# Operating Systems Project 4 Wiki

2018008904 이성진

## Contents



## 1. Design

- xv6는 기본적으로 한 명의 유저만이 os를 사용한다고 가정합니다. 그래서 파일과 디렉토리에 대한 소유권에 대한 개념 역시 없으며, 모든 파일과 디렉토리를 읽고 쓰거나 실행하는 것에 제한이 없습니다.

- 유저 계정 관리 시스템을 추가하여, 기본 계정이자 관리자 계정인 root를 비롯한 여러 유저들이 os를 사용할 수 있도록 변경합니다.

- 처음 xv6가 부팅되면 init 프로세스가 무한 루프를 돌며 sh 프로세스를 생성하는데, 이 두 프로세스 사이에 로그인 기능을 담당하는 유저 프로그램이 실행되도록 구조를 변경하여 로그인에 성공하면 쉘이 실행될 수 있도록 합니다.

- root 유저로 로그인했을 때만 새로운 유저를 추가하거나 기존 유저를 삭제할 수 있도록 하고, passwd 파일에 유저의 아이디와 비밀번호 등의 정보를 저장하도록 했습니다. 새로운 유저가 추가되면 passwd 파일에 해당 정보를 작성하게 되고, 만약 가능하다면 해당 유저가 소유하며 유저와 같은 이름을 가지는 디렉토리를 생성합니다.

- 특정 유저로 로그인했다는 것은 해당 유저의 권한으로 프로세스가 실행되고 있다고 할 수 있습니다. 각각의 파일과 디렉토리는 소유자와 읽기/쓰기/실행 권한을 가지게 되어, 현재 로그인한 유저에 따라 특정 행동이 가능해지거나 불가능해지는 상황이 발생할 수 있도록 했습니다.

- 각 파일 또는 디렉토리의 소유자(또는 root)는 해당 파일 또는 디렉토리의 읽기/쓰기/실행 권한을 변경할 수 있습니다. 기본적으로 파일에 대한 권한은 소유자의 권한과 소유자가 아닌 다른 유저들의 권한에 각각 3가지 권한으로 총 6가지로 이루어집니다.

- 새로 구현된 유저 계정 및 소유, 권한 시스템을 보조하기 위한 기존 유저 프로그램을 수정과 새로운 유저 프로그램이 필요했습니다. ls는 각 파일 또는 디렉토리의 권한과 소유자를 출력하고, 현재 디렉토리 경로를 출력하는 pwd, 유저를 추가/삭제하는 useradd/delete, 권한을 변경하는 chmod와 여러 쉘 내장 명령(logout, whoami 등)을 추가했습니다.

## 2. Implement

**User accounts implementation**

- 로그인 시스템을 구현하기 위하여 sysfile.c에 아래와 같이 로그인 관련 정보를 담는 구조체 ltable과 함수들을 추가했습니다.

```
#define MAX_USER 10
#define MAX_LEN 15

struct{
  int fd, userIdx, userCnt;
  struct file *f;
  char usernames[MAX_USER][MAX_LEN + 1];
  char passwords[MAX_USER][MAX_LEN + 1];
} ltable;
```

`void loadUsers()`

passwd 파일을 읽어온 뒤 유저의 이름과 비밀번호를 ltable에 업데이트합니다. passwd 파일은 첫 번째 유저인 root부터 차례대로 username1-password1/username2-password2/... 와 같이 유저 정보가 저장되어 있습니다.

`int flushUsers()`

ltable에 저장된 유저 정보 목록을 passwd 파일에 업데이트합니다. loadUsers와 마찬가지로 ltable에 저장된 file 구조체를 이용하여 fileread/write를 사용해 I/O를 진행합니다. 기존 passwd의 정보를 모두 삭제하고 처음부터 다시 작성하는 방식으로 이루어집니다.

`int addUser(char *username, char *password)`

입력된 유저가 존재하지 않고 새로운 유저를 추가할 공간이 남은 경우에만 ltable에 새로운 유저를 추가합니다. 이후 flushUsers를 호출하여 새로운 유저 정보를 포함한 전체 정보를 passwd 파일에 저장합니다. root 유저를 제외한 나머지 유저들이 추가되면 mkdir을 이용하여 유저 이름과 동일한 이름을 갖는 디렉토리를 생성합니다. 만약 디렉토리가 성공적으로 생성되었다면, 해당 디렉토리의 소유자를 새로 추가된 유저로 설정합니다.

