Dynamic Memory Allocation Failure Recovery

Asked 16 years ago Modified 15 years, 11 months ago Viewed 4k times

6

I'm working on an embedded processor (400 MHz Intel PXA255 XScale), and I thought I saw one case where there wasn't enough memory to satisfy a 'new' operation. The program didn't crash, so I assumed other threads had freed their memory and it was just a transient thing. This is some pretty critical code, so exiting is not an option, and some sort of error needs to be returned to the remote user.

Would the following small fix be enough to solve the problem, or is there a better way? Before replacing every 'new' with the following code, I thought I'd ask.

```
char someArr[];
do{
    someArr = new char[10];
    Sleep(100); // no justification for choosing 100 ms
} while ( someArr == NULL );
```

Does the Sleep help? Should I set some max number of retries? Is it possible to use static initialization everywhere?

FINAL UPDATE: Thank you very much for the helpful responses, but it turns out there was an error in the code checking for failed memory allocation. I will keep all of these answers in mind, and replace as many malloc's and new's as I can, though (especially in error-handling code).

c++ embedded windows-ce new-operator viper

Share
Improve this question
Follow

edited Jan 10, 2009 at 8:22

asked Dec 15, 2008 at 2:02

Nate Parsons
14.9k ● 13 ● 53 ● 67

9 Answers

Sorted by: Highest score (default)

16

You are trying to solve a global problem through local reasoning. The global problem is that the entire device has a limited amount of RAM (and possibly backing store) for the operating system and all of the applications. To make sure this amount of RAM is not exceeded, you have a few options:

- Each process operates in a fixed amount of RAM to be determined per process at startup time; the programmer does the reasoning to make sure everything fits. So, **yes, it is possible to allocate everything statically**. It's just a lot of work, and **every time you change your system's configuration, you have to reconsider the allocations**.
- Processes are aware of their own memory usage and needs and continually advise each other about how much memory they need. They **cooperate so they don't run out of memory**. This assumes that at least some processes in the system can adjust their own memory requirements (e.g., by changing the size of an internal cache). Alonso and Appel wrote a paper about this approach.
- Each process is aware that memory can become exhausted and can **fail over to a state in which it consumes a minimum amount of memory**. Often this strategy is implemented by having an out-of-memory exception. The exception is handled in or near main() and the out-of-memory event essentially restarts the program from scratch. This failover mode can work if memory grows in response to user requests; if the program's memory requirements grow independent of what the user does, **it can lead to thrashing.**

Your proposal above matches none of the scenarios. Instead, you are hoping **some other process will solve the problem** and the memory you need will eventually appear. You might get lucky. You might not.

If you want your system to work reliably, you would do well to **reconsider the design of every process running on the system** in light of the need to share limited memory. It might be a bigger job than you expected, but if you understand the problem, you can do this. Good luck!

Share Improve this answer Follow

answered Dec 15, 2008 at 2:44

Norman Ramsey

202k ● 62 ● 371 ● 541

> The problem behind all this is that about 99% of all code has no concept how to handle failed memory allocations gracefully. Apps fail in mysterious ways. No free memory brings almost all operating systems to a screeching halt. Sad but true. – Thorsten79 Dec 29, 2008 at 9:52

2

There are a few different ways to attack this - note that the tool instructions will vary a bit, based on what version of Windows CE / Windows Mobile you are using.

Some questions to answer:

1. Is your application leaking memory, leading to this low memory condition?

2. Does your application simply use too much memory at certain stages, leading to this low memory condition?

1 and 2 can be investigated using the Windows CE AppVerifier tool, which can provide detailed memory logging tools for your product. Other heap wrapping tools can also provide similar information (and may be higher-performance), depending on your product design.

http://msdn.microsoft.com/en-us/library/aa446904.aspx

3. Are you allocating and freeing memory very frequently in this process?

Windows CE, prior to OS version 6.0 (don't confuse with Windows Mobile 6.x) had a 32MB / process virtual memory limit, which tends to cause lots of fun fragmentation issues. In this case, even if you have sufficient physical memory free, you might be running out of virtual memory. Use of custom block allocators is usually a mitigation for this problem.

