NAME

wait, **waitid**, **waitpid**, **wait3**, **wait4**, **wait6** - wait for processes to change status

LIBRARY

Standard C Library (libc, -lc)

SYNOPSIS

#include <sys/wait.h>

pid_t
wait(*int *status*);

pid_t
waitpid(*pid_t wpid*, *int *status*, *int options*);

#include <signal.h>

int
waitid(*idtype_t idtype*, *id_t id*, *siginfo_t *info*, *int options*);

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

pid_t
wait3(*int *status*, *int options*, *struct rusage *rusage*);

pid_t
wait4(*pid_t wpid*, *int *status*, *int options*, *struct rusage *rusage*);

pid_t
wait6(*idtype_t idtype*, *id_t id*, *int *status*, *int options*, *struct __wrusage *wrusage*, *siginfo_t *infop*);

DESCRIPTION

The **wait**() function suspends execution of its calling thread until *status* information is available for a child process or a signal is received. On return from a successful **wait**() call, the *status* area contains information about the process that reported a status change as defined below.

The **wait4**() and **wait6**() system calls provide a more general interface for programs that need to wait for specific child processes, that need resource utilization statistics accumulated by child processes, or that require options. The other wait functions are implemented using either **wait4**() or **wait6**().

The **wait6**() function is the most general function in this family and its distinct features are:

All of the desired process statuses to be waited on must be explicitly specified in *options*. The **wait**(), **waitpid**(), **wait3**(), and **wait4**() functions all implicitly wait for exited and trapped processes, but the **waitid**() and **wait6**() functions require the corresponding WEXITED and WTRAPPED flags to be explicitly specified. This allows waiting for processes which have experienced other status changes without having to also handle the exit status from terminated processes.

The **wait6**() function accepts a *wrusage* argument which points to a structure defined as:

```
struct __wrusage {
        struct rusage   wru_self;
        struct rusage   wru_children;
};
```

This allows the calling process to collect resource usage statistics from both its own child process as well as from its grand children. When no resource usage statistics are needed this pointer can be NULL.

The last argument *infop* must be either NULL or a pointer to a *siginfo_t* structure. If non-NULL, the structure is filled with the same data as for a SIGCHLD signal delivered when the process changed state.

The set of child processes to be queried is specified by the arguments *idtype* and *id*. The separate *idtype* and *id* arguments support many other types of identifiers in addition to process IDs and process group IDs.

- If *idtype* is P_PID, **waitid**() and **wait6**() wait for the child process with a process ID equal to (pid_t)id.
- If *idtype* is P_PGID, **waitid**() and **wait6**() wait for the child process with a process group ID equal to (pid_t)id.
- If *idtype* is P_ALL, **waitid**() and **wait6**() wait for any child process and the id is ignored.
- If *idtype* is P_PID or P_PGID and the id is zero, **waitid**() and **wait6**() wait for any child process in the same process group as the caller.

Non-standard identifier types supported by this implementation of **waitid**() and **wait6**() are:

P_UID Wait for processes whose effective user ID is equal to (uid_t) *id*.

P_GID Wait for processes whose effective group ID is equal to (gid_t) *id*.

P_SID Wait for processes whose session ID is equal to *id*. If the child process started its own session, its session ID will be the same as its process ID. Otherwise the session ID of a child process will match the caller's session ID.

P_JAILID Waits for processes within a jail whose jail identifier is equal to *id*.

For the **waitpid**() and **wait4**() functions, the single *wpid* argument specifies the set of child processes for which to wait.

- If *wpid* is -1, the call waits for any child process.
- If *wpid* is 0, the call waits for any child process in the process group of the caller.
- If *wpid* is greater than zero, the call waits for the process with process ID *wpid*.
- If *wpid* is less than -1, the call waits for any process whose process group ID equals the absolute value of *wpid*.

The *status* argument is defined below.

The *options* argument contains the bitwise OR of any of the following options.

WCONTINUED Report the status of selected processes that have continued from a job control stop by receiving a SIGCONT signal.

WNOHANG Do not block when there are no processes wishing to report status.

WUNTRACED Report the status of selected processes which are stopped due to a SIGTTIN, SIGTTOU, SIGTSTP, or SIGSTOP signal.

WSTOPPED An alias for WUNTRACED.

WTRAPPED Report the status of selected processes which are being traced via ptrace(2) and have trapped or reached a breakpoint. This flag is implicitly set for the functions **wait**(), **waitpid**(), **wait3**(), and **wait4**().
For the **waitid**() and **wait6**() functions, the flag has to be explicitly included in *options* if status reports from trapped processes are expected.

WEXITED Report the status of selected processes which have terminated. This flag is implicitly set for the functions **wait**(), **waitpid**(), **wait3**(), and **wait4**().
For the **waitid**() and **wait6**() functions, the flag has to be explicitly included in *options* if status reports from terminated processes are expected.

WNOWAIT Keep the process whose status is returned in a waitable state. The process may be waited for again after this call completes.

For the **waitid**() and **wait6**() functions, at least one of the options WEXITED, WUNTRACED, WSTOPPED, WTRAPPED, or WCONTINUED must be specified. Otherwise there will be no events for the call to report. To avoid hanging indefinitely in such a case these functions return -1 with errno set to EINVAL.

If *rusage* is non-NULL, a summary of the resources used by the terminated process and all its children is returned.

If *wrusage* is non-NULL, separate summaries are returned for the resources used by the terminated process and the resources used by all its children.

