

시스템소프트웨어실습 PA3 리포트

2017313213 박경태

리눅스 환경에서 동작하는 ticketing server 프로그램을 작성하였다.

코드를 설명하는 리포트이며 목차는 아래와 같다.

[목차]

1. 클라이언트 개요
2. 서버 개요
3. Log in
4. Reserve
5. Check reservation
6. Cancel reservation
7. Logout
8. Exit
9. 테스트케이스 실행결과

1. 클라이언트 개요

문제의 명세에 따라, 표준입력 뿐만 아니라 최초 입력 인자의 개수에 따라 일반 텍스트파일로부터도 입력으로 받을 수 있도록 조치하였다.

우선 `get_string_from_fd` 함수는 file descriptor 와 dest 문자열 주소가 주어졌을 때, file descriptor로부터 개행까지를 읽은 뒤 dest 에 덮어쓰는 함수이며 `getline` 의 `filedescriptor` 버전으로 직접 구현하였다.

`parse_query` 함수는 말 그대로 문자열로 전송된 메시지를 3 개의 정수형으로 파싱하며, 만일 인자 개수가 3 개가 안될 경우 `order_cnt!=3` 에 따라 `Invalid query` 라는 메시지를 터미널에 출력 후 -1을 리턴한다. 3 개가 딱 맞을 경우, 양식에 맞는 `query` 라는 뜻이므로 `query` 인자에 각각의 값을 써준 뒤 0 을 반환하는데 이는 함수 실행성공을 의미한다.

`main` 함수에서는 인자의 개수를 체크하여, 3 개라면 terminal input mode 로, 4 개라면 file input mode 로 인식한 뒤 표준입력을 `stdin`에서 `input_fd`로 `dup2` 함수를 통해 덮어쓴다.

그 뒤, `sock`, `connect` 를 통해 서버와 연결하며

`fd_set` 을 이용한 Condition Variable 을 활용하여 서버로부터의 출력 감지 및 서버로의 신호 전송 모두를 동시에 구현하였다.

서버로 `query` 를 보낼 경우, `get_string_from_fd` 함수를 통해 1줄 읽고, `parse_query` 함수를 통해 `query` 를 파싱한 다음 `query` 구조체를 `write` 를 통해 서버로 전송한다.

서버로부터의 응답이 올 경우, 첫 `reply` 는 항상 `query` 자료형으로 오게 되어있으므로 `reply` 의 `q.user` 변수가 -1 인지의 여부를 확인하여 성공동작했는지를 체크하고, `q.action` 을 통해 어느 `action` 인지 확인하며 이를 `switch` 문으로 구현했다.

0 0 0 일 경우, 클라이언트는 폐쇄되고 소켓을 `close` 한 뒤 종료하며

그 외의 경우, 1, 2, 3, 4, 5 각각 서버의 순서와 동일하게 `login`, `reserve`, `check reservation`, `cancel reservation`, `logout` 이다.

4 가지 동작은 서버로부터의 결과를 받아와서 그 결과를 해석하여 출력하는 단순한 동작이나, `check reservation` 동작은 단순 반환값이 아니라 추가적인 정수형 배열이 들어오므로 추가적인 조치가 필요했는데,

먼저 `read int` 를 통해 배열의 길이를 받고, `read buf` 를 통해 배열을 받은 뒤 `for` 문을 `cnt` 만큼 돌면서 `buf` 를 출력함으로써 해당 `user` 가 어느어느 자리를 현재 예약한 상태인지를 표시해주는

