Data/file Term project Final Report

김무준

마재윤

조용래

1. 개요

TCP/IP 기반의 클라이언트 서버 모델 형식으로 클라이언트, 서버 쪽 프로그램, 두 가지 프로그램을 작성한다. 잘 알려진 대로, 클라이언트는 서버쪽에 요청을 보내고 서버는 요청의 종류에 따라 각기 다른 역할을 수행한다. 이번 텀 프로젝트에서는 소켓, 멀티 쓰레드, 파일 입출력, TCP/IP 을 기반 기술로 써야 한다. 쓰레드, 파일 입출력, 서버-클라이언트 모델 3 가지 기술을 잘 활용할 수 있을 만한 주제가 데이터 / 파일 서버라고 생각 했다. 클라이언트의 요청을 서버가 스레드를 만들어 처리하고 주고 받는 것을 파일 형식이므로 파일 입출력이 자연스럽게 들어가기 때문이다.

이번 텀 프로젝트의 핵심은 위의 언급된 기술들을 활용하는 것을 넘어 데이터-파일 서버의 핵심인 클라이언트 다양한 요청을 서버가 적절히 수행하는 것이다. 흔히 파일 서버에서 파일을 대상으로 서버쪽에 무언가를 요청을 한다고 하면, 1) 파일 보내기 2) 파일 받기 3) 파일 삭제 4) 파일 이름 수정 등등 파일에 대한 연산들이 중심이다. 이런 파일 연산들은 자료 전송의 흐름에 따라 세지로 다시 나뉜다. 1)클라이언언트가 자신의 로컬에 있는 파일을 서버쪽에 전송하는 것 2) 클라이언트가 요구한 파일을 서버에서 클라이언트 쪽으로 3) 서버가 관리하고 있는 파일 정보에 대한 수정. 1)의 예시는 ftp 에서와 같이 put 으로 구현된다. 2)는 get 3)은 rename 이나 del 로 구현된다.

파일에 대한 연산 말고도 클라이언트의 ID 와 비밀번호, 사용자 추가하기, 파일 목록, 사용자-파일 권한, 도움말 같은 기능들이 추가적으로 필요한 기능들이다. 여기서 한 가지 빠진 것은 디렉터리 구현인데, 클라이언트가 서버쪽의 디렉터리를 변경하면서 원하는 위치에 파일을 보내거나 할 수 있게 해준다. 아쉽게도 이번 이번 텀 프로젝트에서는 빠지게 되었다.

2. 자료구조

데이터/파일 서버라고 해서 이 둘 사이의 명령어 전달 처리하는 것만 생각하면 안 된다. 이들을 어떻게 보고 어떤 방식으로 다뤄야하는지도 알아야 한다. 서버쪽에서 클라이언트에 대한 정보를 client 클래스 안에 id, password, 권한, 파일에 대한 정보는 해당 파일의 이름과, 파일 경로, 파일 권한으로 나타내었다. 작업 후반에 권한을 추가했기 때문에 기존의 것들과 합쳐야 되는 상황이 왔을 때는 클래스를 설계 하여 모듈 형식으로 갖다 붙였기 때문에 어렵지 않게 추가시킬 수 있었다.

3. 서버

1) 초기화 과정

먼저 서버로써 역할을 하기 위해서 서버용 소켓, 클라이언트용 소켓을 만들고 적절히 초기화 시켜주는 작업이 필요하다. 서버쪽에선 early_stuff() 함수안에 소켓 관련 초기화 코드들을 집어 넣었다. 두번째로 파일 목록, 권한 목록, 클라이언트 정보를 포함한 파일을 메모리로 한다. 해당 파일을 읽은 후 서버 자료구조에 저장한다.

2) 요청 수락과 스레드 처리.

