Subject: CTRL + C = 659 Heap leaks
Posted by Xemuth on Wed, 10 Jul 2019 19:05:31 GMT
View Forum Message <> Reply to Message

Hello,

When I press Ctrl + C on my programme, I got this kind of thing in the Log :

```
Heap leaks detected:

--memory-breakpoint__ 33474 : Memory at 0x09110dd0, size 0xC = 12
   +0 0x09110DD0 10 13 1A 09 00 72 65 65 46 72 65 65             .....reeFree

--memory-breakpoint__ 33473 : Memory at 0x091a1310, size 0x2C = 44
   +0 0x091A1310 7C 12 BA 00 01 00 00 00 38 EF CC 03 B4 2E 40 00     |.......8.....@.
  +16 0x091A1320 00 00 00 00 22 A1 00 00 EC EF CC 03 46 72 65 65     ....".......Free
  +32 0x091A1330 46 72 65 65 46 72 65 65 46 72 65 65             FreeFreeFree

--memory-breakpoint__ 33472 : Memory at 0x09198f10, size 0x2C = 44
   +0 0x09198F10 74 9D B9 00 01 00 00 00 00 00 00 00 00 00 00 00     t...............
  +16 0x09198F20 00 00 00 00 00 00 00 00 00 00 00 00 18 8F 19 09     ................
  +32 0x09198F30 46 72 65 65 46 72 65 65 46 72 65 65             FreeFreeFree

...

*** TOO MANY LEAKS (659) TO LIST THEM ALL
```

Is it preventable ? or it happen only because I shutdowned the code by using Ctrl + C ?

PS : by doing some coding stuff to stop my programm by code, nothing happen

Best Regard

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Novo on Thu, 11 Jul 2019 04:06:31 GMT
View Forum Message <> Reply to Message

Most likely your code has an error (or errors) in its logic. You are, most likely, manually allocating/deallocating memory, or you store pointers to polymorphic classes, which do not have a virtual destructor, in a container. Something like that.
AFAIK, you can click on memory leaks in TheIDE, and it will show you where this memory was allocated. I can be wrong, I haven't seen memory leaks in my code for very very long time :roll:

One interesting thing about U++ is that you almost never need to call new/delete or malloc/free.
It is like programming in Java/JavaScript/Lua/Python/Ruby ...

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Xemuth on Thu, 11 Jul 2019 07:36:09 GMT
View Forum Message <> Reply to Message

Hello Novo, Thanks for you reply.

Quote:You are, most likely, manually allocating/deallocating memory
Don't think so, I never use New or malloc.

Quote:you store pointers to polymorphic classes
In my code I use lot of Ptr of type "DiscordModule", the mother class of lot of object stored in this kind of thing :
Upp::Vector<DiscordModule*> AllModules;
I thought it come from that but by launching my code without filling that vector still result on the same huge amount of memory leak !

Quote:which do not have a virtual destructor, in a container
I tried to declare destructor -> here is definition of my class :
```
class SmartBotUpp{
 private:
  Upp::Vector<DiscordModule*> AllModules;
  Discord bot;

  Upp::String name="";
  Upp::String token="";

  void Event(ValueMap payload);
  Discord* getBotPtr();
 public:
...
};
```

and here is the destructor
```
SmartBotUpp::~SmartBotUpp(){
AllModules.Clear();
}
```

Quote:AFAIK, you can click on memory leaks in TheIDE, and it will show you where this memory was allocated. I can be wrong, I haven't seen memory leaks in my code for very very long time Rolling Eyes

How you do it ? I get all the memory leak by looking log on theIDE but it's not clickable

Quote:One interesting thing about U++ is that you almost never need to call new/delete or malloc/free. Yeah that's exactly what I'm doing. That's why I'm lost :d

Best regard.

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Oblivion on Thu, 11 Jul 2019 07:57:07 GMT
View Forum Message <> Reply to Message

Hello Xemuth,
Quote:
AllModules.Clear();

As far as I can see, with this code you are simply deleting the pointers to allocated objects. This code doesn't delete the objects pointed, If those objects are allocated with the "new" keyword and not deleded elswhere, then the heap leaks you are getting are the natural result. :)

Best regards,
Oblivion

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Xemuth on Thu, 11 Jul 2019 08:10:10 GMT
View Forum Message <> Reply to Message

yeah but the object pointed by this ptr from AllModules are allocated in stack from the main :

```
CONSOLE_APP_MAIN {
 StdLogSetup(LOG_COUT|LOG_FILE);

 SmartBotUpp mybot(myBotID,myBotToken);

 Discord_Overwatch ow("OverWatch","ow");
 mybot.AddModule(&ow);

 Discord_Minecraft mc("Minecraft","mc");
 mybot.AddModule(&mc);

 mybot.Launch();
}
```

So, if I understand well, it should be destroyed at end of the main. I'm right ? :blush:
Maybe I should rewrite the code and try to store references instead of Ptr's.

