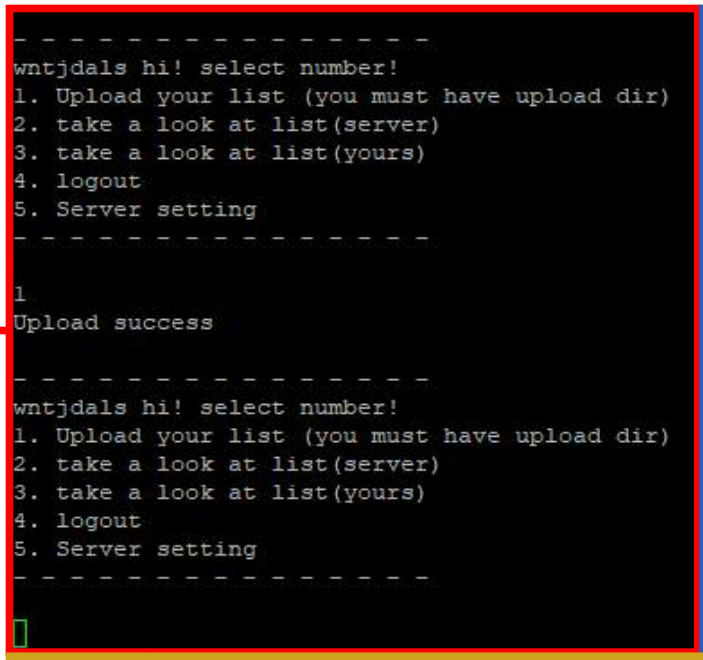
임베디드 운영체제 REPORT #3

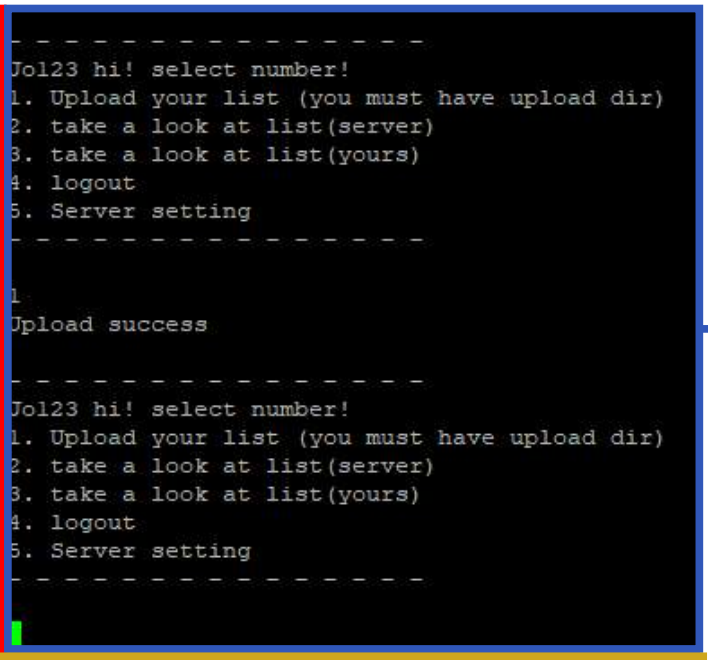
2014146042 주성민

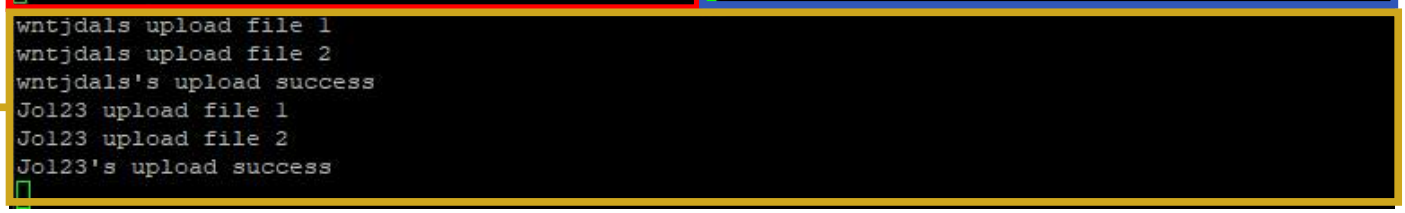
부제 : 텀프로젝트 #3

교수 : 김 수민 교수님

2단계 프로젝트에서 IP와 PORT 번호는 전송하지 않았기 때문에 시나리오를 처음부터 구성하였습니다. 회원가입 알고리즘 설명은 생략하였습니다. wntjdals 계정이 클라이언트1, Jo123 계정이 클라이언트2에서 접속 후 리스트를 업로드하는 과정입니다.







