<Program Assignment4 Report>

2016311821 한승하

본론에 들어가기 앞서, 본 과제를 진행하며 지난 PA2의 db.c 와 db.h 파일 그리고 실습시간에 진행한 Socket Programming 코드의 많은 부분을 사용하였음을 알려드리며, 본 보고서는 수정, 추가한 부분만 서술되었음을 말씀드립니다.

<Client>

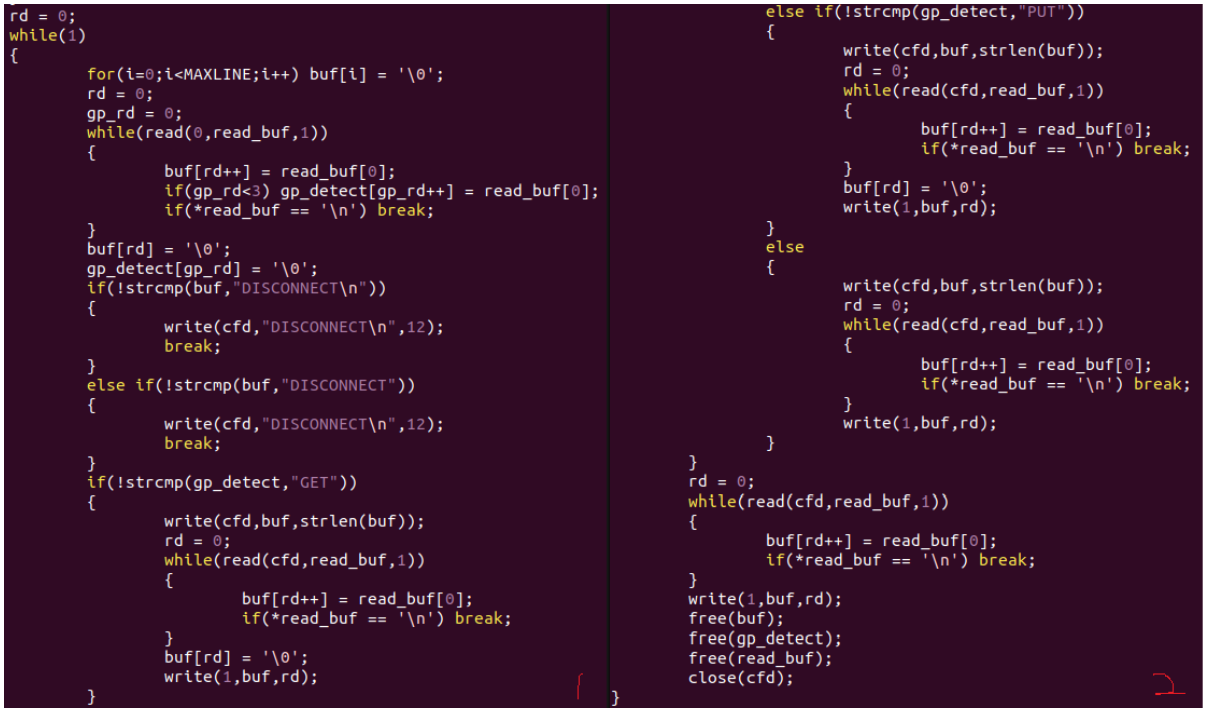
```c
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "unistd.h"
#include <pthread.h>

#define MAXLINE 80
int cfd;

int main (int argc, char *argv[]) {
        int n,rd,gp_rd,i;
        struct hostent *h;
        struct sockaddr_in saddr;
        char* buf = (char*)malloc(sizeof(char)*MAXLINE);
        char *gp_detect = (char*)malloc(sizeof(char)*4);
        char *read_buf = (char*)malloc(sizeof(char));
        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);
        }
```