기능을 구현하였다.

```
20  typedef struct _query {
21      int user;
22      int action;
23      int seat;
24  } query;
25
26  int get_string_from_fd(int fd, char *dest) { // fd로부터 개행까지 읽어온 뒤 개행은 빼고, dest에 넣어줌
27      int rd_bytes;
28      char c;
29      int col = 0;
30      dest[0] = 0; //empty array
31      while (0 < (rd_bytes = read(fd, &c, 1))) { // \n 없이 전부 읽고 저장하기
32          dest[col++] = c;
33          if (c == '\n') {
34              dest[col++] = '\0';
35              break;
36          }
37      }
38      dest[col] = '\0';
39      return strlen(dest);
40  }
41
42  int parse_query(query *q, char *msg) {
43      char *order[MAXLINE];
44      int order_cnt = 0;
45      order[order_cnt++] = strtok(msg, " \t\n");
46      while (order[order_cnt - 1] != NULL) {
47          order[order_cnt] = strtok(NULL, " \t\n");
48          if (order[order_cnt] == NULL)
49              break;
50          else
51              order_cnt++;
52      }
53      order[order_cnt] = NULL;
54      // for (int i = 0; i < order_cnt; i++) {
55      //     printf("order[%d]==%s\n", i, order[i]);
56      // }
57      if (order_cnt != 3) {
58          printf("Invalid query\n");
59          return -1;
60      }
61      q->user = atoi(order[0]);
62      q->action = atoi(order[1]);
63      q->seat = atoi(order[2]);
64      return 0;
65  }
66
```

```
67 ~ int main(int argc, char *argv[]) {
68     // int sock = socket(PF_INET, SOCK_STREAM, 0);
69     int sock, n;
70     char *host = argv[1];
71     int port = atoi(argv[2]);
72     struct hostent *h;
73     struct sockaddr_in saddr;
74     int a1, a2, a3, input_fd;
75     char msg[MAXLINE];
76
77 ~     if (argc == 3) {
78         // printf("Terminal input mode\n");
79 ~     } else if (argc == 4) {
80         // printf("File input mode\n");
81         char *input_path = argv[3];
82         // printf("input_path = \'%s\'\n", input_path);
83         input_fd = open(input_path, O_RDONLY);
84         dup2(input_fd, STDIN_FILENO);
85         close(input_fd);
86 ~     } else {
87         printf("Follow the input rule below\n");
88         printf("=====\\n");
89         printf("argv[1]: server address, argv[2]: port number\\n");
90         printf("or\\n");
91         printf("argv[1]: server address, argv[2]: port number, argv[3]: input file\\n");
92         printf("=====\\n");
93         exit(1);
94     }
95
96 ~     if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
97         printf("socket() failed.\n");
98         exit(1);
99     }
100 ~    if ((h = gethostbyname(host)) == NULL) {
101        printf("invalid hostname %s\\n", host);
102        exit(2);
103    }
104    memset((char *)&saddr, 0, sizeof(saddr));
105    saddr.sin_family = AF_INET;
106    memcpy((char *)&saddr.sin_addr.s_addr, (char *)h->h_addr, h->h_length);
107    saddr.sin_port = htons(port);
108 ~    if (connect(sock, (struct sockaddr *)&saddr, sizeof(saddr)) < 0) {
109        printf("Connection failed\\n");
110        exit(1);
111    }
```

```

112     fd_set originalset, readset;
113     FD_ZERO(&originalset); /* initialize socket set */
114     FD_SET(STDIN_FILENO, &originalset);
115     FD_SET(sock, &originalset);
116     int fdmax = sock, fdnum;
117
118     while (1) {
119         query q;
120         readset = originalset;
121         fdnum = select(fdmax + 1, &readset, NULL, NULL, NULL);
122         if (fdnum < 0) {
123             printf("select() failed.\n");
124             exit(4);
125         }
126         if (fdnum == 0) {
127             // printf("Time out.\n");
128             continue;
129         }
130         for (int i = 0; i < MAXLINE; i++) {
131             msg[i] = 0;
132         }
133
134         for (int i = 0; i < fdmax + 1; i++) {
135             if (!FD_ISSET(i, &readset)) {
136                 // printf("%d fd SKIP\n", i);
137                 continue;
138             }
139             if (i == sock) { // 서버로부터의 응답
140                 n = read(sock, &q, sizeof(query));
141                 int success;
142                 if (q.user == -1)
143                     success = 0;
144                 else
145                     success = 1;
146                 int mode = q.action;
147                 // printf("RECEIVED %d %d %d from server\n", q.user, q.action, q.seat);
148                 switch (mode) {
149                     case 0:
150                         if (q.action == 0 && q.seat == 0 && q.user == 0) { // 서버로부터의 자살명령
151                             close(sock);
152                             exit(0);
153                         }
154                         break;
155                     case 1: // 1. Log in
156                         if (success)
157                             printf("Login success\n");
158                         else
159                             printf("Login failed\n");
160                         break;

```

```
165 |             printf("Reservation failed\n");
166 |         break;
167 |     case 3: //3. Check reservation
168 |         if (success) {
169 |             printf("Reservation: ");
170 |             int cnt;
171 |             int buf[MAXSEAT];
172 |             n = read(sock, &cnt, sizeof(int));
173 |             n = read(sock, &buf, sizeof(int) * MAXSEAT);
174 |             for (int i = 0; i < cnt; i++) {
175 |                 printf("%d ", buf[i]);
176 |             }
177 |             printf("\n");
178 |         } else
179 |             printf("Reservation check failed\n");
180 |         break;
181 |     case 4: //4. Cancel reservation
182 |         if (success)
183 |             printf("Seat %d is canceled\n", q.seat);
184 |         else
185 |             printf("Cancel failed\n");
186 |         break;
187 |     case 5: // 5. Log out
188 |         if (success)
189 |             printf("Logout success\n");
190 |         else
191 |             printf("Logout failed\n");
192 |         break;
193 |     default:
194 |         printf("switch error\n");
195 |         exit(0);
196 |     }
197 | } else if (i == STDIN_FILENO) { // 표준입력 또는 리다이렉션 표준입력이면
198 |     // printf("HERE\n");
199 |     // if (read(STDIN_FILENO, msg, MAXLINE) < 0) {
200 |     while (get_string_from_fd(STDIN_FILENO, msg) != 0) {
201 |         // printf("Received msg = %s\n", msg);
202 |         if (parse_query(&q, msg) != 0) {
203 |             // printf("parse_query error\n");
204 |             // printf("Invalid query");
205 |             continue;
206 |         }
207 |         if (write(sock, &q, sizeof(query)) < 0) {
208 |             printf("write error\n");
209 |             continue;
210 |         }
211 |         // printf("SENT q == %d %d %d\n", q.user, q.action, q.seat);
212 |     }
213 | }
```

```
    }
}

close(sock);

return 0;
}
```

2. 서버 개요

서버는 최대 1024 개의 클라이언트를 동시에 처리할 수 있어야 하며, 문제의 요구사항에 따라 Multiple Thread 로써 구현하였다.

client2user, user2client 배열을 활용해 몇 번의 user 가 몇 번의 client 에 접속한 상태인지, 또는 역으로 몇 번의 클라이언트를 어느 User 가 점유하고 있는지를 확인할 수 있다.

