

시스템프로그래밍실습 프로젝트2 결과보고서



수 강 과 목 : 시스템프로그래밍실습
담 당 교 수 : 우홍욱 교수님
학 과 : 소프트웨어학과
학 번 : 2020315798
이 름 : 최진우
제 출 일 : 2022년 11월 28일

1. 프로젝트 개요

1) 목표 : Ticketing Server 구현

2) 세부 목표

- Socket을 활용한 Server-Client 통신 및 Server 내에서 Pthread와 Mutex를 활용한 Ticketing 구현

2. 프로젝트 개발 환경

1) 개발 언어 : C

2) 개발 환경 : Ubuntu 18.04.5 LTS (성균관대학교 인의예지 클러스터)

3) 컴파일러 : gcc 7.5.0

4) 사용 프로그램 및 목적

· Putty : 컴파일 및 vi 에디터

· File Zilla : SFTP를 이용한 백업 및 테스트 케이스 전송

3. 프로젝트 설계 및 알고리즘

1) 매크로, 구조체, 전역 변수 선언

```
#define MAX_CLIENT_NUM 1024
#define MAX_SEAT_NUM 256

typedef struct _query{
    int user;
    int action;
    int data;
    int fd;
} query;

typedef struct _user{
    int client;
    int passcode;
    int seat_num;
    int isLogin;
    int isRegistered;
} user;

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
user u[MAX_CLIENT_NUM];
int seat_arr[MAX_SEAT_NUM];
```

- MAX_CLIENT_NUM : 최대 Client 수
- MAX_SEAT_NUM : 최대 좌석 수
- query 구조체 : Client에서 보내는 쿼리의 정보들
 - user : 해당 Client에서 접속한 user 번호
 - action : 0~5에 해당 하는 프로시저 분기 번호
 - data : action에 따라 비밀번호 내지는 좌석 번호
 - fd : 해당 Client의 고유 file descriptor 번호
- user 구조체 : User마다 지니는 고유 정보
 - client : 앞서 query에서 불러온 fd 값과 비교용 번호
 - passcode : 해당 user의 비밀번호
 - seat_num : 해당 user의 좌석 번호
 - isLogin : 해당 user의 현재 로그인 여부
 - isRegistered : 해당 user의 가입 여부
 - isReserved : 해당 user의 자리 예약 여부
- lock : Mutex 변수
- u : MAX_CLIENT_NUM 크기의 user형 배열
- seat_arr : MAX_SEAT_NUM 크기의 int형 배열, 각 원소는 -1로 초기화 및 추후 예약 시 해당 user의 인덱스가 할당

2) 소켓 통신용 변수 선언 및 전역 변수 초기화

```
int main(int argc, char *argv[]){
    int listenfd, connfd, caddrlen;
    struct hostent *h;
    struct sockaddr_in saddr, caddr;
    pthread_t thread;
    if(argc < 3){
        printf("argv[1]: server address, argv[2]: port number\n");
        exit(1);
    }

    for(int i=0; i<MAX_CLIENT_NUM; i++){
        u[i].client = 0;
        u[i].passcode = 0;
        u[i].seat_num = -1;
        u[i].isLogin = 0;
        u[i].isRegistered = 0;
        u[i].isReserved = 0;
    }
    for(int i=0; i<MAX_SEAT_NUM; i++){
        seat_arr[i] = -1;
    }
}
```

- listenfd, connfd, caddrlen, *h 등 추후 socket(), bind(), listen() 등 Server단에서 호출할 함수들의 리턴값 저장용 변수 선언
- argc 개수를 통한 예외 처리
- 사용자 배열인 u와 좌석 배열인 seat_arr의 초기화 : 좌석은 -1, 나머지는 0으로 초기화 진행

