

CS 33

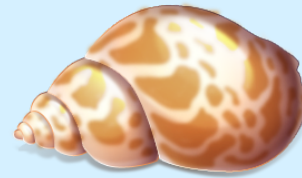
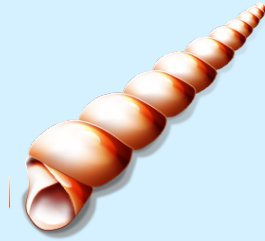
Shells and Files

Shells



- **Command and scripting languages for Unix**
- **First shell: Thompson shell**
 - sh, developed by Ken Thompson
 - released in 1971
- **Bourne shell**
 - also sh, developed by Steve Bourne
 - released in 1977
- **C shell**
 - csh, developed by Bill Joy
 - released in 1978
 - tcsh, improved version by Ken Greer

More Shells



- **Bourne-Again Shell**
 - bash, developed by Brian Fox
 - released in 1989
 - found to have a serious security-related bug in 2014
 - » shellshock
- **Almquist Shell**
 - ash, developed by Kenneth Almquist
 - released in 1989
 - similar to bash
 - dash (debian ash) used for scripts in Debian and Ubuntu Linux
 - » faster than bash
 - » less susceptible to shellshock vulnerability

The File Abstraction

- **A file is a simple array of bytes**
- **A file is made larger by writing beyond its current end**
- **Files are named by paths in a naming tree**
- **System calls on files are synchronous**

Naming

- **(almost) everything has a path name**
 - **files**
 - **directories**
 - **devices (known as *special files*)**
 - » **keyboards**
 - » **displays**
 - » **disks**
 - » **etc.**

I/O System Calls

- **int** file_descriptor = open(pathname, mode [, permissions])
- **int** close(file_descriptor)
- **int** count = read(file_descriptor, buffer_address, buffer_size)
- **int** count = write(file_descriptor, buffer_address, buffer_size)

Uniformity

```
int filefd = open("/home/twd/data", O_RDWR);  
    // opening a normal file  
int devicefd = open("/dev/tty", O_RDWR);  
    // opening a device (one's terminal  
    // or window)  
// file and device are file descriptors  
  
int bytes = read(filefd, buffer, sizeof(buffer));  
write(devicefd, buffer, bytes);
```

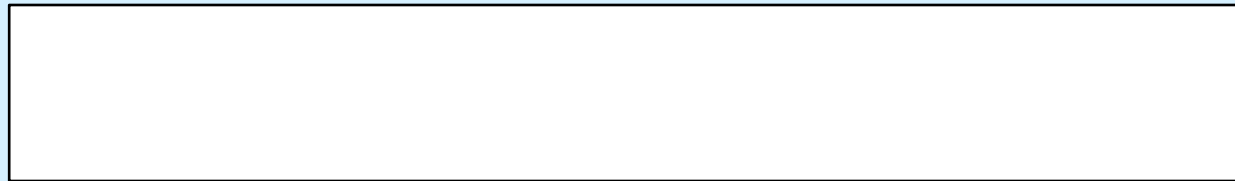
Standard File Descriptors

```
int main( ) {
    char buf[BUFSIZE];
    int n;
    const char *note = "Write failed\n";

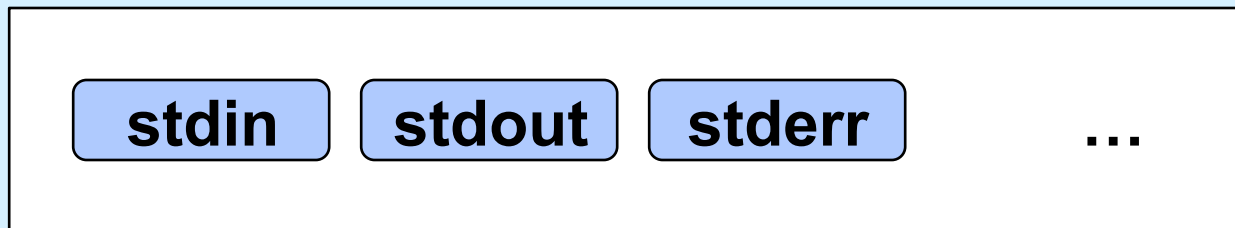
    while ((n = read(0, buf, sizeof(buf))) > 0)
        if (write(1, buf, n) != n) {
            write(2, note, strlen(note));
            exit(1);
        }
    return (0);
}
```