Subject: Re: CTRL + C = 659 Heap leaks

Posted by mirek on Thu, 11 Jul 2019 08:22:25 GMT
View Forum Message <> Reply to Message

Well, your method(s) seems fine (although I would probably use static module variables instead of locals in CONSOLE_APP_MAIN).

So, this is console application in POSIX and you are pressing Ctrl+C to terminate it. AFAIK, Ctrl+C just calls exit, which is what is causing the leaks, as 'mybot' never gets destroyed.

Hard to say if there is anything to fix. This behaviour will only exist in Debug mode, in release all will be fine. If this is too annoying, we might consider installing handler that switches leaks testing off...

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Xemuth on Thu, 11 Jul 2019 08:28:19 GMT
View Forum Message <> Reply to Message

In my code I'm using this lib :

https://github.com/BornTactical/DiscordUpp

in this, Atomic is used and even lib simple example, result in lot of memory leaks.
Maybe it's because of "Atomic" ?

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Xemuth on Thu, 11 Jul 2019 08:30:00 GMT
View Forum Message <> Reply to Message

Hello Mirek, It's clear, Thank you. I though my code was allocating data without desalocating it while running

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Novo on Thu, 11 Jul 2019 14:42:29 GMT
View Forum Message <> Reply to Message

mirek wrote on Thu, 11 July 2019 04:22
So, this is console application in POSIX and you are pressing Ctrl+C to terminate it. AFAIK, Ctrl+C just calls exit, which is what is causing the leaks, as 'mybot' never gets destroyed.

Ctrl+C sends SIGINT, which causes "soft" termination of an app (unlike SIGKILL). It does all cleanup job for the process including stack unwinding (at least in x64 Linux).

Proof:
Memory leak report is a result of a call to UPP::MemoryDumpLeaks(), and it is called by

```
MemDiagCls::~MemDiagCls()
{
 if(--sMemDiagInitCount == 0)
  UPP::MemoryDumpLeaks();
}
```

So, stack gets unwinded, but in case of Xemuth memory doesn't get deallocated.
This is a bug with Xemuth's code. :blush:

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Novo on Thu, 11 Jul 2019 15:10:57 GMT
View Forum Message <> Reply to Message

Xemuth wrote on Thu, 11 July 2019 03:36
Quote:AFAIK, you can click on memory leaks in TheIDE, and it will show you where this memory was allocated. I can be wrong, I haven't seen memory leaks in my code for very very long time Rolling Eyes

How you do it ? I get all the memory leak by looking log on theIDE but it's not clickable

I, probably, saw that somewhere else ...

Another approach:
1) compile your code with a .USEMALLOC flag. That will switch to glibc's allocator
2) run valgrind your_app
3) press CTRL-C

you will get detailed information about memory leaks.

There is also Debug --> "Test in Valgring" in TheIde, but I personally always use valgrind from command line ...

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Xemuth on Thu, 11 Jul 2019 20:13:50 GMT
View Forum Message <> Reply to Message

Hello Novo, Thanks for the precision about valgrind.
Sadly I dont have linux atm. have you a windows alternative :d ?

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Novo on Thu, 11 Jul 2019 20:24:30 GMT
View Forum Message <> Reply to Message

Xemuth wrote on Thu, 11 July 2019 16:13Hello Novo, Thanks for the precision about valgrind.
Sadly I dont have linux atm. have you a windows alternative :d ?
Intel Inspector, but it is not free :(
Move to Linux :roll:

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Novo on Thu, 11 Jul 2019 20:31:56 GMT
View Forum Message <> Reply to Message

Xemuth wrote on Thu, 11 July 2019 16:13Hello Novo, Thanks for the precision about valgrind.
Sadly I dont have linux atm. have you a windows alternative :d ?
Yet another tool:
https://clang.llvm.org/docs/AddressSanitizer.html
https://en.wikipedia.org/wiki/AddressSanitizer

Theoretically, it should work with mingw. It definitely works on Linux :roll:

Subject: Re: CTRL + C = 659 Heap leaks
Posted by mirek on Thu, 11 Jul 2019 22:15:22 GMT
View Forum Message <> Reply to Message

Novo wrote on Thu, 11 July 2019 16:42mirek wrote on Thu, 11 July 2019 04:22
So, this is console application in POSIX and you are pressing Ctrl+C to terminate it. AFAIK, Ctrl+C just calls exit, which is what is causing the leaks, as 'mybot' never gets destroyed.

