8) Try following code. Make /aa/bb and type some text with length longer than 50 bytes. Explain the result.

x=open("/aa/bb", O\_RDONLY, 00777);

y=read(x, buf, 10);

buf[y]=0;

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

lseek(x, 20, SEEK\_SET);

y=read(x, buf, 10);

buf[y]=0;

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

x1=dup(x);

y=read(x1, buf, 10);

buf[y]=0;

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

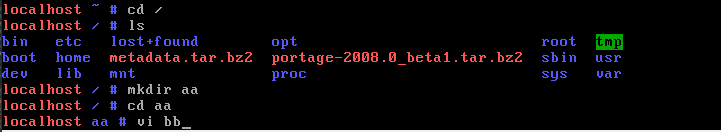
link("/aa/bb", "/aa/newbb");

x2=open("/aa/newbb", O\_RDONLY, 00777);

y=read(x2, buf, 10);

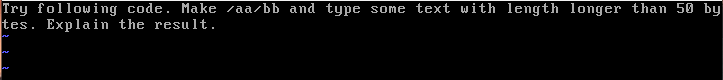
buf[y]=0;

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

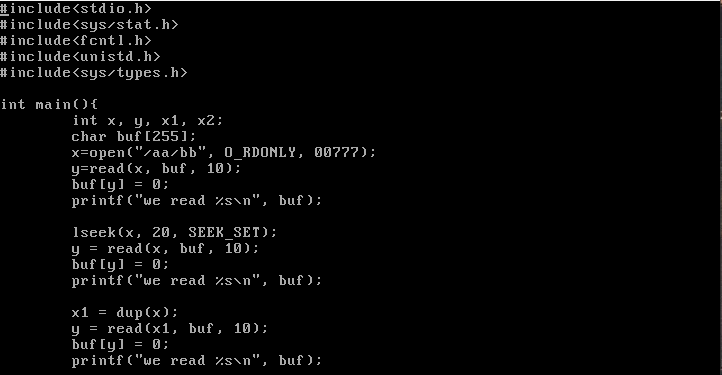


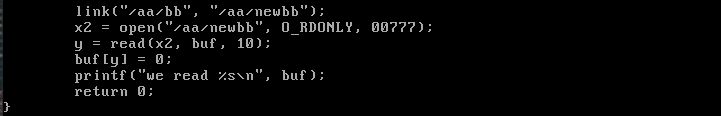
root 위치로 이동하여 파일을 확인해본 결과 aa가 존재하지 않았다.

aa디렉토리를 생성하고 그 안에 bb 파일을 생성한다.

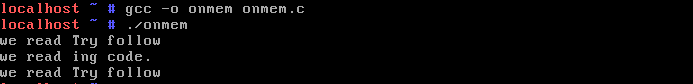


bb에 50byte가 넘는 문자열을 입력하기 위해 해당 문제를 그대로 입력하였다.





해당 문장을 입력하였다.





컴파일 후 실행결과 다음과 같이 출력됨을 알 수 있다.

open은 /aa/bb/의 inode를 찾아서 메모리에 캐시하고, 파일 테이블과 연결한다. F dentry는 /aa/bb의 inode이고, f pos는 0으로 초기화한다. fd table에 비어있는 fd를 할당하고 fd를 return 해준다. 결국 x에 /aa/bb의 fd가 저장되고, read로 10byte만큼 읽었고, 출력하면 10byte만큼 출력된다.

Lseek을 통해 f\_pos의 값을 지정한다. SEEK\_SET은 20으로 지정해주어서 파일의 처음부터 20byte 떨어진 부분부터 읽을 수 있다. 10byte를 읽었으므로 10byte만큼 출력한다.

Dup는 파일 테이블의 fd[x]를 그대로 복사한다. 결국 f\_pos도 같기 때문에 read하고, 출력하면 직전에 읽었던 것을 이어서 b and type이 된다.

링크는 inode를 가지는 파일을 생성하는데, f\_pos가 0이므로 newvbb를 읽으면 처음부터 10byte를 읽게된다.

9) Check the inode number of /aa/bb and /aa/newbb and confirm they are same.

# ls –i /aa/\*



/aa위치에 있는 모든 파일의 inode 번호를 출력하는 명령어를 입력하였다. /aa 디렉토리에는 /aa/bb와 직전에 link를 통해 생성된 /aa/newbb가 존재하였으며 이 둘의 inode 번호는 link로 생성되었으므로 일치하는 것을 알 수 있다.

