

Exec System Call in C

The exec family has many functions in C. These C functions are basically used to run a system command in a separate process that the main program and print the output.

In this article, I am going to talk about the exec family of functions and show you how to use each one of these exec family function in C. So, let's get started.

C System Functions in Exec Family:

The exec function families are defined in the header **unistd.h**. So, you must use this header on the C program where you want to use these functions. The available exec functions along with their function parameters are given below:

- `int execl(const char *path, const char *arg, ..., NULL);`
- `int execlp(const char *file, const char *arg, ..., NULL);`
- `int execv(const char *path, char *const argv[]);`
- `int execvp(const char *file, char *const argv[]);`
- `int execl(const char *path, const char *arg, ..., NULL, char * const envp[]);`
- `int execve(const char *file, char *const argv[], char *const envp[]);`

Let's see what each of these functions do and how to use them.

execl() System Function:

In `execl()` system function takes the path of the executable binary file (i.e. **/bin/ls**) as the first and second argument. Then, the arguments (i.e. - **lh**, **/home**) that you want to pass to the executable followed by **NULL**. Then `execl()` system function runs the command and prints the output. If any error occurs, then `execl()` returns -1. Otherwise, it returns nothing.

Syntax:

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

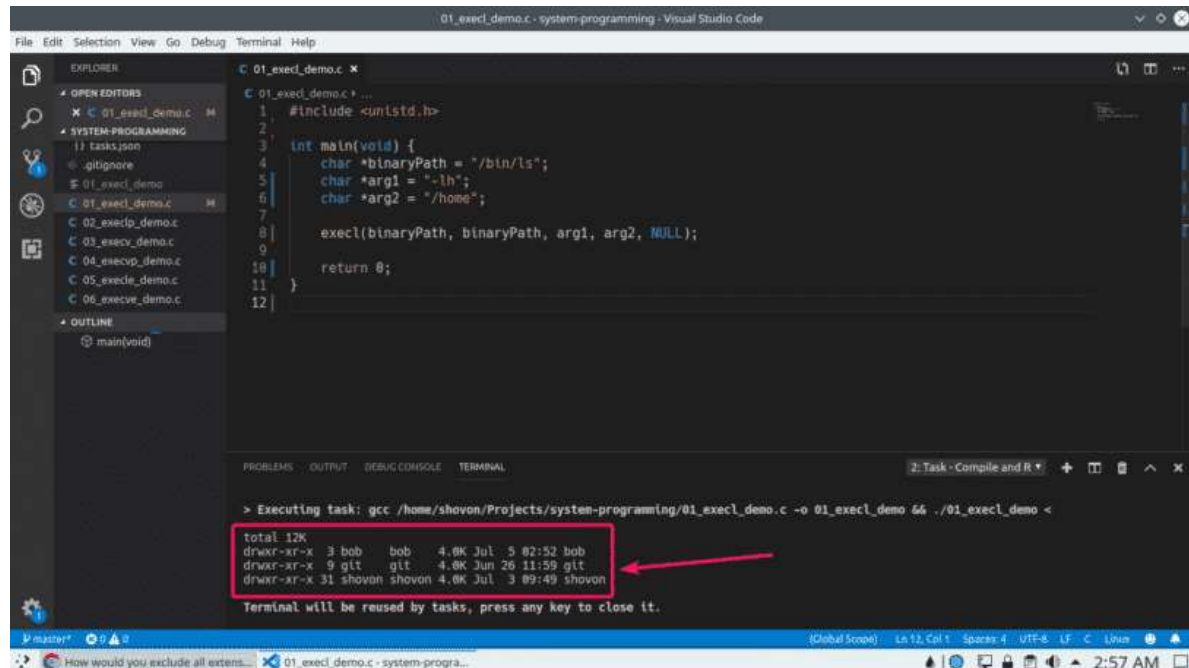
An example of the `execl()` system function is given below:

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

```
int main(void) {  
    char *binaryPath = "/bin/ls";  
    char *arg1 = "-lh";  
    char *arg2 = "/home";  
  
    execl(binaryPath, binaryPath, arg1, arg2, NULL);  
}
```

```
return 0;
}
```

