liboom

Generated by Doxygen 1.9.4

1 File Index 1

1.1 File List . . . 1

2 File Documentation 3

2.1 oomfill.h File Reference . . . 3

2.1.1 Detailed Description . . . 4

2.1.2 Enumeration Type Documentation . . . 4

2.1.2.1 bool . . . 4

2.1.3 Function Documentation . . . 5

2.1.3.1 oomfill_config() . . . 5

2.1.3.2 oomfill_enabled() . . . 6

2.1.4 Variable Documentation . . . 6

2.1.4.1 oomfill_disable . . . 6

2.1.4.2 oomfill_enable . . . 6

2.1.4.3 oomfill_fill . . . 7

2.1.4.4 oomfill_free . . . 8

2.2 oomfill.h . . . 8

2.3 oomstub.h File Reference . . . 9

2.3.1 Detailed Description . . . 9

2.3.2 Function Documentation . . . 9

2.3.2.1 oomstub_getcountdown() . . . 10

2.3.2.2 oomstub_setcountdown() . . . 10

2.4 oomstub.h . . . 10

2.5 oomfill.c File Reference . . . 11

2.5.1 Detailed Description . . . 12

2.5.2 Macro Definition Documentation . . . 12

2.5.2.1 RAMBLOCKS_MAX . . . 12

2.5.3 Function Documentation . . . 12

2.5.3.1 oomfill_config() . . . 12

2.5.3.2 oomfill_enabled() . . . 13

2.5.4 Variable Documentation . . . 13

2.5.4.1 oomfill_disable . . . 13

2.5.4.2 oomfill_enable . . . 14

2.5.4.3 oomfill_fill . . . 14

2.5.4.4 oomfill_free . . . 15

2.6 oomfill.c . . . 15

2.7 oomstub.c File Reference . . . 19

2.7.1 Detailed Description . . . 20

2.7.2 Macro Definition Documentation . . . 20

2.7.2.1 RTLD_NEXT . . . 20

2.7.3 Function Documentation . . . 20

2.7.3.1 calloc() . . . 20

2.7.3.2 malloc() 21
2.7.3.3 oomstub_getcountdown() 21
2.7.3.4 oomstub_setcountdown() 22
2.7.3.5 realloc() 22
2.8 oomstub.c 23
2.9 revision.h File Reference 24
2.9.1 Macro Definition Documentation 24
2.9.1.1 REVISION 24
2.10 revision.h 24

Index **25**

Chapter 1

File Index

1.1 File List

Here is a list of all files with brief descriptions:

oomfill.h
: Library safely (rlimited) consumming RAM to trigger OOMs (only on POSIX systems) 3

oomstub.h
: C memory management diverters 9

oomfill.c
: Library safely (rlimited) consumming RAM to trigger OOMs (only on POSIX systems) 11

oomstub.c
: C memory management stubs 19

revision.h 24

Chapter 2

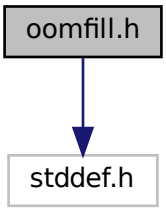
File Documentation

2.1 oomfill.h File Reference

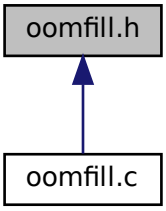
Library safely (rlimited) consumming RAM to trigger OOMs (only on POSIX systems)

```
#include <stddef.h>
```

Include dependency graph for oomfill.h:



This graph shows which files directly or indirectly include this file:



Enumerations

- enum bool { false = (0==1) , true = (1==1) }

Functions

- size_t oomfill_config (const size_t hardlimit)

  Sets the oomfill helpers hard rlimit and enables the oomfill helper features.
- bool oomfill_enabled ()

Variables

- size_t(∗ oomfill_enable)(const size_t softlimit)

  Starts a new oomfill environment or reconfigure the soft limit.
- size_t(∗ oomfill_disable)()

  Stops the current oomfill.
- size_t(∗ oomfill_fill)(const size_t minHeap, const size_t minStack)

  Starts an almost OOM single and simple test.
- void(∗ oomfill_free)()

  Ends a single simple OOM test.

2.1.1 Detailed Description

Library safely (rlimited) consumming RAM to trigger OOMs (only on POSIX systems)

```
void myfunction() {
    void* ptr = NULL;
    // Keep at most 100 bytes available in heap, 4 in stack
    oomfill_fill(100,4);
    ptr=malloc(2000);
    oomfill_free();
    // Actual test outside of the constrained code to avoid failure
    if (NULL == ptr)
        fprintf(stderr,"malloc failed\n");
    else
        free(ptr);
}
size_t ramlimit;
ramlimit=64*1024*1024;
printf ("Limit memory usage to %uMB : ",ramlimit/1024/1024);
ramlimit = oomfill_config(ramlimit);
printf ("%uB (%uMB)\n",ramlimit,ramlimit/1024/1024);
// not constrained (only by the 64MB ramlimit)
myfunction();
// OOM activation to allow memory eating with a safeguard
oomfill_enable(16*1024*1024);
myfunction();
oomfill_disable();
```

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file oomfill.h.

2.1.2 Enumeration Type Documentation

2.1.2.1 bool

```
enum bool
```

Enumerator

false	
true	

Definition at line 58 of file oomfill.h.

