Laboratory 2

Title of the Laboratory Exercise: Programs using file management system calls

1. Introduction and Purpose of Experiment

A system call is a programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. There are different types of system calls developed for various purposes. They are mainly classified as process management, file management, directory management. By solving the problems students will be able to apply file management system calls

Aim and Objectives

Aim

- To develop programs involving file management system calls

Objectives

At the end of this lab, the student will be able to

- Use different file management system calls
- Apply different system calls wherever required
- Create C programs using file management system calls

2. Experimental Procedure

i. Analyse the problem statement
ii. Design an algorithm for the given problem statement and develop a flowchart/pseudo-code
iii. Implement the algorithm in C language
iv. Compile the C program
v. Test the implemented program
vi. Document the Results
vii. Analyse and discuss the outcomes of your experiment

3. Questions

Implement the following command in C

Implement copy command (cp) to copy a file content to other file using file management system calls

4. Calculations/Computations/Algorithms

copy_file(source, destination)

1. open(source, READ_ONLY)
2. open(destination, WRITE_ONLY | CREATE)
3. if either of it fails, return FAILURE
4. create an empty buffer array
5. read 1024 bytes of source and write strlen(source) to destination
6. repeat until EOF is reached
7. close source and destination file_descriptors

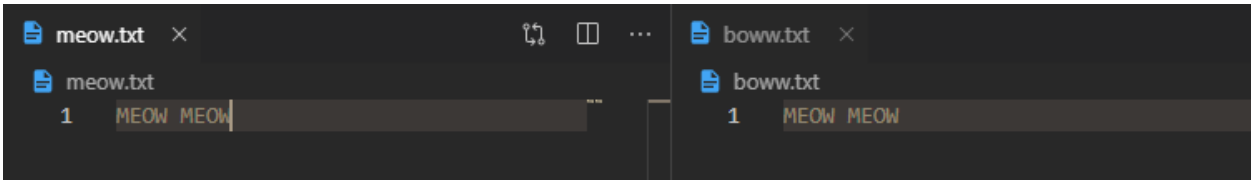
5. Presentation of Results

main.c

```
#include <iostream>

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

#include <string.h>

void help(char* name);

int main(int argc, char* argv[]) {
    if (argc != 3) {
        help(argv[0]);
        return EXIT_FAILURE;
    }

    // open the source file
    int source_fd;
    if ((source_fd = open(argv[1], O_RDONLY)) < 0) {
        std::cout << argv[0] << " cannot open : " << argv[1] << std::endl;

        exit(EXIT_FAILURE);
```

```cpp
    }

    // open the destination file
    int dest_fd;
    if ((dest_fd = open(argv[2], O_WRONLY | O_CREAT)) < 0) {
        std::cout << argv[0] << " cannot open : " << argv[2] << std::endl;

        close(source_fd);

        exit(EXIT_FAILURE);
    }

    // buffer
    char buffer[1024];

    while (read(source_fd, buffer, 1024)) {
        using namespace std;
        cout << buffer;

        write(dest_fd, buffer, strlen(buffer));

        memset(buffer, 0, sizeof(buffer));
    }

    // close the files
    close(source_fd);
    close(dest_fd);
}

void help(char* name) {
    std::cout << "USAGE : " << name << " <source_file> <dest_file>" << std::endl;
    std::cout << "Example : " << name << " meow.txt boww.txt" << std::endl;
}
```

```
shadowleaf@SHADOWLEAF-ROG /mnt/d/University-Work/University-Work-SEM-05/OS-Lab/Lab02
build/Lab02
USAGE : build/Lab02 <source_file> <dest_file>
Example : build/Lab02 meow.txt boww.txt
shadowleaf@SHADOWLEAF-ROG /mnt/d/University-Work/University-Work-SEM-05/OS-Lab/Lab02
build/Lab02 meow.txt boww.txt
MEOW MEOW
shadowleaf@SHADOWLEAF-ROG /mnt/d/University-Work/University-Work-SEM-05/OS-Lab/Lab02
```



Explanation:

Here the content of file meow.txt is copied to boww.txt, initially the boww.txt file does not exist, hence the function defined in the program creates the files, reads the file meow.txt and writes the content until EOF bit is reached. If the program encounters any error during opening of either the source or destination file, it displays the error and closes the source/destination file as necessary.

If the destination file exists and already has some content, then it is overwritten with the source file content.

6. Analysis and Discussions

open, openat, creat - open and possibly create a file

`#include <sys/types.h>`
`#include <sys/stat.h>`
`#include <fcntl.h>`
 	
`int open(const char *pathname, int flags);`
`int open(const char *pathname, int flags, mode_t mode);`
 	
`int creat(const char *pathname, mode_t mode);`
 	
`int openat(int dirfd, const char *pathname, int flags);`
`int openat(int dirfd, const char *pathname, int flags, mode_t mode);`

Description: The **open**() system call opens the file specified by *pathname*. If the specified file does not exist, it may optionally (if **O_CREAT** is specified in *flags*) be created by **open**(). A call to **open**() creates a new *open file description*, an entry in the system-wide table of open files. The open file description records the file offset and the file status flags. A file descriptor is a reference to an open file description; this reference is unaffected if *pathname* is subsequently removed or modified to refer to a different file.

The argument *flags* must include one of the following *access modes*: **O_RDONLY**, **O_WRONLY**, or **O_RDWR**. These requests opening the file read-only, write-only, or read/write, respectively.

7. Conclusions

The *open*() function shall establish the connection between a file and a file descriptor. It shall create an open file description that refers to a file and a file descriptor that refers to that open file description. The file descriptor is used by other I/O functions to refer to that file. The *path* argument points to a pathname naming the file.

Upon successful completion, the function shall open the file and return a non-negative integer representing the lowest numbered unused file descriptor. Otherwise, -1 shall be returned and *errno* set to indicate the error. No files shall be created or modified if the function returns -1.

Some errors that may occur during the program execution are:

1. Search permission is denied on a component of the path prefix.
2. The file already exists and the permissions specified by oflag is denied.
3. The length of the path argument exceeds PATH_MAX, or the pathname is longer than NAME_MAX.
4. The named file resides on a READ_ONLY file system.
5. The named file is a character special file or a block special file.

8. Comments

1. Limitations of Experiments

The experiment is limited to mimicking the actions of cp command with only using open and close syscalls, this raises issues with some of the errors that may occur during the execution that needs to taken care of separately.

2. Limitations of Results

The implemented program assumes that the file does not contain null characters in the middle of the file, here the program will fail to copy the contents of the file properly.

3. Learning happened

The concept of syscalls for reading and writing files in Linux is learnt.

4. Recommendations

To further enhance the program, user permissions from the source files should be copied to the destination file.

File permissions of the source file needs to be checked properly and appropriate error messages should be shown to the user in case of errors.