Name: Ritika Mohanty

Roll Number: 44

ADVANCED OPERATING SYSTEM ASSIGNMENT

**Documentation for the Code:**

This code is a C program that provides several functionalities related to file operations using system calls. It includes functions to create a file, read data from a file, write data to a file, copy the content of a source file to a destination file using a pipe, display file statistics, and create a named pipe for communication between two processes.

**Functions:**

1. createFile:

- Description: Creates a file with specified permissions.

- Function Signature: void createFile(char\* fileName, int permissions)

- Inputs:

- fileName: Name of the file to be created.

- permissions: Permissions for the file in octal format (e.g., 0666).

2. createPipe:

- Description: Creates a named pipe with specified permissions.

- Function Signature: void createPipe(char\* filename, int permissions)

- Inputs:

- filename: Name of the named pipe to be created.

- permissions: Permissions for the named pipe in octal format (e.g., 0666).

3. readFileData:

- Description: Reads data from a file starting from a specified offset and of a specified size.

- Function Signature: void readFileData(char\* fileName, int offset, int size)

- Inputs:

- fileName: Name of the file to read data from.

- offset: Offset in bytes from where to start reading.

- size: Number of bytes to read.

4. writeFileData:

- Description: Writes data to a file starting from a specified offset.

- Function Signature: void writeFileData(char\* fileName, int offset, char\* data)

- Inputs:

- fileName: Name of the file to write data to.

- offset: Offset in bytes from where to start writing.

- data: Data to be written to the file.

5. copyUsingPipe:

- Description: Copies the content of a source file to a destination file using a pipe.

- Function Signature: void copyUsingPipe(char\* sourceFile, char\* destinationFile)

- Inputs:

- sourceFile: Name of the source file to be copied.

- destinationFile: Name of the destination file to copy the content to.

6. displayFileStats:

- Description: Displays file statistics including owner, permissions, inode, and timestamps.

- Function Signature: void displayFileStats(char\* fileName)

- Inputs:

- fileName: Name of the file to display statistics for.

7. createNamedPipeForComm:

- Description: Creates a named pipe for communication between two processes.

- Function Signature: void createNamedPipeForComm(char\* pipeName, int mode, char\* msg)

- Inputs:

- pipeName: Name of the named pipe for communication.

- mode: Mode of the pipe (O\_RDONLY or O\_WRONLY).

- msg: Message to be written to the pipe (only for writing mode).

**Libraries Used**

The code uses several library functions to perform file operations and inter-process communication:

* stdio.h: It is used for input and output operations, such as printf and scanf.
* stdlib.h: It provides general utilities, including exit and atoi.
* unistd.h: It includes API functions like fork, pipe, open, read, write, `lseek.
* lseek: It is used to move the file offset to a specified position within a file.
* fcntl.h: It provides various file control options, such as O\_RDONLY and O\_WRONLY, used for opening pipes in read or write mode.
* sys/types.h: It includes definitions for various types used in system calls, such as ssize\_t (signed size type).
* sys/stat.h: It contains structures and functions used for file status and information, such as stat for retrieving file statistics.

**Library Functions and their Usage:**

1. printf:

- Usage: printf("format string", arguments)

- Description: Prints formatted output to the standard output (console).

2. scanf:

- Usage: scanf("format string", arguments)

- Description: Reads formatted input from the standard input (console).

3. exit:

- Usage: exit(status)

- Description: Terminates the program execution with the specified status code.

4. mknod:

- Usage: mknod(filename, mode, dev)

- Description: Creates a special file (in this case, a named pipe) with the specified mode and device number.

5. open:

- Usage: open(filename, flags, mode)

- Description: Opens the specified file with the given flags and mode. Returns a file descriptor that can be used for further file operations.

6. read:

- Usage: read(fd, buffer, count)

- Description: Reads data from the file associated with the file descriptor (fd) into the buffer. It reads a maximum of count bytes.

7. write:

- Usage: write(fd, buffer, count)

- Description: Writes data from the buffer to the file associated with the file descriptor (fd). It writes a maximum of count bytes.