3) 소켓 통신용 함수 호출

```
// Server side setting
memset((char *)&saddr, 0, sizeof(saddr));
saddr.sin_family = AF_INET;
saddr.sin_addr.s_addr = htonl(INADDR_ANY);
saddr.sin_port = htons(atoi(argv[2]));

if((listenfd = socket(PF_INET, SOCK_STREAM, 0)) < 0){
    printf("socket() failed.\n");
    exit(1);
}
if(bind(listenfd, (struct sockaddr *)&saddr, sizeof(saddr)) < 0){
    printf("bind() failed.\n");
    exit(1);
}
if(listen(listenfd, MAX_CLIENT_NUM) < 0){
    printf("listen() failed.\n");
    exit(1);
}
```

- memset을 통한 saddr 구조체 초기화 진행
- socket(), bind(), listen() 등 소켓 통신용 함수 호출(accept()는 추후 진행)

4) I/O Multiplexing

```
// I/O Multiplexing
fd_set readset, copyset;
FD_ZERO(&readset);
FD_SET(listenfd, &readset);
int fdmax = listenfd, fdnum;

while(1){
    copyset = readset;

    if((fdnum = select(fdmax+1, &copyset, NULL, NULL, NULL)) < 0){
        printf("select() failed.\n");
        exit(0);
    }

    for(int i=0; i<fdmax+1; i++){
        if(FD_ISSET(i, &copyset)){
            if(i == listenfd){
                if((connfd = accept(listenfd, (struct sockaddr *)&caddr, (socklen_t *)&caddrlen)) < 0){
                    printf("accept() failed.\n");
                    continue;
                }
                FD_SET(connfd, &readset);
                if(fdmax < connfd){
                    fdmax = connfd;
                }
                printf("Connected to a new client!\n");
            }
            else{
                query qu;
                if(recv(i, &qu, sizeof(qu), 0) > 0){
                    qu.fd = i;
                    pthread_create(&thread, NULL, &funct_thread, &qu);
                }
                else{
                    FD_CLR(i, &readset);
                    printf("Connection from client %d has been terminated.\n", i-3);
                    close(i);
                }
            }
        }
    }
}

printf("Server Connection terminated.\n");
close(listenfd);
return 0;
```

- 소켓 통신 중 최근에 배운 I/O Multiplexing 방법을 통한 Client 관리
- 초기 연결 시(i==listenfd), accept()를 호출하고 Connected to a new client 출력
- 이후 Client에서 scanf로 받아서 Server에 보내는 구조체 값에 대해 query qu라는 쿼리 구조체에 recv()를 통해 수신
- Client에서 보내는 구조체 내부엔 int형 변수 3개이므로 다음 주소인 qu.fd에 현재 fd 값 할당
- 만들어진 쿼리 구조체를 pthread_create()를 통해 funct_thread라는 스레드 시작 시 호출 함수에 전달
- 연결 종료 시 Connection from client 번호 has been terminated 출력 (fd 번호는 STDERR 이 후부터 지정되므로 i-3으로 번호 출력)

5) Thread function

```
void *funct_thread(void *arg){
    pthread_detach(pthread_self());
    query qu = *(query*)arg;
    int success = 1;
    int fail = -1;
    int result = 0;

    if(qu.user < 0 || qu.user > 1023){
        printf("Unavailable user id.\n");
        send(qu.fd, &fail, sizeof(int), 0);
        return NULL;
    }

    if(qu.user == 0 && qu.action == 0 && qu.data == 0){
        send(qu.fd, seat_arr, sizeof(seat_arr), 0);
    }
    else{
        if(qu.action == 1){
            if(u[qu.user].isRegistered){
                if(u[qu.user].isLogin){
                    printf("User %d is already logged in.\n", qu.user);
                    result = 0;
                }
            }
            else{
                if(u[qu.user].passcode == qu.data){
                    pthread_mutex_lock(&lock);
                    u[qu.user].client = qu.fd;
                    u[qu.user].isLogin = 1;
                    pthread_mutex_unlock(&lock);
                }
            }
        }
    }
}
```

- pthread_detach() : 한 스레드가 종료될 때까지 기다릴 필요가 없이 어떤 Client에서라도 값이 전달되면 실행되어야 함
- query qu = *(query*)arg를 통해 main 함수에서 전달받은 인자 할당
- user 값 예외처리
- user, action, data 모두 0일 경우 Client 좌석 배열 정보 전달
- 이후 각 action 별 분기를 통해 로직 진행