또한 client2fd 배열을 통해 몇 번 client 가 몇 번 fd 와 연결되어있는지를 체크할 수 있다.

또한, c2u_mutex, u2c_mutex 배열을 통해 각각의 client, user 현황을 변경하는 동안 Collision 이 발생하지 않도록 하였다.

최대 1024 가지의 client 를 처리해야 하므로, pthread_t tid 배열도 1024 개로 선언하였으며 각각 User 의 Password 도 보관해야하므로 선언했다.

256 개의 좌석에 대해서도 배열을 선언했으며 또한 seat_mutex 배열을 통해 좌석 예약 또는 예약 해제 도중 충돌이 생기지 않도록 하였다.

추가적으로, 뮤텍스를 lock 또는 unlock 하는 순서를 u2c->c2u->seat 으로 통일시킴으로써 뮤텍스로 인한 데드락이 발생하지 않도록 조치하였다.

query 는 user, action, seat 으로 이루어지는 구조체이며

main 함수는 socket 선언, bind listen 후 무한루프를 돌며 accept 를 하는 방식으로 되어있다. 이 때, accept 할 시 pthread_create 를 통해 해당 클라이언트에 맞는 새로운 스레드에 thread_func 를 실행시킴과 동시에 malloc 시킨 client_sock2 를 인자로 보내 실행시킨다. 이 client_sock2 는 0 0 0 입력을 client 가 보낼 시 EXIT 동작을 통해 close 되게 된다.

thread_func 에서는 우선 pthread_detach 를 통해 스레드를 join 하지 않고도 자원이 해제되도록 처리하였으며, malloc 된 client_sock2 가 arg 인자로써 해제된다. 다만, int 값은 그대로 남기 때문에 file descriptor 는 그대로 사용할 수 있다.

그 이후, while 문을 무한으로 돌면서 query 가 수신되는지를 확인한다.

만일 0 0 0 입력일 경우, 클라이언트에게 0 0 0 이라는 신호를 반송하고 해당 스레드는 종료된다.

또한 0 0 0 이라는 신호를 서버로부터 반송받은 클라이언트 또한 종료된다.

그 이외의 경우, q.action 에 따라 다른 함수가 실행되도록 되어있는데, 이는 switch 문으로 구현되어 효율 및 가독성을 높였다.

1일 경우 login, 2 일 경우 reserve, 3 일 경우 check_reservation, 4 일 경우 cancel_reservation, 5 일 경우 logout 인데 이에 대한 설명은 뒤의 각각의 항목에서 상세히 설명하도록 한다.

```

int client2fd[MAXUSER];
int client2user[MAXUSER]; // -1: 오프라인, 0,1,2,3...: 접속중인 USER 번호
pthread_mutex_t c2u_mutex[MAXUSER];
int user2client[MAXUSER]; // -1: 오프라인, 0,1,2,3...: 접속중인 Client 번호
pthread_mutex_t u2c_mutex[MAXUSER];
pthread_t tid[MAXUSER];
int password[MAXUSER]; // password[idx] = idx 번째 user 의 비밀번호

int seat[MAXSEAT]; // 0: 공석, 1: 예약
pthread_mutex_t seat_mutex[MAXSEAT]; // i번째 자리를 누가 편집중인지의 여부를 나타낸다.

typedef struct _query {
    int user;
    int action;
    int seat;
} query;

int main(int argc, char *argv[]) {
    int n, serv_sock, clnt_sock, caddrlen;
    struct sockaddr_in saddr, caddr;
    initialize();

    if ((serv_sock = socket(PF_INET, SOCK_STREAM, 0)) < 0) {
        printf("socket() failed\n");
        exit(1);
    }
    memset((char *)&saddr, 0, sizeof(saddr));
    saddr.sin_family = AF_INET;
    saddr.sin_addr.s_addr = htons(INADDR_ANY);
    saddr.sin_port = htons(atoi(argv[1]));

    if (bind(serv_sock, (struct sockaddr *)&saddr, sizeof(saddr)) < 0) {
        printf("bind() failed\n");
        exit(1);
    }
    if (listen(serv_sock, MAXUSER) < 0) {
        printf("listen() failed\n");
        exit(1);
    }

    /* You should generate thread when new client accept occurs
     * and process query of client until get termination query */
    int *client_sock2;
    while (1) {
        // printf("\n");
        client_sock2 = (int *)malloc(sizeof(int));
        if ((*client_sock2 = accept(serv_sock, (struct sockaddr *)&caddr, (socklen_t *)&caddrlen)) < 0) {
            printf("accept() failed\n");
            free(client_sock2);
            continue;
        }
        pthread_create(&tid[*client_sock2 - 3], NULL, thread_func, client_sock2);
    }

    return 0;
}

```

3. Log in

진입하기 전에 c2u_mutex 및 u2c_mutex 를 lock 하여 충돌을 방지한다.

클라이언트가 다른 USER 에 의해 이미 로그인된 상태라면 fail

USER 가 다른 클라이언트에 접속중인 상태라면 fail

패스워드가 다르면 fail

그 외의 경우, 첫 접속일 경우 passwd 를 설정하고 로그인을 하며 첫 접속이 아닐 경우 패스워드를 비교한 뒤 동일하면 접속시킨다.

