## Contents

System Calls-fork(),exec() – Exercises ................................................................. 2

Exercise 1 : ........................................................................................................... 2
  Exercise 1 – Questions.......................................................................................... 2
  Exercise 1 – Solutions......................................................................................... 2

Exercise 2 : ........................................................................................................... 3
  Exercise 2–Questions............................................................................................ 4
  Exercise 2–Solutions............................................................................................. 4

Exercise 3 : ........................................................................................................... 5
  Exercise 3–Questions: Part 1 ............................................................................... 6
  Exercise 3–Solutions: Part 1 : ............................................................................ 6
  Exercise 3–Questions: Part 2 ............................................................................... 6
  Exercise 3–Solutions: Part 2 ............................................................................... 7
  Exercise 3–Questions: Part 3 ............................................................................... 7
  Exercise 3–Solutions: Part 3 ............................................................................... 7
  Exercise 3–Questions: Part 4-1 .......................................................................... 8
  Exercise 3–Solutions: Part 4-1 .......................................................................... 8
  Exercise 3–Questions: Part 4-2 .......................................................................... 9
  Exercise 3– Solutions: Part 4-2 ......................................................................... 9

Exercise 4 : ........................................................................................................... 13
  Exercise 4–Questions: ....................................................................................... 14
  Exercise 4–Solutions:.......................................................................................... 14

Exercise 5:............................................................................................................. 15
  Exercise 5–Questions:....................................................................................... 16
  Exercise 5–Solutions:.......................................................................................... 16

Exercise 6:............................................................................................................. 17
  Exercise 6–Questions:....................................................................................... 17
  Exercise 6–Solutions:.......................................................................................... 18
System Calls-fork(),exec() – Exercises

Exercise 1 :

```c
int EXE1() {  
    int i = 5, m = 4;  
    int pid;  
    if ((pid = fork()) == -1) { /* Start a child process. */  
        perror("fork error");  
        exit(EXIT_FAILURE);  
    }  
    /* What happen if the line written here ! i=6;*/  
    m = 3;  
    if (pid != 0) {  
        /*Parent Code*/  
        i = 6;  
        printf("the values of i = %d and m = %d \n", i, m);  
    } else {  
        /*Child Code */  
        printf("the values of i = %d and m = %d \n", i, m);  
    }  
    return 0;  
}
```

Exercise 1 – Questions
1- What is the output of the function EXE1() ?
2- Is the output will changed if we execute the commented line!?

Exercise 1 – Solutions
1- The output is :
    the values of i = 6 and m = 3
    the values of i = 5 and m = 3

2- Yes the out will change, because the pid!=0 is the code just the father execute and the two process have separated memory, then change in one will not affect the other:
    the values of i = 6 and m = 3
    the values of i = 6 and m = 3
Exercise 2:

```c
int EXE2() {
    int pid;
    if ((pid = fork()) == -1) /* Start a child process. */ {
        perror("fork error");
        exit(EXIT_FAILURE);
    }
    if (pid != 0) {
        /*Parent Code*/
        printf("Parent Code :
");
        printf("the values of Parent pid = %d \n", getpid());
        printf("the values of Child pid = %d \n", pid);
        printf("the values of pid = %d \n", getppid());
        wait(0);
    } else {
        /*Child Code */
        printf("Child Code \n");
        printf("the values of pid = %d \n", getpid());
        printf("the values of pid = %d \n", getppid());
    }
    return 0;
}
```

Notes:

- PID = the processes identifier - unique number for process.
- getpid() is a system call, it returns the PID of the current process.
- getppid() is a system call, it returns the parent PID of the current process.
- Linux gives the PID with ascending order - Parent have PID smaller than Child PID.
- we have to check the return value each time we call - system call function
- the parent for all process is called init and his pid = 1;

After running the parent code we get the output and wait:
the values of Parent pid = 3778
the values of Child pid = 3782
the values of pid = 2748
Exercise 2–Questions

1. According to the given output what will be the output in child code?

2. `getppid()` in the parent code return 2748, who is this process?
   
   you have to run “ps aux | grep 2748 “

3. What is the difference from running the code from eclipse (or some other programming software) and running the code from bash?

Note: these value may change in running on different PCs.