2.1.3 Function Documentation

2.1.3.1 oomfill_config()

```
size_t oomfill_config (
            const size_t hardlimit )
```

Sets the oomfill helpers hard rlimit and enables the oomfill helper features.

This function needs to be invoqued BEFORE any other helper from the framework. It will configure an hard RAM limit for the current process and each of his children. Then, it will enable the other oomfill helper functions.

It refuses to set a limit over the actually installed physical RAM to limit pushing RAM pages to the swap. If the requested value is zero, it will default to the actually installed physical RAM.

If anything goes wrong or did not behaves as expected, it abort the process to avoid RAM bombing and swapping.

Despite it was designed to be invoked only once, from the main parent process, it can be invoked several times as long as the hardlimit parameter is always less than the previous call, otherwise it will fail and abort the process.

Parameters

in	*hardlimit*	the size in bytes to limit the process to.

Returns

Returns the actual configured size. It should be the same value as the hardlimit parameter, otherwise something went wrong, it was not detected and the process was not aborted (which is a bug to report)

See also

oomfill_enable

oomfill_disable

oomfill_fill

oomfill_free

Definition at line 457 of file oomfill.c.

2.1.3.2 oomfill_enabled()

```
bool oomfill_enabled ( )
```

Definition at line 511 of file oomfill.c.

2.1.4 Variable Documentation

2.1.4.1 oomfill_disable

```
size_t(* oomfill_disable) () ( )  [extern]
```

Stops the current oomfill.

This function can be invoked any time after the oomfill_config initialized the environment. If invocated before, it will abort the current process to help detecting mistakes in the test code.

This function disables the fill/free oomfill functions and free the allocated RAM before reverting the soft limit to the hard limit value.

It is designed to be invoked t the end of a test case.

Despite it could be invoked twice or without a prior call to oomfill_enable, it is illegal and not allowed to help detecting mistakes in the test code.

Returns

Returns the applied soft limit, which should be the same as the hard limit.

See also

oomfill_enable

oomfill_disable

oomfill_fill

oomfill_free

Definition at line 210 of file oomfill.c.

2.1.4.2 oomfill_enable

```
size_t(* oomfill_enable) (const size_t softlimit) (
            const size_t softlimit )  [extern]
```

Starts a new oomfill environment or reconfigure the soft limit.

This function can only be invoked after at least a first call to oomfill_config to initialize the oomfill helpers environment. If invocated before, it will abort the current process to help detecting mistakes in the test code.

It will (soft)limit the current process and his children to the provided value in bytes. Then, it will enable the fill/free oomfill helper functions which are disabled otherwise.

It is designed to be invoked at the begining of a test case to apply for all tests in this testcase.

In case of any unexpected behavior, it should abort the current process.

Parameters

in	*softlimit*	Soft limit to set in bytes. It has to be less than the hardlimit otherwise it will fail and abort. If set to 0, it will apply the same value as the hardlimit.

Returns

Returns the actually set value as a softlimit, either the requested value or the hardlimit in case of 0 requested.

See also

oomfill_enable

oomfill_disable

oomfill_fill

oomfill_free

Definition at line 171 of file oomfill.c.

2.1.4.3 oomfill_fill

```
size_t(* oomfill_fill) (const size_t minHeap, const size_t minStack) (
            const size_t minHeap,
            const size_t minStack )  [extern]
```

Starts an almost OOM single and simple test.

This function is only enabled after a call to oomfill_enable and will have no effect otherwise. Its goal is to completely fill the RAM until the very last bytes, to keep only between minHeap and maxHeap bytes available in the heap and minStack bytes in the stack.

If oomfill_enable was not invoked beforehand, this function will simply return without any RAM consumption.

It should be called immediately before the test and should be reverted with oomfill_free immediately after. It can create so much pressure on the available memory that a simple printf could fail.

It is designed to fail if invoked twice as this is very probably a mistake in the test code. Should you need to change the RAM filling values, first call oomfill_free and reapply oomfill_fill. This ensure a really wanted behavior and not a mistake in your test code.

Parameters

in	*minHeap*	
in	*minStack*	

Returns

Returns the allocated size to fill the memory

See also

oomfill_enable

oomfill_disable

oomfill_fill

oomfill_free

Definition at line 96 of file oomfill.c.

2.1.4.4 oomfill_free

```
void(* oomfill_free) () ( )  [extern]
```

Ends a single simple OOM test.

If this function is invoked between an oomfill_enable and an oomfill_disable invocations, it deallocates the RAM allocated by the oomfill_fill function. It should be called immediately after the single and simple test because the RAM pressure can even make a printf to fail.

If called without a prior oomfill_enable invocation, this function simply returns without any action.

This function is designed to fail and abort the process if invocated whereas there is no current RAM allocated, when called twice, for example. This helps to avoid mistakes in the test scenario.

Definition at line 125 of file oomfill.c.

