시스템SW실습2 PA4

2016310936 우승민

이번 과제는 PA2 에 진행했던 DB 과제를 사용하여 client 에서 값을 원하는 key 를 넘겨주면 server 에서 응답하여 답변하게 만드는 과제입니다.

먼저 client 의 코드를 보면 main 문에서 stdin 으로 받은 내용을 각각 **host** 와 **port** 로 입력받아 서버와 연결을 시켜주고,

```
int main (int argc, char *argv[]) {

        pthread_t tid;
        void *thread_result;
        int n;
        char buf[MAXLINE];
        struct hostent *h;
        struct sockaddr_in saddr;

        char *host = argv[1];
        int port = atoi(argv[2]);

        if ((cfd= socket(AF_INET, SOCK_STREAM, 0)) < 0) {
                printf("socket() failed.₩n");
                exit(1);
        }
        if ((h = gethostbyname(host)) == NULL) {
                printf("invalid hostname %s₩n", host);
                exit(2);
        }

        bzero((char *)&saddr, sizeof(saddr));
        saddr.sin_family= AF_INET;
        bcopy((char *)h->h_addr, (char *)&saddr.sin_addr.s_addr, h->h_length);
        saddr.sin_port= htons(port);

        if (connect(cfd,(struct sockaddr*)&saddr,sizeof(saddr)) < 0) {
                printf("connect() failed.₩n");
                exit(3);
        }

        pthread_create(&tid, NULL, thread, NULL);

        pthread_join(tid, &thread_result);

        close(cfd);
        return 0;
}
```

그 후 새로운 thread 를 만들어줍니다.

thread 함수에서는 가장 먼저 STDIN 으로 입력 받은 내용을 write 함수를 통해 server 로 전달하는 과정을 진행하게 해주었습니다.

```
void *thread(void *arg){
        int n;
        char bf[MAXLINE];
        char buf[MAXLINE];
        char a[1];
        pthread_detach(pthread_self());
        while(1){
                int i=0;
                while((n=read(0,a,1))!=-1){
                        if(a[0]=='₩n'||a[0]=='₩0')
                                break;
                        bf[i++]=a[0];
                }
                if(i!=0){
                        bf[i]='₩n';
                        bf[i+1]='₩0';
                        write(cfd,bf,i+1);
                }
```

'₩n'가 나올 때까지 한 글자씩 받아 bf 에 저장하여 보내주게 해주었습니다.

```
                n=read(cfd,buf,MAXLINE);
                if(strncmp(buf,"BYE",3)==0){
                        write(1,buf,n);
                        break;
                }
                if(strncmp(buf,"Too many client₩n",13)==0){
                        break;
                }
                write(1,buf,n);
        }

        close(cfd);
        return NULL;
}
```

그 이후 server 에 보낸 내용에 대한 응답을 받게 해주었습니다. 만약 server 로부터 연결이 종료되었다는 **"BYE"** 와, server 에서 지정한 것보다 많은 수의 client 가 연결되었다는 **"Too many client"** 응답을 받으면 while 문을 종료하여 함수를 끝내게 됩니다.

그 외에 **"GET"이나 "PUT", "CONNECT_OK"**와 같은 구문을 server 로부터 받으면 STDOUT 으로 write 하게 해주었습니다.

server 로 넘어가서 main 함수를 보면

```
int main ( int argc , char * argv [])
{
        signal(SIGINT,end);
        signal(SIGTSTP,end);
```

가장 우선적으로 **SIGINT** 와 **SIGTSTP** 의 signal 을 받을 때 강제로 종료가 되는데 이럴 경우에는 메모리가 free 되지 않고 thread 도 다 종료되지 않음으로 **end** 라는 함수에서 처리해주게 하였습니다.

```
void end(int sig){
        for(int i=0; i<idx; i++){
                pthread_cancel(tid[i]);
                pthread_join(tid[i],NULL);
                close(connfdp[i]);
        }
        db_close(DB);
        close(listenfd);
        free(connfdp);
        exit(0);

}
```

End 함수는 **pthread_cancel** 함수를 통해 각 thread 에 종료신호를 보내고 pthread_join 함수로 종료될때까지 기다리게 해주었습니다.

