PROJECT 04

DESIGN

Project04 의 내용은 git의 xv6-public_user 폴더에 구현했음을 참고 부탁드립니다.

User Account

• xv6에 하나 또는 다수의 유저가 등록될 수 있도록 기능을 추가합니다.

• 각 유저는 이름, 비밀번호, 그리고 자신의 디렉토리(root 제외)를 가지고 있습니다.

• 최초에는 root 유저만이 존재합니다. • root 유저는 모든 파일에 대해 그 owner와 동일한 권한을 가집니다.(리눅스에서의 전지전능함과는 차이가 있을 수 있습니다.)

• 유저 이름 및 비밀번호는 모두 알파벳 대소문자 또는 숫자로만 이루어져 있고, 길이는 2이상 15이하로 가정합니다. (예외의 경우를 생각하지 않아도 됩니다.)

• 유저의 최대 수는 root를 포함하여 10입니다.

• Login/Logout

1) 올바른 이름과 비밀번호를 입력하면 로그인 됩니다.

2) logout을 통해 종료할 수 있으며, logout 시 로그인 화면으로 이동합니다.

3) 사용자로 login 시, 프로세스가 해당 유저의 권한으로 실행 됩니다.

• Add User / Delete User

• 유저를 생성하고 삭제하는 시스템콜을 생성합니다.

---

int addUser(char username, char password);

• username의 이름과 password의 비밀번호를 가진 유저를 생성합니다.

• 유저가 성공적으로 생성되었다면 0을 반환합니다.

• 기존에 이미 존재하는 유저 이름인 경우, 혹은 더 이상 유저를 추가할 공간이 없는 경우 -1을 반환합니다.

---

*int deleteUser(char *username);*

• username의 이름을 가진 유저를 삭제합니다.

• root 계정은 삭제할 수 없습니다.

• 성공적으로 삭제되었다면 0을 반환합니다.

• root 계정을 삭제하는 경우, 혹은 존재하지 않는 유저를 삭제하는 경우 -1을 반환합니다.

FILE MODE

• 파일에 owner / others 에 대한 read / write / execute 권한을 부여할 수 있게 합니다. (파일에는 디렉토리도 포함합니다.)

• Owner : 파일을 생성한 유저입니다.

• Others : Owner를 제외한 다른 유저들입니다.

• 최초의 파일 시스템에서의 모든 파일 및 디렉토리의 owner는 root로 합니다.

---

*int chmod(char *pathname, int mode);*

• mode는 다음의 값들을 비트 OR을 수행하여 만들어내는 값입니다.

• #define MODE_RUSR 32 //owner read

• #define MODE_WUSR 16 // owner write

• #define MODE_XUSR 8 // owner execute

• #define MODE_ROTH 4 // others read

• #define MODE_WOTH 2 // others write

• #define MODE_XOTH 1 // others execute

IMPLEMENT

System call 함수

defs.h

```
int             openfile(char *path);
int             addUser(char *username, char *password);
int             deleteUser(char *username);
void                retusername(char *username);
```

---

syscall.h

```
#define SYS_openfile 22
#define SYS_addUser 23
#define SYS_deleteUser 24
#define SYS_retusername 25
```

---

syscall.c

```
extern int sys_openfile(void);
extern int sys_addUser(void);
extern int sys_deleteUser(void);
extern int sys_retusername(void);

extern int sys_openfile(void);
extern int sys_addUser(void);
extern int sys_deleteUser(void);
extern int sys_retusername(void);
```

---

user.h

```
int openfile(char*);
int addUser(char*, char*);
int deleteUser(char*);
void retusername(char*);
```

---

usys.S

```
SYSCALL(openfile)
SYSCALL(addUser)
SYSCALL(deleteUser)
SYSCALL(retusername)
```

User Account 구현 - Login/Logout

init.c

```
openfile("userlist");
printf(1, "init: starting login\n");
```

init.c 파일의 main 함수에서 "userlist" 라는 openfile 시스템 콜 함수를 사용해서 호출을 한다.

추가한 전역 변수

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

user의 정보를 담는 배열인 userlist 와 어떤 user로 login을 했는지 알아보는 for_retusername 을 선언합니다.

---

int openfile(char *path)

```
int
openfile(char *path)
{
  struct inode *ip, *dp;
  char name[DIRSIZ];
  char src[31] = "root 0000";
  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];
        }
    }
```

if 문에서 파일이 없다면, userlist 에 새로운 파일을 만들고, 처음 유저인 root 유저를 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();
    }

      return 1; //ip
    }
```

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

---

init.c

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

    int
    main(void)
    {
      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");
    printf(1, "init: starting login\n");

    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 부팅 후에 login 을 실행하도록 한다.

---

login.c 유저 프로그램을 새로 만들어서 부팅 시, login.c 를 실행한다.

login.c 의 main 함수

```
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);
```

userlist 에서 저장된 내용을 가지고와서 userinfo_tmp 에 저장을 합니다.