8. lseek:

- Usage: lseek(fd, offset, whence)

- Description: Sets the file offset for the file descriptor (fd) based on the specified offset and whence. It allows moving the file pointer to a specific position in the file.

9. close:

- Usage: close(fd)

- Description: Closes the file associated with the file descriptor (fd).

10. stat:

- Usage: stat(filename, stat\_buffer)

- Description: Retrieves information about the specified file and stores it in the stat structure pointed to by stat\_buffer. The stat structure contains details such as file size, permissions, owner, timestamps, etc.

**Input and Output Formats:**

// To create a file with permissions 0666

Input: ./a.out createFile ~/myfile.txt 0666

Output: File created successfully.

// To create a pipe with permissions 0666

Input: ./a.out copyUsingPipe file1.txt file2.txt

Output: File data from file1.txt copied to file file2.txt sucessfully!!

// Read data from the file

Input: ./a.out read file1.txt 5 100

Output: "File data read from file: <data read>"

// Write data to the file

Input Format: ./program write file1.txt 4 "Lorem Ipsum"

Data is now written in file file1.txt

// Display file statistics of a file

Input Format: ./a.out displayStats file1.txt

Output: File Statistics Information for file 'file1.txt':

Owner: 1000

Permissions: 100777

Inode: 178173660257910177

Last Access Time: Tue Jul 4 00:24:15 2023

Last Modification Time: Tue Jul 4 00:24:15 2023

Last Status Change Time: Tue Jul 4 00:24:15 2023

// Copy data from source file to destination file using a pipe

Input: ./a.out copyUsingPipe file1.txt file2.txt

Output: File data from file1.txt copied to file file2.txt sucessfully!!

// Communicate between processes using the named pipe

Input (Reading Process): ./a.out commUsingNamedPipe ~/myPipe 0

Output: Data is now written to the named pipe!

Input (Writing Process): ./a.out commUsingNamedPipe ~/myPipe 1 "This is a message for reading process"

Output: Data read from the pipe: This is a message for reading process

**File Name: createFile.h**

#ifndef SYSCALL\_CREATE\_FILE

#define SYSCALL\_CREATE\_FILE

#include <sys/types.h>

void createFile(char \*filename, int permissions);

#endif

**File Name: createFile.c**

#include "createFile.h"

#include <stdio.h>  // cfor input and output operations ie printf scanf

#include <stdlib.h>  // general utilities: exit and atoi

#include <unistd.h>  // API functions :fork, pipe, open, read, write, lseek close

#include <fcntl.h>  // defines constants: O\_RDONLY and O\_WRONLY O\_CREAT  O\_TRUNC

#include <sys/types.h> // data types used in system calls : mode\_t and off\_t

#include <sys/stat.h> //unmask

// Function to create a file with specified permissions

// Input Format: ./a.out createFile ~/myfile.txt 0666

// Output: "File created successfully."

// 0777 always because it is mounted file system

void createFile(char \*filename, int permissions)

{

    umask(0);

    // O\_WRONLY flag: open a file in write-only mode

    // O\_CREAT: if the file doesn't exist, create it

    // O\_EXCL: and if it does exist, return error

    // int fd = open(filename, O\_WRONLY | O\_CREAT | O\_EXCL, permissions);      // slashes away w and x perimissions from others and group

    int fd = creat(filename, permissions);

    if (fd == -1) {

        printf("Error creating file");

        exit(1);

    }

    printf("File created successfully.\n");

    close(fd);

}

**File Name: createPipe.h**

#ifndef CREATE\_PIPE\_H

#define CREATE\_PIPE\_H

#include <sys/types.h>

void createPipe(char \*pipeName, int permissions);

#endif

**File Name: createFile.c**

#include "createPipe.h"

#include <stdio.h>  // cfor input and output operations ie printf scanf

#include <stdlib.h>  // general utilities: exit and atoi

#include <unistd.h>  // API functions :fork, pipe, open, read, write, lseek close

#include <fcntl.h>  // defines constants: O\_RDONLY and O\_WRONLY O\_CREAT  O\_TRUNC

