

C Programming

# Fork System Call in C

3 years ago • by Shahriar Shovon

fork() system call is used to create child processes in a C program. fork() is used where parallel processing is required in your application. The fork() system function is defined in the headers **sys/types.h** and **unistd.h**. In a program where you use fork, you also have to use wait() system call. wait() system call is used to wait in the parent process for the child process to finish. To finish a child process, the exit() system call is used in the child process. The wait() function is defined in the header **sys/wait.h** and the exit() function is defined in the header **stdlib.h**.

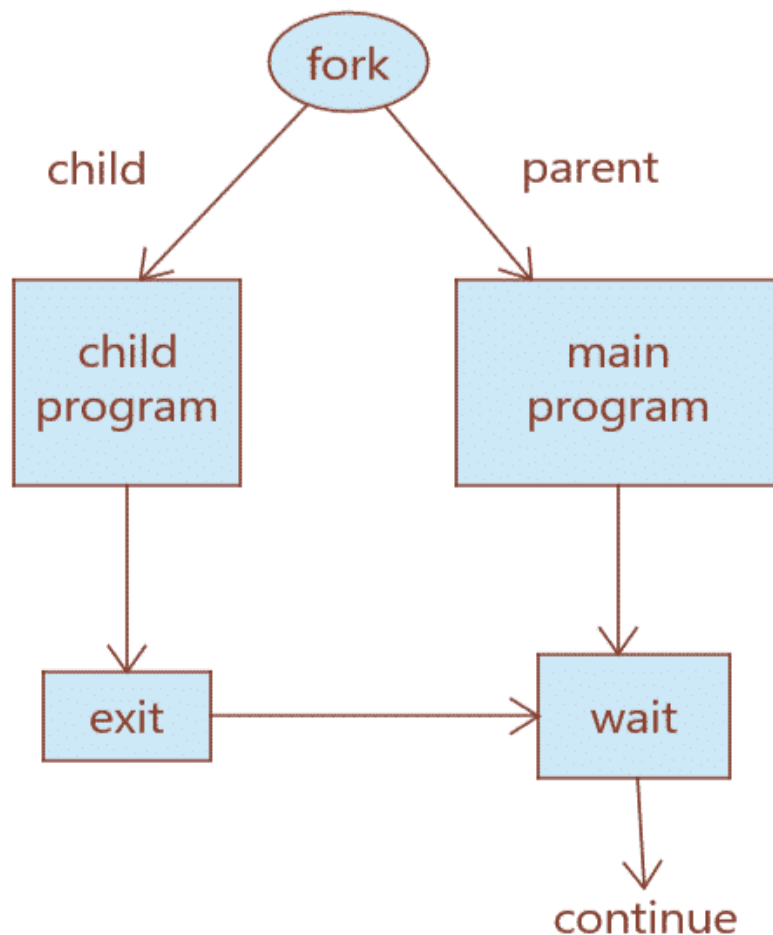
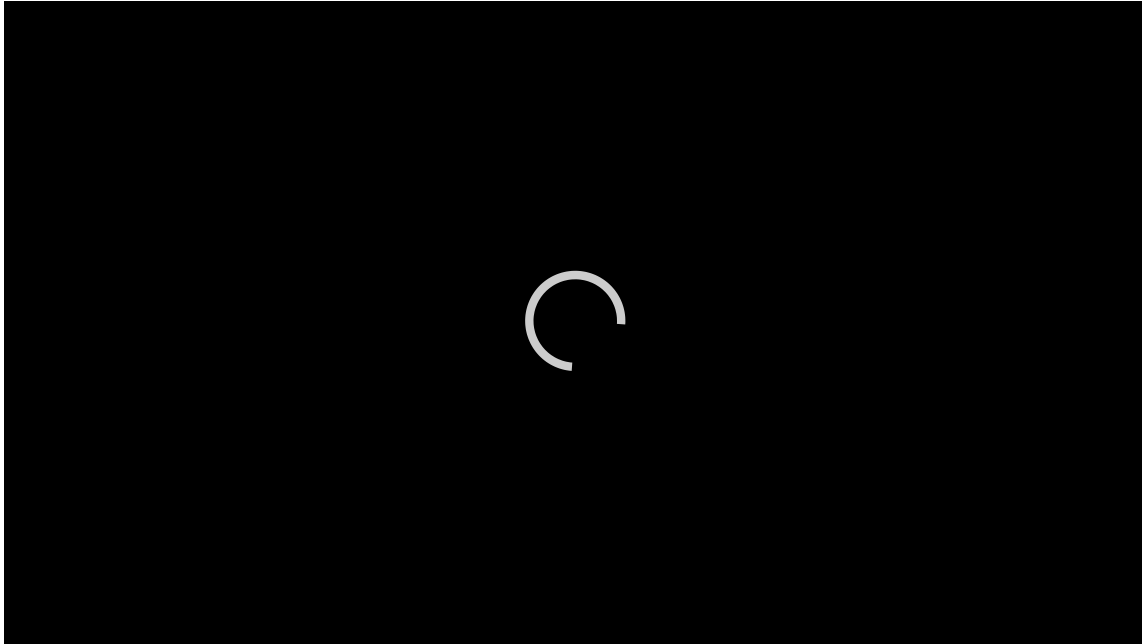


