This request restarts a traced process, given in *pid*, which has been stopped. The *data* parameter may point to a signal ID to deliver to the traced process; if it is zero or SIGSTOP, no signal is delivered to the child. The *addr* is ignored.

PTRACE_DETACH

This request performs the same function, in the same way, as PTRACE_CONT, except that the tracing relationship between the tracing and traced processes is also undone. If the trace was initiated using PTRACE_ATTACH, the original parent-child relationships that existed beforehand are restored.

PTRACE_KILL

This request causes a SIGKILL signal to be sent to the traced process specified in *pid*. The *addr* and *data* parameters are ignored.

PTRACE_PEEKTEXT

This request reads a word at the location *addr* of the traced process *pid*, and returns it to the caller. The *data* parameter is ignored.

PTRACE_PEEKDATA

This request performs identically to the PTRACE_PEEKTEXT request.

PTRACE_PEEKUSER

This request reads a word at offset *addr* in the USER area of the traced process *pid*. The offset must be word-aligned. The *data* parameter is ignored.

PTRACE_POKETEXT

This request writes the word pointed at by *data* to the location *addr* of the traced process *pid*.

PTRACE_POKEDATA

This request performs identically to the PTRACE_POKETEXT request.

PTRACE_POKEUSER

This request writes the word pointed at by *data* to offset *addr* in the USER area of the traced process *pid*. The offset must be word-aligned. Implementations may choose to disallow some modifications to the USER area.

PTRACE_GETREGS

This request copies the general purpose registers from the traced process *pid* to the tracing process at location *data*. This parameter may not be available on all architectures. The *addr* parameter is ignored.

PTRACE_GETFPREGS

This request copies the floating point registers from the traced process *pid* to the tracing process at location *data*. This parameter may not be available on all architectures. The *addr* parameter is ignored.

PTRACE_SETREGS

This request writes the general purpose registers to the traced process *pid* from the tracing process at location *data*. This parameter may not be available on all architectures. Implementations may choose to disallow some register modifications. The *addr* parameter is ignored.

PTRACE_SETFPREGS

This request writes the floating point registers to the traced process *pid* from the tracing process at location *data*. This parameter may not be available on all architectures. Implementations may choose to disallow some register modifications. The *addr* parameter is ignored.

PTRACE_GETSIGINFO

This request writes information about the signal which caused the traced process *pid* to stop to the tracing process at location *data*, as a siginfo_t. The *addr* parameter is ignored.

PTRACE_SETSIGINFO

This request writes signal information to the traced process *pid* from a siginfo_t structure pointed at by *data*, such that it will be used as the signal information by the traced process when it is resumed. The *addr* parameter is ignored.

PTRACE_GETEVENTMSG

This request stores information about the most recent ptrace event for the traced process *pid* in the unsigned long pointed at by *data*. For PTRACE_EVENT_EXIT, this is the exit status of the traced process. For PTRACE_EVENT_FORK, PTRACE_EVENT_VFORK, or PTRACE_EVENT_CLONE, this is the PID of the newly created process. The *addr* parameter is ignored.

PTRACE_SYSCALL

This request performs the same function, in the same way, as PTRACE_CONT, but with the additional step of causing the traced process to stop at the next entry to or exit from a system call. The usual events that would also cause the traced process to stop continue to do so.

PTRACE_SINGLESTEP

This request performs the same function, in the same way, as PTRACE_CONT, but with the additional step of causing the traced process to stop after execution of a single instruction. The usual events that would also cause the traced process to stop continue to do so.

PTRACE_SYSEMU

This request performs the same function, in the same way, as PTRACE_CONT, but with the additional step of causing the traced process to stop on entry to the next syscall, which will then not be executed.

PTRACE_SYSEMU_SINGLESTEP

This request performs the same function, in the same way, as PTRACE_CONT, but with the additional step of causing the traced process to stop on entry to the next syscall, which will then not be executed. If the next instruction is not itself a syscall, the traced process will stop after a single instruction is executed.

PTRACE_SETOPTIONS

This request sets ptrace() options for the traced process *pid* from the location pointed to by *data*. The *addr* is ignored. This location is interpreted as a bitmask of options, as defined by the following flags:

PTRACE_O_TRACESYSGOOD

This option, when set, causes syscall traps to set bit 7 in the signal number.

PTRACE_O_TRACEFORK

This option, when set, causes the traced process to stop when it calls fork(2). The original traced process will stop with SIGTRAP | PTRACE_EVENT_FORK << 8, and the new process will be stopped with SIGSTOP. The new process will also be traced by the tracing process, as if the tracing process had sent the PTRACE_ATTACH request for that process. The PID of the new process may be retrieved with the PTRACE_GETEVENTMSG request.

PTRACE_O_TRACEVFORK

This option, when set, causes the traced process to stop when it calls vfork(2). The original traced process will stop with SIGTRAP | PTRACE_EVENT_VFORK << 8, and the new process will be stopped with SIGSTOP. The new process will also be traced by the tracing process, as if the tracing process had sent the PTRACE_ATTACH request for that process. The PID of the new process may be retrieved with the PTRACE_GETEVENTMSG request.