`int deleteUser(char *username)`

입력된 유저 이름이 root가 아니며 현재 존재하는 유저일 경우에 ltable에서 해당 유저를 제거하고 addUser와 마찬가지로 flushUsers를 호출하여 유저 정보를 passwd 파일에 업데이트합니다.

`void whoami(char *dst)`

입력된 주소 공간에 현재 로그인한 유저 이름을 복사합니다. 만약 아직 부팅이 진행되는 중이어서 아무도 로그인하지 않은 상태라면 “root”를 복사합니다.

`void lbegin(void)`

첫 프로세스인 init에서 호출되는 시스템 콜입니다. loadUsers를 호출하여 passwd 파일의 정보를 읽어옵니다. 만약 passwd 파일이 새로 생성되었거나 내용이 없는 경우, 첫 번째 유저인 root를 생성하기 위해 addUser를 호출하게 됩니다. 이후 현재 로그인한 유저를 나타내는 ltable의 userIdx를 -1로 설정하여 아무도 로그인되지 않은 상태로 설정합니다.

`int login(char *username, char *password)`
입력된 유저 이름과 비밀번호를 바탕으로 로그인을 시도합니다. ltable에 저장되어 있는 유저 정보와 입력된 정보를 비교하여 유저 이름과 비밀번호가 일치하는 경우에 해당 유저의 인덱스를 ltable의 userIdx에 복사하여 현재 로그인한 유저를 표시합니다. 로그인에 성공하는 경우 0, 실패하는 경우 -1을 반환합니다.

- xv6가 부팅하고 난 후엔 init 프로세스가 가장 처음으로 실행됩니다. init 프로세스의 main에선 passwd 파일을 읽어와 계정 정보를 불러오는 lbegin 시스템 콜이 호출된 다음 무한 루프를 돌면서 기존에 sh을 실행하던 것처럼 login 유저 프로그램을 실행하게 됩니다.

```
lbegin();

for(;;){
  printf(1, "init: starting login\n");
  pid = fork();
  if(pid < 0){
    printf(1, "init: 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");
}
```

- login 유저 프로그램은 기존 init 프로세스와 매우 유사한 방식으로 동작합니다. 무한 루프를 돌면서 유저 이름과 비밀번호를 입력받고, login 시스템 콜을 호출하여 로그인을 시도합니다. 로그인에 성공하면 sh을 실행하고 종료될 때 까지 기다립니다. 로그인에 실패하면 다음 루프로 이동해 다시 유저 이름과 비밀번호를 입력받아 로그인을 시도하는 과정을 계속해서 반복하게 됩니다.

- 쉘이 실행되면 기존과 유사하게 작동하다가 내장 명령인 logout을 입력하면 쉘이 종료될 수 있도록 했습니다. logout을 입력한 이후에는 다시 로그인 프로그램으로 돌아가 유저 이름과 비밀번호를 입력하고 로그인을 시도하는 과정을 반복하게 됩니다.

- 그리고 쉘의 내장 명령어로 whoami를 추가하였습니다. 입력할 경우 whoami 시스템 콜을 호출하여, 현재 어떤 유저가 로그인한 상태인지 출력해줍니다. 이 시스템 콜을 활용하여 쉘의 명령 입력시 출력 문구를 보기 좋게 만들었습니다. 리눅스 쉘처럼 현재 로그인한 유저의 이름과 현재 위치한 디렉토리를 출력하여 사용하기 편리하게 만들었습니다.


**File mode implementation**

- 기존에 존재하지 않던 유저 개념을 추가한 다음에는, 이를 바탕으로 각 파일과 디렉토리의 소유자와 권한을 지정할 수 있는 기능을 구현했습니다.

- 파일의 내용을 변경해야 하기 때문에, 파일의 메타데이터를 저장하고 관리하는 dinode, inode, stat 구조체의 내용을 아래와 같이 변경하였습니다. 권한을 지정하는 uint 형식의 값 perm과 파일의 소유자를 저장하는 char 배열 owner를 새로 추가하였습니다.

