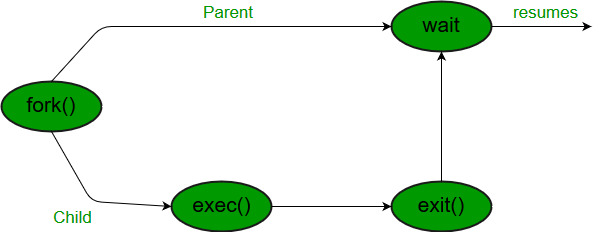
A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent ***continues*** its execution after wait system call instruction.   
Child process may terminate due to any of these:

* 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.



// 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.   
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”.  
***NOTE: “This codes does not run in simple IDE because of environmental problem so use terminal for run the code”***

**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);

/\* Program to demonstrate wait and exit\*/

#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>

int main()

{

pid\_t cpid;

int status = 0;

cpid = fork();

if(cpid == -1)

exit(-1); /\* terminate child \*/

if(cpid == 0){

printf("\nchild executing first its pid = (%d)\n",getpid());

sleep(20);

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

exit(1);

}

else{

printf("\n Parent executing before wait()\n");

cpid = wait(NULL);

//cpid = wait(&status);

printf("\n wait() in parent done\nParent pid = %d\n", getpid());

printf("\n cpid returned is (%d)\n",cpid);

printf("\n status is (%d)\n",status);

}

return 0;

}

#if 0

Process termination status (int) 16 bit number

1. status value if child process has normal exit/termination

15......8 | 7......0

XXXXXXXX | XXXXXXXX

exitStaus | 0

2. killed by signal

15......8 | 7 ......0

unused | X termination signal

|

|-----> core dump flag

#endif

**wait() and waitpid()**

The **wait**() system call suspends execution of the calling process until one of its children

terminates. The call wait(&status) is equivalent to:

waitpid(-1, &status, 0);

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

The **waitpid**() system call suspends execution of the calling process until a child

specified by pid argument has changed state. 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 meaning wait for any child process whose process group ID is equal to the absolute value of pid.

-1 meaning wait for any child process.

0 meaning wait for any child process whose process group ID is equal to that of the

calling process.

> 0 meaning 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 (but not traced via [**ptrace**](https://manpages.ubuntu.com/manpages/trusty/man2/ptrace.2.html)(2)). Status for

traced children which have stopped is provided even if this option is not

specified.

**WCONTINUED** (since Linux 2.6.10)

also return if a stopped child has been resumed by delivery of **SIGCONT**.

(For Linux-only options, see below.)

If status is not NULL, **wait**() and **waitpid**() store status information in the int to which

it points. This integer can be inspected with the following macros (which take the

integer itself as an argument, not a pointer to it, as is done in **wait**() and **waitpid**()!):

**WIFEXITED(**status**)**

returns true if the child terminated normally, that is, by calling [**exit**](https://manpages.ubuntu.com/manpages/trusty/man3/exit.3.html)(3) or

[**\_exit**](https://manpages.ubuntu.com/manpages/trusty/man2/_exit.2.html)(2), or by returning from main().

**WEXITSTATUS(**status**)**

returns the exit status of the child. This consists of the least significant 8

bits of the status argument that the child specified in a call to [**exit**](https://manpages.ubuntu.com/manpages/trusty/man3/exit.3.html)(3) or

[**\_exit**](https://manpages.ubuntu.com/manpages/trusty/man2/_exit.2.html)(2) or as the argument for a return statement in main(). This macro should be

employed only if **WIFEXITED** returned true.

**WIFSIGNALED(**status**)**

returns true if the child process was terminated by a signal.

**WTERMSIG(**status**)**

returns the number of the signal that caused the child process to terminate. This

macro should be employed only if **WIFSIGNALED** returned true.

**WCOREDUMP(**status**)**

returns true if the child produced a core dump. This macro should be employed only

if **WIFSIGNALED** returned true. This macro is not specified in POSIX.1-2001 and is

not available on some UNIX implementations (e.g., AIX, SunOS). Only use this

enclosed in #ifdef WCOREDUMP ... #endif.

**WIFSTOPPED(**status**)**

returns true if the child process was stopped by delivery of a signal; this is

possible only if the call was done using **WUNTRACED** or when the child is being

traced (see [**ptrace**](https://manpages.ubuntu.com/manpages/trusty/man2/ptrace.2.html)(2)).

**WSTOPSIG(**status**)**

returns the number of the signal which caused the child to stop. This macro should

be employed only if **WIFSTOPPED** returned true.

