68659 **NAME**

68660 wait, waitpid — wait for a child process to stop or terminate

68661 **SYNOPSIS**

68662
```
#include <sys/wait.h>
```

68663
```
pid_t wait(int *stat_loc);
```
68664
```
pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

68665 **DESCRIPTION**

68666 The *wait*( ) and *waitpid*( ) functions shall obtain status information pertaining to one of the
68667 caller's child processes. Various options permit status information to be obtained for child
68668 processes that have terminated or stopped. If status information is available for two or more
68669 child processes, the order in which their status is reported is unspecified.

68670 The *wait*( ) function shall suspend execution of the calling thread until status information for one
68671 of the terminated child processes of the calling process is available, or until delivery of a signal
68672 whose action is either to execute a signal-catching function or to terminate the process. If more
68673 than one thread is suspended in *wait*( ) or *waitpid*( ) awaiting termination of the same process,
68674 exactly one thread shall return the process status at the time of the target process termination. If
68675 status information is available prior to the call to *wait*( ), return shall be immediate.

68676 The *waitpid*( ) function shall be equivalent to *wait*( ) if the *pid* argument is (**pid_t**)–1 and the
68677 *options* argument is 0. Otherwise, its behavior shall be modified by the values of the *pid* and
68678 *options* arguments.

68679 The *pid* argument specifies a set of child processes for which *status* is requested. The *waitpid*( )
68680 function shall only return the status of a child process from this set:

68681 • If *pid* is equal to (**pid_t**)–1, *status* is requested for any child process. In this respect,
68682 *waitpid*( ) is then equivalent to *wait*( ).

68683 • If *pid* is greater than 0, it specifies the process ID of a single child process for which *status* is
68684 requested.

68685 • If *pid* is 0, *status* is requested for any child process whose process group ID is equal to that
68686 of the calling process.

68687 • If *pid* is less than (**pid_t**)–1, *status* is requested for any child process whose process group
68688 ID is equal to the absolute value of *pid*.

68689 The *options* argument is constructed from the bitwise-inclusive OR of zero or more of the
68690 following flags, defined in the **<sys/wait.h>** header:

68691 XSI WCONTINUED The *waitpid*( ) function shall report the status of any continued child process
68692 specified by *pid* whose status has not been reported since it continued from a
68693 job control stop.

68694 WNOHANG The *waitpid*( ) function shall not suspend execution of the calling thread if
68695 *status* is not immediately available for one of the child processes specified by
68696 *pid*.

68697 WUNTRACED The status of any child processes specified by *pid* that are stopped, and whose
68698 status has not yet been reported since they stopped, shall also be reported to
68699 the requesting process.

68700 XSI If the calling process has SA_NOCLDWAIT set or has SIGCHLD set to SIG_IGN, and the process
68701 has no unwaited-for children that were transformed into zombie processes, the calling thread
68702 shall block until all of the children of the process containing the calling thread terminate, and
68703 *wait*( ) and *waitpid*( ) shall fail and set *errno* to [ECHILD].

If *wait*( ) or *waitpid*( ) return because the status of a child process is available, these functions shall return a value equal to the process ID of the child process. In this case, if the value of the argument *stat_loc* is not a null pointer, information shall be stored in the location pointed to by *stat_loc*. The value stored at the location pointed to by *stat_loc* shall be 0 if and only if the status returned is from a terminated child process that terminated by one of the following means:

1. The process returned 0 from *main*( ).
2. The process called *_exit*( ) or *exit*( ) with a *status* argument of 0.
3. The process was terminated because the last thread in the process terminated.

Regardless of its value, this information may be interpreted using the following macros, which are defined in **<sys/wait.h>** and evaluate to integral expressions; the *stat_val* argument is the integer value pointed to by *stat_loc*.

WIFEXITED(*stat_val*)
: Evaluates to a non-zero value if *status* was returned for a child process that terminated normally.

WEXITSTATUS(*stat_val*)
: If the value of WIFEXITED(*stat_val*) is non-zero, this macro evaluates to the low-order 8 bits of the *status* argument that the child process passed to *_exit*( ) or *exit*( ), or the value the child process returned from *main*( ).

WIFSIGNALED(*stat_val*)
: Evaluates to a non-zero value if *status* was returned for a child process that terminated due to the receipt of a signal that was not caught (see **<signal.h>**).

WTERMSIG(*stat_val*)
: If the value of WIFSIGNALED(*stat_val*) is non-zero, this macro evaluates to the number of the signal that caused the termination of the child process.

WIFSTOPPED(*stat_val*)