#include <sys/types.h> // data types used in system calls : mode\_t and off\_t

#include <sys/stat.h> //unmask

// Function to create a file with specified permissions

// Input Format: ./a.out createPipe ~/myPipe 0666

// Output: "Pipe created successfully."

void createPipe(char \*filename, int permissions)

{

    // printf("%o\n", permissions);

    int fd = mknod(filename,  S\_IFIFO | permissions, 0); // 0 for device no.

    if (fd == -1) {

        perror("mknod");

        printf("Error creating Pipe\n");

        exit(1);

    }

    printf("Pipe created successfully.\n");

    close(fd);

}

**File Name: readFileData.h**

#ifndef READ\_FILE\_H

#define READ\_FILE\_H

#include <sys/types.h>

void readFileData(char\* fileName, int offset, int size);

#endif

**File Name: readFileData.c**

#include "readFileData.h"

#include <stdio.h>  // cfor input and output operations ie printf scanf

#include <stdlib.h>  // general utilities: exit and atoi

#include <unistd.h>  // API functions :fork, pipe, open, read, write, lseek close

#include <fcntl.h>  // defines constants: O\_RDONLY and O\_WRONLY O\_CREAT  O\_TRUNC

#include <sys/types.h> // data types used in system calls : mode\_t and off\_t

#include <sys/stat.h> // retrieving information about file status and file permissions: stat and chmod.

// Function to read data from a file starting from a specified offset and of a specified size

// Input Format: ./a.out read file1.txt 5 100

// Output: "File data read from file: <data read>"

void readFileData(char\* fileName, int offset, int size) {

    // O\_RDONLY: open file in read only mode

    int fd = open(fileName, O\_RDONLY);

    if (fd == -1) {

        perror("Error in opening file: File doesnot exit. Exiting... ");

        exit(1);

    }

    // SEEK\_SET: relative to the beginning of the file

    // offset: the byte no. read pointer should move to

    if (lseek(fd, offset, SEEK\_SET) == -1) {

        perror("Error seeking file: offset claculation might have exceeded the limits of the file size. Exiting ...");

        close(fd);

        exit(1);                    // signifies that the application failed

    }

    char\* buffer = (char\*)malloc(size+1);     // allocate memory of size: amount to read

    int bytesRead = read(fd, buffer, size);  // read data from a file

    if (bytesRead == -1) {

        perror("Error reading from file");

        close(fd);

        free(buffer);   // free dynamically allocated memory

        exit(1);

    }

    buffer[bytesRead] = '\0';

    printf("File data read from file: %s\n", buffer);

    close(fd);

    free(buffer);  // free dynamically allocated memory

}

**File Name: writeFileData.h**

#ifndef WRITE\_FILE\_H

#define WRITE\_FILE\_H

#include <sys/types.h>

void writeFileData(char\* fileName, int offset, char\* data);

#endif

**File Name: writeFileData.c**

#include "writeFileData.h"

#include <stdio.h>  // cfor input and output operations ie printf scanf

#include <stdlib.h>  // general utilities: exit and atoi

#include <unistd.h>  // API functions :fork, pipe, open, read, write, lseek close

#include <fcntl.h>  // defines constants: O\_RDONLY and O\_WRONLY O\_CREAT  O\_TRUNC

#include <sys/types.h> // data types used in system calls : mode\_t and off\_t

#include <string.h>

// Function to write data to a file starting from a specified offset

// Input Format: ./program write file1.txt 4 "Lorem Ipsum"

// Data is now written in file file1.txt

void writeFileData(char\* fileName, int offset, char\* data) {

    // open file in write only mode

    int fd = open(fileName, O\_WRONLY);

    if (fd == -1) {

        perror("Error in opening file: File doesnot exit. Exiting...");

        exit(1);

    }

    // SEEK\_SET: relative to the beginning of the file

    // offset: the byte no. read pointer should move to

    if (lseek(fd, offset, SEEK\_SET) == -1) {

        perror("Error seeking file: offset claculation might have exceeded the limits of the file size. Exiting ...");

        close(fd);

        exit(1);

