Lab Assignment

(To refresh concepts of Process and Thread)

Description

As you may know a *program* is an executable file residing in a disk file. An executing instance of a program is called a *process*. You will learn many more details about processes in the lecture. In Unix a program is read into memory and executed by the kernel as a result of one of the six `exec` functions.
Every Unix process is guaranteed to have a unique numeric identifier called the *process ID*. The process ID (short: PID) is always a nonnegative integer.
Under Unix, the only way (except for some very special processes) to create a new processes is when an existing process calls the `fork` function. You will find details about `fork` and `wait` in the manual pages (`man 2 fork`).

Assignment 1: fork()

Write a simple program that uses the `fork(2)` function. Start off by completing `fork.c`.
The new process created by fork is called the *child* process. The function is called once but returns **twice**. Both the child and the parent continue executing with the instruction that follows the call to `fork`. Here is a typical example:

```
/* parent creates a process */
pid = fork();
/* the return value is a process ID, that ALWAYS needs to be tested */

switch (pid) {
    case -1:
        /* error: fork was unsuccessful */
        break;
    case 0:
        /* this is the child process */
        /* no process ID */
        /* ... do something ... */
        break;
    default:
        /* this is the parent process */
        /* pid=process ID of the child */
        /* ... */
}
/* both processes continue here */
```

You can list processes by using the `ps(1)` command (see manual by typing `man 1 ps`), or by using the `top` command. Under Unix a process that has terminated, but whose parent has not yet waited for it is called a *zombie*.
Adjust the sleep calls for parent and child and use `ps` to answer these questions (add them as comments at the end of the first program):

1. What status is being printed when the child terminates before the parent, but the parent does not wait (use `ps ajx | grep fork`)?

2. What is the parent process id (PPID) of a child process whose parent terminates before the child?

To make the parent wait for the child, replace the `sleep()` by `pid = waitpid(pid, &status, 0)` in the code for the parent and read about the function by executing `man 2 wait`. `status` is an integer.

Assignment 2: Simple Shell

A simple shell can be written like this:

```
while (1) {
    printf("%% ");
    fgets(command, 80, stdin);
    system(command);
}
```

Write a program that does exactly this, but with your own implementation of the `system(3)` function. Use the `execv()` function to start an external program. (The `execv()` function is just a frontend for the `execve()` function so you should read both manpages to understand what this function does.)
To do an `ls -l` in your shell, you actually need to call `/bin/sh -c "ls -l"`, so `-c` and `ls -l` are parameters to `/bin/sh`. Please note (as you can read from the manpage), that the list of arguments **must** be terminated by a NULL pointer, and this pointer must be cast `(char *) NULL`.
With your program you should be able to create a session like this:
(Note that the % is the prompt of your program).
$ **./myshell**
% **date**
Fri Feb 2 15:40:03 CET 2007
% **who**
hstamerjoh
% **pwd**
/home/hstamerjoh/files
% **ls**
Makefile myshell myshell.c
% **abc**
/bin/sh: abc: command not found
% ^D (Ctrl-D)
$

Your solutions

Name the programs clab_1_x.c.
For each program you must include a comment on the top like the following

```
/*
   320232
   clab_1_1.c
   Firstname Lastname
   email@iu-bremen.de
*/
```

Assignment 3: Implement a Thread using pthread library

The aim of this assignment is to have you feel to understand the concurrency and pseudo-parallelism through thread implementation.

Write a C program to create a user level thread using system call pthread_create( ) and assign the thread to display the "HELLO WORLD" . Use pthread_exit() in your program (if possible) for terminating the thread.
/* You need to put explanatory comment in your program to demonstrate the purpose and why you have used the system calls */

Your Solution

In your command prompt use the command "ps –eLf|more", "ps –T –p <pid>"
Refer this link: https://unixhealthcheck.com/blog?id=465

Hints:
* To know more about pthread_create( ), see **pthread_create (3)** man page and to know more about pthread_exit( ), see **pthread_exit (3)**.

References:

Follow the links given below and do the contents practically in your machine:

1. Using Unix/Linux: Tutorial for Beginners: https://www.cs.sfu.ca/~ggbaker/reference/unix/

2. Learning Bash Scripting for beginners: http://programmingexamples.wikidot.com/bash-scripting

3. POSIX Threads Programming by Blaise Barney, Lawrence Livermore National Laboratory
https://computing.llnl.gov/tutorials/pthreads/

4. POSIX thread (pthread) libraries: YoLinux Tutorial
http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html#BASICS