이 때, password[usernum] 에는 패스워드가 들어있고

client2user[client2fd[client_sock]] 에는 이 클라이언트에 유저가 접속하고 있는지의 여부를

체크하며 있다면 어느 유저가 이 클라이언트를 점유하고 있는지를 확인한다.

user2client[usernum] 은 해당 유저가 어느 클라이언트에 로그인되어 있는 상태인지를 확인한다.

따라서, 이를 통해

"클라이언트가 다른 USER 에 의해 이미 로그인된 상태라면 fail

USER 가 다른 클라이언트에 접속중인 상태라면 fail

패스워드가 다르면 fail"

을 구현가능하며,

로그인의 경우

첫 로그인의 경우 password 배열을 변경하며

로그인은 client2user, user2client 를 각각 수정하는 것으로써 구현할 수 있다.

모든 동작이 종료되면 write 를 통해 클라이언트에게 결과를 전송하며

unlock 을 통해 뮤텍스를 해제한 뒤 동작을 종료한다.

```

/* 뮤텍스 순서: 반드시 c2u->u2c->seat */
// return 1: SUCCESS, return -1: FAIL
int login(int client_sock, int usernum, int passwd) {
    int ret;
    pthread_mutex_lock(&(c2u_mutex[client2fd[client_sock]]));
    pthread_mutex_lock(&(u2c_mutex[usernum]));
    // printf("LOGIN: usernum: %d, passwd: %d\n", usernum, passwd);
    if (client2user[client2fd[client_sock]] != -1) { // 이 클라이언트가 누구든지 이용중이면 로그아웃이 먼저
        // printf("Already Another User is Logged in at THIS CLIENT. LOGOUT FIRST\n");
        ret = -1;
    } else if (user2client[usernum] != -1) { // 이 유저가 어디든지 이미 로그인했다면
        // printf("Already THIS User is Logged in at ANOTHER CLIENT. LOGOUT AT ANOTHER CLIENT\n");
        ret = -1;
    } else if (password[usernum] == -1) { // Registration
        // printf("USER %d First Login: Registration Mode\n", usernum);
        password[usernum] = passwd;
        client2user[client2fd[client_sock]] = usernum;
        user2client[usernum] = client2fd[client_sock];
        ret = 1;
    } else if (password[usernum] != passwd) {
        // printf("Password INCORRECT\n");
        ret = -1;
    } else {
        // printf("USER %d at tid[%d] Login Success\n", usernum, client2fd[client_sock]);
        client2user[client2fd[client_sock]] = usernum;
        user2client[usernum] = client2fd[client_sock];
        ret = 1;
    }
    if (ret != -1) {
        printf("Login success\n");
    } else {
        printf("Login failed\n");
    }
    query q;
    q.user = ret;
    q.action = 1;
    q.seat = ret;
    write(client_sock, &q, sizeof(query));
    pthread_mutex_unlock(&(c2u_mutex[client2fd[client_sock]]));
    pthread_mutex_unlock(&(u2c_mutex[usernum]));
    return ret;
}

```

4. Reserve

진입하기 전에 c2u_mutex 및 u2c_mutex, seat_mutex 를 lock 하여 충돌을 방지한다.

클라이언트가 해당 유저에 로그인되지 않은 상태라면 fail

USER 가 해당 클라이언트에 접속중이지 않은 상태라면 fail

좌석범위를 초과하면 fail

이미 예약된 좌석일 경우 fail

client2user[client2fd[client_sock]] 에는 이 클라이언트에 유저가 접속하고 있는지의 여부를

체크하며 있다면 어느 유저가 이 클라이언트를 점유하고 있는지를 확인한다.

user2client[usernum] 은 해당 유저가 어느 클라이언트에 로그인되어 있는 상태인지를 확인한다.

이를 통해 권한이 부여된 클라이언트와 올바른 유저가 쿼리를 날리고 있는지를 확인할 수 있다.

그 외의 경우라면 좌석예약에 들어간다.

좌석예약은 seat[seatnum] 을 usernum 으로 초기화함으로써 성립한다.

만일 빈 좌석이라면 seat[seatnum] 의 값은 -1 이 된다.

종료하기 이전에 답신 query 를 클라이언트로 전송하며 unlock 이후 종료된다.

return value 는 성공시 -1, 성공시 성공한 seatnum 이다.

```
// return seatnum: SUCCESS, return -1: FAIL
int reserve(int client_sock, int usernum, int seatnum) {
    pthread_mutex_lock(&(c2u_mutex[client2fd[client_sock]]));
    pthread_mutex_lock(&(u2c_mutex[usernum]));
    pthread_mutex_lock(&(seat_mutex[seatnum]));
    // printf("RESERVE: usernum: %d, seatnum: %d\n", usernum, seatnum);
    int ret = -1;
    if (client2user[client2fd[client_sock]] != usernum) { // 이 클라이언트에 유저가 접속하지 않은 상태라면
        // printf("CLIENT %d HAS NO USER. LOGIN FIRST\n", client2fd[client_sock]);
        ret = -1;
    } else if (user2client[usernum] != client2fd[client_sock]) { // 이 유저가 이 클라이언트에 로그인 안되어있다면
        // printf("USER %d HAS NOT LOGGED IN. LOGIN FIRST\n", usernum);
        ret = -1;
    } else if (seatnum > 255 || seatnum < 0) {
        // printf("INVALID SEAT NUMBER %d: 0~255\n", seatnum);
        ret = -1;
    } else if (seat[seatnum] != -1) { // 예약되어있다면
        // printf("SEAT %d Already RESERVED. TRY ANOTHER\n", seatnum);
        ret = -1;
    } else {
        // printf("CLIENT %d reserved USER %d to SEAT %d\n", client2fd[client_sock], usernum, seatnum);
        seat[seatnum] = usernum;
        ret = seatnum;
    }
    if (ret != -1) {
        printf("Seat %d is reserved\n", seatnum);
    } else {
        printf("Reservation failed\n");
    }
    query q;
    q.user = ret;
    q.action = 2;
    q.seat = ret;
    write(client_sock, &q, sizeof(query));

    pthread_mutex_unlock(&(c2u_mutex[client2fd[client_sock]]));
    pthread_mutex_unlock(&(u2c_mutex[usernum]));
    pthread_mutex_unlock(&(seat_mutex[seatnum]));
    return ret;
}
```