Standard I/O Library

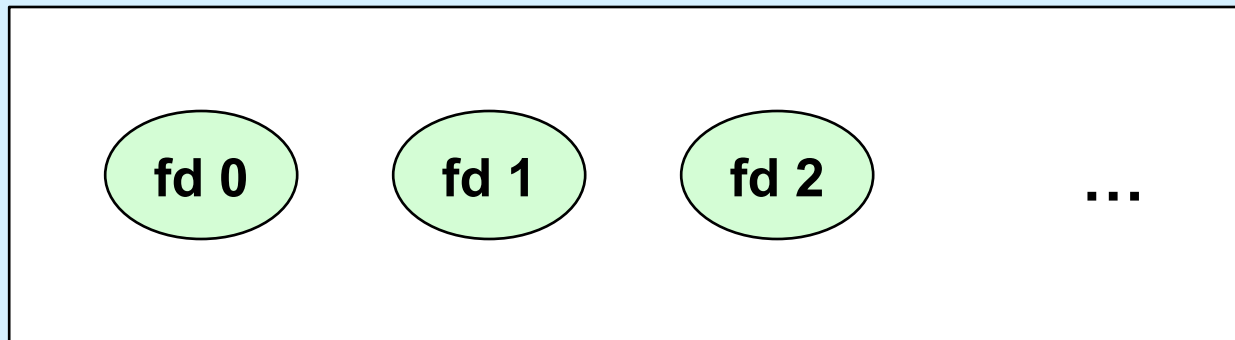
Formatting



Buffering



Syscalls



Standard I/O

```
FILE *stdin;           // declared in stdio.h
FILE *stdout;          // declared in stdio.h
FILE *stderr;          // declared in stdio.h

scanf("%d", &in);       // read via f.d. 0
printf("%d\n", in);     // write via f.d. 1
fprintf(stderr, "there was an error\n");
                        // write via f.d. 2
```

A Program

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: echon reps\n");
        exit(0);
    }
    int reps = atoi(argv[1]);
    if (reps > 2) {
        fprintf(stderr, "reps too large, reduced to 2\n");
        reps = 2;
    }
    char buf[256];
    while (fgets(buf, 256, stdin) != NULL)
        for (int i=0; i<reps; i++)
            fputs(buf, stdout);
    return(0);
}
```

From the Shell ...

```
$ echon 1
```

- ***stdout*** (fd 1) and ***stderr*** (fd 2) go to the display
- ***stdin*** (fd 0) comes from the keyboard

