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)
```

3. function trap() in file <trap.c> is executed, trap() judges the interrupt type as “T_SYSCALL” , then call function syscall() in file <syscall.c>

```
void
trap(struct trapframe *tf)
{
  if(tf->trapno == T_SYSCALL){
    if(cp->killed)
      exit();
    cp->tf = tf;
    syscall();
    if(cp->killed)
      exit();
    return;
  }
```

16. 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;
  }
}
```

17. 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);
}
```

18. argfd() creat a fd, call argint() in file <syscall.c> to fetch the 0th system call argument as a file descriptor to fd,

```
19. static int
20. argfd(int n, int *pfd, struct file **pf)
21. {
22.   int fd;
23.   struct file *f;
24.
25.   if(argint(n, &fd) < 0)
26.     return -1;
27.   if(fd < 0 || fd >= NOFILE || (f=cp->ofile[fd]) == 0)
28.     return -1;
29.   if(pfd)
30.     *pfd = fd;
31.   if(pf)
32.     *pf = f;
33.   return 0;
34. }
```

7. argint() call fetchstr() to fetch the 0^{th} 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