Exercise 2–Solutions

1- The output :
   
   the values of Child pid = 3782
   the values of Parent pid = 3778

2- It’s the pid of “eclipse” the program that run the parent code!, if we run the program from shell the pid will be the pid shell.

   ➔ These programs “eclipse” or shell when they run program they run it as child process

3- The difference is the pid indicate of the grandfather of child process will be to eclipse if run with eclipse and indicate to pid of shell if run in shell.
Exercise 3:

```c
int EXE3(int level) {
    int i, pid1, pid2;
    int status;
    int num_child = 0;
    for (i = 1; i <= level; i++) {
        if (num_child < 2) {
            if ((pid1 = fork()) == -1) {
                perror("fork error");
                exit(EXIT_FAILURE);
            }
            if (pid1 != 0) {
                /*Parent Code*/
                if ((pid2 = fork()) == -1) {
                    perror("fork error");
                    exit(EXIT_FAILURE);
                }
                if (pid2 != 0) {
                    /*Parent Code*/
                    //Part 4 : remove comments and see the result//
                    waitpid(pid1, &status, 0);
                    waitpid(pid2, &status, 0);
                    num_child = 2;
                    printf("Parent Code :n");
                    printf("the values of Parent  pid = %d \n", getpid());
                    printf("the values of Children  pid2 = %d and pid1 = %d \n", pid2, pid1);
                    //
                    Part 3 : remove comments and see the result//
                    sleep(3);
                }
            }
        }
    }
    //Part2 : remove comments and see the result //
    printf("Process PID = %d & his parents %d\n", getpid(), getppid());
    return 0;
}
```

After calling the function with level = 2, we get the output:

Parent Code:
the values of Parent  pid = 4086
the values of Children  pid2 = 4091 and pid1 = 4090

Parent Code:
the values of Parent  pid = 4091
the values of Children  pid2 = 4094 and pid1 = 4093

Parent Code:
the values of Parent  pid = 4090
the values of Children  pid2 = 4096 and pid1 = 4095
**Exercise 3–Questions: Part 1**

1- What does this code do? (How does the system processes look?)

**Exercise 3–Solutions: Part 1**

1- Its look like tree of process, where root is pid 4086

![Process Tree Diagram]

**Exercise 3–Questions: Part 2**

Remove the comments in the part2

After calling the function with level =1, we get the output

Parent Code:
The values of Parent pid = 4567
The values of Children pid2 = 4570 and pid1 = 4569

Process PID = 4567 & his parents 2748
Process PID = 4570 & his parents 1
Process PID = 4569 & his parents 1

1- Explain the contradiction in the output, in other words PID=4495 print that his children are with PIDs = 4570, 4569 while his children print that their parent with PID = 1?
2- Who is the process with PID = 1? What are these processes called?
Exercise 3–Solutions: Part 2

1- The code show that the parent finished before the children, then the children adopted by init. init is process which created and always have pid = 1, then after the father/parent die and child adopted by init, there getppid() will print 1;
2- The process with pid = 1 is alayws the init, and these process called orphans

Exercise 3–Questions: Part 3

Remove the comments in the part3

In this part we use system call sleep(3) (it will block the process for 3 seconds) to let the children finish before their parent.

1- What kind of problem this situation will produce? What are these processes called?

Exercise 3–Solutions: Part 3

1- The child process who terminated and but has not been waited for becomes a "zombie". Although zombies process don’t take much memory, but it take pid number, as we know OS can create finite number of process, as a result zombie process (after create a lot of them) take the free pid’s and we cant produce anymore process! (fork() return error since all the entries of process table are zombies).

While running the code you can run commands in bash and see the result:
- “top”
- ps aux | grep 'Z' - Check the process that have PIDs equal to the output you get!

borak  4944  0.0  0.0  0  0?  Z  16:25  0:00 [SysCall_Process] <defunct>
bork  4945  0.0  0.0  0  0?  Z  16:25  0:00 [SysCall_Process] <defunct>
4944, 4945 are the pid of child processes they are now Zombies!’
**Exercise 3–Questions: Part 4-1**

Part 4: we execute the comments in Part 4 (SEE CODE)
This part are related with other previous parts.