2.2 oomfill.h

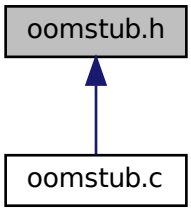
Go to the documentation of this file.

```
00001
00049 #ifndef __OOMFILL_H__
00050 #define __OOMFILL_H__
00051
00052 #ifdef HAVE_CONFIG_H
00053 #include "config.h"
00054 #endif
00055
00056 #include <stddef.h>                              /* size_t */
00057
00058 typedef enum { false = (0==1), true = (1==1) } bool;
00059
00060 size_t oomfill_config(const size_t hardlimit);
00061 extern size_t (*oomfill_enable)(const size_t softlimit);
00062 extern size_t (*oomfill_disable)();
00063 extern size_t (*oomfill_fill)(const size_t minHeap, const size_t minStack);
00064 extern void (*oomfill_free)();
00065 extern bool oomfill_enabled();
00066
00067 #endif  /*__OOMFILL_H__ */
00068
00069 /* vim:  set tw=80:  */
```

2.3 oomstub.h File Reference

C memory management diverters.

This graph shows which files directly or indirectly include this file:



Functions

- void oomstub_setcountdown (const int counter)

  Sets the malloc countdown before triggering a failure.
- int oomstub_getcountdown ()

  Gets the current malloc countdown before triggering a failure.

2.3.1 Detailed Description

C memory management diverters.

This file implements fake malloc, free and realloc functions which can fail and simulate OOM after a countdown. This library is not threadsafe. Inspired from https://github.com/bkthomps/Containers test suite

```
oomstub_setcountdown(3);
assert(NULL!=malloc(123));          // Success
assert(NULL!=realloc(oldptr,123)); // Success
assert(NULL==malloc(123));          // Failure with NULL
assert(NULL!=malloc(123));          // Success
assert(NULL!=malloc(123));          // Success
```

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file oomstub.h.

2.3.2 Function Documentation

2.3.2.1 oomstub_getcountdown()

```
int oomstub_getcountdown ( )
```

Gets the current malloc countdown before triggering a failure.

Returns

The current internal countdown value

See also

malloc

calloc

realloc

Definition at line 72 of file oomstub.c.

2.3.2.2 oomstub_setcountdown()

```
void oomstub_setcountdown (
            const int counter )
```

Sets the malloc countdown before triggering a failure.

Parameters

counter	Value to set in the internal countdown

See also

malloc

calloc

realloc

Definition at line 62 of file oomstub.c.

2.4 oomstub.h

Go to the documentation of this file.

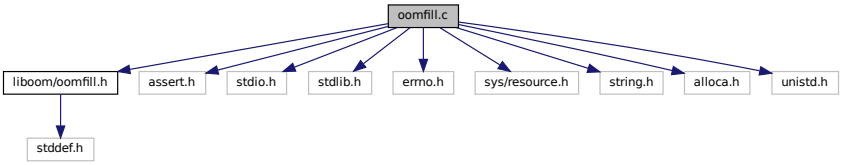
```
00001
00033 #ifndef __OOMSTUB_H__
00034 #define __OOMSTUB_H__
00035
00036 #ifdef HAVE_CONFIG_H
00037 #include "config.h"
00038 #endif
00039
00040 void oomstub_setcountdown(const int counter);
00041 int  oomstub_getcountdown();
00042
00043 #endif /* __OOMSTUB_H__ */
00044
00045 /* vim:  set tw=80:  */
```

2.5 oomfill.c File Reference

Library safely (rlimited) consumming RAM to trigger OOMs (only on POSIX systems)

```
#include "liboom/oomfill.h"
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/resource.h>
#include <string.h>
#include <alloca.h>
#include <unistd.h>
```

Include dependency graph for oomfill.c:



Macros

- #define RAMBLOCKS_MAX 1000

  Maximum number of fragmented blocks to allocate.

Functions

- size_t oomfill_config (const size_t hardlimit)

  Sets the oomfill helpers hard rlimit and enables the oomfill helper features.
- bool oomfill_enabled ()

Variables

- size_t(* oomfill_fill)(const size_t minHeap, const size_t minStack) =oomfill_fill_preinit

  Starts an almost OOM single and simple test.
- void(* oomfill_free)() =oomfill_free_preinit

  Ends a single simple OOM test.
- size_t(* oomfill_enable)(const size_t softlimit) =oomfill_enable_preinit

  Starts a new oomfill environment or reconfigure the soft limit.
- size_t(* oomfill_disable)() =oomfill_disable_preinit

  Stops the current oomfill.

2.5.1 Detailed Description

Library safely (rlimited) consumming RAM to trigger OOMs (only on POSIX systems)

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file oomfill.c.

2.5.2 Macro Definition Documentation

2.5.2.1 RAMBLOCKS_MAX

```
#define RAMBLOCKS_MAX 1000
```

Maximum number of fragmented blocks to allocate.

I never reached more than 350 on my systems. The value needs to be less than UINT_MAX

Definition at line 33 of file oomfill.c.

2.5.3 Function Documentation

2.5.3.1 oomfill_config()

```
size_t oomfill_config (
            const size_t hardlimit )
```

Sets the oomfill helpers hard rlimit and enables the oomfill helper features.

This function needs to be invoqued BEFORE any other helper from the framework. It will configure an hard RAM limit for the current process and each of his children. Then, it will enable the other oomfill helper functions.

It refuses to set a limit over the actually installed physical RAM to limit pushing RAM pages to the swap. If the requested value is zero, it will default to the actually installed physical RAM.

If anything goes wrong or did not behaves as expected, it abort the process to avoid RAM bombing and swapping.

Despite it was designed to be invoked only once, from the main parent process, it can be invoked several times as long as the hardlimit parameter is always less than the previous call, otherwise it will fail and abort the process.