Fig 1: Basic fork() workflow

In this article, I am going to show you how to use fork() system call to create child processes in C. So, let's get started.

---

## MY LATEST VIDEOS



---

## fork() Syntax and Return Value:

The syntax of the fork() system function is as follows:

```
pid_t fork(void);
```

The fork() system function does not accept any argument. It returns an integer of the type **pid\_t**.

On success, fork() returns the PID of the child process which is greater than 0. Inside the child process, the return value is 0. If fork() fails, then it returns -1.

## Simple fork() Example:

A simple fork() example is given below:

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

int main(void) {
    pid_t pid = fork();

    if(pid == 0) {
        printf("Child => PPID: %d PID: %d\n", getppid(), getpid());
        exit(EXIT_SUCCESS);
    }
    else if(pid > 0) {
        printf("Parent => PID: %d\n", getpid());
        printf("Waiting for child process to finish.\n");
        wait(NULL);
        printf("Child process finished.\n");
    }
    else {

```

```

    printf("Unable to create child process.\n");
}

return EXIT_SUCCESS;
}

```

Here, I used `fork()` to create a child process from the main/parent process. Then, I printed the PID (Process ID) and PPID (Parent Process ID) from child and parent process. On the parent process `wait(NULL)` is used to wait for the child process to finish. On the child process, `exit()` is used to finish the child process. As you can see, the PID of the parent process is the PPID of the child process. So, the child process **24738** belongs to the parent process **24731**.

```

01_fork.c - system-programming - Visual Studio Code
C 01_fork.c
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/types.h>
4  #include <sys/wait.h>
5  #include <unistd.h>
6
7  int main(void) {
8      pid_t pid = fork();
9
10     if(pid == 0) {
11         printf("Child => PPID: %d PID: %d\n", getppid(), getpid());
12         exit(EXIT_SUCCESS);
13     } else if(pid > 0) {
14         printf("Parent => PID: %d\n", getpid());
15         printf("Waiting for child process to finish.\n");
16         wait(NULL);
17         printf("Child process finished.\n");
18     } else {
19         printf("Unable to create child process.\n");
20     }
21
22     return EXIT_SUCCESS;
23 }
24
PROBLEMS 0 OUTPUT DEBUG CONSOLE TERMINAL
Parent => PID: 24731
Waiting for child process to finish.
Child => PPID: 24731 PID: 24738
Child process finished.
Terminal will be reused by tasks, press any key to close it.

```

You can also use functions to make your program more modular. Here, I used `processTask()` and `parentTask()` functions for the child and parent processes respectively. This is how `fork()` is actually used.

```

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

void childTask() {
    printf("Hello World\n");
}

void parentTask() {
    printf("Main task.\n");
}

int main(void) {
    pid_t pid = fork();

    if(pid == 0) {
        childTask();
        exit(EXIT_SUCCESS);
    } else if(pid > 0) {
        wait(NULL);
        parentTask();
    } else {
        printf("Unable to create child process.");
    }
}

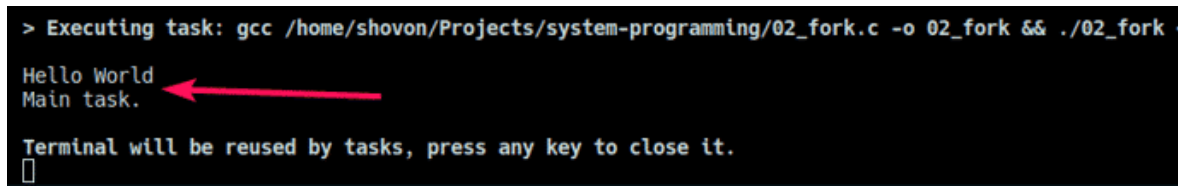
```

```

    return EXIT_SUCCESS;
}

```

The output of the above program:



