**네트워크 프로그래밍 보고서**

20122382

박 준 은

**■ 함수 목록 및 설명**

**[server.c]**

while(1){

clnt\_addr\_size = sizeof(clnt\_addr);

clnt\_sock = accept(serv\_sock, (struct sockaddr\*)&clnt\_addr, &clnt\_addr\_size);

if(clnt\_sock==-1)

continue;

printf("접속완료!!\n");

pid=fork();

if(pid==0){

close(serv\_sock);

childProcess();

}

else{

close(clnt\_sock);

}

}

return 0;

}

처음에 기본적인 통신준비를 한다. 만약 자식이 들어온다면 아래의 코드를 통해 부모와 자식을 분리한다. 성공적으로 부모와 자식이 분리된다면 자식프로세스는 리스닝 소켓을 닫아준다. 부모소켓은 커넥티드 소켓을 닫아준다. 자식프로세스를 계속 진행하기 위해 childProcess()를 실행시킨다.

//로그인 하는부분

while(check){

read(clnt\_sock,buf,MAX);

int i;

for(i=0;i<3;i++)

if(!strcmp(buf,name[i])){

printf("%s님 로그인 완료\n",buf);

strcpy(id,buf);

strcat(path,buf);

strcpy(buf,"111");

write(clnt\_sock,buf,MAX);

check=0;

continue;

}

if(check==1){

strcpy(buf,"000");

write(clnt\_sock,buf,MAX);

}

}

위의 코드는 자식프로세스로부터 아이디를 받아 로그인을 하는 부분이다. 서버에는 기본적으로 juneun, zzang, goodman 이 세가지의 아이디가 등록되어있다. 클라이언트가 이 아이디들을 입력하면 로그인이 성공적으로 진행되고, 루트디렉토리를 [./아이디] 로 잡아준다. 그리고 이 문자열을 path 배열에 입력해준다. 만약 서버에 아이디들의 루트디렉토리가 없다면 새로 생성한다. writePipe=popen(mkdir 아이디,"r"); 를 사용한다.

다음으로 클라이언트로부터 명령어를 받아온다. 클라이언트로 부터 온 배열을 strtok()를 이용해 자른다. 가장 처음에 들어온 문자열을 strcmp()로 구분해 if-else 처리를 한다. 명세서에 명시되어있는 ls, cat, save, cp, cd, mkdir, rmdir, rm 을 모두 구현했다.

else

if(!strcmp(bufTmp,"cat")|| !strcmp(bufTmp,"mkdir")|| !strcmp(bufTmp,"rmdir")||!strcmp(bufTmp,"rm")){

strcat(bufTmp," "); //클라이언트로부터 받고 명령어가 저장된

strcat(bufTmp,path); //문자열 = butTmp

strcat(bufTmp,"/");

token=strtok(buf," "); //클라이언트로부터 받은 전체

token=strtok(NULL,"\n"); //문자열 = buf

strcat(bufTmp,token);

}

rm, mkdir, rmdir, cat 의 명령어들은 모두 [명령어 파일] 형식으로 된 명령어이다. 이 명령어들은 현재 내가 있는 경로에서 실행시켜야 하므로, 파일앞에 [./경로/] 를 붙여주어야 한다. 위의 코드들이 그 과정이다. bufTmp에는 명령어가 저장되어있다. Strcat을 이용해 현재 경로를 붙여넣어준다. 그 후 strtok()을 이용해 파일 이름을 가져와 bufTmp에 붙여준다. 최종적으로 [명령어 ./경로/파일] 형식이 된다.

if(!strcmp(bufTmp,"ls") || !strcmp(bufTmp,"dir")){

strcat(bufTmp," ");

strcat(bufTmp,path);

}

Ls와 dir 명령어의 경우에는 위와같이 [명령어 ./경로] 의 형식으로 변환해준다.

else if(!strcmp(bufTmp,"cd")){

token=strtok(buf," ");

token=strtok(NULL,"\n");

if(!strcmp(token,"..")){

int len=0;

while(path[len] != '\0') len++;

while(path[len] != '/'){

path[len]='\0';

len --;

}

if(len!=1){

path[len] = '\0';

}

if(path[2]=='\0'){

strcat(path,id);

write(clnt\_sock,"루트 디렉토리 입니다\n",MAX);

continue;

}

}

else{

if(path[2] != '\0')

strcat(path,"/");

strcat(path,token);