클라이언트1에서 wntjdals 계정으로 접속 후 해당 클라이언트의 upload 다이렉트에 존재하는 txt 파일들의 목록을 전송하게 됩니다.

클라이언트2에서 Joo123 계정으로 접속 후 해당 클라이언트의 upload 다이렉트에 존재하는 txt 파일들의 목록을 전송하게 됩니다.

서버의 화면입니다. 실제 서버에 로그에 남는 것처럼 클라이언트들의 행동을 기록합니다. 클라이언트의 업로드 요청이 오면 서버는 Total_list.txt 파일에 쓰기동작을 수행하며 새로운 파일 목록을 만들게 됩니다. 이 구간에서 동시에 두 클라이언트가 접속하게 되면 쓰기동작을 할 때 충돌이 생길 수 있기 때문에 세마포를 활용한 임계영역을 구현하여 보호하였습니다. 자세한 설명은 코드부분에 있습니다. 위 사진은 클라이언트1,2가 거의 동시에 업로드 요청을 한 상황이며 조금이라도 빨리 요청한 클라이언트1의 업로드가 먼저 수행되고 그 후에 클라이언트2의 업로드가 수행됩니다.

비교를 확실히 하기 위해 file 목록 한 개를 등록할 때 마다 sleep함수를 사용하여 딜레이를 주어 임계영역을 사용하지 않을 때와 차이를 눈으로 볼 수 있도록 구현하였습니다.

[업로드 과정과 관련된 서버 코드]

```
if(strcmp(msgg,"1") == 0)
{
    while(*Semaphore <= 0);
    *Semaphore = 0;
    sprintf(msgg,"./userslist/%s_list.txt",id);
    upload = fopen(msgg,"w");
    memset(buf,0,512);
    memset(msg,0,512);
    i = 1;
    while(1)
    {
        rcv_byte = recv(new_fd,buf,sizeof(buf),0);
        if(strcmp(buf,"END")==0)
        {
            break;
        }
        sprintf(msg,"%s",buf);
        fprintf(upload,"%s\n",msg);
        printf("%s upload file %d\n",id,i);
        sleep(1);
        i++;
        memset(msg,0,512);
    }
    fclose(upload);
    dir_info = opendir("./userslist");
    upload = fopen("Total_list.txt","w");
    if(NULL != dir_info)
    {
        while(dir_entry = readdir(dir_info))
        {
            if(strstr(dir_entry->d_name,".txt") != NULL)
            {
                sprintf(msgg,"./userslist/%s",dir_entry->d_name);
                f_dir  = fopen(msgg,"r");
                memset(msgg,0,40);
                memset(msg,0,512);
                memset(buf,0,512);
                while(1)
                {
                    if(fgets(msg,sizeof(msg),f_dir)==NULL)
                    {
                        memset(buf,0,512);
                        memset(msg,0,512);
                        break;
                    }
                    else
                    {
                        sprintf(buf,"%d. %s",i,msg);
                        fprintf(upload,"%s",buf);
                        memset(buf,0,512);
                        memset(msg,0,512);
                    }
                }
            }
        }
    }
    closedir(dir_info);
    fclose(upload);
    fclose(f_dir);
    sprintf(msgg,"%s's upload success",id);
    printf("%s\n",msgg);
    memset(msgg,0,40);
    *Semaphore = 1;
}
```

```
static int *Semaphore;
Semaphore = mmap(NULL, sizeof *Semaphore, PROT_READ | PROT_WRITE,
    MAP_SHARED | MAP_ANONYMOUS, -1, 0);

*Semaphore = 1;
```

프로젝트2와 달라진 점은 공유변수 Semaphore를 사용한 것입니다. 프로세스끼리 같은 파일에 동시에 쓰기 명령을 수행하는 것을 방지하기 위해 2진 세마포를 구현하였습니다. 빨간색 네모박스가 임계영역을 보호하는 역할을 합니다. 서버 실행 시 Semaphore 변수를 '1'로 초기화 해줍니다. 먼저 들어온 프로세스는 Semaphore가 '1'이기 때문에 임계영역으로 들어가고, 후에 들어온 프로세스는 먼저 들어간 프로세스가 임계영역을 탈출하기 전까지 대기하게 됩니다.

