Subject: How to respond when memory is exceeded
Posted by koldo on Wed, 16 Oct 2024 07:41:57 GMT
View Forum Message <> Reply to Message

Hi all

Now, when you want to allocate a variable that is too large, the internal function calls Panic() and aborts the program. For the programmer, it is only possible to call InstallPanicMessageBox() first, so that just before closing the program, a defined function is called.

I wanted to ask you if it would be possible that in this case, the user function decides if the program stops or not. For example, imagine a design program that wants to open a file that is too big: instead of crashing the program, the PanicMessageBox() could do a throw with a warning that the file to be opened is too big, and the program would return to the previous state.

Subject: Re: How to respond when memory is exceeded
Posted by Lance on Wed, 16 Oct 2024 13:27:25 GMT
View Forum Message <> Reply to Message

Hello Koldo. It seem to me a very reasonable request.

I am not sure about the situation. So there is no exception like badalloc etc is thrown on allocation failure in our present state?

Subject: Re: How to respond when memory is exceeded
Posted by koldo on Wed, 16 Oct 2024 14:18:46 GMT
View Forum Message <> Reply to Message

Hi Lance

Now, when running this:
Buffer<int> data;
data.Alloc(123456789012, 0);
The calls are these:

Panic() function ends like this:
#ifdef _DEBUG
 __BREAK__;
#endif
 abort();

File Attachments
1) 3qfJAVbx9N.png, downloaded 423 times

Subject: Re: How to respond when memory is exceeded
Posted by Lance on Wed, 16 Oct 2024 15:22:24 GMT
View Forum Message <> Reply to Message

I don't have a computer in front of me right now but I get your point.

Maybe allow MemoryAllic to throw and only uncaught exceptions cause Panic be called?

Yours seems to me to be a typical situation that should be able to be addressed gracefully.

Subject: Re: How to respond when memory is exceeded
Posted by mirek on Sun, 17 Nov 2024 15:07:08 GMT
View Forum Message <> Reply to Message

This is a request that sound reasonable, but in practice is hard to implement, impossible to test and might have less impact than you think.

I mean, there are two reasons for out-of-memory situation:

- a bug. Some parameter in input data wrongly indicates that you have to alloc insane amount of memory and the allocation fails because it exceeds virtual memory. Your example falls into this category. So far I believe it is easier to account for that before the allocation (check the parameter) that trying to make the whole code out-of-memory resistant (e.g. check how String Stream::GetAll(int size) is implemented).

- a real memory exhaustion. That in hosts that we are interested means exhaustion of virtual memory space. In that case it is equally likely that the user part of host will crash too or that the computer will get virtually deadlocked trying to shuffle virtual memory or the process gets ultimately killed by OOM killer. Plus, to make things worse, it does not even has to happen in alloc! (Sometimes to really alloc the memory, you need to alloc it and then write to it...) So the effort you have invested in fixing all the code wents exactly nowehere.

(Fun fact: Last time I tried to demontrate futility of checking for out of memory with simple C code in Windows 10, I got blue screen as a result :)

Mirek

Subject: Re: How to respond when memory is exceeded
Posted by Lance on Mon, 18 Nov 2024 04:24:08 GMT
View Forum Message <> Reply to Message

I see.

Now the question is: if a programmer opt to use memory allocation allocation utility provided by the language or standard libary, is an std::bad_alloc exception guaranteed to be thrown and able to be caught afterwards? Or similar thing will happen that a supposed-to-be successfully-allocated memory fails when trying to write to it later and no mechanism to catch the memory exception exists?

Subject: Re: How to respond when memory is exceeded
Posted by mirek on Mon, 18 Nov 2024 07:01:34 GMT
View Forum Message <> Reply to Message

Lance wrote on Mon, 18 November 2024 05:24I see.

Now the question is: if a programmer opt to use memory allocation allocation utility provided by the language or standard libary, is an std::bad_alloc exception guaranteed to be thrown and able to be caught afterwards?

As far as my knowledge goes, no. But it is some time since I last checked, I suggest you to do some experimentation yourself.

