Department of Computer Science,
Faculty of Mathematical Sciences,
University of Delhi, Delhi 110007

Programming Assignment for
MCSC202 Advanced Operating Systems
(May/June Session-2023)

Submitted to
Dr. Bharti Rana
Assistant Professor,
Department of Computer Science,
University of Delhi

Submitted by:-
Gagan Kumar Soni
MSc Computer Science
Year I Semester II
Roll Number 21

files.h

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

void readFile(int argc, char *argv[]) {
    //./assgn file read file.txt start bytes
    if (argc != 6) {
        printf("Invalid number of arguments.\n");
    } else {
        // Open file in read-only mode
        int fd = open(argv[3], O_RDONLY);
        if (fd == -1) {
            printf("Error in opening file.\n");
        } else {
            // Convert string to integer for start, bytes
            int start = atoi(argv[4]);
            int nbytes = atoi(argv[5]);
            // Move pointer for random read
            if (start != 0) {
                lseek(fd, start, SEEK_SET);
            }
            int n, bytes_read = 0;
            char buff[1]; // Initialize buffer to store data
            printf("\nThe bytes read are: ");
            // Keep reading till the end of the file or max bytes to be read is
reached
            while (((n = read(fd, buff, 1)) > 0) && (bytes_read < nbytes)) {
                printf("%s", buff);
                bytes_read += n;
            }
            // Print total bytes read
            printf("\n\nTotal bytes read: %d\n", bytes_read);
            // Close file
            if (close(fd) < 0) {
                printf("Error in closing file.\n");
            }
        }
    }
}

void infoFile(int argc, char *argv[]) {
    //./assgn file info file.txt
    if (argc != 4) {
        printf("Invalid number of arguments.");
    } else {
        // Define stat to store file info
        struct stat sfile;

        if (stat(argv[3], &sfile) == -1) {
            printf("Error Occurred\n");
        }
        printf("\nInformation for file: %s\n", argv[3]);

        // Accessing data members of stat struct
        printf("\nUser ID of owner: %d", sfile.st_uid);
        printf("\nBlocksize for system I/O: %d", sfile.st_blksize);
        printf("\nGroup ID of owner: %d", sfile.st_gid);
        printf("\nNumber of blocks allocated: %d", sfile.st_blocks);
        printf("\nTotal size, in bytes: %d", sfile.st_size);
        printf("\nNumber of hard links: %u", (unsigned int) sfile.st_nlink);
        printf("\nFile Permissions for User: ");
        printf((sfile.st_mode & S_IRUSR) ? "r" : "-");
        printf((sfile.st_mode & S_IWUSR) ? "w" : "-");
        printf((sfile.st_mode & S_IXUSR) ? "x" : "-");
        printf("\nFile Permissions for Group: ");
        printf((sfile.st_mode & S_IRGRP) ? "r" : "-");
        printf((sfile.st_mode & S_IWGRP) ? "w" : "-");
        printf((sfile.st_mode & S_IXGRP) ? "x" : "-");
        printf("\nFile Permissions for Other: ");
        printf((sfile.st_mode & S_IROTH) ? "r" : "-");
        printf((sfile.st_mode & S_IWOTH) ? "w" : "-");
        printf((sfile.st_mode & S_IXOTH) ? "x" : "-");

        printf("\n");
    }
}

mode_t toMode(char *perm) {
    // Converts string permission to mode_t format
    // Example: "rw-r--r--" to 0644
    int mode = 0;
    int index = 0;
    // Set user permissions
    if (perm[index] == 'r')
        mode |= S_IRUSR;
    if (perm[index + 1] == 'w')
        mode |= S_IWUSR;
    if (perm[index + 2] == 'x')
        mode |= S_IXUSR;

    index += 3;

    // Set group permissions
    if (perm[index] == 'r')
        mode |= S_IRGRP;
    if (perm[index + 1] == 'w')
        mode |= S_IWGRP;
    if (perm[index + 2] == 'x')
        mode |= S_IXGRP;

    index += 3;

    // Set other permissions
    if (perm[index] == 'r')
        mode |= S_IROTH;
    if (perm[index + 1] == 'w')
        mode |= S_IWOTH;
    if (perm[index + 2] == 'x')
        mode |= S_IXOTH;

    return (mode_t)mode;
}

void createFile(int argc, char *argv[]) {
    //./assgn file create file.txt permissions
    if (argc != 5) {
        printf("Invalid number of arguments.\n");
    } else {
        // Convert permission string to mode_t format
        mode_t mode = toMode(argv[4]);