    }

    int bytesWritten = write(fd, data, strlen(data));

    if (bytesWritten == -1) {

        perror("Error in writing to file. Exiting ... ");

        close(fd);

        exit(1);

    }

    printf("Data is now written in file %s \n",  fileName);

    close(fd);

}

**File Name: displayFileStats.h**

#ifndef FILES\_STAT\_H

#define FILES\_STAT\_H

void displayFileStats(char \*filename);

#endif

**File Name: displayFileStats.c**

#include "displayFileStats.h"

#include <stdio.h>  // cfor input and output operations ie printf scanf

#include <stdlib.h>  // general utilities: exit and atoi

#include <sys/types.h> // data types used in system calls : mode\_t and off\_t

#include <sys/stat.h> // retrieving information about file status and file permissions: stat and chmod.

#include <time.h> //converting a time value to a string representation

// Function to display file statistics including owner, permissions, inode, and timestamps

// Input Format: ./a.out displayStats file1.txt

// Output:

// File Statistics Information for file 'file1.txt':

// Owner: 1000

// Permissions: 100777

// Inode: 178173660257910177

// Last Access Time: Tue Jul  4 00:24:15 2023

// Last Modification Time: Tue Jul  4 00:24:15 2023

// Last Status Change Time: Tue Jul  4 00:24:15 2023

void displayFileStats(char\* fileName) {

    struct stat fileStats;

    if (stat(fileName, &fileStats) == -1) {

        perror("Error in getting file stats");

        exit(1);

    }

    printf("File Statistics Information for file '%s':\n", fileName);

    printf("Owner: %d\n", fileStats.st\_uid);

    printf("Permissions: %o\n", fileStats.st\_mode);

    printf("Inode: %lu\n", fileStats.st\_ino);

    printf("Last Access Time: %s", ctime(&fileStats.st\_atime));

    printf("Last Modification Time: %s", ctime(&fileStats.st\_mtime));

    printf("Last Status Change Time: %s", ctime(&fileStats.st\_ctime));

}

**File Name: copyUsingPipe.h**

#ifndef COPY\_FILE\_H

#define COPY\_FILE\_H

void copyUsingPipe(char \*sourceFile, char \*destinationFile);

#endif

**FileName: copyUsingPipe.c**

#include "copyUsingPipe.h"

#include <stdio.h>  // cfor input and output operations ie printf scanf

#include <stdlib.h>  // general utilities: exit and atoi

#include <unistd.h>  // API functions :fork, pipe, open, read, write, lseek close

#include <fcntl.h>  // defines constants: O\_RDONLY and O\_WRONLY O\_CREAT  O\_TRUNC

#include <sys/types.h> // data types used in system calls : mode\_t and off\_t

#include <sys/wait.h>  // For wait function

// Function to copy the content of a source file to a destination file using a pipe

// Input: ./a.out copyUsingPipe file1.txt file2.txt

// Output: File data from file1.txt copied to file file2.txt sucessfully!!

void copyUsingPipe(char\* sourceFile, char\* destinationFile) {

    // stores the 2 file descriptors: pipefd[0]: for reading and pipefd[1]: for writing

    int pipefd[2];

    // attempting to create pipe

    if (pipe(pipefd) == -1) {

        perror("Error in creating pipe. Exiting...");

        exit(1);                            // signifies that the application failed

    }

    /\*

    child process: read from sourceFile and write to pipe(pipefd[1])

    parent process: read from pipe(pipefd[0]) and write to destinationFile

    \*/

    int pid = fork();

    if (pid == -1) {

        perror("Error in fork() command.");

        exit(1);

    }

    else if (pid == 0) {

        // Child Process : performs writing to pipe

        close(pipefd[0]);  // Close read pipefd

        int sourcefd = open(sourceFile, O\_RDONLY);

        if (sourcefd == -1) {

            perror("Error in opening source file. Exiting ...");

            exit(1);