I ran the **ls -lh /home** command using `execl()` system function. As you can see, the correct result is displayed.



```
01_execl_demo.c - system-programming - Visual Studio Code
File Edit Selection View Go Debug Terminal Help

EXPLORER
  OPEN EDITORS
    01_execl_demo.c
  SYSTEM-PROGRAMMING
    tasks.json
    .gitignore
    01_execl_demo
    01_execl_demo.c
    02_execl_demo.c
    03_execl_demo.c
    04_execl_demo.c
    05_execl_demo.c
    06_execl_demo.c
  OUTLINE
    main(void)

C 01_execl_demo.c
1 #include <unistd.h>
2
3 int main(void) {
4     char *binaryPath = "/bin/ls";
5     char *arg1 = "-lh";
6     char *arg2 = "/home";
7
8     execl(binaryPath, binaryPath, arg1, arg2, NULL);
9
10    return 0;
11 }
12

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
2 Task - Compile and R
> Executing task: gcc /home/shovon/Projects/system-programming/01_execl_demo.c -o 01_execl_demo && ./01_execl_demo <
total 12K
drwxr-xr-x 3 bob bob 4.0K Jul 5 02:52 bob
drwxr-xr-x 9 git git 4.0K Jun 26 11:59 git
drwxr-xr-x 31 shovon shovon 4.0K Jul 3 09:49 shovon
Terminal will be reused by tasks, press any key to close it.
```

execlp() System Function:

`execlp()` does not use the **PATH** environment variable. So, the full path of the executable file is required to run it with `execlp()`. `execlp()` uses the **PATH** environment variable. So, if an executable file or command is available in the **PATH**, then the command or the filename is enough to run it, the full path is not needed.

Syntax:

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

We can rewrite the `execl()` example using the `execlp()` system function as follows:

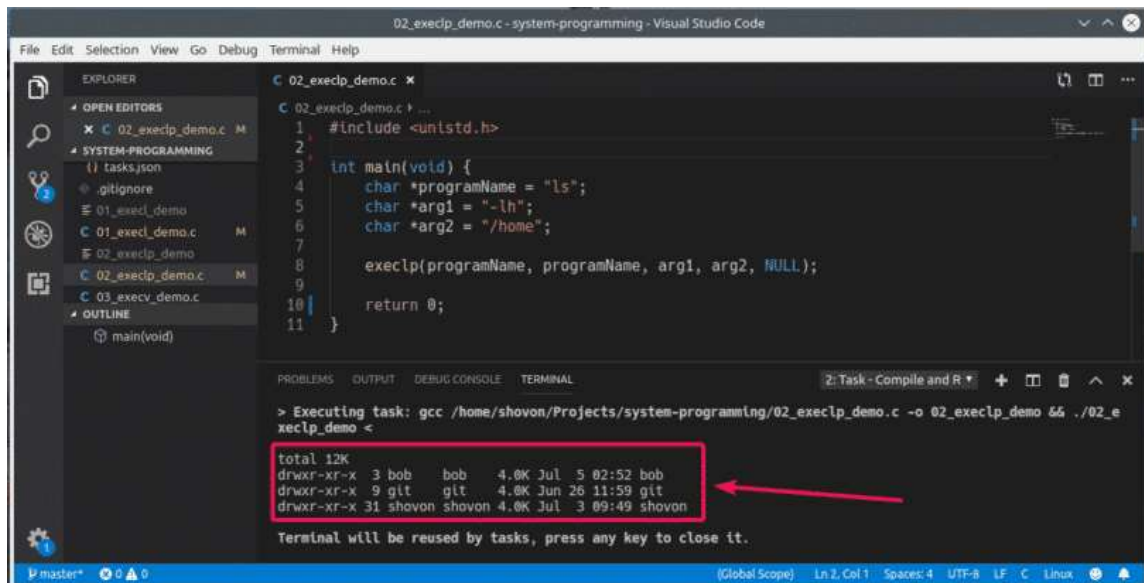
```
#include <unistd.h>
```

```
int main(void) {
    char *programName = "ls";
    char *arg1 = "-lh";
    char *arg2 = "/home";

    execlp(programName, programName, arg1, arg2, NULL);

    return 0;
}
```

