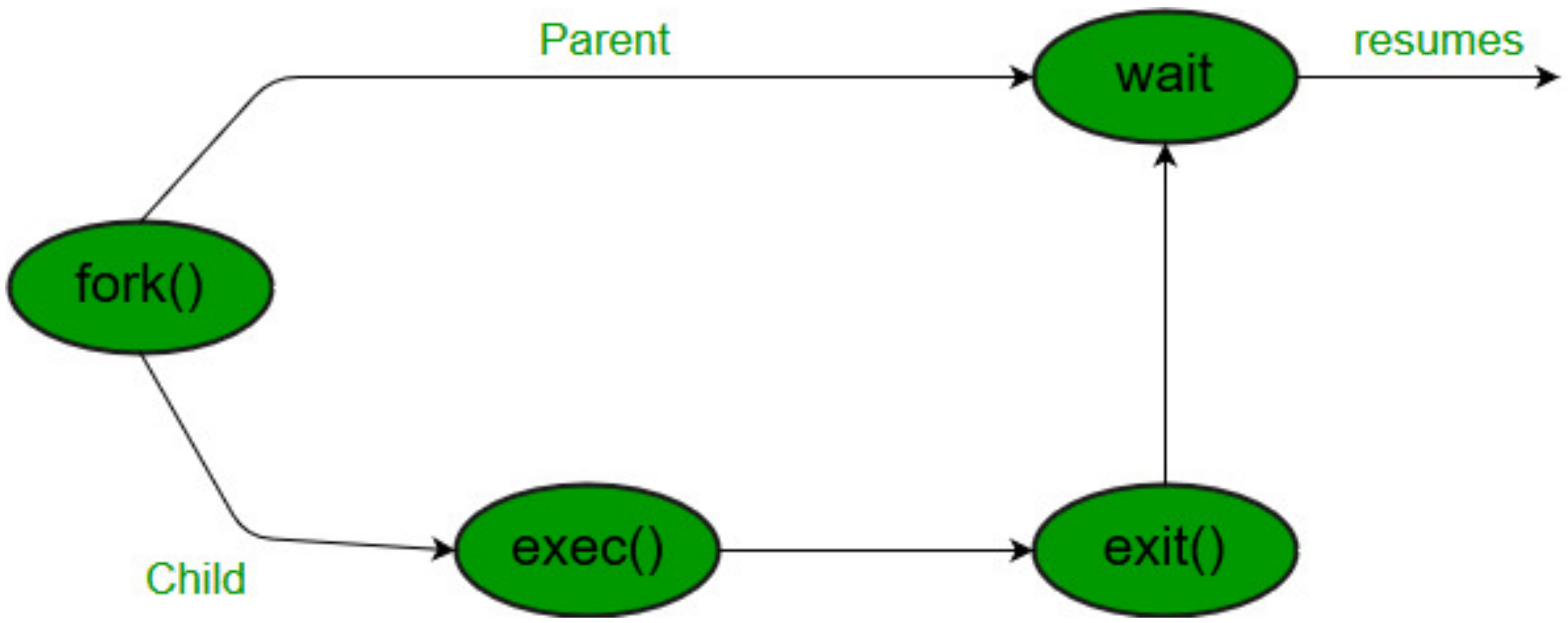
Wait System Call in C

Prerequisite : Fork System call

A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent ***continues*** its execution after wait system call instruction.
Child process may terminate due to any of these:

- It calls exit();
- It returns (an int) from main
- It receives a signal (from the OS or another process) whose default action is to terminate.



Syntax in c language:

```
#include
#include

// take one argument status and returns
// a process ID of dead children.
pid_t wait(int *stat_loc);
```

If any process has more than one child processes, then after calling wait(), parent process has to be in wait state if no child terminates.

If only one child process is terminated, then return a wait() returns process ID of the terminated child process.

then
If more than one child processes are terminated than wait() reap any
arbitrary
arbitrarily child and return a process ID of that child process.
When wait() returns they also define **exit status** (which tells our, a process why terminated) via pointer, If status are not **NULL**.

If any process has no child process then wait() returns immediately "-1".