* 프로젝트2 보고서에 회원가입, 로그인, 업로드에 대해 상세히 설명되어 있습니다.

[업로드 과정과 관련된 클라이언트 코드]

```
if(strcmp(msg,"1") == 0)
{
    dir_info = opendir("./upload");
    if(NULL != dir_info)//갈 곳 된  dir이  아 니 라 면 !
    {
        while(dir_entry = readdir(dir_info))//파 일 을  하 나 씩  읽 어 줌
        {
            if(strstr(dir_entry->d_name,".txt") != NULL)
            {
                sprintf(buf,"%s %s %s %d",dir_entry->d_name,id,USER_IP,USER_PORT);
                send(sockfd,buf,sizeof(buf),0);
                memset(buf,0,512);
            }
        }
        closedir(dir_info);
        send(sockfd,"END",sizeof(msg),0);
        printf("Upload success\n\n");
    }
}
```

빨간색 네모박스가 프로젝트2와 다른 부분입니다. 클라이언트의 upload dir에 있는 파일들을 읽어서 서버에 하나씩 전송하게 됩니다. 서버의 리스트 목록에 PORT와 IP 번호 정보까지 포함하기 위해 클라이언트의 upload 리스트를 전송할 때 같이 전송해 주게 됩니다. 최종 서버에서 클라이언트들에게 보여주는 리스트는 아래와 같습니다.

```
========list========
1. Jo_2.txt Jol23 220.149.128.102 4372
2. Jo_1.txt Jol23 220.149.128.102 4372
3. test3.txt wntjdals 220.149.128.101 4371
4. test2.txt wntjdals 220.149.128.101 4371
====================
```

[다운로드 시나리오]

1. 클라이언트1의 유저가 다운로드 받을 파일을 선택하는 상황입니다. 먼저 클라이언트1의 download 폴더입니다.

```
st20141460420602-c:~/server$ ls
client_1_1_test.c  download  final  upload
st20141460420602-c:~/server$ cd download/
st20141460420602-c:~/server/download$ ls
st20141460420602-c:~/server/download$ cd ..
st20141460420602-c:~/server$ ./final
```

다음과 같이 download폴더를 비워두고 시나리오를 진행하였습니다.

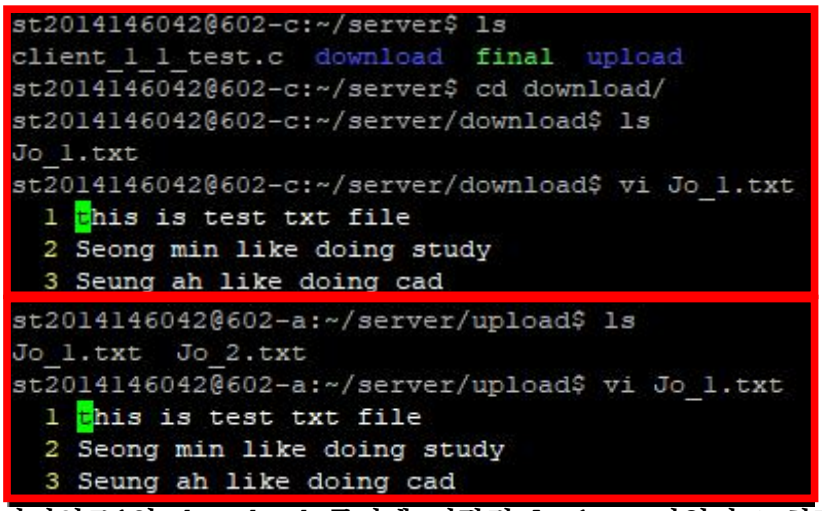
```
========list========
1. Jo_2.txt Jol23 220.149.128.102 4372
2. Jo_1.txt Jol23 220.149.128.102 4372
3. test3.txt wntjdals 220.149.128.101 4371
4. test2.txt wntjdals 220.149.128.101 4371
====================
====================
select number
1. down load file
2. quit
====================
1
====================
select file
====================
2
====================
file : Jo_1.txt
user : Jol23
IP : 220.149.128.102
PORT : 4372

====================
Client-socket() new_fd is OK..
Client-connect() is OK..

Download Start!
you download 1 line
you download 2 line
you download 3 line
Download complete!
- - - - - - - - - - - - - - - -
```

```
- - - - - - - - - - - - - - - -
Jol23 hi! select number!
1. Upload your list (you must have upload dir)
2. take a look at list(server)
3. take a look at list(yours)
4. logout
5. Server setting
- - - - - - - - - - - - - - - -

accept() is OK...

Jo_1.txt
Download Start!
Your file downloaded 1 line
Your file downloaded 2 line
Your file downloaded 3 line
Download End!
```

```
wntjdals login!
wntjdals take a look server list
wntjdals try to download file
wntjdals's download success
```

