PA#4

2016310932 배현웅

1. 전체적인 구조

   A. 서버와 클라이언트의 통신은 과제에서 요구하는 방식으로 구성하였다. “\n” 을 기준으로 데이터를 받아 리다이렉션이 가능하게 하였고 PUT GET 명령어를 통해서 서버에서 요청한 명령을 실행한다.

   B. 클라이언트 하나당 쓰레드 하나를 연결하여 사용하였고. Entry_lock, file_lock 이라는 변수를 이용하여 접근을 제어했다.

   C. Entry_lock 을 이용해 각 엔트리, 해쉬테이블의 한 엔트리(인덱스)의 접근할 수 있는 쓰레드를 하나로 제한을 하였다. File_lock 을 이용하여 테이블에 있는 데이터를 파일로 내릴 때, 파일에 있는 데이터를 읽어올 때 간섭이 생기지 않도록 파일에 접근할 수 있는 쓰레드를 하나로 제어하였다.

2. 함수 설명

   - Client.c

```
37      //send message
38
39      int i=0;
40          while((n = read(0,buf+i,1))>0) { // something in stdin
41              if(buf[i++] != '\n') continue; // until get \n
42              else buf[i] = '\0';
43              //printf("\nbuf:%s",buf);
44              n = write(cfd,buf,i); // send server from stdin
45              memset(buf,'\0',sizeof(buf));
46              i=0;
47              while((n = read(cfd,buf+i,1))>0){ // read from server
48                  if(buf[i++] != '\n') continue; // until get \n
49                  else break;
50              }
51              //printf("buf_serv:%s",buf);
52              n = write(1,buf,i); //
53              if(!strcmp(buf,"BYE\n")){
54                  //eturn 0; // exit
55                  return 0;
56              } else if(!strcmp(buf,"Too many clients\n")){
57                  return 0;
58              }
59              fflush(stdout);
60              memset(buf,'\0',sizeof(buf));
61              i=0;
62          }
63      //}
64      close(cfd);
```

40-42 : stdin 에서 하나씩 받아와서 ‘\n’을 받을 때까지 받고 다음을 진행한다.

44-45 : stdin 에서 받은 데이터를 cfd(서버와 연결되어있는 fd)로 보내주고 buf 을 fflush 한다.

47-52 : 서버에서 보내준 데이터를 ‘\n’ 받을 때까지 받고 클라이언트에 출력해준다.(->명령에 대한 응답)

53-60 : 서버에서 받아온 데이터가 BYE 라면 DISCONNECT 를 보낸 후의 서버 응답이므로 종료하고. 56 줄과 같이 받으면 서버에서 클라이언트를 더 이상 받을 수 없다는 의미이므로 클라이언트를 종료한다. 그후 buf 변수를 초기화해주고 다시 받는다.

- Server.c

```
187     DB = db_open(Lsize);
188     if (DB == NULL) {
189         printf("DB not opened\n");
190         return -1;
191     }
192     printf("DB opened\n");
193     while(1){
194         caddrlen = sizeof(caddr);
195         connfd = (int*)malloc(sizeof(int));
196         if (( *connfd = accept( listenfd , ( struct sockaddr *)& caddr , & caddrlen )
197             printf ("accept() failed.\n");
198             continue;
199         }
200         pthread_create(&client[client_num], NULL, pthread_main, connfd);
201     }
202     db_close(DB);
203     printf("\n2 DB closed\n");
204 }
```

128-192 : DB 오픈후 성공하면 192 줄 출력

193-199 : accept 에 성공하지 못하면 다시 193 줄로 간다.

200 : 연결에 성공했다면 연결된 connfd 와 같이 pthread 를 만든다.

202-203 SIG_INT, SIGTSTP 시그널을 통해 DB 종료를 하기에 문제가 없지만 안정성을 위해 기입.

-pthread_main

```
27  void *pthread_main(void *cfd){
28      int n;
29      char buf[MAXLINE];
30      int connfd = *((int*)cfd);
31      //printf("connfd:%d\n",connfd);
32      pthread_detach(pthread_self());
33      free(cfd);
34      int i=0;
35      if(client_num >= max){ // overcrowd client num
36          if(write(connfd,"Too many clients\n",17));
37          close(connfd);
38          //pthread_exit(NULL);
39          return NULL;
40      }else {
41          client_num++;
42      }
43      int connect=0;
44      while((n=read(connfd,buf+i,1) >0))
45      {
46          if(buf[i++] != '\n') continue; // until get \n
```

32 : detach 를 통해 메인에서 쓰레드가 끝나기를 기다리지 않게 한다.

