NAME

__malloc_hook, __malloc_initialize_hook, __memalign_hook, __free_hook, __realloc_hook, __after_morecore_hook – malloc debugging variables

SYNOPSIS

#include <malloc.h>

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

void *(*__realloc_hook)(void * *ptr***, size_t** *size***, const void *** *caller***);**

void *(*__memalign_hook)(size_t *alignment***, size_t** *size***,**
const void * *caller***);**

void (*__free_hook)(void * *ptr***, const void *** *caller***);**

void (*__malloc_initialize_hook)(void);

void (*__after_morecore_hook)(void);

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:

```
void (*__malloc_initialize_hook)(void) = 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.

CONFORMING TO

These functions are GNU extensions.

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. Programmers should instead preempt calls to the relevant functions by defining and exporting functions such as "malloc" and "free".

EXAMPLE

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 void *(*old_malloc_hook)(size_t, const void *);

/* Override initializing hook from the C library. */
void (*__malloc_initialize_hook) (void) = 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(%u) called from %p returns %p\n",
            (unsigned int) size, caller, result);

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

    return result;
}
```

SEE ALSO

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

COLOPHON

This page is part of release 5.02 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at https://www.kernel.org/doc/man−pages/.