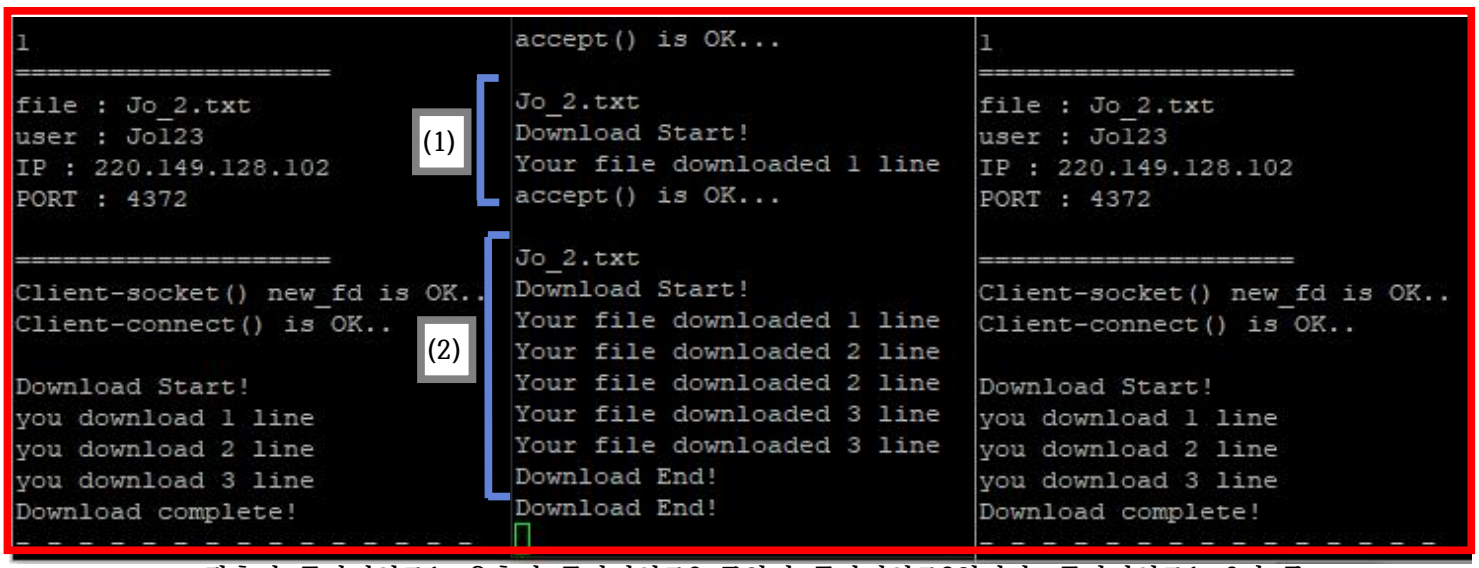
좌측은 클라이언트1에서 클라이언트2의 파일을 다운로드하는 과정이고, 우측상단은 클라이언트2의 로그, 우측 하단은 서버의 로그입니다.

2. 다운로드 후 클라이언트1의 download 폴더와 내용 비교입니다.



상단이 클라이언트1의 download 폴더에 저장된 Jo_1.txt 파일이고 하단이 클라이언트2의 upload 폴더에 저장된 Jo_1.txt 파일입니다. 성공적으로 파일을 다운로드 한 것을 볼 수 있습니다.

3. 두 개의 클라이언트가 동시에 같은 파일을 다운로드하는 상황입니다. 클라이언트2의 Jo_2.txt 파일로 진행하였습니다.



좌측이 클라이언트1, 우측이 클라이언트3 중앙이 클라이언트2입니다. 클라이언트1, 3이 동시에 같은 파일을 요구하는 상황입니다.

(1)-클라이언트1이 Jo_2.txt 파일을 다운로드 하는 상황.

(2)-클라이언트2가 동일 파일을 다운로드 요청하여 클라이언트2의 요청 이후 병행(동시) 다운로드를 진행하는 상황.