5. Check reservation

진입하기 전에 c2u_mutex 및 u2c_mutex, seat_mutex 를 lock 하여 충돌을 방지한다.

client2user[client2fd[client_sock]] 에는 이 클라이언트에 유저가 접속하고 있는지의 여부를 체크하며 있다면 어느 유저가 이 클라이언트를 점유하고 있는지를 확인한다.

user2client[usernum] 은 해당 유저가 어느 클라이언트에 로그인되어 있는 상태인지를 확인한다. 이를 통해 권한이 부여된 클라이언트와 올바른 유저가 쿼리를 날리고 있는지를 확인할 수 있다.

좌석번호가 범위를 초과하지 않는지를 체크하고,

해당 User 앞으로 예약된 좌석이 몇 좌석이 있는지를 for 문을 돌면서 체크하며 cnt 를 ++ 시켜가면서 좌석 수를 확인한다.

한 좌석도 예약하지 못한 상태일 경우, fail 을 의미하는 -1 을 반환하며

그 외의 경우, query 회신 이후 cnt, send 배열을 따로 추가적으로 더 보냄으로써 클라이언트 측에서 몇 번 좌석들이 자신 앞으로 예약되었는지를 확인할 수 있도록 한다.

모든 동작이 완료되었다면, unlock 을 수행한다.

```

// return 1: SUCCESS, 정수열 전송되니까 버퍼 체크 1번 더 해야함., return 0: FAIL
int check_reservation(int client_sock, int usernum, int seatnum) {
    pthread_mutex_lock(&(c2u_mutex[client2fd[client_sock]]));
    pthread_mutex_lock(&(u2c_mutex[usernum]));
    pthread_mutex_lock(&(seat_mutex[seatnum]));
    // printf("CHECK_RESERVATION: seatnum: %d\n", seatnum);
    int ret;
    int cnt = 0;
    int buf[16][16];
    int send[MAXSEAT];
    if (client2user[client2fd[client_sock]] != usernum) { // 이 클라이언트에 유저가 접속하지 않은 상태라면
        // printf("CLIENT %d HAS NO USER. LOGIN FIRST\n", client2fd[client_sock]);
        ret = -1;
    } else if (user2client[usernum] != client2fd[client_sock]) { // 이 유저가 이 클라이언트에 로그인 안되어있다면
        // printf("USER %d HAS NOT LOGGED IN. LOGIN FIRST\n", usernum);
        ret = -1;
    } else if (seatnum > 255 || seatnum < 0) {
        // printf("INVALID SEAT NUMBER %d: 0~255\n", seatnum);
        ret = -1;
    } else {
        for (int r = 0; r < 16; r++) {
            for (int c = 0; c < 16; c++) {
                if (seat[16 * r + c] == usernum) {
                    buf[r][c] = 1;
                    send[cnt++] = 16 * r + c;
                } else {
                    buf[r][c] = 0;
                }
            }
        }
        send[cnt] = 0;
        if (cnt == 0) { // 아무 자리도 예약안했다면
            // printf("USER %d HAS NOT RESERVED ANY SEAT\n", usernum);
            ret = -1;
        } else { // write로 buf 전송
            ret = 1;
        }
    }
    query q;
    q.user = ret;
    q.action = 3;
    q.seat = ret;
    write(client_sock, &q, sizeof(query));
    if (ret != -1) {
        write(client_sock, &cnt, sizeof(int));
        write(client_sock, send, sizeof(send));
        printf("Reservation: ");
        for (int i = 0; i < cnt; i++) {
            printf("%d ", send[i]);
        }
    }
}

```

```
    query q;
    q.user = ret;
    q.action = 3;
    q.seat = ret;
    write(client_sock, &q, sizeof(query));
    if (ret != -1) {
        write(client_sock, &cnt, sizeof(int));
        write(client_sock, send, sizeof(send));
        printf("Reservation: ");
        for (int i = 0; i < cnt; i++) {
            printf("%d ", send[i]);
        }
        printf("\n");
    } else {
        printf("Reservation check failed\n");
    }

    pthread_mutex_unlock(&(c2u_mutex[client2fd[client_sock]]));
    pthread_mutex_unlock(&(u2c_mutex[usernum]));
    pthread_mutex_unlock(&(seat_mutex[seatnum]));
    return ret;
}
```

6. Cancel reservation

진입하기 전에 c2u_mutex 및 u2c_mutex, seat_mutex 를 lock 하여 충돌을 방지한다.