        // Create file with specified permissions
        int fd = open(argv[3], O_CREAT | O_EXCL, mode);

        if (fd == -1) {
            printf("Error in creating file.\n");
        } else {
            printf("File created successfully.\n");
            // Close file
```

```c
                if (close(fd) < 0) {
                    printf("Error in closing file.\n");
                }
            }
        }
}

int findOpenMode(char *m) {
    // Find open mode based on input string
    if (strcmp(m, "r") == 0) {
        return O_RDONLY;
    } else if (strcmp(m, "w") == 0) {
        return O_WRONLY | O_CREAT | O_TRUNC;
    } else if (strcmp(m, "a") == 0) {
        return O_WRONLY | O_CREAT | O_APPEND;
    } else {
        return -1;
    }
}

void openAndCloseFile(int argc, char *argv[]) {
    //./assgn file open file.txt mode
    if (argc != 5) {
        printf("Invalid number of arguments.\n");
    } else {
        int mode = findOpenMode(argv[4]);
        if (mode == -1) {
            printf("Invalid mode.\n");
        } else {
            // Open file with specified mode
            int fd = open(argv[3], mode, 0644);
            if (fd == -1) {
                printf("Error in opening file.\n");
            } else {
                printf("File opened successfully.\n");
                // Close file
                if (close(fd) < 0) {
                    printf("Error in closing file.\n");
                }
            }
        }
    }
}

void writeFile(int argc, char *argv[]) {
    //./assgn file write file.txt start text
    if (argc != 5) {
        printf("Invalid number of arguments.\n");
    } else {
        // Open file in write mode
        int fd = open(argv[3], O_WRONLY);
        if (fd == -1) {
            printf("Error in opening file.\n");
        } else {
            // Convert string to integer for start
            int start = atoi(argv[4]);
            // Move pointer for random write
            if (start != 0) {
                lseek(fd, start, SEEK_SET);
            }
            // Write text to file
            if (write(fd, argv[5], strlen(argv[5])) == -1) {
                printf("Error in writing to file.\n");
            } else {
                printf("Text written to file successfully.\n");
            }
            // Close file
            if (close(fd) < 0) {
```

```
                        printf("Error in closing file.\n");
                    }
                }
            }
}
```

pipes.h

```
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>

void createUnnamedPipe(int argc, char *argv[]){
        //./assgn pipe unnamed text
        if(argc!=4){
                printf("\nInvalid number of arguments.");
                return;
        }
        int fd[2], nbytes;
        pid_t childpid;
        char buffer[128];

        //Create unnamed pipe
        int p=pipe(fd);
        if(p<0){
                printf("\nError in creating named pipe.");
                return;
        }
        //Call fork
        if((childpid = fork()) == -1) {
                perror("fork");
                exit(1);
        }
        //if child process then write into pipe
        if(childpid == 0) {
                //close reading end
                close(fd[0]);
                //Write into pipe from buffer
                nbytes = write(fd[1], argv[3], (strlen(argv[3]) + 1));
                printf("\nBytes written by child process into pipe: %d \n",nbytes);
                exit(0);
        }else {//If parent process then read from child process
                //Close wriring end
                close(fd[1]);
                //Read from pipe into buffer
                nbytes = read(fd[0], buffer, sizeof(buffer));
                printf("\nBytes read by parent process from pipe: %d \n",nbytes);
                printf("\n\nData read by parent process: %s",buffer);
        }
}

void createNamedPipe(int argc, char *argv[]){
        //./assgn pipe named pipename
        if(argc!=4){
                printf("\nInvalid number of arguments.");
                return;
        }
     //Create named pipe
     int result = mknod (argv[3], S_IRUSR| S_IWUSR|S_IFIFO, 0);
        if (result < 0) {
            printf("\nError in creating named pipe.");
            exit (2);
        }else{
                printf("\nNamed pipe created successfully.");
        }
}
```

assignment.c

```
#include <stdio.h>
#include "files.h"
#include "pipes.h"