- 기존 diniode 구조체의 크기는 512의 약수인 64바이트로 이루어져 있습니다. 새로운 멤버를 추가한 이후에도 dinode 구조체의 크기가 512의 약수가 되도록 하기 위해 NDIRECT 값을 7로 조정하여 구조체의 크기가 64바이트를 유지하도록 했습니다.

- inode와 stat은 diniode에 속해 있는 파일의 메타데이터를 그대로 복사해서 사용하므로, 같은 방식으로 perm과 onwer를 추가해 주었습니다.

```
#define NDIRECT 7
#define NINDIRECT (BSIZE / sizeof(uint))
#define MAXFILE (NDIRECT + NINDIRECT)
#define MAX_LEN 15

// On-disk inode structure
// sizeof dinode => 64bytes
struct dinode {
  short type;           // File type
  short major;          // Major device number (T_DEV only)
  short minor;          // Minor device number (T_DEV only)
  short nlink;          // Number of links to inode in file system
  uint size;            // Size of file (bytes)
  uint perm;            // Permission of this file
  char owner[MAX_LEN + 1]; // username of the owner of this file
  uint addrs[NDIRECT+1];   // Data block addresses
};
```

```
struct stat {
  short type;  // Type of file
  int dev;     // File system's disk device
  uint ino;    // Inode number
  short nlink; // Number of links to file
  uint size;   // Size of file in bytes
  uint perm;   // Permission of file
  char owner[MAX_LEN + 1]; // Username of owner
};
```

- 각각의 파일의 권한을 표시하기 위해 fs.h에 과제 명세와 동일하게 소유자/그 외 유저에 대한 권한을 정의했습니다. 각 파일과 디렉토리의 권한은 이 값들이 bit or 연산을 통해 합쳐져 만들어집니다.

```
#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
```

- 모든 파일과 디렉토리들은 생성되는 시점에 권한과 소유자가 정해지고 dinode에 저장되게 됩니다. 만약 첫 부팅 전 fs.img를 새로 생성하는 경우에 생성되는 파일과 디렉토리일 경우, fs.img를 생성하는 과정에서 권한과 소유자를 지정해주어야 합니다. fs.img를 생성하는 코드인 mkfs.c에서 새로운 dinode를 생성하는 함수인 ialloc을 수정하여 기본으로 rw xr-x 권한을 부여하도록 했습니다.

```
uint
ialloc(ushort type)
{
  uint inum = freeinode++;
  struct dinode din;

  bzero(&din, sizeof(din));
  din.type = xshort(type);
  din.nlink = xshort(1);
  din.size = xint(0);
  din.perm = xint(MODE_RUSR | MODE_WUSR | MODE_XUSR | MODE_ROTH | MODE_XOTH);
  strcpy(din.owner, "root");
  winode(inum, &din);
  return inum;
}
```

- mkfs.c가 실행되어 fs.img가 생성된 이후에 추가로 만들어진 파일이나 디렉토리들은 모두 create 함수를 통하여 만들어집니다. create 함수에 새로 생성되는 파일이나 디렉토리의 권한을 타입에 맞게 부여하고 whoami 함수를 이용하여 현재 로그인한 사용자를 소유자로 지정하도록 수정하였습니다.

```
if((ip = ialloc(dp->dev, type)) == 0)
  panic("create: ialloc");

ilock(ip);
ip->major = major;
ip->minor = minor;
ip->nlink = 1;
if(ip->type == T_DIR)
  ip->perm = MODE_RUSR | MODE_WUSR | MODE_XUSR | MODE_ROTH | MODE_XOTH;
else if(ip->type == T_FILE)
  ip->perm = MODE_RUSR | MODE_WUSR | MODE_ROTH;
if(ltable.userCnt == 0){
  strncpy((char*)ip->owner, (const char*)"root", 4);
  ip->owner[4] = 0;
}
else{
  whoami((char*)ip->owner);
}
iupdate(ip);
```

