Base Language - Bug #2657

appserver agent context reset does not deallocate currently allocated memptr buffers

08/24/2015 11:36 AM - Greg Shah

Status:	Closed	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Ovidiu Maxiniuc	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:	Cleanup and Stablization for Server Features		
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

Related issues:

Related to User Interface - Bug #2662: memory leak in the native NCURSES client	**Closed**		
Related to Base Language - Feature #1635: implement MEMPTR/RAW support	**Closed**	**01/19/2013**	**05/03/2013**

History

#1 - 08/24/2015 11:55 AM - Greg Shah

In the 4GL, the memptr implementation is backed by virtual memory allocated by the operating system kernel. Each such allocated block is represented as a separate memptr instance which has an address that is the actual virtual memory address of the start of the allocated block. All 4GL programs that use memptr are supposed to deallocate those memptr instances (using code like SET-SIZE(ptr) = 0.) before ending. If they do not deallocate the buffers, then this is a memory leak if the 4GL process outlives the end of the allocating procedure.

When a process exits, all such buffers are automatically cleaned up by the operating system, since any memory that is allocated for that process is released.

The way that we implement appserver agents, we leave the process running and just reset the context (P2J state). There is very little state stored on the client side. Most state is on the server and is easily reset there. Our memptr implementation does not currently deallocate any still-allocated memptr instances when the context is reset. We need to implement cleanup processing that will deallocate these, otherwise this is an ever growing, never shrinking pool of dangling memptr instances that can never be fixed until the appserver agent process exits.

Presumably, Progress does this automatically during appserver agent state reset. This has no functional benefit other than the natural benefit of using less memory. As such there are no compatibility concerns here. We can implement it however we like.

The plan:

1. Modify MemoryDaemon.allocate() to store each successfully allocated address in a list.
2. Modify MemoryDaemon.deallocate() to remove any successfully deallocated address from that list.
3. Add a cleanup() method to LowLevelBuffer.
4. Implement MemoryDaemon.cleanup() which will iterate through the list of allocated addresses and calls MemoryManager.deallocate() on each one. Then clear the list.
5. Add a cleanup() method override to the memptr.ctxt which will call wa.bufmgr.cleanup().

Constantin: is item 5 the best way to trigger this notification?

#2 - 08/24/2015 12:01 PM - Constantin Asofiei

Greg Shah wrote:

> In the 4GL, the memptr implementation is backed by virtual memory allocated by the operating system kernel. Each such allocated block is represented as a separate memptr instance which has an address that is the actual virtual memory address of the start of the allocated block. All 4GL programs that use memptr are supposed to deallocate those memptr instances (using code like SET-SIZE(ptr) = 0.) before ending. If they do not deallocate the buffers, then this is a memory leak if the 4GL process outlives the end of the allocating procedure.
>
> When a process exits, all such buffers are automatically cleaned up by the operating system, since any memory that is allocated for that process is released.
>
> The way that we implement appserver agents, we leave the process running and just reset the context (P2J state). There is very little state stored on the client side. Most state is on the server and is easily reset there. Our memptr implementation does not currently deallocate any still-allocated memptr instances when the context is reset. We need to implement cleanup processing that will deallocate these, otherwise this is an ever growing, never shrinking pool of dangling memptr instances that can never be fixed until the appserver agent process exits.
>
> Presumably, Progress does this automatically during appserver agent state reset. This has no functional benefit other than the natural benefit of using less memory. As such there are no compatibility concerns here. We can implement it however we like.
>
> The plan:
>
> 1. Modify MemoryDaemon.allocate() to store each successfully allocated address in a list.
> 2. Modify MemoryDaemon.deallocate() to remove any successfully deallocated address from that list.
> 3. Add a cleanup() method to LowLevelBuffer.
> 4. Implement MemoryDaemon.cleanup() which will iterate through the list of allocated addresses and calls MemoryManager.deallocate() on each one. Then clear the list.
> 5. Add a cleanup() method override to the memptr.ctxt which will call wa.bufmgr.cleanup().
>
> Constantin: is item 5 the best way to trigger this notification?

Yes, when the Agent resets the context, it will call cleanup() for all ContextLocal instances being reset.

#3 - 08/30/2015 01:14 PM - Eric Faulhaber

- Assignee set to Eric Faulhaber

- % Done changed from 0 to 50

- Status changed from New to WIP

This change was implemented in task branch 2662a (see #2662) and was merged into trunk revision 10930.

#4 - 08/30/2015 01:15 PM - Eric Faulhaber

From Constantin via email:

> The remaining concerns about code which allocates memory (instead of copying the pointer) are these:
>
> 1. memptr.duplicate() - this is called all throughout the code. Some cases - like ArrayAssigner and AccessorWrapper.get() - might be problematic when working with memptrs. Also, this is called from memptr.deepCopy() - which is executed IIRC when vars are backed up in a transaction block, for undo processing. This again might need to just copy the pointer, and not allocate new memory; but testing is needed to confirm.
> 2. memptr() c'tor
>
> - in the call from XMLCallbackWrapper.resolveEntity:301, it is used as an OUTPUT parameter for the "ResolveEntity" callback. In this case, once work is done with it, I think it should be released - again, testing is needed to confirm this.
> - in XMLCallbackWrapper c'tor, is used as an INPUT parameter for the "Characters" and "IgnorableWhitespace" callbacks. I don't know what the behaviour should be in this case.