10) Try fork() and confirm the parent and child can access the same file.

x=open("/aa/bb", ...);

y=fork();

if (y==0){

z=read(x, buf, 10);

buf[z]=0;

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

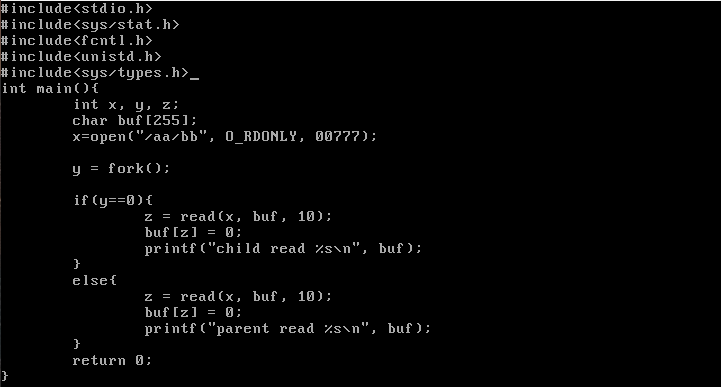
}else{

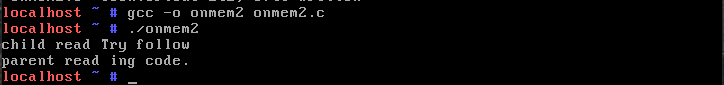
z=read(x, buf, 10);

buf[z]=0;

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

}

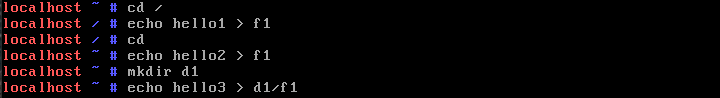




Fork를 하면 task\_struct를 복사한다. Task\_struct의 \*fs와 \*files가 함께 복사되기 때문에 child와 parent는 같은 fd 테이블과 파일 테이블을 가지게 된다. y==0일 때는 child를 의미하며 이 때 10byte를 읽게되고, Try follow를 읽게 된다.

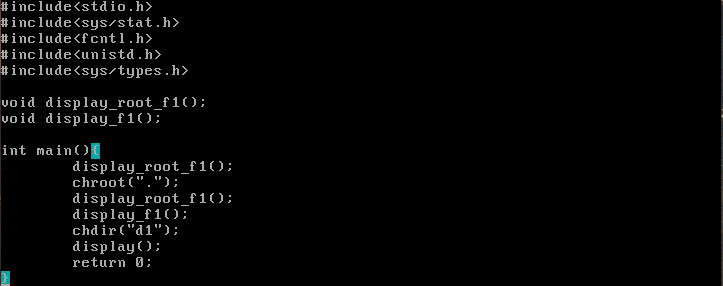
11) (Using "chroot" and "chdir") Do following and explain the result of "ex1".

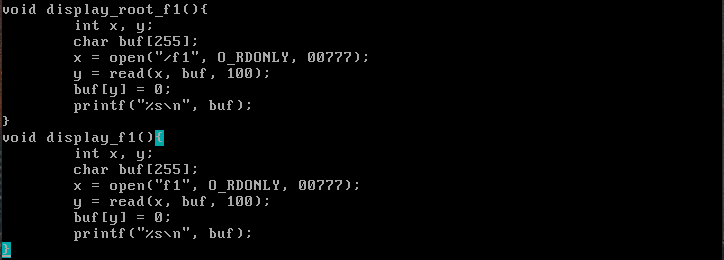
a. Make f1 in several places with different content (in "/", in "/root", and in "/root/d1") as follows.

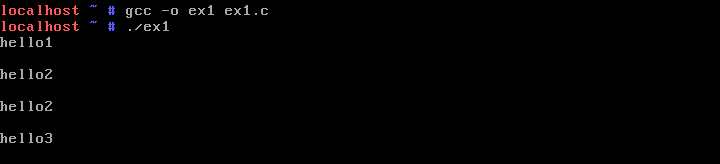


root directory에 f1을 생성하고, home directory에 f1을 생성, 마지막으로 home directory의 d1 디렉토리를 생성하여 f1을 생성한다.

b. Make ex1.c that will display "/f1" before and after "chroot", and "f1" before and after "chdir" as follows.