*데모 확인을 위해 클라이언트2에 로그가 남지만, 메뉴에 맞는 입력만 해주면 언제든지 메인 서버에 대한 클라이언트로 동작 가능합니다.

4. 각 클라이언트의 파일 내용 확인입니다.

```
st20141460420602-c:~/server/download$ ls
Jo_1.txt  Jo_2.txt
st20141460420602-c:~/server/download$ vi Jo_1.txt
st20141460420602-c:~/server/download$ vi Jo_2.txt
  1 this is test txt file
  2 Jol23 has this file
  3 Thank you
~
```

클라이언트1의 다운로드 폴더와 받은 파일입니다. 시나리오1,2에서 받은 파일은 그대로 있고 Jo_2.txt가 새로 받은 파일입니다.

```
st20141460420602-b:~/New folder/download$ ls
Jo_2.txt
st20141460420602-b:~/New folder/download$ vi Jo_2.txt

  1 this is test txt file
  2 Jol23 has this file
  3 Thank you
~
```

클라이언트3의 다운로드 폴더와 받은 파일입니다. 빈 폴더에 새로 받은 Jo_2.txt가 생긴 것을 볼 수 있습니다.

```
st20141460420602-a:~/server/upload$ ls
Jo_1.txt  Jo_2.txt
st20141460420602-a:~/server/upload$ vi Jo_1.txt
st20141460420602-a:~/server/upload$ vi Jo_2.txt
  1 this is test txt file
  2 Jol23 has this file
  3 Thank you
~
```

클라이언트2의 업로드 폴더와 Jo_2.txt파일 내용입니다. 다운로드를 성공적으로 완료한 것을 확인할 수 있습니다.

5. 클라이언트가 파일을 잘못 선택하는 상황입니다.

```
- - - - - - - - - - - - - - - - -
wntjdals hi! select number!
1. Upload your list (you must have upload dir)
2. take a look at list(server)
3. take a look at list(yours)
4. logout
5. Server setting
- - - - - - - - - - - - - - - - -
2
========list========
1. Jo_2.txt Jol23 220.149.128.102 4372
2. Jo_1.txt Jol23 220.149.128.102 4372
3. test3.txt wntjdals 220.149.128.101 4371
4. test2.txt wntjdals 220.149.128.101 4371
====================
====================
select number
1. down load file
2. quit
====================
1
====================
select file
====================
3
you can't download yourfile
```

(1)

```
- - - - - - - - - - - - - - - - -
wntjdals hi! select number!
1. Upload your list (you must have upload dir)
2. take a look at list(server)
3. take a look at list(yours)
4. logout
5. Server setting
- - - - - - - - - - - - - - - - -

2
========list========
1. Jo_2.txt Jol23 220.149.128.102 4372
2. Jo_1.txt Jol23 220.149.128.102 4372
3. test3.txt wntjdals 220.149.128.101 4371
4. test2.txt wntjdals 220.149.128.101 4371
====================
====================
select number
1. down load file
2. quit
====================
1
====================
select file
====================
5
please check file number
```

(2)

(1) - 자신의 파일을 선택하면 오류로 감지하고 해당 경고문을 출력합니다.

(2) - 목록에 없는 번호를 선택하면 오류로 감지하고 해당 경고문을 출력합니다.

6. 클라이언트(User Jo123)가 메뉴에서 Server setting 메뉴를 통해 다운로드를 일시적으로 차단하는 상황입니다.

```
Jo123 hi! select number!
1. Upload your list (you must have upload dir)
2. take a look at list(server)
3. take a look at list(yours)
4. logout
5. Server setting
- - - - - - - - - - - - - - - - - -
5
====================
select number
1. Download block
2. Download permissinon
====================
1
- - - - - - - - - - - - - - - - - -
```

클라이언트2(user Joo123)에서 Server seting 메뉴를 통해 다운로드를 일시적으로 금지하는 상황입니다.

```
========list========
1. Jo_2.txt Jo123 220.149.128.102 4372
2. Jo_1.txt Jo123 220.149.128.102 4372
3. test3.txt wntjdals 220.149.128.101 4371
4. test2.txt wntjdals 220.149.128.101 4371
====================
====================
select number
1. down load file
2. quit
====================
1
====================
select file
====================
1
====================
file : Jo_2.txt
user : Jo123
IP : 220.149.128.102
PORT : 4372

====================
Client-socket() new_fd is OK..
Client-connect() is OK..

Server is closed..
- - - - - - - - - - - - - - - - - -
```

클라이언트1(user wntjdals)에서 user Joo123에게 다운로드 요청을 보내서 연결을 시도합니다. 연결은 성공하지만 서버가 막혀있다는 문구와 함께 다운로드는 실패하는 상황입니다.

