CONTENTS

RELATED

Creating a Queue in C

View

The foreach loop in C++

View

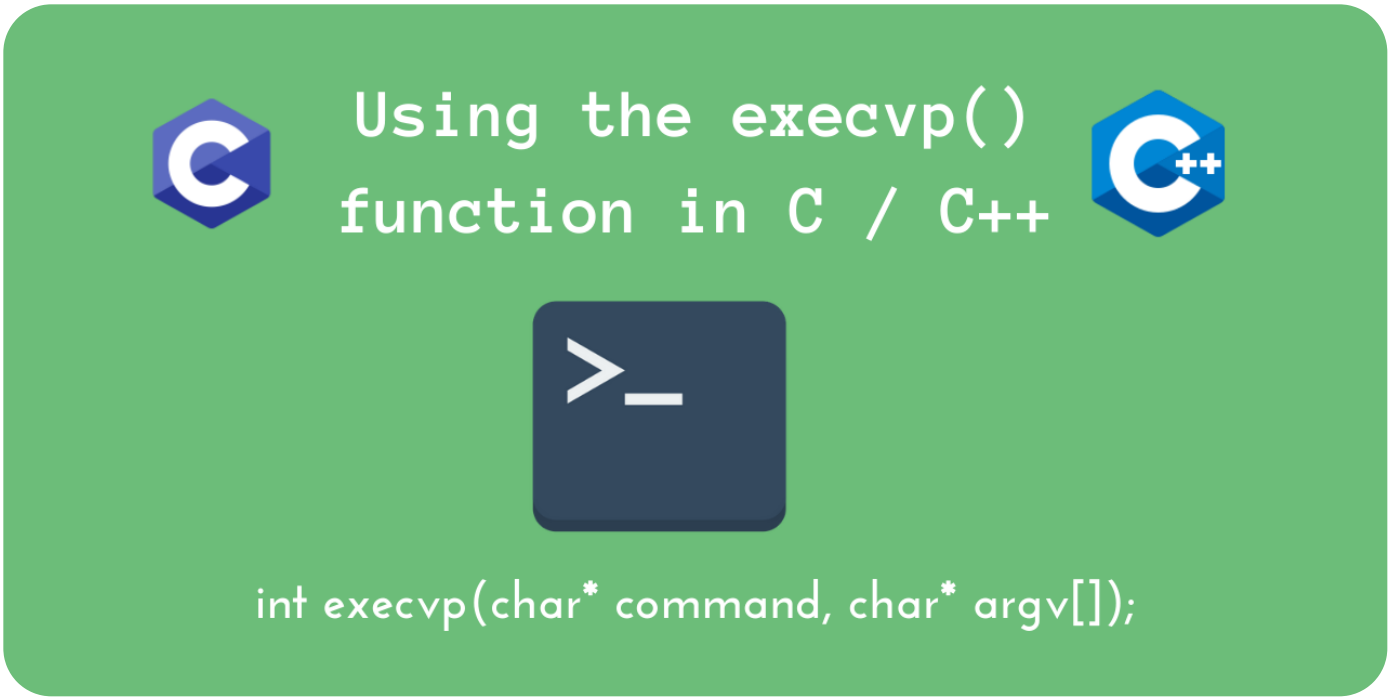
// Tutorial //

How to use the execvp() function in C/C++

Published on August 3, 2022

C++ C Programming

By Vijaykrishna Ram



While we believe that this content benefits our community, we have not yet thoroughly reviewed it. If you have any suggestions for improvements, please let us know by clicking the “report an issue“ button at the bottom of the tutorial.

In this article, we'll take a look at using the execvp() function in C / C++.

In UNIX, the execvp() function is very useful if you want to run another program using our C program.

NOTE: This function is applicable only to UNIX based Operating Systems. It doesn't work on Windows

Let's take a look at executing UNIX commands from our program, using illustrative examples!

Basic Syntax of execvp()

This function takes in the name of the UNIX command to run, as the first argument.

This is there in the `<unistd.h>` header file, so we must include it in our program.

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

Copy

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

Here, we refer to a “command” as any binary executable file that is a part of the `PATH` environment variable. So, if you want to run custom programs, make sure that you add it to your `PATH` variable!

The second argument (`argv`) represents the list of arguments to `command`. This is an array of `char*` strings.