4. Are you allocating very large blocks of memory? (> 2MB)

Related to 3, you could just be exhausting the process virtual memory space. There are tricks, somewhat dependent on OS version, to allocate memory in a shared VM space, outside the process space. If you are running out of VM, but not physical RAM, this could help.

5. Are you using large numbers of DLLs?

Also related to 3, Depending on OS version, DLLs may also reduce total available VM very quickly.

Further jumping off points:

Overview of CE memory tools

http://blogs.msdn.com/ce_base/archive/2006/01/11/511883.aspx

Target control window 'mi' tool

http://msdn.microsoft.com/en-us/library/aa450013.aspx

Share Improve this answer Follow

answered Dec 28, 2008 at 8:00

Teman

36 • 2

There are lots of good things in the other answers, but I did think it worth adding that if all the threads get in a similar loop, then the program will be deadlocked.

2

The "correct" answer to this situation is probably to have strict limits for the different parts of the program to ensure that they don't over consume memory. That would probably require rewriting major sections across all parts of the program.

The next best solution would be to have some callback where a failed allocation attempt can tell the rest of the program that more memory is needed. Perhaps other parts of the program can release some buffers more aggressively than they normally would, or release memory used to cache search results, or something. This would require new code for other parts of the program. However, this could be done incrementally, rather than requiring a rewrite across the entire program.

Another solution would be to have the program protect large (temporary) memory requests with a mutex. It sounds like you are confident that memory will be released soon if you can just try again later. I suggest that you use a mutex for operations that might consume a lot of memory, this will allow the thread to be woken up immediately when another thread has released the memory that is needed. Otherwise your thread will sleep for a tenth of a second even if the memory frees up immediately.

You might also try sleep(0), which will simply hand off control to any other thread that is ready to run. This will allow your thread to regain control immediately if all other threads go to sleep, rather than having to wait out its 100 millisecond sentence. But if at least one thread still wants to run, you will still have to wait until it gives up control. This is typically 10 milliseconds on Linux machines, last I checked. I don't know about other platforms. Your thread may also have a lower priority in the scheduler if it has voluntarily gone to sleep.

Share
Improve this answer
Follow

edited Dec 20, 2008 at 5:10

answered Dec 15, 2008 at 3:31

markets
9,674 ● 7 ● 36 ● 33

1

Based on your question, I'm assuming that your heap is shared between multiple threads.

If it isn't then the code above won't work, because nothing will be freed from the heap while the loop is running.

If the heap is shared, then the above would probably work. However, if you have a shared heap, then calling "new" will probably result in either a spin lock (a similar loop to the one you have, but using CAS instructions), or it will block based on some kernel resources.

In both cases, the loop you have will decrease the throughput of your system. This is because you will either incur more context switches then you need to, or will take

longer to respond to the "memory is now available" event.

I would consider overriding the "new" and "delete" operators. When new fails you can block (or spin lock on a counter variable of some sort) waiting for another thread to free memory, and then delete can either signal the blocked "new" thread or increment the counter variable using CAS.

That should give you better throughput and be a bit more efficent

Share Improve this answer Follow

answered Dec 15, 2008 at 2:16

Scott Wisniewski
25k ● 8 ● 62 ● 90

1

A few points:

- Embedded programs often allocate all memory at startup or use only static memory to avoid situations like this.
- Unless there is something else running on the device that frees memory on a regular basis your solution isn't likely to be effective.
- The Viper I have has 64MB RAM, I don't think they come with less than 32MB, how much memory is your application using?

Share
Improve this answer
Follow

edited Dec 15, 2008 at 2:35

answered Dec 15, 2008 at 2:21

Robert Gamble
109k ● 25 ● 147 ● 138

1

I second that the most sensible thing to do is to use static allocation of memory, so you have some idea of what is going on. Dynamic memory allocation is a bad habit from desktop programming that is not suited on restricted-resource machines (unless you spend a fair bit of time and effort creating a good managed and controlled memory management system).