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

        printf("Welcome to program: %s\n\n", argv[0]);

        if(argc < 3){ //./assgn type operation
       printf("Please pass appropriate number of command line arguments for executing a 
function.\n");
        }
        else{
                //FILE
                if(strcmp(argv[1], "file")==0){
                        if(strcmp(argv[2], "create")==0){
                                createFile(argc,argv);
                        }else if(strcmp(argv[2],"read")==0){
                                readFile(argc,argv);
                        }else if(strcmp(argv[2],"write")==0){
                                writeFile(argc,argv);
                        }else if(strcmp(argv[2],"open")==0){
                                openAndCloseFile(argc,argv);
                        }else if(strcmp(argv[2],"info")==0){
                                infoFile(argc,argv);
                        }else if(strcmp(argv[2],"chmod")==0){
                                changeMode(argc,argv);
                        }else{
                                printf("Invalid command for regular files.\n");
                        }
                }else if(strcmp(argv[1],"pipe")==0){ //PIPE
                        if(strcmp(argv[2],"named")==0){
                                createNamedPipe(argc,argv);
                        }else if(strcmp(argv[2],"unnamed")==0){
                                createUnnamedPipe(argc,argv);
                        }else{
                                printf("Invalid command for directories.\n");
                        }
                }else{
                        printf("Invalid argument. First argument must be a file, pipe or 
directory.\n");
                }
        }
}
```

Documentation:-

The program consists of 4 files (a main file and 3 header files which contains all the functionalities):- assgn.c, files.h, pipes.h

The format for executing the program would be:

