SDM College of Engineering and Technology

Dhavalagiri, Dharwad-580 002. Karnataka State. India.
Email: principal@sdmcet.ac.in, cse.sdmcet@gmail.com
Ph: 0836-2447465/ 2448327 Fax: 0836-2464638 Website: sdmcet.ac.in

Department of
COMPUTER SCIENCE AND ENGINEERING

MINOR WORK REPORT

[22UCSC501-Database Management Systems]

Odd Semester: Sep-Jan-2025

Course Teacher: Prof. Rashmi Patil

2024- 2025

Submitted by
By

Varsha S.B
2SD22CS122
5th Semester B division

Table of Contents

Termwork-1: Write a C Program to study all file operations related SYSTEM CALLS supported by UNIX OS and C libraries for file operations.

Problem Statement: Write a C program to study all file operations related SYSTEM CALLS supported by UNIX OS and C libraries for file operations.

Theory:

Here's a simple C program that demonstrates basic file operations using system calls and C library functions for file operations in a UNIX environment:

Operations Covered:

1. Opening a file:

- open("sample.txt", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR) - Opens or creates a file with read/write permissions for the user.
- fopen("sample.txt", "r+") - Opens a file using the C library for reading and writing.

2. Writing to a file:

- write(fd, data, strlen(data)) - Writes the specified data to the file.
- fwrite(data, 1, strlen(data), fp) (not shown) - Could be used similarly with fopen().

3. Reading from a file:

- read(fd, buffer, bytesWritten) - Reads data from the file.
- fread(buffer, 1, bytesWritten, fp) - Reads data using the C library.

4. Repositioning file offset:

- lseek(fd, 0, SEEK_SET) - Moves the file offset to the beginning of the file.

5. Retrieving file status:

- stat("sample.txt", &fileStat) - Gets the file's metadata like size and owner UID.

6. Closing a file:

- close(fd) and fclose(fp) close the file for system calls and C library respectively.

7. Deleting a file:

- unlink("sample.txt") - Deletes the file from the file system.

Operations Covered:

1. open() and fopen() - Opening a file.
2. read() and fread() - Reading from a file.
3. write() and fwrite() - Writing to a file.
4. close() and fclose() - Closing a file.
5. lseek() - Repositioning read/write file offset.
6. stat() - Retrieving file status.
7. unlink() - Deleting a file.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>
#define BUFFER_SIZE 1024
int main() {
        int fd; // file descriptor for system call operations
        FILE *fp; // file pointer for C library operations
        char buffer[BUFFER_SIZE];
        ssize_t bytesRead, bytesWritten
struct stat fileStat;
   // 1. SYSTEM CALL: OPEN FILE using open()
```

```
fd = open("sample.txt", O_RDWR | O_CREAT, S_IRUSR | S_IWUSR);
if (fd == -1) {
perror("Error opening file with open()");
exit(EXIT_FAILURE); }
printf("File opened successfully using open(), file descriptor: %d\n", fd);
// C LIBRARY: OPEN FILE using fopen()
fp = fopen("sample.txt", "r+");
if (fp == NULL) {
perror("Error opening file with fopen()");
exit(EXIT_FAILURE); }
printf("File opened successfully using fopen()\n");
// 2. SYSTEM CALL: READ FILE using read()
bytesRead = read(fd, buffer, bytesWritten); // Reading the written data
if (bytesRead == -1) {
perror("Error reading from file with read()");
exit(EXIT_FAILURE); }
printf("Data read using read(): %s\n", buffer);
//  C LIBRARY: READ FILE using fread()
memset(buffer, 0, BUFFER_SIZE);  // Clear buffer
size_t freadBytes = fread(buffer, 1, bytesWritten, fp);
if (freadBytes < bytesWritten && ferror(fp)) {
perror("Error reading file with fread()");
exit(EXIT_FAILURE);}
printf("Data read using fread(): %s\n", buffer);
```

