**COMP 3410 -**

**Operating Systems (3, 1, 0)**

**Winter, 2018**

**Lab/assignment 2**

**Due: Thursday February 1st**

In this lab, we will only practice and document the UNIX system calls, only related to the processes. We will work on process creation and termination and also the way a process can be sent to the background.

**Problem:**

1. exec () - The **exec**() family of functions replaces the current process image with a new process image.

2. fork () - **fork** is an operation whereby a process creates a copy of itself

3. wait ()-The **wait**() system call suspends execution of the current process until one of its children terminates.

4. exit () - The function **\_exit**() terminates the calling process "immediately". Any open file descriptors belonging to the process are closed;

5. kill () - The **kill**() system call can be used to send any signal to any process group or process.

To do:

Q1. Get onto the Unix machine and run the system calls, fork(), wait() and exec()

Write a program named as “testOS.c” that executes the “*cat –b –v –t filename*” command in the child process

Details:

1. Type the following program in unix
2. Compile it using gcc as gcc testOS.c
3. Run the program as the following command:

$./a.out filename

What is happening:

1. Your code will *fork()*
2. The child will use the *execl* to call *cat* and use the *filename* passed as an argument on the command line
3. The parent will wait for the child to finish
4. Your program will print from both processes:
   * + The process id
     + The parent id

Code to compile and run

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main (int args, char \*argv[])

{

pid\_t fork\_return;

pid\_t pid;

pid=getpid();

fork\_return = fork();

// When fork() returns -1, an error happened.

if (fork\_return==0)

// When fork() returns 0, we are in the child process.

{

printf(“\n\nThe values are Child ID = %d, Parent ID=%d\n”, getpid(), getppid());

execl(“/bin/cat”, “cat”, “-b”, “-v”, “-t”, argv[1], 0);

}

else

// When fork() returns a positive number, we are in the parent process

// and the return value is the PID of the newly created child process.

{

wait(NULL);

printf(“\nChild Completes “ );

printf(“\nIn the Parent Process\n”);

printf(“Child Id = %d, Parent ID = %d\n”, getpid(), getppid());

}

return 0;

}

Q2. Once the first program is typed, and executed, answer the following questions

* 1. If you try to print a message after the *exec\** call, does it print it? Why? Why not?
     1. It prints the message after the execl statement is completed.
     2. If no argument is passed when running the .out file, the program doesnot print anything after the execl statement.
  2. Who is the parent of your executable program?
     1. The bash process is the parent of the program.
  3. How would you change the code so that the child and parent run concurrently?
     1. Remove the WAIT statement to allow the parent to run without waiting for child

**Running a Process**

When you start a process (run a command), there are two ways you can run it:

* Foreground Processes
* Background Processes

**Foreground Processes:**

By default, every process that you start runs in the foreground. It gets its input from the keyboard and sends its output to the screen.

**Background Processes:**

A background process runs without being connected to your keyboard. If the background process requires any keyboard input, it waits.

The advantage of running a process in the background is that you can run other commands; you do not have to wait until it completes to start another!

The simplest way to start a background process is to add an ampersand ( &) at the end of the command.

**Stopping Processes:**

If a process is running in background mode then first you would need to get its Job ID using **ps** command and after that you can use **kill** command to kill the process as follows:

$ps -f

UID PID PPID C STIME TTY TIME CMD

amrood 6738 3662 0 10:23:03 pts/6 0:00 first\_one

amrood 6739 3662 0 10:22:54 pts/6 0:00 second\_one

amrood 3662 3657 0 08:10:53 pts/6 0:00 -ksh

amrood 6892 3662 4 10:51:50 pts/6 0:00 ps -f

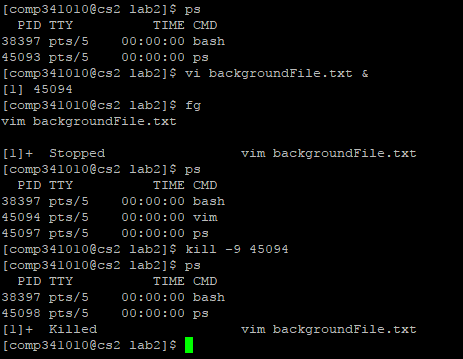
$kill 6738

Terminated

Here **kill** command would terminate first\_one process. If a process ignores a regular kill command, you can use **kill -c** followed by the process ID as follows:

$kill -9 6738

Terminated

Q3. Execute and document the commands to: 

1. Run a process in background
2. Run a process in foreground
3. Call the process from background to foreground
4. Kill the process
5. Show the process status

**Results:**

Hand in:

- A print out of the programs;

- A printout of the output of commands (screen capture)

- Answers to questions listed