I only passed the command name **ls**, not the full path **/bin/ls**. As you can see, I got the same output as before.



```
#include <unistd.h>

int main(void) {
    char *programName = "ls";
    char *arg1 = "-lh";
    char *arg2 = "/home";

    execlp(programName, programName, arg1, arg2, NULL);

    return 0;
}
```

```
> Executing task: gcc /home/shovon/Projects/system-programming/02_execlp_demo.c -o 02_execlp_demo && ./02_execlp_demo <
total 12K
drwxr-xr-x  3 bob   bob   4.0K Jul  5 02:52 bob
drwxr-xr-x  9 git    git   4.0K Jun 26 11:59 git
drwxr-xr-x 31 shovon shovon 4.0K Jul  3 09:49 shovon
```

execv() System Function:

In `execl()` function, the parameters of the executable file is passed to the function as different arguments. With `execv()`, you can pass all the parameters in a NULL terminated array **argv**. The first element of the array should be the path of the executable file. Otherwise, `execv()` function works just as `execl()` function.

Syntax:

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

We can rewrite the `execl()` example as follows:

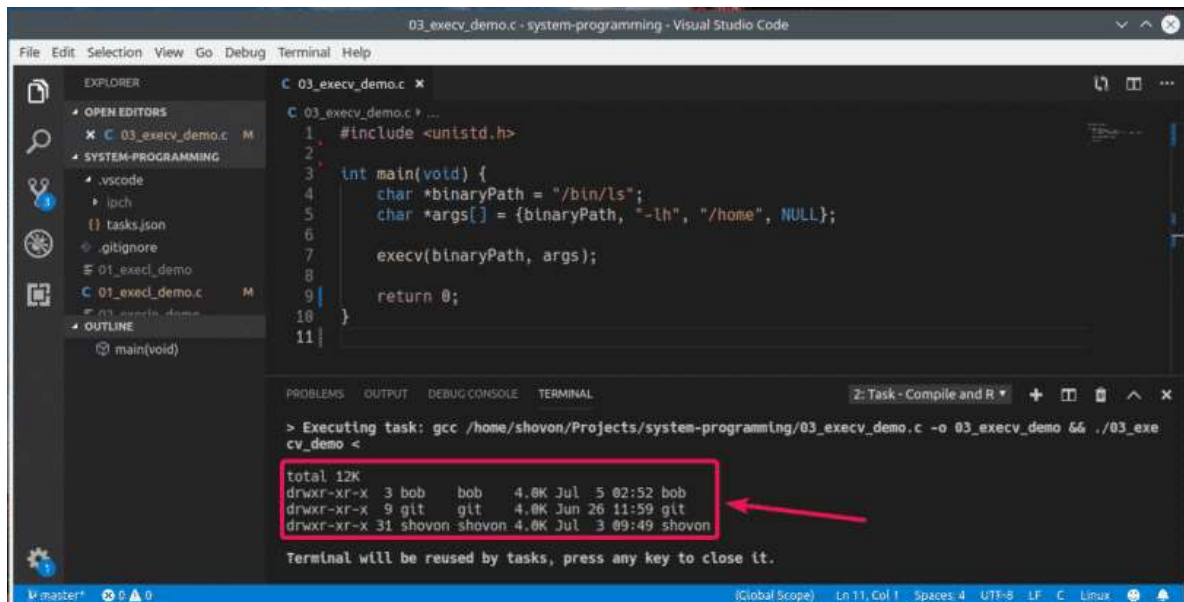
```
#include <unistd.h>

int main(void) {
    char *binaryPath = "/bin/ls";
    char *args[] = {binaryPath, "-lh", "/home", NULL};

    execv(binaryPath, args);

    return 0;
}
```

As you can see, I am getting the correct output.



```
03_execv_demo.c - system-programming - Visual Studio Code
File Edit Selection View Go Debug Terminal Help

EXPLORER
  OPEN EDITORS
    03_execv_demo.c M
  SYSTEM-PROGRAMMING
    .vscode
      lpc
    tasks.json
    .gitignore
    01_execd_demo
    01_execd_demo.c M
    03_execv_demo
    OUTLINE
      main(void)

C 03_execv_demo.c
1 #include <unistd.h>
2
3 int main(void) {
4     char *binaryPath = "/bin/ls";
5     char *args[] = {binaryPath, "-lh", "/home", NULL};
6
7     execvp(binaryPath, args);
8
9     return 0;
10 }
11

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
> Executing task: gcc /home/shovon/Projects/system-programming/03_execv_demo.c -o 03_execv_demo 66 ./03_execv_demo <
total 12K
drwxr-xr-x 3 bob bob 4.0K Jul 5 02:52 bob
drwxr-xr-x 9 git git 4.0K Jun 26 11:59 git
drwxr-xr-x 31 shovon shovon 4.0K Jul 3 09:49 shovon
Terminal will be reused by tasks, press any key to close it.
```

execvp() System Function:

Works the same way as `execv()` system function. But, the `PATH` environment variable is used. So, the full path of the executable file is not required just as in `execclp()`.

Syntax:

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

We can rewrite the `execv()` example as follows:

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

```
int main(void) {
    char *programName = "ls";
    char *args[] = {programName, "-lh", "/home", NULL};

    execvp(programName, args);

    return 0;
}
```

As you can see, the correct output is displayed.

The screenshot shows a Visual Studio Code editor with a C file named `04_execvp_demo.c`. The code defines a `main` function that calls `execvp` with `programName = "ls"` and an array of arguments `args = {"-lh", "/home", NULL}`. The terminal window shows the command `gcc /home/shovon/Projects/system-programming/04_execvp_demo.c -o 04_execvp_demo && ./04_execvp_demo` and its output, which is a directory listing of the `/home` directory. A red box highlights the output, and a red arrow points to it.

```
#include <unistd.h>

int main(void) {
    char *programName = "ls";
    char *args[] = {programName, "-lh", "/home", NULL};

    execvp(programName, args);

    return 0;
}
```

```
> Executing task: gcc /home/shovon/Projects/system-programming/04_execvp_demo.c -o 04_execvp_demo && ./04_execvp_demo <
total 12K
drwxr-xr-x  3 bob   bob   4.0K Jul  5 02:52 bob
drwxr-xr-x  9 git   git   4.0K Jun 26 11:50 git
drwxr-xr-x 31 shovon shovon 4.0K Jul  3 09:49 shovon
```

execle() System Function:

Works just like `execl()` but you can provide your own environment variables along with it. The environment variables are passed as an array **envp**. The last element of the **envp** array should be `NULL`. All the other elements contain the key-value pairs as string.

Syntax:

```
int execl(const char *path, const char *arg, ..., NULL, char * const envp[] );
```

An example of the `execle()` system function is given below:

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

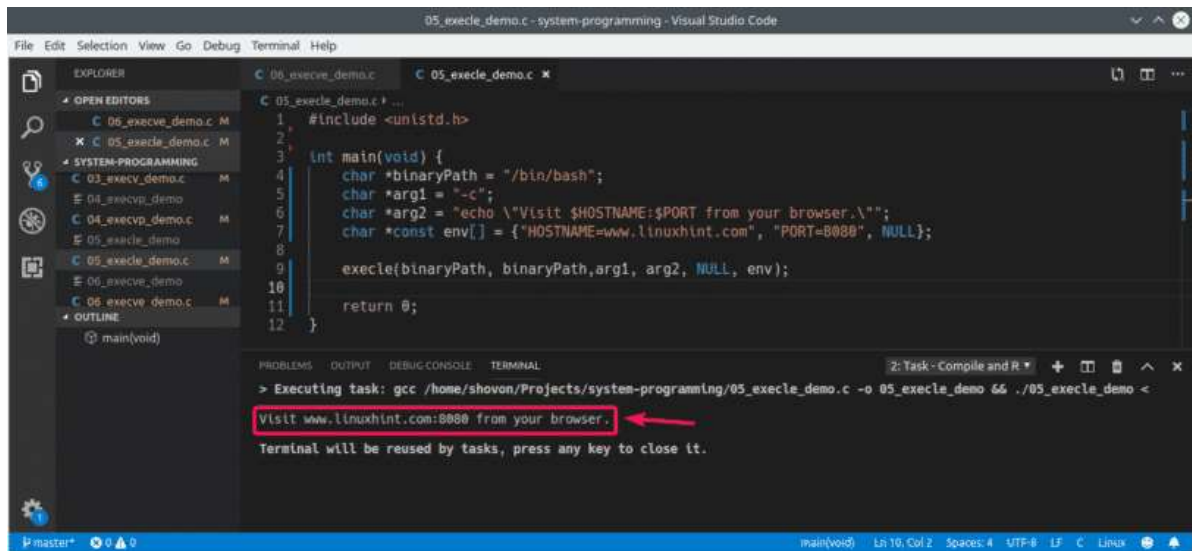
```
int main(void) {
    char *binaryPath = "/bin/bash";
    char *arg1 = "-c";
    char *arg2 = "echo \"Visit $HOSTNAME:$PORT from your browser.\"";
    char *const env[] = {"HOSTNAME=www.linuxhint.com", "PORT=8080", NULL};
}
```

```
execle(binaryPath, binaryPath, arg1, arg2, NULL, env);
```

```
return 0;
```

```
}
```

