Nathan Bender

CS 283 – Systems Programming

H2 – Homework 2

10.6

The output of this program would be:

fd2 = 4

When the first file is opened, the smallest file descriptor that is unused is returned. This is the number 3, since 0, 1 and 2 are open for the files STDIN, STDOUT and STDERR. When the second file is opened, the file descriptors 0,1,2,3 are all being used, so the smallest file descriptor is 4.

10.8

```
#include "csapp.h"


int main(int argc, char ** argv) {
	if (argc != 2) {
		printf("You must include the file descriptor number as an argument!");
		exit(0);
	}

	struct stat stat;
	char *type, *readok ;

	fstat(atoi(argv[1]), &stat);
	if (S_ISREG(stat.st_mode))
		type = "regular";
	else if (S_ISDIR(stat.st_mode))
		type = "directory";
	else
		type = "other";
```

```
        if((stat.st_mode & S_IRUSR))
                readok = "yes";
        else
                readok = "no";

        printf("type : %s, read : %s\n", type, readok);
        exit(0);
}
```

In this question, we were tasked with modifying a program that used the stat function to view information about a file that was passed as a command line argument to the program. Originally, the function used a filename to view the information. The problem asked us to modify the program to use a file descriptor number instead. To use a file descriptor, the fstat function was used instead. I also had to use the atoi function to convert the argument, which was a char*, to an integer, which is the type of a file descriptor.

10.10

```
#include "csapp.h"

int main(int argc, char ** argv) {
        int n;
        rio_t rio;
        char buf[MAXLINE];

        if (argc == 2) {
                int fd = open(argv[1], O_RDONLY, 0);
                if (fd < 0) {
                        printf("Error opening file with name : %s", argv[1]);
                        exit(0);
                }
                dup2(fd, STDIN_FILENO);
                close(fd);
        }

        rio_readinitb(&rio, STDIN_FILENO);
        while ((n = rio_readlineb(&rio, buf, MAXLINE)) != 0)
                rio_writen(STDOUT_FILENO, buf, n);

        exit(0);
}
```

In this question, we were tasked with adding an optional parameter to the a program that copies standard input to standard output. If the argument is given, open that file and print the file to standard output. The

condition, however, was that the original code could not be modified. So, to do this, I had to use input redirection in order to redirect the file to standard input. The code in red above does this – it copies the file descriptor for the file opened to standard input.

tee command

```
#include "csapp.h"

int main(int argc, char ** argv) {
        int n, fd, c;
        rio_t rio;
        char buf[MAXLINE];

        if (argc == 2) {
                fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0644);
                if (fd < 0) {
                        printf("Error opening file with name : %s.\n", argv[1]);
                        exit(0);
                }

                printf("Created and truncated file %s.\n", argv[1]);
        }
        else if ((c = getopt(argc, argv, "a:")) != -1){
                fd = open(optarg, O_WRONLY | O_CREAT | O_APPEND, 0644);
                if (fd < 0) {
                        printf("Error opening file with name : %s.\n", optarg);
                        exit(0);
                }
        }
        else {
                printf("You must supply the file name as a parameter.");
                printf("Please pass the -a argument to append to file.");
                printf("File name should appear after the file.");
                printf("\n\nE.g:");
                printf("./tee hello.txt");
                printf("./tee -a hello.txt");
                exit(0);
        }


        rio_readinitb(&rio, STDIN_FILENO);
        while ((n = rio_readlineb(&rio, buf, MAXLINE)) != 0) {
                rio_writen(STDOUT_FILENO, buf, n);
                rio_writen(fd, buf, n);
        }

        close(fd);
        exit(0);
}
```