```
// 3. SYSTEM CALL: WRITE FILE using write()
    const char *data = "Hello, World! This is a sample text.\n";
    bytesWritten = write(fd, data, strlen(data));
    if (bytesWritten == -1) {
    perror("Error writing to file with write()");
    exit(EXIT_FAILURE);}
    printf("Data written successfully using write(). Bytes written: %zd\n", bytesWritten);
// 4. SYSTEM CALL: CLOSE FILE using close()
    if (close(fd) == -1) {
    perror("Error closing file with close()");
    exit(EXIT_FAILURE); }
    printf("File closed successfully using close()\n");
    //   C LIBRARY: CLOSE FILE using fclose()
    if (fclose(fp) != 0) {
    perror("Error closing file with fclose()");
    exit(EXIT_FAILURE); }
// 5. SYSTEM CALL: REPOSITION FILE OFFSET using lseek()
    off_t offset = lseek(fd, 0, SEEK_SET);
    if (offset == -1) {
    perror("Error with lseek()");
    exit(EXIT_FAILURE); }
  // 6. SYSTEM CALL: GET FILE STATUS using stat()
    if (stat("sample.txt", &fileStat) == -1) {
    perror("Error getting file status with stat()");
```

```
        exit(EXIT_FAILURE); }
        printf("File size: %ld bytes\n", fileStat.st_size);
        printf("File owner UID: %d\n", fileStat.st_uid);
    // 7. SYSTEM CALL: DELETE FILE using unlink()
        if (unlink("sample.txt") == -1) {
        perror("Error deleting file with unlink()");
        exit(EXIT_FAILURE); }
        printf("File deleted successfully using unlink()\n");
        return 0;
}
```

Sample input and output:

Input:

There is no user input, as the program creates, reads, and writes the file internally.

Sample Output:

File opened successfully using open(), file descriptor: 3

Data written successfully using write(). Bytes written: 35

Data read using read(): Hello, World! This is a sample text.

File size: 35 bytes

File owner UID: 1000

File closed successfully using close()

File opened successfully using fopen()

Data read using fread(): Hello, World! This is a sample text.

File closed successfully using fclose()

File deleted successfully using unlink()

References:

1) The UNIX Programming Environment, Brian W.Kernighan and Rob Pike.
2) Unix and Linux System Administration Handbook, Evi Nemeth,Garth Snyder, Trent R.Hein, Ben Whaley, Dan Mackin.

Termwork-2: Write a C program to demonstrate indexing and associated operations.

Problem Statement:

Write a C program to demonstrate indexing and associated operations.

Theory:

Below is a C program that demonstrates indexing and associated operations using an array. The program will include functions to perform operations like:

- Inserting an element at a specific index.
- Deleting an element at a specific index.
- Searching for an element by value.
- Displaying the contents of the array.

Design:

To demonstrate indexing and associated operations in data structures (like arrays, linked lists, or databases), you can design a simple yet illustrative system. Here's an outline:

1. Array-Based Design

Array: A list of elements with indices starting from 0.

Indexing: Access elements by their position in the array.

2. Linked List Design

Nodes: Each node contains data and a reference to the next node.

Indexing: Instead of direct indexing like arrays, linked lists require traversal to reach a specific index.

3. Database Indexing Design

Records: Stored data rows.

Index: A separate structure, like a B-tree or hash table, that speeds up the search for records based on a key.

Program:

```
#include <stdio.h>
#define MAX_SIZE 100  // Maximum size of the array
// Function prototypes
void insert(int arr[], int *n, int element, int index);
void delete(int arr[], int *n, int index);
int search(int arr[], int n, int element);
void display(int arr[], int n);
int main() {
        int arr[MAX_SIZE];
        int n, choice, element, index;
        // Input number of elements
        printf("Enter number of elements in the array: ");
   scanf("%d", &n);
// Input elements of the array
        printf("Enter the elements of the array:\n");
        for (int i = 0; i < n; i++) {
      printf("Element %d: ", i);
      scanf("%d", &arr[i]); }
                while (1) {
        // Menu for operations
      printf("\nChoose an operation:\n");
        printf("1. Insert element\n");
        printf("2. Delete element\n");
```

```
    printf("3. Search element\n");
    printf("4. Display array\n");
    printf("5. Exit\n");
printf("Enter your choice: ");
scanf("%d", &choice);
    switch (choice) {
    case 1:
            // Insert operation
        printf("Enter the element to insert: ");
        scanf("%d", &element);
        printf("Enter the index where the element should be inserted: ");
        scanf("%d", &index);
        insert(arr, &n, element, index);
            break;
          case 2:
            // Delete operation
        printf("Enter the index of the element to delete: ");
        scanf("%d", &index);
    delete(arr, &n, index);
            break;
    case 3:
            // Search operation
        printf("Enter the element to search: ");
        scanf("%d", &element);
```