```
            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];
                }
                login_user(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");

        }

    }
```

username 과 password를 받아 userlist 에서 가져온 내용과 비교합니다.

userlist 와 일치한다면, 이미 등록된 user이므로 kernel 에 user 정보를 전달한 후, wait 상태로 대기한다.

만약 일치하지 않다면, Wrong login information 라는 문구를 띄운 후, 다시 로그인을 시도 한다.

LogOut

sh.c

```
int
main(void)
{
  static char buf[100];
  int fd;

  // Ensure that three file descriptors are open.
  while((fd = open("console", O_RDWR)) >= 0){
    if(fd >= 3){
      close(fd);
      break;
    }
  }

  // Read and run input commands.
  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();
      }
    if(fork1() == 0)
      runcmd(parsecmd(buf));
    wait();
  }
  exit();
}
```

파일의 main 함수에서 logout 이라는 명령어를 입력받으면, exit()를 실행해서 위의 wait 상태였던 것을 다시 login 상태로 돌아가 로그인을 다시 시도한다.

User Add

```
int
addUser(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.

int find_next()

```
int
find_next(){
for(int i = 0 ; i <10; i++){
  if(userlist[i][0]=='\0')
     return i;
   }
return -1;
}
```

find_next 함수는 아무것도 저장되어있지 않다는 의미인 첫번째 글자가 '\0'인 행을 찾아 그 배열의 index 반환해준다. 없다면 -1 을 반환한다.

입력받은 값을, 저장되어있는 유저 목록과 비교한 후 같은 이름을 가진 유저가 이미 존재한다면 -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();
```

예외처리가 되지 않는다면 추가 할 공간이 남아있다는 의미이므로 'username password'를 find_next 로부터 반환받은 번호의 index에 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.

User Delete

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

exist 함수를 사용하여 입력받은 값이 등록된 user인지 확인한다.

int exist(char* username)

```
  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 index를 찾아 그 index의 번호를 반환한다. 존재하지 않는다면 -1 을 반환한다.

```
    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;
      }
```

del 값이 0 (root) 이거나, del 값이 -1 (존재하지 않음) 이면 -1을 return 한다. 그렇지 않다면, 해당의 값을 '\0'으로 초기화 해준다.

Fild Mode 구현 - Change Mode

***fs.h* 에 선언한 mode**

```
#define MODE_RUSR 32 // owner read
#define MODE_WUSR 16 // owner write
#define MODE_XUSR 8  // owner execute
#define MODE_ROTH 4  // others read
#define MODE_WOTH 2  // others write
#define MODE_XOTH 1  // others execute
```

RSULT

Log In/ Log Out

처음 xv6를 실행할 때, 초기 username 인 root 와 password 인 0000을 입력받기 전까지 실행이 되지 않는다.

log in 실패

```
Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap sta8
init: starting login
username: fail
password: 1234
Wrong login information
username: root
password: 1234
Wrong login information
username: fail
password: 0000
Wrong login information
```

log in 성공

```
username: root
password: 0000
$
```

User Add

root 계정으로 log in을 한 다음, useradd_test 인 유저 프로그램을 실행하여 새로운 username 과 비번을 만들어 새로운 계정을 만들 수 있다.

```
$ useradd_test
[Add user]
Username: kim
Password: 9879
Add user successful!
$ ls
.              1 1 512
..             1 1 512
README         2 2 2286
cat            2 3 16412
echo           2 4 15264
forktest       2 5 9572
grep           2 6 18632
init           2 7 15900
kill           2 8 15292
ln             2 9 15148
ls             2 10 17780
mkdir          2 11 15396
rm             2 12 15372
sh             2 13 28160
stressfs       2 14 16280
usertests      2 15 67392
wc             2 16 17148
zombie         2 17 14964
useradd_test   2 18 15508
userdelete_tes 2 19 15420
login          2 20 18852
console        3 21 0
userlist       2 22 310
kim            1 23 32
```

log out을 한 후, 새로운 user id로 로그인

```
Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap sta8
init: starting login
username: kim
password: 9879
$
```

중복되는 username을 만들려하는 경우

```
username: kim
password: 9879
$ useradd_test
[Add user]
Username: kim
Password: 1234
Add user failed!
$
```

logout 을 입력하여 loggout하고 login 을 입력하여 root 계정을 변환

```
$ logout
username: kim
password: 9879
$ login
username: root
password: 0000
$
```

User Delete

root 계정은 삭제 할 수 없으면, root 계정으로 login 되어있을 경우에만 다른 username 을 삭제할 수 있다.

root 계정 삭제 시도

```
$ userdelete_test
[Delete user]
Username: root
Delete user failed!
```

root 계정에서 다른 계정 삭제

```
$ userdelete_test
[Delete user]
Username: kim
Delete user successful!
$
```

```
Booting from Hard Disk..xv6...
cpu0: starting 0
sb: size 1000 nblocks 941 ninodes 200 nlog 30 logstart 2 inodestart 32 bmap sta8
init: starting login
username: kim
password: 9879
Wrong login information
```

TROUBLE SHOOTING

1.문자열 받기

문자열을 받을 때, 띄어쓰기 같은 개행문자까지 문자열로 입력이 되어서 처음 비교할 때, 어려움을 가졌다. 이를 해결하고자 strcmp 같은 함수 등을 사용하여, 개행문자 앞까지만 문자열을 받고, 이를 문자열과 비교했다.

2. user 프로그램의 정보를 kernel 로 전달

login.c 에서의 현재 누가 login 중인지에 관한 정보를 kernel 영역으로 가져와야해서 시스템콜을 만들어 해결했다.

3. ilock(), begin_op(), end_op()이

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 분기점 있어서 정확히 들어맞게 이들을 선언하는데 어려움을 겪었다.