그 후 **db_close** 를 통해 자원을 해제한 후 현재까지 진행된 상황을 파일로 저장하게 만들어주었습니다.

여기서 DB, tid, listenfd 는 전체적으로 함수에 사용할 수 있도록 전역변수로 해주었습니다.

```
#include "db.h"

#define MAXLINE 80

db_t *DB;

int N=0;
int *connfdp;
int listenfd;
int idx;
int connfd;
int size;
int x=0;
int b=0;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t w = PTHREAD_MUTEX_INITIALIZER;
pthread_t tid[10];
```

다시 main 함수로 돌아가면, 사용할 변수를 초기화 해주고, server 에 입력받은 **port**, 접속 가능한 최대 client 수(**x**), DB 의 **size** 의 수를 받아줍니다. 그 후 DB 를 open 해줍니다.

```
int caddrlen ;
struct sockaddr_in saddr, caddr ;
int port =atoi ( argv [1]);
size = atoi(argv[3]);
x = atoi(argv[2]);

DB = db_open(size);
```

```
if (( listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf ("socket() failed.₩n");
        exit(1);
}

bzero ((char *)& saddr, sizeof ( saddr ));
saddr.sin_family = AF_INET;
saddr.sin_addr.s_addr = htonl (INADDR_ANY);
saddr.sin_port =htons (port);

if (bind( listenfd , ( struct sockaddr *)& saddr, sizeof ( saddr )) < 0) {
        printf ("bind() failed.₩n");
        exit(2);
}


if (listen( listenfd , 5) < 0) {
        printf ("listen() failed.₩n");
        exit(3);
}
connfdp = (int *)malloc(sizeof(int)*10);
```

이 부분은 수업때 사용하였던 자료를 활용하여 작성하였고, 다만 connfdp 는 여러 client 를 따로 나타내기 위해 포인터로 만든 것 이므로, int 형으로 여러 개 할당해주었습니다.

```
while (1) {

        caddrlen = sizeof ( caddr );
        if (( connfd = accept( listenfd , ( struct sockaddr *)& caddr , & caddrlen )) < 0) {
                printf ("accept() failed.₩n");
                continue;
        }

        pthread_mutex_lock (&lock);
        connfdp[idx] =connfd;
        pthread_mutex_unlock (&lock);
        if(idx < x)
                pthread_create(&tid[idx], NULL,thread, connfdp );
        else{
                printf("Too many client₩n");
                if(write(connfdp[idx], "Too many client₩n", 16));
                close(connfdp[idx]);
        }

}
```

While 문 안에서 각각의 client 와 접속하게 만들어 주었습니다. 여기서 **idx** 는 현재 연결된 client 수를 나타내고 **x** 보다 작으면 thread 를 새로 만들게 해주었고, 아니면 최대 client 접속 수를 초과한 것이므로, **"Too many client"**라는 error 메시지를 화면에 띄우며 client 에 보내게 해주었습니다.

```
        close(listenfd);
        printf ("connection terminated.₩n");
        db_close(DB);
        free(connfdp);
        return 0;
}
```

while 문 이후로는 열어준 것을 전부 닫고 main 문이 종료되게 해주었습니다.

```
void *thread(void *c){
        char key[MAXLINE];
        char buf[MAXLINE];
        int n;
        int cnt, buf_len, val_len;
        pthread_mutex_lock (&lock);
        int my = idx;
        idx++;
        pthread_mutex_unlock (&lock);
        pthread_detach(pthread_self());
        int i=0;
        int k=0;
        char *val;
```

thread 함수에서 대부분의 변수는 db 함수들에 사용할 목적으로 만들어 주었고, my 는 연결된 client 를 나타내는 목적으로 만들어주었습니다. 다른 client 와 겹치는 일이 없도록 lock 을 걸어주었습니다.

