Advanced Unix Programming Lab Assignment 3

111403050 Abhijeet Kale

111403067 Sagar Panchmatia

111403076 Nikunj Singh

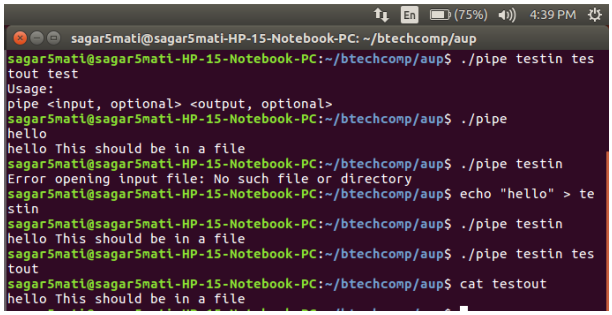
Question 1: Using dup function redirect stdin to file1 and stdout to file2. Read a line using scanf and write the same using printf. Verify the contents of both files.

➔ dup function returns a new file descriptor which can be used interchangeably be used with the old file descriptor. dup2 on the other hand takes new file descriptor as argument and tries to close any file stream attached to it and then assigns that descriptor to the old descriptor. So we can pass 0 which is stdin and 1 which is stdout as arguments to dup2 function, close them and use scanf and printf to write to stdin and stdout as they now point to our files.

Code:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>

int main(int argc, char *argv[]) {
	int infile, outfile;
	char str[10];

	infile = 0;
	outfile = 1;
	if(argc > 3) {
		printf("Usage:\npipe <input, optional> <output, optional>\n");
		return 1;
	}

	if(argc > 1) {
		if((infile = open(argv[1], O_RDONLY)) == -1) {
			perror("Error opening input file");
			return -1;
		}
		dup2(infile, 0);
	}

	if(argc > 2) {
		if((outfile = open(argv[2], O_WRONLY | O_CREAT
			| O_TRUNC, 00664)) == -1) {
		perror("Error opening output file");
```

```
            return -1;
            }
            dup2(outfile, 1);
        }

        scanf("%s", str);
        printf("%s This should be in a file\n", str);

        return 0;
}
```



Execution

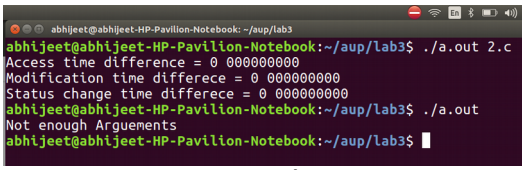
Question 2: Does calling stat function change any of the time values? Verify with a program.

➔ A file has three timestamps. Access time gives the last time a file was read or accessed. Modification time tells last time the contents of the file were modified. Change time keeps track of the changes in the meta data, like the one returned by stat command. As stat only reads the meta data none of the timestamps are changed. The progarm below returns the difference between the timestamps.

Code:

```
#include <stdio.h>

#include <sys/types.h>

#include <errno.h>

#include <sys/stat.h>

void print_time_difference(struct stat sb1, struct stat sb2) {

	printf("Access time difference = ");

	printf("%lld %.9ld\n", (long long)(sb1.st_atim.tv_sec - sb2.st_atim.tv_sec),
sb1.st_atim.tv_nsec - sb2.st_atim.tv_nsec);

	printf("Modification time differece = ");

	printf("%lld %.9ld\n", (long long)(sb1.st_mtim.tv_sec - sb2.st_mtim.tv_sec),
sb1.st_mtim.tv_nsec - sb2.st_mtim.tv_nsec);

	printf("Status change time differece = ");

	printf("%lld %.9ld\n", (long long)(sb1.st_ctim.tv_sec - sb2.st_ctim.tv_sec),
sb1.st_ctim.tv_nsec - sb2.st_ctim.tv_nsec);

}

int main(int argc, char *argv[]) {

	if(argc < 2) {

		printf("Not enough Arguements\n");

		return 1;

	}

	struct stat sb1, sb2;

	if(stat(argv[1], &sb1)) {

		perror("");

		return 1;

