[Operating System Project03]

Users

한양대학교 컴퓨터소프트웨어학부
2015005141 최연수
수업번호 : 11797

1. 프로젝트 목표

- xv6 파일 시스템에는 유저의 개념을 도입하여 파일들에 접근 권한을 둘 수 있도록 한다.

2. 구현 명세

2.1 User accounts

하나 또는 다수의 유저(root 포함 최대 10 개)가 등록될 수 있도록 기능을 추가한다.
각 유저는 유저의 이름, 비밀번호(길이가 1 이상 15 이하의 알파벳 혹은 숫자)를 가지고있고,
root 를 제외하고 자신의 디렉토를 가지고있다.
유저의 이름/비밀번호 목록은 root DIR 의 파일에 저장한다.
초기 root 의 비밀번호는 1234 이다.

- Login / Logout

  올바른 유저의 이름/비밀번호가 입력되면 로그인된다.
  logout 을 통해 sh 을 종료할 수 있으며, 종료 시 로그인화면으로 돌아간다.
  특정 사용자로 로그인하면 프로세스가 해당 유저의 권한으로 실행된다.

- Add user / Delete user

  유저를 생성하고 삭제하며 두가지의 시스템콜이 존재한다.
  두 시스템콜은 root 만이 호출 가능하며, root 계정이 아니면 -1 을 반환한다.

  - int useradd(char *username, char *password)

    username 과 password 를 입력받아 해당 유저를 생성한다.
    성공적으로 생성되면 0 을 반환한다.
    이름이 이미 존재하거나, 더이상 유저를 추가할 공간이 없는 경우 -1 을 반환한다.

  - int userdel(char *userrname)

    username 의 이름을 가진 유저를 삭제하지만 root 는 지울 수 없다,
    성공적으로 삭제하면 0 을 반환하고, root 계정을 삭제하려하거나 존재하지 않는 유저를
    삭제하려 한 경우 -1 을 반환한다.

3. 구현

3.1 systemcall

user.h

```
int openfile(char*);
int useradd(char*, char*);
int userdel(char*);
void retusername(char*);
```

usys.S

```
SYSCALL(openfile)
SYSCALL(useradd)
SYSCALL(userdel)
SYSCALL(retusername)
```

defs.h

```
int             openfile(char *path);
int             useradd(char *username, char *password);
int             userdel(char *username);
void            retusername(char *username);
```

syscall.c

```
extern int sys_openfile(void);
extern int sys_useradd(void);
extern int sys_userdel(void);
extern int sys_retusername(void);
```

```
[SYS_openfile] sys_openfile,
[SYS_useradd]  sys_useradd,
[SYS_userdel]  sys_userdel,
[SYS_retusername]   sys_retusername,
```

syscall.h

```
#define SYS_openfile 32
#define SYS_useradd 33
#define SYS_userdel 34
#define SYS_retusername 35
```

3.2 필요 함수 구현

3.2.1 전역변수

```
char userlist[10][31];
char for_retusername[15];
```

'username & pawword 가 최대 15 이고, username 과 password 를 구분하기위한 공백을 함께 저장하는데, 총 10 개를 저장하기 위해 (10 x 31) 배열을 선언한다.
어떤 user 로 login 했는지 저장하기 위해 for_retusername[15]을 선언한다.

3.2.2 mystrcmp

```
int
mystrcmp(const char *p, const char *q)
{
  while(*p && *p == *q)
    p++, q++;
  return (uchar)*p - (uchar)*q;
}
```

두 문자열이 같은지 비교하는 함수이다. 같다면 0 을 return 한다.

3.2.3 mystrcat

```
char*
mystrcat(char *username, const char *password)
{
    int i = strlen(username)-1;
    int j = 0;
    int fin = strlen(password)-1;

    username[i++]=' ';
    while(j<fin){
        username[i++] = password[j++];

    }
    return username;
}
```

username 과 password 를 공백으로 구분가능한 하나의 문자열로 만들어준다.
get 으로 username 과 password 를 받을 땐 마지막에 개행을 함께 받기때문에 길이에서 1 씩 빼준다.
login 시 쉽게 구별하기 위해 구현한다.

3.2.4 mystrcat2

```
char*
mystrcat2(char *username, const char *password)
{
    int i = strlen(username);
    int j = 0;
    int fin = strlen(password);

    username[i++]=' ';
    while(j<fin){
        username[i++] = password[j++];

    }
    return username;
}
```

mystrcat 과 같은 역할을 하지만 공백이 입력되지 않는 경우(argument 로 인자를 받은 경우)를 위해 선언한다.

