B. Siva Shirish-192324016

25.Construct a C program to implement the I/O system calls of UNIX (fcntl, seek, stat, opendir, readdir)

Aim:

To implement a C program that demonstrates the usage of UNIX I/O system calls like `fcntl`, `seek`, `stat`, `opendir`, and `readdir`.

Algorithm:

1. Open a file using `open` system call.
2. Use `fcntl` to manipulate file descriptor properties.
3. Use `lseek` to reposition the file offset.
4. Use `stat` to retrieve file status information.
5. Use `opendir` to open a directory and `readdir` to read its contents.
6. Display the results of each operation.

Procedure:

1. Include the necessary headers (`fcntl.h`, `unistd.h`, `sys/stat.h`, etc.).
2. Use appropriate system calls to perform file and directory operations.
3. Handle errors appropriately (e.g., check return values).
4. Display the results of the operations.

Code:

```
#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#include <sys/stat.h>

#include <dirent.h>

#include <string.h>
```

```
int main() {
    int fd;
    char *fileName = "testfile.txt";
    char writeBuffer[] = "Sample data for I/O system calls demonstration.\n";
    char readBuffer[128];

    // Create a file and write data
    fd = open(fileName, O_CREAT | O_RDWR, 0644);
    if (fd < 0) {
        perror("Error opening/creating file");
        exit(EXIT_FAILURE);
    }
    write(fd, writeBuffer, strlen(writeBuffer));

    // Seek to the beginning of the file
    if (lseek(fd, 0, SEEK_SET) < 0) {
        perror("Error seeking in file");
        close(fd);
        exit(EXIT_FAILURE);
    }

    // Read data from the file
    ssize_t bytesRead = read(fd, readBuffer, sizeof(readBuffer) - 1);
```

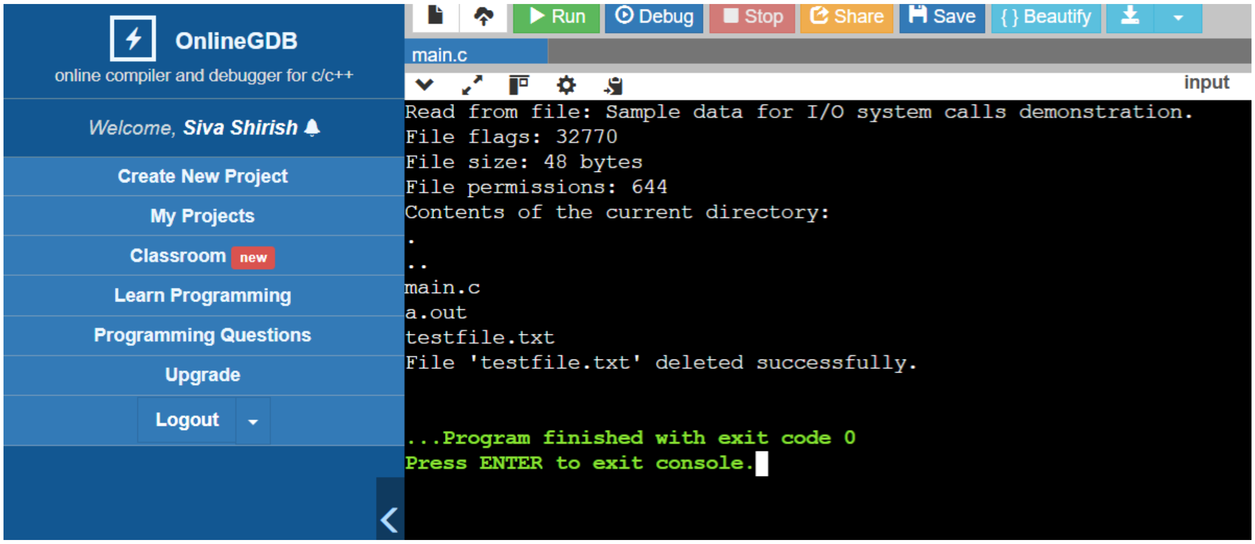
```
if (bytesRead < 0) {

    perror("Error reading file");

    close(fd);

    exit(EXIT_FAILURE);

}

readBuffer[bytesRead] = '\0';

printf("Read from file: %s", readBuffer);


// File status using fcntl

int flags = fcntl(fd, F_GETFL);

if (flags < 0) {

    perror("Error getting file flags");

} else {

    printf("File flags: %d\n", flags);

}


close(fd);


// File status using stat

struct stat fileStat;

if (stat(fileName, &fileStat) < 0) {

    perror("Error getting file status");

    exit(EXIT_FAILURE);
```

```
}
printf("File size: %ld bytes\n", fileStat.st_size);
printf("File permissions: %o\n", fileStat.st_mode & 0777);

// Directory operations using opendir and readdir
DIR *dir = opendir(".");
if (dir == NULL) {
    perror("Error opening directory");
    exit(EXIT_FAILURE);
}

struct dirent *entry;
printf("Contents of the current directory:\n");
while ((entry = readdir(dir)) != NULL) {
    printf("%s\n", entry->d_name);
}

closedir(dir);

// Clean up
if (unlink(fileName) < 0) {
    perror("Error deleting file");
    exit(EXIT_FAILURE);
    }

    printf("File '%s' deleted successfully.\n", fileName);



    return 0;
}
```

Result:

1. A file named `testfile.txt` is created or opened.
2. File descriptor properties are modified using `fcntl`.
3. The file offset is repositioned using `lseek`.
4. File details like size and permissions are fetched using `stat`.
5. Directory contents are listed using `opendir` and `readdir`.

Output:



```
Read from file: Sample data for I/O system calls demonstration.
File flags: 32770
File size: 48 bytes
File permissions: 644
Contents of the current directory:
.
..
main.c
a.out
testfile.txt
File 'testfile.txt' deleted successfully.


...Program finished with exit code 0
Press ENTER to exit console.
```