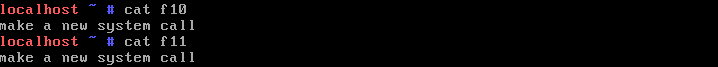
display\_root\_f1()을 통해 실제 /에 존재하는 f1을 출력하였고 그 결과 hello1이 출력되었다.

이후 chroot(“.”)을 통해 root를 .으로 변경해준다. 이는 현재 위치인 home directory를 root로 변경해준다. 이후 display\_root\_f1을 통해 home directory에 저장된 f1가 출력된다.

이후 현재 위치의 f1을 출력한다. 위와 같이 home directory에 저장된 f1이 출력된다.

chdir을 통해 현재 위치를 d1으로 이동하고 현재 위치의 f1을 출력한다.

12) Make a new system call, “show\_fpos()”, which will display the current process ID and the file position for fd=3 and fd=4 of the current process. Use this system call to examine file position as follows. (Use %lld to print the file position since f\_pos is long long integer)



x=open("f1", .............);

y=open("f2", .............);

show\_fpos(); // f\_pos right after opening two files

read(x, buf, 10);

read(y, buf, 20);

show\_fpos(); // f\_pos after reading some bytes



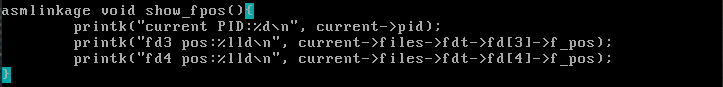
해당 위치로 이동하여



56번에 show\_fpos를 선언한다.

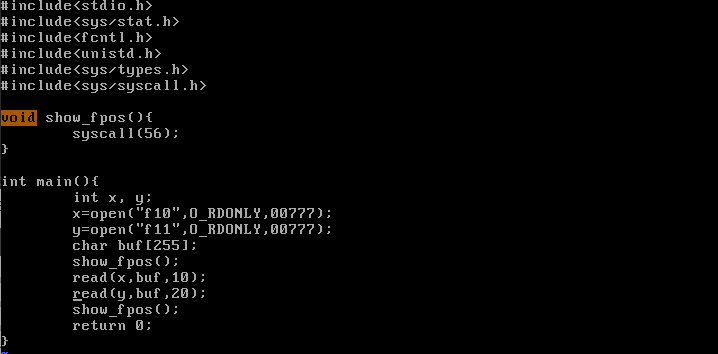


해당 위치로 이동하여



show\_fpos에 알맞은 함수를 정의한다.

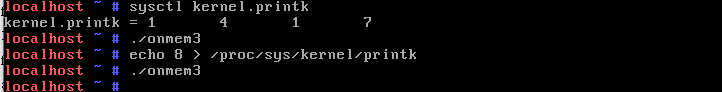
이후 recomplie & reboot를 진행한다.



코드를 작성하고 컴파일 하였다.

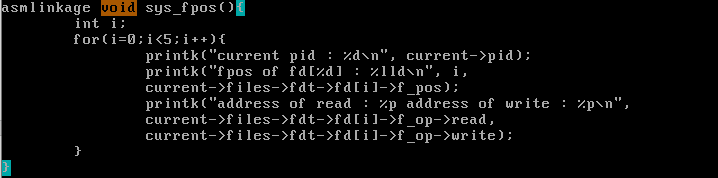


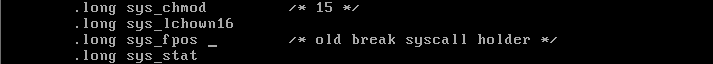
아무것도 출력되지 않은 모습이다.

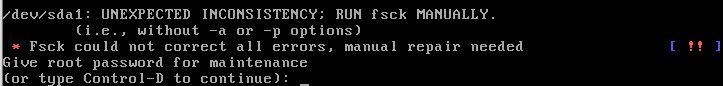


로그 레벨 문제라고 생각하여 수정해보았지만 역시 아무것도 출력되지 않았다.

13) Modify your show\_fpos() such that it also displays the address of f\_op->read and f\_op->w->e function for fd 0, fd 1, fd 2, fd 3, and fd 4, respectively. Find the corresponding function names in System.map. Why the system uses different functions for fd 0, 1, 2 and fd 3 or 4?