3.2.5 strcpy

```
char*
strcpy(char *dest, const char *src){
    char *tmp = dest;
    while((*dest++ = *src++) != '\0')
        ;
    return tmp;
}
```

src 에 있는 내용을 dest 로 복사해준다.

3.2.6 mystrcpy

```
char*
mystrcpy(char *dest, const char *src)
{
    char *tmp = dest;
    while((*dest++ = *src++ - 1)!= '\0')
        ;
    return tmp;
}
```

strcpy 와 같은 역할을 하지만 src 마지막에 개행이 있는 경우를 위해 선언한다.

3.3 User Accounts

3.3.1 openfile

```
{
  int pid, wpid;

  if(open("console", O_RDWR) < 0){
    mknod("console", 1, 1);
    open("console", O_RDWR);
  }
  dup(0);  // stdout
  dup(0);  // stderr

  openfile("userlist");
```

openfile 함수는 init.c 에서 유저프로세스가 만들어진 후 openfile 시스템콜을 이용해서 호출된다.

```
int
openfile(char *path)
{
  struct inode *ip, *dp;
  char name[DIRSIZ];
  char src[31] = "root 1234";
  char tmp[31]="t";


  if((ip = namei(path)) == 0){
      if((dp = nameiparent(path, name)) == 0)
          return 0;
      ilock(dp);
      begin_op();
      if((ip = dirlookup(dp, name, 0)) != 0){
          iunlockput(dp);
          ilock(ip);
          if(ip->type == 2){// T_FILE : 2
              iunlockput(ip);
              end_op();
              return 1;
          }
          iunlockput(ip);
          end_op();
          return 0;
      }
      if((ip = ialloc(dp->dev, 2)) == 0)
        panic("create: ialloc");

      ilock(ip);
      ip->major = 3;// T_DEV
      ip->minor = 3;
      ip->nlink = 1;// T_DIR
      iupdate(ip);
      char init[10][31];
      if(dirlink(dp, name, ip->inum) < 0)
          panic("create: dirlink");
      for(int i=0 ; i<10; i++){
          for(int j=0 ;j<31; j++)
              init[i][j]='\0';
      }

      writei(ip,*init,0,sizeof(init));
      writei(ip, src, 0, 31);
      iupdate(ip);
      iunlockput(ip);
      iunlockput(dp);
      end_op();
      count[0]++;

      for(int i=0 ; i< sizeof(src) ; i++){
          userlist[0][i] = src[i];
      }
}
```

userlist 파일이 없다면 새로 파일을 만들어 주고 최초유저인 root 유저의 정보를 파일에 써준다. 이때, 파일의 비어있는 공간들을 '\0'으로 초기화해준다.
root 유저에 대한 정보를 전역변수 userlist 의 0 번지에 저장해준다.

```
else{
    for(int i=0;i<10;i++){
        for(int j=0 ; j<31; j++){
            userlist[i][j] = '\0';
        }
    }
    begin_op();
    ilock(ip);
    for(int i=0 ; i< 10; i++){
        readi(ip, tmp ,i * 31,31);
        for(int j=0 ; j< sizeof(tmp) ; j++){
          userlist[i][j] = tmp[j];
        }
    }
    iunlockput(ip);
    end_op();
}
```

```
int
sys_openfile(void){
    char *path;
    if(argstr(0,&path)<0)
        return -1;
    return  openfile(path);

}
```

userlist 파일이 존재하는 경우엔 파일의 내용을 읽어와서 전역변수 배열에 다시 저장하여 유저 목록을 만든다.

3.4 login

```
char *argv[] = { "login", 0 };
  for(;;){


    pid = fork();
    if(pid<0){
        printf(1,"init login: fork failed\n");
        exit();
    }
    if(pid == 0){
        exec("login",argv);
        printf(1, "init: exec login failed\n");
        exit();
    }
    while((wpid=wait()) >= 0 && wpid != pid)
        printf(1,"zombie!\n");

  }
```

