CIS 415 Operating Systems

Assignment 2 Report

Submitted to:

Prof. Allen Malony

Author:

Xuehai Zhou

Report

Introduction

This program is a Master Control Program, which was been considered one of the predecessors of the modern operating system. In part 1, the program will take an input file with command lines and read the command and create child processes to execute those commands. Based on create and execute commands and programs, in part 2 the parent process will wait all child processes to finish executing. In part 3, we execute programs with round-robin that each process gets execute in a certain amount of time then go to the next process until all processes get done. In part 4, we read status of each processes from proc/PID/status and print out their information to simulate the usage of the top command.

Background

In regards of finish this assignment, we need to know the usage of fork(), execvp(), wait()/waitpid(), sigwait(), kill(), alarm() and the usage of a signal set and signal handler function, etc. For instance,

1. *fork() will create a child process and return 0, so when PID is equal to 0, we run the child process.*
2. *execvp() will take a command or program as the first parameter, also take the arguments as the second parameter to execute a program.*
3. *wait()/ waitpid() will wait a process to finish, then the current process finish itself.*
4. *sigwait() will pause the process until it receive a signal that is in the signal set.*
5. *kill() can pass a signal to a process with the PID passed in. Signals can be seen by type in command kill with flag -l in console.*
6. *In order to get sigwait() working, we want to initialize a signal set and use sigaddset() and sigaction() to register signals.*
7. *alarm(unsigned int s) will have a int number passed in and arrange SIGALRM to be delivered to the calling process in s seconds.*
8. *./proc is a pseudo-filesystem that provides an interface to kernel data structure and process the information we want to extract to be printed out in part 4.*

Implementation

Part 1, we read each line from the input file and tokenize the programs and arguments, and store them in an argument array. Then we fork() to create child processes. If PID is equal to 0, we are in child processes, so we call execvp to run each program in the argument arrays. Finally, we call waitpid() in parent process to wait all child processes to finish.

Part 2, we want to learn the usage of signals and signal set. We initialize a signal set by signal_t and register signals by sigaddset(). After we set up the signal set, we call sigwait() in each child processes to pause. In parent process, we pass in SIGUSR1 to resume all child processes. Then we pass in SIGSTOP to stop all child processes. Then we pass in SIGCONT to restart every child process. Then the parent process waits all child to finish. Then the program is finished.

Part 3, based on what we already have in part 2. We add a round robin algorithm. We schedule all child processes by switch passing in SIGCONT and SIGSTOP. We call alarm() and sigwait() sandwiched by SIGCONT and SIGSTOP to decide how long each process run in each circle of the scheduling. More information about alarm() and sigwait(): alarm() delivers SIGALRM for several seconds and sigwait() waits for SIGALRM, so that the scheduling will success. If the process pool has only last one process to run, we no

longer need to schedule that process. Therefore, we count alive processes each time after each circle of the scheduling. If alive process is equal or less than 1, we break the loop. And the parent process wants to wait the last process to end. Then the program is done.

Part 4, on the top of part 3, we write a function to read file from proc/PID/status and print out information. Since in -std=gnu99 we cannot use function itoa(), we want to convert PID into a string we use sprintf(). We open the file and getline() and parse the data by using strtok(). Then we choose the line which process important information and print.

Performance Results and Discussion

All parts from 1 to 4 are success and no memory leak existing. In part 1, the program can read input file and store programs and arguments in an array and go to execvp() successfully. The program creates child process success. And parent process will always wait all children finished. In part 2, child processes waiting signal succeeds and signal passed in succeed as well. In part 3, the round robin scheduling succeeds. Each program will run 1 second each time and go to next program and run until the last program remaining. Then run the last program and ends exits main. In part 4, the program can read from proc/PID and print out the processes, which can also show each program is running or stopped.

Conclusion

This project is a good training to give a general idea how the operating system creates processes and how it executes programs in the backstage. And also, along the way to implement the round robin algorithm, we can have an in-depth understanding of process scheduling.