Subject: Re: How to respond when memory is exceeded
Posted by koldo on Mon, 18 Nov 2024 07:17:18 GMT
View Forum Message <> Reply to Message

For my part, I have tried to modify the code so that in this situation an exception is thrown, and the behaviour has been excellent.

It is sad that in the old days you could check if a malloc() returned null, but now you are forced to close the program.

Mirek, with a very small change you will give the programmer a choice.

Subject: Re: How to respond when memory is exceeded
Posted by mirek on Mon, 18 Nov 2024 07:39:57 GMT
View Forum Message <> Reply to Message

koldo wrote on Mon, 18 November 2024 08:17For my part, I have tried to modify the code so that in this situation an exception is thrown, and the behaviour has been excellent.

It is sad that in the old days you could check if a malloc() returned null, but now you are forced to close the program.

Mirek, with a very small change you will give the programmer a choice.

NP, I can add something. Maybe you can add some code here so that I can check your problem and solution?

Subject: Re: How to respond when memory is exceeded
Posted by koldo on Mon, 18 Nov 2024 09:09:28 GMT

View Forum Message <> Reply to Message

Hello Mirek

This is a very simple test code to force the problem:

```
GUI_APP_MAIN
{
 WithMain<TopWindow> dlg;

 CtrlLayout(dlg, "Memory test");

 dlg.butTest.WhenAction = []{
  try {
   Buffer<int> data;
   data.Alloc(123456789012, 0);

   PromptOK("Memory is alloc");
  } catch (const std::bad_alloc& e) {
          Exclamation("Caught bad_alloc");
  }
 };

 dlg.Execute();
}
```

And this is the change in U++:

```
void OutOfMemoryPanic(size_t size)
{
 throw std::bad_alloc();

 /*char h[200];
 snprintf(h, 200, "Out of memory!\nU++ allocated memory: %d KB", MemoryUsedKb());
 Panic(h);*/
}
```

When pushing the button 5 times, no problem happens. Visually monitoring the Windows Task Manager, MemoryTest.exe seems to use the same memory all the time.
However, pushing the button one more time, MemoryLimitKb breached! Panic appears. It happens the same in debug and in release.

File Attachments

1) MemoryTest.7z, downloaded 99 times
2) firefox_A5OdAivPSK.png, downloaded 350 times

Subject: Re: How to respond when memory is exceeded
Posted by mirek on Mon, 18 Nov 2024 11:31:19 GMT
View Forum Message <> Reply to Message

koldo wrote on Mon, 18 November 2024 10:09Hello Mirek

This is a very simple test code to force the problem:

```
GUI_APP_MAIN
{
 WithMain<TopWindow> dlg;

 CtrlLayout(dlg, "Memory test");

 dlg.butTest.WhenAction = []{
  try {
   Buffer<int> data;
   data.Alloc(123456789012, 0);

   PromptOK("Memory is alloc");
  } catch (const std::bad_alloc& e) {
          Exclamation("Caught bad_alloc");
  }
 };

 dlg.Execute();
}
```

And this is the change in U++:

```
void OutOfMemoryPanic(size_t size)
{
 throw std::bad_alloc();

 /*char h[200];
 snprintf(h, 200, "Out of memory!\nU++ allocated memory: %d KB", MemoryUsedKb());
 Panic(h);*/
}
```

When pushing the button 5 times, no problem happens. Visually monitoring the Windows Task Manager, MemoryTest.exe seems to use the same memory all the time.
However, pushing the button one more time, MemoryLimitKb breached! Panic appears. It happens the same in debug and in release.

That is "buggy code" example. For something more real, try

```
#include <CtrlLib/CtrlLib.h>

using namespace Upp;

GUI_APP_MAIN
{
 TopWindow win;
 win.Title("Testing");
 win.WhenClose << [] {
  try {
   for(;;) {
    byte *data = new byte[1024 * 4096];
    for(int i = 0; i < 1024; i++)
     data[i * 4096] = 1;
   }
  } catch (const std::bad_alloc&) {
   Exclamation("Caught bad_alloc");
  }
 };
 win.Run();
}
```

And try that in Linux...

