NAME

hcreate, hdestroy, hsearch, hcreate_r, hdestroy_r, hsearch_r - hash table management

SYNOPSIS

#include <search.h>

int hcreate(size_t *nel***);**

ENTRY *hsearch(ENTRY *item***, ACTION** *action***);**

void hdestroy(void);

#define _GNU_SOURCE /* See feature_test_macros(7) */
#include <search.h>

int hcreate_r(size_t *nel***, struct hsearch_data ****htab***);**

int hsearch_r(ENTRY *item***, ACTION** *action***, ENTRY *****retval***,
 struct hsearch_data ****htab***);**

void hdestroy_r(struct hsearch_data **htab***);**

DESCRIPTION

The three functions **hcreate**(), **hsearch**(), and **hdestroy**() allow the caller to create and manage a hash search table containing entries consisting of a key (a string) and associated data. Using these functions, only one hash table can be used at a time.

The three functions **hcreate_r**(), **hsearch_r**(), **hdestroy_r**() are reentrant versions that allow a program to use more than one hash search table at the same time. The last argument, *htab*, points to a structure that describes the table on which the function is to operate. The programmer should treat this structure as opaque (i.e., do not attempt to directly access or modify the fields in this structure).

First a hash table must be created using **hcreate**(). The argument *nel* specifies the maximum number of entries in the table. (This maximum cannot be changed later, so choose it wisely.) The implementation may adjust this value upward to improve the performance of the resulting hash table.

The **hcreate_r**() function performs the same task as **hcreate**(), but for the table described by the structure **htab*. The structure pointed to by *htab* must be zeroed before the first call to **hcreate_r**().

The function **hdestroy**() frees the memory occupied by the hash table that was created by **hcreate**(). After calling **hdestroy**() a new hash table can be created using **hcreate**(). The **hdestroy_r**() function performs the analogous task for a hash table described by **htab*, which was previously created using **hcreate_r**().

The **hsearch**() function searches the hash table for an item with the same key as *item* (where the same is determined using strcmp(3)), and if successful returns a pointer to it.

The argument *item* is of type *ENTRY*, which is defined in *<search.h>* as follows:

```
typedef struct entry {
char *key;
void *data;
} ENTRY;
```

The field *key* points to a null-terminated string which is the search key. The field *data* points to data that is associated with that key.

The argument *action* determines what **hsearch**() does after an unsuccessful search. This argument must either have the value **ENTER**, meaning insert a copy of *item* (and return a pointer to the new hash table entry as the function result), or the value **FIND**, meaning that NULL should

be returned. (If *action* is **FIND**, then *data* is ignored.)

The **hsearch_r**() function is like **hsearch**() but operates on the hash table described by **htab*.
The **hsearch_r**() function differs from **hsearch**() in that a pointer to the found item is returned
in **retval*, rather than as the function result.

RETURN VALUE

hcreate() and **hcreate_r**() return nonzero on success. They return 0 on error, with *errno* set to
indicate the cause of the error.

On success, **hsearch**() returns a pointer to an entry in the hash table. **hsearch**() returns NULL
on error, that is, if *action* is **ENTER** and the hash table is full, or *action* is **FIND** and *item* can-
not be found in the hash table. **hsearch_r**() returns nonzero on success, and 0 on error. In the
event of an error, these two functions set *errno* to indicate the cause of the error.

ERRORS

hcreate_r() and **hdestroy_r**() can fail for the following reasons:

EINVAL
> *htab* is NULL.

hsearch() and **hsearch_r**() can fail for the following reasons:

ENOMEM
> *action* was **ENTER**, *key* was not found in the table, and there was no room in the table
> to add a new entry.

ESRCH
> *action* was **FIND**, and *key* was not found in the table.

POSIX.1-2001 specifies only the **ENOMEM** error.

ATTRIBUTES

Multithreading (see pthreads(7))

The **hcreate**(), **hsearch**(), and **hdestroy**() functions use a global space for storing the table, so
they are not thread-safe.

The **hcreate_r**(), **hsearch_r**(), and **hdestroy_r**() functions are thread-safe.

CONFORMING TO

The functions **hcreate**(), **hsearch**(), and **hdestroy**() are from SVr4, and are described in
POSIX.1-2001. The functions **hcreate_r**(), **hsearch_r**(), and **hdestroy_r**() are GNU extensions.

NOTES

Hash table implementations are usually more efficient when the table contains enough free space
to minimize collisions. Typically, this means that *nel* should be at least 25% larger than the max-
imum number of elements that the caller expects to store in the table.

The **hdestroy**() and **hdestroy_r**() functions do not free the buffers pointed to by the *key* and
data elements of the hash table entries. (It can't do this because it doesn't know whether these
buffers were allocated dynamically.) If these buffers need to be freed (perhaps because the pro-
gram is repeatedly creating and destroying hash tables, rather than creating a single table whose
lifetime matches that of the program), then the program must maintain bookkeeping data struc-
tures that allow it to free them.

BUGS

SVr4 and POSIX.1-2001 specify that *action* is significant only for unsuccessful searches, so that an
ENTER should not do anything for a successful search. In libc and glibc (before version 2.3),
the implementation violates the specification, updating the *data* for the given *key* in this case.

Individual hash table entries can be added, but not deleted.

EXAMPLE

The following program inserts 24 items into a hash table, then prints some of them.

```
#include <stdio.h>
#include <stdlib.h>
#include <search.h>

static char *data[] = { alpha, bravo, charlie, delta,
echo, foxtrot, golf, hotel, india, juliet,
kilo, lima, mike, november, oscar, papa,
quebec, romeo, sierra, tango, uniform,
victor, whisky, x-ray, yankee, zulu
};

int
main(void)
{
ENTRY e, *ep;
int i;

hcreate(30)

for (i = 0; i < 24; i++) {
e.key = data[i];
/* data is just an integer, instead of a
pointer to something */
e.data = (void *) i;
ep = hsearch(e, ENTER);
/* there should be no failures */
if (ep == NULL) {
fprintf(stderr, entry failedn);
exit(EXIT_FAILURE);
}
}

for (i = 22; i < 26; i++) {
/* print two entries from the table, and
show that two are not in the table */
e.key = data[i];
ep = hsearch(e, FIND);
printf(%9.9s -> %9.9s:%dn, e.key,
ep ? ep->key : NULL, ep ? (int)(ep->data) : 0);
}
hdestroy();
exit(EXIT_SUCCESS);
}
```

SEE ALSO

bsearch(3), lsearch(3), malloc(3), tsearch(3)

COLOPHON

This page is part of release 3.74 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at http://www.kernel.org/doc/man-pages/.