35-42 : max(최대 클라이언트 수)보다 client_num(현재 클라이언트 수)이 넘기면 “Too many clients” 를 클라이언트로 보내주어 클라이언트 종료. 아니라면 클라이언트 수 증가

43 : CONNECT 연결 되었는지 확인하는 변수

44~ 클라이언트에서 받은 데이터를 ‘\n’ 기준으로 받아 명령 처리한다.

```
47          if(connect == 0){
48              if(!strcmp(buf,"CONNECT\n")){
49                  connect=1;
50                  if(write(connfd,"CONNECT_OK\n",11));
51                  memset(buf,'\0',sizeof(buf));
52                  i=0;
53                  continue;
54              } else {
55                  if(write(connfd,"UNDEFINED PROTOCOL\n",19)); //client
56                  memset(buf,'\0',sizeof(buf));
57                  i=0;
58                  continue;
59              }
60          } else {
```

47-59 : CONNECT 라는 데이터를 받으면 CONNECT_OK 를 클라이언트에 보내주고 초기화, 그 후 명령어를 처리한다. 아니라면 UNDEFINED PROTOCOL 을 보내주고 다시 대기한다.

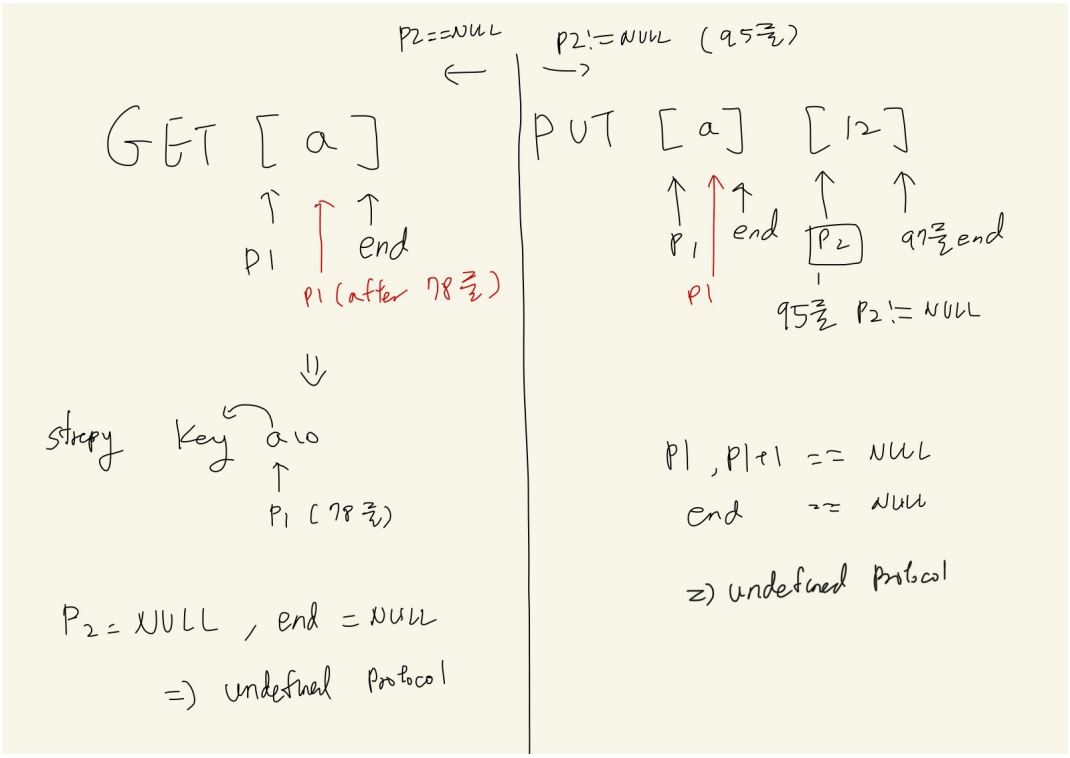
```
60          } else {
61              if(!strncmp(buf,"DISCONNECT\n",11)){ // disconnect
62                  if(write(connfd,"BYE\n",4));
63                  memset(buf,'\0',sizeof(buf));
64                  close(connfd);
65                  client_num--;
66                  return NULL;
67                  //pthread_exit(NULL);
68              } else {
```