	}

	if(stat(argv[1], &sb2)) {
```

```
        perror("");
        return 1;
    }
    print_time_difference(sb1, sb2);
    return 0;
}
```



Execution

Question 3: unmask() always sets the process umask and, at the same time, returns a copy of the old umask. How can we obtain a copy of the current process umask while leaving it unchanged? Write a program to demonstrate.

➔ The above program demonstrates that umask() always sets the process umask and, at the same time, returns a copy of the old umask. Hence we have obtained the current process umask without leaving it unchanged i.e. again restoring it.

Code:

```
#include <stdio.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>


mode_t current_umask;


mode_t read_umask(void);


int main(int argc, char *argv[]) {

	mode_t val = read_umask();

	printf("umask is %d\n", val);

}


mode_t read_umask() {

	// Sets the process umask and returns a copy of old umask

	current_umask = umask(0);

	// again setting it with current_umask wihout leaving it unchanged

	umask(current_umask);

	// returning the current umask

	return current_umask;

}
```

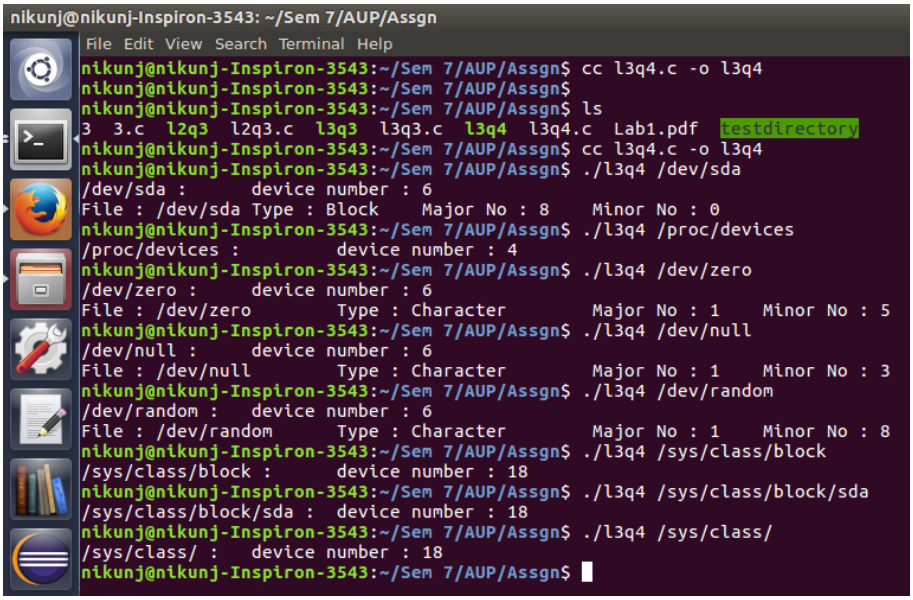
Question 4: Display the device number for the filename input as command line argument. If it is a character or block special file, then display its major and minor numbers.

➔ The stat struct has been used to retrieve the values of device number for device files. The values of major and minor number of the character special and block special files are also retrieved. Various files were given as input.

Code:

```
#include <stdio.h>

#include <errno.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>



struct stat buffer;


int main(int argc, char *argv[]) {

	if(argc < 2) {

		printf("Insufficient arguments\n");

		return -1;

	}

	char *filename = argv[1];

	if(stat(filename, &buffer) < 0) {

		printf("Error\n");

		return -1;

	}


	// Printing the device number

	printf("%s : \tdevice number : %ld\n", filename, buffer.st_dev);


	// Checking whether character file

	if(S_ISCHR(buffer.st_mode)) {
```

```
        printf("File : %s\tType : Character\tMajor No : %d\tMinor No : %d\n",
filename, major(buffer.st_rdev), minor(buffer.st_rdev));
    }

    // Checking whether a block file
    else if(S_ISBLK(buffer.st_mode)) {
        printf("File : %s\tType : Block\tMajor No : %d\tMinor No : %d\n",
filename, major(buffer.st_rdev), minor(buffer.st_rdev));
    }
    return 0;
}
```



Execution