- 각 유저가 특정 파일 또는 디렉토리를 사용하려고 할 때는 해당 파일이나 디렉토리에 부여된 권한이 현재 로그인한 사용자에게 해당 작업을 허용하는지 확인해야 합니다.
- 이를 위해 특정 파일 또는 디렉토리에서 할 수 있는 작업을 알려주는 함수 `checkmod`를 정의하였습니다. `checkmod`는 특정 `inode`의 정보를 `whoami`로 현재 로그인한 유저 정보를 불러온 다음 현재 유저가 소유자(또는 `root`)인지 아닌지 판단합니다. 그 다음 현재 유저에게 `read/write/execute` 권한이 있는지 판단하여 `OR` 연산을 통해 만들어진 결과를 반환하게 됩니다.

```
int checkmod(struct stat st){

  int perm = st.perm, result = 0;
  const char *owner = st.owner;
  char username[MAX_LEN + 1];

  if(st.type == T_DEV)
    return R_OK | W_OK | X_OK;

  whoami(username);
  if(equals(owner, username) || equals("root", username)){
    // current user is either owner or root
    if(perm & MODE_RUSR)
      result |= R_OK;
    if(perm & MODE_WUSR)
      result |= W_OK;
    if(perm & MODE_XUSR)
      result |= X_OK;
  }
  else{ // owner != current user
    if(perm & MODE_ROTH)
      result |= R_OK;
    if(perm & MODE_WOTH)
      result |= W_OK;
    if(perm & MODE_XOTH)
      result |= X_OK;
  }
  return result;
}
```

- checkmod 가 반환하는 값은 fs.h에 MODE_XXXX와 유사하게 정의한 R_OK, W_OK, X_OK 값을 bit or 연산을 이용해서 만들어집니다.

```
#define R_OK 4  // Read permission checked
#define W_OK 2  // Write permission checked
#define X_OK 1  // Execute permission checked
```

- 파일을 다루는 각 함수에서 권한을 확인하기 위해선 다음과 같은 과정이 이루어져야 합니다.
1. 해당 파일의 inode를 찾은 후 stati를 호출하여 inode의 정보를 불러옵니다.
2. inode의 stat로 checkmod를 호출하여 가능한 권한 정보를 불러옵니다.
3. checkmod의 결과값과 사용하려는 모드 비트와 or 연산하여 가능한지 확인합니다.
4. 값이 확인되었다면 계속 진행하고, 권한이 허용되지 않았다면 작업을 중단합니다.

- 과제 명세에 나와있는 파일 관리 시스템의 각 부분에 이러한 단계를 거쳐 권한을 판별하는 코드를 삽입하여 파일 시스템에 소유자와 권한의 개념을 도입했습니다.
- 먼저 namex 함수에서 경로를 한 단계씩 따라갈 때마다 각 디렉토리에 execute 권한이 있는지 확인하게 했습니다. 다만 부팅 직후 init과 login이 실행되는 도중에는 로그인된 유저가 없는 상태이기 때문에, pid가 3 이상인 경우에만 권한을 확인합니다.

```
if(myproc()->pid > 2){
  stati(ip, &st);

  if(!(checkmod(st) & X_OK)){
    iunlockput(ip);
    return 0;
  }
}
```

- exec 함수 역시 마찬가지로 실행하려는 파일에 execute 권한이 있는지 확인합니다. namex와 같은 이유로 인해 init과 login 이후의 pid가 3 이상인 경우에만 권한을 확인합니다.

```
if(curproc->pid > 2){
  stati(ip, &st);

  if(!(checkmod(st) & X_OK)){
    goto bad;
  }
}
```

- create 함수에서는 두 가지 부분에서 권한을 확인해야 합니다. 먼저 생성하려는 파일이 이미 존재하는 경우, 해당 파일에 write 권한이 있는지 확인합니다. dirlookup의 결과가 0이 아닐 경우에 반환된 inode에 write 권한이 있는지 확인하는 코드를 삽입했습니다.

```
if((ip = dirlookup(dp, name, 0)) != 0){ // File already exists!
  iunlockput(dp);
  ilock(ip);

  if(type == T_FILE && ip->type == T_FILE){
    if(ltable.userCnt > 0){
      stati(ip, &st);

      if(!(checkmod(st) & W_OK)){
        iunlock(ip);
        return 0;
      }
```

- 만약 create 함수에서 생성하려는 파일이 존재하지 않는 경우, 새로운 파일을 생성하게 되는데, 그 파일이 위치할 디렉토리에 write 권한이 있는지 확인합니다. 생성하려는 위치 디렉토리의 inode를 checkmod를 사용하여 검사합니다.

```
stati(dp, &st);
if(!(checkmod(st) & W_OK)){
  iunlock(dp);
  return 0;
}
```

