NAME

malloc, calloc, free, memalign, realloc, valloc, alloca – memory allocator

SYNOPSIS

#include <stdlib.h>

void *malloc(size_t *size***);**

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

void free(void **ptr***);**

void *memalign(size_t *alignment***, size_t** *size***);**

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

void *valloc(size_t *size***);**

#include <alloca.h>
void *alloca(size_t *size***);**

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( )** , **calloc( )** or **realloc( )**. After **free( )** is performed this space is made available for further allocation. If *ptr* is a **NULL** pointer, no action occurs.

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

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

memalign( ) allocates *size* bytes on a specified alignment boundary, and returns a pointer to the allocated block. The value of the returned address is guaranteed to be an even multiple of *alignment*. Note: the value of *alignment* must be a power of two, and must be greater than or equal to the size of a word.

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 will be unchanged up to the lesser of the new and old sizes. If *ptr* is **NULL**, **realloc( )** behaves like **malloc( )** for the specified size. If *size* is zero and *ptr* is not a null pointer, the object pointed to is freed.

valloc( ) is equivalent to **memalign(sysconf(_SC_PAGESIZE),size)**.

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

malloc( ), **realloc( )**, **memalign( )**, and **valloc( )** will fail if there is not enough available memory.

alloca( ) allocates *size* bytes of space in the stack frame of the caller, and returns a pointer to the allocated block. This temporary space is automatically freed when the caller returns. Note: if the allocated block is beyond the current stack limit, the resulting behavior is undefined.

RETURN VALUES

If there is no available memory, **malloc( )**, **realloc( )**, **memalign( )**, **valloc( )**, and **calloc( )** return a null pointer. When **realloc( )** returns **NULL**, the block pointed to by *ptr* is left intact. If *size*, *nelem*, or *elsize* is 0, a unique pointer to the arena is returned.

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), **getrlimit**(2), **bsdmalloc**(3X), **malloc**(3X), **mapmalloc**(3X)

WARNINGS

Undefined results will occur if the size requested for a block of memory exceeds the maximum size of a process's heap, which may be obtained with **getrlimit( )**.

alloca( ) is machine-, compiler-, and most of all, system-dependent. Its use is strongly discouraged.

NOTES

Comparative Features of **malloc**(3C), **bsdmalloc**(3X), and **malloc**(3X)**:**

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

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