init.c 에서 xv6 가 부팅 후에 sh 대신 login 을 실행하도록 한다.

```
char *argv[] = {"sh",0};


int
main()
{
    char username[15];
    char password[15];
    char userinfo[15];
    char userinfo_tmp[15];
    char *total;
    int fd,pid;
    int ret=-1;
    char tmp[10][31];

    while(1){
        fd = open("userlist", T_FILE);
        read(fd, &tmp,sizeof(tmp));
        ret = -1;
        for(int i =0 ; i<15; i++){
            username[i]='\0';
            password[i]='\0';
            userinfo[i]='\0';
            userinfo_tmp[i]='\0';
        }

        printf(1,"username: ");
        gets(username,sizeof(username));
        printf(1,"password: ");
        gets(password,sizeof(password));
        strcpy(userinfo_tmp,username);

        total = mystrcat(username,password);
        for(int i = 0 ; i <10; i++){
            if(mystrcmp(tmp[i],total)==0){
                ret = 1;
                break;
            }
        }
        if(ret==1){
            for(int i=0 ;i<sizeof(userinfo_tmp); i++){
                if(userinfo_tmp[i]=='\n')
                    break;
                userinfo[i]=userinfo_tmp[i];
            }
            retusername(userinfo);

            pid = fork();
            if(pid <0){
                printf(1,"login: fork failed\n");
                exit();
            }
            if(pid == 0){
                exec("sh",argv);
                printf(1,"login: exec sh failed\n");
                exit();
            }
            else
                pid = wait();
        }
        else
            printf(1,"Wrong login information\n");
    }
}
```

login 은 유저프로그램으로, login.c 를 작성한다.

usrlist 에 저장된 내용을 읽어와 tmp 에 저장한다.

username 과 password 를 받아 띄어쓰기로 구분되는 하나의 문장으로 만든 후 위에서 읽어온 내용과 비교한다.

userlist 와 일치한다면 등록된 user 이므로 retusername 함수를 통해서 kernel 에 user 정보를 전달한 후 sh 을 실행하고 wait 상태에 들어간다

일치하지 않는다면 wrong message 를 띄운 후 다시 로그인 시도를 하게 한다.

```
void
retusername(char* username){
    strcpy(for_retusername, username);
}

int
sys_retusername(void){
    char *username;
    if(argstr(0,&username)<0)
        return -1;
    retusername(username);
    return 1;
}
```

retusername 은 인자로 입력받은 값을 전역변수 for_retusername 에 복사해주는 함수이다.

3.5 logout

```
while(getcmd(buf, sizeof(buf)) >= 0){
  if(buf[0] == 'c' && buf[1] == 'd' && buf[2] == ' '){
    // Chdir must be called by the parent, not the child.
    buf[strlen(buf)-1] = 0;  // chop \n
    if(chdir(buf+3) < 0)
      printf(2, "cannot cd %s\n", buf+3);
    continue;
  }
  if(buf[0] =='l' && buf[1] =='o' && buf[2]=='g' &&
     buf[3] =='o' && buf[4]=='u' && buf[5] == 't' && buf[6] == '\n'){
     exit();
  }
```

sh.c 에 logout 을 입력하면 shell 을 종료하는 내장 명령을 추가하였다.
종료가 되면 wait 상태였던 login 으로 다시 돌아가 로그인을 다시 시도하게 한다.

3.6 useradd

```
int
sys_useradd(void){

    char *username,*password;
    char *root = "root";
    if(mystrcmp(for_retusername,root))
        return -1;
    if(argstr(0,&username)<0)
        return -1;
    if(argstr(1,&password)<0)
        return -1;
    return useradd(username,password);
}
```

현재 user(for_retusername)가 root 가 아니라면 -1 을 반환한다.

```
int
useradd(char *username, char *password){
    char *total;
    char dirpath[15];
    char check[16];
    char *path = "userlist";
    int next;
    struct inode *ip;
    next = find_next();
    if(next == -1){
        return -1;
    }
    else{
        for(int i=0 ; i< 10 ; i++){
            for(int k=0 ; k< 16 ; k++)
                check[k] = '\0';
            for(int j=0; j< 16; j++){
                if(userlist[i][j]==' ')
                    break;
                check[j]=userlist[i][j];
            }
            if(mystrcmp(check, username) == 0)// there's match
                return -1;
        }
```

find_next 함수를 이용해, 유저를 추가 할 공간이 없는 경우엔 -1 을 반환한다. 입력받은 값을, 저장되어있는 유저 목록과 비교한 후 같은 이름을 가진 유저가 이미 존재한다면 -1 을 반환한다.

```
// find next location in userlist
// if it is over 10, return -1
int
find_next(){
    for(int i = 0 ; i <10; i++){
        if(userlist[i][0]=='\0')
            return i;
    }
    return -1;
}
```

find_next 함수는 첫번째 글자가 '\0'인 행을 찾아. (아무것도 저장되어있지 않다는 의미) 그 번호를 반환해준다. 없다면 -1 을 반환한다

```
strcpy(dirpath,username);
total = mystrcat2(username, password);
count[next]++;
for(int i=0 ; i< 31 ; i++){
    userlist[next][i] = total[i];
}

ip = namei(path);
begin_op();
ilock(ip);
writei(ip, total, next*31, 31);
iupdate(ip);
iunlockput(ip);
end_op();
```