```
            index = search(arr, n, element);
            if (index != -1) {
            printf("Element found at index %d\n", index);
            } else {
            printf("Element not found in the array\n"); }
            break;
        case 4:
            // Display array
            display(arr, n);
            break;
        case 5:
            printf("Exiting program.\n");
            return 0;
        default:
            printf("Invalid choice! Please choose a valid operation.\n");}}
return 0; }
// Function to insert an element at a specific index
void insert(int arr[], int *n, int element, int index) {
        if (index < 0 || index > *n) {
      printf("Invalid index! Insertion failed.\n");
        return; }
        // Shift elements to the right to make space
        for (int i = *n; i > index; i--) {
        arr[i] = arr[i - 1];
```

```
        }
        // Insert the element
        arr[index] = element;
        (*n)++;
    printf("Element inserted successfully.\n");
}
// Function to delete an element at a specific index
void delete(int arr[], int *n, int index) {
        if (index < 0 || index >= *n) {
       printf("Invalid index! Deletion failed.\n");
        return;}
        // Shift elements to the left to fill the gap
        for (int i = index; i < *n - 1; i++) {
        arr[i] = arr[i + 1];
        }
 (*n)--;
   printf("Element deleted successfully.\n"); }
// Function to search for an element by value
int search(int arr[], int n, int element) {
        for (int i = 0; i < n; i++) {
        if (arr[i] == element) {
        return i;  // Return index if element is found
        }
}
```

```
        return -1;  // Return -1 if element is not found
}
// Function to display the elements of the array
void display(int arr[], int n) {
        printf("Array elements: ");
        for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
}
```

Sample input and output:

Enter number of elements in the array: 5

Enter the elements of the array:

Element 0: 10

Element 1: 20

Element 2: 30

Element 3: 40

Element 4: 50

Choose an operation:

1. Insert element

2. Delete element

3. Search element

4. Display array

5. Exit

Enter your choice: 1

Enter the element to insert: 25

Enter the index where the element should be inserted: 2

Element inserted successfully.

Choose an operation:

1. Insert element

2. Delete element

3. Search element

4. Display array

5. Exit

Enter your choice: 4

Array elements: 10 20 25 30 40 50

References:

1) The C Programming Language, Brian W. Kernighan and Dennis M.Ritchie.
2) C Programming: A Modern Approach, K.N. King.

Termwork-3: Write a Java Program to access the given excel file with known file format.

Problem Statement:

Write a Java Program to access the given excel file with known file format.

Theory:

To access and manipulate an Excel file in Java, you can use the Apache POI library, which provides support for reading and writing files in Excel format (.xls and .xlsx). Below is a simple Java program that reads data from an Excel file:

Maven Dependency (Apache POI)

Make sure to include the Apache POI library in your project. If you're using Maven, add the following dependencies to your pom.xml:

Program:

```
<dependencies>
        <!-- Apache POI for Excel -->
        <dependency>
      <groupId>org.apache.poi</groupId>
      <artifactId>poi-ooxml</artifactId>
        <version>5.2.3</version>
        </dependency>
        <!-- Apache POI dependencies -->
        <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-collections4</artifactId>
        <version>4.4</version>
        </dependency>
</dependencies>
Java Program to Read Excel File
import org.apache.poi.ss.usermodel.*;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
public class ExcelReader {
        public static void main(String[] args) {
        String excelFilePath = "path/to/your/excel/file.xlsx";
```

```
try {
// Open the Excel file
FileInputStream file = new FileInputStream(new File(excelFilePath));
// Create a workbook instance
Workbook workbook = new XSSFWorkbook(file);
// Get the first sheet
Sheet sheet = workbook.getSheetAt(0);
// Iterate through the rows
for (Row row : sheet) {
        // Iterate through the cells in each row
        for (Cell cell : row) {
        switch (cell.getCellType()) {
        case STRING:
            System.out.print(cell.getStringCellValue() + "\t");
                break;
        case NUMERIC:
            System.out.print(cell.getNumericCellValue() + "\t");
                break;
        case BOOLEAN:
            System.out.print(cell.getBooleanCellValue() + "\t");
                break;
        case FORMULA:
            System.out.print(cell.getCellFormula() + "\t");
                break;
```

```
                default:
                    System.out.print("Unknown Value\t");
                }
                }
                System.out.println(); // Move to next line
            }
            // Close the workbook and file stream
            workbook.close();
            file.close();
            } catch (IOException e) {
            e.printStackTrace();
            }
            }
}
```

References:

1) Herbert Schildt, Java-The Complete Reference, 9th Edition, Tata McGraw Hill, 2014.
2) Java Programming: From Problem Analysis to Program Design, D.S. Malik.