NAME

malloc, free, realloc, calloc, mallopt, mallinfo – memory allocator

SYNOPSIS

cc [*flag* ...] *file* ... **–lmalloc** [*library* ...]

#include <stdlib.h>

void ∗**malloc(size_t** *size***);**

void free(void ∗*ptr***);**

void ∗**realloc(void** ∗*ptr***, size_t** *size***);**

void ∗**calloc(size_t** *nelem***, size_t** *elsize***);**

#include <malloc.h>

int mallopt(int *cmd***, int** *value***);**

struct mallinfo mallinfo(void);

MT-LEVEL

Safe

DESCRIPTION

malloc( ) and **free( )** provide a simple general-purpose memory allocation package.

malloc( ) returns a pointer to a block of at least *size* bytes suitably aligned for any use.

The argument to **free( )** is a pointer to a block previously allocated by **malloc( )**; after **free( )** is performed this space is made available for further allocation, and its contents have been destroyed (but see **mallopt( )** below for a way to change this behavior). If *ptr* is a null pointer, no action occurs.

Undefined results occur if the space assigned by **malloc( )** is overrun or if some random number is handed to **free( ) .**

realloc( ) changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents are unchanged up to the lesser of the new and old sizes. If *ptr* is a null pointer, **realloc( )** behaves like **malloc( )** for the specified size. If *size* is zero and *ptr* is not a null pointer, the object it points to is freed.

calloc( ) allocates space for an array of *nelem* elements of size *elsize*. The space is initialized to zeros.

mallopt( ) provides for control over the allocation algorithm. The available values for *cmd* are:

M_MXFAST
: Set *maxfast* to *value.* The algorithm allocates all blocks below the size of *maxfast* in large groups and then doles them out very quickly. The default value for *maxfast* is 24.

M_NLBLKS
: Set *numlblks* to *value*. The above mentioned ''large groups'' each contain *numlblks* blocks. *numlblks* must be greater than 0. The default value for *numlblks* is 100.

M_GRAIN
: Set *grain* to *value*. The sizes of all blocks smaller than *maxfast* are considered to be rounded up to the nearest multiple of *grain*. *grain* must be greater than 0. The default value of *grain* is the smallest number of bytes that will allow

alignment of any data type. Value will be rounded up to a multiple of the default when *grain* is set.

M_KEEP Preserve data in a freed block until the next **malloc( )**, **realloc( )**, or **calloc( )**. This option is provided only for compatibility with the old version of **malloc( )** and is not recommended.

These values are defined in the **<malloc.h>** header.

mallopt( ) may be called repeatedly, but may not be called after the first small block is allocated.

mallinfo( ) provides instrumentation describing space usage. It returns the **mallinfo** structure with the following members:

```
int  arena;        /* total space in arena */
int  ordblks;      /* number of ordinary blocks */
int  smblks;       /* number of small blocks */
int  hblkhd;       /* space in holding block headers */
int  hblks;        /* number of holding blocks */
int  usmblks;      /* space in small blocks in use */
int  fsmblks;      /* space in free small blocks */
int  uordblks;     /* space in ordinary blocks in use */
int  fordblks;     /* space in free ordinary blocks */
int  keepcost;     /* space penalty if keep option */
                   /* is used */
```

The **mallinfo** structure is defined in the **<malloc.h>** header.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

RETURN VALUES

malloc( ), **realloc( )**, and **calloc( )** return a **NULL** pointer if there is not enough available memory. When **realloc( )** returns **NULL**, the block pointed to by *ptr* is left intact. If **mallopt( )** is called after any allocation or if *cmd* or *value* are invalid, non-zero is returned. Otherwise, it returns zero.

ERRORS

If **malloc( )**, **calloc( )**, or **realloc( )** returns unsuccessfully, **errno** will be set to indicate the following:

ENOMEM *size* bytes of memory exceeds the physical limits of your system, and cannot be allocated.

EAGAIN There is not enough memory available AT THIS POINT IN TIME to allocate *size* bytes of memory; but the application could try again later.

SEE ALSO

brk(2), **malloc**(3C), **bsdmalloc**(3X)

NOTES

Note that unlike **malloc**(3C), this package does not preserve the contents of a block when it is freed, unless the **M_KEEP** option of **mallopt( )** is used.

Undocumented features of **malloc**(3C) have not been duplicated.

Function prototypes for **malloc( )**, **realloc( )**, **calloc( )**, and **free( )** are also defined in the **<malloc.h>** header for compatibility with old applications. New applications should include **<stdlib.h>** to access the prototypes for these functions.
Comparative Features of **malloc**(3X), **bsdmalloc**(3X), and **malloc**(3C)**:**

- These **malloc**(3X) routines are space-efficient, but have slower performance.
- The **bsdmalloc**(3X) routines afford better performance, but are space-inefficient.
- The standard, fully SCD-compliant **malloc**(3C) routines are a trade-off between performance and space-efficiency.

free( ) does not set **errno.**