: Evaluates to a non-zero value if *status* was returned for a child process that is currently stopped.

WSTOPSIG(*stat_val*)
: If the value of WIFSTOPPED(*stat_val*) is non-zero, this macro evaluates to the number of the signal that caused the child process to stop.

XSI WIFCONTINUED(*stat_val*)
: Evaluates to a non-zero value if *status* was returned for a child process that has continued from a job control stop.

SPN It is unspecified whether the *status* value returned by calls to *wait*( ) or *waitpid*( ) for processes created by *posix_spawn*( ) or *posix_spawnp*( ) can indicate a WIFSTOPPED(*stat_val*) before subsequent calls to *wait*( ) or *waitpid*( ) indicate WIFEXITED(*stat_val*) as the result of an error detected before the new process image starts executing.

It is unspecified whether the *status* value returned by calls to *wait*( ) or *waitpid*( ) for processes created by *posix_spawn*( ) or *posix_spawnp*( ) can indicate a WIFSIGNALED(*stat_val*) if a signal is sent to the parent's process group after *posix_spawn*( ) or *posix_spawnp*( ) is called.

If the information pointed to by *stat_loc* was stored by a call to *waitpid*( ) that specified the
XSI WUNTRACED flag and did not specify the WCONTINUED flag, exactly one of the macros WIFEXITED(**stat_loc*), WIFSIGNALED(**stat_loc*), and WIFSTOPPED(**stat_loc*) shall evaluate to a non-zero value.

If the information pointed to by *stat_loc* was stored by a call to *waitpid*( ) that specified the WUNTRACED and WCONTINUED flags, exactly one of the macros WIFEXITED(**stat_loc*), WIFSIGNALED(**stat_loc*), WIFSTOPPED(**stat_loc*), and WIFCONTINUED(**stat_loc*) shall evaluate to a non-zero value.

If the information pointed to by *stat_loc* was stored by a call to *waitpid*( ) that did not specify the WUNTRACED or WCONTINUED flags, or by a call to the *wait*( ) function, exactly one of the macros WIFEXITED(**stat_loc*) and WIFSIGNALED(**stat_loc*) shall evaluate to a non-zero value.

If the information pointed to by *stat_loc* was stored by a call to *waitpid*( ) that did not specify the WUNTRACED flag and specified the WCONTINUED flag, or by a call to the *wait*( ) function, exactly one of the macros WIFEXITED(**stat_loc*), WIFSIGNALED(**stat_loc*), and WIFCONTINUED(**stat_loc*) shall evaluate to a non-zero value.

If _POSIX_REALTIME_SIGNALS is defined, and the implementation queues the SIGCHLD signal, then if *wait*( ) or *waitpid*( ) returns because the status of a child process is available, any pending SIGCHLD signal associated with the process ID of the child process shall be discarded. Any other pending SIGCHLD signals shall remain pending.

Otherwise, if SIGCHLD is blocked, if *wait*( ) or *waitpid*( ) return because the status of a child process is available, any pending SIGCHLD signal shall be cleared unless the status of another child process is available.

For all other conditions, it is unspecified whether child *status* will be available when a SIGCHLD signal is delivered.

There may be additional implementation-defined circumstances under which *wait*( ) or *waitpid*( ) report *status*. This shall not occur unless the calling process or one of its child processes explicitly makes use of a non-standard extension. In these cases the interpretation of the reported *status* is implementation-defined.

If a parent process terminates without waiting for all of its child processes to terminate, the remaining child processes shall be assigned a new parent process ID corresponding to an implementation-defined system process.

RETURN VALUE

If *wait*( ) or *waitpid*( ) returns because the status of a child process is available, these functions shall return a value equal to the process ID of the child process for which *status* is reported. If *wait*( ) or *waitpid*( ) returns due to the delivery of a signal to the calling process, −1 shall be returned and *errno* set to [EINTR]. If *waitpid*( ) 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*, 0 is returned. Otherwise, (**pid_t**)−1 shall be returned, and *errno* set to indicate the error.

ERRORS

The *wait*( ) function shall fail if:

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

[EINTR] The function was interrupted by a signal. The value of the location pointed to by *stat_loc* is undefined.

The *waitpid*( ) function shall fail if:

[ECHILD] The process specified by *pid* does not exist or is not a child of the calling process, or the process group specified by *pid* does not exist or does not have any member process that is a child of the calling process.

68792 [EINTR] The function was interrupted by a signal. The value of the location pointed to
68793 by *stat_loc* is undefined.

68794 [EINVAL] The *options* argument is not valid.

68795 **EXAMPLES**

68796 **Waiting for a Child Process and then Checking its Status**

68797 The following example demonstrates the use of *waitpid*(), *fork*(), and the macros used to
68798 interpret the status value returned by *waitpid*() (and *wait*()). The code segment creates a child
68799 process which does some unspecified work. Meanwhile the parent loops performing calls to
68800 *waitpid*() to monitor the status of the child. The loop terminates when child termination is
68801 detected.