- sys_open 함수에서도 두 가지 권한을 확인해야 합니다. 열기 모드가 O_RDONLY 또는 O_RDWR 일 때 read 권한이 있는지 확인하고, 열기 모드가 O_WRONLY 또는 O_RDWR 일 때 write 권한이 있는지 확인합니다.

```
stati(ip, &st);
check = checkmod(st);

if(((omode == O_RDONLY) | (omode & O_RDWR))){
  if(!(check & R_OK)){
    iunlock(ip);
    end_op();
    return -1;
  }
}
if(((omode & O_WRONLY) | (omode & O_RDWR))){
  if(!(check & W_OK)){
    iunlock(ip);
    end_op();
    return -1;
  }
}
```

- sys_chdir 함수의 경우에는 목적지에 execute 권한이 있는지 확인합니다.

```
ilock(ip);
stati(ip, &st);
if(ip->type != T_DIR){
  cprintf("chdir %s: not a directory.\n", path);
  printstat(st);
  iunlockput(ip);
  end_op();
  return -1;
}
if(!(checkmod(st) & X_OK)){
  iunlock(ip);
  end_op();
  return -1;
}
```

- 마지막으로 sys_unlink 함수의 경우에는 삭제하려는 파일이 있는 디렉토리를 먼저 찾은 다음, 해당 디렉토리에 write 권한이 있는지 확인합니다.

```
begin_op();
if((dp = nameiparent(path, name)) == 0){
  end_op();
  return -1;
}

ilock(dp);

stati(dp, &st);
if(!(checkmod(st) & W_OK)){
  iunlock(dp);
  end_op();
  return -1;
}
```

**Change mode implementation**

- 각 파일과 디렉토리의 소유자(또는 root)는 해당 파일 또는 디렉토리의 권한을 변경할 수 있습니다. 이는 시스템 콜인 chmod를 사용하여 이루어질 수 있습니다.
- chmod 함수는 권한을 변경할 파일의 경로와 변경하고자 하는 모드를 입력받습니다. 오직 root 또는 파일의 소유자만 권한을 변경할 수 있기 때문에, 현재 유저가 누구인지 확인하고, 권한이 있다면 inode의 값을 수정하고 iupdate를 호출하여 디스크에 변경 사항을 반영합니다.

```
int chmod(char *path, int mode)
```

```
if(!(equals("root", curUser) || equals(st.owner, curUser))){
  iunlock(ip);
  end_op();
  return -1;
}

ip->perm = mode;
iupdate(ip);
iunlock(ip);
end_op();
return 0;
```

**Modification of ls**

- 유저 프로그램인 ls는 현재 디렉토리의 디렉토리 엔트리를 확인하여 해당 파일 또는 디렉토리들의 정보들을 출력하는 함수입니다.
- 파일 구조에 새로운 정보인 권한과 소유자가 추가되었기 때문에, ls 프로그램이 해당 정보들을 수정할 수 있도록 합니다.
- ls 프로그램은 stati 함수를 내부적으로 호출하여 정보를 복사해오기 때문에, stati 함수가 새로 추가된 perm과 owner의 값을 복사해올 수 있도록 수정했습니다. 그리고 추가된 정보들을 출력할 수 있도록 ls 함수를 수정했습니다.

```
void
stati(struct inode *ip, struct stat *st)
{
  st->dev = ip->dev;
  st->ino = ip->inum;
  st->type = ip->type;
  st->nlink = ip->nlink;
  st->size = ip->size;
  st->perm = ip->perm;
  strncpy((char*)st->owner, (char*)ip->owner, MAX_LEN + 1);
}
```

```
root) /$ ls
.               drwxr-x 1 1 512B        root
..              drwxr-x 1 1 512B        root
README          -rwxr-x 2 2 2.3KB       root
cat             -rwxr-x 2 3 16.0KB      root
echo            -rwxr-x 2 4 15.8KB      root
forktest        -rwxr-x 2 5 9.3KB       root
grep            -rwxr-x 2 6 18.7KB      root
init            -rwxr-x 2 7 15.0KB      root
kill            -rwxr-x 2 8 15.1KB      root
ln              -rwxr-x 2 9 15.6KB      root
ls              -rwxr-x 2 10 21.9KB     root
mkdir           -rwxr-x 2 11 15.1KB     root
rm              -rwxr-x 2 12 15.8KB     root
sh              -rwxr-x 2 13 30.4KB     root
stressfs        -rwxr-x 2 14 16.8KB     root
usertests       -rwxr-x 2 15 66.8KB     root
wc              -rwxr-x 2 16 17.2KB     root
zombie          -rwxr-x 2 17 15.8KB     root
my_userapp      -rwxr-x 2 18 16.9KB     root
project01       -rwxr-x 2 19 15.8KB     root
login           -rwxr-x 2 20 16.0KB     root
useradd         -rwxr-x 2 21 15.1KB     root
userdelete      -rwxr-x 2 22 15.2KB     root
chmod           -rwxr-x 2 23 16.0KB     root
modtest         -rwxr-x 2 24 18.7KB     root
console         *DEVICE 3 25 0B         root
passwd          -rw-r-- 2 26 20B        root
user            drw-r-- 1 27 32B        user
root) /$
```