client2user[client2fd[client_sock]] 에는 이 클라이언트에 유저가 접속하고 있는지의 여부를 체크하며 있다면 어느 유저가 이 클라이언트를 점유하고 있는지를 확인한다.
user2client[usernum] 은 해당 유저가 어느 클라이언트에 로그인되어 있는 상태인지를 확인한다.
이를 통해 권한이 부여된 클라이언트와 올바른 유저가 퀴리를 날리고 있는지를 확인할 수 있다.

```
// return seatnum: SUCCESS, return 0: FAIL
int cancel_reservation(int client_sock, int usernum, int seatnum) {
    pthread_mutex_lock(&(c2u_mutex[client2fd[client_sock]]));
    pthread_mutex_lock(&(u2c_mutex[usernum]));
    pthread_mutex_lock(&(seat_mutex[seatnum]));
    // printf("CANCEL_RESERVATION: seatnum: %d\n", seatnum);
    int ret;
    if (client2user[client2fd[client_sock]] != usernum) { // 이 클라이언트에 유저가 접속하지 않은 상태라면
        // printf("CLIENT %d HAS NO USER. LOGIN FIRST\n", client2fd[client_sock]);
        ret = -1;
    } else if (user2client[usernum] != client2fd[client_sock]) { // 이 유저가 이 클라이언트에 로그인 안되어있다면
        // printf("USER %d HAS NOT LOGGED IN. LOGIN FIRST\n", usernum);
        ret = -1;
    } else if (seatnum > 255 || seatnum < 0) {
        // printf("INVALID SEAT NUMBER %d: 0~255\n", seatnum);
        ret = -1;
    } else {
        if (seat[seatnum] != usernum) {
            // printf("The Seat %d is NOT RESERVED by USER %d\n", seatnum, usernum);
            ret = -1;
        } else {
            seat[seatnum] = -1;
            ret = seatnum;
            // printf("The Seat %d SUCCESSFULLY CANCELLED by USER %d\n", seatnum, usernum);
        }
    }
    if (ret != -1) {
        printf("Seat %d is canceled\n", seatnum);
    } else {
        printf("Cancel failed\n");
    }

    query q;
    q.user = ret;
    q.action = 4;
    q.seat = ret;
    write(client_sock, &q, sizeof(query));

    pthread_mutex_unlock(&(c2u_mutex[client2fd[client_sock]]));
    pthread_mutex_unlock(&(u2c_mutex[usernum]));
    pthread_mutex_unlock(&(seat_mutex[seatnum]));
    return ret;
}
```

좌석번호가 범위를 초과하지 않는지를 체크하고,

해당 User 앞으로 예약된 좌석이 몇 좌석이 있는지를 for 문을 돌면서 체크하며 cnt 를 ++ 시켜가면서 좌석 수를 확인한다.

한 좌석도 예약하지 못한 상태일 경우, 또는 해당 좌석이 이미 공백인 상태일 경우 fail 을 의미하는 -1 을 반환하며

해당 유저앞으로 예약된 좌석이 맞다면, seat[seatnum] 을 -1 로 바꾼 뒤 return 을 통해 cancel 을 한다.

그리고 cancel 의 결과를 클라이언트로 전송한다.

모든 동작이 완료되었다면, unlock 을 수행한다.

7. Logout

진입하기 전에 c2u_mutex 및 u2c_mutex 를 lock 하여 충돌을 방지한다.

client2user[client2fd[client_sock]] 에는 이 클라이언트에 유저가 접속하고 있는지의 여부를 체크하며 있다면 어느 유저가 이 클라이언트를 점유하고 있는지를 확인한다.
user2client[usernum] 은 해당 유저가 어느 클라이언트에 로그인되어 있는 상태인지를 확인한다.
이를 통해 권한이 부여된 클라이언트와 올바른 유저가 쿼리를 날리고 있는지를 확인할 수 있다.

client2user[client2fd[client_sock]] 와 user2client[usernum] 을 -1 로 초기화함으로써 로그아웃을 구현한다.

그리고 로그아웃의 결과를 클라이언트로 전송한다.

모든 동작이 완료되었다면, unlock 을 수행한다.

```
// return 1: SUCCESS, return 0: FAIL
int logout(int client_sock, int usernum) {
    pthread_mutex_lock(&(c2u_mutex[client2fd[client_sock]]));
    pthread_mutex_lock(&(u2c_mutex[usernum]));
    int ret;
    // printf("LOGOUT: CLIENT %d USERNUM %d\n", client2fd[client_sock], usernum);
    if (client2user[client2fd[client_sock]] == -1) { // 이 클라이언트가 이미 로그아웃이면
        // printf("Already This CLIENT has been LOGGED OUT\n");
        ret = -1;
    } else if (user2client[usernum] == -1) { // 이 유저가 이미 로그아웃이면
        // printf("Already This USER has been LOGGED OUT\n");
        ret = -1;
    } else {
        client2user[client2fd[client_sock]] = -1;
        user2client[usernum] = -1;
        ret = 1;
    }

    if (ret != -1) {
        printf("Logout success\n");
    } else {
        printf("Logout failed\n");
    }

    query q;
    q.user = ret;
    q.action = 5;
    q.seat = ret;
    write(client_sock, &q, sizeof(query));

    pthread_mutex_unlock(&(c2u_mutex[client2fd[client_sock]]));
    pthread_mutex_unlock(&(u2c_mutex[usernum]));
    return ret;
}
```

