Lab N°03 : Process Management

Exercise III.1: (Creating Processes With system())

The C **stdlib** library **system()** function is the simplest; it takes a command line, the same as you'd type it at a shell prompt, and executes it.

- The syntax of the **system()** function is : **int system(const char *string)**
- This function creates a new process, which terminates at the end of the command.
- The process calling system is suspended until the end of the command.
- It returns **0** if the command is successfully executed.
- It returns a non-zero value if command execution is not completed.

Q1) Write a program to run the "**ls l /**" shell command using the **system()** function.

Exercise III.2: (The getpid() Function)

Q1) What is the result displayed on the screen when each of the following two programs is run?

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void main() {
        printf("One pid=%d\n",getpid());
        printf("Two pid=%d\n",getpid());
        printf("Three pid=%d\n",getpid());
}
```

Q2) How many processes will be created? Explain

Exercise III.3: (The fork() Primitive)

Q1) What is the result displayed on the screen when each of the following two programs is run?

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
void main() {
        int pid_1;
        printf("One pid=%d pid_1=%d\n",getpid(),pid_1);
        pid_1=fork();
        printf("Two pid=%d pid_1=%d\n",getpid(),pid_1);
        printf("Three pid=%d pid_1=%d\n",getpid(),pid_1);
}
```

Q2) How many processes will be created? Explain

Exercise III.4: (Parent & Child Processes)

Q1) Using this program, explain what the output will be at **LINE A**.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int value = 5;
int main() {
 pid_t pid;
 pid = fork();
 if (pid == 0) {                                  /* Child process */
       value += 15;
       return 0;
 } else if (pid > 0) {                            /* Parent process */
       wait(NULL);
       printf("PARENT: value = %d", value);       /* LINE A */
       return 0;
 }
}
```

Q2) Including the initial parent process, how many processes are created by this program.

Exercise III.5: (Multiple fork())

Consider the following program (**exo5_fork**):

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
main( ) {
        fork();
        fork();
        fork();
}
```

Q1) Without running the program, give the number of processes generated.

Q2) Use the **`getpid()`** and **`getppid()`** system calls to build the tree of generated processes.

Q3) Replace the first call to "**`fork()`**" with a call to "**`execl()`**", which executes the program itself. What happens?

Q4) Replace the second call to "**`fork()`**" with a call to "**`execl()`**", which executes the program itself. What happens now?

Exercise III.6: (Family Tree)

Consider the 3 programs defined below:

```c
#include<unistd.h>
main()
{
  fork() && fork();
}
```

```c
#include<unistd.h>
main()
{
 (fork()|| fork());
}
```

```c
#include<unistd.h>
main()
{
  fork()&&(fork()||fork());
  sleep(5)
}
```

Q1) Without running them, draw the family tree generated by their execution.

Q2) Implement and run these programs to compare results.

Exercise III.7: (fork() & wait())

Q1) What is the result displayed on the screen when each of the following two programs is run?

```c
int main (int argc, char *argv[])
{
   int i;
   for (i =0 ; i <4 ; i++)
       fork();
   printf ("A\n");
}
```

```c
int main (int argc, char *argv[]){
   int i;
   for (i = 0; i < 4; i++)
     if (fork () == 0)
     {
        printf ("A\n");
        exit (0);
     }
   printf ("B\n");
   while (wait(NULL) >= 0);
}
```

Q2) What is the role of the loop: **`while (wait(NULL) >= 0);`**?

Exercise III.8: (Function wait())

Consider the following program:

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main (int argc, char *argv[]){
   pid_t id;
   id = getpid();
   int i;
   for (i = 0; i < 3; i++)
        fork ();
   /*1*/
   if (getpid () == id)
      printf ("The grandparent: %d child of the shell: %d ends\n",
                   getpid(), getppid());
   else
     printf ("The process: %d child  of : %d ends \n", getpid(), getppid());
   return (0);
}
```

Q1) Run this program several times. What do you notice?

Q2) Just before the comment **`/*1*/`**, add the code: **`while (wait(NULL)>= 0);`** and re-execute the program several times.

What do you notice? Deduce the role of the added code.