How do I mock memory allocation failures?

Asked 10 years, 6 months ago Active 10 years, 6 months ago Viewed 2k times

3

3

I want to extensively test some pieces of C code for memory leaks.

On my machine I have 4 Gb of RAM, so it's very unlikely for a dynamic memory allocation to fail. Still I want to see the comportment of the code if memory allocation fails, and see if the recover mechanism is "strong" enough.

What do you suggest ? How do I emulate an environment with lower memory specs ? How do i mock my tests ?

EDIT: I want my tests to be code independent. I only have "access" to return values for different functions in the library I am testing. I am not supposed to write "test logic" inside the code I am testing.

c testing memory-leaks mocking memory-management

edited Apr 7 '10 at 11:57

asked Apr 7 '10 at 11:51

Andrei Ciobanu
11.2k 20 71 115

3 what does code independant mean? – Jennifer Zouak Apr 7 '10 at 11:58

if it really should be code independant, then this question does not belong here (on Server Fault ?). You can easily set up a VM with as few memory as you want. – kriss Apr 7 '10 at 12:02

Is valgrind not available? Massif is an *excellent* heap profiling tool. – Tim Post ♦ Apr 7 '10 at 12:21

6 Answers

Active | Oldest | Votes

3

To do this code independently, I'd suggest using a virtual machine with a much smaller memory setting and running your code within that. Otherwise, you are left with testing on actual systems with smaller memory configurations.

answered Apr 7 '10 at 12:00

Lazarus
36.6k 4 39 52

1. You still have zero chance to test all possible allocation failures and their handling. – zzz777 Dec 27 '18 at 3:28

Can you expand on that, i.e., what are the categories of allocation failure that you envisage? – Lazarus Jan 20 '19 at 9:39

In unit test code, I override all standard new and delete operations. I can set size for the next allocation failure, and how many allocations it should handle before returning failure. Then I do normal code coverage and see what allocations I should fail. This way is very flaky but the only one I came up with. Moreover it is a completely useless exercise just to have good code coverage numbers. None of the systems I know will function at all when run out of memory and none of the systems I am working on will run out of memory unless there is a leak. – zzz777 Jan 21 '19 at 1:27

6

Make a wrapper malloc and free, you can put your own logic there for when it fails and when it doesn't.

Then:

```
#define malloc(X) (myMalloc(X))
#define free(X) (myFree(X))
#define realloc(X, Y) (myRealloc(X, Y))
#define calloc(X, Y) (myCalloc(X, Y))
#define valloc(X) (myValloc(X))
```

You can `#define` and `#undef` the macros as you want throughout your code.

edited Apr 7 '10 at 12:04

answered Apr 7 '10 at 11:55

Brian R. Bondy
303k 110 568 615

ad another problem is that with a really small memory footprint it can also be stack limit that is reached before heap limit. Hence memory starvation does not necessarilly happen on mallocs but could happen at any function call that usually works... :-(– kriss Apr 7 '10 at 12:33

3

If this is a unix -type of system, then you can do the same thing that jemalloc does. Implement your own malloc, compile it as .so -library and start the test with LD_PRELOAD=yourmalloc.so

You could build in a setting that will allow you to control how it operates:

```
void tester() {
     your_malloc_allow_allocation(1024*1024*4); // allow 4 more megs of allocation

     librarycall();
     // ... handle errors..

     your_malloc_reset_limits(); // drop limits
}
```

answered Apr 7 '10 at 13:18

Pasi Savolainen
2,204 1 19 32

1

When I had to do this, I made a small program which used up all the memory quickly. Something like this:

```
// Beware, brain-compiled code ahead:
#include <stdlib.h>

bool alloc(size_t n)
{
    return NULL != malloc(n);
}

int main()
{
    size_t n = 1;
    for(;;) {
        while( alloc(n) )
            n*=2;
        do
            n/=2;
        while( !alloc(n) );
    }
}
```

My experience with the NT line of Windows was that, while the OS was rock-solid, just about everything else crashed on me, including the debugger. No fun to work that way.

answered Apr 7 '10 at 12:01

sbi
198k 44 232 423

1

type in *(char*)0; when you want your code to fail or store it in allocated var

(obviously it should be activated/desactivated by some #define setted by makefile)

The idea is to precisely control which malloc you want to fail. I used the previous trick to ensure what would be the behavior of a program I wrote if a segfault ever occured. I was able to check that the exception was caught and that there was no memory leak with allready allocated object.

You can also add some kind of counter to make your alloc return 0 only after a while and check that your code corretly handle this case (and, for instance, correctly free parts of what your program choose to destroy to handle memory starvation).

edited Apr 7 '10 at 12:23

answered Apr 7 '10 at 11:53

kriss
20.5k 13 86 109

How is this helpful ? Can you please be more specific ? – Andrei Ciobanu Apr 7 '10 at 11:54

when malloc fail that's what it will do, return 0... you can easily fake it. I'm not saying anything more. – kriss Apr 7 '10 at 11:55

Ok, probably i haven't being descriptive enough. But I want my tests to be code independent. – Andrei Ciobanu Apr 7 '10 at 11:57

If you are suggesting to have a #define that replaces malloc calls with (char *)0, then this will fail on the very first allocation, which is not very good test. – Laurynas Biveinis Apr 7 '10 at 12:01

that's not what I'm saying. I'm just stating that you can control malloc behavior by setting 0 in some cases. And you can exactly control which malloc you want to test failing. If you set a global wrapper you will test something, but not something very controlled as any malloc could fail in the lifecycle of the program. It's very like a monkey test. I happened to use the previous trick to see how my program (multithreaded) behave when a segfault occur. So I really voluntarily started some segfaults using previous trick. – kriss Apr 7 '10 at 12:07

I

0

disable swap, then at the beginning of your app do `while(malloc(1000000));`

This will leave the environment with less than 1MB of free memory for the rest of your app to run. Adjust the value for other results.

answered Apr 7 '10 at 12:00

SF.
11.9k 11 61 100