Describtion of the close(fd)

Step 1: User calls close(fd)

This will invoke a function defined in ***usys.S***

```
SYSCALL(close)
```

The following macro will execute, ***eax*** means the value of SYS_close should be put in the ***eax*** register.

```
#define SYSCALL(name) \
  .globl name; \
  name: \
    movl $SYS_ ## name, %eax; \
    int $T_SYSCALL; \
    ret
```

This value of SYS_close comes from the definition in ***syscall.h***. Since 21 is the value of SYS_close, the ***eax*** register will have 21.

```
#define SYS_close  21
```

The value of ***int*** becomes 64 as the value of T_SYSCALL is 64 in ***traps.h***:

```
#define T_SYSCALL       64      // system call
```

Step 2: trap function call

The trap function in ***trap.c*** is called next:

```
void trap(struct trapframe *tf) {
    ...
}
```

In this function, we check if the passed trapno is a valid systemcall. If the syscall is valid, then it calls ***syscall()***;

Step 3: syscall function

The syscall function is defined in the ***syscall.c***, the function checks the value of the ***eax*** register. If the value is a valid trap number.

```
void syscall(void) {
 ...
}
```

Then the function calls the function as defind in ***syscall.c***

```
static int (*syscalls[])(void) = {
    [SYS_close]    sys_close,
    ...
};
```

The sys_close function is defined in the ***sysfile.c*** file.

```
int sys_close(void) {
  ...
}
```

Step 4: sys_close

The sys_close function calles argfd() as defined in ***sysfile.c***, since with a non-existing file descriptor, it returns ***-1***.

```
static int argfd(int n, int *pfd, struct file **pf) {
  int fd;
  struct file *f;

  if(argint(n, &fd) < 0)
    return -1;
  if(fd < 0 || fd >= NOFILE || (f=proc->ofile[fd]) == 0)
    return -1;
  if(pfd)
    *pfd = fd;
  if(pf)
    *pf = f;
  return 0;
}
```