NOTE: "This codes does not run in simple IDE because of environmental problem so use terminal for run the code"

Examples:

filter_none

edit

play_arrow

brightness_4

```
#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>



int main()

{

    pid_t cpid;

    if (fork()== 0)

        exit(0);

    else

        cpid = wait(NULL);
    printf("Parent pid = %d\n", getpid());

    printf("Child pid = %d\n", cpid);



    return 0;

}
```

Output:

```
Parent pid = 12345678
Child pid = 89546848
```

filter_none

edit

play_arrow

brightness_4

```
#include<stdio.h>

#include<sys/wait.h>

#include<unistd.h>



int main()

{

    if (fork()== 0)

        printf("HC: hello from child\n");

    else

    {

        printf("HP: hello from parent\n");

        wait(NULL);

        printf("CT: child has terminated\n");

    }
    printf("Bye\n");

    return 0;

}
```

Output: depend on environment

```
HC: hello from child
HP: hello from parent
CT: child has terminated
     (or)
HP: hello from parent
HC: hello from child
CT: child has terminated    // this sentence does
                            // not print before HC
                            // because of wait.
```

Child status information:
Status information about the child reported by wait is more than just the exit status of the child, it also includes

- normal/abnormal termination
- termination cause
- exit status

For find information about status, we use
WIF….macros

1. **WIFEXITED(status)**: child exited normally
• **WEXITSTATUS(status)**: return code when child exits

2. **WIFSIGNALED(status)**: child exited because a signal was not caught
• **WTERMSIG(status)**: gives the number of the terminating signal

3. **WIFSTOPPED(status)**: child is stopped
• **WSTOPSIG(status)**: gives the number of the stop signal

```
/*if we want to prints information about a signal */
void psignal(unsigned sig, const char *s);
```

Examples:
Check output of the following program.

filter_none

edit

play_arrow

brightness_4

```
#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>



void waitexample()

{

    int stat;




    if (fork() == 0)

        exit(1);

    else

        wait(&stat);

    if (WIFEXITED(stat))

        printf("Exit status: %d\n", WEXITSTATUS(stat));

    else if (WIFSIGNALED(stat))

        psignal(WTERMSIG(stat), "Exit signal");
}



int main()

{

    waitexample();

    return 0;

}
```

Output:

```
Exit status: 1
```

We know if more than one child processes are terminated, then wait() reaps any arbitrarily child process but if we want to reap any specific child process, we use ***waitpid()*** function.

> ***Syntax in c language:***
> *pid_t waitpid (child_pid, &status, options);*

Options Parameter

- If 0 means no option parent has to wait for terminates child.
- If **WNOHANG** means parent does not wait if child does not terminate just check and return waitpid().(not block parent process)
- If child_pid is **-1** then means any ***arbitrarily child***, here waitpid() work same as wait() work.

Return value of waitpid()

- pid of child, if child has exited
- 0, if using WNOHANG and child hasn't exited

filter_none

edit

play_arrow

brightness_4

```c
#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>



void waitexample()

{

    int i, stat;

    pid_t pid[5];

    for (i=0; i<5; i++)

    {

        if ((pid[i] = fork()) == 0)

        {

            sleep(1);

            exit(100 + i);

        }

    }






    for (i=0; i<5; i++)

    {

        pid_t cpid = waitpid(pid[i], &stat, 0);

        if (WIFEXITED(stat))

            printf("Child %d terminated with status: %d\n",
                        cpid, WEXITSTATUS(stat));

    }

}



int main()

{

    waitexample();

    return 0;

}
```

Output:

```
Child 50 terminated with status: 100
Child 51 terminated with status: 101
Child 52 terminated with status: 102
Child 53 terminated with status: 103
Child 54 terminated with status: 104
```

Here, Children pids depend on the system but in order print all child information.

Recommended Posts:

- pipe() System call

- Accept system call
- dup() and dup2() Linux system call
- Difference between Call by Value and Call by Reference
- fork() to execute processes from bottom to up using wait()
- Conditional wait and signal in multi-threading
- Can we call an undeclared function in C++?
- A C/C++ Function Call Puzzle
- Is it possible to call constructor and destructor explicitly?
- Difference between CALL and JUMP instructions
- system() in C/C++
- Amazing stuff with system() in C / C++
- Input-output system calls in C | Create, Open, Close, Read, Write
- OpenMP | Introduction with Installation Guide

Article Tags :
C
system-programming

thumb_up
3

2.8

Based on **5** vote(s)