close()

Definition of close()

The close system call releases a file descriptor, making it free for reuse by a future open, pipe, or dup system call (see below). A newly allocated file descriptor is always the lowest-numbered unused descriptor of the current process

Description of the program

This document traces the close() system call from its invocation in user program and follows the lines of code which are executed to return a failure message for an invalid file descriptor

▼ testClose.c is created to provide an invalid fd and to run close(fd). This c file imports everything from user.h. So we navigate to user.h

```
int fd = 2343;
close(fd);
```

▼ user.h has definitions for function calls in the following way. Based on the comment //system calls indicated that the close is implemented as a system call sys_close().

```
// system calls
int close(int);
```

▼ syscall.c uses extern keyword which basically makes it open to be implemented in any file located in xv6 directory. Since sys_close() is related to files, we can follow sys_close()

```
// extern keyword means that it can be called from anywhere as long as its implmented
extern int sys_close(void);
```

▼ sysfile.c has an implementation of sys_close() as follows. argfd takes a struct for file and the file descriptor and prepares the file to be closed, opened etc. argf(0, &fd, &f) returns a value of -1 if the file passed is invalid

```
int
sys_close(void)
{
  int fd;
  cprintf("sys_close has been invoked\n");
  struct file *f;

  if(argfd(0, &fd, &f) < 0){
    // we added this print line to check if this condition passes
    cprintf("the condition passed for argfd i.e. file couldn't be found? \n");
    return -1;
  }
  // this sets the file to close if it passes the previous condition.
  myproc()->ofile[fd] = 0;
  fileclose(f);
  return 0;
}
```

▼ argfd() makes certain checks if the file doesn't exist. That method returns a -1 if the file is closed and this is how the function checks if the file passed is invalid.

```
static int
argfd(int n, int *pfd, struct file **pf)
{
  // cprintf("argfd is executed \n");
  int fd;
  struct file *f;

  if(argint(n, &fd) < 0)
    return -1;
  if(fd < 0 || fd >= NOFILE || (f=myproc()->ofile[fd]) == 0){
    cprintf("(f = %d) \n",f);
    // cprintf("NOFILE number: %d \n",NOFILE);
    cprintf("file wasn't found \n");
    return -1;
  }
  if(pfd)
    *pfd = fd;
  if(pf)
    *pf = f;
  return 0;
}
```

▼ (f=myproc()→ofile[fd]) == 0) condition is satisfied in the previous code block which basically signifies myproc() in proc.c checks the status of a file and returns 0 if its not open. If the file is open, it should return 1.

▼ The following process termination is brought up by the CPU to signify that file is not well handled with and it needs to kill the child process.

```
pid 10 testClose: trap 14 err 5 on cpu 0 eip 0xffffffff addr 0xffffffff--kill proc
```

▼ When I execute testClose on xv6, it returns me the following to help me navigate the program.

```
fork1 is called
execcmd has been executed
runcmd has started execution
testClose has been executed
sys_close has been invoked
(f = 0)
file wasn't found i think
the condition passed for argfd i.e. file couldn't be found?
pid 11 testClose: trap 14 err 5 on cpu 0 eip 0xffffffff addr 0xffffffff--kill proc
```