```
$ echon 1 > Output
```

- ***stdout*** goes to the file “Output” in the current directory
- ***stderr*** goes to the display
- ***stdin*** comes from the keyboard

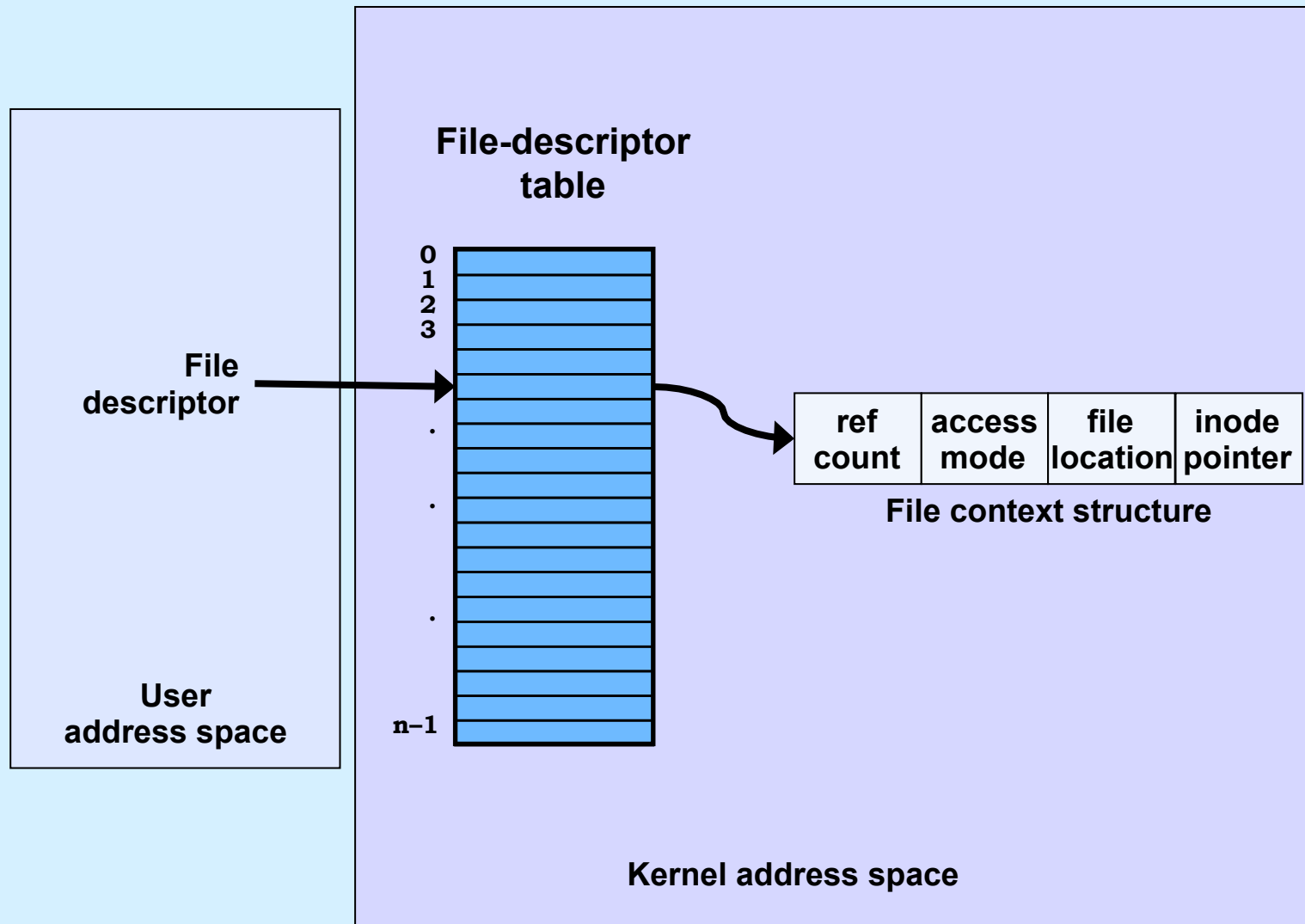
```
$ echon 1 < Input
```

- ***stdin*** comes from the file “Input” in the current directory

Running It

```
if (fork() == 0) {  
    /* set up file descriptor 1 in the child process */  
    close(1);  
    if (open("/home/twd/Output", O_WRONLY) == -1) {  
        perror("/home/twd/Output");  
        exit(1);  
    }  
    char *argv[] = {"echon", "2", NULL};  
    execv("/home/twd/bin/echon", argv);  
    exit(1);  
}  
  
/* parent continues here */  
  
while (pid != wait(0))      /* ignore the return code */  
    ;
```

File-Descriptor Table



Allocation of File Descriptors

- **Whenever a process requests a new file descriptor, the lowest-numbered file descriptor not already associated with an open file is selected; thus**