        }

        char buffer[1024];

        int bytesRead;

        while (( bytesRead = read(sourcefd, buffer, 1024) ) > 0) {

            int bytesWritten = write(pipefd[1], buffer, bytesRead);

            if (bytesWritten == -1) {

                perror("Error in writing to pipe. Exiting ..");

                close(sourcefd);

                close(pipefd[1]);

                exit(1);

            }

        }

        close(sourcefd);

        close(pipefd[1]);  // Close write pipefd

        exit(0);

    }

    else {

        // Parent process : reading from pipe

        close(pipefd[1]);  // Close unused write end

        int destinationfd = open(destinationFile, O\_WRONLY | O\_CREAT | O\_TRUNC, S\_IRUSR | S\_IWUSR);

        if (destinationfd == -1) {

            perror("Error in opening destination file");

            exit(1);

        }

        char buffer[1024];

        int bytesRead;

        while ((bytesRead = read(pipefd[0], buffer, sizeof(buffer))) > 0) {

            int bytesWritten = write(destinationfd, buffer, bytesRead);

            if (bytesWritten == -1) {

                perror("Error writing to destination file");

                close(destinationfd);

                close(pipefd[0]);

                exit(1);

            }

        }

        close(destinationfd);

        close(pipefd[0]);

        wait(NULL);  // Wait for the child process to finish

    }

    printf("File data from %s copied to file %s sucessfully!!\n", sourceFile, destinationFile);

}

**File Name: createNamedForComm.h**

#ifndef CREATE\_NAMED\_FILE\_H

#define CREATE\_NAMED\_FILE\_H

#include <sys/types.h>

void createNamedPipeForComm(char\* pipeName, int mode, char \*msg);

#endif

**File Name: createNamedForComm.c**

#include "createNamedPipeForComm.h"

#include <stdio.h>  // cfor input and output operations ie printf scanf

#include <stdlib.h>  // general utilities: exit and atoi

#include <unistd.h>  // API functions :fork, pipe, open, read, write, lseek close

#include <fcntl.h>  // defines constants: O\_RDONLY and O\_WRONLY O\_CREAT  O\_TRUNC

#include <string.h>

// Function to create a named pipe for communication between two processes

// Input (Reading Process): ./a.out commUsingNamedPipe ~/myPipe 0

// Output: Data is now written to the named pipe!

// Input(Writing Process): ./a.out commUsingNamedPipe ~/myPipe 1 "This is a message for reading process"

// Output: Data read from the pipe: This is a message for reading process

void createNamedPipeForComm(char\* pipeName, int mode, char\* msg) {

    if(mode == O\_RDONLY){

        // open named pipe for reading

        int fd = open(pipeName, O\_RDONLY);

        if (fd == -1){

            printf("Error opening named pipe for reading. Exiting...\n");

            exit(1);

        }

        char buffer[1024];

        int bytesRead;

        bytesRead = read(fd, buffer, 1024);

        // while(bytesRead = read(fd, buffer, 1024)){

            // printf("Data read from the pipe: %s\n", buffer);

        // }

        if (bytesRead == -1) {

            perror("Error in reading from named pipe. Exiting...");

            close(fd);

            exit(1);

        }

        printf("Data read from the pipe: %s\n", buffer);

        close(fd);

    }

    else if(mode == O\_WRONLY){

        int fd = open(pipeName, O\_WRONLY);

        if (fd == -1) {

            perror("Error in openeing named pipe. Exiting ...");

            exit(1);

        }

        int bytesWritten = write(fd, msg, strlen(msg));

        if (bytesWritten == -1) {

            perror("Error in writing to named pipe. Exiting ...");

            close(fd);

            exit(1);

        }

        printf("Data is now written to the named pipe!");

        close(fd);

    }

    else{

        printf("Invalid pipe mode specified.\n");

        exit(1);

    }

}

**File Name: main\_file.c**

#include <stdio.h>  // cfor input and output operations ie printf scanf

#include <stdlib.h>  // general utilities: exit and atoi

#include <unistd.h>  // API functions :fork, pipe, open, read, write, lseek close

