NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES
ISLAMABAD

OPERATING SYSTEMS LAB SPRING 2022

# Lab Manual 03
# Exec System Call

*A family of six functions*

***Madiha Umar***
madiha.umar@nu.edu.pk
***Maryam Shahbaz***
maryam.shahbaz@nu.edu.pk
***Naveed Khurshid***
naveed.khurshid@nu.edu.pk

## 1 EXEC() SYSTEM CALL

Forking provides a way for an existing process to start a new one, but what about the case where the new process is not part of the same program as parent process? This is the case in the shell; when a user starts a command it needs to run in a new process, but it is unrelated to the shell. This is where the exec system call comes into play. exec will replace the contents of the currently running process with the information from a program binary.

1. The *exec* function is actually a family of six functions each with slightly different calling conventions and use.
2. Like *fork*, *exec* is declared in <unistd.h>.
3. *exec* completely replaces the calling process's image with that of the program being executed.
4. *fork* creates a new process, and so generates a new PID. *exec* initiates a new program, replacing the original process. Therefore, the PID of an *execed* process doesn't change.

5. If successful, the *exec* calls do not return as the calling image is lost.

## 2 Prototypes Of Exec()

Prototypes of exec() are:

```
int execl (const char *path, const char *arg, ...);

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

int execv(const char *path, char *const argv[]);

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

int execle(const char *path, const char *arg, char *const envp[]);

int execve(const char *path, const char *argv[], char *const envp[]);
```

## 3 Naming Convention of Exec()

The naming convention for the "exec" system calls reflects their functionality:

1. The function starts with *exec* indicating that one of the function from *exec* family is called.
2. The next letter in the call name indicates if the call takes its arguments in a "list format" (i.e. literally specified as a series of arguments) or as a pointer to an "array of arguments".
   (a) The functions execl, execlp, and execle require each of the command-line arguments to the new program to be specified as separate arguments, terminated by a NULL pointer.
   (b) For the other functions (execv, execvp, execve), we have to build an array of pointers to the arguments.
3. The presence of a "p" indicates the current PATH string should be used when the system searches for executable files. (If the executable file is script file, the shell is invoked to execute the script. The shell is then passed the specified argument information).
4. The functions whose name end with an *e* allow to pass array to set new environment. Otherwise the calling process copy the existing environment for the new program.

| Letters | Explanation |
|---|---|
| l | argv is specified as a list of arguments. |
| v | argv is specified as a vector (array of character pointers). |
| e | environment is specified as an array of character pointers. |
| p | PATH is searched for command, and command can be a shell program |

## 4 EXECL()

```
int execl(const char *path, const char *arg, ...)
```

1. Takes the path name of an executable program (binary file) as its first argument.
2. The rest of the arguments are a list of command line arguments to the new program.
3. The list is terminated with a null pointer.
4. Examples:

```
execl("/bin/ls", "ls", "-l", "/usr", NULL);
execl("/home/lab/Desktop/a.out","./a.out","arg 1","arg 2","arg n",NULL)
```

### EXAMPLE 01

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

int main(){
        pid_t childpid = fork();
        if(childpid==0){
                printf("I am a child proce with pid = %d \n",getpid());
                printf("The next statement is execl and ls will run \n");
                execl("/bin/ls","ls","-l","/usr",NULL);
                printf("Execl failed");
        }
        else if(childpid>0){
                wait(NULL);
                printf("\n I am parent process with pid = %d
                                        and finished waiting \n",getpid());
        }
        return 0;
}
```

**EXAMPLE 02**

Apart from executing the shell commands we can run other executables as well. In this example we will write a code in *C* and create an executable named as *execExam*. Given below is a simple code in *C* that prints the arguments passed to it through terminal.