[중요 코드설명]

다운로드와 관련된 코드(서버)입니다.

```
if(strcmp(msgg,"2") == 0)
{
    memset(msgg,0,40);
    //do nothing
}
else if(strcmp(msgg,"1") == 0)
{
    memset(msgg,0,40);
    printf("%s try to download file\n",id);
    rcv_byte = recv(new_fd,msgg,sizeof(msgg),0);
    row = atoi(msgg); //문자열을 인트로 변경
    memset(msgg,0,40);
    upload = fopen("Total_list.txt","r");
    temp_row = 1;
    memset(msg,0,512);
    while(1)
    {
        fgets(msg,sizeof(msg),upload);
        if(temp_row == row)
        {
            if(strcmp(msg,"\0") == 0)
            {
                send(new_fd,"please check file number\n",sizeof(msg),0)
                break;
            }
            else
            {
                send(new_fd,"Uploader Information\n",sizeof(msg),0);
                send(new_fd,msg,sizeof(msg),0);
                memset(msg,0,512);// 다운로드 정보 전송
                break;
            }
        }
        else
        {
            memset(msg,0,512);
            temp_row++;
        }
    }
    memset(msgg,0,40);
    rcv_byte = recv(new_fd,msgg,sizeof(msgg)+1,0);
    printf("%s\n",msgg);
    fclose(upload);
}
```

클라이언트에서 리스트에있는 파일을 선택하면 수행되는 부분입니다. 변수 row를 사용하여 잘못된 입력을 감지하는 알고리즘을 구현하였습니다.

ex) 리스트가 4번까지 있는데 5번을 입력하는 경우.

정상적인 선택을 했다면 다운로드를 시작할 준비를 합니다.

다운로드와 관련된 코드(클라이언트)입니다.

```
else
{
    memset(buf,0,512);
    rcv_byte = recv(sockfd,down_data,sizeof(down_data),0);
    data = strtok(down_data, " ");
    while (data != NULL)                // 자른 문자열이 나오지 않을 때까지 반복
    {
        sprintf(list_data[index],"%s",data);
        index++;
        data = strtok(NULL, " ");       // 다음 문자열을 잘라서 포인터를 반환
    }
    index = 0;
    if(strstr(id,list_data[2]) != NULL)
    {
        printf("you can't download yourfile\n");
        memset(msg,0,40);
        sprintf(msg,"%s fail to download fail\n",id);
        send(sockfd,msg,sizeof(msg),0);
        memset(msg,0,40);
    }
    else
    {
        printf("====================\n");
        printf("file : %s\n",list_data[1]);
        printf("user : %s\n",list_data[2]);
        printf("IP : %s\n",list_data[3]);
        printf("PORT : %s\n",list_data[4]);
        printf("====================\n");
        new_fd = socket(AF_INET, SOCK_STREAM,0);
        memset(msg,0,40);
```

목록에 있는 파일을 선택했다면 파일에 대한 정보를 수신하게 되는 부분입니다. 한 줄에 파일이름, 유저, IP, PORT 가 모두 담겨있기 때문에 띄어쓰기로 정보를 나누어 2차원 배열에 저장하게 됩니다. (변수 list_data = 2차원 배열 선언)

먼저 세 번째 배열의 유저의 이름 정보를 통해 자신의 파일인지 아닌지를 검사하게 됩니다. 만약 자신의 파일을 선택했다면 경고문과 함께 다운로드에 실패하게 되고, 그렇지 않다면 다운로드받을 파일의 정보를 한눈에 알아보기 쉽게 정렬하여 표시해줍니다. 리스트의 목록을 띄어쓰기로 구분했기 때문에 파일의 번호부터 포트번호까지 순차적으로 배열에 들어가고, 파일번호를 제외한 정보들이 표기가 됩니다.