기본적인 초기화 과정 끝났으면 클라이언트의 요청을 기다리는 무한루프 쪽으로 넘어간다. 여기에서 accept 함수가 리턴하는 서술자를 이용하여 클라이언트 쪽과 통신을 할 수 있게 된다. Accept 호출 후 pthread_create()함수를 호출하여 스레드가 생성되고 클라이언트의 모든 요청을 처리하게 된다. 이 프로그램에서 사실상 가장 중요한 역할을 하고 있다. 스레드를 담당하는 함수는 thread_request_process() 이고 인수로 클라이언트 소켓 서술자를 넘겨 주었다.

3) void *thread_request_process(void *arg) 함수 분석

1. 첫번째로 할 일은 클라이언트의 로그인 처리이다. 해당 함수는 void *thread_request_process(void *arg) 로 구현 되었다. 초기화 단계에서 클라이언트의 ID 와 패스워드가 있는 파일을 읽어 cli_list 에 저장해 두는데 여기서 입력된 ID 와 비밀번호를 점검하여 맞으면 넘어가고 틀리면 메시지를 띄워 거부한다.

2 클라이언트가 입력한 명령어는 read 로 읽어와 buf 문자형 배열에 저장한다. 이후 buf 의 n 개의 앞자리를 구분점으로 삼는다.

3 Get : get A 형식으로 입력이 들어 왔으면 먼저 서버는 현재 파일 목록에서 A 라는 파일 이름을 가지고 있는지 확인한다. isInList()함수가 이 역할을 한다. 없으면 no 라고 준다. 만약 있으면 권한 문제로 넘어간다. 클라이언트의 권한과 파일의 권한을 비교하여 적절하면 해당 파일을 넘겨주는 SendingFile()함수로 넘어가고 틀리면 오류 메시지를 전송한다.

```
void SendingFile(string &name, int &client_sockfd) { // t
    string targetfile = papers_path;
    targetfile += name;
    ifstream in_text(targetfile.c_str());
    string text_buf;
    while(getline(in_text, text_buf)) {
        text_buf += "\n";
        write(client_sockfd, text_buf.c_str(), BUFSIZE);
    }
    write(client_sockfd, "$", BUFSIZE);
}
```

< 실제 파일을 전송하는 SendingFile >

```
if (strncmp(buf, "get", 3) == 0) {
    string file_name = getTitle(3, buf); // second argument
    bool exist = isInList(file_name);
    cout << "request text : " << file_name << endl;
    if(exist == false) {
        write(client_sockfd, "no", BUFSIZE);
    } else {

        if(am.doesHeHaveAPermissionOnThatFile(client_name, file_name)) {
            SendingFile(file_name, client_sockfd);
        } else {
            cout << client_name << "don't have permission for " << file_name << e
            write(client_sockfd, "noaccess", BUFSIZE);
        }
    }
```

< 명령어 처리를 담당하는 부분>
get 의 변형으로 mget 도 있는데 get A B C 와 같이 여러 개의 파일을 보내준다.

4 Add : add 는 클라이언트가 서버로 보내는 파일이다. 먼저 해당 파일 이름이 이미 리스트에 존재하는지 확인 후 없으면 파일을 받아오는 RecevingFile()함수를 호출한다. 새로운 파일에 대해 권한은 기본 권한(GENERAL)을 부여한다.

5 Del : del A 가 들어오면 A 를 리스트에서 찾고 이 파일과 관련되어 있는 자료 구조에서도 삭제한다. 즉 리스트 목록, 권한 리스트, 실제 파일의 삭제 총 3 가지 연산이 수행된다. 메모리 상의 리스트에서 삭제 한 후, 실제 파일을 업데이트 해준다. 권한 리스트도 마찬가지다. 이것을 보면 메모리 상의 변화를 즉각적으로 반영하는 write-through 정책을 사용하였다. 실제 파일의 삭제는 시스템 콜을 사용해야 한다. System(“rm –f A”)를 호출한다.

6 help : cmd_list 파일의 명령어 형식들을 읽어 보여준다.