#5 - 03/30/2016 12:54 PM - Eric Faulhaber

- Assignee deleted (Eric Faulhaber)

#6 - 04/16/2016 01:30 PM - Eric Faulhaber

- Assignee set to Constantin Asofiei

Re-assigning this so we can address the remaining concerns in note 4.

#7 - 07/06/2016 03:43 PM - Ovidiu Maxiniuc

- Assignee changed from Constantin Asofiei to Ovidiu Maxiniuc

Task branch 2657b was created from trunk 11066.
Latest revision (11067) contains:

- memptr.duplicate() copies buffer 'pointer' instead of deep copy;
- in LowLevelBuffer & MemoryDaemon incrementReference() method was added and deallocate() returns true if successful to support reference counter;
- removed memptr#REAL_LOWER_BOUND. Apparently there is no lower limit. From my tests memptr can be initialized with 1 byte if needed and negative sizes cannot be observed (see uast/memptr/memptr_sizes.p in testcases);
- condition 12118 was deactivated as consequence;
- fix in ArrayAssigner.subscriptWorker() method. If multiple elements were to be displayed and the first encountered a recoverable error, the subsequent extent derefereces were incorrectly failing;
- added static check() method in RangeCheck (minor performance improvement);
- memptr.toStringMessage() method was fixed to reflect result from 4GL.

There are a few more TODOs left behind that I will address tomorrow. I also have a short list of test that on P2J are not matching 4GL. The strangest is the fact that uninitialized memptrs sent as parameters are at the same time unknown and not unknown values :O.

#8 - 07/06/2016 06:38 PM - Greg Shah

I haven't gotten a chance to review your changes yet. But I did want to mention that I implemented the negative buffer size behavior for specific reasons. Please see:

memptr_initialized_with_big_enough_negative_size.p
memptr_initialized_with_small_negative_size.p
memptr_negative_size.p
memptr_set_size_negative_after_set_pointer.p

#9 - 07/07/2016 10:57 AM - Ovidiu Maxiniuc

Greg,
It looks like those testcases are deprecated. Apparently they were written for an older version of OpenEdge.
I tried the 9.1D but they also failed to compile (lack of support for int64 and longchar). Indeed, on some conditions, I was able to write some testcase that indeed displayed ANY negative size with get-size(m). The same code on OE 10.2B it displayed the unsigned value (Ex: 4294966296 for -1000).

#10 - 07/07/2016 11:04 AM - Greg Shah

> It looks like those testcases are deprecated. Apparently they were written for an older version of OpenEdge.
> I tried the 9.1D but they also failed to compile (lack of support for int64 and longchar). Indeed, on some conditions, I was able to write some testcase that indeed displayed ANY negative size with get-size(m). The same code on OE 10.2B it displayed the unsigned value (Ex: 4294966296 for -1000).

I wrote those testcases using 10.2B and lindev01 to test them. They all worked at the time I wrote them 3 years ago. See #1635.

#11 - 07/07/2016 11:36 AM - Greg Shah

There is no revision 11067 in 2657b on devsrv01.

#12 - 07/07/2016 11:48 AM - Greg Shah

I just retested these on lindev01. They all work as expected, except that I do see BASE64-ENCODE of an large negative sized buffer should cause an error 12118. at the end of memptr_initialized_with_big_enough_negative_size.p.

#13 - 07/07/2016 12:39 PM - Ovidiu Maxiniuc

Greg Shah wrote:

> There is no revision 11067 in 2657b on devsrv01.

I did the commit operation again, this time on the correct branch.

> I just retested these on lindev01. They all work as expected, except that I do see BASE64-ENCODE of an large negative sized buffer should cause an error 12118. at the end of memptr_initialized_with_big_enough_negative_size.p.

I used the windev01 for testing. It looks like the differences we are seeing is because the P4GL implementation is different between Windows and Linux. For example, check uast/memptr/memptr_sizes.p (I committed last night). It reports 0 errors on both 9/10 OE on Windows, but on linux will print 6 errors! I wonder what's the result on Solaris?

#14 - 07/07/2016 04:49 PM - Greg Shah

Code Review Task Branch 2657b Revision 11067

1. The SecurityOps.base64Encode() changes remove code that I believe is needed for some cases. I suspect this may be a windows vs linux scenario again.

