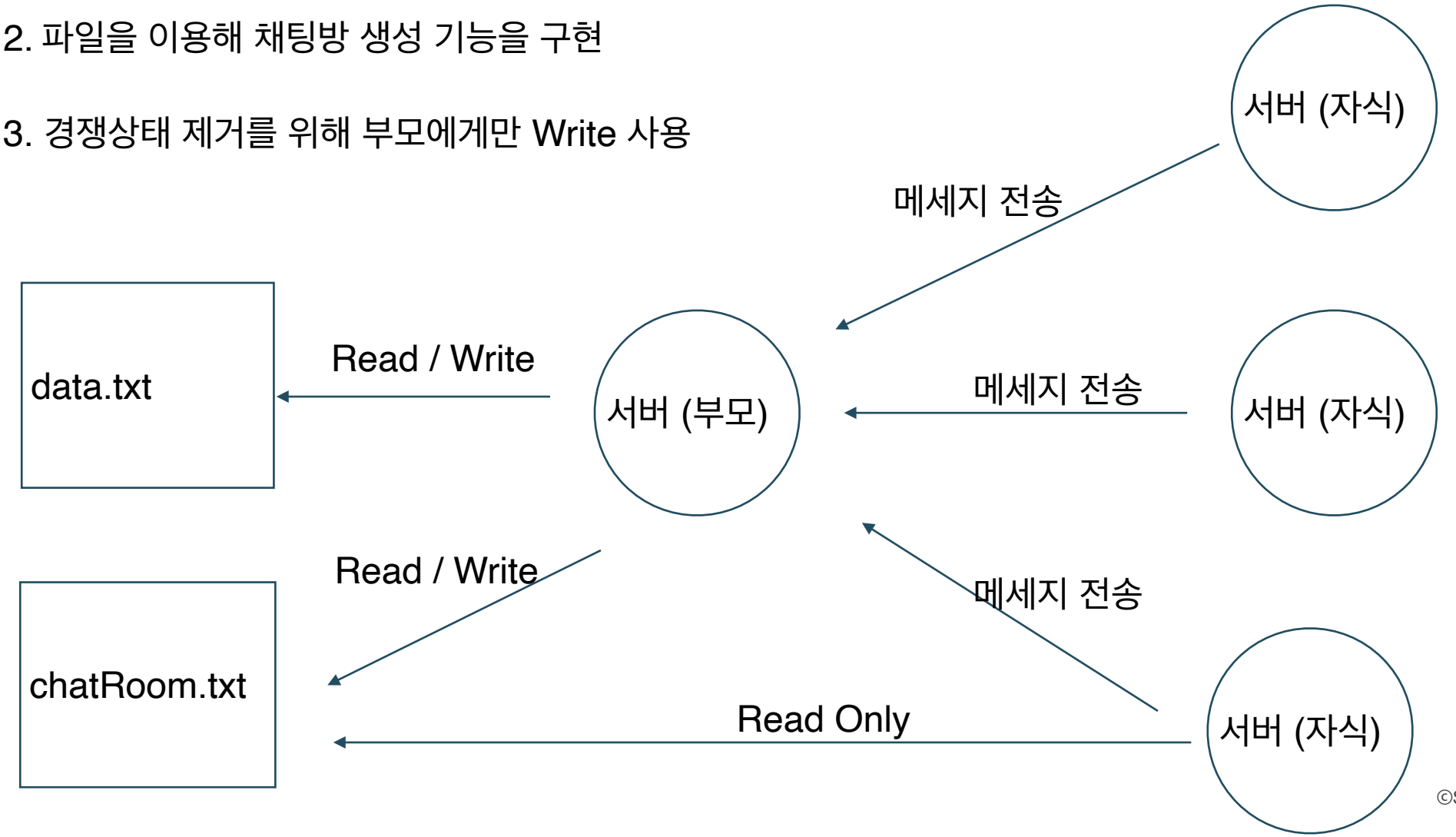
Part 2

서버 프로그램 상세

1. 파일을 이용해 회원가입/로그인 을 구현

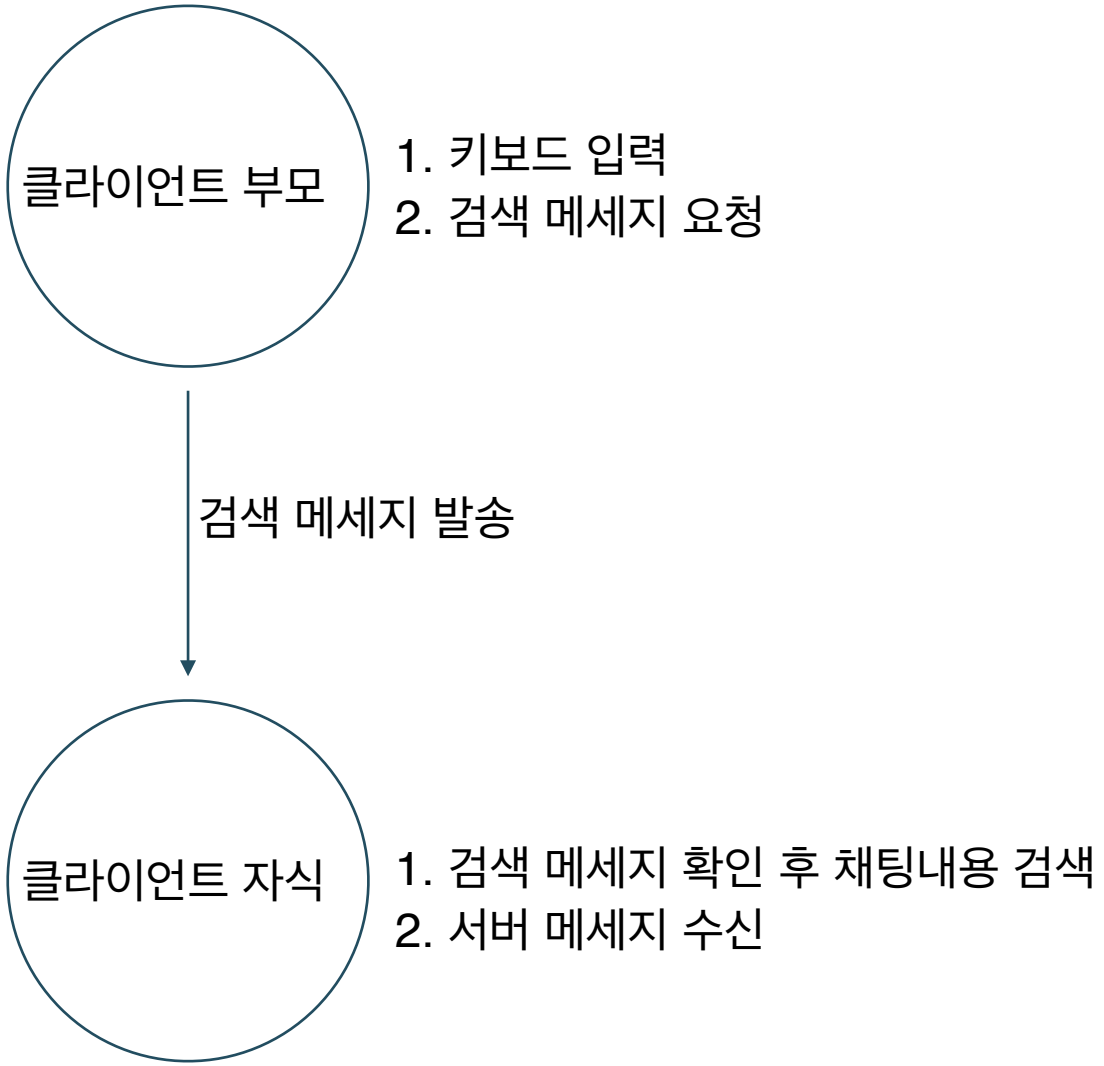
2. 파일을 이용해 채팅방 생성 기능을 구현

3. 경쟁상태 제거를 위해 부모에게만 Write 사용



Part 2

클라이언트 프로그램 설계 상세



3

보완할 점

보완할 점

1. 구현을 위한 완벽한 자료구조 설계의 미흡

현재는 두개의 struct로 정보를 저장하는데 조금 더 개선의 여지가 있어 보임

```
typedef struct userinfo {
    char userName[10];
    char message[BUFSIZ];
    int csockId;
    int room_number;
} userinfo;

typedef struct {
    pid_t pid;
    int csock;
} ProcessInfo;
```

보완할 점

2. 연결리스트로 로그인 한 사용자 저장 필요

현재는 배열을 사용해서 동적으로 사용자를 저장할 수가 없는 상황임

LinkedList 와 같은 자료구조를 이용해 동적으로 사용자를 받아 성능 개선을

기대해 볼 수 있음

```
userLoginInfo users[MAX_USERS]; // 사용자 정보를 저장할 배열
int pipe_fd[MAX_CLIENTS][2];
```

보완할 점

3. 자동화 된 테스트 필요

현재는 일일히 수기로 클라이언트를 만들어 테스트를 하였음

테스트에 시간이 굉장히 많이 걸리고, 정교한 테스팅이 불가능함.

테스트 코드를 작성함으로써 시간을 줄이고 테스트 정확도를 높일 수 있음

부하 테스트를 작성함으로써 해당 프로그램이 어느 정도의 사용자를 확보할 수 있는지 테스트 가능

```
// 여러 개의 클라이언트 스레드 생성