That said, whatever, I think this really is totally futile, but I am going to change the code to actually throw bad_alloc, then catch it in main and do Panic there. I am NOT going to change the U++ code to work correctly in case bad_alloc is thrown, but maybe I will be able to compile some rules (like everything in try block has to be destructed). What a waste of time...

Subject: Re: How to respond when memory is exceeded
Posted by Lance on Mon, 18 Nov 2024 15:15:28 GMT
View Forum Message <> Reply to Message

No std::bad_alloc thrown. Test program killed by os.

Test on Linux 6.8.0-48-generic

```
#include <iostream>
#include <memory>
#include <format>
#include <unistd.h>

using namespace std;
```

```
unsigned long long getTotalSystemMemory()
{
    long pages = sysconf(_SC_PHYS_PAGES);
    long page_size = sysconf(_SC_PAGE_SIZE);
    return pages * page_size;
}

int main(int argc, const char *argv[])
{
 auto size = getTotalSystemMemory()/4;

 std::unique_ptr<char []> p=std::make_unique<char[]>(size);

 while(true){
  auto new_size = size *1.1;
  try{
   p = std::make_unique<char []>(new_size);
  }catch(std::bad_alloc&){
   break;
  }
  size = new_size;
  std::cout<<std::format("{} bytes allocated ok.\n", size);
 }
 std::cout<<std::format("We believe {} bytes has been successfully allocated."
  "Now test writing each byte...\n", size);


 for(size_t i=0; i< size; ++i)
  p[i] = i%256;

 std::cout<<"Program finished normally!"<<std::endl;
 return 0;
}
```

Output:

```
9190172569 bytes allocated ok.
10109189825 bytes allocated ok.
11120108807 bytes allocated ok.
12232119687 bytes allocated ok.
13455331655 bytes allocated ok.
14800864820 bytes allocated ok.
16280951302 bytes allocated ok.
17909046432 bytes allocated ok.
Killed
```

Subject: Re: How to respond when memory is exceeded
Posted by koldo on Mon, 18 Nov 2024 18:36:07 GMT

View Forum Message <> Reply to Message

mirek wrote on Mon, 18 November 2024 12:31koldo wrote on Mon, 18 November 2024 10:09Hello Mirek

This is a very simple test code to force the problem:

```
GUI_APP_MAIN
{
 WithMain<TopWindow> dlg;

 CtrlLayout(dlg, "Memory test");

 dlg.butTest.WhenAction = []{
  try {
   Buffer<int> data;
   data.Alloc(123456789012, 0);

   PromptOK("Memory is alloc");
  } catch (const std::bad_alloc& e) {
         Exclamation("Caught bad_alloc");
  }
 };

 dlg.Execute();
}
```

And this is the change in U++:

```
void OutOfMemoryPanic(size_t size)
{
 throw std::bad_alloc();

 /*char h[200];
 snprintf(h, 200, "Out of memory!\nU++ allocated memory: %d KB", MemoryUsedKb());
 Panic(h);*/
}
```

When pushing the button 5 times, no problem happens. Visually monitoring the Windows Task Manager, MemoryTest.exe seems to use the same memory all the time.
However, pushing the button one more time, MemoryLimitKb breached! Panic appears. It happens the same in debug and in release.

That is "buggy code" example. For something more real, try

```
#include <CtrlLib/CtrlLib.h>

using namespace Upp;

GUI_APP_MAIN
{
 TopWindow win;
 win.Title("Testing");
 win.WhenClose << [] {
  try {
   for(;;) {
    byte *data = new byte[1024 * 4096];
    for(int i = 0; i < 1024; i++)
     data[i * 4096] = 1;
   }
  } catch (const std::bad_alloc&) {
   Exclamation("Caught bad_alloc");
  }
 };
 win.Run();
}
```

And try that in Linux...

