Operating Systems

Assignment 03 – "Processes" [2017/2018]

Introduction

A process is an instance of a program that is executing. In this assignment the student will learn how to create processes and how to understand the UNIX process model.

Objectives

Students concluding this work successfully should be able to:

- Create and manage processes in Unix, using C.

Support Material

- K. A. Robbins, S. Robbins, "Unix Systems Programming: Communication, Concurrency, and Threads", Prentice Hall:
  - Chapter 2 - Programs, Processes and Threads
  - Chapter 3 - Processes in UNIX
- "Programming in C and Unix":
  - Process Control: <stdlib.h>,<unistd.h>

Exercises

1. Father and son

Analyse, compile and run the following code (file *father_son.c*).

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(void) {
  pid_t mypid;
  pid_t childpid;

  mypid = getpid();
  childpid = fork();

  if (childpid == 0) {
    printf("child mypid :%ld\n", (long)mypid);
    printf("child getpid() :%ld\n", (long)getpid());
    printf("child getppid() :%ld\n", (long)getppid());
  } else {
    printf("mypid :%ld\n", (long)mypid);
    printf("getpid() :%ld\n", (long)getpid());
    printf("getppid() :%ld\n", (long)getppid());
  }
  return 0;
}
```

a) Evaluate the output of this code and identify the differences between *mypid* and *getpid()* in each scenario. Why is the result different every time you run the program?

b) What happens to a child process when a parent process ends? What becomes the value of its *getppid()*?

c) Who is the father of the original process?

d) What happens if inside the *else* block you add a *sleep(5)*?

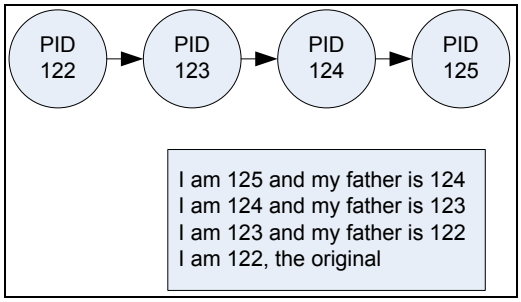
NOTES:

- Use the command *ps* in the console to proper evaluate the hierarchy of processes and to identify zombie and orphan processes.

- Use the online manual to access a command/function specification (*man [section] name*)

2. Reverse print

Implement a program that creates a sequence of *n* processes and where each process prints it's PID and its parent PID. The output of this program must be in the reverse order of the processes creation. The following figure shows an example of the expected result.



a) Implement this program using the *sleep* function.

b) Implement this program without the *sleep* function and using the *wait()* function.

3. Decrypt text file

Create a program that uses the information of a file with pairs of encrypted/readable words to decrypt a text file (by replacing all encrypted tokens with the corresponding word).

Implement the following functionality:

a) Create a list of pairs of readable/encrypted words with the contents of the file to decrypt. Name the file *decrypt.txt*.

b) Open the encrypted text file (file name received by a program parameter and using "txt" extension) and read it line by line. For each line create a new process (worker) to handle the decryption of the line.

c) The worker, searches the text line for all the encrypted words and replaces them by the corresponding decrypted word. After the decryption process the worker should print the decrypted line:

```
"line 4: operating system is an essential component of the"
```

d) Make sure the main process only exits after all translations finish. Use the *wait()* or *waitpid()* functions to implement a mechanism to wait for all child processes. Only after all child processes terminate the main process should end.

e) Before the main process exits, it should print the number of lines translated:

```
"Translation complete. 24 lines translated."
```