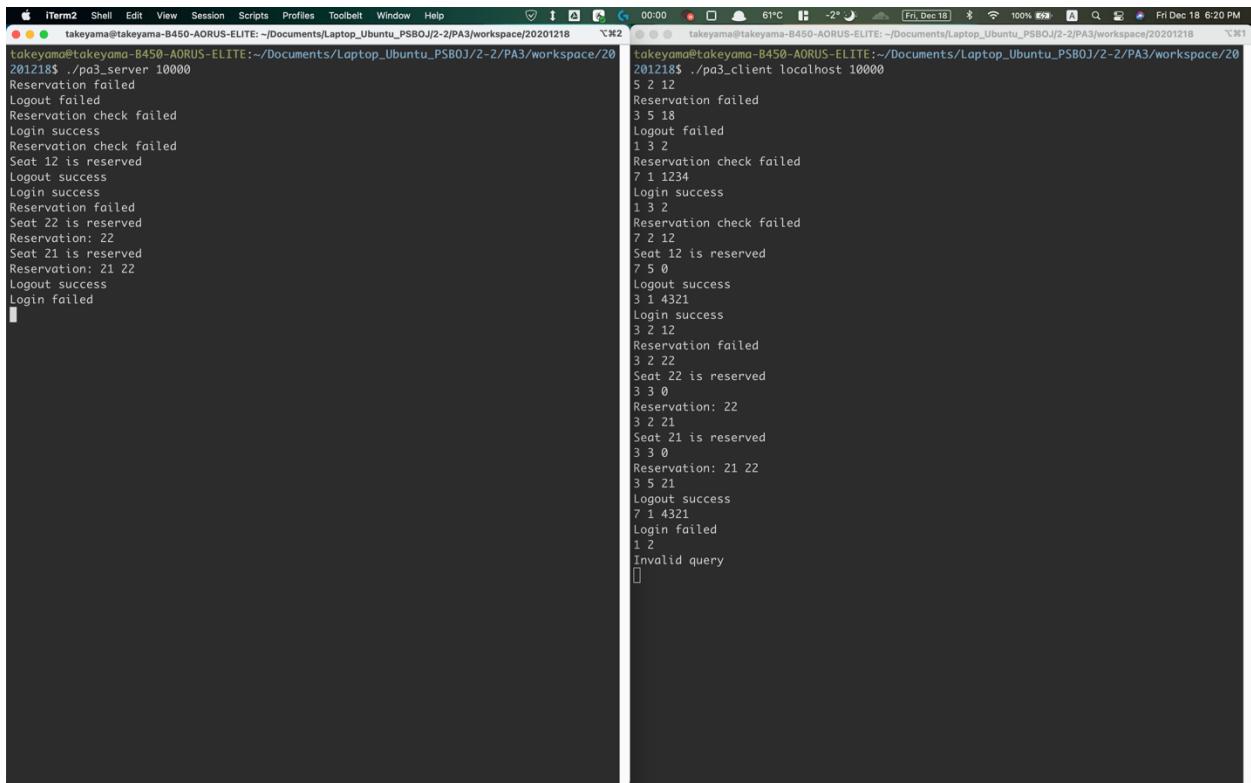

8. Exit

0 0 0 을 입력받는다면, exit 동작을 수행하는데
동일한 0 0 0 query 를 클라이언트로 반송하며
close(client_sock2) 를 통해 소켓을 닫고
return NULL 을 통해 해당 스레드를 최종적으로 종료시킨다.
이 때, thread_detach 되어있으므로 따로 join 은 필요하지 않고 자원은 자동반납된다.

```
289 void *thread_func(void *arg) {
290     int n;
291     query q;
292     int client_sock2 = *((int *)arg);
293     pthread_detach(pthread_self());
294     free(arg);
295     // printf("THREAD's CLIENT_SOCK2: %d\n", client_sock2);
296     while (1) {
297         if ((n = read(client_sock2, &q, sizeof(query))) < 0)) {
298             printf("read ERROR\n");
299             continue;
300         }
301         // printf("tid[%d] got q: %d %d %d\n", client_sock2 - 3, q.user, q.action, q.seat);
302         int mode = q.action;
303         int usernum = q.user;
304         int passwd = q.passwd;
305         int seatnum = q.seat;
306         int result;
307         switch (mode) {
308             case 0:
309                 if (q.user == 0 && q.action == 0 && q.seat == 0) { // CLIENT EXIT
310                     // printf("PLEASE DIE, tid[%d]\n", client_sock2 - 3);
311                     write(client_sock2, &q, sizeof(query));
312                     close(client_sock2);
313                     return NULL;
314                 }
315                 break;
316             case 1:
317                 result = login(client_sock2, usernum, passwd);
318                 break;
319             case 2:
320                 result = reserve(client_sock2, usernum, seatnum);
321                 break;
322             case 3:
323                 result = check_reservation(client_sock2, usernum, seatnum);
324                 break;
325             case 4:
326                 result = cancel_reservation(client_sock2, usernum, seatnum);
327                 break;
328             case 5:
329                 result = logout(client_sock2, usernum);
330                 break;
331             default:
332                 printf("switch ERROR\n");
333                 break;
334         }
335     }
336     close(client_sock2);
337     return NULL;
338 }
```