That said, whatever, I think this really is totally futile, but I am going to change the code to actually throw bad_alloc, then catch it in main and do Panic there. I am NOT going to change the U++ code to work correctly in case bad_alloc is thrown, but maybe I will be able to compile some rules (like everything in try block has to be destructed). What a waste of time...
Hi Mirek

I think that the for with allocs is really an example of bad code.
Imagine that you are doing a new Notepad++, and the user wants to open a file of, let's say, XXX Tb.
Two options:

a) Your Notepad++ would say "File too big", and will follow working normally.
b) Exclamation() dialog, and program closes

I prefer option a)

Subject: Re: How to respond when memory is exceeded
Posted by mirek on Mon, 18 Nov 2024 18:48:20 GMT

koldo wrote on Mon, 18 November 2024 19:36mirek wrote on Mon, 18 November 2024 12:31koldo wrote on Mon, 18 November 2024 10:09Hello Mirek

This is a very simple test code to force the problem:
```
GUI_APP_MAIN
{
 WithMain<TopWindow> dlg;

 CtrlLayout(dlg, "Memory test");

 dlg.butTest.WhenAction = []{
  try {
   Buffer<int> data;
   data.Alloc(123456789012, 0);

   PromptOK("Memory is alloc");
  } catch (const std::bad_alloc& e) {
	  Exclamation("Caught bad_alloc");
  }
 };

 dlg.Execute();
}
```

And this is the change in U++:
```
void OutOfMemoryPanic(size_t size)
{
 throw std::bad_alloc();

 /*char h[200];
 snprintf(h, 200, "Out of memory!\nU++ allocated memory: %d KB", MemoryUsedKb());
 Panic(h);*/
}
```
When pushing the button 5 times, no problem happens. Visually monitoring the Windows Task Manager, MemoryTest.exe seems to use the same memory all the time.
However, pushing the button one more time, MemoryLimitKb breached! Panic appears. It happens the same in debug and in release.

That is "buggy code" example. For something more real, try

```
#include <CtrlLib/CtrlLib.h>

using namespace Upp;

GUI_APP_MAIN
```

```
{
 TopWindow win;
 win.Title("Testing");
 win.WhenClose << [] {
  try {
   for(;;) {
    byte *data = new byte[1024 * 4096];
    for(int i = 0; i < 1024; i++)
     data[i * 4096] = 1;
   }
  } catch (const std::bad_alloc&) {
   Exclamation("Caught bad_alloc");
  }
 };
 win.Run();
}
```

And try that in Linux...

That said, whatever, I think this really is totally futile, but I am going to change the code to actually throw bad_alloc, then catch it in main and do Panic there. I am NOT going to change the U++ code to work correctly in case bad_alloc is thrown, but maybe I will be able to compile some rules (like everything in try block has to be destructed). What a waste of time...
Hi Mirek

I think that the for with allocs is really an example of bad code.
Imagine that you are doing a new Notepad++, and the user wants to open a file of, let's say, XXX Tb.
Two options:

a) Your Notepad++ would say "File too big", and will follow working normally.
b) Exclamation() dialog, and program closes

I prefer option a)

c) theide just views the file in read-only mode.

I prefer option c)

But really, notepad will need to allocate individual lines while loading, so the allocation pattern will look just like my example. Not like yours. Billions of small allocations until the memory is exhausted.

Mirek

Subject: Re: How to respond when memory is exceeded
Posted by koldo on Mon, 18 Nov 2024 19:01:36 GMT
View Forum Message <> Reply to Message

Hi Mirek

Notepad++ was just a simple example easy to understand.
The real application is far more complex.
Please don't think this is a waste of time.

Subject: Re: How to respond when memory is exceeded
Posted by mirek on Tue, 19 Nov 2024 11:14:42 GMT
View Forum Message <> Reply to Message

So it is now implemented in "bad_alloc" branch. Basically, memory panic now throws bad_alloc which then caught in APP_MAIN to do normal Panic, so you can try and catch it.

Note that U++ code reaction to bad_alloc right now is

a) uncertain
b) likely and the ideal I will be striving upon, it is only safe to completely destruct U++ structures created in bad_alloc try block, there is no guarantee that any other operation works. Example:

```
VectorMap<String, String> map;
try {
   map.Add("Hi", "there");
}
catch(std::bad_alloc) {}
```

is bad code. Instead you can use

```
VectorMap<String, String> map;
try {
   VectorMap<String, String> map2;
   map2.Add("Hi", "there");
   map = pick(map);
}
catch(std::bad_alloc) {}
```