NAME

__malloc_hook, __malloc_initialize_hook, __memalign_hook, __free_hook, __realloc_hook, __after_morecore_hook − malloc debugging variables (DEPRECATED)

LIBRARY

Standard C library (*libc*, *−lc*)

SYNOPSIS

```
#include <malloc.h>

typeof(void *(size_t size, const void *caller))
  *volatile __malloc_hook;

typeof(void *(void *p, size_t size, const void *caller))
  *volatile __realloc_hook;

typeof(void *(size_t align, size_t size, const void *caller))
  *volatile __memalign_hook;

typeof(void *(void *p, const void *caller))
  *volatile __free_hook;

typeof(void (void))       *__malloc_initialize_hook;
typeof(void (void)) *volatile __after_mrecore_hook;
```

DESCRIPTION

The GNU C library lets you modify the behavior of **malloc**(3), **realloc**(3), and **free**(3) by specifying appropriate hook functions. You can use these hooks to help you debug programs that use dynamic memory allocation, for example.

The variable **__malloc_initialize_hook** points at a function that is called once when the malloc implementation is initialized. This is a weak variable, so it can be overridden in the application with a definition like the following:

```
typeof(void (void))  *__malloc_initialize_hook = my_init_hook;
```

Now the function *my_init_hook*() can do the initialization of all hooks.

The four functions pointed to by **__malloc_hook**, **__realloc_hook**, **__memalign_hook**, **__free_hook** have a prototype like the functions **malloc**(3), **realloc**(3), **memalign**(3), **free**(3), respectively, except that they have a final argument *caller* that gives the address of the caller of **malloc**(3), etc.

The variable **__after_morecore_hook** points at a function that is called each time after **sbrk**(2) was asked for more memory.

STANDARDS

GNU.

NOTES

The use of these hook functions is not safe in multithreaded programs, and they are now deprecated. From glibc 2.24 onwards, the **__malloc_initialize_hook** variable has been removed from the API, and from glibc 2.34 onwards, all the hook variables have been removed from the API. Programmers should instead preempt calls to the relevant functions by defining and exporting **malloc**(), **free**(), **realloc**(), and **calloc**().

EXAMPLES

Here is a short example of how to use these variables.

```
#include <stdio.h>
#include <malloc.h>

/* Prototypes for our hooks */
static void my_init_hook(void);
static void *my_malloc_hook(size_t, const void *);

/* Variables to save original hooks */
```

```
static typeof(void *(size_t, const void *))  *old_malloc_hook;

/* Override initializing hook from the C library */
typeof(void (void))  *__malloc_initialize_hook = my_init_hook;

static void
my_init_hook(void)
{
    old_malloc_hook = __malloc_hook;
    __malloc_hook = my_malloc_hook;
}

static void *
my_malloc_hook(size_t size, const void *caller)
{
    void *result;

    /* Restore all old hooks */
    __malloc_hook = old_malloc_hook;

    /* Call recursively */
    result = malloc(size);

    /* Save underlying hooks */
    old_malloc_hook = __malloc_hook;

    /* printf() might call malloc(), so protect it too */
    printf("malloc(%zu) called from %p returns %p\n",
            size, caller, result);

    /* Restore our own hooks */
    __malloc_hook = my_malloc_hook;

    return result;
}
```

SEE ALSO

mallinfo(3), **malloc**(3), **mcheck**(3), **mtrace**(3)