6) Mutex

```
if(u[qu.user].passcode == qu.data){
    pthread_mutex_lock(&lock);
    u[qu.user].client = qu.fd;
    u[qu.user].isLogin = 1;
    pthread_mutex_unlock(&lock);
```

로그인 로직

```
pthread_mutex_lock(&lock);
u[qu.user].client = qu.fd;
u[qu.user].passcode = qu.data;
u[qu.user].isRegistered = 1;
u[qu.user].isLogin = 1;
pthread_mutex_unlock(&lock);
```

회원가입 로직

```
pthread_mutex_lock(&lock);
u[qu.user].isReserved = 1;
u[qu.user].seat_num = qu.data;
seat_arr[qu.data] = qu.user;
pthread_mutex_unlock(&lock);
```

자리 예약 로직

```
pthread_mutex_lock(&lock);
u[qu.user].isReserved = 0;
u[qu.user].seat_num = -1;
seat_arr[qu.data] = -1;
pthread_mutex_unlock(&lock);
```

자리 예약 취소 로직

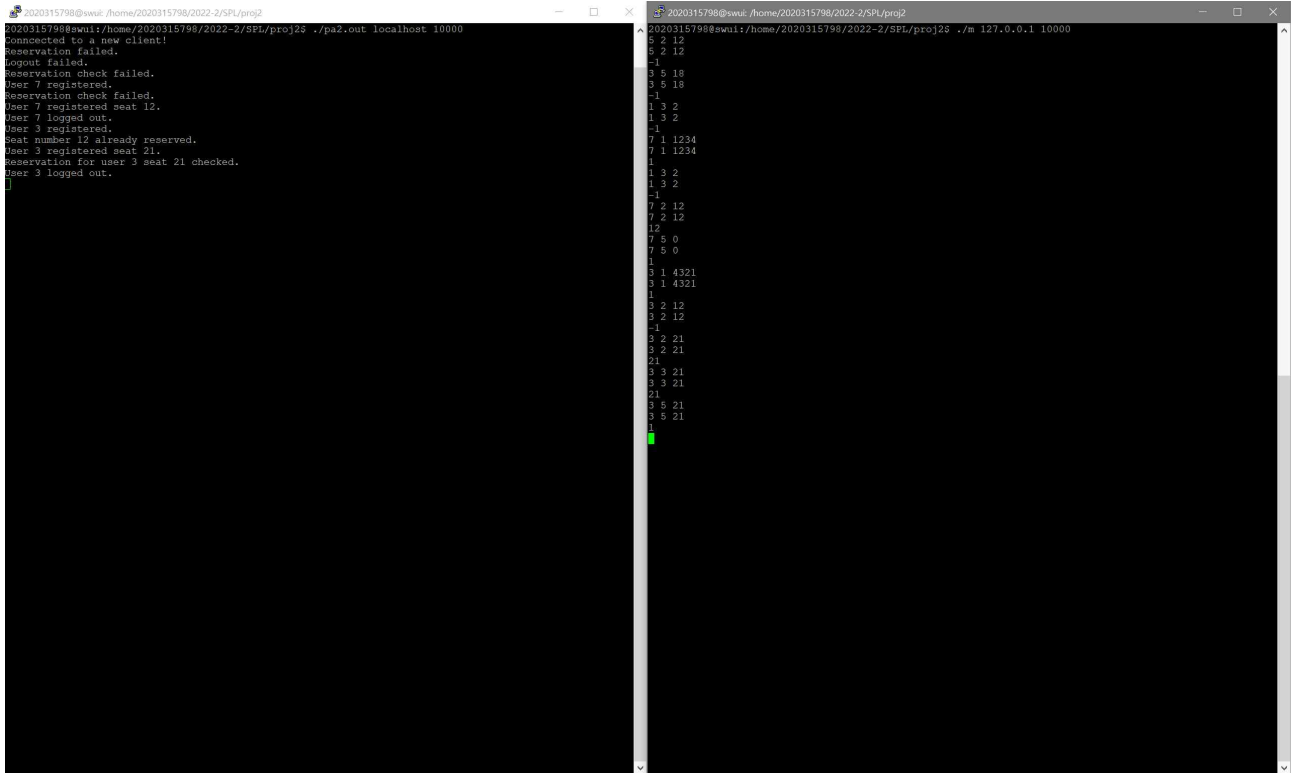
```
pthread_mutex_lock(&lock);
u[qu.user].client = 0;
u[qu.user].isLogin = 0;
pthread_mutex_unlock(&lock);
```