다운로드와 관련된 코드(클라이언트)(Cont'd)

```
if(error_flag == 1)//서 버 가  닫 혀 있 으 면  에 러 !
{
    printf("%s's p2p server is closed\n",list_data[2]);
    memset(msg,0,40);
    sprintf(msg,"%s's download fail\n",id);
    send(sockfd,msg,sizeof(msg)+1,0);
    error_flag = 0;
}
else
{
    memset(msg,0,40);
    rcv_byte = recv(new_fd,msg,sizeof(buf),0);
    if(strstr(msg,"Start") == NULL)
    {
        printf("%s\n",msg);
        memset(msg,0,40);
    }
    else if(strstr(msg,"Start") != NULL)
    {
        int temp_i = 1;
        printf("%s\n",msg);
        send(new_fd,list_data[1],sizeof(buf),0);
        sprintf(temp,"./download/%s",list_data[1]);
        f = fopen(temp,"w");
        memset(buf,0,512);
        while(1)
        {
            rcv_byte = recv(new_fd,buf,sizeof(buf),0);
            if(strcmp(buf,"END") == 0)
            {
                memset(buf,0,512);
                break;
            }
            else
            {
                fprintf(f,"%s",buf);
                memset(buf,0,512);
                printf("you download %d line\n",temp_i);
                temp_i++;
            }
        }
        printf("Download complete!\n");
        fclose(f);
    }
    memset(msg,0,40);
    send(new_fd,"end",sizeof(msg),0);//끝 내 는  시 그 널
    close(new_fd);
    sprintf(msg,"%s's download success\n",id);
    send(sockfd,msg,sizeof(msg),0);
}
```

파일에 대한 정보를 이용해서 또 다른 클라이언트에 접속하게 되는데, 만약 해당 클라이언트의 서버에 연결이 되지 않는다면 에러문구와 함께 다운로드를 실패합니다.

또 다른 클라이언트에게 "Download Start!"라는 문구를 수신하면서 파일을 한 줄씩 다운받게 됩니다. 병행 다운로드를 구현하기 위해 현재 몇 줄 까지 다운로드 했는지 표시하게 했습니다. strcmp함수로 "END"문자열을 감지하여 반복문에서 탈출한 후 다운로드 알고리즘을 종료하게 됩니다.

다운로드와 관련된 코드(클라이언트)(Cont'd)

- 다음은 클라이언트가 파일 전송을 위해 다운로드를 원하는 클라이언트의 서버가 되는 코드입니다.

```
else if (pid == 0)
{
    while(1)
    {
        sin_size = sizeof(struct sockaddr_in);
        new_fd = accept(p2p_fd, (struct sockaddr *)&their_addr, &sin_size);
        pid_t pid_1;
        pid_1 = fork();
        int down_load_flag = 0;
        if (pid_1 > 0)
        {
            close(new_fd);
        }
        else if(pid_1 == 0)
        {
            if(*static_signal == 0)
            {
                send(new_fd,"Server is closed..",sizeof(msg),0);
            }
            else
            {
                send(new_fd,"Download Start!",sizeof(msg),0);
                printf("accept() is OK...\n\n");
                memset(msg,0,40);
                rcv_byte = recv(new_fd,msg,sizeof(msg),0);//파일명
                printf("%s\n",msg);
                dir_info = opendir("./upload");
                if(NULL != dir_info)//잘못된 dir이 아니라면 ! // upload 폴더에 파일이 있는지 다시한번 확인!
                {
                    while(dir_entry = readdir(dir_info))//파일을 하나씩 읽어줌
                    {
                        if(strcmp(dir_entry->d_name,msg) ==0)
                        {
                            down_load_flag = 1;
                        }
                    }
                    closedir(dir_info);
                }
```

p2p 통신을 위해 클라이언트 코드도 서버의 코드와 마찬가지로 fork를 사용하게 됩니다. 서버 코드와는 반대로 자식 프로세스에서 accept를 수행합니다.

병행 다운로드 알고리즘을 구현하기 위해 자식 프로세스안에서 fork를 한번 더 실행하였습니다. 데모결과 오류 없이 병행 다운로드를 성공하였습니다.

공유 메모리 static_signal을 사용하여 다음 두 가지 알고리즘을 구현하였습니다.

- 클라이언트 코드를 실행하더라도 로그인을 하기 전에 초기값을 0으로 하여 로그인도 하지 않았는데 다른 클라이언트에서 다운로드가 가능했던 오류를 해결.
- 값이 1일때만 다운로드를 가능하게 설정하여 사용자가 값을 0으로 설정하게 되면 다운로드 잠금 기능이 됨.

