System Call Trace: close()

1. close(fd) is called where fd is an invalid file descriptor.

```
5 int main(int argc, char *argv[]) {
6     int fd = open("fileNotExist.bat", O_RDONLY);
7     close(fd);
8     exit();
9 }
```

2. The program traces the close() function to **usys.S** where SYSCALL(close)is defined at line:17.

```
16 SYSCALL(write)
17 SYSCALL(close)
18 SYSCALL(kill)
```

3. In **usys.S** the register %eax is set to sys_close, the value for which is defined in **syscall.h**
   *1. setting the **%eax** register*

```
4 #define SYSCALL(name) \
5   .globl name; \
6   name: \
7     movl $SYS_ ## name, %eax; \
8     int $T_SYSCALL; \
9     ret
10
```

   2. the value of **%eax** is set to 21

```
19 #define SYS_unlink 18
20 #define SYS_link   19
21 #define SYS_mkdir  20
22 #define SYS_close  21
```

4. Interrupt T_SYSCALL is raised. T_SYSCALL is defined in **traps.h** as 64.

This is where the PERL script in **usys.S** calls **trapasm.S**

```
# Call trap(tf), where tf=%esp
pushl %esp
call trap
addl $4, %esp
```

The 'call trap' code sends us to **trap.c**.
At line 43, syscall() is called. This function call takes us to **syscall.c**.

```
36 void
37 trap(struct trapframe *tf)
38 {
39   if(tf->trapno == T_SYSCALL){
40     if(myproc()->killed)
41       exit();
42     myproc()->tf = tf;
43     syscall();
44     if(myproc()->killed)
45       exit();
46     return;
47   }
48
```

The function is responsible for reading the contents of the register %eax (which is stored by the trapframe of the current process). Since eax stores 21, it calls sys_close() in **sysfile.c**

```
133 void
134 syscall(void)
135 {
136   int num;
137   struct proc *curproc = myproc();
138
139   num = curproc->tf->eax;
140   if(num > 0 && num < NELEM(syscalls) && syscalls[num]) {
141     curproc->tf->eax = syscalls[num]();
142   } else {
```

5. In **sysfile.c,** the sys_close() function is invoked.

```
93 int
94 sys_close(void)
95 {
96   int fd;
97   struct file *f;
98
99   if(argfd(0, &fd, &f) < 0)
100    return -1;
101  myproc()->ofile[fd] = 0;
102  fileclose(f);
103  return 0;
104 }
```

6. sys_close() calls the function argfd() which returns -1 to sys_close() when invalid file descriptor is detected.

```
21 static int
22 argfd(int n, int *pfd, struct file **pf)
23 {
24   int fd;
25   struct file *f;
26
27   if(argint(n, &fd) < 0)
28     return -1;
29   if(fd < 0 || fd >= NOFILE || (f=myproc()->ofile[fd]) == 0)
30     return -1;
31   if(pfd)
32     *pfd = fd;
33   if(pf)
34     *pf = f;
35   return 0;
36 }
```

7. sys_close() checks if the value returned by argfd is below 0. If it is, then it returns -1.

Finally, %eax is set to -1. Which is the return value of close(fd).