9. 실행결과

우선 PA3.pdf 5 페이지에 있는 단일 클라이언트에 대해 발생하는 쿼리문 상황의 예제를 실행시



The screenshot shows two terminal windows side-by-side. The left window is titled 'takeyama@takeyama-B450-AORUS-ELITE: ~/Documents/Laptop_Ubuntu_PSBOJ/2-2/PA3/workspace/20201218' and contains the output of the 'pa3_server' command. The right window is titled 'takeyama@takeyama-B450-AORUS-ELITE: ~/Documents/Laptop_Ubuntu_PSBOJ/2-2/PA3/workspace/20201218' and contains the output of the 'pa3_client' command. Both windows show a series of commands being issued and their corresponding responses.

```
takeyama@takeyama-B450-AORUS-ELITE:~/Documents/Laptop_Ubuntu_PSBOJ/2-2/PA3/workspace/20201218$ ./pa3_server 10000
Reservation failed
Logout failed
Reservation check failed
Login success
Reservation check failed
Seat 12 is reserved
Logout success
Login success
Reservation failed
Seat 22 is reserved
Reservation: 22
Seat 21 is reserved
Reservation: 21 22
Logout success
Login failed
[...]
takeyama@takeyama-B450-AORUS-ELITE:~/Documents/Laptop_Ubuntu_PSBOJ/2-2/PA3/workspace/20201218$ ./pa3_client localhost 10000
20201218$ Reservation failed
5 2 12
Logout failed
3 5 18
Reservation failed
3 5 2
Logout failed
7 1 1234
Reservation check failed
7 1 1234
Login success
1 3 2
Reservation check failed
7 2 12
Seat 12 is reserved
7 5 0
Logout success
3 1 4321
Login success
3 2 12
Reservation failed
3 2 22
Seat 22 is reserved
3 3 0
Reservation: 22
3 2 21
Seat 21 is reserved
3 3 0
Reservation: 21 22
3 5 21
Logout success
7 1 4321
Login failed
1 2
Invalid query
[]
```

좌가 서버, 우가 클라이언트일 때

	< Client > [user, action, data]	< Terminal >
These queries are not logged in user's query. So not accepted.	Input - 5, 2, 12 Input - 3, 5, 18 Input - 1, 3, 2	Reservation failed Logout failed Reservation check failed
User 7 login success with password '1234'	Input - 7, 1, 1234	Login success
Not logged in user's query. So failed	Input - 1, 3, 2	Reservation check failed
User 7 reserved seat 12	Input - 7, 2, 12	Seat 12 is reserved
Logged out	Input - 7, 5, 0	Logout success
User 3 login success with password '4321'	Input - 3, 1, 4321	Login success
User 7 already reserved seat 12. So failed	Input - 3, 2, 12	Reservation failed
User 3 reserved seat 21	Input - 3, 2, 21	Seat 21 is reserved
User 3 reserved seat 22	Input - 3, 2, 22	Seat 22 is reserved
User 3 check reservation	Input - 3, 3, 0	Reservation: 22, 23
Logged out	Input - 3, 5, 21	Logout success
Invalid user's password. So failed	Input - 7, 1, 4321	Login failed
Invalid query	Input - 1, 2	Invalid query

위와 같은 정확한 결과가 나오는 것을 확인할 수 있다.

다중 클라이언트 접속상황을 가정한 쿼리문 상황의 예제를 실행 시

```

takeyama@takeyama-B450-AORUS-ELITE:~/Documents/Laptop_Ubuntu_PSBOJ/2-2/PA3/workspace/2021218$ ./pa3_server 10000
bind() failed
takeyama@takeyama-B450-AORUS-ELITE:~/Documents/Laptop_Ubuntu_PSBOJ/2-2/PA3/workspace/2021218$ ./pa3_server 10001
Login success
Login failed
Login success
Seat 10 is reserved
Reservation failed
Seat 10 is canceled
Seat 10 is reserved
Logout success
Logout success
Login failed
Login success
Reservation check failed
Reservation failed
Reservation check failed
Seat 240 is reserved
Reservation failed
Seat 210 is reserved
Reservation: 210 240
[]

takeyama@takeyama-B450-AORUS-ELITE:~/Documents/Laptop_Ubuntu_PSBOJ/2-2/PA3/workspace/2021218$ ./pa3_client localhost 10001
5 1 1
Login failed
3 1 7
Login success
3 2 10
Reservation failed
3 2 10
Seat 10 is reserved
3 5 1
Logout success
5 1 11
Login failed
5 1 1
Login success
5 3 0
Reservation check failed
5 2 10
Reservation failed
5 3 0
Reservation check failed
5 2 240
Seat 240 is reserved
5 2 510
Reservation failed
5 2 210
Seat 210 is reserved
5 3 0
Reservation: 210 240
0 0 0
takeyama@takeyama-B450-AORUS-ELITE:~/Documents/Laptop_Ubuntu_PSBOJ/2-2/PA3/workspace/2021218$ []

```

좌상단이 서버, 좌하단이 Client1, 우측이 Client2

< Client 1 >

< Client 2 >

5, 1, 1

5, 1, 1

Failed because user 5 already logged in at Client 1

3, 1, 7

5, 2, 10

3, 2, 10

Failed because user 5 already reserved seat 10

5, 4, 10

3, 2, 10

5, 5, 1

3, 5, 1

5, 1, 11

Failed because user 5's password is 1

위와 동일하게 제대로 실행됨을 확인할 수 있고, 0 0 0 수신 시 클라이언트가 종료되는 것 또한 확인할 수 있다.