본 과제에서 사용된 client.c 코드입니다. 위 사진에 첨부된 부분은 이미 지난 실습 시간에 다룬 내용으로써, client와 server의 연결을 보여주고 있습니다.

```
while ((n = read(cfd, buf, 1)) > 0) {
        if(*buf == 'F')
        {
                printf("TOO MANY CLIENTS!! ( CONNECTION DENIED ) \n");
                exit(0);
        }
        else if(*buf == 'C') break;
}
int stops = 0;
while(1)
{
        rd = 0;
        while(read(0,read_buf,1))
        {
                buf[rd++] = read_buf[0];
                if(*read_buf == '\n') break;
        }
        buf[rd] = '\0';
        if(!strcmp(buf,"CONNECT\n")) stops = 1;
        write(cfd,buf,rd);
        rd = 0;
        while(read(cfd,read_buf,1))
        {
                buf[rd++] = read_buf[0];
                if(*read_buf == '\n') break;
        }
        write(1,buf,rd);
        if(stops) break;
}
```

위 사진이 이번 과제 추가한 코드의 시작 부분입니다. Connect가 성공한 client는 server로부터 char형 한글자의 확인 응답을 기다립니다. 후에 서술할 server에서 연결되어 있는 client수를 판단하여 연결할 수 있는 경우 break하여 진행을, 그렇지 않을 경우 에러 메시지를 출력하고 종료합니다.

이후 client는 connect입력을 대기하는 상태로 들어갑니다. stdin으로 받은 입력을 new_line을 기준으로 striping하여 (Redirection input을 고려) Connect가 아닌 다른 입력을 하였을 경우, server로부터 undefined protocol 메시지를 받아 출력할 것이고, connect일 경우 connect_ok를 받아 출력한 후 break로 connection phase를 빠져나갑니다.



```
rd = 0;
while(1)
{
        for(i=0;i<MAXLINE;i++) buf[i] = '\0';
        rd = 0;
        gp_rd = 0;
        while(read(0,read_buf,1))
        {
                buf[rd++] = read_buf[0];
                if(gp_rd<3) gp_detect[gp_rd++] = read_buf[0];
                if(*read_buf == '\n') break;
        }
        buf[rd] = '\0';
        gp_detect[gp_rd] = '\0';
        if(!strcmp(buf,"DISCONNECT\n"))
        {
                write(cfd,"DISCONNECT\n",12);
                break;
        }
        else if(!strcmp(buf,"DISCONNECT"))
        {
                write(cfd,"DISCONNECT\n",12);
                break;
        }
        if(!strcmp(gp_detect,"GET"))
        {
                write(cfd,buf,strlen(buf));
                rd = 0;
                while(read(cfd,read_buf,1))
                {
                        buf[rd++] = read_buf[0];
                        if(*read_buf == '\n') break;
                }
                buf[rd] = '\0';
                write(1,buf,rd);
        }
        else if(!strcmp(gp_detect,"PUT"))
        {
                write(cfd,buf,strlen(buf));
                rd = 0;
                while(read(cfd,read_buf,1))
                {
                        buf[rd++] = read_buf[0];
                        if(*read_buf == '\n') break;
                }
                buf[rd] = '\0';
                write(1,buf,rd);
        }
        else
        {
                write(cfd,buf,strlen(buf));
                rd = 0;
                while(read(cfd,read_buf,1))
                {
                        buf[rd++] = read_buf[0];
                        if(*read_buf == '\n') break;
                }
                write(1,buf,rd);
        }
    }
    rd = 0;
    while(read(cfd,read_buf,1))
    {
            buf[rd++] = read_buf[0];
            if(*read_buf == '\n') break;
    }
    write(1,buf,rd);
    free(buf);
    free(gp_detect);
    free(read_buf);
    close(cfd);
}
```

이후 코드는 disconnect, put, get 명령을 탐지하는 phase입니다. 위와 마찬가지로 newline을 기준으로 striping하여 strcmp로 각각 disconnect인지, GET으로 시작하는지, PUT으로 시작하는지 (GET, PUT에 대한 탐지는 별도의 gp_detect 버퍼를 사용하여 3글자만을 비교할 수 있게 하였습니다.) 를 탐색합니다. (Disconnect에 대한 탐지는 redirection의 경우 맨 마지막 줄일 경우를 고려하여 newline이 있는 상황과 없는 상황 모두 탐지되도록 하였습니다.) 이후 아래와 같이 행동합니다.

