BAHIRDAR INSTITUTE TECHNOLOGY

Name:Yonas Abate

ID:1602838

DEPT:2nd year software engineering

SUBMITTED TO:

Lec.Wondimu

System Call Implementation in BeOS - Detailed Explanation

Introduction

- **This document provides a detailed explanation of system call implementation in the BeOS operating system, based on the student's individual project. It focuses on how system calls interface between user applications and the OS kernel, and provides an example of file manipulation using C++.**

System Calls in Depth

- System calls act as the interface between user applications and the operating system's kernel. They enable user programs to request services provided by the kernel, such as file access, memory allocation, and process control.
- The basic flow of a system call is as follows:

- 1. User Application: Requests a service.
- 2. Library/Wrapper Function: Invokes the system call.
- 3. Kernel: Performs the requested task.
- 4. System Call Interface: Switches to kernel mode to perform privileged operations.

Types of System Calls

1. Process Control: fork(), exec(), exit()
2. File Manipulation: open(), read(), write(), close()
3. Memory Management: mmap(), brk()
4. Communication: socket(), bind(), listen()

Example: File Manipulation in BeOS

Below is an example of using system calls in BeOS to create, write to, and verify a file. This is implemented using C++:

```
#include <iostream>
#include <fstream>
#include <cstdio>
using namespace std;

int main() {
    ofstream file("newfile.txt");
    if (!file) {
        cerr << "File creation failed." << endl;
        return 1;
    }
    file << "This is a system call demonstration." << endl;
    file.close();

    FILE *checkFile = fopen("newfile.txt", "r");
    if (checkFile) {
        cout << "File created successfully." << endl;
        fclose(checkFile);
    } else {
        cerr << "File does not exist." << endl;
    }
    return 0;
}
```

Explanation

- ofstream: Opens a file for output. If it doesn’t exist, it's created.
- file << : Writes a string into the file.
- fclose(): Closes the file, ensuring data is saved and resources are freed.
- fopen(): Opens the file in read mode to check if it exists.

If fopen() succeeds, the program confirms that the file was created.