REPORT

보고서 작성 서약서

1. 나는 타학생의 보고서를 복사(Copy)하지 않았습니다.
2. 나는 타학생의 보고서를 인터넷에서 다운로드 하여 대체하지 않았습니다.
3. 나는 타인에게 보고서 제출 전에 보고서를 보여주지 않았습니다.
4. 보고서 제출 기한을 준수하였습니다.

나는 보고서 작성시 위법 행위를 하지 않고,
성.균.인으로서 나의 명예를 지킬 것을 약속합니다.

과　　목 : 시스템 SW 실습 2
담당교수: 정진규
학　　과 : 반도체시스템공학과
학　　번 : 2018314788
이　　름: 오해성
제 출 일 : 2020.12.11

서 론

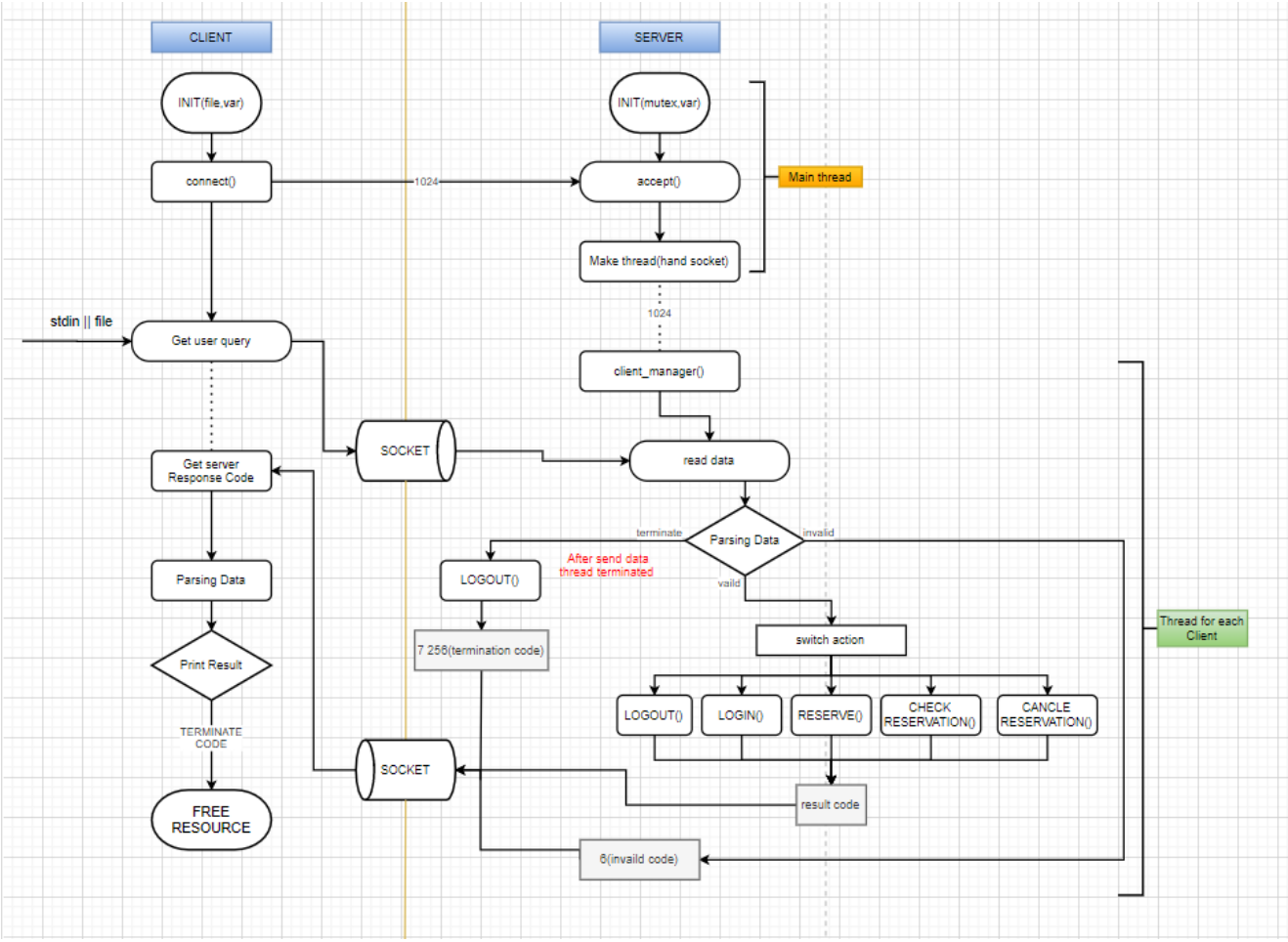
이 리포트는 시스템 소프트웨어 실습 2 3차 과제에 설계와 구현에 대한
간략한 설명을 담고있습니다,