Client 와 접속할 때 가장 먼저해야 하는 일은 client 로부터 **"CONNECT"**라는 신호를 받는 것입니다. 그렇기에 다른 어떤 단어가 오더라고 **"CONNECT"**라는 문구가 오지 않으면 while 문을 계속 돌아 아래로 내려가지 않게 해주었습니다.

```
while ((n = read( connfdp[my], key, MAXLINE)) > 0)
        {
        if(strncmp(key,"CONNECT₩n",7)==0){
                key[7]='₩0';
                strcat(key,"_ok₩n");
                                if(write(connfdp[my], key, n+3));
                break;
        }
}
```

그 이후로는 client 와 접속이 완료되었으므로 client 로부터 모든 응답을 받을 수 있게 해주었습니다.

```
while ((n = read( connfdp[my], key, MAXLINE)) > 0)
        {
        pthread_mutex_lock (&lock);
        b++;
        if(b==1)
                pthread_mutex_lock (&w);
        pthread_mutex_unlock (&lock);
        if(strncmp(key,"DISCONNECT₩n",10)==0){
                strcpy(key,"BYE₩n");
                if(write(connfdp[my], key, 4));
                idx--;
                return NULL;
        }
```

여기서 client 간의 간섭이 생기어 key 가 겹치는 상황을 방지하기 위해서 lock 을 걸어주었습니다.

만약 **"DISCONNECT"**라는 신호를 받으면 **"BYE"**라는 신호를 client 에 보내고 접속중인 client 가 하나 줄어들게 된 것이므로 **idx** 를 1 줄여주었습니다.

```
else{
        strcpy(buf,key);
        buf[n]='₩0';
        i=0;
        k=0;
        while(buf[i]!='₩0'){
                if(buf[i]==' ')
                        break;
                i++;
        }
        if(i==n){
                if(write(connfdp[my],"UNDEFINED PROTOCOL₩n", 19));
                continue;
        }
```

Key 는 계속 받은 신호를 파악하기 위해 건드리지 않고, buf 에 옮겨서 사용하였습니다. 신호를 받을 때 만약 띄어쓰기가 없이 입력을 받았다면 형식에서 벗어난 구문이므로 **"UNDEFINED PROTOCOL"**을 client 에 보내주게 해주었습니다. (n 은 받은 구문의 크기이므로)

```
i=i+2;
while(i!=n)
        buf[k++]=buf[i++];
buf[k-1]='₩0';
buf_len = strlen(buf);
```

이후에 buf 에 **']'**괄호와 띄어쓰기 부문을 넘기기 위해 i 를 2 증가해주어, **GET [apple] -> apple]** 처럼 buf 에 담아 주었습니다.

```
if(strncmp(key,"GET",3)==0){
        char v[15];
        buf[k-2]='₩0';
        buf_len--;
        val = db_get(DB, buf, buf_len, &val_len);
        if (val == NULL) {
                if(write(connfdp[my],"GETINV₩n", 7));
        } else{
                strcpy(key, "GETOK [");
                strcat(key, buf);
                sprintf(v, "[%d]₩n", *((int *)val));
                strcat(key, "] ");
                strcat(key, v);
                if(write(connfdp[my], key, strlen(key)));
        }
}
```

현재 buf 에 apple]과 같이 저장되어 있으므로 마지막 괄호를 지워주고, db_get 함수로 apple 의 현재 입력받은 값을 **val** 로 저장하게 해주었습니다.

만약 **val** 이 NULL 이면 값이 없는 것이므로 **"GETINV"**를 출력해주고, 아니면, **val** 값을 sprintf 를 통해 출력할 수 있게 해주었습니다.

```
else if(strncmp(key,"PUT",3)==0){
        val = (char*)malloc(sizeof(int));
        i=0;
        k=0;
        char a[15];
        while(buf[i]!='₩0'){
                if(buf[i]==' ')
                        break;
                i++;
        }
        int t =i;
        if(i==buf_len){
                if(write(connfdp[my],"UNDEFINED PROTOCOL₩n", 19));
                continue;
        }
        i=i+2;
        while(buf[i]!=']')
                a[k++]=buf[i++];
        buf[t-1]='₩0';
        a[k]='₩0';
        *(int*)val = atoi(a);
        cnt = *((int*)val);
        db_put(DB, buf,strlen(buf), (char *)&cnt, sizeof(int));
        if(write(connfdp[my], "PUTOK₩n",6));
}
```

Key 에 **"PUT [apple] [4]"**을 입력 받았으면 현재 buf 가 **"apple] [4]"**로 입력되어 있습니다.