## 3. Result

**User account management result**

- 빌드 후 첫 부팅 시에는 root 계정으로만 로그인이 가능합니다.

```
init: starting login
Enter username: root
Enter password: ****
login successful
```

- 첫 로그인 뒤 유저를 추가, 삭제가 가능하고, 변경된 유저 정보들은 passwd 파일에 저장된 것을 볼 수 있습니다.

```
root) /$ cat passwd
root-0000/root) /$
root) /$ useradd
[Add user]
Username: user2
Password: 1234
Add user successful!
root) /$ cat passwd
root-0000/user2-1234/root) /$
root) /$ userdelete
[Delete user]
Username: user2
Delete user successful!
root) /$ cat passwd
root-0000/root) /$
root) /$
```

- 추가된 유저로 로그인하여 해당 유저의 권한으로 OS를 사용하거나 logout 명령어를 활용해 현재 쉘을 종료시키고 새로운 유저로 로그인할 수 있습니다.

```
root) /$ logout
Enter username: user
Enter password: ****
login successful
user) /$ cd user
user) /user$ logout
Enter username: root
Enter password: ****
login successful
root) /$ userdelete
[Delete user]
Username: user
Delete user successful!
```

**File mode result**

- root 유저는 chmod 시스템 콜을 사용해 다른 모든 유저의 파일과 디렉토리의 권한을 변경할 수 있습니다.
- 아래 실행 결과를 보면 user의 소유인 디렉토리 user의 권한을 root가 변경할 수 있다는 것을 알 수 있습니다.

```
Enter username: root
Enter password: ****
login successful
root) /$ chmod 64 user
chmod successful
root) /$ ls
.               drwxr-x 1 1 512B          root
..              drwxr-x 1 1 512B          root
README          -rwxr-x 2 2 2.3KB         root
cat             -rwxr-x 2 3 16.0KB        root
echo            -rwxr-x 2 4 15.8KB        root
forktest        -rwxr-x 2 5 9.3KB         root
grep            -rwxr-x 2 6 18.7KB        root
init            -rwxr-x 2 7 15.0KB        root
kill            -rwxr-x 2 8 15.1KB        root
ln              -rwxr-x 2 9 15.6KB        root
ls              -rwxr-x 2 10 21.9KB       root
mkdir           -rwxr-x 2 11 15.1KB       root
rm              -rwxr-x 2 12 15.8KB       root
sh              -rwxr-x 2 13 30.4KB       root
stressfs        -rwxr-x 2 14 16.8KB       root
usertests       -rwxr-x 2 15 66.8KB       root
wc              -rwxr-x 2 16 17.2KB       root
zombie          -rwxr-x 2 17 15.8KB       root
my_userapp      -rwxr-x 2 18 16.9KB       root
project01       -rwxr-x 2 19 15.8KB       root
login           -rwxr-x 2 20 16.0KB       root
useradd         -rwxr-x 2 21 15.1KB       root
userdelete      -rwxr-x 2 22 15.2KB       root
chmod           -rwxr-x 2 23 16.0KB       root
modtest         -rwxr-x 2 24 18.7KB       root
console         *DEVICE 3 25 0B           root
passwd          -rw-r-- 2 26 20B          root
user            drw-r-- 1 27 32B          user
root) /$
```