Recompile을 하고 reboot를 하려 하였으나 MyLiunx와 MyLinux3는 아예 열리지 않았고,MyLinux2는 해당 오류와 함께 /dev is broken이라는 오류가 함께 발생하였다.



Ctrl + d를 한 뒤 이후 실행을 해보아도 출력되지 않았다.

14) Use show\_fpos() to explain the result of the following code. File f1 has “ab” and File f2 has “q”. When you run the program, File f2 will have “ba”. Explain why f2 have “ba” after the execution.

int f1, f2, x; char buf[10];

f1=open(“./f1”, O\_RDONLY, 00777);

f2=open(“./f2”,O\_WRONLY, 00777);

printf(“f1 and f2 are %d %d\n”, f1, f2); // make sure they are 3 and 4

x=fork();

if (x==0){

show\_fpos();

read(f1,buf,1);

sleep(2);

show\_fpos();

write(f2, buf, 1);

}else{

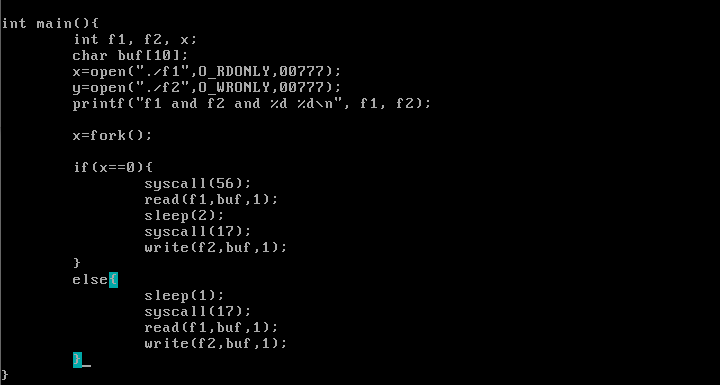
sleep(1);

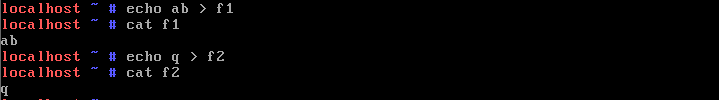
show\_fpos();

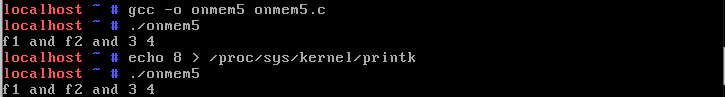
read(f1,buf,1);

write(f2,buf,1);

}







또 출력이 안되는 모습이다.

15) Find corresponding kernel code for each step below in open and read system calls:

x=open(fpath, .......);

1) find empty fd

2) search the inode for "fpath"

2-1) if "fpath" starts with "/", start from "fs->root" of the current process

2-2) otherwise, start from "fs->pwd"

2-3) visit each directory in "fpath" to find the inode of the "fpath"

2-4) while following mounted file path if it is a mounting point.

3) find empty file{} entry and fill-in relevant information.

4) chaining

5) return fd

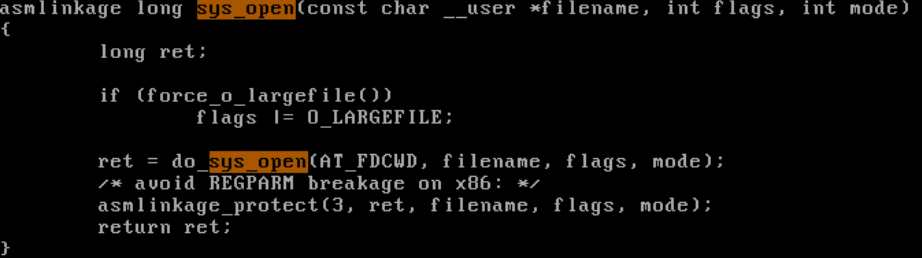
read(x, buf, n);