7 Rename : rename A B 형식으로 A 파일 이름를 B 로 바꾼다. 역시 파일 점검이 먼저 A 가 존재하는지 확인 없다면 B 가 이미 목록에 있는 파일명인지 확인하고 없다면 rename_file(A, B)

를 호출한다. 다음은 rename_file 함수의 구현.

```
void rename_file(string old_file, string new_file) {
    list<string>::iterator l_iter = find(papers.begin(), paper
    *l_iter = new_file;
    ofstream out_paper_list("papers");
    // update papers list file to reflect new file name
    l_iter = papers.begin();
    for(; l_iter != papers.end(); ++l_iter) {
        out_paper_list << *l_iter << endl;
    }


    // update real file name by using syscall mv
    string first_arg = "./" + papers_path + "/" + old_file;
    string second_arg = "./" + papers_path + "/" + new_file;
    string arg = "mv " + first_arg + " " + second_arg;
    system(arg.c_str());

}
```

A 를 파일 목록에서 제거 하고, system(mv A B)형식으로 mv 명령어의 이름 바꾸는 기능을 시스템 콜에 사용하였다.

8 list : 현재 파일 목록을 보여준다.

9 View : 클라이언트 쪽에서 서버가 보내준 자료를 저장하지는 않고 터미널 창에 띄우는 역할만 수행.

10 Quit : 클라이언트가 서버와의 연결을 끝내려 한다는 의미이므로 close 로 연결을 끊는다.

11 다음의 일련의 명령어들은 슈퍼 유저라는 개념 하에서 설계 된 것이다. 슈퍼 유저란 클라이언트가 감히 접근하지 못하는 명령어를 마음대로 쓸 수 있는 유저를 지칭한다. 여러명의 사용자가 한 시스템을 이용하는데 있어 시스템 관리자가 필요한 법이다. grant : grant A B 형식으로 사용자 A 의 권한을 B 로 변경한다. 권한의 종류는 3 가지 NONE, GENERAL, SPECIAL 로 지정할 수 있다.

12 Fgrant : fgrant A B. A 파일의 권한을 B 로 수정한다.

13 uadd : uadd clientID password permission 형식으로 입력하면 새로운 클라이언트 계정을 permssion 으로 등록한다.

```
else if( strncmp(buf, "uadd", 4) == 0) {
   if(super_flag == false) {
      write(client_sockfd, "notsuper", BUFSIZE);
      continue;
   }
   string tmp(&buf[0], &buf[strlen(buf)]);
   vector<string> tokens = StringTokenizer::getTokens(tmp);
   if(tokens.size() != 4) {
      write(client_sockfd, "wrongformat", BUFSIZE);
      continue;
   }
   if(  (strcmp(tokens[3].c_str(), "NONE") != 0) &&
        ( strcmp(tokens[3].c_str(),"GENERAL") != 0) &&
        ( strcmp(tokens[3].c_str(), "SPECIAL") != 0)) {

       write(client_sockfd, "fail_permission", BUFSIZE);
       continue;
    }
   int per_int = am.IntValue_for_permission(tokens[3]);
   am.add_new_client_and_permission(tokens[1], per_int);
   am.write_client_access(access_path);

   client_ele cli_tmp(tokens[1], tokens[2]);
   cli_object.add_client(cli_tmp);
   cli_object.write_client_list(client_path);

   write(client_sockfd, "success", BUFSIZE);
   string result_str = tokens[1];
   result_str += " has been registered successfully !";
   write(client_sockfd, result_str.c_str(), BUFSIZE);
```

