NAME

system_data_types − overview of system data types

DESCRIPTION

sigevent

Include: <*signal.h*>. Alternatively, <*aio.h*>, <*mqueue.h*>, or<*time .h*>.

```
struct sigevent {
    int             sigev_notify; /* Notification type */
    int             sigev_signo;  /* Signal number */
    union sigval    sigev_value;  /* Signal value */
    void          (*sigev_notify_function)(union sigval);
                                  /* Notification function */
    pthread_attr_t *sigev_notify_attributes;
                                  /* Notification attributes */
};
```

For further details about this type, see **sigevent**(7).

Versions: <*aio.h*> and <*time.h*> define *sigevent* since POSIX.1-2008.

Conforming to: POSIX.1-2001 and later.

See also: **timer_create**(2), **getaddrinfo_a**(3), **lio_listio**(3), **mq_notify**(3)

See also the *aiocb* structure in this page.

siginfo_t

Include: <*signal.h*>. Alternatively, <*sys/wait.h*>.

```
typedef struct {
    int      si_signo;  /* Signal number */
    int      si_code;   /* Signal code */
    pid_t    si_pid;    /* Sending process ID */
    uid_t    si_uid;    /* Real user ID of sending process */
    void    *si_addr;   /* Address of faulting instruction */
    int      si_status; /* Exit value or signal */
    union sigval si_value;  /* Signal value */
} siginfo_t;
```

Information associated with a signal. For further details on this structure (including additional, Linux-specific fields), see **sigaction**(2).

Conforming to: POSIX.1-2001 and later.

See also: **pidfd_send_signal**(2), **rt_sigqueueinfo**(2), **sigaction**(2), **sigwaitinfo**(2), **psiginfo**(3)

sigset_t

Include: <*signal.h*>. Alternatively, <*spawn.h*>, or<*sys/select.h*>.

This is a type that represents a set of signals. According to POSIX, this shall be an integer or structure type.

Conforming to: POSIX.1-2001 and later.

See also: **epoll_pwait**(2), **ppoll**(2), **pselect**(2), **sigaction**(2), **signalfd**(2), **sigpending**(2), **sigprocmask**(2), **sigsuspend**(2), **sigwaitinfo**(2), **signal**(7)

sigval

Include: <*signal.h*>.

```
union sigval {
    int     sigval_int; /* Integer value */
    void   *sigval_ptr; /* Pointer value */
};
```

Data passed with a signal.

Conforming to: POSIX.1-2001 and later.

See also: **pthread_sigqueue**(3), **sigqueue**(3), **sigevent**(7)

See also the *sigevent* structure and the *siginfo_t* type in this page.

NOTES

The structures described in this manual page shall contain, at least, the members shown in their definition, in no particular order.

Most of the integer types described in this page don't have a corresponding length modifier for the **printf**(3) and the **scanf**(3) families of functions. To print a value of an integer type that doesn't have a length modifier, it should be converted to *intmax_t* or *uintmax_t* by an explicit cast. To scan into a variable of an integer type that doesn't have a length modifier, an intermediate temporary variable of type *intmax_t* or *uintmax_t* should be used. When copying from the temporary variable to the destination variable, the value could overflow. If the type has upper and lower limits, the user should check that the value is within those limits, before actually copying the value. The example below shows how these conversions should be done.

Conventions used in this page

In "Conforming to" we only concern ourselves with C99 and later and POSIX.1-2001 and later. Some types may be specified in earlier versions of one of these standards, but in the interests of simplicity we omit details from earlier standards.

In "Include", we first note the "primary" header(s) that define the type according to either the C or POSIX.1 standards. Under "Alternatively", we note additional headers that the standards specify shall define the type.

EXAMPLES

The program shown below scans from a string and prints a value stored in a variable of an integer type that doesn't have a length modifier. The appropriate conversions from and to *intmax_t*, and the appropriate range checks, are used as explained in the notes section above.

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>

int
main (void)
{
    static const char *const str = "500000 us in half a second";
    suseconds_t us;
    intmax_t    tmp;

    /* Scan the number from the string into the temporary variable. */

    sscanf(str, "%jd", &tmp);

    /* Check that the value is within the valid range of suseconds_t. */

    if (tmp < -1 || tmp > 1000000) {
        fprintf(stderr, "Scanned value outside valid range!\n");
        exit(EXIT_FAILURE);
    }

    /* Copy the value to the suseconds_t variable 'us'. */
```

```
        us = tmp;

        /* Even though suseconds_t can hold the value -1, this isn't
           a sensible number of microseconds. */

        if (us < 0) {
            fprintf(stderr, "Scanned value shouldn't be negative!\n");
            exit(EXIT_FAILURE);
        }

        /* Print the value. */

        printf("There are %jd microseconds in half a second.\n",
                (intmax_t) us);

        exit(EXIT_SUCCESS);
    }
```

SEE ALSO

feature_test_macros(7), **standards**(7)