```
#include <fcntl.h>
#include <unistd.h>
```

```
close(0);
fd = open("file", O_RDONLY);
```

- **will always associate *file* with file descriptor 0 (assuming that *open* succeeds)**

Redirecting Output ... Twice

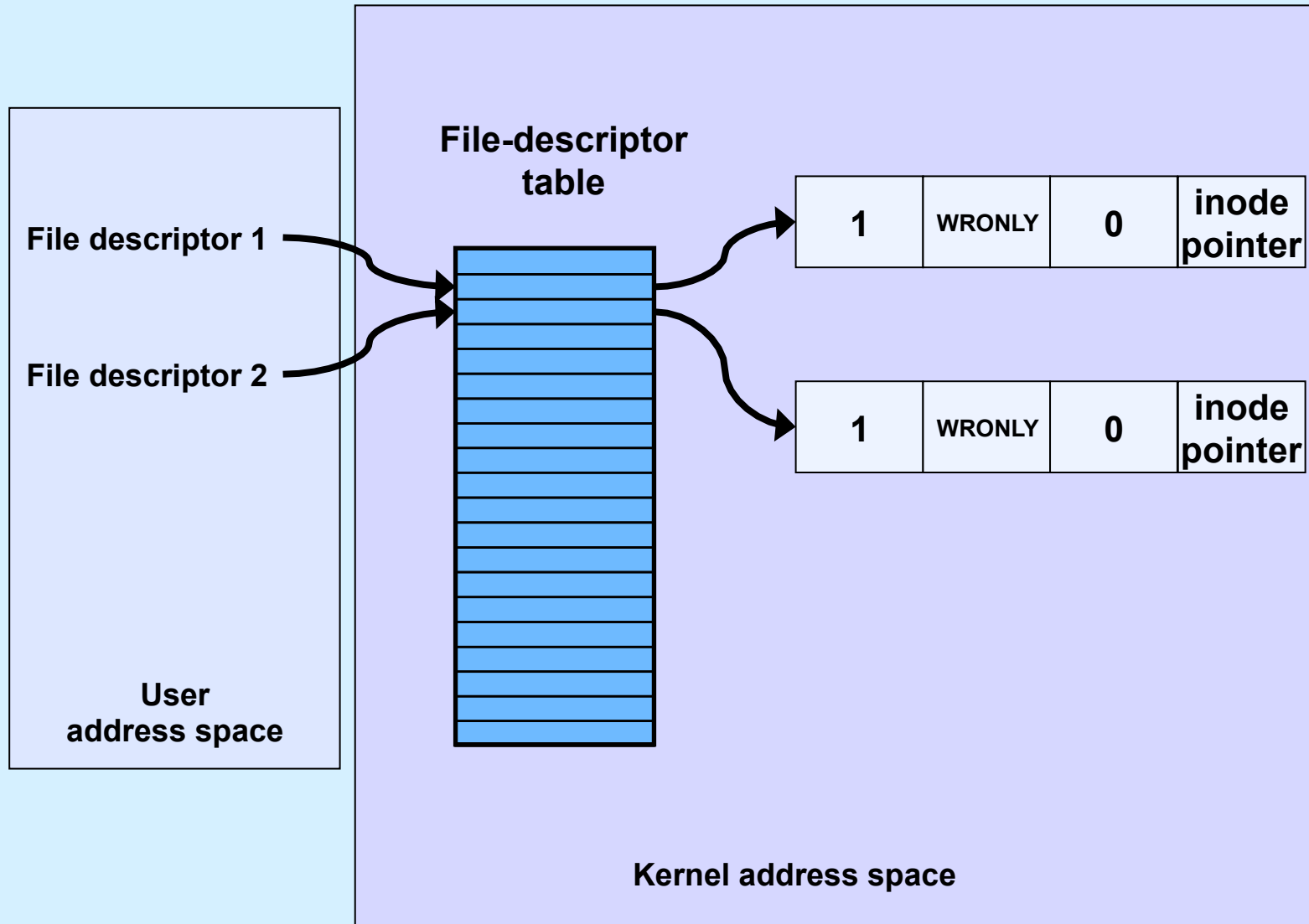
```
if (fork() == 0) {
    /* set up file descriptors 1 and 2 in the child process */
    close(1);
    close(2);
    if (open("/home/twd/Output", O_WRONLY) == -1) {
        exit(1);
