**Lab 1: Process Handling:**

fork() creates a child process that differs from the parent process only in its PID and PPID, and in the fact that resource utilizations are set to 0. File locks and pending signals are not inherited.

Under Linux, fork() is implemented using copy-on-write pages, so the only penalty that it incurs is the time and memory required to duplicate the parent's page tables, and to create a unique task structure for the child.

**Return Value:**

On success, the PID of the child process is returned in the parent's thread of execution, and a 0 is returned in the child's thread of execution. On failure, a -1 will be returned in the parent's context, no child process will be created, and errno will be set appropriately.

System call fork() is used to create processes. It takes no arguments and returns a process ID. The purpose of fork() is to create a new process, which becomes the child process of the caller. After a new child process is created, both processes will execute the next instruction following the fork() system call. Therefore, we have to distinguish the parent from the child. This can be done by testing the returned value of fork():

- If fork() returns a negative value, the creation of a child process was unsuccessful.
- fork() returns a zero to the newly created child process itself.
- fork() returns a positive value, the process ID of the child process, to the parent. The returned process ID is of type pid_t defined in sys/types.h. Normally, the process ID is an integer. Moreover, a process can use function getpid() to retrieve the process ID assigned to this process.

Therefore, after the system call to fork(), a simple test can tell which process is the child. Please note that UNIX will make an exact copy of the Parent's address space and give it to the child. Therefore, the parent and child processes have separate address spaces.

The fork() function is used to create a new process from an existing process. The new process is called the child process, and the existing process is called the parent. You can tell which is the child's pid returned to him, but the child gets 0 returned to him.

The execvp(), execlp() family of commands can be used to execute an application from a process. The system call execvp(),execlp() replaces the executing process by a new process image which executes the application specified as its parameter. Arguments can also be specified.

***Sample Program:***

1. Check how many process id are printed by executing this program

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

int main() {

int q; printf("%d %d\n",getpid(),getppid());

q=fork(); printf("%d %d\n",getpid(),getppid(),q);

return 0; }
```

2. How many X will be printed by following program ?

```
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int main(){

p=fork(); q=fork();

If(p=0)
fork();

fork(); printf("X");

return 0;}
```

3. How many X and Y are printed by the following program?

```
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int main() {

p=fork(); fork();

q=fork();

if(p==q)

printf("X");

else

printf("Y"); return 0; }
```

4. The following program creates process B and C. The parent of B is A and the parent of C is B.

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

int main() {

q=fork(); if(q=0) fork();

printf(“%d %d\n”,getpid(),getppid());

sleep(1); return 0; }
```

5. How many times the “good morning INDIA” will be printed by the following program.

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

int main() {

fork();
fork();
fork();

printf(“good morning INDIA”);

return 0;

}
```

6.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/ipc.h>

main()
{
char *arglist[3];
arglist[0] = (char *)malloc(4*sizeof(char));
strcpy(arglist[0],"cal");
arglist[1] = (char *)malloc(5*sizeof(char)); strcpy(arglist[1],"2012");
arglist[2] = NULL;

        /* Call execvp */
        execvp("cal",arglist);
/* The execvp() system call does not return. Note that the
           following statement will not be executed.
        */

        printf("This statement is not executed if execvp succeeds.\n");
}
```

7.
```
#include <stdio.h>
#include <unistd.h>
#include <sys/ipc.h>

main()
{
        execlp("cal","cal","2012",NULL);
        /* The execlp() system call does not return. Note that the
           following statement will not be executed */
        printf("This statement is not executed if execlp succeeds.\n");
}
```

**Assignments:**

1. Write a program which creates 10 additional process. Every process prints its id. Use only 4 fork()s.

2. Write a program to create 6 child process by using only one fork (). Print the no. of childs with their process id.

3. Write a program which creates process B,C,D and E. The parent of the B,C and D is A. The parent of E is B.

4. Write a program with execvp and execlp, which lists all the files with all the attributes in the current folder.