}

write(clnt\_sock,"이동완료\n",MAX);

continue;

}

Cd 명령어는 path를 바꾸어 주었다. [cd 디렉토리] 형식으로 클라이언트가 전송해준다면, “..” 일 경우와 아닐때로 구분한다. “..”이 아닐경우에는 기존 path에 ‘/’를 붙이고 디렉토리를 붙여준다. “..”일 경우에는 상위 디렉토리로 가는 명령어이기 때문에 path에서 현재 디렉토리를 지워준다. 현재가 만약 route 디렉토리라면 이동하지 않고 루트 디렉토리라고 클라이언트에게 알려준다.

else if(!strcmp(bufTmp,"cp")){

strcat(bufTmp," ");

strcat(bufTmp,path);

strcat(bufTmp,"/");

token=strtok(buf," ");

token=strtok(NULL," ");

strcat(bufTmp,token);

strcat(bufTmp," ");

strcat(bufTmp,path);

strcat(bufTmp,"/");

token=strtok(NULL,"\n");

strcat(bufTmp,token);

}

Cp 명령어는 뒤에 [cp 파일1 파일2] 형식이다. 파일1 파일2에 모두 path를 붙혀준다.

else if(!strcmp(bufTmp,"save")){

int fd;

char pathTmp[30] = {0};

char buf2[MAX] = {0};

strcpy(pathTmp,path);

token = strtok(NULL,"\n");

strcat(pathTmp,"/");

strcat(pathTmp,token);

fd = open(pathTmp,O\_WRONLY|O\_CREAT);

read(clnt\_sock,buf2,MAX);

write(fd,buf2,sizeof(buf2));

strcpy(buf2,"저장 되었습니다.\n\0");

write(clnt\_sock,buf2,MAX);

continue; }

Save 명령어는 [svae 파일] 형식이다. Path에 “/파일이름”을 붙혀서 경로를 만들고, 그 경로를 이용해 쓰기전용 파일을 하나 생성한다. 클라이언트에게 한번 더 read()를 해 그 파일에 적을 내용을 받는다. 그 내용을 파일에 적어준다.

만약 exit를 클라이언트가 보내온다면 프로세스를 종료시켜준다.

char result[MAX]={0};

writePipe=popen(bufTmp,"r");

fread( (void\*)result, sizeof(char), MAX, writePipe );

write(clnt\_sock,result,MAX);

pclose(writePipe);

명령어를 모두 구분해서 클라이언트가 현재 위치에서 명령어를 실행하도록 하는 작업이 끝났다. 위 작업들은 모두 bufTmp에 저장된다. Popen()을 이용해 클라이언트가 요구한 명령어를 실행한다. 그 후 fread()를 사용해 result 배열에 결과값을 저장한다. Result를 클라이언트에게 전송해준다.

**[client.c]**

while(1){

printf("id를 입력하세요 : ");

fgets(buf,MAX-1,stdin);

chop(buf);

write(sock,buf,MAX);

read(sock,buf,MAX);

if(buf[0]=='1'){

printf("로그인 완료\n");

break;

}

else if(buf[0] == '0')

printf("없는 id입니다!\n");

}

처음에 로그인 했을 때 아이디를 입력하는 부분이다. 서버가 가지고있는 아이디를 입력할 때까지 계속 입력하게 만든다. 성공적으로 입력하면 로그인 완료라는 메세지를 띄워준다.

while(1){

char path[30]= {0};

memset(buf,0,MAX);

read(sock,path,PATH);

printf("%s",path);

printf("$ ");

fgets(buf,MAX-1,stdin);

chop(buf);

if(!strcmp(buf,"exit")){

write(sock,buf,MAX);

printf("로그아웃\n");

break;

}

else if(buf[0]=='s' && buf[1]=='a' && buf[2]=='v' && buf[3]=='e'){

char buf2[MAX] = {0};

printf("저장할 내용을 입력하세요 : ");

fgets(buf2,MAX-1,stdin);

chop(buf2);

write(sock,buf,MAX);

write(sock,buf2,MAX);

read(sock,buf,MAX);

printf("%s\n",buf);

}

else{

write(sock,buf,MAX);

printf("send : %s\n",buf);

read(sock,buf,MAX);

printf("%s\n",buf);

}

}

서버와 클라이언트가 통신하는 부분이다. 클라이언트는 먼저 서버로부터 현재 path를 read()한다. 클라이언트의 터미널에 [경로]$ 로 보이도록 한다. 클라이언트가 서버로 전송할 명령어를 입력한다. 입력한 부분이 exit일 경우와 save 명령어를 사용할 경우 그렇지 않을 경우로 나누어 진행한다.