다운로드와 관련된 코드(클라이언트)(Cont'd)

```
if(down_load_flag == 1)
{
    int temp_i = 1;
    printf("Download Start!\n");
    sprintf(temp,"./upload/%s",msg);
    f = fopen(temp,"r");
    memset(msg,0,40);
    memset(buf,0,512);
    while(1)
    {
        fgets(buf,sizeof(buf),f);
        if(strcmp(buf,"\0") == 0)
        {
            send(new_fd,"END",sizeof(msg),0);
            memset(buf,0,512);
            break;
        }
        else
        {
            send(new_fd,buf,sizeof(buf),0);
            memset(buf,0,512);
            printf("Your file downloaded %d line\n",temp_i);
            temp_i++;
            sleep(1);
        }
    }
    printf("Download End!\n");
    fclose(f);
}
else
{
    send(new_fd,"Download Fail!",sizeof(msg),0);
    printf("fail!\n");
}
```

실제 다운로드를 시작하고 나서는 기존의 파일전송에서 사용한 알고리즘과 크게 차이는 없습니다. 병행 다운로드 확인을 위해 sleep 함수로 1초씩 딜레이를 주었습니다. 만약 다운로드를 실패했다면, 다운로드를 시도한 클라이언트에게 "Download Fail!" 문자열을 송신하게 됩니다.

[추가적인 코드설명 - 클라이언트]

다운로드 알고리즘에서 설명하지 못한 부분입니다.

```
            else if(strcmp(msg,"4") == 0)
            {
                memset(msg,0,40);
                break;//elseif "4" ==  logout
            }
            else if(strcmp(msg,"5") == 0)
            {
                memset(msg,0,40);
                printf("====================\n");
                printf("select number\n1. Download block\n2. Download permissinon\n");
                printf("====================\n");
                scanf("%s",msg);
                if(strcmp(msg,"1") == 0)
                {
                    *static_signal = 0;
                }
                else if(strcmp(msg,"2") == 0)
                {
                    *static_signal = 1;
                }
                else
                {
                    printf("Plz check option number\n");
                }
            }
            memset(buf,0,512);
            rcv_byte = recv(sockfd,buf,sizeof(buf),0);
            printf("%s\n",buf);
            memset(buf,0,512);
        }
        break;
    }
}
close(p2p_fd);
close(sockfd);
kill(pid,SIGINT);
return 0;
```

메뉴에서 4번을 누르게 되면 메인 while 구문을 탈출하고 전체 코드를 종료하게 됩니다. Ctr + c 명령어가 아닌 번호 입력을 통한 종료를 위해 부모 프로세스의 말단에서 kill(pid,SIGINT)구문으로 동시 정상 종료를 구현하였습니다.

Server setting 메뉴입니다. 번호 입력에 따라 공유변수 static_signal의 값을 변경해주어 다운로드 차단기능을 구현하였습니다.

[소감]

처음에 리눅스에서 디렉토리 이름 변경하는 과정도 어려워 했는데, 텀 프로젝트를 수행하면서 리눅스 환경과 많이 친숙해지고, 리눅스를 다루는 능력도 많이 성장하게 된 것 같습니다. 처음에는 3단계까지 도달하기도 어려울 것 같았는데, 막상 차근차근 진행하다 보니 여러 가지 아이디어들이 떠올라 몇 가지 기능들을 추가하게 됐습니다. 윈도우에서 C언어, JAVA를 이용한 텀 프로젝트는 분담 후 합치는 과정이 어렵지 않았는데, 리눅스에서 분담했던 내용을 합치는 과정이 생각보다 오래 걸렸습니다. 원래는 make를 사용해서 분담한 내용을 합쳐보려고 했었지만 실패했는데, 학기가 끝나고 세 단계의 텀 프로젝트를 정리하면서 다시 한번 도전해 보기로 하였습니다.

저희 조는 가장 현실적인 프로젝트를 구현하는 것을 목표로 하였습니다. 서버가 다운되도 회원들의 정보가 유지되는 회원가입, 동시에 두 클라이언트에 파일을 전송할 수 있는 병행 다운로드, 잘못된 입력에 대해서는 오류로 판단하고 무시할 수 있는 견고함 이 세 가지에 중점을 두고 프로젝트를 수행하였습니다.

운영체제에 관한 이론과 실습을 통해 원초적인 운영체제인 리눅스를 직접 경험하면서 많은 경험과 성장을 하게 되었습니다. 좋은 강의를 만들어 주신 교수님께 감사하며 텀 프로젝트를 마치겠습니다.