**wait() and waitpid()**

C program on how to use **wait system call** in C and print "exit status" value on WIFEXITED and WIFSIGNALED

The wait() and waitpid() Functions

There are certain situations where when a child process terminates or changes state then the parent process should come to know about the change of the state or termination status of the child process. In that case functions like wait() are used by the parent process where the parent can query the change in state of the child process using these functions.

**Definition of wait():**

**pid\_t wait(int \*status);**

For the cases where a parent process has more than one child processes, there is a function waitpid() that can be used by the parent process to query the change state of a particular child.

**Definition of waitpid():**

**pid\_t waitpid(pid\_t pid, int \*status, int options);**

By default,  waitpid() waits only for terminated children, but this behavior is modifiable via the options argument, as described below.

The value of pid can be:

< -1 : Wait for any child process whose process group ID is equal to the absolute value of pid.

-1 : Wait for any child process.

0 : Wait for any child process whose process group ID is equal to that of the calling process.

> 0 : Wait for the child whose process ID is equal to the value of pid.

The value of options is an OR of zero or more of the following constants:

WNOHANG : Return immediately if no child has exited.

WUNTRACED : Also  return if a child has stopped. Status for traced children which have stopped is provided even if this option is not specified.

WCONTINUED : Also return if a stopped child has been resumed by delivery of SIGCONT.

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.
* 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.
* If more than one child processes are terminated than wait() reap any 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”.

**/\* C program to demonstrate working of wait() \*/**

#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>

int main()

{

    pid\_t cpid;

    if (fork()== 0) {

        printf("\n child pid : %d\n", getpid());

        exit(0);           /\* terminate child \*/

    }

    else

    {

        /\* wait():

           on success, returns the process ID of the terminated child;

           on error, -1 is returned.\*/

        cpid = wait(NULL); /\* reaping parent \*/

    }

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

    printf("Child pid = %d\n", cpid);

    return 0;

}

**// C program to demonstrate working of status from wait.**

#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>

void waitexample()

{

    int stat;

    // This status 1 is reported by WEXITSTATUS

    if (fork() == 0)

        /\* to check the WIFEXITED case, uncomment out the below and comment out while(1) \*/

        //exit(5);

        /\* to check the WIFSIGNALED case, comment out the below and uncomment the above \*/

        while(1);

    else {

        wait(&stat);

        if (WIFEXITED(stat))

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

        else if (WIFSIGNALED(stat)) {

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

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

        }

    }

}

// Main Program

int main()

{

    waitexample();

    return 0;

}

|  |
| --- |
| **// C program to demonstrate working of wait()**  #include<stdio.h>  #include<stdlib.h>  #include<sys/wait.h>  #include<unistd.h>    int main()  {      pid\_t cpid;      if (fork()== 0)          exit(0);           /\* terminate child \*/      else          cpid = wait(NULL); /\* reaping parent \*/      printf("Parent pid = %d\n", getpid());      printf("Child pid = %d\n", cpid);        return 0;  } |

Output:

Parent pid = 12345678

Child pid = 89546848

|  |
| --- |
| **// C program to demonstrate working of wait()**  #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

Bye

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.

Bye

----------------------------------------------------------------------------------------------