Ctrl+C sends SIGINT, which causes "soft" termination of an app (unlike SIGKILL). It does all cleanup job for the process including stack unwinding (at least in x64 Linux).

Proof:
Memory leak report is a result of a call to UPP::MemoryDumpLeaks(), and it is called by
```
MemDiagCls::~MemDiagCls()
{
 if(--sMemDiagInitCount == 0)
  UPP::MemoryDumpLeaks();
}
```

So, stack gets unwinded, but in case of Xemuth memory doesn't get deallocated.
This is a bug with Xemuth's code. :blush:

I believe you are wrong. I believe Ctrl+C is equivalent of calling exit(). In that case, global variables get destructed, but local variables do not. This is also quite clear for the implementation of signals - they are asynchronouse and thus do not have to use the same stack so no stack unwinding is possible.

BTW, destructing global variables is the mechanism used to activate the leak checker :)

Mirek

Subject: Re: CTRL + C = 659 Heap leaks
Posted by mirek on Thu, 11 Jul 2019 22:18:45 GMT
View Forum Message <> Reply to Message

BTW, about hunting down leaks (even if I believe there are not any here):

In the log you can see:

--memory-breakpoint__ 33474

If the allocation pattern is stable (like the number is always the same in any run), you can resolve the leak by putting this to commandline when running the code - this will cause an exception when the same allocation is done again.

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Novo on Thu, 11 Jul 2019 22:46:23 GMT
View Forum Message <> Reply to Message

mirek wrote on Thu, 11 July 2019 18:15
I believe you are wrong. I believe Ctrl+C is equivalent of calling exit(). In that case, global variables get destructed, but local variables do not. This is also quite clear for the implementation of signals - they are asynchronouse and thus do not have to use the same stack so no stack unwinding is possible.

BTW, destructing global variables is the mechanism used to activate the leak checker :)

Mirek
I saw that MemDiagCls is a global static ...
Let's say only global statics get destroyed, and stack doesn't get unwinded. In such case every time you press CTRL-C you should get memory leaks. But this never happens to my apps ...
BTW, info for stack unwinding is always included into app, especially when you compile it with exception support (this is related to x64 Win and POSIX ABI).

"no stack unwinding is possible" - try to run an app with gdb and press CTRL-C. gdb will stop each thread in your app and will show you call stacks for each thread. Call stack for each thread can we unwinded, IMHO ...

This is my understanding of this process ...

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Xemuth on Fri, 12 Jul 2019 07:28:16 GMT
View Forum Message <> Reply to Message

Quote:Move to Linux Rolling Eyes Yeah it's plannified :d

Quote:https://clang.llvm.org/docs/AddressSanitizer.html
https://en.wikipedia.org/wiki/AddressSanitizer
Will try it, if Mirek proposition don't help me in reliable way
Quote:In the log you can see:

--memory-breakpoint__ 33474

If the allocation pattern is stable (like the number is always the same in any run), you can resolve the leak by putting this to commandline when running the code - this will cause an exception when the same allocation is done again.

By looking from Internet, SIGINT signal send by "Ctrl + C" on windows 10
Quote:SIGINT: This signal interrupts a process immediately.The default action of this signal is to terminate a process gracefully .It can be handled , ignored or caught .It can be sent from a terminal as input characters .this signal is generated when a user presses Ctrl+C.

Seems like you must handle it. So I guess Upp or Std have handler somewhere in the code ? Also it mean the process probably the same on Windows and Linux ?
Furthermore, By looking on SigInt I find a funny picture and I'd share it :d

Subject: Re: CTRL + C = 659 Heap leaks
Posted by mirek on Fri, 12 Jul 2019 07:44:12 GMT
View Forum Message <> Reply to Message

[quote title=Novo wrote on Fri, 12 July 2019 00:46]mirek wrote on Thu, 11 July 2019 18:15

"no stack unwinding is possible" - try to run an app with gdb and press CTRL-C. gdb will stop each thread in your app and will show you call stacks for each thread. Call stack for each thread can we unwinded, IMHO ...

