CIS 415 Operating Systems

Project 2 Report Collection

Submitted to:

Prof. Allen Malony

Author:

*Amelia Bates*

*abates*

*951915806*

**Report**

**Introduction**

*Project 2 of CS 415 Operating system involves the development of Master Control Program (MCP) to manages and schedules multiple processes. This project uses system calls such as fork(), exec(), and wait(). As well as signals such as SIGUR1, SIGSTOP, and SIGCONT. Through this project, I explored operating system concepts, such as process creation, signal handling, and scheduling, which are critical for efficient system performance. The project is divided into four sections, each progressively building on the capabilities of the MCP*

**Background**

*This project involves the key concepts of process management and scheduling:*

***System Calls (fork(), exec(), wait())****:*

* *fork(): used to create a new child process which duplicates the parent process.*
* *exec(): executes a specified command.*
* *wait(): blocks the parent process until the termination of child processes*

***Signals (SIGUSR1, SIGSTOP, SIGCONT)****:*

* *SIGUSR1: A user-defined signal to pause processes.*
* *SIGSTOP: Pauses the execution of a process until a SIGCONT signal is received.*
* *SIGCONT: Resumes a paused process.*

***Scheduling****:*

* *Round-Robin Scheduling: A preemptive scheduling algorithm where each of the processes gets an equal time slice (1). Using the alarm() function allows setting timers for the context switching, ensuring fair CPU allocation among the processes.*

**Implementation**

*This project is separated into four different parts:*

*Part 1: This program reads the commands from the input file and launches each as a separate process.*

*Part 2: This program uses signal handling to pause the processes before they begin to be executed. This is done by having the forked child processes wait for a signal (SIGUSR1) before executing the command in the child process through exec(). Then the parent process sends the signal to all of the child processes which causes them to start. The process are also suspended and resumed using signals (SIGSTOP and SIGCONT).*

*Part 3: This program implements the round-robin scheduler. This is done by using the alarm() function to set a timer for the time slices. And then once a alarm signal is received, the current process is suspended and the next one is started.*

*Part 4: This program displays system resource information using the /proc filesystem.*

**Performance Results and Discussion**

*This project was tested with multiple input files:*

*The provided file:*

*ls -a -r -s*

*sleep 1*

*invalid name*

*./iobound -seconds 10*

*./cpubound -seconds 10*

*Test file #1:*

*ls -l*

*echo Hello World*

*pwd*

*Test file #2:*

*ls -l*

*sleep 2*

*echo "hello world"*

***Results***

*Part 1: Successfully launched and terminated all processes sequentially.*

*Part 2: Correctly synchronized process start and stop using signals. The processes did not start execution until the parent sent SIGUSR1.*

*Part 3: The round-robin scheduler was implemented successfully. Even with mixed workloads, the time slices ensured responsiveness.*

*Part 4: Displayed accurate CPU and memory usage statistics, providing insights into resource consumption.*

*All 4 parts were checked for memory leaks. All allocated memory was freed and there were no leaks or errors.*

**Conclusion**

*Overall, this project was a valuable learning experience to understand process management, signal handling, and scheduling in UNIX. In the future, more error handling would be beneficial, such as implementing SIGINT signal; handler to terminate the running processes for a graceful exit. In addition, implementing additional scheduling algorithms would be a beneficial feature to explore in the future.*