```
68802 #include <stdio.h>
68803 #include <stdlib.h>
68804 #include <unistd.h>
68805 #include <sys/wait.h>
68806 ...

68807 pid_t child_pid, wpid;
68808 int status;

68809 child_pid = fork();
68810 if (child_pid == -1) {      /* fork() failed */
68811     perror("fork");
68812     exit(EXIT_FAILURE);
68813 }

68814 if (child_pid == 0) {       /* This is the child */
68815     /* Child does some work and then terminates */
68816     ...

68817 } else {                    /* This is the parent */
68818     do {
68819         wpid = waitpid(child_pid, &status, WUNTRACED
68820 #ifdef WCONTINUED       /* Not all implementations support this */
68821         | WCONTINUED
68822 #endif
68823         );
68824         if (wpid == -1) {
68825             perror("waitpid");
68826             exit(EXIT_FAILURE);
68827         }

68828         if (WIFEXITED(status)) {
68829             printf("child exited, status=%d\n", WEXITSTATUS(status));

68830         } else if (WIFSIGNALED(status)) {
68831             printf("child killed (signal %d)\n", WTERMSIG(status));

68832         } else if (WIFSTOPPED(status)) {
68833             printf("child stopped (signal %d)\n", WSTOPSIG(status));

68834 #ifdef WIFCONTINUED     /* Not all implementations support this */
68835         } else if (WIFCONTINUED(status)) {
68836             printf("child continued\n");
```



```
#endif
        } else {    /* Non-standard case -- may never happen */
            printf("Unexpected status (0x%x)\n", status);
        }
    } while (!WIFEXITED(status) && !WIFSIGNALED(status));
}
```

Waiting for a Child Process in a Signal Handler for SIGCHLD

The following example demonstrates how to use *waitpid*() in a signal handler for SIGCHLD without passing –1 as the *pid* argument. (See the APPLICATION USAGE section below for the reasons why passing a *pid* of –1 is not recommended.) The method used here relies on the standard behavior of *waitpid*() when SIGCHLD is blocked. On historical non-conforming systems, the status of some child processes might not be reported.

```
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

#define CHILDREN 10

static void
handle_sigchld(int signum, siginfo_t *sinfo, void *unused)
{
    int status;

    /*
     * Obtain status information for the child which
     * caused the SIGCHLD signal and write its exit code
     * to stdout.
    */
    if (sinfo->si_code != CLD_EXITED)
    {
        static char msg[] = "wrong si_code\n";
        write(2, msg, sizeof msg - 1);
    }
    else if (waitpid(sinfo->si_pid, &status, 0) == -1)
    {
        static char msg[] = "waitpid() failed\n";
        write(2, msg, sizeof msg - 1);
    }
    else if (!WIFEXITED(status))
    {
        static char msg[] = "WIFEXITED was false\n";
        write(2, msg, sizeof msg - 1);
    }
    else
    {
        int code = WEXITSTATUS(status);
        char buf[2];
        buf[0] = '0' + code;
```



```
68885                buf[1] = '\n';
68886                write(1, buf, 2);
68887            }
68888        }

68889        int
68890        main(void)
68891        {
68892            int i;
68893            pid_t pid;
68894            struct sigaction sa;

68895            sa.sa_flags = SA_SIGINFO;
68896            sa.sa_sigaction = handle_sigchld;
68897            sigemptyset(&sa.sa_mask);
68898            if (sigaction(SIGCHLD, &sa, NULL) == -1)
68899            {
68900                perror("sigaction");
68901                exit(EXIT_FAILURE);
68902            }

68903            for (i = 0; i < CHILDREN; i++)
68904            {
68905                switch (pid = fork())
68906                {
68907                case -1:
68908                    perror("fork");
68909                    exit(EXIT_FAILURE);
68910                case 0:
68911                    sleep(2);
68912                    _exit(i);
68913                }
68914            }