PTRACE_O_TRACECLONE

This option, when set, causes the traced process to stop when it calls clone(2). The original traced process will stop with SIGTRAP | PTRACE_EVENT_CLONE << 8, and the new process will be stopped with SIGSTOP. The new process will also be traced by the tracing process, as if the tracing process had sent the PTRACE_ATTACH request for that process. The PID of the new process may be retrieved with the PTRACE_GETEVENTMSG request. Under certain circumstances, clone(2) calls by the traced process will generate events and information consistent with the PTRACE_O_TRACEVFORK or PTRACE_O_TRACEFORK options above.

PTRACE_O_TRACEEXEC

This option, when set, causes the traced process to stop when it calls execve(2). The traced process will stop with SIGTRAP | PTRACE_EVENT_EXEC << 8.

PTRACE_O_TRACEVFORKDONE

This option, when set, causes the traced process to stop at the completion of its next vfork(2) call. The traced process will stop with SIGTRAP | PTRACE_EVENT_EXEC << 8.

`PTRACE_O_TRACEEXIT`

This option, when set, causes the traced process to stop upon exit. The traced process will stop with SIGTRAP | PTRACE_EVENT_EXIT << 8, and its exit status can be retrieved with the PTRACE_GETEVENTMSG request. The stop is guaranteed to be early in the process exit process, meaning that information such as register status at exit is preserved. Upon continuing, the traced process will immediately exit.

Return Value

On success, `ptrace()` shall return the requested data for `PTRACE_PEEK` requests, or zero for all other requests. On error, all requests return -1, with `errno` set to an appropriate value. Note that -1 may be a valid return value for `PTRACE_PEEK` requests; the application is responsible for distinguishing between an error condition and a valid return value in that case.

Errors

On error, `ptrace()` shall set `errno` to one of the regular error values below:

`EBUSY`

An error occurred while allocating or freeing a debug register.

`EFAULT`

The request attempted to read from or write to an invalid area in the memory space of the tracing or traced process.

`EIO`

The request was invalid, or it attempted to read from or write to an invalid area in the memory space of the tracing or traced process, or it violated a word-alignment boundary, or an invalid signal was given to continue the traced process.

`EINVAL`

An attempt was made to set an invalid option.

`EPERM`

The request to trace a process was denied by the system.

`ESRCH`

The process requested does not exist, is not being traced by the current process, or is not stopped.

putwc_unlocked

Name

putwc_unlocked — non-thread-safe putwc

Description

putwc_unlocked() is the same as putwc(), except that it need not be thread-safe. That is, it may only be invoked in the ways which are legal for getc_unlocked().

putwchar_unlocked

Name

putwchar_unlocked — non-thread-safe putwchar

Description

putwchar_unlocked() is the same as putwchar(), except that it need not be thread-safe. That is, it may only be invoked in the ways which are legal for getc_unlocked().

pwrite64

Name

pwrite64 — write on a file (Large File Support)

Synopsis

```
#include <unistd.h>
ssize_t pwrite64(int fd, const void * buf, size_t count, off64_t offset);
```

Description

pwrite64() shall write *count* bytes from *buf* to the file associated with the open file descriptor *fd*, at the position specified by *offset*, without changing the file position.

pwrite64() is a large-file version of the pwrite() function as defined in POSIX 1003.1-2008 (ISO/IEC 9945-2009). It differs from pwrite() in that the *offset* parameter is an off64_t instead of an off_t

Return Value

On success, pwrite64() shall return the number of bytes actually written. Otherwise pwrite() shall return -1 and set errno to indicate the error.

Errors

See pwrite() for possible error values.

random_r

Name

`random_r` — reentrantly generate pseudorandom numbers in a uniform distribution

Synopsis

```
#include <stdlib.h>
int random_r(struct random_data * buffer, int32_t * result);
```

Description

The interface `random_r()` shall function in the same way as the interface `random()`, except that `random_r()` shall use the data in *`buffer`* instead of the global random number generator state.

Before it is used, *`buffer`* must be initialized, for example, by calling `lcong48_r()`, `seed48_r()`, or `srand48_r()`, or by filling it with zeroes.

readdir64_r

Name

`readdir64_r` — read a directory (Large File Support)

Synopsis

```
#include <dirent.h>
int readdir64_r(DIR * dirp, struct dirent64 * entry, struct dirent64
* * result);
```

Description

The `readdir64_r()` function is a large file version of `readdir_r()`. It shall behave as `readdir_r()` in POSIX 1003.1-2008 (ISO/IEC 9945-2009), except that the *`entry`* and *`result`* arguments are `dirent64` structures rather than `dirent`.

Return Value

See `readdir_r()`.

Errors

See `readdir_r()`.

regexec

Name

`regexec` — regular expression matching

Description

The `regexec()` function shall behave as specified in *POSIX 1003.1-2008 (ISO/IEC 9945-2009)*, except with differences as listed below.