    }
    if (open("/home/twd/Output", O_WRONLY) == -1) {
        exit(1);
    }
    char *argv[] = {"echon", 2, NULL};
    execv("/home/twd/bin/echon", argv);
    exit(1);
}
/* parent continues here */
```


From the Shell ...

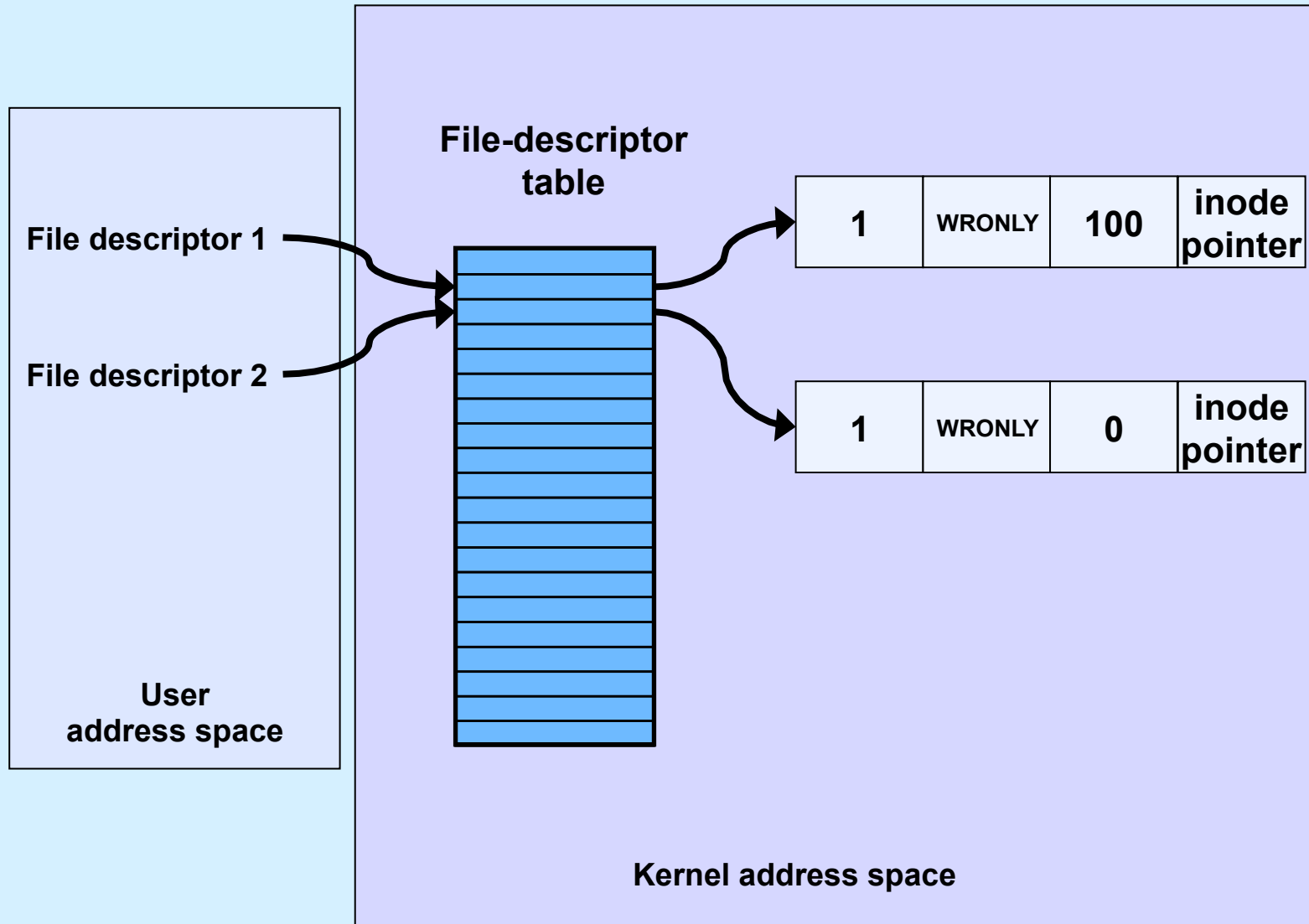
```
$ echon 1 >Output 2>Output
```

– **both stdout and stderr go to Output file**

Redirected Output



Redirected Output After Write



Quiz 1

- **Suppose we run**