순서는 다음과 같습니다,

1. 프로그램 전체 실행 흐름에 대한 설명
2. 클라이언트 프로그램에 대한 설명
    - 사용 자료형과 함수 선언
    - 클라이언트 메인 플로우
3. 서버 프로그램에 대한 설명
    - 사용 자료형과 함수 선언
    - 서버 메인 쓰레드 플로우
    - 서버 생성 쓰레드 플로우
    - 쿼리 처리 5가지 함수
4. 결론

본 론

Flow Chart



전체 흐름을 간단하게 도식화 하였으며
그림과 같이 클라이언트는 접속 후 stdin 이나 인자를 통해 쿼리를 받은 후
이를 그대로 서버에 전송해 주게 됩니다. (클라이언트 측에서는 결과값을 제외한 데이터를 처리하지 않습니다)
서버에서 전송된 쿼리를 처리 후 결과 코드를 (action result) 와 같이 넘겨주는데
이를 파싱 후 파싱한 결과를 토대로 알맞은 출력을 하게 됩니다.
이 때 종료 코드라면 아무것도 출력하지 않고 프로그램이 종료되게 됩니다.

서버 측에서는 최대 1024 까지의 클라이언트 접속이 가능하며
클라이언트의 접속이 발생할 때 마다 해당 클라이언트를 관리하는
쓰레드를 생성하여 이 쓰레드에서 유저 쿼리를 전송받고 처리 후 결과값을 클라이언트에게 전송하게 됩니다,

코드의 가독성을 신경써서 프로그램을 작성하였기 때문에
자세한 부분은 코드를 보고 직관적으로 이해할 수 있을 것 입니다,

클라이언트

클라이언트 사용 자료형,함수

쿼리의 가능한 종류 7가지 (유효쿼리,비유효쿼리,종료쿼리) 를 열거형으로 작성하였습니다.

서버에서 받은 데이터를 parsing 함수를 통해 result 구조체로 변경 뒤

print_result 함수에서 이를 받아 출력을 처리합니다.

parsing 함수에서 동적 할당 된 result 구조체는 print_result 함수에 끝에서 해제됩니다.

또한 36번줄,37번줄의 result 구조체 field 오로지 세번째 쿼리인 check_reservation 쿼리에만

사용됩니다.

```
13  typedef enum _ACTION
14  {
15      LOGIN = 1,
16      RESERVE,
17      CHECK_RESERVATION,
18      CANCLE_RESERVATION,
19      LOGOUT,
20      INVALID,
21      TERMINATE
22  }ACTION;
23
24  typedef enum _bool
25  {
26      false,
27      true
28  }bool;
29
30  typedef struct _result
31  {
32      ACTION action;
33      int val;
34
35      /* for result of check_reservation query */
36      bool data[MAX_SEATS];
37      int idx;
38  } result;
39
40  /* tool function define */
41  result * parsing(char str[],int n);
42  bool print_result(result *r);
```

클라이언트 메인 함수 동작

메인 함수에서는 인자가 4개인 경우 우측과 같이 입력을 받을 파일을 4번째 인자 파일로 설정합니다.

```
52      if (argc == 3) ifd = stdin;
53      else if (argc == 4) ifd = fopen(argv[3],"r");
54      else
55      {
56          printf("Follow the input rule below\n");
```

이후 서버에 소켓을 통해 연결한 후 우측과 같이 설정된 파일에서 계속해서 입력을 받고

서버에 소켓을 통해 전송한 뒤 결과값을 받아 parsing 후 print 하게 됩니다.