예외에 걸리지 않았다면 추가 할 공간이 남아있다는 의미이므로 'username password'를 find_next 로부터 반환받은 번호의 행에 write 해준다.

```
        struct inode *ip2, *dp;
        char name[DIRSIZ];

        if((dp = nameiparent(dirpath,name))==0)
            return 0;
        ilock(dp);
        begin_op();

        if((ip2 = dirlookup(dp,name,0))!=0){
            iunlockput(dp);
            ilock(ip2);

            iunlockput(ip2);
            end_op();
            return 0;

        }
        if((ip2 = ialloc(dp->dev, 1))==0){
            panic("create: ialloc");
        }

        ilock(ip2);
        ip2->major = 3;
        ip2->minor = 3;
        ip2->nlink = 1;
        iupdate(ip2);

        iupdate(dp);
        if(dirlink(ip2,".",ip2->inum) < 0 || dirlink(ip2, "..", dp->inum)<0)
            panic("create dots");

        if(dirlink(dp, name, ip2->inum)<0)
            panic("create: dirlink");

        iupdate(ip2);
        iunlockput(ip2);
        iupdate(dp);
        iunlockput(dp);
        end_op();
    }
    return 0;
}
```

유저가 생성될 때 이 유저와 이름이 같은 이름의 디렉토리를 하나 생성한다.

3.7 userdel

```
int
sys_userdel(void){
    char* root = "root";
    if(mystrcmp(for_retusername,root))
        return -1;
    char *username;
    if(argstr(0,&username)<0)
        return -1;
    return userdel(username);
}
```

현재 user(for_retusername)가 root 가 아니라면 -1 을 반환한다.

```
int
userdel(char *username){
    int del,cnt=0;
    char *path = "userlist";
    struct inode *ip;

    del = exist(username);
    if(del == 0)
        return -1;
    if(del == -1)
        return -1;
    for(int i = 0 ; i < 31; i++){
        userlist[del][i] = '\0';
    }
    count[del]--;
    for(int i=0 ; i<10;i++)
        cnt+= count[i];
    ip = namei(path);
    begin_op();
    ilock(ip);
    writei(ip,*userlist ,0, sizeof(userlist));
    iupdate(ip);
    iunlockput(ip);
    end_op();

    return 0;
}
```

exist 함수를 이용하여 입력받은 값이 존재하는 유저인지 확인한다.
지우려는 값이 root 이거나(del 이 0), 존재하지않는다면 (del 이 -1) -1 을 return 한다.
그렇지 않다면 지울 값이 있다는 의미이므로 해당 행을 모두 '\0'로 초기화해준다.
이후 update 된 userlist 를 파일에 다시 써준다.

```
int
exist(char *username){
    char store[15];
    char blank = ' ';
    for(int i=0 ; i< 10 ; i++){
        for(int i=0 ; i<15; i++){
            store[i] = '\0';
        }
        for(int j=0 ; j< 15; j++){
            if(userlist[i][j] == blank)
                break;
            store[j] = userlist[i][j];

        }
        if(mystrcmp(store, username) == 0){
            return i;
        }
    }

    return -1;
}
```

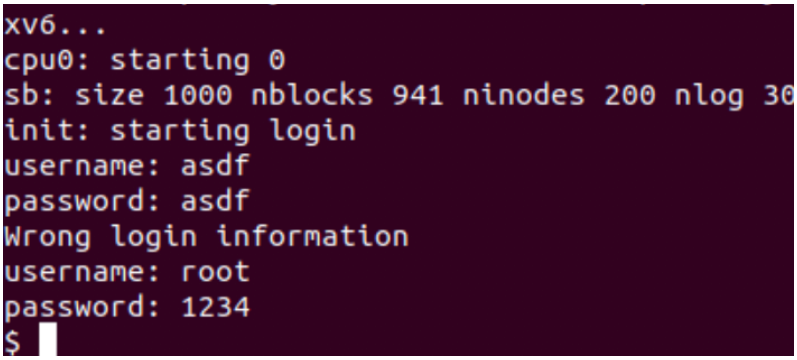
입력받은 username 과 일치하는 행을 찾아 그 행의 번호를 반환한다.
존재하지 않는다면 -1 을 반환한다.