```
#include <stdio.h>
void main( int argc , char *argv[] ) {
   int i ;
   for( i=0 ; i < argc ; i++ ) {
      printf( "Argument %d : %s\n" , i , argv [ i ] ) ;
   }
}
```

Now compile the code and create an object file *execExam* :

```
gcc -o execExam filename.c
```

Following program will now create a new process using fork system call. Then the child process will run the executable of the above program using *execl()*.

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

int main(){
        pid_t childpid = fork();
        if(childpid==0){
                printf("I am a child proce with pid = %d \n",getpid());
                printf("The next statement is execl and execExam executable
                                        will run \n");
                execl("/home/sarosh/Desktop/execExam","ls","-l","/usr",NULL);
                printf("Execl failed");
        }
        else if(childpid>0){
                wait(NULL);
                printf("\n I am parent process with pid = %d and
                                        finished waiting \n",getpid());
        }
        return 0;
}
```

## 5 EXECLP()

```
int execlp(const char *file, const char *arg, ...)
```

1. Same as execl(), except that the program name doesn't have to be a full path name and it can be a shell program instead of an executable module. The program directory should be in your PATH variable.
2. The rest of the arguments are a list of command line arguments to the new program.
3. The list is terminated with a null pointer.
4. Example:

```
execlp("ls", "ls", "-l", "/usr", NULL);
execlp("cat", "cat", "filename", NULL);
execlp("./a.out", "a.out", "arg 1", NULL);
```

### Example 01

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

int main(){
        pid_t childpid = fork();
        if(childpid==0){
                printf("I am a child proce with pid = %d \n",getpid());
                printf("The next statement is execlp and ls will run \n");
                execlp("ls","ls","-l","/usr",NULL);
                printf("Execl failed");
        }
        else if(childpid>0){
                wait(NULL);
                printf("\n I am parent process with pid = %d and
                                finished waiting \n",getpid());
}
return 0;
}
```

## 6 EXECV()

```
int execv(const char *path, char *const argv[]);
```

1. Takes the path name of an executable program (binary file) as it first argument.
2. The second argument is a pointer to a list of character pointers (like argv[]) that is passed as command line arguments to the new program.
3. The list is terminated with a null pointer.

4. Example:

   ```
   char *args[] = { "cat", "filename1", NULL };
   execv("/bin/cat", args);

   char *args[] = { "./a.out", "arg1", "arg2", NULL };
   execv("/home/lab/Desktop/a.out",args);
   ```

**EXAMPLE 01**

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

int main(){
        pid_t childpid = fork();
        if(childpid==0){
                printf("I am a child proce with pid = %d \n",getpid());
                printf("The next statement is execv and ls will run \n");
                char* argv[]={"ls","-l","/usr",NULL};
                execv("/bin/ls",argv);
                printf("Execl failed");
        }
        else if(childpid>0){
                wait(NULL);
                printf("\n I am parent process with pid = %d and
                                finished waiting \n",getpid());
        }
        return 0;
}
```

## 7 EXECVP()

```
int execvp(const char *file, char *const argv[]);
```

1. Same as execv(), except that the program name doesn't have to be a full path name, and it can be a shell program instead of an executable module.
2. The second argument is a pointer to a list of character pointers (like argv[]) that is passed as command line arguments to the new program.
3. The list of character pointer should be terminated with a null pointer.
4. Example:

   ```
   char *args[] = { "cat", "f1", "f2", NULL };
   execvp("cat", args);
   ```

**EXAMPLE 01**

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

int main(){
        pid_t childpid = fork();
        if(childpid==0){
                printf("I am a child proce with pid = %d \n",getpid());
                printf("The next statement is execv and ls will run \n");
                char* argv[]={"ls","-l","/usr",NULL};
                execvp("ls",argv);
                printf("Execl failed");
        }
        else if(childpid>0){
                wait(NULL);
                printf("\n I am parent process with pid = %d and
                                finished waiting \n",getpid());
}
return 0;
}
```

## 8 EXECLE()

```
int execle(const char *path, const char *arg, char *const envp[])
```

1. Same as execl(), except that the end of the argument list is followed by a pointer to a null-terminated list of character pointers that is passed as the environment of the new program.
2. Example:

```
char *env[] = { "export TERM=vt100", "PATH=/bin:/usr/bin", NULL };
execle("/bin/cat", "cat", "f1", NULL, env);
```

## 9 EXECVE()

```
int execve(const char*path, const char *argv[], char *const envp[]);
```

1. Same as execv(), except that a third argument is given as a pointer to a list of character pointers (like argv[]) that is passed as the environment of the new program.
2. Example:

```
char *env[]={ "export TERM=vt100", "PATH=/bin:/usr/bin", NULL};
char *args[]={ "cat", "f1", NULL };
execve("/bin/cat", args, env);
```