MARTe2-core - Design Improvements #167

Memory management

13.07.2015 11:26 - Filippo Sartori

Status:	Closed	**Spent time:**	0.00 hour
Priority:	Normal		
Assignee:			
Category:			

Description

Main issues:

1) Memory need to support all type of operating systems - those with just a heap - those with multiple heaps - those with swappable memory etc...
2) new/free need to support objects allocated in different memories
3) new/free shall not raise exceptions
4) Shared Memory Support needs to be separated into a module supporting Processes
5) It should be possible to express the need for non-swappable/ fully committed memory !!heap!!
6) It should be possible to describe where objects are allocated in the CDB.
7) It should be possible to test memory addresses for access validity --> support to drivers

Development ideas:

a) MemoryHeap object - properties: NO-SWAP - platform specic memory pools selector
b) RawMemoryHeaps are pre-constructed for each platform so that all memory is made available
c) RawMemoryHeap database is pre-constructed for each platform --> from there univokely instanciate MemoryHeap classes
d) Additional Memory Heaps can be built on virtual memory systems using special properties like NO-SWAP etc...
e) new/delete are private -- objects created only using factories where memory heap can be specified.
f) Object should provide an API to allow retrieving the heap that was used to constract the object itself.--> so that the object internals can be built on the same heap.
g) Standard Object Factory methods shall be provided as part of Object: ObjectReference Create(Heap&) ObjectReference.Destroy() ObjectReference CreateConfigured(configuration&)
h) MemoryHeaps are Objects and are contained in the root of the main Object DataBase so that Object Factories can find the MmeoryHeaps to be used during construction

Concepts for a possible solution:

1) Global Memorymanager object consisting of the following fields:
HeapManager *heaps[MAX_N_HEAPS]; -->points to heap objects
struct HeapMemoryRangeEntry{
void * address;
int heapNo;
bool startOfHeap; // false = end of heap
} heapLUT[MAX_N_HEAPS*2]; --> allows identifying to what heap an address belongs to. Supports also nested heaps. Entries are sorted so that addresses are sequential.
2) HeapManager object with virtual public methods Malloc(),Free(),Realloc(),CompactMemory(),...
specialized Heapmanagers with different policies: RTHeapManager {malloc from unused memory;free to stack of freed memory; compactmemory to sort out fragmentation}, MicroHeapManager {malloc 0 overhead --> malloc(1) = 1byte used; free not working, compactmemory not working}
HeapManager contructor options: (memoryStart,memoryEnd) ; (HeapManager&, memorySize) ; (addressSpace, flags={ growable/committed})....
3) free operates by searching the pointer in the LUT above and then calling the corresponding Heap-->free.

ISSUE linux/windows malloc operates both by taking blocks from the heap and by calling mmap/VirtualMalloc. So there is no guarantee of sequentiality in the memory blocks of the standard malloc.
Therefore the free shall search for in the LUT and upon failure assume that it belongs to standard free.

History

#1 - 20.07.2015 12:19 - Filippo Sartori

Concepts for a possible solution:

1) Global Memorymanager object consisting of the following fields:
HeapManager *heaps[MAX_N_HEAPS]; -->points to heap objects
struct HeapMemoryRangeEntry{
void * address;
int heapNo;
bool startOfHeap; // false = end of heap
} heapLUT[MAX_N_HEAPS*2]; --> allows identifying to what heap an address belongs to. Supports also nested heaps. Entries are sorted so that addresses are sequential.
2) HeapManager object with virtual public methods Malloc(),Free(),Realloc(),CompactMemory(),...
specialized Heapmanagers with different policies: RTHeapManager {malloc from unused memory;free to stack of freed memory; compactmemory to sort out fragmentation}, MicroHeapManager {malloc 0 overhead --> malloc(1) = 1byte used; free not working, compactmemory not working}
HeapManager contructor options: (memoryStart,memoryEnd) ; (HeapManager&, memorySize) ; (addressSpace, flags={ growable/committed})....
3) free operates by searching the pointer in the LUT above and then calling the corresponding Heap-->free.

ISSUE linux/windows malloc operates both by taking blocks from the heap and by calling mmap/VirtualMalloc. So there is no guarantee of sequentiality in the memory blocks of the standard malloc.
Therefore the free shall search for in the LUT and upon failure assume that it belongs to standard free.

#2 - 04.08.2015 09:34 - André Neto

- Subject changed from MEMORY MANAGEMENT to Memory management

#3 - 04.08.2015 09:55 - André Neto

- Description updated

#4 - 04.08.2015 09:55 - André Neto

- Description updated

#5 - 28.02.2020 16:27 - André Neto

- Status changed from New to Closed

- Assignee deleted (Filippo Sartori)

- Priority changed from High to Normal