1) DISCONNECT

   DISCONNECT이 들어오면, client는 phase를 빠져나간 후, server에서 종료 메시지를 받아 출력한 후 ("BYE") 사용한 자원을 해지하고 연결을 종료합니다.

2) GET, PUT

   PUT, GET의 경우 client는 서버에게 GET 의 전문을 전송하고, SERVER의 입력을 받아 출력한 후 다시 stdin 입력을 기다립니다 (while문 맨 위로 올라갑니다)

3) Else

   위 세가지 경우가 아닌 경우, Server에 전문을 전송 후, UNDIFINED PROTOCOL 메시지를 받아 출력합니다.

-Server-

```
#define MAXLINE 80
typedef struct thread_argvss
{
        pthread_mutex_t lock;
        int t_num;
}thread_argvs;
int connfd;
thread_argvs *thread_argv = NULL;
void *thread(void *thread_argv);

db_t *DB;
int max_connection;

void handler(int SIG)
{
        db_close(DB);
        exit(0);
}
```

Server의 전역변수 부분입니다. Lock과, 연결 수를 체크할 t_num을 묶어주었습니다. 이후 connection을 받을 connfd와 tread함수를 명시해주고, DB, max_connection또한 선언해 주었습니다. Handler는 비정상 종료 (SIGINT, SIGSTP) 상황에서 자원을 해지해주고 종료할 수 있도록 처리해 주었습니다.

```
int main ( int argc , char * argv [])
{
        signal(SIGINT, handler);
        signal(SIGTSTP, handler);
        int i;
        thread_argv = (thread_argvs*)malloc(sizeof(thread_argvs));
        pthread_mutex_init (&thread_argv->lock,(pthread_mutexattr_t *)NULL);
        thread_argv->t_num = 0;

        int listenfd, caddrlen ;
        struct hostent *h;
        struct sockaddr_in saddr, caddr ;
        int port =atoi(argv [1]);
        max_connection = atoi(argv[2]);
        int size = atoi(argv[3]);
        DB = db_open(size);


        srand(time(NULL));

        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);
```

```
        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 ,max_connection ) < 0) {
                printf ("listen() failed.\n");
                exit(3);
        }
        pthread_t tid;
         while (1) {
                caddrlen = sizeof ( caddr );
                if (( connfd = accept( listenfd , ( struct sockaddr *)& caddr , & caddrlen )) < 0) {
                        printf ("accept() failed.\n");
                        continue;
                }
                pthread_mutex_lock(&thread_argv->lock);
                pthread_create(&tid, NULL, thread, thread_argv);
                pthread_detach (tid);
        }
}
```

main문은 실습시간 진행한 SOCKET PROGRAMING과 같습니다. Signal을 이용하여 비정상 종료에 대한 handler로 넘어갈 수 있게 해주었고, argv로부터 최대 연결 개수와, 포트번호, DB size를 받아와 DB_open으로 open해 주었습니다. Connection을 기다리며 connection이 있을 때 마다 pthread_create로 lock을 걸고 thread를 생성해 통신할 수 있게 해주었습니다. 이후 detach를 통해 종료시 자원을 해지, 수거할 수 있게 해 주었습니다. Lock은 thread 자아식별 후 unlock합니다.

```
void *thread(void *thread_argvs) /* Thread routine */
{
        int n;
        int my_id = connfd;
        if (thread_argv->t_num == max_connection)
        {
                if(write(my_id , "F",1));
                pthread_mutex_unlock(&thread_argv->lock);
                pthread_exit(NULL);
        }
        else if(write(my_id, "C",1));
        thread_argv->t_num++;
        pthread_mutex_unlock(&thread_argv->lock);
        char *buf = (char*)malloc(sizeof(char)*MAXLINE);
        char *valbuf = (char*)malloc(sizeof(char)*MAXLINE);
        char *valuebuf = (char*)malloc(sizeof(char)*10);
        char *read_buf = (char*)malloc(sizeof(char));
        char *gp_detect = (char*)malloc(sizeof(char)*4);
        int rd,gp_rd;
        int key_len;
        int i;
        int val;
        rd = 0;
```