```
gcc -o assgn assgn.c
./assgn type operation parameter
```

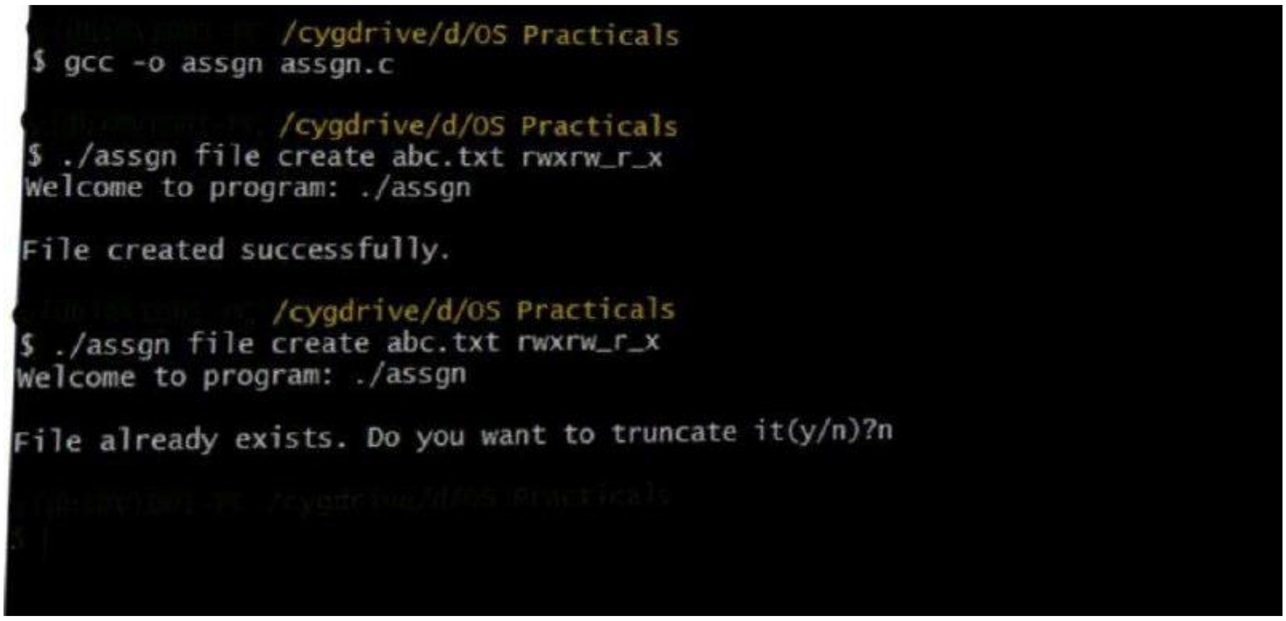
where type = file, pipe

The system calls implemented are:

- files:
  - create
  - open
  - close
  - write
  - read
  - chmod
  - stat
- pipes:
  - pipe (unnamed)
  - mknod (named)

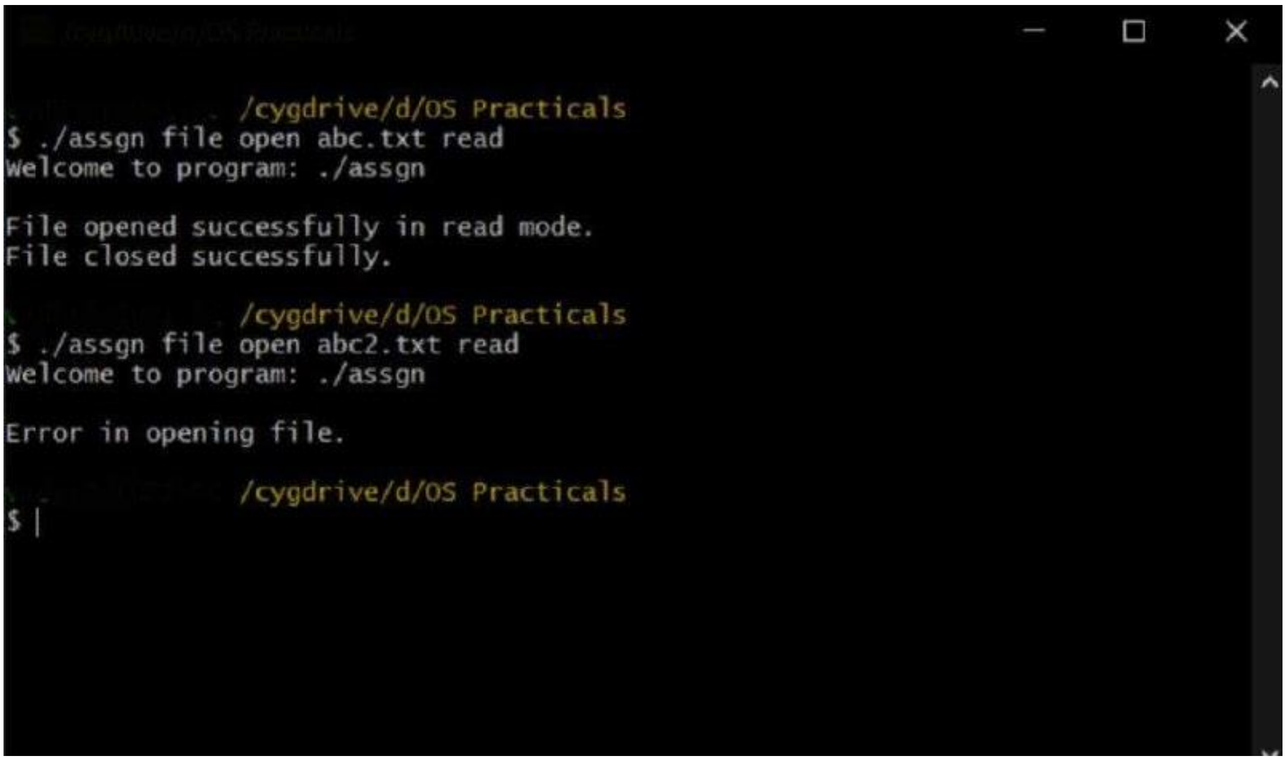
FILES

1. create



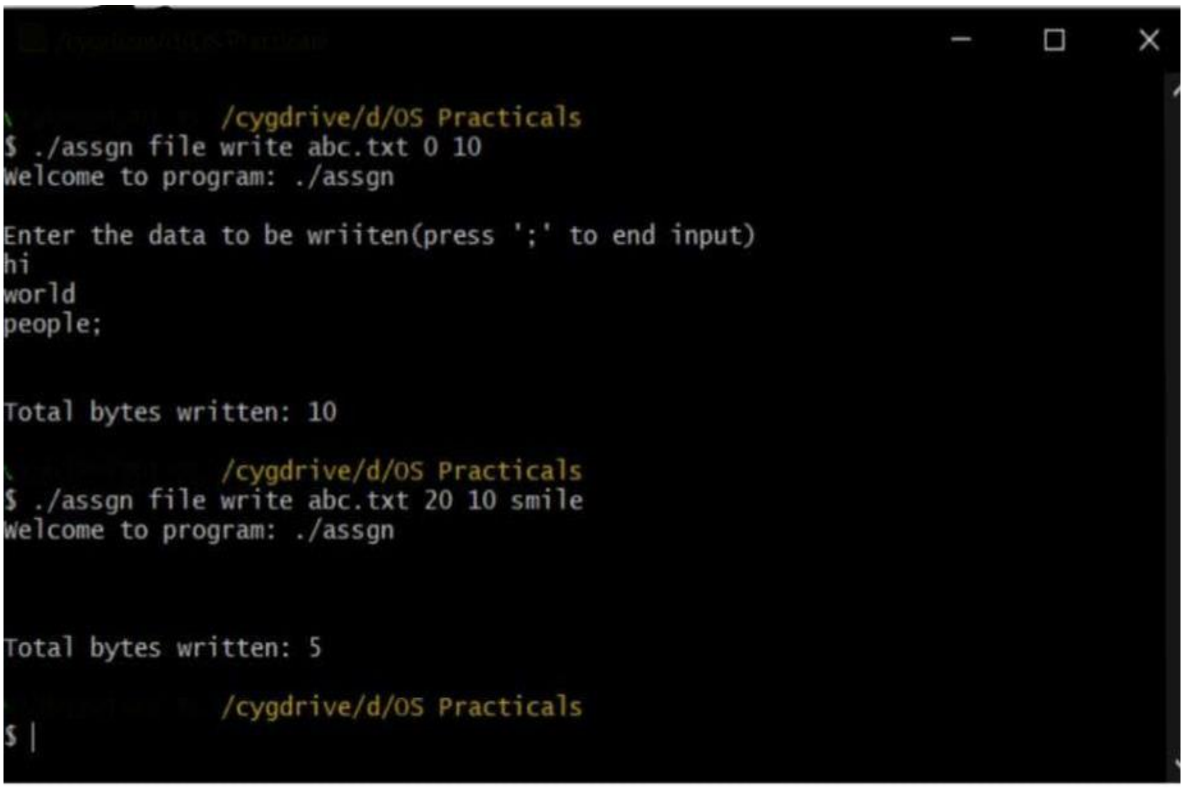
```
[illegible] /cygdrive/d/OS Practicals
$ gcc -o assgn assgn.c