I passed two environment variables **HOSTNAME** and **PORT** to the `execle()` function. As you can see, I can access them from the executable `/bin/bash`.



```
05_execl_demo.c - system-programming - Visual Studio Code
File Edit Selection View Go Debug Terminal Help

EXPLORER
  OPEN EDITORS
    C 06_execl_demo.c M
    C 05_execl_demo.c M
  SYSTEM-PROGRAMMING
    C 03_execl_demo.c M
    C 04_execl_demo.c M
    C 05_execl_demo.c M
    C 06_execl_demo.c M
  OUTLINE
    main(void)

C 05_execl_demo.c
1 #include <unistd.h>
2
3 int main(void) {
4     char *binaryPath = "/bin/bash";
5     char *arg1 = "-c";
6     char *arg2 = "echo \"Visit $HOSTNAME:$PORT from your browser.\"";
7     char *const env[] = {"HOSTNAME=www.linuxhint.com", "PORT=8080", NULL};
8
9     execl(binaryPath, binaryPath, arg1, arg2, NULL, env);
10
11     return 0;
12 }
```

```
2: Task - Compile and R...
> Executing task: gcc /home/shovon/Projects/system-programming/05_execl_demo.c -o 05_execl_demo 66 ./05_execl_demo <
Visit www.linuxhint.com:8080 from your browser.
Terminal will be reused by tasks, press any key to close it.
```

execve() System Function:

Just like `execl()` you can provide your own environment variables along with `execve()`. You can also pass arguments as arrays as you did in `execv()`.

Syntax:

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

The `execl()` example can be rewritten as follows:

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

```
int main(void) {
    char *binaryPath = "/bin/bash";
    char *const args[] = {binaryPath, "-c", "echo \"Visit $HOSTNAME:$PORT
    from your browser.\"";
    char *const env[] = {"HOSTNAME=www.linuxhint.com", "PORT=8080", NULL};

    execve(binaryPath, args, env);

    return 0;
}
```

As you can see, we get the same output as in the `execl()` example.

The screenshot shows the Visual Studio Code interface with a C program named `06_execve_demo.c` open in the editor. The program uses `execve` to execute `/bin/bash` with specific arguments and environment variables. The terminal at the bottom shows the command to compile and run the program, and the output of the executed process.

```
06_execve_demo.c - system-programming - Visual Studio Code
File Edit Selection View Go Debug Terminal Help

EXPLORER
  OPEN EDITORS
    C 06_execve_demo.c M
  SYSTEM-PROGRAMMING
    05_execve_demo
    C 05_execve_demo.c M
    06_execve_demo
    C 06_execve_demo.c M
    07_fork
    C 07_fork.c
    08_fork.c
  OUTLINE
    main(void)

C 06_execve_demo.c
1 #include <unistd.h>
2
3 int main(void) {
4     char *binaryPath = "/bin/bash";
5     char *const args[] = {binaryPath, "-c", "echo \\Visit $HOSTNAME:$PORT from your browser.\\", NULL};
6     char *const env[] = {"HOSTNAME=www.linuxhint.com", "PORT=8888", NULL};
7
8     execve(binaryPath, args, env);
9
10    return 0;
11 }
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
> Executing task: gcc /home/shovon/Projects/system-programming/06_execve_demo.c -o 06_execve_demo && ./06_execve_demo <
Visit www.linuxhint.com:8888 from your browser.
Terminal will be reused by tasks, press any key to close it.
```

So, that's how you use the exec function family in C for system programming in Linux. Thanks for reading this article.