1. When user program execute read(fd,10,n)
2. file <usys.S> is executed, this file define the global symbol “read”, transform the “read” to SYS\_read, move it to %eax, and then make an interrupt of type T\_SYSCALL

#define SYSCALL(name) \

.globl name; \

name: \

movl $SYS\_ ## name, %eax; \

int $T\_SYSCALL; \

ret

SYSCALL(fork)

SYSCALL(exit)

SYSCALL(wait)

SYSCALL(pipe)

SYSCALL(read)

1. function trap() in file <trap.c> is executed, trap() judges the interrupt type as “T\_SYSCALL” , then call function syscall() in file <syscall.c>
2. void
3. trap(struct trapframe \*tf)
4. {
5. if(tf->trapno == T\_SYSCALL){
6. if(cp->killed)
7. exit();
8. cp->tf = tf;
9. syscall();
10. if(cp->killed)
11. exit();
12. return;
13. }
14. In file <syscall.c> define the function pointer array of system calls, call the sys\_read() according to the “SYS\_read” in %eax

//define extern function pointer of system calls

//define the function pointer array of system calls

static int (\*syscalls[])(void) = {

[SYS\_chdir] sys\_chdir,

[SYS\_close] sys\_close,

[SYS\_dup] sys\_dup,

[SYS\_exec] sys\_exec,

[SYS\_exit] sys\_exit,

[SYS\_fork] sys\_fork,

[SYS\_fstat] sys\_fstat,

[SYS\_getpid] sys\_getpid,

[SYS\_kill] sys\_kill,

[SYS\_link] sys\_link,

[SYS\_mkdir] sys\_mkdir,

[SYS\_mknod] sys\_mknod,

[SYS\_open] sys\_open,

[SYS\_pipe] sys\_pipe,

[SYS\_read] sys\_read,

[SYS\_sbrk] sys\_sbrk,

[SYS\_sleep] sys\_sleep,

[SYS\_unlink] sys\_unlink,

[SYS\_wait] sys\_wait,

[SYS\_write] sys\_write,

};

call the sys\_read() according the “SYS\_read” in %eax

void

syscall(void)

{

int num;

num = cp->tf->eax;

if(num >= 0 && num < NELEM(syscalls) && syscalls[num])

cp->tf->eax = syscalls[num]();

else {

cprintf("%d %s: unknown sys call %d\n",

cp->pid, cp->name, num);

cp->tf->eax = -1;

}

}

1. sys\_read() is in file <sysfile.c>, sys\_read() will create a file pointer f, call argfd() which is also in file<sysfile.c>, pass the (0,0,&f) to the argfd();

int

sys\_read(void)

{

struct file \*f;

int n;

char \*p;

if(argfd(0, 0, &f) < 0 || argint(2, &n) < 0 || argptr(1, &p, n) < 0)

return -1;

return fileread(f, p, n);

}

1. argfd() creat a fd, call argint() in file <syscall.c> to fetch the 0th system call argument as a file descriptor to fd,
2. static int
3. argfd(int n, int \*pfd, struct file \*\*pf)
4. {
5. int fd;
6. struct file \*f;
7. if(argint(n, &fd) < 0)
8. return -1;
9. if(fd < 0 || fd >= NOFILE || (f=cp->ofile[fd]) == 0)
10. return -1;
11. if(pfd)
12. \*pfd = fd;
13. if(pf)
14. \*pf = f;
15. return 0;
16. }

7. argint() call fetchstr() to fetch the 0th argument to fd, program return -1 to argfd(), since fd is not defined,

if(fd < 0 || fd >= NOFILE || (f=cp->ofile[fd]) == 0) return -1;

will be true and -1 is returned to sys\_read()

sys\_read() will also set curproc->tf->eax with -1 in syscall()

-1 is returned to user program by usys.S