68915            /* Wait for all the SIGCHLD signals, then terminate on SIGALRM */
68916            alarm(3);
68917            for (;;)
68918                pause();
68919        }
```

APPLICATION USAGE

68920 68921 68922 68923 68924 68925 68926

Calls to *wait*( ) will collect information about any child process. This may result in interactions with other interfaces that may be waiting for their own children (such as by use of *system*( )). For this and other reasons it is recommended that portable applications not use *wait*( ), but instead use *waitpid*( ). For these same reasons, the use of *waitpid*( ) with a *pid* argument of –1, and the use of *waitid*( ) with the *idtype* argument set to P_ALL, are also not recommended for portable applications.

RATIONALE

68927 68928 68929 68930 68931 68932 68933

A call to the *wait*( ) or *waitpid*( ) function only returns *status* on an immediate child process of the calling process; that is, a child that was produced by a single *fork*( ) call (perhaps followed by an *exec* or other function calls) from the parent. If a child produces grandchildren by further use of *fork*( ), none of those grandchildren nor any of their descendants affect the behavior of a *wait*( ) from the original parent process. Nothing in this volume of POSIX.1-2008 prevents an implementation from providing extensions that permit a process to get *status* from a grandchild

or any other process, but a process that does not use such extensions must be guaranteed to see *status* from only its direct children.

The *waitpid*( ) function is provided for three reasons:

1. To support job control
2. To permit a non-blocking version of the *wait*( ) function
3. To permit a library routine, such as *system*( ) or *pclose*( ), to wait for its children without interfering with other terminated children for which the process has not waited

The first two of these facilities are based on the *wait3*( ) function provided by 4.3 BSD. The function uses the *options* argument, which is equivalent to an argument to *wait3*( ). The WUNTRACED flag is used only in conjunction with job control on systems supporting job control. Its name comes from 4.3 BSD and refers to the fact that there are two types of stopped processes in that implementation: processes being traced via the *ptrace*( ) debugging facility and (untraced) processes stopped by job control signals. Since *ptrace*( ) is not part of this volume of POSIX.1-2008, only the second type is relevant. The name WUNTRACED was retained because its usage is the same, even though the name is not intuitively meaningful in this context.

The third reason for the *waitpid*( ) function is to permit independent sections of a process to spawn and wait for children without interfering with each other. For example, the following problem occurs in developing a portable shell, or command interpreter:

```
stream = popen("/bin/true");
(void) system("sleep 100");
(void) pclose(stream);
```

On all historical implementations, the final *pclose*( ) fails to reap the *wait*( ) *status* of the *popen*( ).

The status values are retrieved by macros, rather than given as specific bit encodings as they are in most historical implementations (and thus expected by existing programs). This was necessary to eliminate a limitation on the number of signals an implementation can support that was inherent in the traditional encodings. This volume of POSIX.1-2008 does require that a *status* value of zero corresponds to a process calling *_exit*(0), as this is the most common encoding expected by existing programs. Some of the macro names were adopted from 4.3 BSD.

These macros syntactically operate on an arbitrary integer value. The behavior is undefined unless that value is one stored by a successful call to *wait*( ) or *waitpid*( ) in the location pointed to by the *stat_loc* argument. An early proposal attempted to make this clearer by specifying each argument as **stat_loc* rather than *stat_val*. However, that did not follow the conventions of other specifications in this volume of POSIX.1-2008 or traditional usage. It also could have implied that the argument to the macro must literally be **stat_loc*; in fact, that value can be stored or passed as an argument to other functions before being interpreted by these macros.

The extension that affects *wait*( ) and *waitpid*( ) and is common in historical implementations is the *ptrace*( ) function. It is called by a child process and causes that child to stop and return a *status* that appears identical to the *status* indicated by WIFSTOPPED. The *status* of *ptrace*( ) children is traditionally returned regardless of the WUNTRACED flag (or by the *wait*( ) function). Most applications do not need to concern themselves with such extensions because they have control over what extensions they or their children use. However, applications, such as command interpreters, that invoke arbitrary processes may see this behavior when those arbitrary processes misuse such extensions.

Implementations that support **core** file creation or other implementation-defined actions on termination of some processes traditionally provide a bit in the *status* returned by *wait*( ) to indicate that such actions have occurred.

Allowing the *wait*( ) family of functions to discard a pending SIGCHLD signal that is associated with a successfully waited-for child process puts them into the *sigwait*( ) and *sigwaitinfo*( ) category with respect to SIGCHLD.

This definition allows implementations to treat a pending SIGCHLD signal as accepted by the process in *wait*( ), with the same meaning of "accepted" as when that word is applied to the *sigwait*( ) family of functions.

Allowing the *wait*( ) family of functions to behave this way permits an implementation to be able to deal precisely with SIGCHLD signals.

In particular, an implementation that does accept (discard) the SIGCHLD signal can make the following guarantees regardless of the queuing depth of signals in general (the list of waitable children can hold the SIGCHLD queue):

1. If a SIGCHLD signal handler is established via *sigaction*( ) without the SA_RESETHAND flag, SIGCHLD signals can be accurately counted; that is, exactly one SIGCHLD signal will be delivered to or accepted by the process for every child process that terminates.
2. A single *wait*( ) issued from a SIGCHLD signal handler can be guaranteed to return immediately with status information for a child process.
3. When SA_SIGINFO is requested, the SIGCHLD signal handler can be guaranteed to receive a non-null pointer to a **siginfo_t** structure that describes a child process for which a wait via *waitpid*( ) or *waitid*( ) will not block or fail.
4. The *system*( ) function will not cause the SIGCHLD handler of a process to be called as a result of the *fork*( )/*exec* executed within *system*( ) because *system*( ) will accept the SIGCHLD signal when it performs a *waitpid*( ) for its child process. This is a desirable behavior of *system*( ) so that it can be used in a library without causing side-effects to the application linked with the library.

An implementation that does not permit the *wait*( ) family of functions to accept (discard) a pending SIGCHLD signal associated with a successfully waited-for child, cannot make the guarantees described above for the following reasons:

Guarantee #1
: Although it might be assumed that reliable queuing of all SIGCHLD signals generated by the system can make this guarantee, the counter-example is the case of a process that blocks SIGCHLD and performs an indefinite loop of *fork*( )/*wait*( ) operations. If the implementation supports queued signals, then eventually the system will run out of memory for the queue. The guarantee cannot be made because there must be some limit to the depth of queuing.

Guarantees #2 and #3
: These cannot be guaranteed unless the *wait*( ) family of functions accepts the SIGCHLD signal. Otherwise, a *fork*( )/*wait*( ) executed while SIGCHLD is blocked (as in the *system*( ) function) will result in an invocation of the handler when SIGCHLD is unblocked, after the process has disappeared.

Guarantee #4
: Although possible to make this guarantee, *system*( ) would have to set the SIGCHLD handler to SIG_DFL so that the SIGCHLD signal generated by its *fork*( ) would be discarded (the SIGCHLD default action is to be ignored), then restore it to its previous setting. This would have the undesirable side-effect of discarding all SIGCHLD signals pending to the process.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *exit*( ), *fork*( ), *system*( ), *waitid*( )

XBD Section 4.11 (on page 110), **<signal.h>**, **<sys/wait.h>**

CHANGE HISTORY

First released in Issue 1. Derived from Issue 1 of the SVID.

Issue 5

