NAME

wait, wait3, wait4, waitpid, WIFSTOPPED, WIFSIGNALED, WIFEXITED – wait for process to terminate or stop

SYNOPSIS

/usr/ucb/cc [*flag* ...] *file* ...

#include <sys/wait.h>

int wait(*statusp***)**
int **statusp***;**

int waitpid(*pid*, *statusp*, *options***)**
int *pid***;**
int **statusp***;**
int *options***;**

#include <sys/time.h>
#include <sys/resource.h>

int wait3(*statusp*, *options*, *rusage***)**
int **statusp***;**
int *options***;**
struct rusage **rusage***;**

int wait4(*pid*, *statusp*, *options*, *rusage***)**
int *pid***;**
int **statusp***;**
int *options***;**
struct rusage **rusage***;**

WIFSTOPPED(*status***)**
int *status***;**

WIFSIGNALED(*status***)**
int *status***;**

WIFEXITED(*status***)**
int *status***;**

DESCRIPTION

wait( ) delays its caller until a signal is received or one of its child processes terminates or stops due to tracing. If any child process has died or stopped due to tracing and this has not been reported using **wait( )**, return is immediate, returning the process ID and exit status of one of those children. If that child process has died, it is discarded. If there are no children, return is immediate with the value −1 returned. If there are only running or stopped but reported children, the calling process is blocked.

If *status* is not a **NULL** pointer, then on return from a successful **wait( )** call the status of the child process whose process ID is the return value of **wait( )** is stored in the **wait( )** union pointed to by *status*. The **w_status** member of that union is an **int**; it indicates the cause of termination and other information about the terminated process in the following manner:

- If the low-order 8 bits of **w_status** are equal to 0177, the child process has stopped; the 8 bits higher up from the low-order 8 bits of **w_status** contain the number of the signal that caused the process to stop. See **ptrace**(2) and **sigvec**(3B).

- If the low-order 8 bits of **w_status** are non-zero and are not equal to 0177, the child process terminated due to a signal; the low-order 7 bits of **w_status** contain the number of the signal that terminated the process. In addition, if the low-order seventh bit of **w_status** (that is, bit 0200) is set, a ''core image'' of the process was produced; see **sigvec**(3B).
- Otherwise, the child process terminated due to an **exit()** call; the 8 bits higher up from the low-order 8 bits of **w_status** contain the low-order 8 bits of the argument that the child process passed to **exit()**; see **exit**(2).

waitpid() behaves identically to **wait()** if *pid* has a value of −1 and *options* has a value of zero. Otherwise, the behavior of **waitpid()** is modified by the values of *pid* and *options* as follows:

pid specifies a set of child processes for which status is requested. **waitpid()** only returns the status of a child process from this set.

- If *pid* is equal to −1, status is requested for any child process. In this respect, **waitpid()** is then equivalent to **wait()**.
- If *pid* is greater than zero, it specifies the process ID of a single child process for which status is requested.
- If *pid* is equal to zero, status is requested for any child process whose process group ID is equal to that of the calling process.
- If *pid* is less than −1, status is requested for any child process whose process group ID is equal to the absolute value of *pid*.

options is constructed from the bitwise inclusive OR of zero or more of the following flags, defined in the header **<sys/wait.h>**:

WNOHANG
: **waitpid()** does not suspend execution of the calling process if status is not immediately available for one of the child processes specified by *pid*.

WUNTRACED
: The status of any child processes specified by *pid* that are stopped, and whose status has not yet been reported since they stopped, are also reported to the requesting process.

wait3() is an alternate interface that allows both non-blocking status collection and the collection of the status of children stopped by any means. The *status* parameter is defined as above. The *options* parameter is used to indicate the call should not block if there are no processes that have status to report (**WNOHANG**), and/or that children of the current process that are stopped due to a **SIGTTIN**, **SIGTTOU**, **SIGTSTP**, or **SIGSTOP** signal are eligible to have their status reported as well (**WUNTRACED**). A terminated child is discarded after it reports status, and a stopped process will not report its status more than once. If *rusage* is not a **NULL** pointer, a summary of the resources used by the terminated process and all its children is returned. Only the user time used and the system time used are currently available. They are returned in **rusage.ru_utime** and **rusage.ru_stime**, respectively.

