National University of Computer and Emerging Sciences

Lab Manual 04
Operating Systems

(Lab Topic : System Calls)

Department of Computer Science
FAST-NU, Lahore, Pakistan

exec()

The exec family of functions replaces the current running process with a new process. It can be used to run a C program by using another C program. It comes under the header file unistd.h.

There are many members in the exec family i.e.

- execl ()
- execlp ()
- execv ()
- execvp ()
- execlpe ()
- execvpe ()

1. **execl** *int execl (const char * path, const char * arg, … , NULL)*

   Parameters:
   1. Takes the path of the executable binary file.
   2. Takes arguments (can take more than one argument)
   3. Takes NULL

If an error occurs it returns -1 else it returns nothing.

2. **execlp** *int execlp (const char * file, const char * arg, … , NULL);*

   Parameters:
   4. Takes the executable binary file.
   5. Takes arguments (can take more than one argument)
   6. Takes NULL

If an error occurs it returns -1 else it returns nothing.

1. **execvp**
   *int execl (const char * file, char * const argv[]);*

   Parameters:
   a. Takes executable file.

Takes arguments (we can pass all arguments in a NULL-terminated array **argv**)

We will be seeing "**excelp**" for now.

The **execlp() function** replaces the current process image with a new process image specified by file. The new image is constructed from a regular, executable file called the new process image file. No return is made because the calling process image is replaced by the new process image. **Syntax**

```
#include <unistd.h>

int execlp(
    const char *file,
    const char *arg0,
    const char *arg1,
    …
    const char *argn,
    NULL
);
```

Arguments

file

Used to construct a pathname that identifies the new process image file. If the file argument contains a slash character, the file argument is used as the pathname for the file.

arg0, …, argn

Pointers to NULL-terminated character strings. These strings constitute the argument list available to the new process image. You must terminate the list with a NULL pointer. **The arg0 argument must point to a filename that's associated with the process being started and cannot be NULL.**

Example

Program to create image for

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char *arg[])
{
    printf(arg[1]);
    return 0;
}
```

Execlp program to run the Process

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int main(int argc, char *arg[])
{
    int x = execlp("./a.out", "a.out", "Hello", NULL);
    return 0;
}
```

You can also use execlp to run unix system commands such as ls , cp , mkdir etc. Example is shown below

Example: creating a new directory named Lab using execlp

```c
#include <stdlib.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
    // creating a new directory named Lab using execlp
    int x = execlp("mkdir", "mkdir", "LabTestFolder", NULL);
    return 0;
}
```

Fork and Exec together

Example example.c

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char *argv[])
{
   printf("PID of example.c = %d\n", getpid());
   pid_t p;
   p = fork();
   if(p==-1)
   {
      printf("There is an error while calling fork()");
   }
   if(p==0)
   {
   printf("We are in the child process\n");
   printf("Calling hello.c from child process\n");
   char *args[] = {"Hello", "C", "Programming", NULL};
   execv("./hello", args);
   }
   else
   {
      printf("We are in the parent process");
   }
   return 0;
}
```

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
   printf("We are in Hello.c\n");
   printf("PID of hello.c = %d\n", getpid());
   return 0;
}
```

OUTPUT:

```
PID of example.c = 4790
We are in Parent Process
We are in Child Process
Calling hello.c from child process
We are in hello.c
PID of hello.c = 4791
```

Make a folder named after your rollnumber on the Desktop. Make separate folders within this folder for each task.

Task 1: execv ( ) (2 marks)

- Write a C program called **main.c** and replace it using **execv** with another program named **my_info.c**.
- The second process should print the name, roll number, and your semester number. Explain verbally to the instructor the working of the function of **execv**.
- Print **process ID** before calling execv in **main.c**
- Also print **process ID** in **my_info.c** before printing your own information.

Task 2: execlp ( ) and execvp ( ) (5 marks)

- With **execlp** create a directory named **demo_folder** using **mkdir** command.
- With **execvp** create 2 new files **file1.txt** and **file2.txt** in the **demo_folder** you have created using **touch** command. Syntax of touch command:

```
touch filename.txt
```

- You can pass **multiple filenames** as arguments to touch command to create multiple files.
- List the files in the directory **demo_folder** using **ls** command, and call ls command using **execvp**.
- Now you have to remove the **demo_folder** using **rm** command. This will be called using **execvp**. Note: Your will have to pass extra arguments to rm command i.e. **-rf** to remove a directory.

Hint: Use fork to create 4 children for all 4 tasks above. You will have to use sleep() and wait(NULL) appropriately to sync the processes, otherwise you might face errors or unwanted results.

Task 3: execv ( ) (3 marks)

- Create a program that makes a child and that child replaces itself via **execv** with another program called **Fibonacci**.

- This Fibonacci program takes a command line argument **n.**
- You have to print first **n** numbers from the Fibonacci series in that new program.