[illegible] /cygdrive/d/OS Practicals
$ ./assgn file create abc.txt rwxrw_r_x
Welcome to program: ./assgn

File created successfully.

[illegible] /cygdrive/d/OS Practicals
$ ./assgn file create abc.txt rwxrw_r_x
Welcome to program: ./assgn

File already exists. Do you want to truncate it(y/n)?n

[illegible] /cygdrive/d/OS Practicals
$ |
```

2. open and close



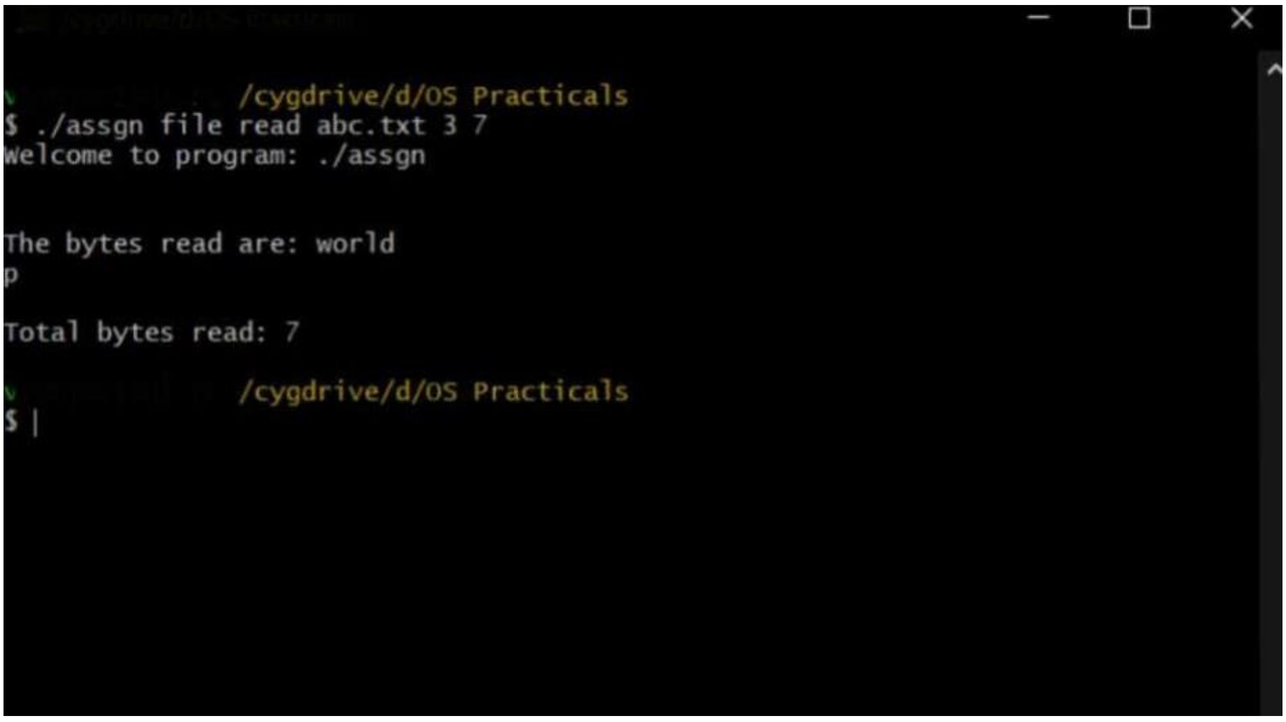
```
[illegible] /cygdrive/d/OS Practicals
$ ./assgn file open abc.txt read
Welcome to program: ./assgn

File opened successfully in read mode.
File closed successfully.

