Taken from die.net waitpid(2) - Linux man page

wait, waitpid - wait for process to change state

wait and waitpid syscalls are part of the <sys/types.h> and <sys/wait.h> header files.

Definition

pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);

Description

All of these system calls are used to wait for state changes in a child of the calling process, and obtain information about the child whose state has changed. A state change is considered to be: the child terminated; the child was stopped by a signal; or the child was resumed by a signal. In the case of a terminated child, performing a wait allows the system to release the resources associated with the child; if a wait is not performed, then the terminated child remains in a "zombie" state (see NOTES below).

If a child has already changed state, then these calls return immediately. Otherwise they block until either a child changes state or a signal handler interrupts the call (assuming that system calls are not automatically restarted using the SA_RESTART flag of sigaction(2)). In the remainder of this page, a child whose state has changed and which has not yet been waited upon by one of these system calls is termed waitable.

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);

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.

WUNTRACED
also return if a child has stopped (but not traced via ptrace(2)). 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.
WIFEXITED(status)
returns true if the child terminated normally, that is, by calling exit(3) or _exit(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(3) or _exit(2) or as the argument for a return statement in main(). This macro should only be employed 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 only be employed if WIFSIGNALED returned true.
WCOREDUMP(status)
returns true if the child produced a core dump. This macro should only be employed if WIFSIGNALED returned true.
WIFSTOPPED(status)
returns true if the child process was stopped by delivery of a signal; this is only possible if the call was done using WUNTRACED or when the child is being traced (see ptrace(2)).
WSTOPSIG(status)
returns the number of the signal which caused the child to stop. This macro should only be employed if WIFSTOPPED returned true.
WIFCONTINUED(status)
returns true if the child process was resumed by delivery of SIGCONT.

The following program demonstrates the use of this program:

```
#include<stdio.h>
#include<stdlib.h>

int main(void)
{
	int exit_status =10;

	int child = fork();
	printf("parent pid %d ppid %d\n",getpid(),getppid());

	if(child!=0)
	{
	printf("inside parent pid %d ppid %d\n",getpid(),getppid());
	waitpid(child,&exit_status, WUNTRACED | WCONTINUED)  ;

	if(WIFEXITED(0))
	{
	printf("child exit status = %d\n",WEXITSTATUS(exit_status));
	printf("parent exited\n");
	}

	}
	else{
	printf("inside child pid %d ppid %d\n",getpid(),getppid());
	printf("child exit\n");
	exit(0);
	}
}
```

OUTPUT ON TERMINAL

```
guest-4dfjxw@Lab301-7:~/$ ./a.out
parent pid 3189 ppid 2777
inside parent pid 3189 ppid 2777
parent pid 3190 ppid 3189
inside child pid 3190 ppid 3189
child exit
child exit status = 0
parent exited
```