이때 파싱된 결과가 종료 코드에 해당하면 반복문을 종료하게 됩니다.

파일 입력의 경우 파일을 전부 읽을 때 까지 종료 쿼리가 발생하지 않았을 수 있기 때문에

추가적으로 101 ~ 105 라인을 통해 예외처리 하였습니다

```
85      while((n=getline(&(buf),&tli,ifd))!=-1)
86      {
87          write(client_socket,buf,n);
88          n = read(client_socket,buf,MAX_LINE);
89
90          // printf("\n %s \n",buf); // DEBUG
91
92          result * ret = parsing(buf,n);
93
94          if(print_result(ret))
95          {
96              term = true;
97              break;
98          }
99      }
100
101     if(ifd != stdin && !term)
102     {
103         write(client_socket,"0 0 0\n",7);
104         n = read(client_socket,buf,MAX_LINE);
105     }
106
107     if(ifd != stdin) fclose(ifd);
108     close(client_socket);
```

서 버

서버 사용 자료형,함수

서버의 10번줄부터 29번 까지는 클라이언트와 유사합니다.
이를 따로 헤더파일로 빼는게 좋아보입니다.

유저로부터 받은 데이터를 파싱하여 담는 query 구조체와
reservation 쿼리를 처리하기 위한 좌석 구조체를 설정하였습니다.

최대 클라이언트 접속 제한을 1024로 걸었기 때문에
클라이언트 접속과 종료시에 비어있는 tid 를 체크하기 위해
checker 배열과 overmutex 뮤텍스를 사용합니다.

또한 유저의 로그인 상태와 비밀번호 값을 관리하기 위해
user_ck,user_pw 배열을 사용하고
user_ck의 값이 0인 경우 최초 접속(계정 생성) 으로 처리하였습니다.

클라이언트로 부터 받은 데이터를 쿼리로 전환하는 parsing 함수와
쿼리의 유효성을 판단하기 위한 함수와 invaild 함수
종료 쿼리임을 확인하는 terminate 함수가 있고

나머지는 유저의 쿼리를 처리하기 위한 5가지 함수가 존재합니다.

```c
#define MAX_SEATS 256
#define MAX_CLIENT 1024
#define MAX_LINE 1024

/* static data define */
typedef enum _ACTION
{
    LOGIN = 1,
    RESERVE,
    CHECK_RESERVATION,
    CANCLE_RESERVATION,
    LOGOUT
}ACTION;

typedef enum _bool
{
    false,
    true
}bool;

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

typedef struct _seat
{
    int user_id;
    pthread_mutex_t mutex;
} seat;

pthread_t tid[MAX_CLIENT];

/* for client manage */
pthread_mutex_t overmutex;
bool checker[MAX_CLIENT];

/* for login manage */
pthread_mutex_t loginmutex;
int user_pw[MAX_CLIENT];
int user_ck[MAX_CLIENT];

/* for seat manage */
seat arr[MAX_SEATS];

/* tool function define*/
bool invalid(query q){return ((q.user < 0 || q.user > 1023) || (q.action < 1 || q.action > 5) || q.data==-1);}
bool terminate(query q){return !(q.action || q.data || q.user);}
query parsing(char str[],int n);

/* query action handle function */
bool login(query q,int id,int cfd);
void rev(query q,int cfd);
void check_rev(query q,int cfd);
void cancle_rev(query q,int cfd);
bool logout(query q,int id,int cfd);

```

서버 메인 쓰레드 동작

서버는 실행 후 필요한 변수들의 값을 초기화하고
서버 소켓을 생성 후 다음과 같은 반복문을 돌게됩니다.