14 madd : madd 는 클라이언트가 서버로 여러 개의 파일을 보낼 때 사용한다. 먼저 해당 파일 이름이 이미 리스트에 존재하는지 확인 후 없으면 파일을 받아오는 RecevingFile()함수를 호출한다. 새로운 파일에 대해 권한은 기본 권한(GENERAL)을 부여한다.

```
else if(strncmp(buf, "madd", 4) == 0){

   string tmp(&buf[0], &buf[strlen(buf)]);
   vector<string> tokens = StringTokenizer::getTokens(tmp);

   bool exist;
   for(int i=1; i<tokens.size(); ++i){
       exist = isInList(tokens[i]);

       if(exist == true) {
           write(client_sockfd,"already exist", BUFSIZE);
       } else {
           char check[BUFSIZE];
           read(client_sockfd, check, BUFSIZE);
           if(strcmp(check, "no") == 0)
               break;
           ReceevingFile(tokens[i], client_sockfd);
           am.add_new_paper_and_permission(tokens[i], 1);
           am.write_file_access(access_path);
       }
   }
```

15. getsize : 클라이언트에게 현재 서버에 저장되어 있는 파일의 개수를 보여준다. paper.size() 를 사용하여 구현하였다.

```
} else if(strncmp(buf, "getsize", 7) == 0){
    write(client_sockfd,papers.size(), BUFSIZE);
```

16. calist : 클라이언트가 super 등급일 경우에만 수행할 수 있는 명령어이고, 현재 등록되어있는 사용자들의 id 와 등급을 보여준다. 반복문을 사용하여 client list 의 size 만큼 돌며 i 번째의 id + " " + permission + " " 식으로 모든 사용자 정보를 저장 후 클라이언트에게 넘겨준다.

```
} else if(strncmp(buf, "calist", 6) == 0){
    if(super_flag == false) {
        write(client_sockfd, "notsuper", BUFSIZE);
        continue;
    }
    string temp_client_info = am.get_client_list(0).get_id();
    temp_client_info += " ";
    temp_client_info += am.aux_print(am.get_client_permission(am.get_client_list(0).get_id()));
    temp_client_info += " ";
    for(int i=1; i<am.get_client_list_size(); i++){
        temp_client_info += am.get_client_list(i).get_id();
        temp_client_info += " ";
        temp_client_info += am.aux_print(am.get_client_permission(am.get_client_list(i).get_id()));
        temp_client_info += " ";
    }
    write(client_sockfd, temp_client_info, BUFSIZE);
```

17. mdel

mdel A B C .. 와 같이 입력을 받아 Multi Delete 한다. del 함수를 응용하여 기존의 del 명령어의 경우 한번의 호출에 하나의 파일만 삭제하는 명령어 였다면, mdel 은 한번의 호출에 여러 개의 파일을 삭제하는 명령어이다. 수행되는 연산은 del 과 같으며 메모리 상의 리스트, 실제 파일, 권한 리스트 모두 삭제 및 변경을 한다.

18. clist

최고권한(Super) 사용자만 사용할 수 있는 명령어이다. Client 측에서 Super 권한을 가진 사용자가 로그인을 하고 Clist 를 입력하게 되면 현재 서버측에 저장되어 있는 사용자들의 ID 와 PASSWD 를 화면에 출력한다.

4. 클라이언트

클라이언트 프로그램은 사용자로부터 명령어을 처리합니다. 파일을 보내기도 하고 받아오기도 합니다. 서버에 비해 단순한 구조를 가지며 주로 하는 일은 파일 입출력, 서버쪽으로 보낸 명령어의 결과를 사용자에게 띄워줍니다. 명령어는 서버 프로그램에 나온 것과 똑같습니다.

5. 일정 & 분담

	4-3	4-4	4-5	5-1	5-2	5-3	5-4	6-1	6-2

김무준	클라이언트 관계 정의		ftp 기본 명령어		슈퍼유저 명령어		테스트
마재윤	서버 관계 정의		ftp 기본 명령어		슈퍼유저 명령어		
조용래	서버 - 클라이언트 관계, 연산 정의		ftp 기본 명령어		권한, 수퍼유저 명령어		