The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

Issue 6

The following new requirements on POSIX implementations derive from alignment with the Single UNIX Specification:

- The requirement to include **<sys/types.h>** has been removed. Although **<sys/types.h>** was required for conforming implementations of previous POSIX specifications, it was not required for UNIX applications.

The following changes were made to align with the IEEE P1003.1a draft standard:

- The processing of the SIGCHLD signal and the [ECHILD] error is clarified.

The semantics of WIFSTOPPED(*stat_val*), WIFEXITED(*stat_val*), and WIFSIGNALED(*stat_val*) are defined with respect to *posix_spawn*( ) or *posix_spawnp*( ) for alignment with IEEE Std 1003.1d-1999.

The DESCRIPTION is updated for alignment with the ISO/IEC 9899: 1999 standard.

IEEE Std 1003.1-2001/Cor 2-2004, item XSH/TC2/D6/145 is applied, adding the example to the EXAMPLES section.

Issue 7

SD5-XSH-ERN-202 is applied.

APPLICATION USAGE is added, recommending that the *wait*( ) function not be used.

An additional example for *waitpid*( ) is added.

69052 **NAME**

69053 waitid — wait for a child process to change state

69054 **SYNOPSIS**

69055 `#include <sys/wait.h>`

69056 `int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);`

69057 **DESCRIPTION**

69058 The *waitid*( ) function shall suspend the calling thread until one child of the process containing
69059 the calling thread changes state. It records the current state of a child in the structure pointed to
69060 by *infop*. The fields of the structure pointed to by *infop* are filled in as described for the
69061 SIGCHLD signal in **<signal.h>**. If a child process changed state prior to the call to *waitid*( ),
69062 *waitid*( ) shall return immediately. If more than one thread is suspended in *wait*( ), *waitid*( ), or
69063 *waitpid*( ) waiting for termination of the same process, exactly one thread shall return the process
69064 status at the time of the target process termination.

69065 The *idtype* and *id* arguments are used to specify which children *waitid*( ) waits for.

69066 If *idtype* is P_PID, *waitid*( ) shall wait for the child with a process ID equal to (**pid_t**)*id*.

69067 If *idtype* is P_PGID, *waitid*( ) shall wait for any child with a process group ID equal to (**pid_t**)*id*.

69068 If *idtype* is P_ALL, *waitid*( ) shall wait for any children and *id* is ignored.

69069 The *options* argument is used to specify which state changes *waitid*( ) shall wait for. It is formed
69070 by OR'ing together the following flags:

69071 WCONTINUED Status shall be returned for any child that was stopped and has been
69072 continued.

69073 WEXITED Wait for processes that have exited.

69074 WNOHANG Do not hang if no status is available; return immediately.

69075 WNOWAIT Keep the process whose status is returned in *infop* in a waitable state. This
69076 shall not affect the state of the process; the process may be waited for again
69077 after this call completes.

69078 WSTOPPED Status shall be returned for any child that has stopped upon receipt of a signal.

69079 Applications shall specify at least one of the flags WEXITED, WSTOPPED, or WCONTINUED to
69080 be OR'ed in with the *options* argument.

69081 The application shall ensure that the *infop* argument points to a **siginfo_t** structure. If *waitid*( )
69082 returns because a child process was found that satisfied the conditions indicated by the
69083 arguments *idtype* and *options*, then the structure pointed to by *infop* shall be filled in by the
69084 system with the status of the process. The *si_signo* member shall always be equal to SIGCHLD.

69085 **RETURN VALUE**

69086 If WNOHANG was specified and status is not available for any process specified by *idtype* and
69087 *id*, 0 shall be returned. If *waitid*( ) returns due to the change of state of one of its children, 0 shall
69088 be returned. Otherwise, –1 shall be returned and *errno* set to indicate the error.

69089 **ERRORS**

69090 The *waitid*( ) function shall fail if:

69091 [ECHILD] The calling process has no existing unwaited-for child processes.

69092 [EINTR] The *waitid*( ) function was interrupted by a signal.

69093 [EINVAL] An invalid value was specified for *options*, or *idtype* and *id* specify an invalid
69094 set of processes.

69095 **EXAMPLES**

69096 None.

69097 **APPLICATION USAGE**

69098 Calls to *waitid*( ) with *idtype* equal to P_ALL will collect information about any child process.
69099 This may result in interactions with other interfaces that may be waiting for their own children
69100 (such as by use of *system*( )). For this reason it is recommended that portable applications not
69101 use *waitid*( ) with idtype of P_ALL. See also APPLICATION USAGE for *wait*( ).

69102 **RATIONALE**

69103 None.

69104 **FUTURE DIRECTIONS**

69105 None.

69106 **SEE ALSO**

69107 *exec*, *exit*( ), *wait*( )

69108 XBD **<signal.h>**, **<sys/wait.h>**

69109 **CHANGE HISTORY**

69110 First released in Issue 4, Version 2.

69111 **Issue 5**

69112 Moved from X/OPEN UNIX extension to BASE.

69113 The DESCRIPTION is updated for alignment with the POSIX Threads Extension.

69114 **Issue 6**

69115 The normative text is updated to avoid use of the term "must" for application requirements.

69116 **Issue 7**

69117 Austin Group Interpretation 1003.1-2001 #060 is applied, updating the DESCRIPTION.

69118 The *waitid*( ) function is moved from the XSI option to the Base.

69119 APPLICATION USAGE is added, recommending that the *waitid*( ) function not be used with
69120 *idtype* equal to P_ALL.

69121 The description of the WNOHANG flag is updated.

69122 **NAME**

69123 waitpid — wait for a child process to stop or terminate

69124 **SYNOPSIS**

69125 ```
#include <sys/wait.h>
```

69126 ```
pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

69127 **DESCRIPTION**

69128 Refer to *wait( )*.