클라이언트로 부터 접속이 들어오면
accept 함수를 통해 client 소켓을 받은 후
유저 쿼리를 처리하기 위한 쓰레드를 생성하고 소켓을 인자로 넘겨줍니다.

```c
    while(1)
    {
        caddrlen = sizeof(caddr);
        if((cfd = accept(serverSocket,(struct sockaddr *)&caddr,(socklen_t *)&caddrlen))<0) printf("accept() failed.\n");
        else
        {
            int d = -1;
            pthread_mutex_lock(&overmutex);
            while(++d<MAX_LINE && checker[d]);
            checker[d] =true;
            pthread_mutex_unlock(&overmutex);

            copyfd[d] = cfd;
            pthread_create(&tid[d],NULL,client_manager,(void*)&copyfd[d]);
        }
    }
```

이때 비어있는 tid 배열을 찾기 위해 overmutex 락을 건 후 반복문을 수행합니다.

또한 생성 쓰레드에 포인터로 소켓을 넘겨주게 되기 때문에

생성 쓰레드에 cfd 값이 할당되기 전에 accept 가 발생하여 cfd 값이 바뀌는 경우를

방지하기 위해 copyfd 라는 배열을 따로 선언하여 이 배열에 소켓 값을 복사한 후 주소를 넘겨주는 식으로

구현하였습니다.

생성 쓰레드 동작

인자로 받은 소켓을 복사하여 할당하고

함수 내부 연산을 위한 변수들을 설정 후

반복문 내에서 소켓을 통해 클라이언트의 데이터를 수신하게 됩니다.

id 변수의 경우 로그인 후 쓰레드 함수 내에서도 접속된 유저의 id 를 확인하기 위해 사용됩니다.

```
69  /* thread movement */
70  void * client_manager(void *arg)
71  {
72      int client_fd = *((int *)arg);
73      ssize_t n;
74      char buf[MAX_LINE];
75      int id = -1;
76
77      /* handle opr */
78      while((n = read(client_fd,buf,MAX_LINE)) > 0)
79      {
80          //write(STDOUT_FILENO,buf,n); // DEBUG
81
82          bool inv = false;
83          int spcheck = 0;
84          for(int i=0;i<n;++i)
85          {
86              spcheck += buf[i]==' ';
87              if((buf[i]>='0' && buf[i] <='9') || (buf[i]=='\n' || buf[i]=='\0' || buf[i] == ' ')) continue;
88              inv = true;
89          }
90
91          if(spcheck != 2) inv = true;
92
93          if(inv)
94          {
95              write(client_fd,"6",2);
96              continue;
97          }
98
99          query q = parsing(buf,n);
100
101         // printf("user : %d , action : %d, data : %d\n",q.user,q.action,q.data); // DEBUG
102
103         if(terminate(q))
104         {
105             if(id != -1)
106             {
107                 q.user = id;
108                 logout(q,id,client_fd);
109             }
110
111             write(client_fd,"7 256",6);
112             break;
113         }
114         else if(invalid(q))
115         {
116             write(client_fd,"6",2);
117             continue;
118         }
```

이때 82번 부터 91번 라인까지 쿼리로 전환할 수 있는 형식인지 판단하기 위해

일차적인 유효성 검증을 거치며 유효하지 않다면 이 결과를 클라이언트에 전달합니다.

이후 데이터를 파싱하여 query 구조체에 담은 후

종료코드인지 유효한 코드인지를 검증하게 됩니다.

종료코드의 경우 유저가 접속되있는 상태이면 logout 후

클라이언트에게 결과값을 보내고 반복문을 벗어나게 됩니다.