**We execute the comments in Part 2**
This is cutting part of the output!
Parent Code:
the values of Parent pid = 5625
the values of Children pid2 = 5631 and pid1 = 5630

1- What is the output of the code will be, what is the order if the execution?
You can write the all the output

**Exercise 3–Solutions: Part 4-1**

1- Process PID = 5631 & his parents 5625
   Process PID = 5630 & his parents 5625
   Parent Code:
   the values of Parent pid = 5625
   the values of Children pid2 = 5631 and pid1 = 5630
   Process PID = 5625 & his parents 3900

The order of execution will be the two children after that the parent.
We execute the comments in Part 3 and Part 2

Exercise 3–Questions: Part 4-2

1- Is using one `waitpid()` will solve the problem in Part 3 ? Which order of executions of processes will produce problems and which ones not ?
   - Think that other child we don’t wait for it.

Exercise 3– Solutions: Part 4-2

1- No, it will not solve the creation of zombies , `waitpid()` will wait for one process, the other process will terminated but has not been waited for it Then it will become zombie.
Before we discuss the family functions of “exec”, there is question:
Why Unix/Linux are commands like “ls”, “pwd” … You can run them without
absolute path from any place or from any directory but user programs we have to
give the full path?

To answer this question we want to know what is Environment? What are
Environment Variables?

When you start a program a new process is created. Process runs within what is
called an environment. This particular environment would be having some
characteristics/variables which the program/process may interact with.

Environment is set of variables that the process may need.

Example:
If our program using UNIX commands without full path (“ls”, “pwd”, ..) just by
pass the name of the command, then to run them successfully they use in one of
the variables in the Environment of the process/program.

$PATH: this variable contains a colon (:) separated list of directories in which
your system looks for executable files. One of them is /bin that contain all UNIX
commands.

echo $PATH

/usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games

Every program runs in its own environment. You can set parameters in this
environment so that the running program can find desired values when it runs.

To see a list of the environment variables that are already set on your machine,
type the following:
```
env or printenv

....

DESKTOP_SESSION=ubuntu
PWD=/home/borak/workspace/SystemCalls
GNOME_KEYRING_PID=2344
LANG=en_US.UTF-8
MANDATORY_PATH=/usr/share/gconf/ubuntu.mandatory.path
UBUNTU_MENUPROXY=libappmenu.so
COMPIZ_CONFIG_PROFILE=ubuntu
GDMSESSION=ubuntu
SHLVL=1
HOME=/home/borak
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LOGNAME=borak

.....
```
The exec family of functions shall replace the current process with new process, and will be no return from successful exec.

```c
exce1,exce1e,execlp,execv,execve,execvp ...
```

1 vs v: whether you want to pass the parameters to the exec'ed program as

1- l: individual parameters in the call (variable argument list): exce1(), exce1e(), exce1p(), and exce1pe()

   ls - is command that give you list directory contents

   execl("/bin/ls", "/bin/ls", "-r", "-t", "-l", (char *) 0);

2- v: as an array of char* excev(), execve(), execvp(), and execvpe()

   char *args[] = { "/bin/ls", "-r", "-t", "-l", (char *) 0 }; 
   excev("/bin/ls", args); // equal to excev(args[0],args);

The two commands are do the same job, but the format are not.
**Exercise 4:**
We have a program called func1

```c
int main(int argc, char **argv) {
    printf("I am one of EXEC family, and called as child process \n");
    printf("\%d\n", argc);
    printf("%s,%s\n", argv[0], argv[1]);
    return 0;
}
```

func1 will be run as a child process with one of the exec functions.

```c
int EXE4() {
    int pid, pid2;
    if ((pid = fork()) == -1) { /* Start a child process. */
        perror("fork error");
        exit(EXIT_FAILURE);
    }
    if (pid != 0) {
        /* Parent Code */
        if ((pid2 = fork()) == -1) { /* Start a child process. */
            perror("fork error");
            exit(EXIT_FAILURE);
        }
        if (pid2 != 0) {
            printf("Parent Code \n");
        } else {
            /* Child Code */
            // FULL PATH
            // execl("/home/borak/workspace/SystemCalls/func1", "func1", "SAM",
            //       (char*) 0);
            execl("func1", "func1", "SAM", (char*) 0);
            printf("Ending-----\n");
        }
    } else {
        /* Child Code */
        char *args[] = { "func1", "SAM", (char*) 0 };
        execv(args[0], args);
        // printf("Ending-----\n");
    }
    return 0;
}
```
**Exercise 4–Questions:**

1- If func1 exist in same directory.  
   What does the code in exercise 4 print?

