**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 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 execle(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.

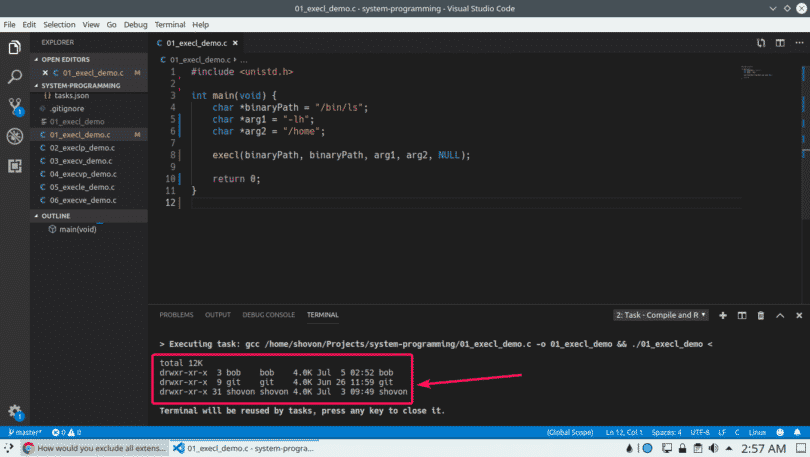
**Syntax:**

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.



**execlp() System Function:**

execl() does not use the **PATH** environment variable. So, the full path of the executable file is required to run it with execl(). 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.

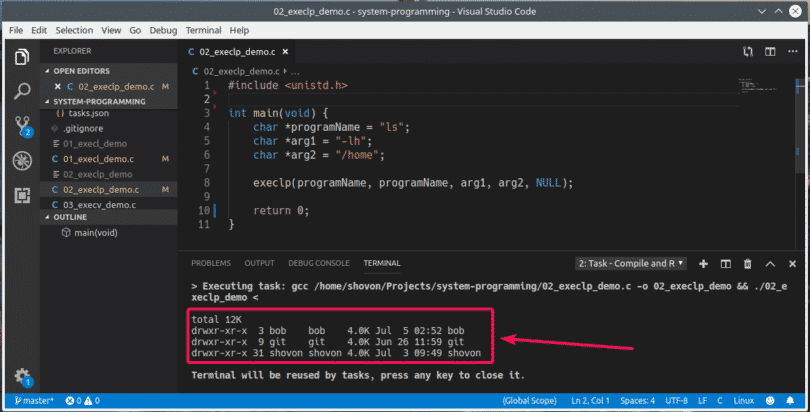
**Syntax:**

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.



**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.

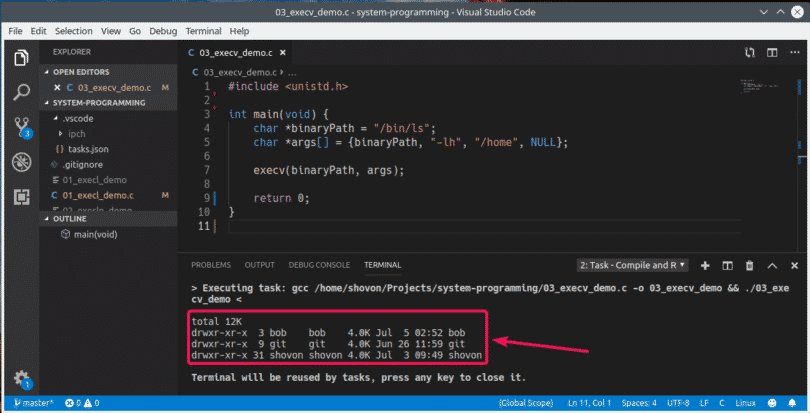
**Syntax:**

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.



**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 execlp().

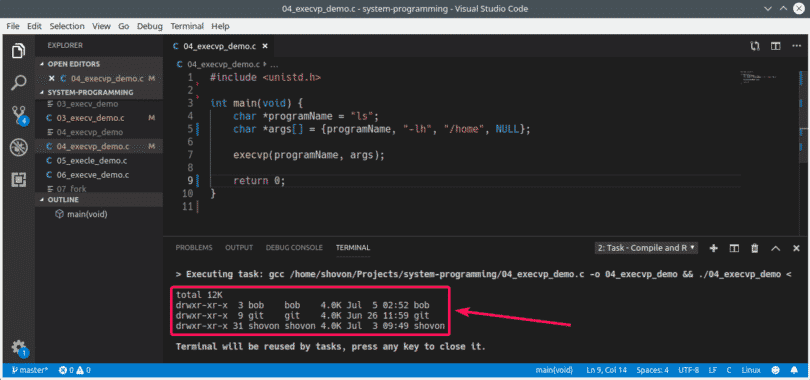
**Syntax:**

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.



**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.

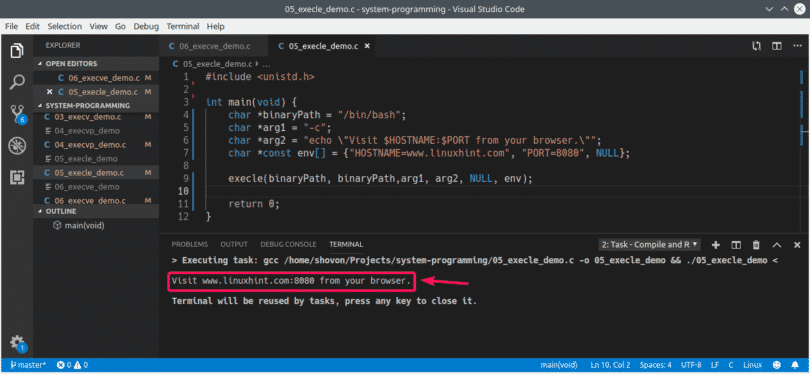
**Syntax:**

int execle(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**.



**execve() System Function:**

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

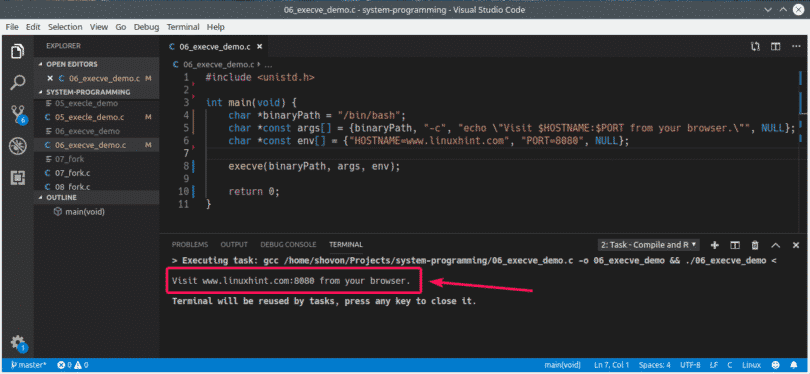
**Syntax:**

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

The execle() 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."", NULL};  
  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 execle() example.



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