만약 유저가 접속되있지 않다면 로그인 액션을 제외한 모든

액션은 fail 처리되며 유저가 접속해있다면 해당 유저의 쿼리를 제외한

모든 쿼리는 fail 처리 되기 때문에

이를 121번 라인부터 127번 라인에서 사전에 처리해주게 됩니다.

```
120
121         if((id!=-1 && id!=q.user) || (id == -1 && q.action != LOGIN))
122         {
123             char tmp[30];
124             int t = sprintf(tmp,"%d -1",q.action)+1;
125             write(client_fd,tmp,t);
126             continue;
127         }
128
129         switch(q.action)
130         {
131             case LOGIN:
132                 if(login(q,id,client_fd)) id = q.user;
133                 break;
134             case RESERVE:
135                 rev(q,client_fd);
136                 break;
137             case CHECK_RESERVATION:
138                 check_rev(q,client_fd);
139             break;
140             case CANCLE_RESERVATION:
141                 cancle_rev(q,client_fd);
142             break;
143             case LOGOUT:
144                 if(logout(q,id,client_fd)) id = -1;
145             break;
146             default:
147                 write(client_fd,"6",2);
148             break;
149         }
150
151     }
152
153     pthread_mutex_lock(&overmutex);
154     checker[client_fd] =false;
155     pthread_mutex_unlock(&overmutex);
156
157     return NULL;
158 }
```

이후 switch 문을 통해

해당하는 액션의 함수를 호출합니다.

각 함수는 쿼리를 처리한 후 결과값을

소켓을 통해 클라이언트에게 전달합니다.

쿼리 처리 함수

로그인 관련 함수

유저의 로그인 쿼리의 경우의 수는

- 해당 유저 id의 접속이 한번도 발생하지 않은 경우
- 해당 유저 id에 이미 접속된 클라이언트가 있는 경우
- 해당 유저 id에 접속된 클라이언트가 없는 경우

다음과 같이 나뉩니다.

로그인 함수에서는 user_ck 의 값을 통해 이 세가지 경우를 먼저 판별 후

success 결과값을 통해 클라이언트에게 결과를 전달합니다.

로그아웃 함수에서는 간단히 user_ck 값을 -1로 설정하여

다른 유저가 로그인 할 수 있는 상태임을 나타냅니다.

이 두 함수는 쓰레드간 공유하는 변수인 user_ck 와 user_pw에 접근하기전

미리 설정된 loginmutex로 상호배제 상태를 만드는 것을 확인할 수 있습니다.

```
249  bool login(query q,int id,int cfd)
250  {
251      bool success = false;
252
253      pthread_mutex_lock(&loginmutex);
254
255      int status = user_ck[q.user];
256
257      switch (status)
258      {
259      case -1: // can login
260          success = (q.data == user_pw[q.user]);
261          if(success) user_ck[q.user] = 1;
262          break;
263      case 0: // sign up
264          user_pw[q.user] = q.data;
265          user_ck[q.user] = 1;
266          success = true;
267          break;
268      case 1: // can't login
269      default:
270          break;
271      }
272
273      pthread_mutex_unlock(&loginmutex);
274
275      if(success) write(cfd,"1 1",4);
276      else write(cfd,"1 -1",5);
277
278      return success;
279  }
280
281  bool logout(query q,int id,int cfd)
282  {
283      pthread_mutex_lock(&loginmutex);
284      user_ck[q.user] = -1;
285      pthread_mutex_unlock(&loginmutex);
286
287      if(q.action == LOGOUT) write(cfd,"5 1",4);
288
289      return true;
290  }
```

예약 관련 함수

유효한 좌석의 범위는 0부터 255 까지이기 때문에
두 함수 모두 쿼리의 데이터 필드가 유효한지 먼저 판단합니다.

예약 관련 함수의 경우 해당 좌석 정보에 접근하기 전
해당하는 좌석의 뮤텍스를 먼저 걸게 됩니다.