[illegible] /cygdrive/d/OS Practicals
$ ./assgn file open abc2.txt read
Welcome to program: ./assgn

Error in opening file.

[illegible] /cygdrive/d/OS Practicals
$ |
```

3. write



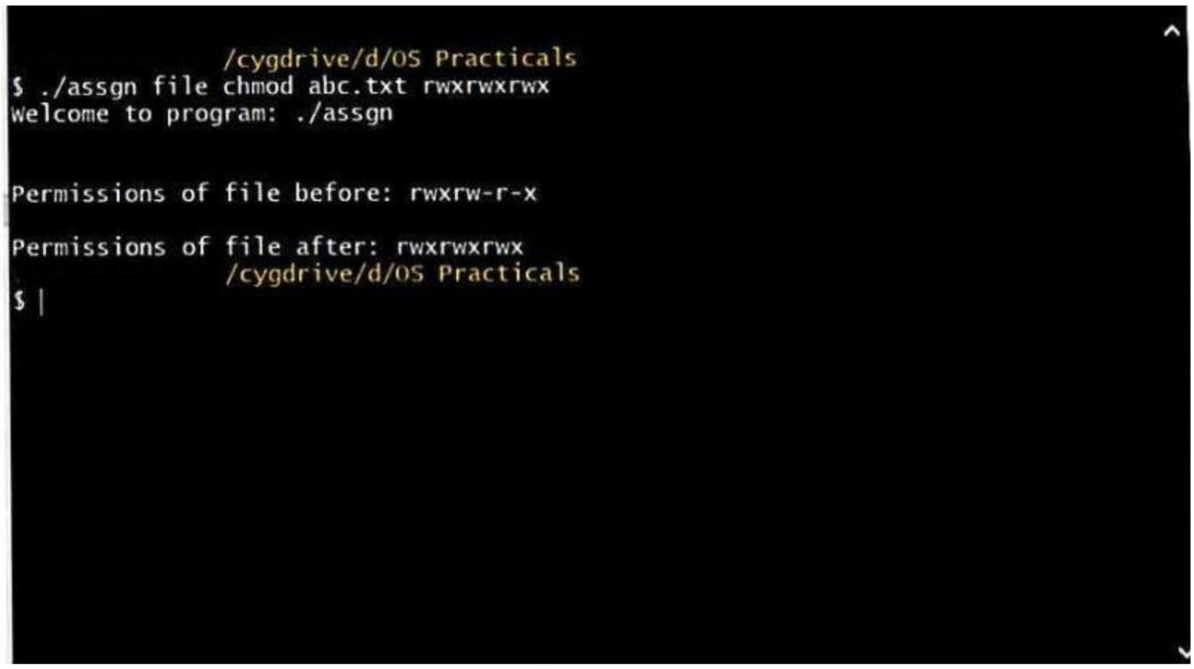
```
[redacted] /cygdrive/d/OS Practicals
$ ./assgn file write abc.txt 0 10
Welcome to program: ./assgn

Enter the data to be wriiten(press ';' to end input)
hi
world
people;


Total bytes written: 10

[redacted] /cygdrive/d/OS Practicals
$ ./assgn file write abc.txt 20 10 smile
Welcome to program: ./assgn