1) go to the inode for x

2) read n bytes starting from the current file position

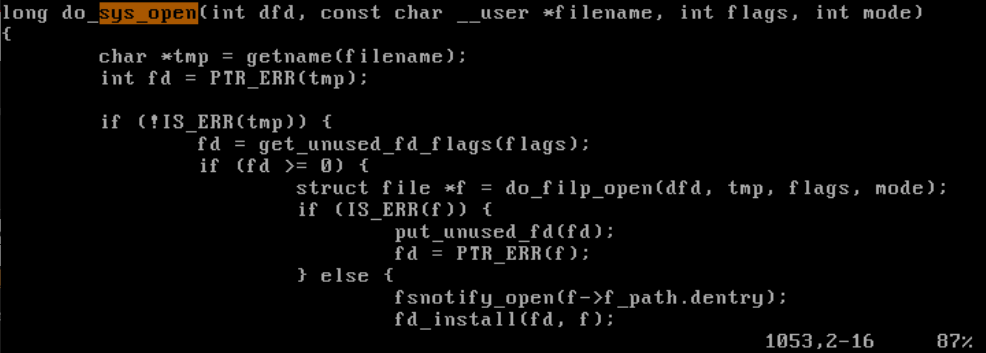
3) save the data in buf

4) increase the file position by n



Sys\_open 함수 안에는 do\_sys\_open을 호출 하는 것을 볼 수 있었다.

따라서 do\_sys\_open을 찾아가 보았다.



Sys\_open 함수는 fs/open.c에 있다.

여기서 fd=get\_unused\_fd\_flags(flags); 라는 명령이 empty fd를 나타내는 것 같다.



1) empty fd를 찾았다.

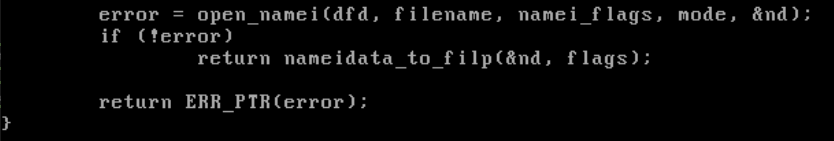
2) if (fd>=0) // fd가 0이상인 것부터 fpath에해당하는 inode를 찾는다



if(IS\_ERR(f) // fpath에해당하는 file pointer를 찾음



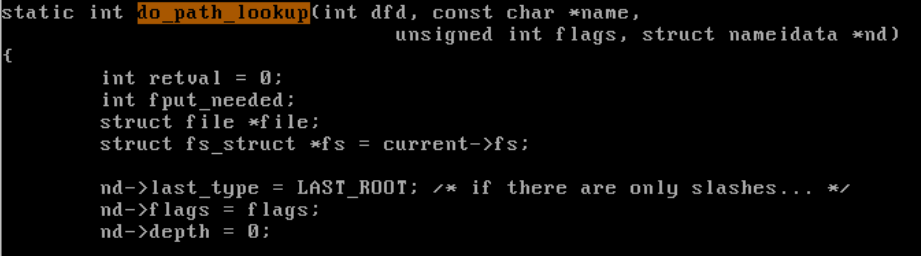
Do\_filp\_open 함수를 찾아가보니

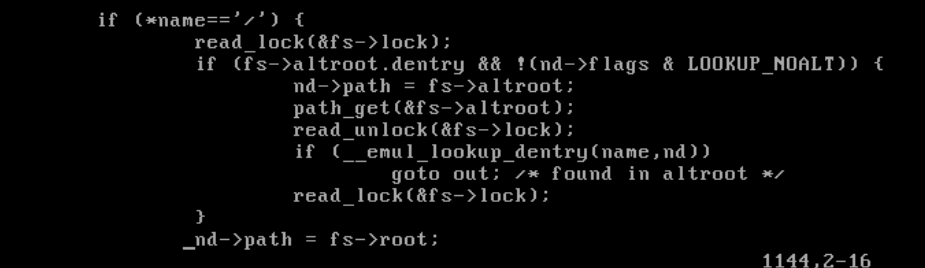


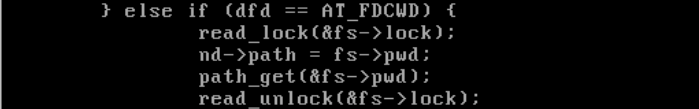
Nameidata\_to\_filp 부분에서 3),4)번을 하는 것이라고 예상했다.

open\_namei안에 path\_lookup\_open안에 \_\_path\_lookup\_intent\_open안에 do\_path\_lookup에서 fpath가 절대경로인지 아닌지 판단하는 부분을 찾았다.

Do\_path\_loockup 은 fs/namei.c에 존재하였다.