When the **WNOHANG** option is specified and no processes have status to report, **wait3( )** returns 0. The **WNOHANG** and **WUNTRACED** options may be combined by ORing the two values.

wait4( ) is another alternate interface. With a *pid* argument of 0, it is equivalent to **wait3( )**. If *pid* has a nonzero value, then **wait4( )** returns status only for the indicated process ID, but not for any other child processes.

WIFSTOPPED, **WIFSIGNALED**, **WIFEXITED**, are macros that take an argument *status*, of type **int**, as returned by **wait( )**, or **wait3( )**, or **wait4( )**. **WIFSTOPPED** evaluates to true (1) when the process for which the **wait( )** call was made is stopped, or to false (0) otherwise. **WIFSIGNALED** evaluates to true when the process was terminated with a signal. **WIFEXITED** evaluates to true when the process exited by using an **exit**(2) call.

RETURN VALUES

If **wait( )** or **waitpid( )** returns due to a stopped or terminated child process, the process ID of the child is returned to the calling process. Otherwise, a value of –1 is returned and **errno** is set to indicate the error.

If **wait( )** or **waitpid( )** return due to the delivery of a signal to the calling process, a value of –1 is returned and **errno** is set to EINTR. If **waitpid( )** function was invoked with **WNOHANG** set in *options*, it has at least one child process specified by *pid* for which status is not available, and status is not available for any process specified by *pid*, a value of zero is returned. Otherwise, a value of –1 is returned, and **errno** is set to indicate the error.

wait3( ) and **wait4( )** returns 0 if **WNOHANG** is specified and there are no stopped or exited children, and returns the process ID of the child process if it returns due to a stopped or terminated child process. Otherwise, they returns a value of –1 and sets **errno** to indicate the error.

ERRORS

wait( ), **wait3( )** or **wait4( )** will fail and return immediately if one or more of the following are true:

ECHILD — The calling process has no existing unwaited-for child processes.

EFAULT — The *status* or *rusage* arguments point to an illegal address.

waitpid( ) may set **errno** to:

ECHILD — The process or process group specified by *pid* does not exist or is not a child of the calling process.

EINTR — The function was interrupted by a signal. The value of the location pointed to by *statusp* is undefined.

EINVAL — The value of *options* is not valid.

wait( ), and **wait3( )**, and **wait4( )** will terminate prematurely, return –1, and set **errno** to EINTR upon the arrival of a signal whose **SV_INTERRUPT** bit in its flags field is set (see **sigvec**(3B) and **siginterrupt**(3B)). **signal**(3B), sets this bit for any signal it catches.

SEE ALSO

exit(2), **ptrace**(2), **wait**(2), **waitpid**(2), **getrusage**(3C), **siginterrupt**(3B), **signal**(3B), **sigvec**(3B), **signal**(3C)

NOTES

Use of these interfaces should be restricted to only applications written on BSD platforms. Use of these interfaces with any of the system libraries or in multi-thread applications is unsupported.

If a parent process terminates without waiting on its children, the initialization process (process ID = 1) inherits the children.

wait( ), and **wait3( )**, and **wait4( )** are automatically restarted when a process receives a signal while awaiting termination of a child process, unless the **SV_INTERRUPT** bit is set in the flags for that signal.

Calls to **wait( )** with an argument of **0** should be cast to type '**int ∗**', as in:

wait((int ∗)0)

Previous SunOS releases used **union wait ∗statusp** and **union wait status** in place of **int ∗statusp** and **int** status. The union contained a member **w_status** that could be treated in the same way as *status*.

Other members of the **wait** union could be used to extract this information more conveniently:

- If the **w_stopval** member had the value **WSTOPPED**, the child process had stopped; the value of the **w_stopsig** member was the signal that stopped the process.
- If the **w_termsig** member was non-zero, the child process terminated due to a signal; the value of the **w_termsig** member was the number of the signal that terminated the process. If the **w_coredump** member was non-zero, a core dump was produced.
- Otherwise, the child process terminated due to a call to **exit( )**. The value of the **w_retcode** member was the low-order 8 bits of the argument that the child process passed to **exit( )**.

union wait is obsolete in light of the new specifications provided by *IEEE Std 1003.1-1988* and endorsed by *SVID89* and *XPG3*. SunOS Release 4.1 supports **union wait** for backward compatibility, but it will disappear in a future release.