유저의 id 값은 음수일 수 없으므로
좌석의 user_id 값이 -1로 설정된 경우 빈좌석임을 나타냅니다.
따라서 예약 함수는 빈자리이면 본인의 id로 예약하고

예약 취소 함수는 해당 자리의 id가 본인의 id 와 같으면
좌석 상태를 빈좌석으로 만듭니다.

```
292  void rev(query q,int cfd)
293  {
294      if(q.data < 0 || q.data > 255) // overflow
295      {
296          write(cfd,"2 -1",5);
297          return;
298      }
299
300      bool ret = false;
301
302      pthread_mutex_lock(&arr[q.data].mutex);
303      ret = (arr[q.data].user_id==-1);
304      if(ret) arr[q.data].user_id = q.user;
305      pthread_mutex_unlock(&arr[q.data].mutex);
306
307      char tmp[30];
308      int t = sprintf(tmp,"2 %d",q.data)+1;
309      if(ret) write(cfd,tmp,t);
310      else write(cfd,"2 -1",5);
311  }
312
313  void cancle_rev(query q,int cfd)
314  {
315      if(q.data < 0 || q.data > 255) // overflow
316      {
317          write(cfd,"4 -1",5);
318          return;
319      }
320
321      bool ret = false;
322
323      pthread_mutex_lock(&arr[q.data].mutex);
324      ret = (arr[q.data].user_id == q.user);
325      if(ret) arr[q.data].user_id = -1;
326      pthread_mutex_unlock(&arr[q.data].mutex);
327
328      char tmp[30];
329      int t = sprintf(tmp,"4 %d",q.data)+1;
330      if(ret) write(cfd,tmp,t);
331      else write(cfd,"4 -1",5);
332
333      return;
334  }
```

예약 확인 함수

모든 좌석 중 본인의 좌석인지 상태를 나타내기 위한 ckr 배열과
한 좌석도 예약된 좌석이 없으면 fail 을 반환해야 하기 때문에 이를 나타내기 위한 fck 변수를 사용합니다.

모든 시트를 돌면서 본인의 좌석인지 확인 후
ckr 배열에 이 정보를 업데이트 합니다.

이후 한 좌석이라도 예약된 좌석이 있는 경우
각 좌석의 정보를 순서대로 0 과 1로 나타내 유저에게 전송합니다.

```
336  void check_rev(query q,int cfd)
337  {
338      bool ckr[MAX_SEATS];
339      bool fck = false;
340
341      for(int i=0; i<MAX_SEATS;++i)
342      {
343          pthread_mutex_lock(&arr[i].mutex);
344          ckr[i] = (arr[i].user_id == q.user);
345          fck |= ckr[i];
346          pthread_mutex_unlock(&arr[i].mutex);
347      }
348
349      if(!fck) write(cfd,"3 -1",5);
350      else
351      {
352          char str[MAX_LINE] = "3 ";
353
354          for(int i=0;i<MAX_SEATS;++i)
355          {
356              if(ckr[i]) strcat(str,"1 ");
357              else strcat(str,"0 ");
358          }
359
360          write(cfd,str,strlen(str)+1);
361
362      }
363  }
```

결 론

이번 과제에서는 간단한 서버를 구현하였습니다.

서버 관련 지식이 없어 여러 프로젝트 중에 헤메었던 적이 많은데
이번 과제를 진행하면서 서버가 어떠한 식으로 동작하는지 감을 잡을 수 있었습니다,

제 프로그램에서는 최대한 클라이언트 쪽 연산을 줄이고
서버 쪽에서 처리하도록 설계하였습니다.

또한 클라이언트가 서버에 보내는 데이터는 가공하지 않으며
서버 쪽에서 가공하게 되고 서버 쪽에서 클라이언트에 보내게 되는 데이터는
최대한 간결하게 1부터 7까지 나타나는 action 값과 나머지 결과값을 ascii 값으로 전달하였습니다.

c로 구현하다 보니 효율적인 자료구조를 사용하기 보다는
배열과 for 문으로 문제를 해결한 지점이 있는데
실제 서버는 성능적인 고려가 중요하므로 균형 이진 트리나 해싱을 활용한 map으로
자료들을 관리하는 것이 좋아 보입니다.