3. System Call Study – `vfork()`

3.1 Introduction to System Calls

In any modern operating system, **system calls** serve as the critical bridge between user-level programs and the kernel, the core of the operating system. These calls allow programs to request services such as file handling, process creation, memory allocation, and device control from the operating system.

Among these system calls, those related to **process creation** are fundamental to multitasking and parallel execution. One particularly efficient process creation system call is `vfork()`—a variation of the more commonly known `fork()`.

3.2 Understanding `vfork()`

The `vfork()` system call is used to create a new process (child) from an existing one (parent), but with a key difference: it avoids copying the parent's address space. Instead, the child **shares the parent's memory** temporarily until it either executes a new program using `exec()` or terminates using `_exit()`.

This makes `vfork()` significantly faster and less resource-intensive than `fork()`, especially when the child process intends to immediately launch another program.

How `vfork()` Works:

- A **child process** is created that shares the **same memory space** as the parent.
- The **parent process is suspended** until the child either calls `exec()` or `_exit()`.
- The child must **not modify** any variables or data in the shared memory space.
- It is ideal in situations where the child process is only a transition to another program.

3.3 C++ Code Example Using `vfork()`

Here's a practical implementation of `vfork()` in a simple C program

```cpp
#include <iostream>
#include <unistd.h>
#include <cstdlib>

int main() {
    pid_t pid;

    std::cout << "Before vfork()" << std::endl;

    pid = vfork();

    if (pid < 0) {
        std::perror("vfork failed");
        std::exit(1);
    }

    if (pid == 0) {
        // Child process
        std::cout << "Child process. PID: " << getpid() << std::endl;
        _exit(0);  // Use _exit() to avoid corrupting parent memory
    } else {
        // Parent process
        std::cout << "Parent process. PID: " << getpid() << std::endl;
    }

    return 0;
};
```

Compilation & Execution

To compile and run the above program:

Summary

Through this project, I gained valuable insights into Linux-based operating systems like Mageia, the power of virtualization in enhancing system flexibility, and the role of system calls like vfork() in efficient process management. These fundamental concepts are essential building blocks of modern operating systems and are crucial knowledge for software engineering professionals.