```

> Executing task: gcc /home/shovon/Projects/system-programming/02_fork.c -o 02_fork && ./02_fork
Hello World
Main task.
Terminal will be reused by tasks, press any key to close it.

```

## Running Multiple Child Processes using fork() and Loop:

You can also use loop to create as many child processes as you need. In the example below, I have created 5 child processes using for loop. I also printed the PID and PPID from the child processes.

```

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

int main(void) {
    for(int i = 1; i <= 5; i++) {
        pid_t pid = fork();

        if(pid == 0) {
            printf("Child process => PPID=%d, PID=%d\n", getppid(), getpid());
            exit(0);
        }
        else {
            printf("Parent process => PID=%d\n", getpid());
            printf("Waiting for child processes to finish...\n");
            wait(NULL);
            printf("child process finished.\n");
        }
    }

    return EXIT_SUCCESS;
}

```

As you can see, the Parent process ID is the same in all the child processes. So, all of them belong to the same parent. They also execute in linear fashion. One after the other.

Controlling child processes is a sophisticated task. If you learn more about Linux system programming and how it works, you will be able to control the flow of these processes anyway you like.

```

> Executing task: gcc /home/shovon/Projects/system-programming/03_fork.c -o 03_fork && ./03_fork <
Parent process => PID=28751
Waiting for child processes to finish...
Child process => PPID=28751, PID=28758
child process finished.
Parent process => PID=28751
Waiting for child processes to finish...
Child process => PPID=28751, PID=28759
child process finished.
Parent process => PID=28751
Waiting for child processes to finish...
Child process => PPID=28751, PID=28760
child process finished.
Parent process => PID=28751
Waiting for child processes to finish...
Child process => PPID=28751, PID=28761
child process finished.
Parent process => PID=28751
Waiting for child processes to finish...
Child process => PPID=28751, PID=28762
child process finished.

Terminal will be reused by tasks, press any key to close it.

```

## Real Life Example:

Different complex mathematical computations such as md5, sha256 etc hash generation requires a lot of processing power. Instead of computing things like that in the same process as the main program, you can just calculate the hash on a child process and return the hash to the main process.

In the following example, I've generated a 4-digit PIN code in a child process and send it to the parent process, the main program. Then, I printed the PIN code from there.

```

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

int getPIN() {
    // use PPID and PID as the seed
    srand(getpid() + getppid());
    int secret = 1000 + rand() % 9000;
    return secret;
}

int main(void) {
    int fd[2];
    pipe(fd);
    pid_t pid = fork();

    if(pid > 0) {
        close(0);
        close(fd[1]);
        dup(fd[0]);

        int secretNumber;
        size_t readBytes = read(fd[0], &secretNumber, sizeof(secretNumber));

        printf("Waiting for PIN...\n");
        wait(NULL);
        printf("Bytes read: %ld\n", readBytes);
        printf("PIN: %d\n", secretNumber);
    }
    else if(pid == 0) {
        close(1);
        close(fd[0]);
        dup(fd[1]);

        int secret = getPIN();
        write(fd[1], &secret, sizeof(secret));
        exit(EXIT_SUCCESS);
    }

    return EXIT_SUCCESS;
}

```

As you can see, each time I run the program, I get a different 4-digit PIN code.

So, that's basically how you use `fork()` system call in Linux. Thanks for reading this article.

## ABOUT THE AUTHOR



### Shahriar Shovon

Freelancer & Linux System Administrator. Also loves Web API development with Node.js and JavaScript. I was born in Bangladesh. I am currently studying Electronics and Communication Engineering at Khulna University of Engineering & Technology (KUET), one of the demanding public engineering universities of Bangladesh.

[View all posts](#)

## RELATED LINUX HINT POSTS

C Uppercase to Lowercase

How to bit flip in C

How to reverse an array in C

String to integer in C

How to Debug Segmentation Faults  
in C?

Strchr Metohd in C

Static Functions in the C Computer  
Language

Linux Hint LLC, editor@linuxhint.com  
1309 S Mary Ave Suite 210, Sunnyvale, CA  
94087

AN ELITE CAFEMEDIA PUBLISHER