Total bytes written: 5

[redacted] /cygdrive/d/OS Practicals
$ |
```

4. read



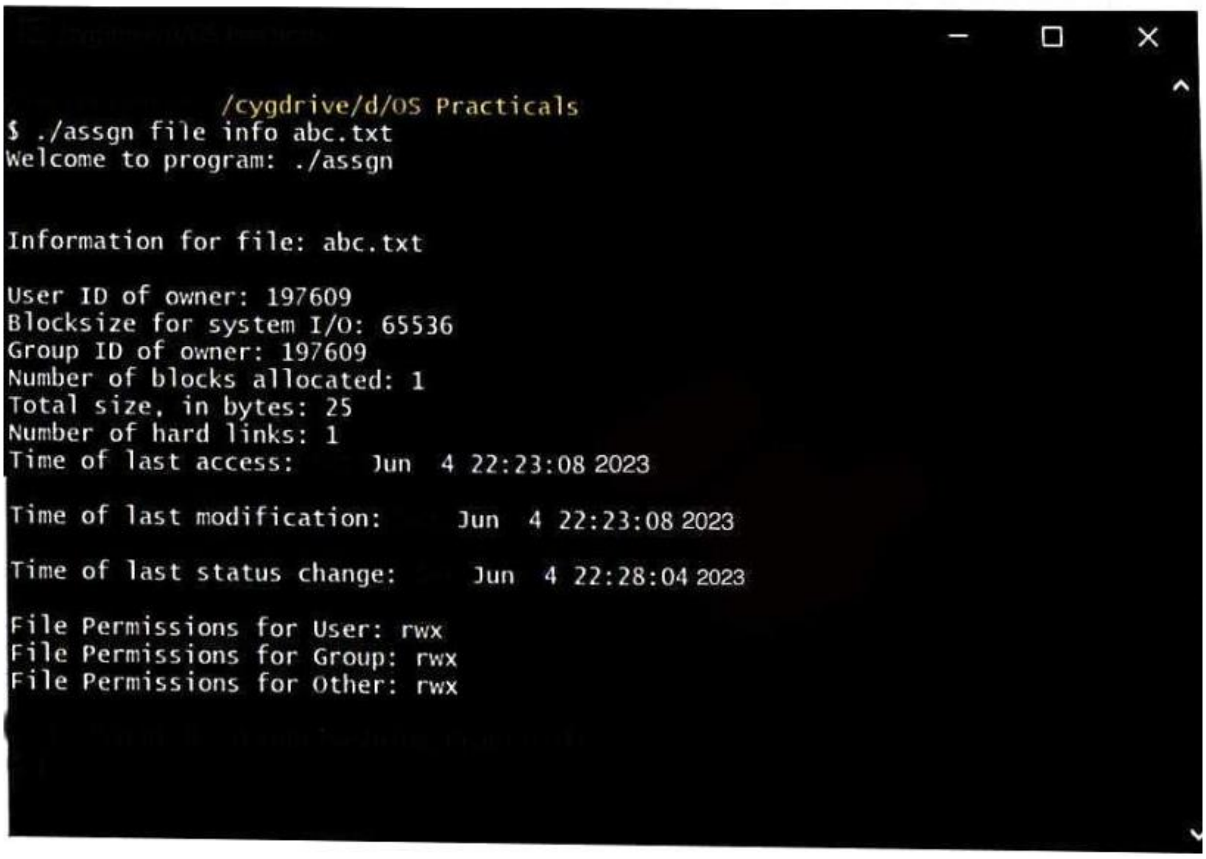
```
[redacted] /cygdrive/d/OS Practicals
$ ./assgn file read abc.txt 3 7
Welcome to program: ./assgn


The bytes read are: world
p

Total bytes read: 7

[redacted] /cygdrive/d/OS Practicals
$ |
```

5. chmod



```
                /cygdrive/d/OS Practicals
$ ./assgn file chmod abc.txt rwxrwxrwx
Welcome to program: ./assgn

Permissions of file before: rwxrw-r-x

Permissions of file after: rwxrwxrwx
                /cygdrive/d/OS Practicals
$ |
```

6. stat

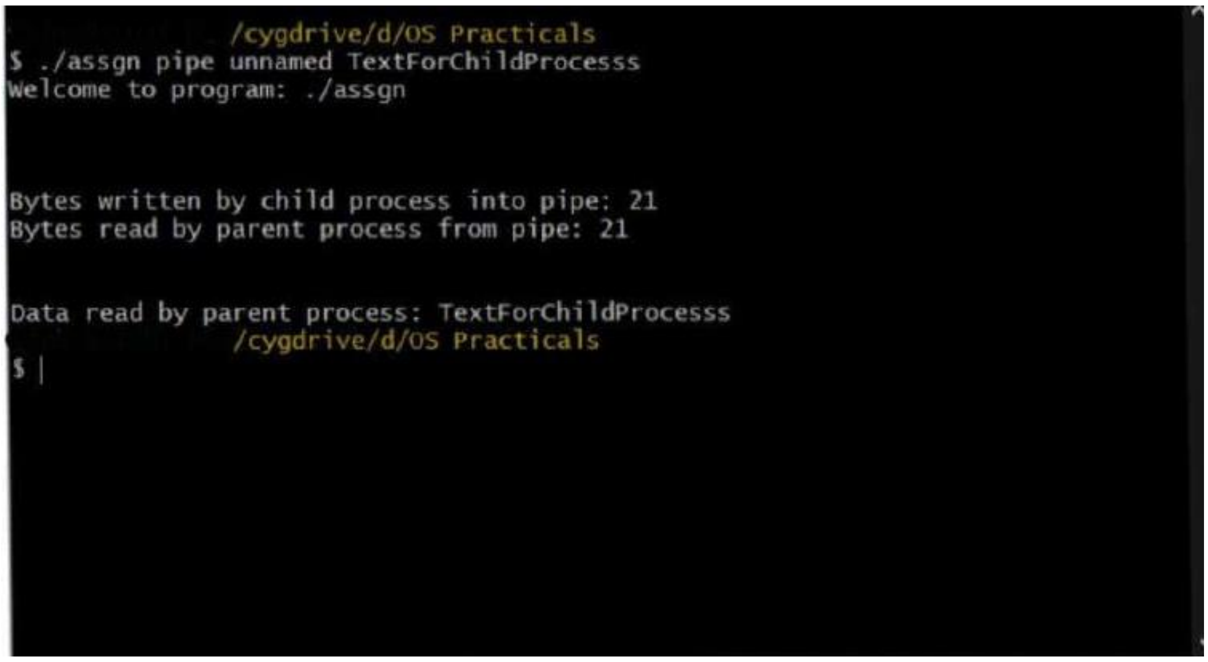


