CIS 415 Operating Systems

Project <MCP: Ghost in the Shell> Report Collection

Submitted to:

Prof. Allen Malony

Author:

<Tong Guan>

<UO ID - tongg>

<951871617>

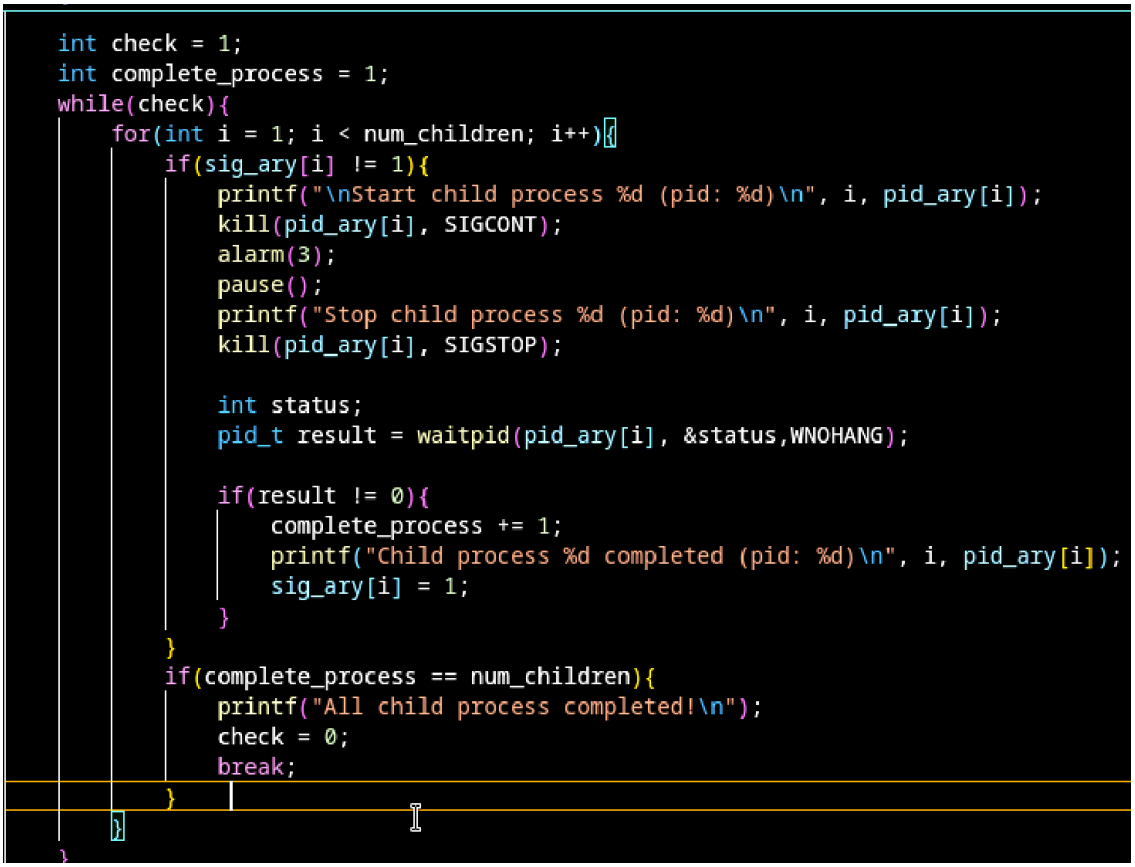
Report

Introduction

This is a project about MCP (Multi-Processing Control Program). By creating child processes based on different commands(workloads), we improve our system performance and efficiency. In the Part1, we practiced basic fork()and wait() in process, understanding how to copy from parent process and replace the context from the input. In the Part2, we focused on implementing communication between the parent and child processes. We control and managed different child processes through signal setting and sending from the parent process. In the Part3, we used a Round-Robin algorithm to allocate CPU resources to ensure that each child process gets enough time to run. In the Part4, we printed the I/O, memory resources, and execution time accessed by different processes during their execution.

Background & Implementation

In the design of the CPU Scheduling in the Part3, I reserved the communication mechanism via signal from the Part2. After child processes were created , all of them are wait in place restricted by sigwait(). Then, only by receiving the allowed signal SIGCONT, each of them start executing sequentially.



```
int check = 1;
int complete_process = 1;
while(check){
    for(int i = 1; i < num_children; i++){
        if(sig_ary[i] != 1){
            printf("\nStart child process %d (pid: %d)\n", i, pid_ary[i]);
            kill(pid_ary[i], SIGCONT);
            alarm(3);
            pause();
            printf("Stop child process %d (pid: %d)\n", i, pid_ary[i]);
            kill(pid_ary[i], SIGSTOP);

            int status;
            pid_t result = waitpid(pid_ary[i], &status,WNOHANG);

            if(result != 0){
                complete_process += 1;
                printf("Child process %d completed (pid: %d)\n", i, pid_ary[i]);
                sig_ary[i] = 1;
            }
        }
        if(complete_process == num_children){
            printf("All child process completed!\n");
            check = 0;
            break;
        }
    }
}
```

Next, considering of the timer setting up, I chose to use SIGALRM, alarm(), and pause() as interruption /alarm mechanisms. When the time set in alarm expires(in my program, the timer is set to 3 seconds), the alarm_handler function is called (instead of terminating the entire program), and the paused program is resumed. The current child process is being stopped and the system switch to the execution of the next child process. Observing the wait() will control the execution of the child processes and it's hard to interrupt it in my way, I chose not to use wait-related statements and instead write a while loop to control all child processes. This while loop has a flag called 'check'. When all child processes have completed their execution, 'check' changes from 1 to 0, and the while loop terminates. The state of all child processes are initialized to 0, and when the child process complete its task, the state will change from 1 to 0, thereby avoiding wasting time and resources on processes that have already finished.

Performance Results and Discussion

All four parts of my program have successfully met the project requirements, and there are no memory leaks detected under Valgrind's checking. I did think about allocating space in every potential error-prone area to prevent the program from being abruptly interrupted or terminated without specific error messages. Additionally, I think and designed the control of the program's output through printf statements to make it appear more concise, clean, and organized.

Conclusion

To be honest, I really enjoyed the process of implementing this project, especially in the Part3 where I had to choose the right algorithm for better CPU and process scheduling. This part didn't have very specific steps on how to implement it; instead, it gave us a lot of design and implementation freedom. While working on this part, I repeatedly searched and think about the concepts and methods of signals, alarms and the control of child processes, and try my best to write a good combination to address all the requirements. In discussions with the awesome TA Alex, I learned that there are other ways to achieve the goals(might be more simple and beautiful）, and I felt excited and happy about the freedom and openness of the programming.