**WIFCONTINUED(**status**)**

(since Linux 2.6.10) returns true if the child process was resumed by delivery of

**SIGCONT**.

/\* Program to demonstrate wait and exit\*/

/\* Use of WNOHANG => This is a non blocking call to waitpid()\*/

#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>

#include <errno.h>

extern int errno;

int main()

{

pid\_t cpid;

pid\_t cpid2;

pid\_t ret\_pid;

int status = 0;

cpid = fork();

if(cpid == -1)

exit(-1); /\* terminate child depending on use case\*/

if(cpid == 0){ /\* This is child process \*/

printf("\nchild-1 executing its pid = (%d)\n",getpid());

sleep(10);

printf("Child1 exited\n");

exit(0);

}

else{/\* This is parent process\*/

printf("\n Parent executing before wait() Parent pid is (%d)\n",getpid());

ret\_pid = waitpid(80, &status, WNOHANG);

//ret\_pid = waitpid(cpid, &status, 0);

if (ret\_pid == -1){

perror("Error is:");

printf("\n waitpid returned error (%d)\n",errno);

}

printf("\n cpid returned is (%d)\n",ret\_pid);

printf("\n status is (%d)\n",status);

printf("\n Parent exited\n");

}

return 0;

}

**Output:**

Parent executing before wait() Parent pid is (20734)

Error is:: No child processes

waitpid returned error (10)

cpid returned is (-1)

status is (0)

Parent exited

child-1 executing its pid = (20735)

Child1 exited

Parent executing before wait() Parent pid is (27140)

child-1 executing its pid = (27141)

Child1 exited

cpid returned is (27141)

status is (0)

Parent exited

#if 0

refer to below for details on errno

man errno

#endif

/\* Program to demonstrate wait and exit\*/

#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>

int main()

{

pid\_t cpid;

pid\_t cpid2;

pid\_t ret\_pid;

int status = 0;

cpid = fork();

if(cpid == -1)

exit(-1); /\* terminate child depending on use case\*/

if(cpid == 0){

printf("\nchild-1 executing its pid = (%d)\n", getpid());

sleep(10);

printf("Child1 exited\n");

exit(0);

}

else{

cpid2 = fork();

if(cpid2 == -1){

exit(-1);

}

if(cpid2 == 0){

printf("\nchild-2 executing its pid = (%d)\n",getpid());

sleep(5);

printf("Child2 exited\n");

exit(1);

}

printf("\n Parent executing before wait() Parent pid is (%d)\n",getpid());

ret\_pid = waitpid(cpid2, &status, 0);

printf("\n cpid returned is (%d)\n",ret\_pid);

printf("\n status is (%d)\n",status);

ret\_pid = waitpid(cpid, &status, 0);

printf("\n cpid returned is (%d)\n", ret\_pid);

printf("\n status is (%d)\n",status);

printf("\n Parent exited\n");

}

return 0;

}

**Output:**

Parent executing before wait() Parent pid is (21359)

child-1 executing its pid = (21360)

child-2 executing its pid = (21361)

Child2 exited

cpid returned is (21361)

status is (256)

Child1 exited

cpid returned is (21360)

status is (0)

Parent exited

/\* Program to demonstrate wait and exit\*/

#include<stdio.h>

#include<stdlib.h>

#include<sys/wait.h>

#include<unistd.h>

int main()

{

pid\_t cpid;

pid\_t cpid2;

int status = 0;

cpid = fork();

if(cpid == -1)

exit(-1); /\* terminate child \*/

if(cpid == 0){

printf("\nfirst child executing its pid = (%d)\n",getpid());

sleep(20);

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

exit(1);

}

else{

cpid2 = fork();

if(cpid2 == 0){

printf("\nsecond child executing its pid = (%d)\n",getpid());

sleep(20);

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

exit(1);

}

printf("\n Parent executing before wait()\n");

while(1)

{

pid\_t id2 = waitpid(cpid, &status, WNOHANG|WUNTRACED|WCONTINUED);

printf("status == %d\n", status);

sleep(1);

if(id2==-1)

{break;}

if(id2>0){

//cpid = wait(&status);

if(WIFEXITED(status)){

printf("The exit status set %d \n", WEXITSTATUS(status));

}

else if (WIFSIGNALED(status)){

printf("The termination signal is %d \n", WTERMSIG(status));

}

else if (WIFSTOPPED(status)){

printf("The stopped signal is : %d \n",WSTOPSIG(status));

}

else if (WIFCONTINUED(status))

printf("The prosess is resume\n");

}

}

}

printf("The parent process completed .......\n");

return 0;

}

**Output:**

/tmp/blLBxPEwlS.o

Parent executing before wait()

status == 0

second child executing its pid = (21895)

first child executing its pid = (21894)

status == 0

status == 0

status == 0

status == 0

status == 0

status == 0

status == 0

status == 0

status == 0

status == 0

status == 0

status == 0

status == 0

status == 0

status == 0

status == 0

status == 0

status == 0

status == 0

Child pid = 21895

Child pid = 21894

status == 256

The exit status set 1

status == 256

The parent process completed .......