**NAME**

getspnam, getspnam_r, getspent, getspent_r, setspent, endspent, fgetspent, fgetspent_r – get password entry

**SYNOPSIS**

**#include <shadow.h>**

**struct spwd *getspnam(const char ***name***);**

**struct spwd *getspnam_r(const char ***name,* **struct spwd ***result,* **char ***buffer,* **int** *buflen***);**

**struct spwd *getspent(void);**

**struct spwd *getspent_r(struct spwd ***result,* **char ***buffer,* **int** *buflen***);**

**void setspent(void);**

**void endspent(void);**

**struct spwd *fgetspent(FILE ***fp***);**

**struct spwd *fgetspent_r(FILE ***fp,* **struct spwd ***result,* **char ***buffer,* **int** *buflen***);**

**MT-LEVEL**

See the subsection "Reentrant Interfaces" in the **DESCRIPTION** section of this page.

**DESCRIPTION**

These functions are used to obtain shadow password entries. An entry may come from any of the sources for **shadow** specified in the **/etc/nsswitch.conf** file (see **nsswitch.conf**(4)).

**getspnam( )** searches for a shadow password entry with the login name specified by the character string parameter *name*.

The functions **setspent( )**, **getspent( )**, and **endspent( )** are used to enumerate shadow password entries from the database.

**setspent( )** sets (or resets) the enumeration to the beginning of the set of shadow password entries. This function should be called before the first call to **getspent( )**. Calls to **getspnam( )** leave the enumeration position in an indeterminate state.

Successive calls to **getspent( )** return either successive entries or NULL, indicating the end of the enumeration.

**endspent( )** may be called to indicate that the caller expects to do no further shadow password retrieval operations; the system may then close the shadow password file, deallocate resources it was using, and so forth. It is still allowed, but possibly less efficient, for the process to call more shadow password functions after calling **endspent( )**.

**fgetspent( )**, unlike the other functions above, does not use **nsswitch.conf**; it reads and parses the next line from the stream *f*, which is assumed to have the format of the **shadow** file (see **shadow**(4)).

**Reentrant Interfaces**

The functions **getspnam( )**, **getspent( )**, and **fgetspent( )** use static storage that is re-used in each call, making these routines unsafe for use in multithreaded applications.

The functions:
**getspnam_r( )**,
**getspent_r( )**,
and
**fgetspent_r( )**
provide reentrant interfaces for these operations.

Each reentrant interface performs the same operation as its non-reentrant counterpart, named by removing the ''**_r**'' suffix. The reentrant interfaces, however, use buffers supplied by the caller to store returned results, and are safe for use in both single-threaded and multithreaded applications.

Each reentrant interface takes the same parameters as its non-reentrant counterpart, as well as the following additional parameters. The parameter *result* must be a pointer to a **struct spwd** structure allocated by the caller. On successful completion, the function returns the shadow password entry in this structure. The parameter *buffer* must be a pointer to a buffer supplied by the caller. This buffer is used as storage space for the shadow password data. All of the pointers within the returned **struct spwd** *result* point to data stored within this buffer (see **RETURN VALUES**). The buffer must be large enough to hold all of the data associated with the shadow password entry. The parameter *buflen* should give the size in bytes of the buffer indicated by *buffer.*

For enumeration in multithreaded applications, the position within the enumeration is a process-wide property shared by all threads. **setspent( )** may be used in a multithreaded application but resets the enumeration position for all threads. If multiple threads interleave calls to **getspent_r( )**, the threads will enumerate disjoint subsets of the shadow password database.

Like its non-reentrant counterpart, **getspnam_r( )** leaves the enumeration position in an indeterminate state.

## RETURN VALUES

Password entries are represented by the **struct spwd** structure defined in **<shadow.h>**:

```
struct spwd{
      char *sp_namp;      /* login name */
      char *sp_pwdp;      /* encrypted passwd */
      long sp_lstchg;     /* date of last change */
      long sp_min;        /* min days to passwd change */
      long sp_max;                /* max days to passwd change*/
      long sp_warn;       /* warning period */
      long sp_inact;      /* max days inactive */
      long sp_expire;     /* account expiry date */
      unsigned long sp_flag;  /* not used */
};
```

See **shadow**(4) for more information on the interpretation of this information.

The functions **getspnam( )**and **getspnam_r( )** each return a pointer to a **struct spwd** if they successfully locate the requested entry; otherwise they return NULL.

The functions **getspent( )**,**getspent_r( )**, **fgetspent( )**, and **fgetspent( )** each return a pointer to a **struct spwd** if they successfully enumerate an entry; otherwise they return NULL, indicating the end of the enumeration.

The functions **getspnam( )**, **getspent( )**, and **fgetspent( )** use static storage, so returned data must be copied before a subsequent call to any of these functions if the data is to be saved.

When the pointer returned by the reentrant functions **getspnam_r( )**, **getspent_r( )**, and **fgetspent_r( )** is non-NULL, it is always equal to the *result* pointer that was supplied by the caller.

## ERRORS

The reentrant functions **getspnam_r( )**, **getspent_r( )**, and **fgetspent_r( )** will return NULL and set **errno** to **ERANGE** if the length of the buffer supplied by caller is not large enough to store the result. See **intro**(2) for the proper usage and interpretation of **errno** in multithreaded applications.

## FILES

**/etc/shadow**
**/etc/nsswitch.conf**
**/etc/passwd**

## SEE ALSO

**nispasswd**(1), **passwd**(1), **yppasswd**(1), **intro**(3) **getlogin**(3C), **getpwnam**(3C), **nsswitch.conf**(4), **passwd**(4), **shadow**(4)

## WARNINGS

The reentrant interfaces **getspnam_r( )**, **getspent_r( )**, and **fgetspent_r( )** are included in this release on an uncommitted basis only, and are subject to change or removal in future minor releases.

## NOTES

Programs that use the interfaces described in this manual page cannot be linked statically since the implementations of these functions employ dynamic loading and linking of shared objects at run time.

When compiling multithreaded applications, see **intro**(3), *Notes On Multithread Applications*, for information about the use of the **_REENTRANT** flag.

Use of the enumeration interfaces **getspent( )** and **getspent_r( )** is not recommended; enumeration is supported for the shadow file, NIS, and NIS+, but in general is not efficient and may not be supported for all database sources. The semantics of enumeration are discussed further in **nsswitch.conf**(4).

Access to shadow password information may be restricted in a manner depending on the database source being used. Access to the **/etc/shadow** file is generally restricted to processes running as the super-user (root). Other database sources may impose stronger or less stringent restrictions.

When NIS is used as the database source, the information for the shadow password entries is obtained from the ''passwd.byname'' map. This map stores only the information for the **sp_namp** and **sp_pwdp** fields of the **struct spwd** structure. Shadow password entries obtained from NIS will contain the value -1 in the remainder of the fields.

When NIS+ is used as the database source, and the caller lacks the permission needed to retrieve the encrypted password from the NIS+ ''passwd.org_dir'' table, the NIS+ service returns the string ''*NP*'' instead of the actual encrypted password string. The functions described on this page will then return the string ''*NP*'' to the caller as the value of the member **sp_pwdp** in the returned shadow password structure.