60~ CONNECT 명령어를 받은 이후(연결이 수립) 처리하는 부분

61-66 : DISCONNECT 를 받으면 BYE 를 클라이언트에 보내준 후, client_num 감소, 쓰레드 종료

```
71                  char* val;
72                  val = (char*)calloc(sizeof(char),MAXLINE);
73                  //char val[MAXLINE];
74                  char key[MAX_KEYLEN];
75                  int key_len, val_len;
76                  // parse "buf" GET [KEY]
77                  char *p1 = strchr(buf,'[');
78                  if((p1!= NULL)&&(p1+1) != NULL) p1 +=1;
79                  else {
80                      if(write(connfd,"UNDEFINED PROTOCOL\n",19)); //cl
81                      memset(buf,'\0',sizeof(buf));
82                      i=0;
83                      continue;
84                  }
85                  char *p2 = strchr(p1,'[');
86                  char *end = strchr(buf,']');
87                  if(end == NULL){
88                      if(write(connfd,"UNDEFINED PROTOCOL\n",19)); //cl
89                      memset(buf,'\0',sizeof(buf));
90                      i=0;
91                      continue;
92                  }
93                  end[0] = '\0';
94                  strcpy(key,p1);
95                  if(p2 != NULL){
96                      p2 +=1;
97                      end = strchr(p2,']');
98                      if(end == NULL){
99                          if(write(connfd,"UNDEFINED PROTOCOL\n",19));
100                         memset(buf,'\0',sizeof(buf));
101                         i=0;
102                         continue;
103                     }
104                     end[0] = '\0';
105                     memcpy(val,p2,end-p2+1);
```

71-74 : val 변수에 value 값을 저장하고, key 변수에 key 값을 저장하기 위해 선언.



받은 데이터 파싱의 원리는 위와 같다. Strchr 함수를 통해서 데이터를 받고 [와] 를 포인터로 주소를 정하여 '[' 하나 뒤부터 ']'을 널로 바꾸어 주어 저장해준다.

GET 은 p2(두번째 '[' 찾는 포인터] 가 널이므로 104 줄의 if 문 의 과정을 넘어가고, PUT 같은 경우는 if 문을 실행한다. 올바르지 않은 데이터가 들어오면 NULL 이 만들어지므로('[' 와 ']' 을 찾을 수 없거나 '[' 이후에 문자가 없는경우) – UNDEFINED PROTOCOL 을 출력하고 다시 입력을 받는다.

```
107
108             char send_data[MAX_KEYLEN*2];
109             key_len = strlen(key);
110             //DB_GET = read
111             if(!strncmp(buf,"GET",3)){ // db_get - read
112
113                 val = db_get(DB, key, key_len, &val_len);
114                 if (val == NULL) {
115                     sprintf(send_data,"GETINV\n");
116                     if(write(connfd,send_data,strlen(send_data)));
117                 } else {
118                     sprintf(send_data,"GETOK [%s] [%d]\n", key, *((int*)val));
119                     if(write(connfd,send_data,strlen(send_data)));
120                     free(val);
121                 }
```

111- 113 앞에 GET 이라는 문자를 받으면 113 번 함수를 실행할 준비를 한다.

114-121 데이터가 없으면 GETINV 출력, 값이 존재하면 GETOK 출력

```
122                 } else if(!strncmp(buf,"PUT",3)){ // db_put - write
123                     if(p2 == NULL){
124                         if(write(connfd,"UNDEFINED PROTOCOL\n",19)); //client
125                         memset(buf,'\0',sizeof(buf));
126                         i=0;
127                         continue;
128                     }
129                     db_put(DB, key, key_len, val , sizeof(int));
130                     sprintf(send_data,"PUTOK\n");
131                         if(write(connfd,send_data,strlen(send_data)));
132                     free(val);
133                     //if(write(connfd,"PUTOK\n",6);
134                 } else {
135                     if(write(connfd,"UNDEFINED PROTOCOL\n",19)); //client
136                     memset(buf,'\0',sizeof(buf));
137                     i=0;
138                     continue;
139                 }
140                 i=0;
141                 memset(send_data,'\0',sizeof(send_data));
142                 memset(buf,'\0',sizeof(buf));
143             }
144             i=0;
145             memset(buf,'\0',sizeof(buf));
146         }
147     }
148     close(connfd);
149     client_num--;
150     return NULL;
```

122~ PUT 명령어 실행준비

123-128 : p2 가 널이라면 두번째 '[‘을 찾지못했다는 것이므로 잘못된 명령어 처리한다.