Parameters

in	*hardlimit*	the size in bytes to limit the process to.

Returns

Returns the actual configured size. It should be the same value as the hardlimit parameter, otherwise something went wrong, it was not detected and the process was not aborted (which is a bug to report)

See also

oomfill_enable
oomfill_disable
oomfill_fill
oomfill_free

Definition at line 457 of file oomfill.c.

2.5.3.2 oomfill_enabled()

```
bool oomfill_enabled ( )
```

Definition at line 511 of file oomfill.c.

2.5.4 Variable Documentation

2.5.4.1 oomfill_disable

```
size_t(* oomfill_disable) () ( ) =oomfill_disable_preinit
```

Stops the current oomfill.

This function can be invoked any time after the oomfill_config initialized the environment. If invocated before, it will abort the current process to help detecting mistakes in the test code.

This function disables the fill/free oomfill functions and free the allocated RAM before reverting the soft limit to the hard limit value.

It is designed to be invoked t the end of a test case.

Despite it could be invoked twice or without a prior call to oomfill_enable, it is illegal and not allowed to help detecting mistakes in the test code.

Returns

Returns the applied soft limit, which should be the same as the hard limit.

See also

oomfill_enable
oomfill_disable
oomfill_fill
oomfill_free

Definition at line 210 of file oomfill.c.

2.5.4.2 oomfill_enable

```
size_t(* oomfill_enable) (const size_t softlimit) (
            const size_t softlimit ) =oomfill_enable_preinit
```

Starts a new oomfill environment or reconfigure the soft limit.

This function can only be invoked after at least a first call to oomfill_config to initialize the oomfill helpers environment. If invocated before, it will abort the current process to help detecting mistakes in the test code.

It will (soft)limit the current process and his children to the provided value in bytes. Then, it will enable the fill/free oomfill helper functions which are disabled otherwise.

It is designed to be invoked at the begining of a test case to apply for all tests in this testcase.

In case of any unexpected behavior, it should abort the current process.

Parameters

in	*softlimit*	Soft limit to set in bytes. It has to be less than the hardlimit otherwise it will fail and abort. If set to 0, it will apply the same value as the hardlimit.

Returns

Returns the actually set value as a softlimit, either the requested value or the hardlimit in case of 0 requested.

See also

oomfill_enable
oomfill_disable
oomfill_fill
oomfill_free

Definition at line 171 of file oomfill.c.

2.5.4.3 oomfill_fill

```
size_t(* oomfill_fill) (const size_t minHeap, const size_t minStack) (
            const size_t minHeap,
            const size_t minStack ) =oomfill_fill_preinit
```

Starts an almost OOM single and simple test.

This function is only enabled after a call to oomfill_enable and will have no effect otherwise. Its goal is to completely fill the RAM until the very last bytes, to keep only between minHeap and maxHeap bytes available in the heap and minStack bytes in the stack.

If oomfill_enable was not invoked beforehand, this function will simply return without any RAM consumption.

It should be called immediately before the test and should be reverted with oomfill_free immediately after. It can create so much pressure on the available memory that a simple printf could fail.

It is designed to fail if invoked twice as this is very probably a mistake in the test code. Should you need to change the RAM filling values, first call oomfill_free and reapply oomfill_fill. This ensure a really wanted behavior and not a mistake in your test code.

Parameters

in	*minHeap*	
in	*minStack*	

Returns

Returns the allocated size to fill the memory

See also

oomfill_enable

oomfill_disable

oomfill_fill

oomfill_free

Definition at line 96 of file oomfill.c.

2.5.4.4 oomfill_free

```
void(* oomfill_free) () ( ) =oomfill_free_preinit
```

Ends a single simple OOM test.

If this function is invoked between an oomfill_enable and an oomfill_disable invocations, it deallocates the RAM allocated by the oomfill_fill function. It should be called immediately after the single and simple test because the RAM pressure can even make a printf to fail.

If called without a prior oomfill_enable invocation, this function simply returns without any action.

This function is designed to fail and abort the process if invocated whereas there is no current RAM allocated, when called twice, for example. This helps to avoid mistakes in the test scenario.

Definition at line 125 of file oomfill.c.