```
                /cygdrive/d/OS Practicals
$ ./assgn file info abc.txt
Welcome to program: ./assgn


Information for file: abc.txt

User ID of owner: 197609
Blocksize for system I/O: 65536
Group ID of owner: 197609
Number of blocks allocated: 1
Total size, in bytes: 25
Number of hard links: 1
Time of last access:     Jun  4 22:23:08 2023

Time of last modification:     Jun  4 22:23:08 2023

Time of last status change:     Jun  4 22:28:04 2023

File Permissions for User: rwx
File Permissions for Group: rwx
File Permissions for Other: rwx
```

PIPES

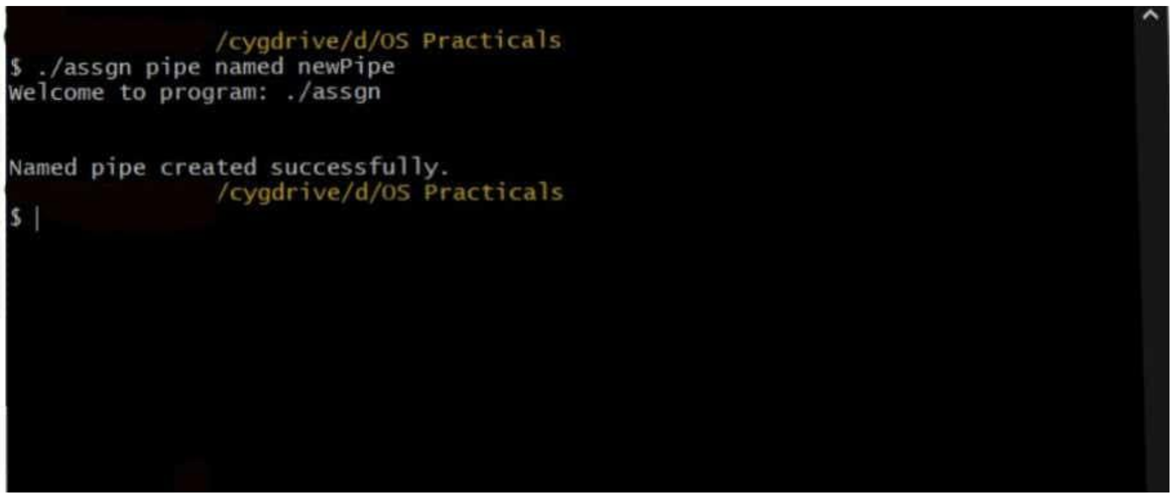
1. pipe

Since unnamed pipes can’t be used outside process so here an unnamed pipe is created and then used by child and parent process to write and read respectively.



```
[redacted] /cygdrive/d/OS Practicals
$ ./assgn pipe unnamed TextForChildProcesss
Welcome to program: ./assgn


Bytes written by child process into pipe: 21
Bytes read by parent process from pipe: 21


Data read by parent process: TextForChildProcesss
[redacted] /cygdrive/d/OS Practicals
$ |
```

2. mknod



```
[redacted] /cygdrive/d/OS Practicals
$ ./assgn pipe named newPipe
Welcome to program: ./assgn


Named pipe created successfully.
[redacted] /cygdrive/d/OS Practicals
$ |
```