## NAME

libmpalloc − dynamic memory allocation replacement library

## SYNOPSIS

```
#include <mpalloc.h>

void *MP_MALLOC(void *ptr, size_t count, typename type);
void *MP_CALLOC(void *ptr, size_t count, typename type);
char *MP_STRDUP(char *ptr, const char *str);
void *MP_REALLOC(void *ptr, size_t count, typename type);
void MP_FREE(void *ptr);
__mp_failhandler MP_FAILURE(__mp_failhandler func);
```

## DESCRIPTION

The *mpalloc library* contains release implementations of all of the mpatrol library functions, with all of its checking, debugging and tracing features disabled. It is fully link-compatible with the mpatrol library and so can be linked in instead of the mpatrol library in order to quickly disable all of its features without requiring a complete recompilation of all of the source files in a project. It also contains implementations of the MP_MALLOC family of functions that can be used in a release environment.

All of the function definitions in *mpatrol.h* can be disabled by defining the NDEBUG preprocessor macro, which is the same macro used to control the behaviour of the assert function. If NDEBUG is defined then no macro redefinition of functions will take place and all special mpatrol library functions will evaluate to empty statements. The *mpalloc.h* header file will also be included in this case. It is intended that the NDEBUG preprocessor macro be defined in release builds.

The mpalloc library contains functional replacements for all of the mpatrol library's dynamic memory allocation and memory operation functions, mainly for use in situations where not all of the source files in a project have been recompiled with the NDEBUG preprocessor macro in order to remove mpatrol. However, not all of these functions can be fully implemented using ANSI C and so may contain some limitations. The only recommended solution for a final release is to perform a complete recompile with NDEBUG defined.

## FUNCTIONS

The following 6 functions are provided as convenient alternatives to the ANSI C dynamic memory allocation functions (although strdup is not strictly an ANSI C function). They are implemented as preprocessor macro functions which may evaluate their arguments more than once, so extra care should be taken to avoid passing arguments with side-effects. None of the functions return NULL if no memory is available and instead abort the program with a useful error message indicating where the call to allocate memory came from and what was being allocated. To use these you should include the *mpalloc.h* header file:

### MP_MALLOC

Allocates *count* uninitialised items of type *type* from the heap, sets *ptr* to the result and returns a suitably-cast pointer to the first item of the allocation. The pointer returned will be suitably aligned for holding items of type *type*. If *count* is *0* then it will be implicitly rounded up to *1*. If there is not enough space in the heap then the program will be aborted after calling the allocation failure handler, which by default writes an appropriate error message to the standard error file stream. The allocated memory in *ptr* must be deallocated with MP_FREE or reallocated with MP_REALLOC.

### MP_CALLOC

Allocates *count* zero-initialised items of type *type* from the heap, sets *ptr* to the result and returns a suitably-cast pointer to the first item of the allocation. The pointer returned will be suitably aligned for holding items of type *type*. If *count* is *0* then it will be implicitly rounded up to *1*. If there is not enough space in the heap then the program will be aborted after calling the allocation failure handler, which by default writes an appropriate error message to the standard error file stream. The allocated memory in *ptr* must be deallocated with MP_FREE or reallocated with MP_REALLOC.

MP_STRDUP

Allocates exactly enough memory from the heap to duplicate *str* (including the terminating nul character), sets *ptr* to the result and returns a suitably-cast pointer to the first byte of the allocation after copying *str* to the newly-allocated memory. The pointer returned will have no alignment constraints and can be used to store character data up to the length of *str*. If there is not enough space in the heap then the program will be aborted after calling the allocation failure handler, which by default writes an appropriate error message to the standard error file stream. The allocated memory in *ptr* must be deallocated with MP_FREE or reallocated with MP_REALLOC.

MP_REALLOC

Resizes the memory allocation beginning at *ptr* to *count* items of type *type* and returns a suitably-cast pointer to the first item of the new allocation after copying *ptr* to the newly-allocated memory, which will be truncated if *count* is smaller than the original number of items. The pointer returned will be suitably aligned for holding items of type *type*. If *ptr* is NULL then the call will be equivalent to MP_MALLOC. If *count* is *0* then it will be implicitly rounded up to *1*. If *count* is greater than the original number of items then the extra space will be filled with uninitialised bytes. If there is not enough space in the heap then the program will be aborted after calling the allocation failure handler, which by default writes an appropriate error message to the standard error file stream. The allocated memory must be deallocated with MP_FREE and can be reallocated again with MP_REALLOC.

MP_FREE

Frees the memory allocation beginning at *ptr* so the memory can be reused by another call to allocate memory, and sets *ptr* to NULL after freeing the memory. If *ptr* is NULL then no memory will be freed.

MP_FAILURE

Installs an allocation failure handler specifically for use with MP_MALLOC, MP_CALLOC, MP_STRDUP and MP_REALLOC and returns a pointer to the previously installed handler, normally the default handler if no handler had been previously installed. This will be called by the above functions when there is not enough space in the heap for them to satisfy their allocation request. The default allocation failure handler will terminate the program after writing an error message to the standard error file stream indicating where the original allocation request took place and what was being allocated.

## SEE ALSO

mpatrol(1), mprof(1), mptrace(1), mleak(1), mpsym(1), mpedit(1), hexwords(1), libmpatrol(3), malloc(3), assert(3).

The mpatrol manual and reference card.

http://www.cbmamiga.demon.co.uk/mpatrol/

## AUTHOR

Graeme S. Roy <graeme.roy@analog.com>

## COPYRIGHT