## NAME

**hashinit**, **hashinit_flags**, **hashdestroy**, **phashinit**, **phashinit_flags** - manage kernel hash tables

## SYNOPSIS

**#include <sys/malloc.h>**
**#include <sys/systm.h>**
**#include <sys/queue.h>**

*void **
**hashinit**(*int nelements, struct malloc_type *type, u_long *hashmask*);

*void*
**hashinit_flags**(*int nelements, struct malloc_type *type, u_long *hashmask, int flags*);

*void*
**hashdestroy**(*void *hashtbl, struct malloc_type *type, u_long hashmask*);

*void **
**phashinit**(*int nelements, struct malloc_type *type, u_long *nentries*);

**phashinit_flags**(*int nelements, struct malloc_type *type, u_long *nentries, int flags*);

## DESCRIPTION

The **hashinit**(), **hashinit_flags**(), **phashinit**() and **phashinit_flags**() functions allocate space for hash tables of size given by the argument *nelements*.

The **hashinit**() function allocates hash tables that are sized to largest power of two less than or equal to argument *nelements*. The **phashinit**() function allocates hash tables that are sized to the largest prime number less than or equal to argument *nelements*. The **hashinit_flags**() function operates like **hashinit**() but also accepts an additional argument *flags* which control various options during allocation. **phashinit_flags**() function operates like **phashinit**() but also accepts an additional argument *flags* which control various options during allocation. Allocated hash tables are contiguous arrays of LIST_HEAD(3) entries, allocated using malloc(9), and initialized using LIST_INIT(3). The malloc arena to be used for allocation is pointed to by argument *type*.

The **hashdestroy**() function frees the space occupied by the hash table pointed to by argument *hashtbl*. Argument *type* determines the malloc arena to use when freeing space. The argument *hashmask* should be the bit mask returned by the call to **hashinit**() that allocated the hash table. The argument *flags* must be used with one of the following values.

HASH_NOWAIT Any malloc performed by the **hashinit_flags**() and **phashinit_flags**() function will not be allowed to wait, and therefore may fail.

HASH_WAITOK Any malloc performed by **hashinit_flags**() and **phashinit_flags**() function is allowed to wait for memory. This is also the behavior of **hashinit**() and **phashinit**().

## IMPLEMENTATION NOTES

The largest prime hash value chosen by **phashinit**() is 32749.

## RETURN VALUES

The **hashinit**() function returns a pointer to an allocated hash table and sets the location pointed to by *hashmask* to the bit mask to be used for computing the correct slot in the hash table.

The **phashinit**() function returns a pointer to an allocated hash table and sets the location pointed to by *nentries* to the number of rows in the hash table.

## EXAMPLES

A typical example is shown below:

```
...
static LIST_HEAD(foo, foo) *footable;
static u_long foomask;
...
footable = hashinit(32, M_FOO, &foomask);
```

Here we allocate a hash table with 32 entries from the malloc arena pointed to by M_FOO. The mask for the allocated hash table is returned in *foomask*. A subsequent call to **hashdestroy**() uses the value in *foomask*:

```
...
hashdestroy(footable, M_FOO, foomask);
```

## DIAGNOSTICS

The **hashinit**() and **phashinit**() functions will panic if argument *nelements* is less than or equal to zero.

The **hashdestroy**() function will panic if the hash table pointed to by *hashtbl* is not empty.

## SEE ALSO

LIST_HEAD(3), malloc(9)

## BUGS

There is no **phashdestroy**() function, and using **hashdestroy**() to free a hash table allocated by **phashinit**() usually has grave consequences.