2.6 oomfill.c

Go to the documentation of this file.

```
00001
00017 #include "liboom/oomfill.h"                      /* OOM simulation */
00018
00019 #include <assert.h>
00020 #include <stdio.h>                              /* printf */
00021 #include <stdlib.h>                             /* abort() */
00022 #include <errno.h>                              /* errno */
00023 #include <sys/resource.h>                       /* setrlimit/getrlimit */
00024 #include <string.h>                             /* strerror() */
00025 #include <alloca.h>                             /* alloca() */
00026 #include <unistd.h>                             /* sysconf() */
00027
00033 #define RAMBLOCKS_MAX 1000
00034
00038 static void* _oomblocks[RAMBLOCKS_MAX] = {0};
00039
```

```
00041 static int checked_getrlimit(int resource, struct rlimit *rlim) {
00042     /* Get current limit values */
00043     if (getrlimit(resource, rlim) != 0) {
00044         /* Can occur, thus not ignored, but impossible to trigger for gcov/lcov */
00045         fprintf (stderr,"%s:%d getrlimit() failed with errno=%d %s\n",
00046                  __FILE__,__LINE__, errno,strerror(errno));
00047         abort();
00048     }
00049     return 0;
00050 }
00051
00060 static size_t oomfill_fill_preinit(const size_t minHeap, const size_t minStack) {
00061     (void)minHeap;
00062     (void)minStack;
00063     assert(NULL==_oomblocks[0]);
00064     return 0;
00065 }
00066
00096 size_t (*oomfill_fill)(const size_t minHeap, const size_t minStack)=oomfill_fill_preinit;
00097
00106 static void oomfill_free_preinit() {
00107     assert(NULL==_oomblocks[0]);
00108 }
00109
00125 void (*oomfill_free)()=oomfill_free_preinit;
00126
00135 static size_t oomfill_enable_preinit(const size_t softlimit) {
00136     (void)softlimit;
00137
00138     /* Should not be called without configuration */
00139     fprintf(stderr,"%s:%d oomfill_enable called without oomfill_config before\n",__FILE__,__LINE__);
00140     abort();
00141 }
00142
00171 size_t (*oomfill_enable)(const size_t softlimit)=oomfill_enable_preinit;
00172
00181 static size_t oomfill_disable_preinit() {
00182     /* Should not be called before configuration */
00183     fprintf(stderr,"%s:%d oomfill_disable called without oomfill_config before\n",__FILE__,__LINE__);
00184     abort();
00185 }
00186
00210 size_t (*oomfill_disable)()=oomfill_disable_preinit;
00211
00223 static size_t oomfill_getbiggestblock(void** p_ramblock) {
00224     /* This function could receive an optimized "max" value and return the final
00225 * "max" value to the caller.  So, it could be used as the next invocation
00226 * starting "max" value.  This would save few loop iterations, at the cost of
00227 * extra stack usage, I chose to not pass this value as a parameter to avoid
00228 * consuming stack.
00229 * The function starts to search between 0..rlim_cur which has very little
00230 * impact given the Olog2 complexity.  It could be optimized and start to
00231 * search between 0..sysconf(AVPHYSPAGE*PAGESIZE), assuming that sysconf is
00232 * faster than few loop iterations.  */
00233
00234     /* Use static to avoid stack allocation/free */
00235     static size_t max,cur;
00236     static struct rlimit limit;
00237
00238     assert(NULL!=p_ramblock);
00239     assert(NULL==*p_ramblock);
00240
00241     /* Get the current limits */
00242     checked_getrlimit(RLIMIT_AS, &limit);
00243     max = limit.rlim_cur;
00244
00245     /* Restart the whole process if cur can not be allocated at the end */
00246     while ((max>0)&&(NULL==*p_ramblock)) {
00247         static size_t min;
00248         /* Iterate quickly (Olog2) to converge to the biggest available RAM block */
00249         min = 0;
00250         while (max>min) {
00251             cur = min+(max-min)/2; /* To avoid overflow */
00252             if (NULL==(*p_ramblock = malloc(cur))) {
00253                 max = cur;
00254             } else {
00255                 min = cur+1;
00256                 free(*p_ramblock);
00257             }
00258         }
00259         cur -= 1;
00260         *p_ramblock=malloc(cur);
00261     }
00262
00263     return cur;
00264 }
00265
```

```
00273 static size_t oomfill_fill_postinit(const size_t minHeap, const size_t minStack) {
00274     unsigned int l_numblock;
00275     size_t l_sum;
00276     void* volatile l_reservedheap;
00277     void* l_reservedstack;
00278 
00279     /* Probably a mistake in the test code, do not accept despite we could */
00280     if (NULL!=_oomblocks[0]) {
00281         /* Dirty hack to free some RAM and allow abort to SIGABRT */
00282         free(_oomblocks[0]);
00283         fprintf (stderr,"%s:%d oomblocks are already allocated\n", __FILE__,__LINE__);
00284         abort();
00285     }
00286 
00287     /* Reserve/Protect stack bytes which will be auto freed at return */
00288     /* A failure means a stackoverflow, which is not recoverable and will abort
00289 * the process anyway.  */
00290     if (0<minStack)
00291         l_reservedstack=alloca(minStack);
00292     (void)l_reservedstack;
00293 
00294     /* Reserve/Protect heap bytes which will be released before return */
00295     l_reservedheap=NULL;
00296     if (0<minHeap)
00297         if (NULL==(l_reservedheap=malloc(minHeap))) {
00298             /* This will fail if minHeap is higher than rlimit */
00299             fprintf(stderr,"%s:%d Failed to reserve minheap bytes\n",__FILE__,__LINE__);
00300             abort();
00301         }
00302 
00303     /* Find and allocate the biggest available RAM blocks until no mre RAM
00304 * available or all _oomblocks are allocated */
00305     l_numblock=0;
00306     l_sum = 0;
00307     l_sum += oomfill_getbiggestblock(&(_oomblocks[l_numblock++]));
00308     while ((NULL!=_oomblocks[l_numblock-1])&&((RAMBLOCKS_MAX-1)>l_numblock))
00309         l_sum += oomfill_getbiggestblock(&(_oomblocks[l_numblock++]));
00310     /* Either already NULL, which stopped the while loop or reached the last
00311 * slot in the table, which stopped the while loop and the slot has to be
00312 * set to NULL */
00313     _oomblocks[l_numblock]=NULL;
00314 
00315     /* There can be less than 4 bytes available at this point !!!  */
00316 
00317     /* Make the protected heap bytes available again */
00318     if (NULL!=l_reservedheap)
00319         free(l_reservedheap);
00320 
00321     return l_sum;
00322 }
00323 
00331 static void oomfill_free_postinit() {
00332     unsigned int l_i;
00333 
00334     /* Despite we could manage, abort to help detecting mistakes in test code */
00335     if (NULL==_oomblocks[0]) {
00336         fprintf(stderr,"%s:%d no blocks to free in oomfill_free.\n",__FILE__,__LINE__);
00337         abort();
00338     }
00339 
00340     /* Actually free allocated blocks and set their pointer to NULL */
00341     l_i = 0;
00342     while ((l_i<(RAMBLOCKS_MAX-1))&&(_oomblocks[l_i])) {
00343         free(_oomblocks[l_i]);
00344         _oomblocks[l_i] = NULL;
00345         l_i+=1;
00346     }
00347 }
00348 
00356 static size_t oomfill_enable_postinit(const size_t softlimit) {
00357     struct rlimit limit;
00358     size_t l_softlimit = softlimit;
00359 
00360     /* Probably a bug in oomfill */
00361     assert(NULL==_oomblocks[0]);
00362 
00363     /* Get current limit values */
00364     checked_getrlimit(RLIMIT_AS, &limit);
00365 
00366     /* Check the requested value */
00367     if (0==l_softlimit) {
00368         /* Defaults to available physical RAM if requested 0 */
00369         l_softlimit = limit.rlim_max;
00370     };
00371 
00372     /* Soft limit available RAM */
00373     limit.rlim_cur = l_softlimit;
```

```
00374
00375     /* Abort if setrlimit fails to avoid RAM bombing */
00376     if (setrlimit(RLIMIT_AS, &limit) != 0) {
00377         fprintf (stderr,"%s:%d setrlimit(cur=%lu, max=%lu) with errno=%d %s\n",
00378                  __FILE__,__LINE__,
00379                  (unsigned long)limit.rlim_cur, (unsigned long)limit.rlim_max,
00380                  errno,strerror(errno));
00381         abort();
00382     }
00383
00384     /* Activate fill/free functions */
00385     oomfill_fill = oomfill_fill_postinit;
00386     oomfill_free = oomfill_free_postinit;
00387
00388     /* Return the actual current soft limit */
00389     return limit.rlim_cur;
00390 }
00391
00399 static size_t oomfill_disable_postinit() {
00400     struct rlimit limit;
00401
00402     if (NULL!=_oomblocks[0]) {
00403         fprintf(stderr,"%s:%d RAM still allocated while calling oomfill_disable\n",__FILE__,__LINE__);
00404         abort();
00405     }
00406
00407     /* Do not allow calling disable if not enabled to detect test mistakes */
00408     if ((oomfill_fill!=oomfill_fill_postinit)||(oomfill_free!=oomfill_free_postinit)) {
00409         fprintf(stderr,"%s:%d Impossible to disable oomfill if not previously
      enabled\n",__FILE__,__LINE__);
00410         abort();
00411     }
00412
00413     /* Reset the soft limit to hard limit */
00414     oomfill_enable_postinit(0);
00415
00416     /* Get current limit values */
00417     checked_getrlimit(RLIMIT_AS, &limit);
00418
00419     /* Restore disabled functors */
00420     oomfill_fill = oomfill_fill_preinit;
00421     oomfill_free = oomfill_free_preinit;
00422
00423     /* Return the actual current soft limit, which is equal to hard limit */
00424     return limit.rlim_cur;
00425 }
00426
00457 size_t oomfill_config(const size_t hardlimit) {
00458     struct rlimit limit;
00459     size_t l_avail;
00460     size_t l_hardlimit = hardlimit;
00461
00462     /* Probably a test implementation mistake */
00463     if (NULL!=_oomblocks[0]) {
00464         fprintf(stderr,"%s:%d Calling oomfill_config with allocated RAM blocks is not allowed.\n",
00465                 __FILE__,__LINE__);
00466         abort();
00467     }
00468
00469     /* Find *installed* physical RAM, 0 in case of failure */
00470     l_avail = (sysconf(_SC_PHYS_PAGES) * sysconf(_SC_PAGESIZE));
00471
00472     /* Get current limit values */
00473     checked_getrlimit(RLIMIT_AS, &limit);
00474     if (0==l_hardlimit) {
00475         /* Defaults to available physical RAM or already set rlimit
00476 * if 0 requested */
00477         l_hardlimit=(limit.rlim_max<l_avail?limit.rlim_max:l_avail);
00478     } else if (l_hardlimit>l_avail) {
00479         /* Fails if request is over available physical RAM to avoir swapping */
00480         fprintf(stderr,"%s:%d Requesing a limit %lu bigger than *installed* RAM %lu is not
      allowed.\n",
00481                 __FILE__,__LINE__,
00482                 (unsigned long)l_hardlimit,(unsigned long)l_avail);
00483         abort();
00484     }
00485
00486     /* Hard limit available RAM to hardlimit globally with no way back */
00487     limit.rlim_cur = l_hardlimit;
00488     limit.rlim_max = l_hardlimit;
00489
00490     /* Abort if setrlimit fails to avoid RAM bombing */
00491     if (setrlimit(RLIMIT_AS, &limit) != 0) {
00492         fprintf (stderr,"%s:%d setrlimit(cur=%lu, max=%lu) with errno=%d %s\n",
00493                  __FILE__,__LINE__, (unsigned long)limit.rlim_cur,
00494                  (unsigned long)limit.rlim_max, errno,strerror(errno));
00495         /* Get current limit values */
```