```
% echon 3 >Output 2>Output
```

- **The input line is**

```
X
```

- **What is the final content of Output?**

a) reps too large, reduced to 2\nX\nX\n

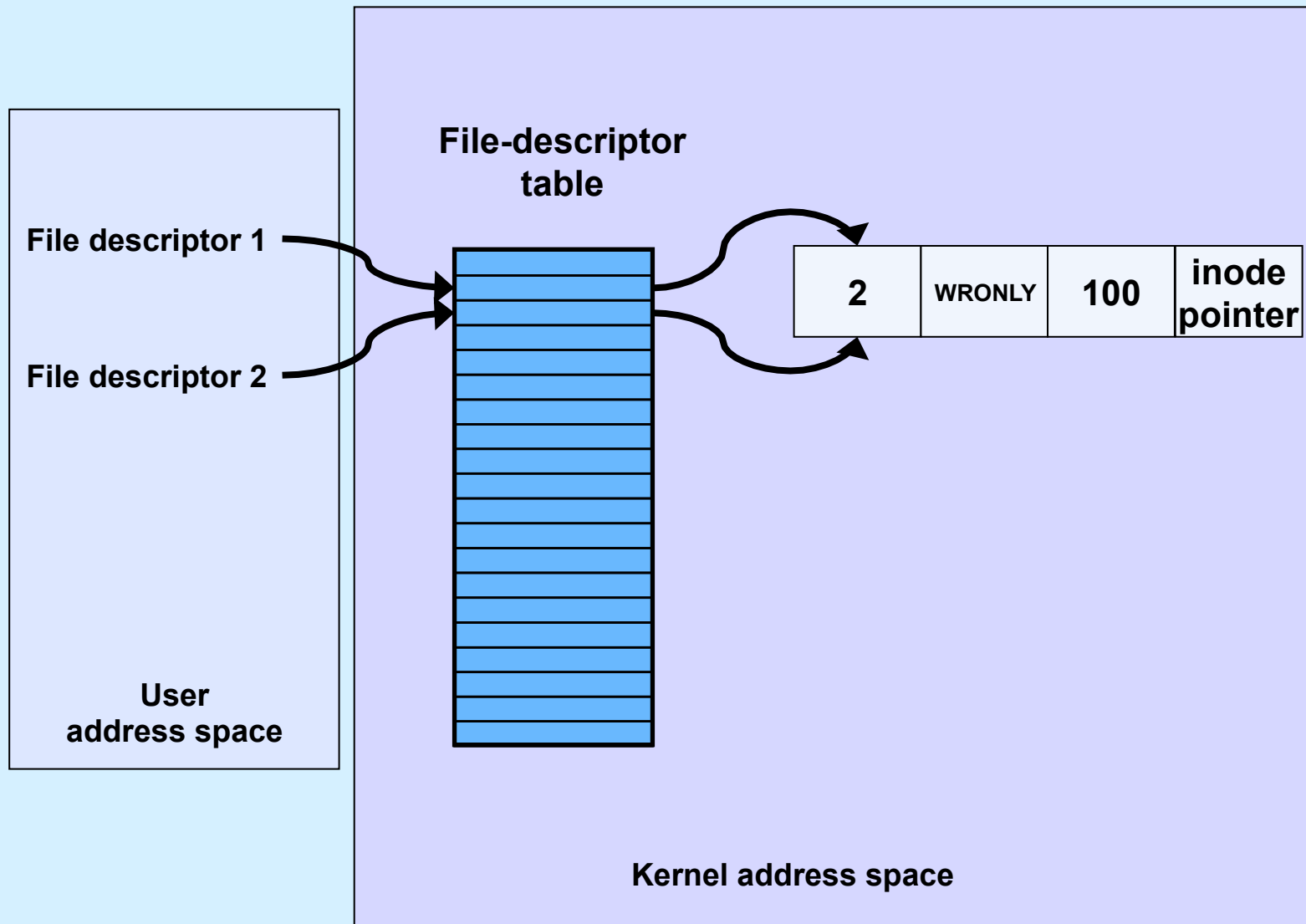
b) X\nX\nreps too large, reduced to 2\n

c) X\nX\n too large, reduced to 2\n

Sharing Context Information

```
if (fork() == 0) {  
    /* set up file descriptors 1 and 2 in the child process */  
    close(1);  
    close(2);  
    if (open("/home/twd/Output", O_WRONLY) == -1) {  
        exit(1);  
    }  
    dup(1); /* set up file descriptor 2 as a duplicate of 1 */  
    char *argv[] = {"echon", 2};  
    execv("/home/twd/bin/echon", argv);  
    exit(1);  
}  
/* parent continues here */
```

Redirected Output After Dup



From the Shell ...