통신의 시작 부분입니다. Thread는 각자 my_id에 자신이 연결된 connfd번호를 저장합니다. 연속으로 많은 요청이 와도 thread는 각각 하나의 연결을 담당할 수 있습니다. 이후 최대 연결 개수와 현재 연결 개수를 비교하여 이미 최대 연결만큼 통신 중 일 경우, 에러를 출력하고 lock을 푼 뒤, 통신을 종료합니다. 통신 가능한 상태인 경우 현재 통신 개수를 1 증가시킨 후 lock을 풀어줍니다. 이후 이번과제에서 사용할 variable들을 선언해 줍니다.

```
while(read(my_id,read_buf,1))
{
        buf[rd++] = *read_buf;
        if(*read_buf == '\n')
        {
                if(!strncmp(buf,"CONNECT",7))
                {
                        if(write(my_id , "CONNECT_OK\n" ,11));
                        break;
                }
                else
                {
                        if(write(my_id , "UNDEFINED PROTOCOL\n" ,19));
                        rd = 0;
                }
        }
}
```

Client와 동일한 CONNECT 입력 대기 phase입니다. Newline 기준으로 striping하여, CONNECT가 들어왔을 경우 CONNECT_OK를 보내고 phase를 벗어납니다. 아닐 경우 UNDEFINED PROTOCOL을 전송하고 입력을 대기합니다.

```
while(1)
{
        rd = 0;
        gp_rd = 0;
        while(read(my_id,read_buf,1))
        {
                if(*read_buf == '\n') break;
                buf[rd++] = read_buf[0];
                if(gp_rd<3) gp_detect[gp_rd++] = read_buf[0];
        }
        buf[rd] = '\0';
        gp_detect[gp_rd] = '\0';
        if(!strcmp(buf,"DISCONNECT"))
        {
                if(write(my_id ,"BYE\n" ,4));
                break;
        }
```

이후 server는 다음 phase에서 루프를 돌게 됩니다. newline기준 입력이 DISCONNECT일 경우 bye를 전송하고 phase를 벗어납니다.

```
        free(buf);
        free(valbuf);
        free(valuebuf);
        free(read_buf);
        free(gp_detect);
        pthread_mutex_lock(&thread_argv->lock);
        thread_argv->t_num--;
        close(my_id);
        pthread_mutex_unlock(&thread_argv->lock);
        pthread_exit(NULL);
}
```

Phase를 벗어난 이후, 사용한 자원들을 해제하고, lock을 건 뒤 현재입력을 1 감소시키고 통신을 종료합니다.

```
if(!strcmp(gp_detect,"GET"))
{
        rd = 0;
        i = 0;
        while(buf[i] != '[') i++;
        i++;
        while(buf[i] != ']')
        {
                valbuf[rd++] = buf[i];
                i++;
        }
        valbuf[rd] = '\0';
        key_len = strlen(valbuf);
        val = db_get(DB, valbuf, key_len);
        if (val == 0)
        {
                if(write(my_id,"GETIN\n",6));
        }
        else
        {
                strcpy(buf,"GETOK [");
                strcat(buf,valbuf);
                strcat(buf,"] ");
                sprintf(valbuf,"[%d]\n",val);
                strcat(buf,valbuf);
                strcat(buf,"\0");
                if(write(my_id,buf,strlen(buf)));
                for(i=0;i<MAXLINE;i++) buf[i] = '\0';
        }
}
```