4. 실행방법

1. xv6-public 폴더에서 "make clean"을 입력한다.
2. 이후 "make fs.img"를 입력해준다.
3. "./run_xv6.sh"로 프로그램을 실행시켜준다.
4. 초기값인 username: root, password : 1234 로 로그인 할 수 있다.

5. 실행결과

5.1 로그인 실패 & 성공

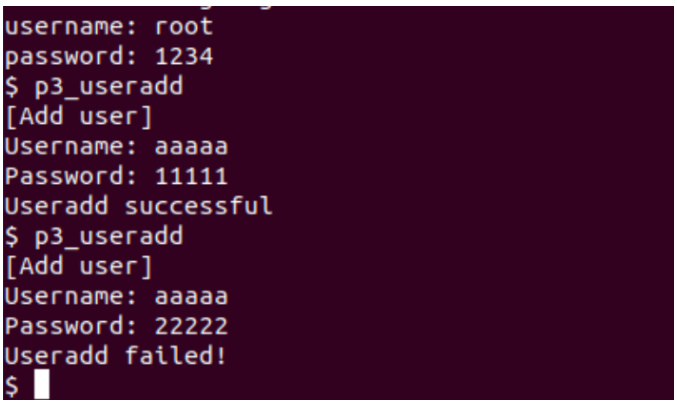


처음 root 계정 이외에 user 를 입력했을 때, wrong 메시지와 함께 다시 로그인을 요구함을 볼 수 있다.
root 계정을 입력시에는 로그인에 성공한다.

5.2 useradd

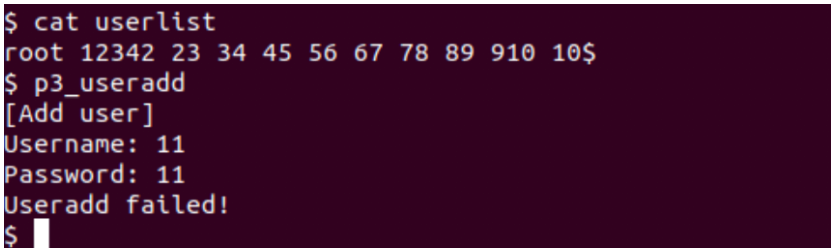
root 계정으로 로그인하여 'asdf' user 를 추가하면, 성공했다는 메시지와 함께 DIR 이 추가됨을 볼 수 있다.

5.3 이미 있는 이름 add 시도



같은 이름을 가진 user 를 추가하려하면 실패한다.

5.4 10 개 이상 user 시도

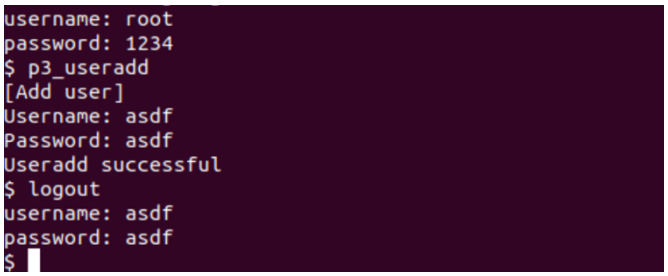


user 를 10 개 이상 추가하려하면 실패한다.

5.5 root 아닌 user 가 useradd

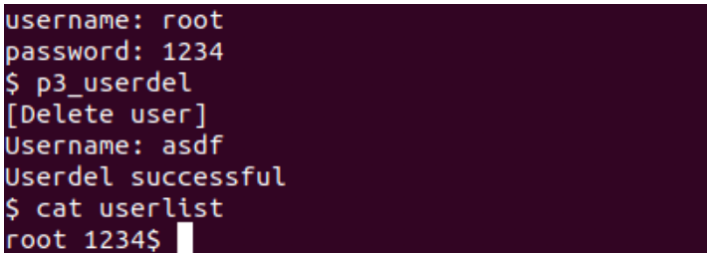
root 가 아닌 asdf(root 가 아닌 user)로 로그인 후 useradd 를 사용하려하면 실패한다.

5.6 로그아웃 후 새로 생성한 user 로 로그인



user asdf 생성 후 로그인이 가능하다.