```
$ echon 3 >Output 2>&1
```

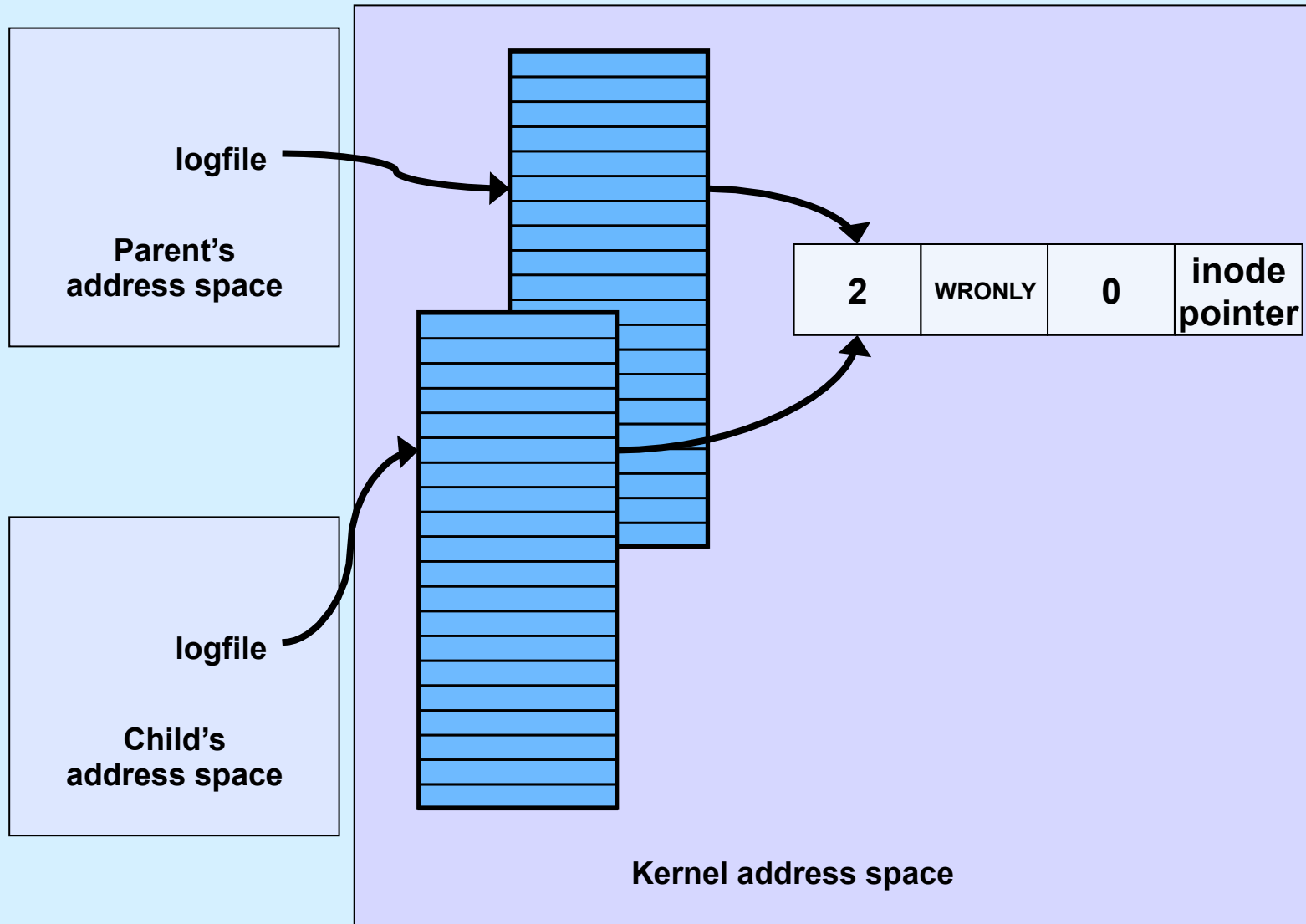
– **stdout goes to Output file, stderr is the dup of fd 1**

Fork and File Descriptors

```
int logfile = open("log", O_WRONLY);
if (fork() == 0) {
    /* child process computes something, then does: */
    write(logfile, LogEntry, strlen(LogEntry));
    ...
    exit(0);
}

/* parent process computes something, then does: */
write(logfile, LogEntry, strlen(LogEntry));
...
```


File Descriptors After Fork



Quiz 2

```
int main() {  
    if (fork() == 0) {  
        fprintf(stderr, "Child");  
        exit(0);  
    }  
    printf("Parent");  
}
```

Suppose the program is run as:

```
% prog >file 2>&1
```

What is the final content of file?

- a) either “ChildParent” or “ParentChild”
 - b) either “Childt” or “Parent”
 - c) either “Child” or “Parent”
-