Ctrl+C here goes to gdb, which catches and handles it.... So unrelated.

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

using namespace Upp;

EXITBLOCK {
 RLOG("Exit");
}

CONSOLE_APP_MAIN
{
 StdLogSetup(LOG_COUT|LOG_FILE);
 Buffer<byte> x(1000);
 RLOG("Here");
 for(;;) Sleep(1);
 RLOG("After ctrl+c");
}
```

In win32, if you run this in console and press Ctrl+C:

```
PS C:\xxx> ./Console
Here
Exit


Heap leaks detected:

--memory-breakpoint__ 736 : Memory at 0x0000000006b1d570, size 0x4CC = 1228
    +0 0x0000000006B1D570 46 72 65 65 46 72 65 65 46 72 65 65 46 72 65 65
FreeFreeFreeFree
   +16 0x0000000006B1D580 46 72 65 65 46 72 65 65 46 72 65 65 46 72 65 65
FreeFreeFreeFree
   +32 0x0000000006B1D590 46 72 65 65 46 72 65 65 46 72 65 65 46 72 65 65
FreeFreeFreeFree
   +48 0x0000000006B1D5A0 46 72 65 65 46 72 65 65 46 72 65 65 46 72 65 65
FreeFreeFreeFree
PS C:\xxx>
```

Subject: Re: CTRL + C = 659 Heap leaks
Posted by mirek on Fri, 12 Jul 2019 07:55:27 GMT

View Forum Message <> Reply to Message

Interestingly, in Linux, process is terminated without any cleanup:

```
cxl@Linux:~$ ./CtrlC
Here
^C
cxl@Linux:~$
```

so no leaks reported...

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Novo on Mon, 15 Jul 2019 04:15:32 GMT
View Forum Message <> Reply to Message

mirek wrote on Fri, 12 July 2019 03:44
In win32, if you run this in console and press Ctrl+C:

```
PS C:\xxx> ./Console
Here
Exit


Heap leaks detected:

--memory-breakpoint__ 736 : Memory at 0x0000000006b1d570, size 0x4CC = 1228
    +0 0x0000000006B1D570 46 72 65 65 46 72 65 65 46 72 65 65 46 72 65 65
FreeFreeFreeFree
   +16 0x0000000006B1D580 46 72 65 65 46 72 65 65 46 72 65 65 46 72 65 65
FreeFreeFreeFree
   +32 0x0000000006B1D590 46 72 65 65 46 72 65 65 46 72 65 65 46 72 65 65
FreeFreeFreeFree
   +48 0x0000000006B1D5A0 46 72 65 65 46 72 65 65 46 72 65 65 46 72 65 65
FreeFreeFreeFree
PS C:\xxx>
```

This explains everything.
On Windows stack doesn't get unwinded, but descructors of global objects are still called. This is not a correct behavior.
In the old 32-bit Win ABI CRT was responsible for calling descructors of global objects. I do not know how this is implemented in the x64 ABI, most likely the same way.

When I strace my app in Linux I do see an exit_group(0) call on normal run (no CTRL-C).
I do not see any system calls when I press CTRL-C ...
This is an interesting topic ...
I guess, SIGINT is handled completely by glibc in Linux ...
Sending SIGINT directly to a process from another shell doesn't change anything ...

Subject: Re: CTRL + C = 659 Heap leaks
Posted by mirek on Mon, 15 Jul 2019 06:17:42 GMT

View Forum Message <> Reply to Message

Novo wrote on Mon, 15 July 2019 06:15
This is not a correct behavior.

There is no correct behaviour. This is not defined anywhere in c++ standard.

Mirek

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Novo on Mon, 15 Jul 2019 15:16:29 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 15 July 2019 02:17Novo wrote on Mon, 15 July 2019 06:15
This is not a correct behavior.

There is no correct behaviour. This is not defined anywhere in c++ standard.

Mirek

In this case this is a bug with U++. MemDiagCls shouldn't be a global object. It should be created on stack (by GUI_APP_MAIN or by CONSOLE_APP_MAIN). IMHO.

Subject: Re: CTRL + C = 659 Heap leaks
Posted by mirek on Mon, 15 Jul 2019 17:35:07 GMT
View Forum Message <> Reply to Message

Novo wrote on Mon, 15 July 2019 17:16mirek wrote on Mon, 15 July 2019 02:17Novo wrote on Mon, 15 July 2019 06:15
This is not a correct behavior.

There is no correct behaviour. This is not defined anywhere in c++ standard.

Mirek

In this case this is a bug with U++. MemDiagCls shouldn't be a global object. It should be created on stack (by GUI_APP_MAIN or by CONSOLE_APP_MAIN). IMHO.

Then it would not work. Think about all that memory allocated in global objects... (and worse, not in initialization phase).

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Novo on Mon, 15 Jul 2019 18:18:00 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 15 July 2019 13:35Novo wrote on Mon, 15 July 2019 17:16mirek wrote on Mon, 15 July 2019 02:17Novo wrote on Mon, 15 July 2019 06:15
This is not a correct behavior.

There is no correct behaviour. This is not defined anywhere in c++ standard.

Mirek

In this case this is a bug with U++. MemDiagCls shouldn't be a global object. It should be created on stack (by GUI_APP_MAIN or by CONSOLE_APP_MAIN). IMHO.

Then it would not work. Think about all that memory allocated in global objects... (and worse, not in initialization phase).

Instead of MemDiagCls you can put on stack a guard-object, which will detect that its destructor got called and set a flag on MemDiagCls.
IMHO, the problem is fixable ...

Subject: Re: CTRL + C = 659 Heap leaks
Posted by mirek on Mon, 15 Jul 2019 22:11:40 GMT
View Forum Message <> Reply to Message

Novo wrote on Mon, 15 July 2019 20:18mirek wrote on Mon, 15 July 2019 13:35Novo wrote on Mon, 15 July 2019 17:16mirek wrote on Mon, 15 July 2019 02:17Novo wrote on Mon, 15 July 2019 06:15
This is not a correct behavior.

There is no correct behaviour. This is not defined anywhere in c++ standard.

Mirek

In this case this is a bug with U++. MemDiagCls shouldn't be a global object. It should be created on stack (by GUI_APP_MAIN or by CONSOLE_APP_MAIN). IMHO.

Then it would not work. Think about all that memory allocated in global objects... (and worse, not in initialization phase).

Instead of MemDiagCls you can put on stack a guard-object, which will detect that its destructor

got called and set a flag on MemDiagCls.
IMHO, the problem is fixable ...

Yeah, but then I will not catch leaks in global objects.

I could install Ctrl+C handler and do the same trick (disable leaks checker), but is it worth the effort?

Mirek

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Novo on Tue, 16 Jul 2019 00:11:55 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 15 July 2019 18:11
Yeah, but then I will not catch leaks in global objects.

I do not understand why. Just do not show memory leak report when an object on stack wasn't destroyed.

Subject: Re: CTRL + C = 659 Heap leaks
Posted by mirek on Tue, 16 Jul 2019 07:38:07 GMT
View Forum Message <> Reply to Message

Novo wrote on Tue, 16 July 2019 02:11mirek wrote on Mon, 15 July 2019 18:11
Yeah, but then I will not catch leaks in global objects.

I do not understand why. Just do not show memory leak report when an object on stack wasn't destroyed.

Maybe we are nor speaking about the same thing...

```
struct Foo {
   byte *ptr;

   ~Foo() { if(ptr) delete[] ptr; } // this might be missing
};

Foo foo;

CONSOLE_APP_MAIN {
   foo.ptr = new byte[1000];
}
```

If you couple leaks detection to stack, either that leak is wrongly ignored or not leak wrongly reported (depending on what exactly you do).

Really the only sensible thing to do is either somehow make Ctrl+C call Exit by installing exception handler in Win32, but that is hard as the handler is running in different thread, or make it disable leak checks completely in Win32 in Ctrl+C exception handler.

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Novo on Wed, 17 Jul 2019 16:44:24 GMT

View Forum Message <> Reply to Message

mirek wrote on Tue, 16 July 2019 03:38
Maybe we are nor speaking about the same thing...

What I mean is

```
struct MemDiagCls {
 MemDiagCls();
 ~MemDiagCls();

 static void ShowReport() { show = true; }
 static bool IsShowReport() { return show; }

private:
 static bool show;
};
bool MemDiagCls::show = false;

