# CIS 415 Operating Systems

Project 2 Report Collection

Submitted to:
Prof. Allen Malony

Author:
*Antonio Silva Paucar*

Antonio Silva Paucar

# Report

## Introduction

The following report is about project number two; MCP Ghost in the shell. In for part, we have to fork processes, execute them, use alarms and signals to wake them up and wait for the children to finish up to terminate the parent.

## Background

After the experience with project number one, to parse the command lines were no challenge at all. A lot of the information necessary for the project was found in the textbook, but also some online references were necessary.

## Implementation

One of the struggles (and personal defeat) for me was to try to use sigwait(). I found very confusing the man text and I didn't really found very much examples of how to use it. I tried to implement it in several ways, but I did not succeed. Hence, I had to find another solution. I used a combination of signals, alarms and loops to create a similar effect. For instance:

```
int indicator = 1;//if 1, true and while loop continue

void continue_loop(int signo){// gets signals and change indicator to be able to continue with the loop.

  indicator = 0;

}
```

The code above would be called by:

```
if (signal(SIGUSR1, continue_loop) == SIG_ERR || signal(SIGALRM, relog) == SIG_ERR) {// to get the signals

  return -1;

}
```

And this would be activated when a signal would be sent. For releasing the SIGURS1 signal, I did the following:

```
indicator = 1;// set to 1 to keep the loop rolling till the signal is sent and the value is changed to 1.

while(indicator){// trap the execution of exec till the sigusr1 signal is sent by kill()

  sleep(1);

}

execvp(tokens[0],tokens);
```

The snippet above would loop until all the children are forked. The way we send the signal is by waiting all the children as forked and in a infinite loop. Once all of them are forked, the main process or parent will continue and execute the instructions that are designed to release the exec()

```
for (int i = 0; i < number_lines; i++) {//launching exec for all the fork children and stopping to wait for the schedule

  kill(child[i],SIGUSR1);

  kill(child[i],SIGSTOP);

}
```

After this, the code would be ready to complete their task or be processed by a scheduler. In my code, I avoided dividing the file into headers as much as I can, to leave the most important parts of the process clear in one page.

For the scheduler (part 3), I did the following:

```
 while(1) {
   int finish = waitpid(child[i], &status, WNOHANG);
   if (finish == 0)//if it is not finish, it gives 0
   {
     strcpy(temp,lista[i]);
     temp[strlen(temp)-1] = 0;//temporal to get the command line that is being processing.
     printf(ANSI_COLOR_CYAN"\n ------- Starting process number %d (%s). -------"ANSI_COLOR_RESET"\n", i, temp);
     kill(child[i], SIGCONT);//stop child
     tiempo = 1;//set up infinite loop till alarm is send
     alarm(PROCESS_RUNNING_TIME);//PROCESS_RUNNIN_TIME is defined as “3” for 3 seconds.
     while(tiempo){}//loop waiting for the alarm to send back signal.
     kill(child[i], SIGSTOP);
     if (!waitpid(child[i], &status, WNOHANG)) {
         printf(ANSI_COLOR_CYAN" ------- Stoping process number %d and restarting the next process in line. -------
"ANSI_COLOR_RESET"\n", i );
     }else{
         printf(ANSI_COLOR_CYAN" ------- Done with process number %d. Now next process in line -------"ANSI_COLOR_RESET"\n",
i );
     }
   }
```

We can see how we first resume one process that we stop just right after the exec() order was issue, we process it for 3 seconds (there is a loop after it that, the same way the we stopped the exec() order, stop the signal that is supposed to stop the process and allow a new one to enter into the cpu) and then we send an alarm to the handler, which change the value of the condition to allow to reach the signaler to stop the current process and restart the loop with the next process in line.

For part 3, I added some prints to signal each step of the process; when stop, when restart, when is done, when is not the right command, etc. The lines are shown in green color.

## Performance Results and Discussion

I didn’t have success with the use of sigwait(), so I just can wonder if it is more efficient than the solution I found.

I guess the faster we do the switch between process, the more efficient the scheduler will be. (creates the illusion of parallel processing).

## Conclusion

Project 2 has been a good way to understand how the illusion of multiprocessing is achieved by a CPU. The acceleration of the exchange between process to ms instead of seconds can make it seems that several processes are executed at the same time.

Bibliography:

https://www.geeksforgeeks.org/clear-console-c-language/
https://alvinalexander.com/linux/unix-linux-process-memory-sort-ps-command-cpu/
http://man7.org/linux/man-pages/man2/alarm.2.html
https://www.tutorialspoint.com/c_standard_library/c_function_signal.htm
https://www.geeksforgeeks.org/wait-system-call-c/