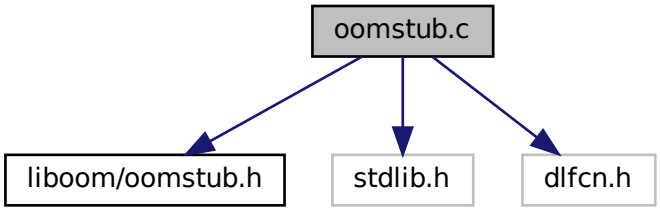
```
00496         checked_getrlimit(RLIMIT_AS, &limit);
00497         fprintf (stderr,"%s:%d getrlimit() is cur=%lu, max=%lu\n",
00498                  __FILE__,__LINE__, (unsigned long)limit.rlim_cur,
00499                  (unsigned long)limit.rlim_max);
00500         abort();
00501     }
00502
00503     /* Activate enable/disable functions */
00504     oomfill_enable = oomfill_enable_postinit;
00505     oomfill_disable = oomfill_disable_postinit;
00506
00507     /* Return the configured limit hard=soft */
00508     return limit.rlim_max;
00509 }
00510
00511 bool oomfill_enabled() {
00512     return (oomfill_fill==oomfill_fill_postinit?true:false);
00513 }
00514 /* vim:  set tw=80:  */
```