- 아래 예시처럼 root가 아닌 user로 로그인한 뒤 root가 소유한 passwd의 권한을 바꾸려고 할 경우 실패하는 결과를 볼 수 있습니다.

```
Enter username: user
Enter password: ****
login successful
user) /$ chmod 77 passwd
[stat] owner: root / mode: rw-r-- / type: T_FILE
chmod failed!
user) /$ chmod 77 user
chmod successful
user) /$ ls
.               drwxr-x 1 1 512B          root
..              drwxr-x 1 1 512B          root
README          -rwxr-x 2 2 2.3KB         root
cat             -rwxr-x 2 3 16.0KB        root
echo            -rwxr-x 2 4 15.8KB        root
forktest        -rwxr-x 2 5 9.3KB         root
grep            -rwxr-x 2 6 18.7KB        root
init            -rwxr-x 2 7 15.0KB        root
kill            -rwxr-x 2 8 15.1KB        root
ln              -rwxr-x 2 9 15.6KB        root
ls              -rwxr-x 2 10 21.9KB       root
mkdir           -rwxr-x 2 11 15.1KB       root
rm              -rwxr-x 2 12 15.8KB       root
sh              -rwxr-x 2 13 30.4KB       root
stressfs        -rwxr-x 2 14 16.8KB       root
usertests       -rwxr-x 2 15 66.8KB       root
wc              -rwxr-x 2 16 17.2KB       root
zombie          -rwxr-x 2 17 15.8KB       root
my_userapp      -rwxr-x 2 18 16.9KB       root
project01       -rwxr-x 2 19 15.8KB       root
login           -rwxr-x 2 20 16.0KB       root
useradd         -rwxr-x 2 21 15.1KB       root
userdelete      -rwxr-x 2 22 15.2KB       root
chmod           -rwxr-x 2 23 16.0KB       root
modtest         -rwxr-x 2 24 18.7KB       root
console         *DEVICE 3 25 0B           root
passwd          -rw-r-- 2 26 20B          root
user            drwxrwx 1 27 32B          user
user) /$
```

- 변경된 모드가 제대로 적용되는지 확인하기 위하여 권한이 적용된 파일들을 이용한 작업을 진행한 결과들입니다.

- root 유저가 소유하고 있는 passwd 파일의 권한을 00(read, write, execute 모두 불가능)으로 지정하고 cat을 이용하여 출력을 시도하면, 아무리 파일의 소유자라고 하더라도 read 권한이 없기 때문에 파일의 내용을 볼 수 없습니다.
- 다시 passwd의 권한을 read가 가능하도록 변경하면, cat을 이용해 파일의 내용을 정상적으로 출력할 수 있게 됩니다.

```
wc              -rwxr-x 2 16 17.0KB      root
zombie          -rwxr-x 2 17 15.7KB      root
my_userapp      -rwxr-x 2 18 16.8KB      root
project01       -rwxr-x 2 19 15.7KB      root
login           -rwxr-x 2 20 16.9KB      root
useradd         -rwxr-x 2 21 15.0KB      root
userdelete      -rwxr-x 2 22 15.1KB      root
chmod           -rwxr-x 2 23 15.2KB      root
modtest         -rwxr-x 2 24 18.6KB      root
console         *DEVICE 3 25 0B          root
passwd          -rw-r-- 2 26 20B         root
user            drwxr-x 1 27 32B         user
root) /$ chmod 00 passwd
chmod successful
root) /$ cat passwd
cat: cannot open passwd
root) /$ chmod 77 passwd
chmod successful
root) /$ cat passwd
root-0000/user-1234/root) /$
root) /$
```