2 - If func1 not exist in same directory,  
   Is exec fail? What will happen if func1/exec fail? Is the printing will be different from (1)?

**Exercise 4–Solutions:**

1- Parent Code  
   I am one of EXEC family, and called as child process  
   2  
   func1,SAM  
   I am one of EXEC family, and called as child process  
   2  
   func1,SAM  
   The exec will success in running “func1” and printing the results.  
   When exec success it did not return to the code.

2- Parent Code  
   Ending------Error in execv()  
   Ending------Error in execl()  
   The exec will fail in running “func1”, and printing the lines after exec.  
   When exec fail it return to the code.
e: The versions with an 'e' at the end let you additionally pass an array of char* that are a set of strings added to the spawned processes environment before the exec'ed program launches.

For example we have the possibility to extend the $PATH variable.

p : The versions with 'p' in there use the environment path variable to search for the executable file named to execute. ($PATH)

The versions without the 'p' require an absolute or relative file path to be prepended to the filename of the executable if it is not in the current working directory.

**Exercise 5:**

```c
int EXE5() {
    int pid, pid2;
    if ((pid = fork()) == -1) { /* Start a child process. */
        perror("fork error");
        exit(EXIT_FAILURE);
    }
    if (pid != 0) {
        /*Parent Code*/
        if ((pid2 = fork()) == -1) { /* Start a child process. */
            perror("fork error");
            exit(EXIT_FAILURE);
        }
        if (pid2 != 0) {
            printf("Parent Code \n");
        } else {
            /*Child Code */
            execvp("ls", "ls", (char*)0);
            printf("Ending-----\n");
        }
    } else {
        /*Child Code */
        char *args[] = {"ls", (char*)0};
        execv(args[0], args);
        printf("Ending-----\n");
    }
    return 0;
}
```
In this exercise we use “ls” command, it’s exist in /bin directory and this directory are in path variable $PATH.

**Exercise 5–Questions:**

1 - Is exec fail? Which one? What does the code print?
2- How can you fix the problem? Give 2 methods

**Exercise 5–Solutions:**

1-\(\text{exclp}()\) success and \(\text{execv}()\) fail ,because p extension let search in the paths that are in variable $PATH and the path /bin one of them .

The output:

Parent Code  
Ending-----> Error! : pid1  
Debug ProcessManagement.c // the ls result

2- you can fix by use p-extension or full path.

USE p extensions – FIX

\[
\text{exclp}("ls", "ls", (\text{char}*) 0);  
\text{char} *\text{args}[] = \{ "ls", (\text{char}*) 0 \};  
\text{execvp}(\text{args}[0], \text{args});  
\]

FULL PATH - FIX

\[
\text{execl}("/bin/ls", "ls", (\text{char}*) 0);  
\text{char} *\text{args}[] = \{ "/bin/ls", (\text{char}*) 0 \};  
\text{execv}(\text{args}[0], \text{args});  
\]
If we want to run user-program what do we need to add to the code to work correctly. Let’s take func1 and assume it’s not exist in the same directory.

**Exercise 6:**

```c
int EXE6() {
    int pid, pid2;
    if ((pid = fork()) == -1) { /* Start a child process. */
        perror("fork error");
        exit(EXIT_FAILURE);
    }
    if (pid != 0) {
        /*Parent Code*/
        if ((pid2 = fork()) == -1) { /* Start a child process. */
            perror("fork error");
            exit(EXIT_FAILURE);
        }
        if (pid2 != 0) {
            printf("Parent Code
");
        } else {
            /*Child Code */
            execvp("func1", "func1", (char*) 0);
            printf("Ending-----> Error
");
        }
    } else {
        /*Child Code */
        char *args[] = { "func1", (char*) 0 };
        execv(args[0], args);
        printf("Ending-----> Error\n");
    }
    return 0;
}
```

**Exercise 6–Questions:**

1. Is exec fail? Which one? What does the code print?
2. How can you fix the problem? Give 2 methods
Exercise 6–Solutions:

1- The two of them are failed, func1 is not exist in any path of $PATH, and not exist in the current directory.
   the output :
   Parent Code
   Ending-----> Error
   Ending-----> Error

2- We can add the path of func1 to parameter $PATH, and use execvp(),execlp() instead.