#include <fcntl.h>  // defines constants: O\_RDONLY and O\_WRONLY O\_CREAT  O\_TRUNC

#include <sys/types.h> // data types used in system calls : mode\_t and off\_t

#include <string.h> // for strcmp

// Task: create a file

#include "createFile.h"

// Task: create a named pipe

#include "createPipe.h"

// Task: read data from a file

#include "readFileData.h"

// Task: write data from a file

#include "writeFileData.h"

// Task: display status of file

#include "displayFileStats.h"

// Task: Create an unnamed pipe designed for copying a file's content

#include "copyUsingPipe.h"

// Task: Create a named pipe to help communicate between two processes

#include "createNamedPipeForComm.h"

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

    if (argc < 2) {

        printf("Error, insufficient parameters, please enter parameters in the following format: %s createFile <task> [arguments]\n", argv[0]);

        return 0;

    }

    char\* task = argv[1];

    if (strcmp(task, "createFile") == 0){   // Create file

        if (argc < 4) {

            printf("Error, insufficient parameters. Format expected: %s createFile <file name> <permissions>\n", argv[0]);

            return 0;

        }

        // strtol:  to convert the permission provided as a string to an integer value

        char\* fileName = argv[2];

        int permissions = strtol(argv[3], NULL, 8); // 8: Number Base (Octal-format'0666')

        createFile(fileName, permissions);

    }

    if (strcmp(task, "createPipe") == 0){   // Create pipe

        if (argc < 4) {

            printf("Error, insufficient parameters. Format expected: %s createPipe <pipe\_name> <permissions>\n", argv[0]);

            return 0;

        }

        // strtol:  to convert the permission provided as a string to an integer value

        char\* pipeName = argv[2];

        int permissions = strtol(argv[3], NULL, 8); // 8: Number Base (Octal-format'0666')

        createPipe(pipeName, permissions);

    }

    else if(strcmp(task, "read") == 0){

        if (argc < 5) {

            printf("Error, insufficient parameters. Format expected: %s read <file name> <offset> <amt2read>\n", argv[0]);

            return 0;

        }

        char\* fileName = argv[2];

        int offset = atoi(argv[3]);

        int amt2read = atoi(argv[4]);

        readFileData(fileName, offset, amt2read);

    }

    else if(strcmp(argv[1], "write") == 0){

        if (argc < 5) {

            printf("Error, insufficient parameters. Format expected: %s write <file name> <offset> <amt2write> <data>\n", argv[0]);

            return 0;

        }

        char\* fileName = argv[2];

        int offset = atoi(argv[3]);

        char\* data = argv[4];

        writeFileData(fileName, offset, data);

    }

    else if(strcmp(argv[1], "displayStats") == 0){

        if (argc < 3) {

            printf("Error, insufficient parameters. Format expected: %s displayStats <file name> \n", argv[0]);

            return 0;

        }

        char\* fileName = argv[2];

        displayFileStats(fileName);

    }

    else if(strcmp(argv[1], "copyUsingPipe") == 0){

        if (argc < 4) {

            printf("Error, insufficient parameters. Format expected: %s copyUsingPipe <sourceFile> <destinationFile>\n", argv[0]);

            return 0;

        }

        char\* sourceFile = argv[2];

        char\* destinationFile = argv[3];

        copyUsingPipe(sourceFile, destinationFile);

    }

    else if(strcmp(argv[1], "commUsingNamedPipe") == 0){

        if (argc < 4) {

            printf("Error, insufficient parameters. Format expected:  %s commUsingNamedPipe <pipe name> <mode> //ifwritemode<msg>\n", argv[0]);

            return 0;

        }

            char\* pipeName = argv[2];

            int mode = atoi(argv[3]);

            char\* msg = "";

            if(mode == O\_WRONLY){

                if(argc <5){

                    printf("Error, insufficient parameters. Format expected:  %s commUsingNamedPipe <pipe name> <mode> <msg>\n", argv[0]);

                }

                msg = argv[4];

            }

            createNamedPipeForComm(pipeName, mode, msg);

    }

    return 0;

}