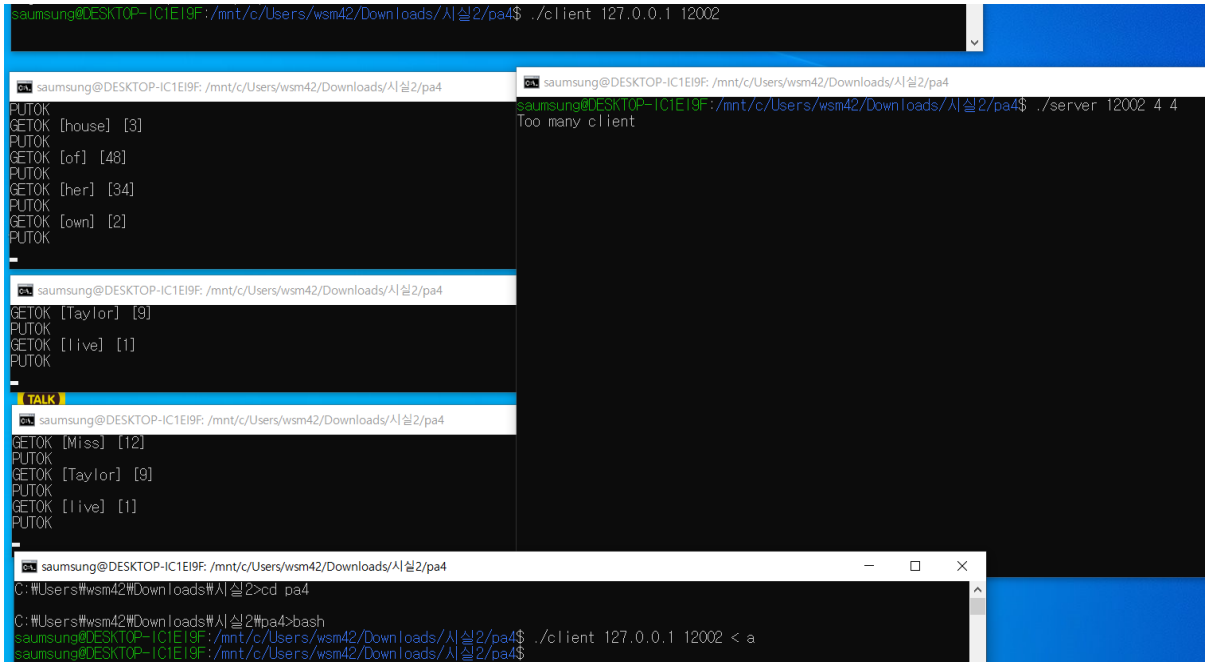
따라서 다시 띄어쓰기로 구분하여 "apple]", "[4]"를 구분하고 각각의 괄호를 지워준 후 buf 에는 단어만 a 에는 숫자만 입력되게 해주었습니다. 그 후 **cnt** 에 db_put 함수에 맞는 형태로 넣어주었습니다.

```
                        else{
                                if(write(connfdp[my],"UNDEFINED PROTOCOL₩n", 19));
                                continue;
                        }


                }
                pthread_mutex_lock (&lock);
                b--;
                if(b==0)
                        pthread_mutex_unlock (&w);
                pthread_mutex_unlock (&lock);
        }
        free(val);
        return NULL;
}
```

이외의 경우에는 전부 허용되지 않는 명령어이므로 다시 client 에서 명령어를 받습니다.

While 문 가장 아래에서 lock 을 다시 해제해준 후 함수 마지막에는 자원을 해제해준 후 thread 함수를 종료합니다.



4 개에 5 개를 접속시도를 하면 server 는 **"Too many client"**를 출력하고, 추가하려던 client 는 바로 종료됩니다.