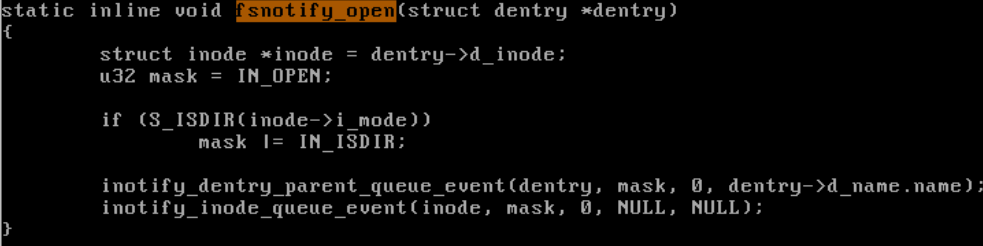


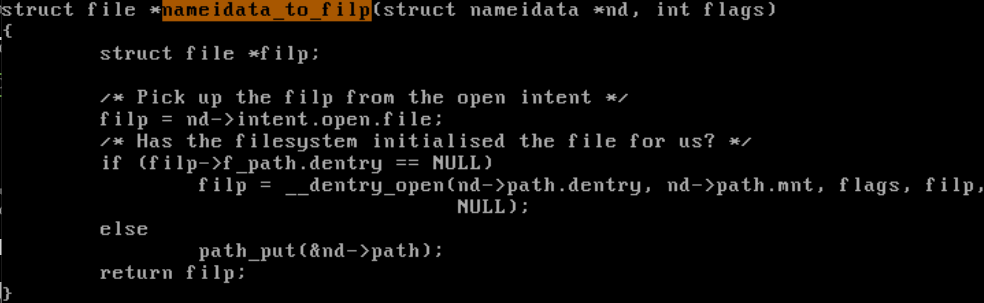


먼저 root를 찾고, root에 없으면 fs->pwd를 찾는 것을 알 수 있다.

3. else { // find empty file{}

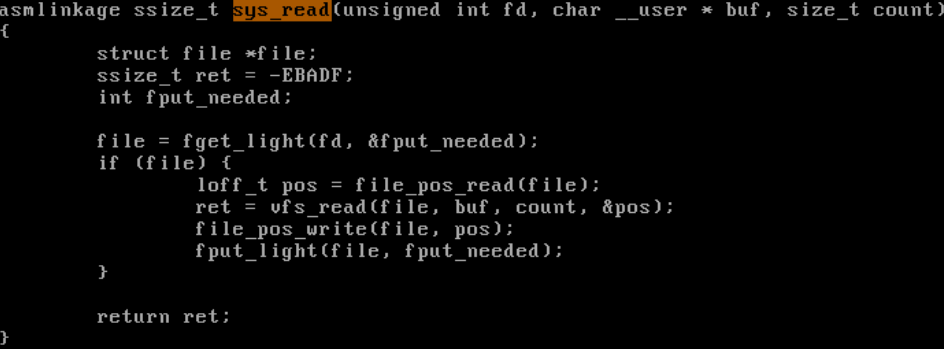
Fsnotify\_open 함수는 include/linux/fsnotify.h에 정의되어 있었다.





\_\_dentry\_open 을 통해 chaining 하면서 return fd를 하는 것을 알 수 있었다.

Read()