129-134 : PUT 명령어 실행하고 클라이언트로 결과 보내준다.

134-139 : GET,PUT 받지 않고 다른 명령어 받으면 다시 입력받을 준비를 한다.

141-142 : 다시쓰기 위한 널 초기화

148-150: 쓰레드가 끝날려고 하면 close 하고 클라이언트 수를 감소시킨 후(149 줄) 종료한다.

여러 쓰레드에서 명령어 처리하기 위한 쓰레드 락 설정 (IN DB.C)

```
26  #define MAXSIZE 100000
27  int* reader_num;
28  pthread_mutex_t read_lock = PTHREAD_MUTEX_INITIALIZER;
29  pthread_mutex_t file_lock= PTHREAD_MUTEX_INITIALIZER;
30
31  int table_num=0; // count for how many element in table
32  int num=-1; // for making file number
```

28-29 : 쓰레드 락을 위한 변수 설정 및 초기화

Entry_lock 변수는 db.h 파일에 설정되어있다.

DB_OPEN 함수

```
76      db_t *db = NULL;
77      db = (db_t*)malloc(sizeof(db_t)*size);
78      Lsize = size;
79      entry_lock = (pthread_mutex_t*)malloc(sizeof(pthread_mutex_t)*size);
80      reader_num =(int*)malloc(sizeof(int)*size);
81      for(int i=0; i<size; i++){ // initializer
82          (db+i)->next = NULL;
83          reader_num[i]=0;
84          pthread_mutex_init(&entry_lock[i],NULL);
85      }
86      pthread_mutex_init(&file_lock,NULL);
```

79-86 : 변수 초기화 및 데이터 초기화

DB_CLOSE 함수

```
91  void db_close(db_t *db)
92  {
93      //free(db);
94      // insert loading table
95
96      // should be check when write is end
97      for(int i=0; i<Lsize; i++){
98          pthread_mutex_lock(&entry_lock[i]);
99      }
100     if(table_num==0) { // no date to store
101         return;
102         free(db);
103     }
104     for(int i=0; i<Lsize; i++){
105         pthread_mutex_unlock(&entry_lock[i]);
106     }
107         char * buf1 = malloc(sizeof(char)*(1300));
108     pthread_mutex_lock(&file_lock);
109         num++;
110         sprintf(buf1,"./db/file-%d",num);
111         int fd = open(buf1,O_RDWR | O_CREAT | O_TRUNC,0755);
112         for(int i =0; i<Lsize; i++){
```

97-99 : DB 를 파일로 내리기 전에 HASH_TABLE 전체 엔트리에 락을 걸어 해쉬테이블에 접근하는 함수가 없을 때 까지 대기한다.

100-103 : 해쉬테이블에 아무런 값이 없다면 그냥 종료한다.

104-106 : 제출 함수에는 없앴습니다.

108 : 파일 락을 걸어 파일에 접근하는 함수가 존재한다면 기다리고 진행합니다.

109~ 그후의 함수는 동일합니다.

```
131             free(buf1);
132             close(fd);
133             free(db);
134             free(reader_num);
135         pthread_mutex_unlock(&file_lock);
136         pthread_mutex_destroy(entry_lock);
137         pthread_mutex_destroy(&read_lock);
138         pthread_mutex_destroy(&file_lock);
139     }
```

할당한 데이터 해제, 135-136 사이에 entry_lock 해제를 해놓았습니다.