로그아웃 로직

쓰레드 호출 함수에서 위 5가지 경우에 대해 멀티 쓰레드가 공유하는 전역 변수에 접근하고 있습니다. 따라서 각 전역 변수의 접근마다 write 전에 mutex lock을 걸고 write이 끝나면 mutex unlock을 하여 동기화 문제를 해결했습니다.

4. 프로젝트 결과

1) 과제 계획서 1번 예시



```

2020315798@swul: /home/2020315798/2022-2/SPL/proj2
2020315798@swul: /home/2020315798/2022-2/SPL/proj2
Connected to a new client!
Reservation failed.
Logout failed.
Reservation check failed.
User 7 registered.
Reservation check failed.
User 7 registered seat 12.
User 7 logged out.
User 3 registered.
Seat number 12 already reserved.
User 3 registered seat 21.
Reservation for user 3 seat 21 checked.
User 3 logged out.

5 2 12
5 2 12
-1
3 5 18
3 5 18
-1
1 2 2
1 3 2
-1
7 1 1234
7 1 1234
1
1 2 2
1 3 2
-1
7 2 12
7 2 12
12
7 5 0
7 5 0
1
3 1 4321
3 1 4321
1
3 2 12
3 2 12
-1
3 2 21
3 2 21
21
3 3 21
3 3 21
21
3 5 21
3 5 21
1

```

과제 계획서에 나온 1번째 예시로, Client에서 특정 User 번호로 로그인하지 않고 진행할 경우 -1이 출력되고 로그인한 후에 좌석을 예약할 경우 예약된 좌석 번호가 나오는 것을 확인할 수 있었습니다. 또한, 다른 User 번호로 이미 예약된 좌석을 예약할 경우 -1을 리턴하고 좌석을 확인할 때는 좌석 번호를 리턴한 것을 확인했습니다.

2) 과제 계획서 2번 예시

```

2020315798@swu: /home/2020315798/2022-2/SPL/proj2
2020315798@swu: /home/2020315798/2022-2/SPL/proj2$ ./pa2.out localhost 10001
Connected to a new client!
User 5 registered.
User 5 is already logged in.
User 3 registered.
User 5 registered seat 10.
Seat number 10 already reserved.
User 5 cancelled seat 10.
User 3 registered seat 10.
User 5 logged out.
User 3 logged out.
User 5 login info incorrect.
User 5 login.
Reservation for user 5 seat 1 not checked.

2020315798@swu: /home/2020315798/2022-2/SPL/proj2
2020315798@swu: /home/2020315798/2022-2/SPL/proj2$ ./m 127.0.0.1 10001
S 1 1
S 1 1
S 2 10
S 2 10
S 4 10
S 4 10
S 5 1
S 5 1
1
1
S 1 1
S 1 1
S 1 7
S 1 7
S 2 10
S 2 10
-1
S 2 10
S 2 10
10
S 5 1
S 5 1
1
S 1 11
S 1 11
-1
S 1 1
S 1 1
1
S 3 1
S 3 1
-1
1

```

과제 계획서에 나온 두 번째 예시입니다. 이미 초회 로그인 시도를 통해 회원가입을 하여 로그인했을 경우 다른 Client에서 해당 User로 로그인이 불가능한 것을 확인했습니다. 마찬가지로 좌석이 중복해서 예약되지 않는 것을 확인했고 비밀번호가 설정된 User에 대해서 다른 비밀번호로 로그인이 불가능 합니다.