2. In regard to the comment in setPointerValue() regarding should this instance be also tracked as reference?, the answer is NO. This method is used when the pointer is obtained from some other source. There are no guarantees that the pointer is valid and there is no obligation to deallocate it. One common case is to call an operating system API or DLL/SO library and then use a returned pointer via this mechanism. If the user were to deallocate that pointer, at best it would fail (because it was allocated using a heap or OS allocation method that is separate from the P2J C runtime code) and at worst it might cause a segmentation fault for the process. In regard to the what happens to the size? question, the size is maintained independently and can be set from the calling converted 4GL code. We already handle that behavior the same way as the 4GL. The size in such a case must be set independently and it is used without any checking. In fact, there is no way for the size to be determined OR validated by P2J. Modern OS platforms are not guaranteed to provide the mechanism for this to normal applications.

#15 - 07/11/2016 05:59 PM - Ovidiu Maxiniuc

Task Branch 2657b updated. Last revision revision 11068.

I identified the cause of the differences we see because of the different builds of Progress OE 10.2B:

- On windev01 the system is running an windows version. It seems the most stable when relatively to accessing native memory (you cannot allocate negative memory or, if you succeed, all access is restricted)
- On lincon01, the system is running an x86-64 (ELF 64-bit LSB executable image). I guess this is our main target. Returned pointers are 64-bit large as expected. It allows any positive or negative size to be allocated with set-size.
- On lindev01, the file reports an ELF 32-bit LSB executable, Intel 80386 binary. Because of this architecture constraints, the memptr support is clumsy. Accessing negative allocated memory usually abends the process with memory map/dumps. I was able to duplicate some of your tests with lindev01. Particularly the -7 bound you identified when implementing memptr. I think emulating this is not in out targets so I simplified the code by removing this code.

> 1. The SecurityOps.base64Encode() changes remove code that I believe is needed for some cases. I suspect this may be a windows vs linux scenario again.

I tried to identify such case but I could not. Not even on the 32bit version from lindev01. I left the message in the javadoc in case that we will be able to identify such conditions. I suspect that this may occur when the memory is really limited but this is not strictly related to this particular function.

> 2. In regard to the comment in setPointerValue() regarding should this instance be also tracked as reference?, the answer is NO. This method is used when the pointer is obtained from some other source. There are no guarantees that the pointer is valid and there is no obligation to deallocate it. One common case is to call an operating system API or DLL/SO library and then use a returned pointer via this mechanism. If the user were to deallocate that pointer, at best it would fail (because it was allocated using a heap or OS allocation method that is separate from the P2J C runtime code) and at worst it might cause a segmentation fault for the process. In regard to the what happens to the size? question, the size is maintained independently and can be set from the calling converted 4GL code. We already handle that behavior the same way as the 4GL. The size in such a case must be set independently and it is used without any checking. In fact, there is no way for the size to be determined OR validated by P2J. Modern OS platforms are not guaranteed to provide the mechanism for this to normal applications.

I did some tests with simple app-server to test the automatic cleanup at the end of a session. I saw no mem-leaks, all allocated memory is freed at session reset on the agents. All access to duplicate() and from XML are cleaned up.

#16 - 07/12/2016 11:11 AM - Greg Shah

Code Review Task Branch 2657b Revision 11068

I'm OK with the changes, but I've checked in some additional comments to make things more clear. Please review the comments.

#17 - 07/12/2016 12:12 PM - Ovidiu Maxiniuc

I totally agree with all your changes.
I am going to put this update to full runtime test. Conversion test is not necessary for changes in this branch.

#18 - 07/15/2016 12:54 PM - Greg Shah

How is testing coming along?

#19 - 07/15/2016 01:03 PM - Ovidiu Maxiniuc

Greg Shah wrote:

> How is testing coming along?

The update PASSED the regression testing on devsrv01. It also had no failures/errors in allSearchTests and <app_name>Test on ETF. Probably within the hour I will have the result for batchProgramTest. I did not reconvert the server so the keywordTest is irrelevant on my workstation.

If the current ETF test is fine (batch), I intend to merge the update into trunk.

LE: GES removed the customer app name from this entry.

#20 - 07/15/2016 02:16 PM - Greg Shah

> If the current ETF test is fine (batch), I intend to merge the update into trunk.

Perfect!

#21 - 07/15/2016 02:21 PM - Ovidiu Maxiniuc

As noted in #3109 note 87 the batchProgramTest finished flawlessly. I am rebasing now to 11069 and commit to trunk.

#22 - 07/15/2016 02:54 PM - Ovidiu Maxiniuc

2657b was rebased and committed to trunk as revision 11070. Notification email was sent to team. The test results on devsrv01 and the branch were archived.

#23 - 07/15/2016 04:10 PM - Greg Shah

- % Done changed from 50 to 100

#24 - 07/21/2016 01:22 PM - Greg Shah

- Status changed from WIP to Closed

#25 - 08/09/2016 12:10 PM - Constantin Asofiei

Ovidiu Maxiniuc wrote:

> The strangest is the fact that uninitialized memptrs sent as parameters are at the same time unknown and not unknown values :O.

These are documented in #2375.

#26 - 11/16/2016 12:06 PM - Greg Shah

- Target version changed from Milestone 11 to Cleanup and Stablization for Server Features