GET과 PUT은 별도의 gp_detect로 3글자만 비교하여 탐지할 수 있게 해 주었습니다. GET일 경우 [와]사이 key를 추출하여 db_get 을 통해 db탐색을 실행해 줍니다. 0이 return 되었을 경우 GETIN을, 아닐 경우 형식에 맞춰 GETOK [key] [value]를 출력해줍니다.

```
else if(!strcmp(gp_detect,"PUT"))
{
        rd = 0;
        i = 0;
        while(buf[i] != '[') i++;
        i++;
        while(buf[i] != ']')
        {
                valbuf[rd++] = buf[i];
                i++;
        }
        valbuf[rd] = '\0';
        key_len = strlen(valbuf);
        rd = 0;
        while(buf[i] != '[') i++;
        i++;
        while(buf[i] != ']')
        {
                valuebuf[rd++] = buf[i];
                i++;
        }
        valuebuf[rd] = '\0';
        val = atoi(valuebuf);
        db_put(DB,valbuf,key_len,val);
        if(write(my_id,"PUTOK\n",6));
        for(i=0;i<10;i++) valuebuf[i] = '\0';
        for(i=0;i<MAXLINE;i++) buf[i] = '\0';
        for(i=0;i<MAXLINE;i++) valbuf[i] = '\0';
}
```

PUT 루틴도 GET과 동일합니다. 두개의 []사이에서 각각 key와 value를 추출하여 db_put으로 입력한 뒤, PUTOK를 출력해줍니다.

```
else
{
        if(write(my_id, "UNDEFINED PROTOCOL\n" ,20));
}
```

이외의 경우 UNDEFINED PROTOCOL을 전송합니다.

-db.c 수정 및 LOCK-

```
typedef struct db {
        struct db *left;
        struct db *right;
        int num;
        char* word;
} db_t;

db_t *db_open(int size);
void db_close(db_t *db);
void db_put(db_t *db, char *key, int key_len,int value);
int db_get(db_t *db, char *key, int key_len);
void save_and_free(db_t *db,int idx);
int findw(db_t *db,char* key,int idx);
```

Db.h 파일입니다. Db_get, db_put함수의 인자에 소정의 수정을 하였습니다.

```
pthread_mutex_t db_lock = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t db_read_lock = PTHREAD_MUTEX_INITIALIZER;
int reader_num;
```

Db.c 파일에 lock과 reader수를 샐 수 있는 변수를 선언하여 reader_writer lock을 구현하였습니다.

```
void db_close(db_t *db)
{
        int i;
        pthread_mutex_lock(&db_lock);
        for(i=0;i<dbsize;i++)
        {
                if(db[i].left != NULL) save_and_free(db[i].left,i);
                if(db[i].right != NULL) save_and_free(db[i].right,i);
        }
        free(db);
        pthread_mutex_unlock(&db_lock);
}
```

Db_close의 경우 lock을 전체 테이블에 걸어 파일로 내려보내줍니다.

```
void db_put(db_t *db, char *key, int key_len,int value)
{
        int i;
        int idx =0;
        for(i=0;i<key_len;i++) idx = idx + (int)key[i];
        idx = idx%dbsize;
        if(wordcount == dbsize)
        {
                pthread_mutex_lock(&db_lock);
                for(i=0;i<dbsize;i++)
                {
                        if(db[i].left != NULL) save_and_free(db[i].left,i);
                        if(db[i].right != NULL) save_and_free(db[i].right,i);
                        db[i].left = NULL;
                        db[i].right = NULL;
                }
                new_file_name++;
                wordcount = 0;
                pthread_mutex_unlock(&db_lock);
        }
```