Here, `argv` contains the complete command, along with it’s arguments.

For example, the below array follows the format of `argv`.

```
char* argument_list[] = {"ls", "-l", NULL}; // NULL terminated array of char* st Copy

// Ok! Will execute the command "ls -l"
execvp("ls", argument_list);
```

This array **MUST** be `NULL` terminated, i.e, the last element of `argv` must be a `NULL` pointer.

What happens to our C program now?

This function will give the control of the current process (C program) to the command. So, the C program is instantly replaced with the actual command.

So, anything that comes after `execvp()` will NOT execute, since our program is taken over completely!

However, if the command fails for some reason, `execvp()` will return -1.

So, whenever you use `execvp()`, if you want to maintain your C program, you generally use `fork()` to first spawn a new process, and then use `execvp()` on that new process.

This is called the “fork-exec” model, and is the standard practice for running multiple processes using C.

Let’s now look at some examples, to understand this function better. We’ll also be using `fork()` along with `execvp()`, so that we can still have our C program with us!

Using execvp() in C / C++ - Some Examples

If you want to see what exactly happens if you try to use `execvp()` without spawning a new process using `fork()`. the below program shows this.

We'll be executing "ls -l" from our C program.

Notice that the `printf()` statement after `execvp()` is NOT executed, since the other process has taken control!

Copy

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

int main() {
    char* command = "ls";
    char* argument_list[] = {"ls", "-l", NULL};

    printf("Before calling execvp()\n");

    // Calling the execvp() system call
    int status_code = execvp(command, argument_list);

    if (status_code == -1) {
        printf("Process did not terminate correctly\n");
        exit(1);
    }

    printf("This line will not be printed if execvp() runs correctly\n");

    return 0;
}
```

Output

Copy

```
Before calling execvp()
total 3
-rwxrwxrwx 1 user user 22088 May 30 16:37 a.out
-rwxrwxrwx 1 user user 16760 May 30 16:37 sample
-rw-rw-rw- 1 user user  1020 May 30 16:37 sample.c
```

As you can see, the part after `execvp()` does not execute at all, since "ls -l" took control of our process!

Let's re-write the same example, but let's enclose the `execvp()` system call inside another process, using `fork()`.

Let's see what happens now.

Copy

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

```
int main() {
    char* command = "ls";
    char* argument_list[] = {"ls", "-l", NULL};

    printf("Before calling execvp()\n");

    printf("Creating another process using fork()...\n");

    if (fork() == 0) {
        // Newly spawned child Process. This will be taken over by "ls -l"
        int status_code = execvp(command, argument_list);

        printf("ls -l has taken control of this child process. This won't execute unless
```



```
    return 0;
}
```

Output

```
Before calling execvp()
Creating another process using fork()...
This line will be printed
user@shell:$ total 3
-rwxrwxrwx 1 user user 22088 May 30 16:37 a.out
-rwxrwxrwx 1 user user 16760 May 30 16:37 sample
-rw-rw-rw- 1 user user  1020 May 30 16:37 sample.c
```

Copy

If you're in a shell, the output may look weird, but that's because multiple processes ran in parallel! Both the outputs were indeed printed, so we've been able to resolve our problem.

Conclusion

We learned about using the execvp() function in C / C++, to execute other programs from our C program. However, note that this will give the other program complete control of our process.

Due to this, we need to enclose this under another process, using the `fork()` system call. Hopefully, this made sense to you, and you were able to run other programs, while still being able to have control over your C program!

References

- Linux manual page on the execvp() function in C

About the authors

Vijaykrishna Ram Author

Still looking for an answer? Ask a question

Search for more help

Was this helpful? Yes No

Comments

JournalDev • March 3, 2022

You show this working for c/c++, however, when attempting to compile this code I get the error that C++ forbids converting a string constant to ‘char*’

- Kenneth

Popular Topics

Ubuntu

Linux Basics

JavaScript

Python

MySQL

Docker

Kubernetes

All tutorials →

Free Managed Hosting →

Get our biweekly newsletter

Sign up for Infrastructure as a Newsletter.

Sign up →

Hollie's Hub for Good

Working on improving health and education, reducing inequality, and spurring economic growth? We'd like to help.

Learn more →

Become a contributor

You get paid; we donate to tech nonprofits.

Learn more →

Featured on Community