MemDiagCls::~MemDiagCls()
{
 if(--sMemDiagInitCount == 0)
  if (IsShowReport())
   UPP::MemoryDumpLeaks();
}

struct LeakRepGuard {
 ~LeakRepGuard();
};

LeakRepGuard::~LeakRepGuard() {
 MemDiagCls::ShowReport();
}

#define CONSOLE_APP_MAIN \
void ConsoleMainFn_(); \
 \
int main(int argc, char *argv[]) { \
 UPP::LeakRepGuard __; \
 UPP::AppInit__(argc, (const char **)argv); \
 UPP::AppExecute__(ConsoleMainFn_); \
 UPP::AppExit__(); \
 return UPP::GetExitCode(); \
} \
 \
void ConsoleMainFn_()
```

A call to UPP::MemoryDumpLeaks() will be disabled if there was no stack unwinding. In all other cases it will work as before.
Leak report makes no sense when destructors of objects on stack were not called.
IMHO, this should be thread-safe because there is only one main() function ...

Subject: Re: CTRL + C = 659 Heap leaks
Posted by mirek on Wed, 17 Jul 2019 18:25:13 GMT
View Forum Message <> Reply to Message

Novo wrote on Wed, 17 July 2019 18:44mirek wrote on Tue, 16 July 2019 03:38
Maybe we are nor speaking about the same thing...

What I mean is

```
struct MemDiagCls {
 MemDiagCls();
 ~MemDiagCls();

 static void ShowReport() { show = true; }
 static bool IsShowReport() { return show; }

private:
 static bool show;
};
bool MemDiagCls::show = false;

MemDiagCls::~MemDiagCls()
{
 if(--sMemDiagInitCount == 0)
  if (IsShowReport())
   UPP::MemoryDumpLeaks();
}

struct LeakRepGuard {
 ~LeakRepGuard();
};
```

```
LeakRepGuard::~LeakRepGuard() {
 MemDiagCls::ShowReport();
}

#define CONSOLE_APP_MAIN \
void ConsoleMainFn_(); \
 \
int main(int argc, char *argv[]) { \
 UPP::LeakRepGuard __; \
 UPP::AppInit__(argc, (const char **)argv); \
 UPP::AppExecute__(ConsoleMainFn_); \
 UPP::AppExit__(); \
 return UPP::GetExitCode(); \
} \
 \
void ConsoleMainFn_()
```

A call to UPP::MemoryDumpLeaks() will be disabled if there was no stack unwinding. In all other cases it will work as before.
Leak report makes no sense when destructors of objects on stack were not called.
IMHO, this should be thread-safe because there is only one main() function ...

Ah, I see, in other words: "or make it disable leak checks completely in Win32 in Ctrl+C exception handler." (your method is more generic but the effect is the same).

Subject: Re: CTRL + C = 659 Heap leaks
Posted by mirek on Wed, 17 Jul 2019 21:04:35 GMT
View Forum Message <> Reply to Message

Novo wrote on Wed, 17 July 2019 18:44
Leak report makes no sense when destructors of objects on stack were not called.

This is debatable. It would warn you about calling "exit(0)" or equivalent, which is IMO good.

Subject: Re: CTRL + C = 659 Heap leaks
Posted by Novo on Wed, 17 Jul 2019 21:46:30 GMT
View Forum Message <> Reply to Message

mirek wrote on Wed, 17 July 2019 17:04Novo wrote on Wed, 17 July 2019 18:44
Leak report makes no sense when destructors of objects on stack were not called.

This is debatable. It would warn you about calling "exit(0)" or equivalent, which is IMO good.

This leak report is very confusing. This thread is a proof of that.
You can do something like this instead ...

```
MemDiagCls::~MemDiagCls()
{
 if(--sMemDiagInitCount == 0)
  if (IsShowReport())
   UPP::MemoryDumpLeaks();
  else
   UPP::Cerr() << "The World is going to end soon!" << UPP::EOL;
}
```

Subject: Re: CTRL + C = 659 Heap leaks
Posted by mirek on Tue, 23 Jul 2019 07:36:02 GMT

View Forum Message <> Reply to Message

OK:

```
* c:\xxx\Console.exe 23.07.2019 09:35:21, user: cxl

Here
Exit
Application was terminated in a non-standard way (e.g. exit(x) call or Ctrl+C)
```