Differences

Certain aspects of regular expression matching are optional; see Regular Expressions.

scandir64

Name

`scandir64` — scan a directory (Large File Support)

Synopsis

```
#include <dirent.h>
int scandir64(const char * dir, const struct dirent64 ** namelist,
int (*sel) (const struct dirent64 *), int (*compar) (const struct
dirent64 **, const struct dirent64 **));
```

Description

`scandir64()` is a large-file version of the `scandir()` function as defined in POSIX 1003.1-2008 (ISO/IEC 9945-2009). If differs only in that the *`namelist`* and the paramters to the selection function *`sel`* and comparison function *`compar`* are of type `dirent64` instead of type `dirent`.

scanf

Name

`scanf` — convert formatted input

Description

The `scanf()` family of functions shall behave as described in POSIX 1003.1-2008 (ISO/IEC 9945-2009), except as noted below.

Differences

The `%s`, `%S` and `%[` conversion specifiers shall accept an option length modifier `a`, which shall cause a memory buffer to be allocated to hold the string converted. In such a case, the argument corresponding to the conversion specifier should be a reference to a pointer value that will receive a pointer to the allocated buffer. If there is insufficient memory to allocate a buffer, the function may set `errno` to ENOMEM and a conversion error results.

> **Note:** This directly conflicts with the ISO C (1999) usage of `%a` as a conversion specifier for hexadecimal float values. While this conversion specifier should be supported, a format specifier such as `"%aseconds"` will have a different meaning on an LSB conforming system.

sched_getaffinity

Name

`sched_getaffinity` — retrieve the affinity mask of a process

Synopsis

```
#include <sched.h>
int sched_getaffinity(pid_t pid, unsigned int cpusetsize, cpu_set_t *
mask);
```

Description

`sched_getaffinity()` shall retrieve the affinity mask of a process.

The parameter *`pid`* specifies the ID for the process. If *`pid`* is `0`, then the calling process is specified instead.

The parameter *`cpusetsize`* specifies the length of the data pointed to by *`mask`*, in bytes. Normally, this parameter is specified as `sizeof(cpu_set_t)`.

Return Value

On success, `sched_getaffinity()` shall return `0`, and the structure pointed to by *`mask`* shall contain the affinity mask of the specified process.

On failure, `sched_getaffinity()` shall return `-1` and set `errno` as follows.

Errors

EFAULT

Bad address.

EINVAL

`mask` does not specify any processors that exist in the system, or *`cpusetsize`* is smaller than the kernel's affinity mask.

ESRCH

The specified process could not be found.

See Also

`sched_setscheduler()`, `sched_setaffinity()`.

sched_setaffinity

Name

`sched_setaffinity` — set the CPU affinity mask for a process

Synopsis

```
#include <sched.h>
int sched_setaffinity(pid_t pid, unsigned int cpusetsize, cpu_set_t *
mask);
```

Description

`sched_setaffinity()` shall set the CPU affinity mask for a process.

The parameter *`pid`* specifies the ID for the process. If *`pid`* is 0, then the calling process is specified instead.

The parameter *`cpusetsize`* specifies the length of the data pointed to by *`mask`*, in bytes. Normally, this parameter is specified as `sizeof(cpu_set_t)`.

The parameter *`mask`* specifies the new value for the CPU affinity mask. The structure pointed to by *`mask`* represents the set of CPUs on which the process may run. If *`mask`* does not specify one of the CPUs on which the specified process is currently running, then `sched_setaffinity()` shall migrate the process to one of those CPUs.

Setting the mask on a multiprocessor system can improve performance. For example, setting the mask for one process to specify a particular CPU, and then setting the mask of all other processes to exclude the CPU, dedicates the CPU to the process so that the process runs as fast as possible. This technique also prevents loss of performance in case the process terminates on one CPU and starts again on another, invalidating cache.

Return Value

On success, `sched_setaffinity()` shall return `0`.

On failure, `sched_setaffinity()` shall return `-1` and set `errno` as follows.

Errors

EFAULT

Bad address.

EINVAL

`mask` does not specify any processors that exist in the system, or *`cpusetsize`* is smaller than the kernel's affinity mask.

EPERM

Insufficient privileges. The effective user ID of the process calling `sched_setaffinity()` is not equal to the user ID or effective user ID of the specified process, and the calling process does not have appropriate privileges.

ESRCH

The specified process could not be found.

See Also

`sched_setscheduler()`, `sched_getaffinity()`.

sched_setscheduler

Name

`sched_setscheduler` — set scheduling policy and parameters

Synopsis

```
#include <sched.h>
int sched_setscheduler(pid_t pid, int policy, const struct sched_param
* param);
```

Description

The `sched_setscheduler()` shall behave as described in POSIX 1003.1-2008 (ISO/IEC 9945-2009), except as noted below.

Return Value

On success, 0 is returned instead of the former scheduling policy.

seed48_r

Name

`seed48_r` — reentrantly generate pseudorandom numbers in a uniform distribution

Synopsis

```
#include <stdlib.h>
int seed48_r(unsigned short[3] seed16v, struct drand48_data * buffer);
```

Description

The interface `seed48_r()` shall function in the same way as the interface `seed48()`, except that `seed48_r()` shall use the data in *`buffer`* instead of the global random number generator state.