If *infop* is non-NULL, a siginfo_t structure is returned with the *si_signo* field set to SIGCHLD and the *si_pid* field set to the process ID of the process reporting status. For the exited process, the *si_status* field of the siginfo_t structure contains the full 32 bit exit status passed to _exit(2); the *status* argument of other calls only returns 8 lowest bits of the exit status.

When the WNOHANG option is specified and no processes wish to report status, **waitid**() sets the *si_signo* and *si_pid* fields in *infop* to zero. Checking these fields is the only way to know if a status change was reported.

When the WNOHANG option is specified and no processes wish to report status, **wait4**() and **wait6**() return a process id of 0.

The **wait**() call is the same as **wait4**() with a *wpid* value of -1, with an *options* value of zero, and a *rusage* value of NULL. The **waitpid**() function is identical to **wait4**() with an *rusage* value of NULL. The older **wait3**() call is the same as **wait4**() with a *wpid* value of -1. The **wait4**() function is identical to **wait6**() with the flags WEXITED and WTRAPPED set in *options* and *infop* set to NULL.

The following macros may be used to test the current status of the process. Exactly one of the following four macros will evaluate to a non-zero (true) value:

WIFCONTINUED(*status*)

True if the process has not terminated, and has continued after a job control stop. This macro can be true only if the wait call specified the WCONTINUED option.

WIFEXITED(*status*)
True if the process terminated normally by a call to _exit(2) or exit(3).

WIFSIGNALED(*status*)
True if the process terminated due to receipt of a signal.

WIFSTOPPED(*status*)
True if the process has not terminated, but has stopped and can be restarted. This macro can be true only if the wait call specified the WUNTRACED option or if the child process is being traced (see ptrace(2)).

Depending on the values of those macros, the following macros produce the remaining status information about the child process:

WEXITSTATUS(*status*)
If **WIFEXITED**(*status*) is true, evaluates to the low-order 8 bits of the argument passed to _exit(2) or exit(3) by the child.

WTERMSIG(*status*)
If **WIFSIGNALED**(*status*) is true, evaluates to the number of the signal that caused the termination of the process.

WCOREDUMP(*status*)
If **WIFSIGNALED**(*status*) is true, evaluates as true if the termination of the process was accompanied by the creation of a core file containing an image of the process when the signal was received.

WSTOPSIG(*status*)
If **WIFSTOPPED**(*status*) is true, evaluates to the number of the signal that caused the process to stop.

NOTES

See sigaction(2) for a list of termination signals. A status of 0 indicates normal termination.

If a parent process terminates without waiting for all of its child processes to terminate, the remaining child processes are re-assigned to the reaper of the exiting process as the parent, see procctl(2) PROC_REAP_ACQUIRE. If no specific reaper was assigned, the process with ID 1, the init process,

becomes the parent of the orphaned children by default.

If a signal is caught while any of the **wait**() calls are pending, the call may be interrupted or restarted when the signal-catching routine returns, depending on the options in effect for the signal; see discussion of SA_RESTART in sigaction(2).

The implementation queues one SIGCHLD signal for each child process whose status has changed; if **wait**() returns because the status of a child process is available, the pending SIGCHLD signal associated with the process ID of the child process will be discarded. Any other pending SIGCHLD signals remain pending.

If SIGCHLD is blocked and **wait**() returns because the status of a child process is available, the pending SIGCHLD signal will be cleared unless another status of the child process is available.

RETURN VALUES

If **wait**() returns due to a stopped, continued, 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 **wait6**(), **wait4**(), **wait3**(), or **waitpid**() returns due to a stopped, continued, or terminated child process, the process ID of the child is returned to the calling process. If there are no children not previously awaited, -1 is returned with *errno* set to ECHILD. Otherwise, if WNOHANG is specified and there are no stopped, continued or exited children, 0 is returned. If an error is detected or a caught signal aborts the call, a value of -1 is returned and *errno* is set to indicate the error.

If **waitid**() returns because one or more processes have a state change to report, 0 is returned. If an error is detected, a value of -1 is returned and *errno* is set to indicate the error. If WNOHANG is specified and there are no stopped, continued or exited children, 0 is returned. The *si_signo* and *si_pid* fields of *infop* must be checked against zero to determine if a process reported status.

The **wait**() family of functions will not return a child process created with pdfork(2) unless specifically directed to do so by specifying its process ID.

ERRORS

The **wait**() function will fail and return immediately if:

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

[ECHILD] No status from the terminated child process is available because the calling process has asked the system to discard such status by ignoring the signal SIGCHLD or setting the flag SA_NOCLDWAIT for that signal.

[EFAULT] The *status* or *rusage* argument points to an illegal address. (May not be detected before exit of a child process.)

[EINTR] The call was interrupted by a caught signal, or the signal did not have the SA_RESTART flag set.

[EINVAL] An invalid value was specified for *options*, or *idtype* and *id* do not specify a valid set of processes.

SEE ALSO

_exit(2), procctl(2), ptrace(2), sigaction(2), exit(3), siginfo(3)

STANDARDS

The **wait**(), **waitpid**(), and **waitid**() functions are defined by POSIX; **wait6**(), **wait4**(), and **wait3**() are not specified by POSIX. The **WCOREDUMP**() macro is an extension to the POSIX interface.

The ability to use the WNOWAIT flag with **waitpid**() is an extension; POSIX only permits this flag with **waitid**().

HISTORY

The **wait**() function appeared in Version 1 AT&T UNIX.