NAME

calloc, malloc, free, realloc − Allocate and free dynamic memory

SYNOPSIS

#include <stdlib.h>

void *calloc(size_t *nmemb*, **size_t** *size***);**
void *malloc(size_t *size***);**
void free(void **ptr***);**
void *realloc(void **ptr*, **size_t** *size***);**

DESCRIPTION

calloc() allocates memory for an array of *nmemb* elements of *size* bytes each and returns a pointer to the allocated memory. The memory is set to zero.

malloc() allocates *size* bytes and returns a pointer to the allocated memory. The memory is not cleared.

free() frees the memory space pointed to by *ptr*, which must have been returned by a previous call to **malloc()**, **calloc()** or **realloc()**. Otherwise, or if **free(***ptr***)** has already been called before, undefined behaviour occurs. If *ptr* is **NULL**, no operation is performed.

realloc() changes the size of the memory block pointed to by *ptr* to *size* bytes. The contents will be unchanged to the minimum of the old and new sizes; newly allocated memory will be uninitialized. If *ptr* is **NULL**, the call is equivalent to **malloc(size)**; if size is equal to zero, the call is equivalent to **free(***ptr***)**. Unless *ptr* is **NULL**, it must have been returned by an earlier call to **malloc()**, **calloc()** or **realloc()**.

RETURN VALUE

For **calloc()** and **malloc()**, the value returned is a pointer to the allocated memory, which is suitably aligned for any kind of variable, or **NULL** if the request fails.

free() returns no value.

realloc() returns a pointer to the newly allocated memory, which is suitably aligned for any kind of variable and may be different from *ptr*, or **NULL** if the request fails or if size was equal to 0. If **realloc()** fails the original block is left untouched − it is not freed or moved.

CONFORMING TO

ANSI−C

SEE ALSO

brk(2)

NOTES

The Unix98 standard requires **malloc()**, **calloc()**, and **realloc**() to set *errno* to ENOMEM upon failure. Glibc assumes that this is done (and the glibc versions of these routines do this); if you use a private malloc implementation that does not set *errno*, then certain library routines may fail without having a reason in *errno*.

cated chunk or freeing the same pointer twice.

Recent versions of Linux libc (later than 5.4.23) and GNU libc (2.x) include a malloc implementation which is tunable via environment variables. When **MALLOC_CHECK_** is set, a special (less efficient) implementation is used which is designed to be tolerant against simple errors, such as double calls of **free()** with the same argument, or overruns of a single byte (off-by-one bugs). Not all such errors can be proteced against, however, and memory leaks can result. If **MALLOC_CHECK_** is set to 0, any detected heap corruption is silently ignored; if set to 1, a diagnostic is printed on stderr; if set to 2, **abort()** is called immediately. This can be useful because otherwise a crash may happen much later, and the true cause for the problem is then very hard to track down.