Exit일 경우 서버에게 exit을 전송하고 프로세서를 종료한다. Exit이 아니고 save가 아닐경우 입력받은 문자열을 그대로 서버에게 전송한다. 그리고 서버가 전송한 결과를 read()한다.

Svae일 경우 [save 파일명]을 서버에게 전송한다. 그 후 한번 더 클라이언트가 저장할 내용을 입력받아 서버에게 전송한다. 그리고 그 결과값을 read()해 클라이언트에게 보여준다.

**■ 기능구현 여부 구현여부 구현여부**

|  |  |  |  |
| --- | --- | --- | --- |
| **ls(dir)** | **o** | **로그인** | **o** |
| **cat** | **o** | **멀티프로세싱** | **o** |
| **mkdir** | **o** |  |  |
| **rmdir** | **o** |  |  |
| **cp** | **o** |  |  |
| **cd** | **o** |  |  |
| **save** | **o** |  |  |
| **rm** | **o** |  |  |

**■ 프로그램 구조도**

**Client**  **Server**

Loop (서버가 “111” 보낼때까지)

Id 전송

결과 전송

Loop (클라이언트가 exit 할때까지)

현재 경로(path) 전송

If(exit)

exit 전송 시키고 break

If(save)

명령어 전송

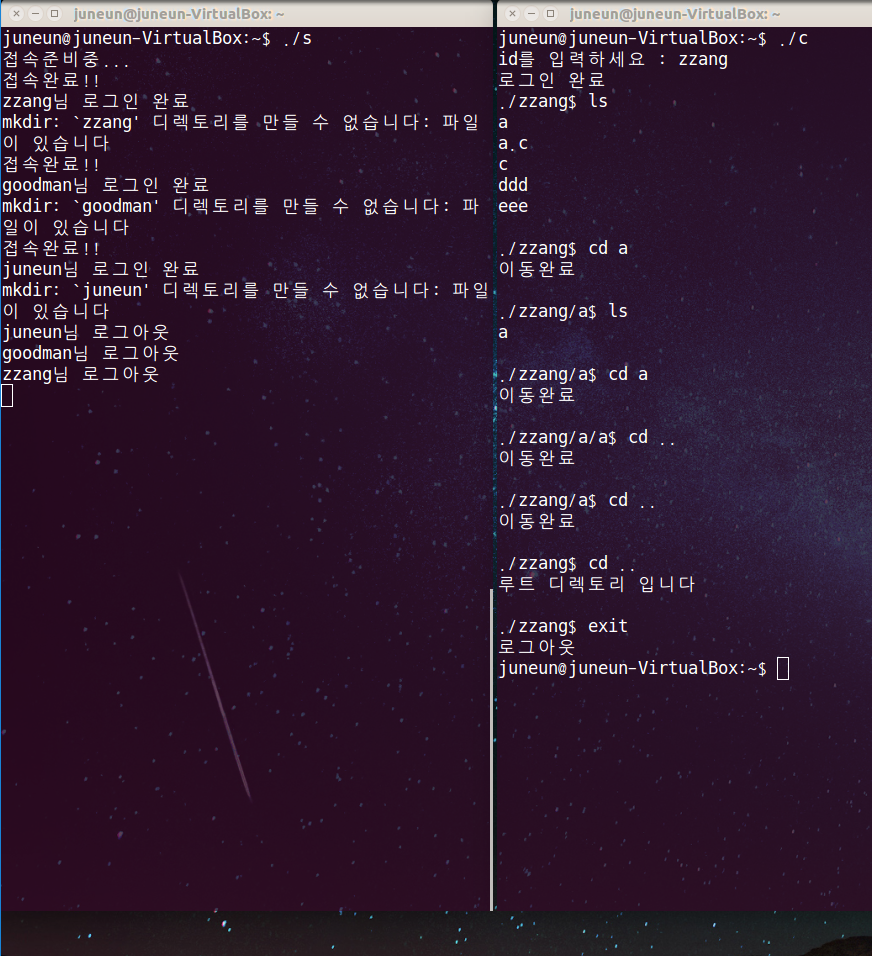
파일내용 전송

결과 전송

else

파일내용 전송

결과 전송

**■ 샘플** **스크린샷.**