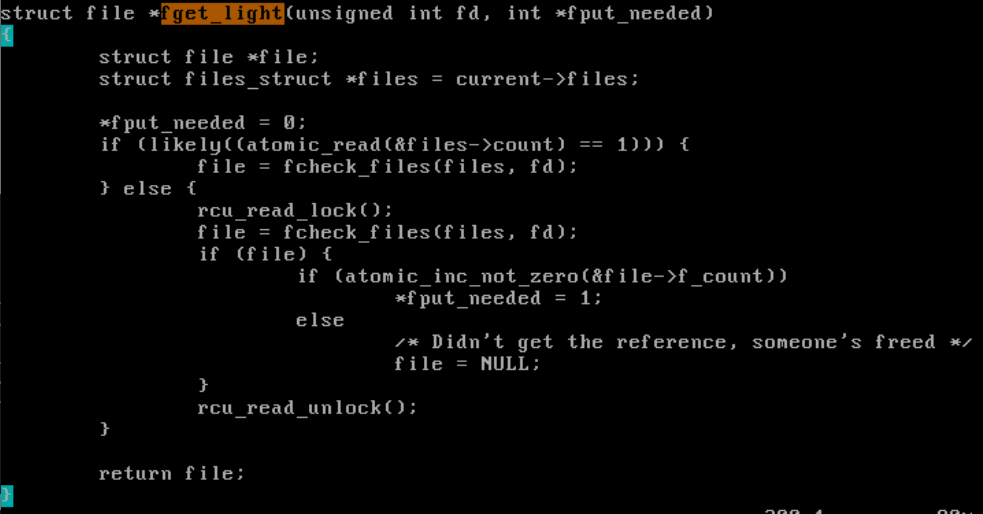


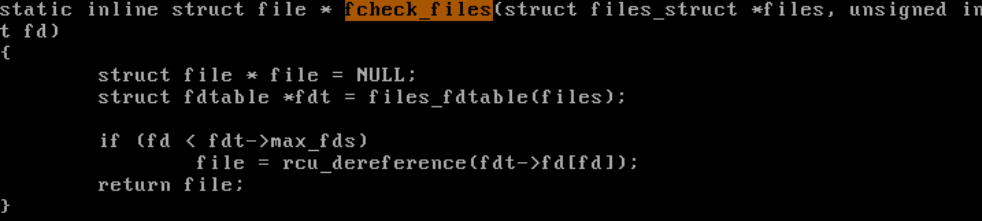
1) 은 Fget\_light을 통해 진행 되고,

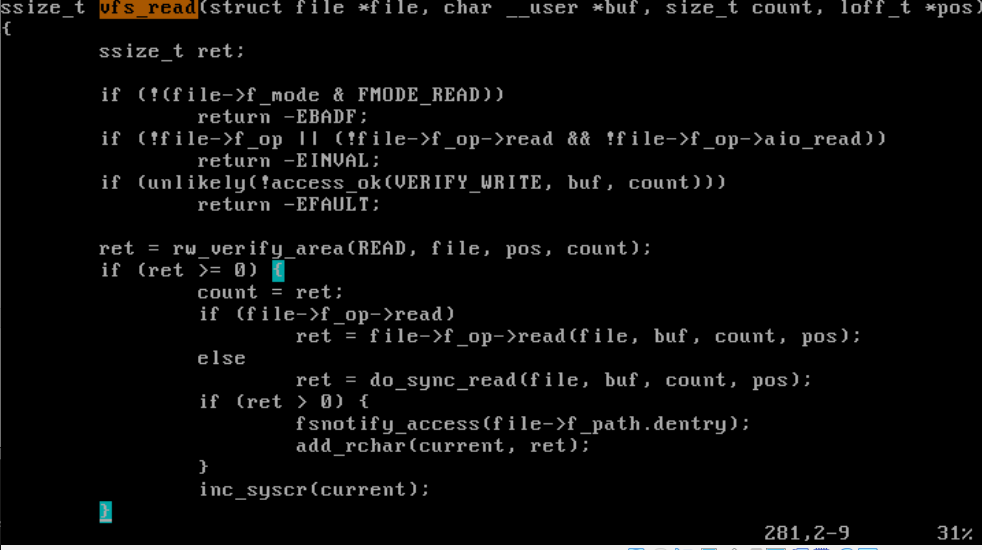
2) 는 vfs\_read() 을 통해 진행 될 것이다.

3) 은 file\_pos\_write()을 통해 진행되고,

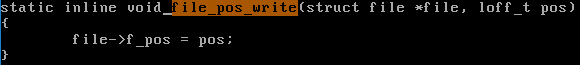
4) 는 fput\_light()를 통해 진행 될 것이다.



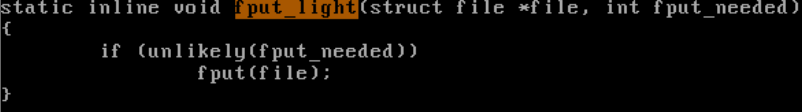




2)번, 3)번 은 ref = rw\_verify\_area에서 진행된 후 if(file->f\_op->read) 조건문 안에서 진행되어진다.



4)번은 fput\_light에서진행된다.



16) Make a file, /f1. Write some text in it.

# cd /

# vi f1

..........

#

Try to read this file before “mount\_root”, after “mount\_root”, after sys\_mount(“.”, “/”, ...), and after sys\_chroot(“.”) in init/do\_mounts.c/prepare\_namespace(). Explain what happens and why. For this problem, the kernel\_init process should exec to /sbin/init.