NAME

GC_malloc, GC_malloc_atomic, GC_free, GC_realloc, GC_enable_incremental, GC_register_finalizer, GC_malloc_ignore_off_page, GC_malloc_atomic_ignore_off_page, GC_set_warn_proc - Garbage collecting malloc replacement

SYNOPSIS

```
#include <gc/gc.h>
void * GC_malloc(size_t size);
void * GC_malloc_atomic(size_t size);
void GC_free(void *ptr);
void * GC_realloc(void *ptr, size_t size);
void GC_enable_incremental(void);
void * GC_malloc_ignore_off_page(size_t size);
void * GC_malloc_atomic_ignore_off_page(size_t size);
void GC_set_warn_proc(void (*proc)(char *, GC_word));

cc ... -lgc
```

DESCRIPTION

GC_malloc and *GC_free* are plug-in replacements for standard malloc and free. However, *GC_malloc* will attempt to reclaim inaccessible space automatically by invoking a conservative garbage collector at appropriate points. The collector traverses all data structures accessible by following pointers from the machines registers, stack(s), data, and bss segments. Inaccessible structures will be reclaimed. A machine word is considered to be a valid pointer if it is an address inside an object allocated by *GC_malloc* or friends.

In most cases it is preferable to call the macros GC_MALLOC, GC_FREE, etc. instead of calling GC_malloc and friends directly. This allows debugging versions of the routines to be substituted by defining GC_DEBUG before including gc.h.

See the documentation in the include files gc_cpp.h and gc_allocator.h, as well as the gcinterface.md file in the distribution, for an alternate, C++ specific interface to the garbage collector. Note that C++ programs generally need to be careful to ensure that all allocated memory (whether via new, malloc, or STL allocators) that may point to garbage collected memory is either itself garbage collected, or at least traced by the collector.

Unlike the standard implementations of malloc, *GC_malloc* clears the newly allocated storage. *GC_malloc_atomic* does not. Furthermore, it informs the collector that the resulting object will never contain any pointers, and should therefore not be scanned by the collector.

GC_free can be used to deallocate objects, but its use is optional, and generally discouraged. *GC_realloc* has the standard realloc semantics. It preserves pointer-free-ness. *GC_register_finalizer* allows for registration of functions that are invoked when an object becomes inaccessible.

The garbage collector tries to avoid allocating memory at locations that already appear to be referenced before allocation. (Such apparent ‘‘pointers’’ are usually large integers and the like that just happen to look like an address.) This may make it hard to allocate very large objects. An attempt to do so may generate a warning.

GC_malloc_ignore_off_page and *GC_malloc_atomic_ignore_off_page* inform the collector that the client code will always maintain a pointer to near the beginning (i.e. within the first heap block) of the object, and that pointers beyond that can be ignored by the collector. This makes it much easier for the collector to place large objects. These are recommended for large object allocation. (Objects expected to be > ~100 KB should be allocated this way.)

It is also possible to use the collector to find storage leaks in programs destined to be run with standard malloc/free. The collector can be compiled for thread-safe operation. Unlike standard malloc, it is safe to call malloc after a previous malloc call was interrupted by a signal, provided the original malloc call is not resumed.

The collector may, on rare occasion, produce warning messages. On UNIX machines these appear on stderr. Warning messages can be filtered, redirected, or ignored with *GC_set_warn_proc* This is recommended for production code. See gc.h for details.

Fully portable code should call *GC_INIT* from the primordial thread of the main program before making any other GC calls. On most platforms this does nothing and the collector is initialized on first use. On a few platforms explicit initialization is necessary. And it can never hurt.

Debugging versions of many of the above routines are provided as macros. Their names are identical to the above, but consist of all capital letters. If GC_DEBUG is defined before gc.h is included, these routines do additional checking, and allow the leak detecting version of the collector to produce slightly more useful output. Without GC_DEBUG defined, they behave exactly like the lower-case versions.

On some machines, collection will be performed incrementally after a call to *GC_enable_incremental.* This may temporarily write protect pages in the heap. See the README file for more information on how this interacts with system calls that write to the heap.

Other facilities not discussed here include limited facilities to support incremental collection on machines without appropriate VM support, provisions for providing more explicit object layout information to the garbage collector, more direct support for ‘‘weak’’ pointers, support for ‘‘abortable’’ garbage collections during idle time, etc.

PORT INFORMATION

In this (FreeBSD package) installation, *gc.h* and *gc_cpp.h* will probably be found in */usr/local/include*, and the library in */usr/local/lib.*

This library has been compiled as drop-in replacements for malloc and free (which is to say, all malloc calls will allocate garbage-collectable data). There is no need to include "gc.h" in your C files unless you want access to the debugging (and other) functions defined there, or unless you want to explicitly use *GC_malloc_uncollectable* for some allocations. Just link against them whenever you want either garbage collection or leak detection.

The C++ header file, "gc_cpp.h", *is* necessary for C++ programs, to obtain the appropriate definitions of the *new* and *delete* operators. The comments in both of these header files presently provide far better documentation for the package than this man page; look there for more information.

This library is compiled without (explicit) support for the experimental *gc* extension of *g++*. This may or may not make a difference.

SEE ALSO

The README and gc.h files in the distribution. More detailed definitions of the functions exported by the collector are given there. (The above list is not complete.)

The web site at http://www.hboehm.info/gc/ (or https://github.com/ivmai/bdwgc/).

Boehm, H., and M. Weiser, "Garbage Collection in an Uncooperative Environment", "Software Practice & Experience", September 1988, pp. 807-820.

The malloc(3) man page.

AUTHOR

Hans-J. Boehm (boehm@acm.org). Some of the code was written by others (see the AUTHORS file for the details), most notably by Alan Demers, and, recently, Ivan Maidanski.