2.7 oomstub.c File Reference

C memory management stubs.

```
#include "liboom/oomstub.h"
#include <stdlib.h>
#include <dlfcn.h>
```

Include dependency graph for oomstub.c:



Macros

- #define RTLD_NEXT ((void *) -1L)

Functions

- void oomstub_setcountdown (const int counter)

  Sets the malloc countdown before triggering a failure.
- int oomstub_getcountdown ()

  Gets the current malloc countdown before triggering a failure.
- void * malloc (size_t size)

  stdc malloc stub function
- void * calloc (size_t count, size_t size)

  stdc calloc stub function
- void * realloc (void *ptr, size_t new_size)

  stdc realloc stub function

2.7.1 Detailed Description

C memory management stubs.

This file implements fake malloc, free and realloc functions which can fail and simulate OOM after a countdown. This library is not threadsafe. Inspired from https://github.com/bkthomps/Containers test suite

Author

François Cerbelle (Fanfan), francois@cerbelle.net

Definition in file oomstub.c.

2.7.2 Macro Definition Documentation

2.7.2.1 RTLD_NEXT

```
#define RTLD_NEXT ((void *) -1L)
```

Definition at line 28 of file oomstub.c.

2.7.3 Function Documentation

2.7.3.1 calloc()

```
void * calloc (
            size_t count,
            size_t size )
```

stdc calloc stub function

Parameters

count	How many contiguous records to allocate
size	Size in bytes of a single record

Returns

Pointer to the allocated heap block

Return values

NULL	The allocation failed

This function is called instead of the upstream calloc function. If not already done, it creates a backup ot the upstream function pointer. If the countdown equals 1, it simulates OOM and returns NULL. Otherwise, it calls the upstream calloc and returns its return value. Each call to this stub function decrements the countdown until it reaches 0.

Definition at line 113 of file oomstub.c.

2.7.3.2 malloc()

```
void * malloc (
            size_t size )
```

stdc malloc stub function

Parameters

size	Size in bytes to try to allocate

Returns

Pointer to the allocated heap block

Return values

NULL	The allocation failed

This function is called instead of the upstream malloc function. If not already done, it creates a backup ot the upstream function pointer. If the countdown equals 1, it simulates OOM and returns NULL. Otherwise, it calls the upstream malloc and returns its return value. Each call to this stub function decrements the countdown until it reaches 0.

Definition at line 87 of file oomstub.c.

2.7.3.3 oomstub_getcountdown()

```
int oomstub_getcountdown ( )
```

Gets the current malloc countdown before triggering a failure.

Returns

The current internal countdown value

See also

malloc

calloc

realloc

Definition at line 72 of file oomstub.c.

2.7.3.4 oomstub_setcountdown()

```
void oomstub_setcountdown (
            const int counter )
```

Sets the malloc countdown before triggering a failure.

Parameters

counter	Value to set in the internal countdown

See also

malloc

calloc

realloc

Definition at line 62 of file oomstub.c.

2.7.3.5 realloc()

```
void * realloc (
            void * ptr,
            size_t new_size )
```

stdc realloc stub function

Parameters

ptr	Pointer to the dynamic heap block to resize
new_size	New size in bytes to request

Returns

Pointer to the reallocated heap block

Return values

NULL	The allocation failed (important details in description)

This function is called instead of the upstream realloc function. If not already done, it creates a backup ot the upstream function pointer. If the countdown equals 1, it simulates OOM and returns NULL. Otherwise, it calls the upstream realloc and returns its return value. Each call to this stub function decrements the countdown until it reaches 0.

