CS 2301-01 Operating Systems 1

Instructor: Professor Ravichandran Purushothaman

Name: Ryohei Hayashi

Learning Journal 2

**1. Discuss the various categories of system calls and explain the functionality of any one system call that is handled by the operating system with an example.**

System calls provide an interface for application programs to communicate with the operating system kernel. This allows programs to access hardware resources and perform process management. System calls can be categorized into five main categories:

1. Process Control: This includes operations like creating and terminating processes and setting process attributes.
2. File Operations: This involves reading, writing, creating, and deleting files. The open() system call, for instance, is used to open a file.
3. Device Management: This includes reading from and writing to devices, and configuring device settings.
4. Information Maintenance: This involves retrieving system information.
5. Communication: This includes inter-process communication functionalities.

As an example, consider the open() system call. The open() system call is used to open a file, making it accessible for reading, writing, or both. This system call returns a file descriptor, which is a reference to the opened file. The file descriptor is then used by other system calls, such as read(), write(), and close(), to perform operations on the file.

Here is an example of using the open() system call in C:



In this example, the open() system call is used to open a file named "example.txt" in read-only mode (O\_RDONLY). If the file is successfully opened, a file descriptor is returned. This file descriptor is then used by the read() system call to read data from the file into a buffer. After reading, the buffer is null-terminated and its contents are printed. Finally, the close() system call is used to close the file.

The open() system call demonstrates how a program can interact with the file system to perform essential operations like reading a file's contents. This interaction between user-level programs and the operating system kernel is crucial for managing resources and ensuring secure, efficient access to hardware components.

**2. Give an example to illustrate the various process states. Why is it expensive to switch between processes? Additionally, discuss the circumstances under which a parent process can stop a child process from executing.**

Linux has five main process states:

1. Running: The process is currently being executed by the CPU.
2. Waiting: The process is waiting for some resources. This can be further divided into short-term and long-term waiting.
3. Stopped: The process is stopped and waiting for an external signal.
4. Zombie: The process has finished execution, but its parent process has not yet read its exit status.
5. Idle: The process is temporarily paused but not fully stopped.

Switching between processes is expensive because it requires a context switch. During a context switch, the state of the currently running process must be saved, and the state of the new process must be loaded. This involves saving and restoring various information, such as CPU registers, memory maps, and file descriptors, which consumes time and resources.

A parent process can stop a child process from executing under several circumstances:

* When the child process finishes execution. For instance, the exit() system call can terminate a child process.
* When the parent process uses the kill system call to terminate the child process. For example, kill(pid, SIGTERM) can terminate a specific process.
* When the parent process uses the wait system call to wait for the child process to finish. This makes the parent process wait until the child process exits and retrieves its exit status.
* When the child process is dependent on the parent process, and the parent process terminates. In such cases, the child process is automatically cleaned up by the system.

These examples demonstrate that process management and state transitions are crucial functionalities of the operating system, contributing to efficient resource management and system stability.

Word Count: 567

References

1. GeeksforGeeks. (n.d.). *Linux system call in detail.* Retrieved from https://www.geeksforgeeks.org/linux-system-call-in-detail/
2. CBT Nuggets. (n.d.). *What are the 5 Linux Process States?* Retrieved from https://www.cbtnuggets.com/blog/certifications/open-source/what-are-the-5-linux-process-states