DB_PUT

```
160         sprintf(buf,"%d",new->value);
161         int add=0;
162         pthread_mutex_lock(&entry_lock[entry]);
163         while(cur->next != NULL){
164             cur = cur->next;
165             if(strcmp(cur->key,key) == 0) { // if in the table
166                 cur->value = val_int;
167                 add=1; //table_num++;
168                 break;
169             }
170         }
171         if(add ==0){ // not in table
172             //memset(buf,'\0',sizeof(buf));
173             cur->next = new;
174             table_num++;
175         }
176         pthread_mutex_unlock(&entry_lock[entry]);
177         // table is full
```

\

162 – 176 : 해쉬 함수를 통해 얻은 entry 에 락을 걸어 다른 함수가 접근하지 못하게 합니다.

```
178         if(table_num >= Lsize){
179             for(int i=0; i<Lsize; i++){
180                 pthread_mutex_lock(&entry_lock[i]);
181             }
182             char * buf1 = malloc(sizeof(char)*1300);
183         pthread_mutex_lock(&file_lock);
184             num++;
185             sprintf(buf1,"./db/file-%d",num);
186             int fd = open(buf1,O_RDWR | O_CREAT | O_TRUNC,0755);
187             int trash=0;
188             for(int i =0; i<Lsize; i++){
189                 db_t* cu = (db+i); // to search table
```

178~ : table 이 꽉 찼을 때 내리는 것. 그래서 179-181 줄에서 해쉬테이블 전체에 락을 걸고, 183 줄과 같이 file 에 접근해야하므로 file_lock 설정

```
203             close(fd);
204             if(trash <=0) {// write nothing but create
205                 unlink(buf1);
206                 num--;
207             }
208             free(buf1);
209             table_num=0;
210         pthread_mutex_unlock(&file_lock);
211         for(int i=0; i<Lsize; i++){
212             pthread_mutex_unlock(&entry_lock[i]);
213         }
214         }
215
```

걸어 주었던 락 해제한다.

DB_GET

```
219  char *db_get(db_t *db, char *key, int key_len,
220               int *val_len) // finding if the key in db
221          // val_len == 4 in char size
222  {
223      char *value = NULL;
224      pthread_mutex_lock(&read_lock);
225      int entry = hash(key,key_len,Lsize);
226      reader_num[entry]++;
227      if(reader_num[entry] ==1) pthread_mutex_lock(&entry_lock[entry]);
228      pthread_mutex_unlock(&read_lock);
229      db_t *cur = db+entry;
230      // in db, db[entry] and find it
231      int flag=0;
232      if(cur->next != NULL){ // in db there is no data
```

224-228 : 넣으려는 변수의 해쉬값 = entry 이므로 그 엔트리에 접근할 수 없게 락을 걸어준다.

```
248      }
249      pthread_mutex_lock(&read_lock);
250      reader_num[entry]--;
251      if(reader_num[entry] ==0) pthread_mutex_unlock(&entry_lock[entry]);
252      pthread_mutex_unlock(&read_lock);
253
254      if(flag==1) return value; //
255      // find in the file
256      for(int k=num; k>=0; k--){
257          char *buf1 = malloc(sizeof(char)*MAXSIZE);
```


```
248      }
249      pthread_mutex_lock(&read_lock);
250      reader_num[entry]--;
251      if(reader_num[entry] ==0) pthread_mutex_unlock(&entry_lock[entry]);
252      pthread_mutex_unlock(&read_lock);
253
254      if(flag==1) return value; //
255      // find in the file
256      for(int k=num; k>=0; k--){
257          char *buf1 = malloc(sizeof(char)*MAXSIZE);
```


249-252 : 접근한 이후 락 해제

254 : 테이블에서 찾았다면 (flag==1) value 출력해주고 함수 종료

256~ : 파일에서 데이터 찾는다.

```
262         if(file_size <=0) {
263             free(buf1);
264             continue;
265         }
266         char* s1 = (char*)malloc(sizeof(char)*(file_size));
267     pthread_mutex_lock(&file_lock); // to avoid num change
268         int fd = open(buf1, O_RDONLY);
269         int trash = read(fd,s1,sizeof(char)*file_size);
270     pthread_mutex_unlock(&file_lock);
271         trash++;
272         int sp =0;
273       while(sp<file_size) { // check line (number:Lsize)
```

267-270 : 파일에 접근하여 열 때 file_lock 을 통해 파일에 접근하는 함수가 없도록 한다. 파일의 정보를 읽은 후에는
락 해제