Also, check what features the OS in your device (assuming it has one, high-end ARM devices like this one tends to run an OS) has for handling memory.

Share Improve this answer Follow

answered Dec 17, 2008 at 8:36

jakobengblom2
5,843 ● 2 ● 28 ● 34

You use C++. So you can make use of some C++ utilities to make your life easier. For example, why not use new_handler?

1

```
void my_new_handler() {
    // make room for memory, then return, or throw bad_alloc if
    // nothing can be freed.
}

int main() {
    std::set_new_handler(&my_new_handler);

    // every allocation done will ask my_new_handler if there is
    // no memory for use anymore. This answer tells you what the
    // standard allocator function does:
    // https://stackoverflow.com/questions/377178
}
```

In the new_handler, you could send all applications a signal so that they know that memory is needed for some application, and then wait a bit to give other applications the time to fulfill the request for memory. Important is that you **do something** and not **silently hope** for available memory. The new operator will call your handler again if still not enough memory is available, so you don't have to worry about whether or not all applications have free'ed the needed memory already. You can also **overload operator new** if you need to know the size of memory that is needed in the new_handler. See my other answer on how to do that. This way, you have **one central place** to handle memory problems, instead of many places concerned with that.

Share

Improve this answer

Follow

edited May 23, 2017 at 10:33

Community Bot

1 ● 1

answered Dec 28, 2008 at 7:28

Johannes Schaub - litb

506k ● 131 ● 917 ● 1.2k

1

As others have mentioned, ideally, you would avoid this problem by up-front design and software architecture, but I'm assuming at this point that really isn't an option.

As another post mentions, it would be good to wrap the logic in some utility functions so that you don't end up writing the out-of-memory code all of the place.

To get to the real problem, you try to use a shared resource, memory, but are unable to because that shared resource is being used by another thread in the system. Ideally what you would like to do is wait for one of the other threads in the system to release the resource you need, and then acquire that resource. If you had a way of intercepting all the allocation and free calls you could set-up something so that the allocating thread blocked until memory was available, and the freeing signalled the allocating thread when memory was available. But I'm going to assume that is simply too much work.

Given the constraints of not being able to totally re-architect the system, or re-write the memory allocator, then I think your solution is the most practical, as long as you

(and the others on your team), understand the limitations, and the problems it will cause down the track.

Now to improve your specific approach, you may want to measure the workloads to see how often memory is allocated and freed. This would give you a better ay of calculating what the retry interval should be.

Secondly, you way want to try increasing the time-out for each iteration to reduce the load of that thread on the system.

Finally, you definitely should have some time of error/panic case if the thread is unable to make progress after some number of iterations. This will let you at least see the potential live-lock case you may encounter if all threads are waiting for another thread in the system to free memory. You could simply pick a number of iterations based on what is empirically shown to work, or you could get smarter about it and keep track of how many threads are stuck waiting for memory, and if that ends up being all threads panic.

Note: This is obviously not a perfect solution and as the other posters have mentioned a more global view of the application as a whole is needed to solve the problem correctly, but the above is a practical technique that should work in the short-term.

Share Improve this answer Follow

answered Dec 29, 2008 at 11:46

benno

2,147 ● 1 ● 20 ● 24

0

Surely it would depend on whether you have a reasonable expectation of memory becoming available in the 100 (millisecond?) sleep? Certainly, you should limit the number of times it attempts.

To me something doesn't smell right here. Hmmm...

Embedded systems typically need to be extremely deterministic - perhaps you should review the entire system and head of the potential for this to fail up front; and then just fail hard it it actually happens in practice.

Share Improve this answer Follow

answered Dec 15, 2008 at 2:06

L. Cornelius Dol

63.9k ● 26 ● 141 ● 189