Programming Assignment #2

2017313150

허종원

1. 구조체

```
...

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

typedef struct{
  int login;
  int passCode;
  int reserved;
} user;

user users[MAX_CLIENTS];
int seats[MAX_SEATS];

...
```

과제 설명에서 주어진 query 구조체와 더불어 user 구조체를 사용했습니다.

User 구조체는 MAX_CLIENTS(1024)만큼의 크기를 할당되며 memset 함수를 통해 -1로 초기화합니다.

login – 유저가 로그인했는지 안했는지를 판별해주며 1일 경우 로그인된 상태로 인식합니다.

passcode – 유저의 패스코드를 저장하는 변수로 초기 값은 역시 -1입니다.

reserved – 한 유저는 한 좌석만 예약할 수 있기 때문에 예약을 성공했을 경우 그 좌석의 번호로 저장해줍니다.

2. 전반적인 흐름 및 코드 설명

우선 서버를 시작하고 소켓을 받아들일 수 있는 준비합니다. 물론 그과정에서 memset함수를 통해 users 변수를 초기화하는 과정도 진행합니다.

```
…
while(1){
    clientLen = sizeof(cli_addr);
   if((connFd = accept(serverSocket, (struct sockaddr*)&cli_addr, (socklen_t
    *)&clientLen))< 0){
      printf ("accept() failed.\n");
      continue;
    };
    pthread_mutex_lock(&clients_mutex);
    if(cli_count == MAX_CLIENTS ){
       printf("Max clients.\n");
      close(connFd);
       continue;
    }
    clients[cli_count++]=connFd;
    pthread_mutex_unlock(&clients_mutex);
    pthread_create(&tid, NULL, handle_client,(void *)&connFd);
    pthread_detach(tid);

}
  …
```

Main 함수의 while 문을 통해 연결을 받아들입니다.

받아들일 때 현재 접속해있는 Client의 수를 계산해야해서 mutex로 lock을 걸고 If 문을 통해 client의 수가 Max일 경우 연결을 종료시킵니다.

아닐 경우 clients 변수 각 연결된 client의 accept 값을 저장해줍니다. 이를 통해 나중에 서버가 특정조건에 의해 종료될 때 -1값을 보내주고 전체 연결을 종료시킵니다.

Lock을 푼 후 정상적으로 Thread를 생성합니다.

2-1. 함수

함수는 총 7개로 구성됩니다.

```
void *handle_client(void *args);  //쓰레드를 진입했을 때 다루는 함수
int check_login(query *q); //로그인 했는지 확인
int login(query *q); //로그인 action 1
int reserve(query *q); // 예약 action 2
int check_reserve(query *q); //예약 확인 action 3
int cancel_reserve(query *q); //예약 취소 action 4
int log_out(query *q); // 로그 아웃 action 5
int isFull(); //종료 조건, 모든 좌석이 예약됐는지 확인
```

Action의 경우 단순히 Mutex를 통해 Lock을 걸고 각변수를 체크하는 if 문만 존재하기 때문에 Thread Main 함수만 담겠습니다.

```
void *handle_client(void *args){
  int clint_sock=*((int*)args);
  query *q=(query *)malloc(sizeof(query));
  int receive,result;
  while((receive=recv(clint_sock, q, sizeof(query), 0))>0){
    if(q->action ==0 && q->user==0 && q->data==0){
      send(clint_sock,seats,sizeof(seats),0);
    }
    else if (q->action < 1 || q->action > 5){
      result = -1;
      send(clint_sock,&result,sizeof(result),0);
    }
    else if(q->user < 0 || q->user > 1023){
      result = -1;
      send(clint_sock,&result,sizeof(result),0);
    }
    if(q->action == 1){
      result= login(q);
    }
    else{
      result = check_login(q);
    }
    send(clint_sock,&result,sizeof(result),0);

    pthread_mutex_lock(&clients_mutex);
    if (isFull() == 1){
      for(int i = 0; i<cli_count; i++){
        int end=-1;
        send(clients[i],&end,sizeof(end),0);
        close(clients[i]);
      }
      exit(1);
    }
    pthread_mutex_unlock(&clients_mutex);
  }
```

```
  pthread_mutex_lock(&clients_mutex);
  for(int i=0; i<cli_count; i++)   {
    if(clint_sock == clients[i]){
      while(i++ < cli_count-1){
        clients[i]=clients[i+1];
      }
      break;
    }
  }
  cli_count--;
  pthread_mutex_unlock(&clients_mutex);
  free(q);
  close(clint_sock);
  pthread_exit(NULL);
}
```

우선 accept를 통해 반환된 socket discriptor를 인자로 받습니다. 그리고 반복해서 Client에게서 입력을 받습니다. 가장 먼저 Action과 User, Data가 모두 0인지를 확인하고 그 경우 좌석의 정보를 보내줍니다.

그 후 Action과 User의 데이터 범위의 유효성을 판단해준 뒤 각 Action에 맞는 함수로 전달해줍니다.

각 함수들은 Users 구조체에서 각각의 조건에 따라 데이터를 판별해준뒤 결과값을 result 변수에 전달해줍니다.

그 후 결과값을 Client에게 보내줍니다.

다시 반복문을 돌기 전에 IsFull() 함수를 통해 모든 좌석이 예약되었는지 판별합니다.

```
int isFull(){
  int result = 1;
  for(int i= 0; i<MAX_SEATS; i++){
    if(seats[i] == -1){
      result = -1;
    }
  }
  return result;
}
```

모든 좌석이 예약 되어있을 경우 1을 반환하고 한자리라도 -1 즉, 빈좌석이 있을경우 -1을 반환합니다. 만약 1일 경우 앞서 저장해준 clients 배열을 통해 모든 Clients 에게 -1을 반환한 뒤 연결을 종료하고 Sever 역시 종료합니다.

다른 케이스에 의해 종료될 경우 Clients 변수에서 그 Client를 삭제해준뒤 cli_count를 줄여주고 앞서 생성한 변수들 및 연결을 종료하고 Thread를 종료합니다.