- root가 아닌 user로 로그인하고 자신이 소유하고 있는 user 디렉토리의 모드를 execute가 불가능하게 변경한 다음 해당 디렉토리로 이동하려고 시도한 결과입니다.
- 디렉토리로 이동할 때는 해당 디렉토리에 대한 execute 권한이 필요하기 때문에 cd 명령은 실패하게 됩니다.
- user 디렉토리의 모드를 77로 변경하여 모든 작업(read, write, execute)이 허용된 후에는 그 디렉토리로 이동할 수 있고, write 권한이 있어 mkdir로 새로운 디렉토리를 생성할 수도 있습니다.

```
useradd         -rwxr-x 2 21 15.0KB      root
userdelete      -rwxr-x 2 22 15.1KB      root
chmod           -rwxr-x 2 23 15.2KB      root
modtest         -rwxr-x 2 24 18.6KB      root
console         *DEVICE 3 25 0B          root
passwd          -rwxrwx 2 26 20B         root
user            drwxr-x 1 27 32B         user
user) /$ chmod 66 user
chmod successful
user) /$ cd user
cannot cd user
user) /$ chmod 77 user
chmod successful
user) /$ cd user
user) /user$ mkdir d1
exec mkdir failed
user) /user$ /mkdir d1
user) /user$ cd d1
user) /user/d1$
```

# 4. Troubleshooting

- 기존의 유저와 소유 및 권한 개념이 없는 OS에서 파일시스템을 수정하여 기능들을 추가하는 과정에서 다음과 같은 문제 상황과 해결 방안을 발견했습니다.

**Handling files(open, create, write) in kernel mode**

- 기존 xv6는 유저 레벨에서 파일을 문자열 경로를 통해 접근해서 사용하기 위한 추상화가 구현되어 있었는데, 커널 모드에서 해당 기능들은 시스템 콜로 구현되어 있어 사용하기 어려운 상황이 있었습니다.
- 유저 모드에서 사용하는 것과 유사하게 간편하게 사용하면서도 커널 모드에서 파일을 열고 닫을 수 있게 하기 위해, sys_open과 sys_mkdir과 같은 함수를 실제 작동하는 코드와 유저 프로그램으로부터 argument를 받아오는 부분을 분리하였습니다.
- open, mkdir 등을 분리하고 sys_open과 sys_mkdir은 단순 argument를 받아 넘겨주는 방식으로 변경하는 것을 통해 기존 시스템 콜의 기능도 유지하면서, 커널 모드에서도 유저 모드에서 사용하는 것과 매우 유사하게 파일을 관리할 수 있도록 했습니다.
- 이렇게 새로 만든 API를 통해 passwd 파일을 손쉽게 관리할 수 있었습니다.

**Creating a new user’s directory**

- addUser를 사용하여 새로운 유저를 성공적으로 실행한 경우, 해당 유저의 이름과 동일한 이름의 디렉토리를 생성해야 합니다. 또한 해당 디렉토리의 소유자는 새로 생성된 유저로 정해야 했습니다.
- 하지만 오직 root만이 addUser를 정상적으로 사용할 수 있었기 때문에, root 유저가 로그인한 상태에서 addUser를 실행했을 때 생성된 디렉토리의 이름은 새로운 유저의 이름과 동일하지만 소유자가 root로 지정되는 문제가 있었습니다.
- 이 문제를 해결하기 위해, addUser의 실행 과정 중 새로운 유저가 추가되고 유저 목록에 등록되면, mkdir을 실행하기 직전 현재 로그인한 유저를 알려주는 ltable.userIdx의 값을 저장해두고 새로 만든 유저의 index로 지정하여 잠깐 동안 새로운 유저의 권한으로 로그인한 상태처럼 만들었습니다.
- 이 상태에서 mkdir을 호출하면 새로 만들어진 유저의 소유인 디렉토리가 만들어지게 되고, 작업이 끝나면 원래 로그인 되어있던 유저의 index를 다시 ltable.userIdx에 저장하여 원상복귀시킵니다.

**Misuse of locking system during file operations**

- file system을 다루는 과정에서 iget-iput 또는 ilock-iunlock과 같이 항상 짝으로 같이 실행되어야 하는 함수들, 특히 locking system을 사용하는 과정에서 실수가 잦았습니다.
- 알 수 없는 이유로 OS가 동작 도중 정지하는 경우가 있었는데, Ctrl+P를 눌러 procdump를 호출해 보면 실행 중이던 프로세스가 sleeplock을 aquire하지 못해 sleep 상태에 들어간 상태라는 것을 알 수 있었습니다.
- iget 또는 ilock을 직간접적으로 호출하는 함수들 중 조기 종료하는 경우에 이 함수들에 대응되는 iput 또는 iunlock 함수가 반드시 실행되도록 점검한 뒤에는 이러한 프리징 문제가 해결되었습니다.

한 학기동안 고생 많으셨습니다.
감사합니다. -END-