System Programming II
Exercise On Processes
Name: Akech Dau Atem
Reg: SCT211-0535/2022

Exercise 1: Consider the code snippet below

```c
#include <stdio.h>
#include <unistd.h>

int main() {

int c = 5;
pid_t pid = fork();
if (pid == 0) { // First child process
c += 5;
} else {
pid = fork(); // Second fork (creates a new child)
c += 10; // Parent process after second fork
if (pid == 0) { // New child process
c += 10;
}
}
fork(); // Third fork
printf("%d\n", c); // Print the value of c in each process
return 0;
}
```

a) How many processes are created by the initial running of this program including the initial program created by running this program

First fork(): Two processes: the parent and the first child

Second **fork()**: Each of the two processes calls fork() again, creating two more processes (so now there are 4 processes in total).

All four processes call fork(), creating four more processes, leading to 8 (eight) total processes.

b) Show at least two possible outcomes of the program above after coding it in C and running it

- First possible output

  2 10

  2 25

  2 15
- Second possible output

10
15
25

Exercise 2

`The program uses fork()` and `printf()`. **How many `X`, `Y` and `Z` will be printed?**

```c
/* Exercise */
#include <stdio.h>
#include <unistd.h>
int main(int argc,char *argv[]){
 printf("X\n");
 fork();
 printf("Y\n");
 fork();
 printf("Y\n");
 return 0;
}
```

```
PROBLEMS  DEBUG CONSOLE  TERMINAL  OUTPUT  PORTS
Loaded '/usr/lib/libc++.1.dylib'. Symbols loaded.
X
6 Y
The program '/Users/macbook/c/Exercise_2' has exited with code 0 (0x00000000).
```

Total output:

- 1 X initially x is printed
- 6 Ys. (the firsrt fork() print 2 Ys and have two processes . Again the fork() prints 4 Ys and has four process giving total 6 Ys printed)
- 0 Z

Exercise 3:

Write another program using `fork()`. The child process need to print “Niko Juja” and the parent process to print “ ICS2305 ni softlife” . The child process should print first ---this can be done without calling **wait()** in the parent..

Hint : use of for loop and sleep

```c
// NAME: AKECH DAU ATEM  Reg No: SCT211-0535/2022

#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid = fork(); // Create a child process
    if (pid == 0) {
        // Child process PID is 0
        for (int i = 0; i < 3; i++) {
            printf("Niko Juja\n");   // Print the child process
            sleep(1);  // Let the child process print first by sleeping
        }
    } else {
        // Parent process PID is not 0
        for (int i = 0; i < 3; i++) {
            printf("ICS2305 ni softlife\n");    // Print the parent process
            sleep(1);  // Sleep to allow child process to print first
        }
    }
    return 0;
}
// The output of the code is:
// ICS2305 ni softlife
// Niko Juja
// ICS2305 ni softlife
// Niko Juja
```

Exercise 4:

Write a C program that prints the process ID , priorities and parent ID of all programs currently in the RAM ----

```c
// NAME: AKECH DAU ATEM Reg No: SCT211-0535/2022

#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <dirent.h>
#include <stdlib.h>
```

```c
// Check if string is numeric (valid PID)
int is_numeric(const char *str) {
while (*str) {
if (*str < '0' || *str > '9') return 0;
str++;
}
return 1;
}

int main() {
// Open /proc directory NB: Linux specific code
DIR *proc_dir = opendir("/proc");
if (proc_dir == NULL) {
perror("Error opening /proc");
return 1;
}

struct dirent *entry;
// Loop through /proc entries
while ((entry = readdir(proc_dir)) != NULL) {
if (is_numeric(entry->d_name)) { // Process only numeric (PID) entries
pid_t pid = atoi(entry->d_name); // Convert to PID
pid_t ppid = getppid(); // Get parent PID
int priority = getpriority(PRIO_PROCESS, pid); // Get priority

// Print process info
printf("Process ID: %d\n", pid);
printf("Parent ID: %d\n", ppid);
printf("Priority: %d\n\n", priority);
}
}

closedir(proc_dir); // Close directory
return 0;
}
```

Exercise 5:

Write program to illustrate the usage of execlp(), execle(), execv(), execvp(), exeve() system calls, ensure that in your program, there are enough comments explaining each of the working

```c
// NAME: AKECH DAU ATEM RegNo: SCT211-0535/2022

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

// Demonstration of exec family functions
```

```c
int main() {
// Example of execlp: executing 'ls' with options
printf("Using execlp to list files with details:\n");
execlp("ls", "ls", "-l", NULL);
// If execlp is successful, this comment won't be printed

// Example of execle: running 'env' command with a custom environment
printf("Using execle to print environment variables:\n");
char *envp[] = {"USER=akech", "HOME=/Users/macbook", NULL}; // Custom environment
execle("/usr/bin/env", "env", NULL, envp); // Execute 'env' to print environment variables

// Example of execv: running 'echo' with an array of arguments
printf("Using execv to print a message:\n");
char *args[] = {"/bin/echo", "This is an execv example", NULL}; // Arguments array
execv("/bin/echo", args); // Running 'echo' with arguments

// Example of execvp: running 'cat' with a file
printf("Using execvp to display the content of a file:\n");
char *args2[] = {"cat", "/etc/hosts", NULL}; // Arguments array
execvp("cat", args2); // Searches for 'cat' in PATH

// Example of exeve: running 'ls' in a specific directory with a custom environment
printf("Using exeve to list files in the root directory:\n");
execve("/bin/ls", args, envp); // Running 'ls' with custom environment

// This code won't execute if exec calls are successful
printf("Exec call failed, so this is printed.\n");

return 0;
}
```