5.7 userdel



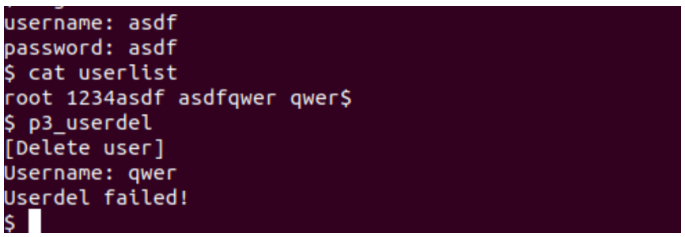
5.8 없는 user 제거시도

root, asdf 만 있는 상태에서 1234 를 지우려하면 실패함을 볼 수 있다.

5.9 root 제거시도

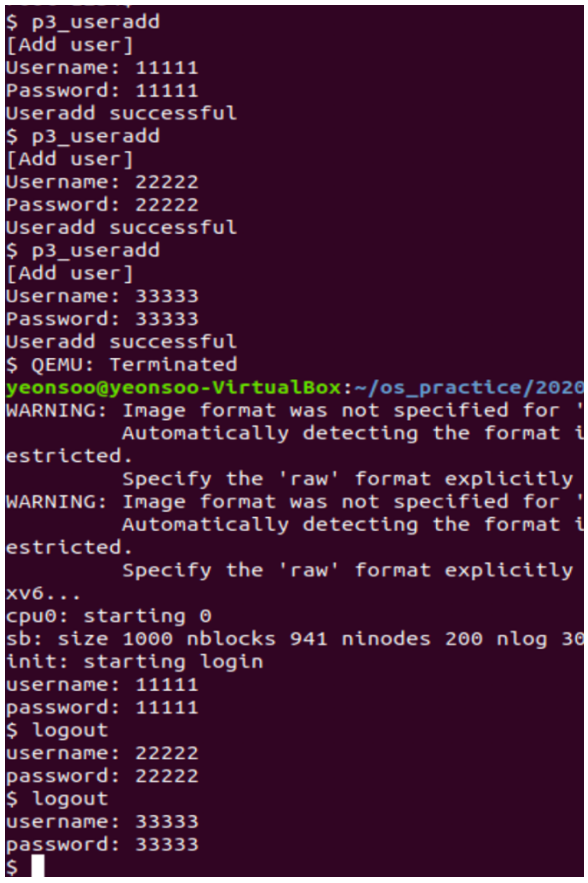
root 를 지우려 시도하면 실패한다.

5.10 root 가 아닌 user 가 userdel



root 가 아닌 user 가 userdel 을 호출하면 실패한다.

5.11 xv6 재시동 후 파일 유지



user 를 추가하고, 재부팅을 해도 그 정보가 그대로 유지된다.

6. 트러블슈팅

6.1 초기화

username 과 password 를 저장할 때, 배열을 사용하다보니, 추가하고 삭제하는 과정에서 사라지지않는 문자들이 발생하였다. 예를들어 username 이 5 자리였다 3 자리가 되면 앞선 username 의 2 개의 문자가 사라지지않고, 뒤에 영향을 주었다. 이를 해결하기 위해서, 문자를 사용하기 전 초기화를 해주었다.

이 문제는 파일에서도 발생하였다. 파일에 유저정보를 적을 때 초기화 없이 적다보니 중간중간 쓰레기값이 들어가게 되었고 컨트롤을 할 수 없게되었다. 때문에 파일을 처음 만들 때 부터, 배열과 파일 모두 '/0'으로 초기화하여 문제를 해결하였다.

6.2 개행제거

문자열을 입력 받을때 gets 을 사용하여 입력을 받았는데, 이때 개행이 문자열에 포함되어 받아져, 같음을 비교할 때 문제가 발생하였다. 이를 해결하기 위해서 strcmp 와 같은 함수를 사용할 때, 개행 앞부분까지만 문자열을 받아 비교를 진행하였다.

6.3 ilock(), begin_op(), end_op()

readi, writei 를 사용하기 위해서 해당 inode 를 사용전에 lock, 사용 후에 unlock 해주어야했고, 변화를 주기전에 begin_op() 끝날 때 end_op()를 입력해주어야 됐는데, return 과 같은 여러 분기지점이 있어서 정확히 들어맞게 이들을 선언하는데 어려움을 겪었다. 하나하나 매칭을 해 가면서 정확하게 구현할 수 있었다.

6.4 user 프로그램의 정보를 kernel 로 전달

어떤 프로세스가 현재 로그인중인지 알기위해선 login.c 에서의 정보를 kernel 영역으로 가져왔어야됐다. 이 정보를 가져오기위한 시스템콜을만들어 해결할 수 있었다.