Note

The failure countdown should not apply when shrinking the size. It is meaningless as it can not fail in real life. It would need bookkeeping to detect arbitrary shrinks. Thus, this stub function also counts shrinking calls in the countdown and apply failures. The tradeof is that the function returns NULL in case of a simulated failure and a pointer in case of success for any case but not for the "free" case. In the "free" case (new_size==0), it returns a not-null value in case of failure and NULL in case of success.

Warning

if the compiler optimizes a realloc(ptr,0) with a direct free(ptr) call, the countdown is not decremented. I can not detect this condition and I have no tradeoff/workaround. Diverting free() would decrement the countdown when not expected.

Definition at line 153 of file oomstub.c.

2.8 oomstub.c

Go to the documentation of this file.

```
00001
00023 #include "liboom/oomstub.h"
00024 #include <stdlib.h>                             /* NULL */
00025 #include <dlfcn.h>
00026
00027 #ifndef RTLD_NEXT
00028 #define RTLD_NEXT ((void *) -1L)
00029 #endif
00030
00036 static int oomstub_countdown = 0;
00037
00042 static void *(*real_malloc)(size_t);
00043
00048 static void *(*real_calloc)(size_t, size_t);
00049
00054 static void *(*real_realloc)(void *, size_t);
00055
00062 void oomstub_setcountdown(const int counter) {
00063     oomstub_countdown = counter;
00064 }
00065
00072 int  oomstub_getcountdown() {
00073     return oomstub_countdown;
00074 }
00075
00087 void *malloc(size_t size) {
00088     if (!real_malloc) {
00089         *(void **) (&real_malloc) = dlsym(RTLD_NEXT, "malloc");
00090     }
00091     if (oomstub_countdown == 1) {
00092         oomstub_countdown = 0;
00093         return NULL;
00094     }
00095     if (oomstub_countdown > 0) {
00096         oomstub_countdown--;
00097     }
00098     return real_malloc(size);
00099 }
```

```
00100
00113 void *calloc(size_t count, size_t size) {
00114     if (!real_calloc) {
00115         *(void **) (&real_calloc) = dlsym(RTLD_NEXT, "calloc");
00116     }
00117     if (oomstub_countdown == 1) {
00118         oomstub_countdown = 0;
00119         return NULL;
00120     }
00121     if (oomstub_countdown > 0) {
00122         oomstub_countdown--;
00123     }
00124     return real_calloc(count, size);
00125 }
00126
00153 void *realloc(void *ptr, size_t new_size) {
00154     if (!real_realloc) {
00155         *(void **) (&real_realloc) = dlsym(RTLD_NEXT, "realloc");
00156     }
00157     if (oomstub_countdown == 1) {
00158         oomstub_countdown = 0;
00159         if (new_size > 0)
00160             return NULL;
00161         else
00162             return ptr;
00163     }
00164     if (oomstub_countdown > 0) {
00165         oomstub_countdown--;
00166     }
00167
00168     return real_realloc(ptr, new_size);
00169 }
00170
00171 /* vim:  set tw=80:  */
```

2.9 revision.h File Reference

Macros

- #define REVISION "ed5c251ac160"

2.9.1 Macro Definition Documentation

2.9.1.1 REVISION

```
#define REVISION "ed5c251ac160"
```

Definition at line 3 of file revision.h.

2.10 revision.h

Go to the documentation of this file.

```
00001 /* This file is updated in the distdir before creating the dist archive */
00002 #ifndef REVISION
00003 #define REVISION "ed5c251ac160"
00004 #endif
```

Index

bool
  oomfill.h, 4

calloc
  oomstub.c, 20

false
  oomfill.h, 5

malloc
  oomstub.c, 21

oomfill.c, 11, 15
  oomfill_config, 12
  oomfill_disable, 13
  oomfill_enable, 13
  oomfill_enabled, 13
  oomfill_fill, 14
  oomfill_free, 15
  RAMBLOCKS_MAX, 12
oomfill.h, 3, 8
  bool, 4
  false, 5
  oomfill_config, 5
  oomfill_disable, 6
  oomfill_enable, 6
  oomfill_enabled, 5
  oomfill_fill, 7
  oomfill_free, 8
  true, 5
oomfill_config
  oomfill.c, 12
  oomfill.h, 5
oomfill_disable
  oomfill.c, 13
  oomfill.h, 6
oomfill_enable
  oomfill.c, 13
  oomfill.h, 6
oomfill_enabled
  oomfill.c, 13
  oomfill.h, 5
oomfill_fill
  oomfill.c, 14
  oomfill.h, 7
oomfill_free
  oomfill.c, 15
  oomfill.h, 8
oomstub.c, 19, 23
  calloc, 20
  malloc, 21
  oomstub_getcountdown, 21
  oomstub_setcountdown, 22
  realloc, 22
  RTLD_NEXT, 20
oomstub.h, 9, 10
  oomstub_getcountdown, 9
  oomstub_setcountdown, 10
oomstub_getcountdown
  oomstub.c, 21
  oomstub.h, 9
oomstub_setcountdown
  oomstub.c, 22
  oomstub.h, 10

RAMBLOCKS_MAX
  oomfill.c, 12
realloc
  oomstub.c, 22
REVISION
  revision.h, 24
revision.h, 24
  REVISION, 24
RTLD_NEXT
  oomstub.c, 20

true
  oomfill.h, 5