for (int i = 0; i < NUM_CLIENTS; i++) {
    thread_ids[i] = i;
    if (pthread_create(&threads[i], NULL, client_thread, (void *)&thread_ids[i
        perror("pthread_create()");
        return -1;
    }
    usleep(100000); // 0.1초 대기하여 클라이언트 생성 시 간격을 둠
}
```

보완할 점

4. 파일을 사용하지 않고 파이프 사용 필요

파이프를 두개를 사용할 수 있다는 사실을 몰라서 파이프 대신 파일을 사용하였으나,

파일을 사용하는 방식이 어떤 플랫폼,버전, 특정 상황에서 일관되게 동작하는 것을 기대하지 못함

ex) 서버가 실수로 2번 실행되면 예상하는 것과 다르게 동작할 수 있음.

이런 경우 문제 상황 인식에 오랜 시간이 걸림.

또한 OSX, Windows , linux에서 전부 일관되게 동작한다는 보장이 없음

4

프로그램 실행 방법

Part 4

실행 방법

1. 각 폴더에서 build 폴더 생성 (mkdir build)

2. build 폴더 이동 cd build

3. cmake ..

4. make

5-1 클라이언트 실행 시 ./client (서버 IP)

5-2 서버 실행시 ./server

*** 서버는 데몬으로 실행되기 때문에 다시 실행시 반드시 지워주어야 함

git으로 받아서도 실행 가능함

Cmake 라즈베리 파이의 경우 버전을 조정했지만 그래도 make가 안된다면 버전 조정 필요

버전 문제가 있다면 apt update 사용

*** 혹시나 프로젝트 실행이 되지 않으면 Git에서 clone 후 실행 가능

프로젝트 git 링크 : https://github.com/YOUNGHO0/ChattingService