Db_put에서 size가 꽉 차 파일로 내려 보내야 하는 경우 전체 table에 lock을 걸어주었습니다.

```
int isthere = findw(db,key,idx);
if(isthere == 0)
{
        wordcount++;
        db_t *insert = (db_t*)malloc(sizeof(db_t));
        insert->word = (char*)malloc(sizeof(char)*(key_len+1));
        for(i=0;i<key_len;i++)
        {
                insert->word[i] = key[i];
        }
        insert->word[key_len] = '\0';
        insert->num = value;
        insert->left = NULL;
        insert->right = NULL;
        db_t *find = NULL;
        pthread_mutex_lock(&db_lock);
        if(key[0]>db[idx].word[0])
        {
                if(db[idx].right == NULL)
                {
                        db[idx].right = insert;
                        pthread_mutex_unlock(&db_lock);
                        return;
                }
                else find = db[idx].right;
        }
        else
        {
                if(db[idx].left == NULL)
                {
                        db[idx].left = insert;
                        pthread_mutex_unlock(&db_lock);
                        return;
                }
                else find = db[idx].left;
        }
```

이외의 경우, hash index기준 table에 접속하기 직전 lock을 걸고, insert이후 lock을 풀어주었습니다. Db_put의 경우 writer이기 때문에 단일 lock으로 처리해 주었습니다.

```
                while(1)
                {
                        if(insert->word[0]>find->word[0])
                        {
                                if(find->right != NULL) find = find->right;
                                else{
                                        find->right = insert;
                                        pthread_mutex_unlock(&db_lock);
                                        return;
                                }
                        }
                        else
                        {
                                if(find->left != NULL) find = find->left;
                                else{
                                        find->left = insert;
                                        pthread_mutex_unlock(&db_lock);
                                        return;
                                }
                        }
                }
        }
        else
        {
                db_t *find;
                pthread_mutex_lock(&db_lock);
                if(key[0]>db[idx].word[0]) find = db[idx].right;
                else find = db[idx].left;
                while(find != NULL)
                {
                        if(strcmp(key,find->word) == 0)
                        {
                                find->num = value;
                                pthread_mutex_unlock(&db_lock);
                                break;
                        }
                        if(key[0]>find->word[0]) find = find->right;
                        else find = find->left;
                }
        }
}
```

다른 부분에서도 동일하게 idx기준 접속 직전 lock insert이후 unlock 처리하여 주었습니다.

```
int db_get(db_t *db, char *key, int key_len)
{
        int value = 0;
        int idx = 0;
        char* rword = (char*)malloc(sizeof(char)*(key_len+1));
        char* num = (char*)malloc(sizeof(char)*8);
        char* buffer = (char*)malloc(sizeof(char));
        int i,j,tmp = 0;
        int found;
        db_t *find;
        for(i=0;i<key_len;i++) idx = idx + (int)key[i];
        idx = idx%dbsize;
        //enter Entry
        pthread_mutex_lock(&db_read_lock);
        reader_num++;
        if(reader_num == 1) pthread_mutex_lock(&db_lock);
        pthread_mutex_unlock(&db_read_lock);
        if(key[0]>db[idx].word[0]) find = db[idx].right;
        else find = db[idx].left;
        while(find != NULL)
        {
```

Db_get의 경우 reader이기에 reader_lock으로 처리해 주었습니다. 위와 동일하게 idx기준 탐색을 시작하는 시점에 걸어주었습니다.

```
        }
        pthread_mutex_lock(&db_read_lock);
        reader_num--;
        if(reader_num == 0) pthread_mutex_unlock(&db_lock);
        pthread_mutex_unlock(&db_read_lock);
        free(buffer);
        free(num);
        free(rword);
        return value;
```

이후 탐색을 끝내고 return하기 이전에 reader_lock을 해지해 주었습니다.

-Makefile-

```
CC = gcc
CFLAGS = -g -O

CSRCS = server.c db.c
TARGET = server
OBJECTS = $(CSRCS:.c=.o)

all : $(TARGET) client

$(TARGET) : $(OBJECTS)

        $(CC) -pthread $(OBJECTS) -o $@
client: client.o

        $(CC) -o client client.o
.c.o:

        $(CC) $(CFLAGS) -c $< -o $@
client.o: client.c

        $(CC) -c -o client.o client.c
clean:

        rm -f $(OBJECTS) $(TARGET) client.o client
```

Makefile또한 소정의 수정을 거쳤습니다.

감사합니다.