Java Memory Management on HP-UX

Laksh Venkatasubramanian

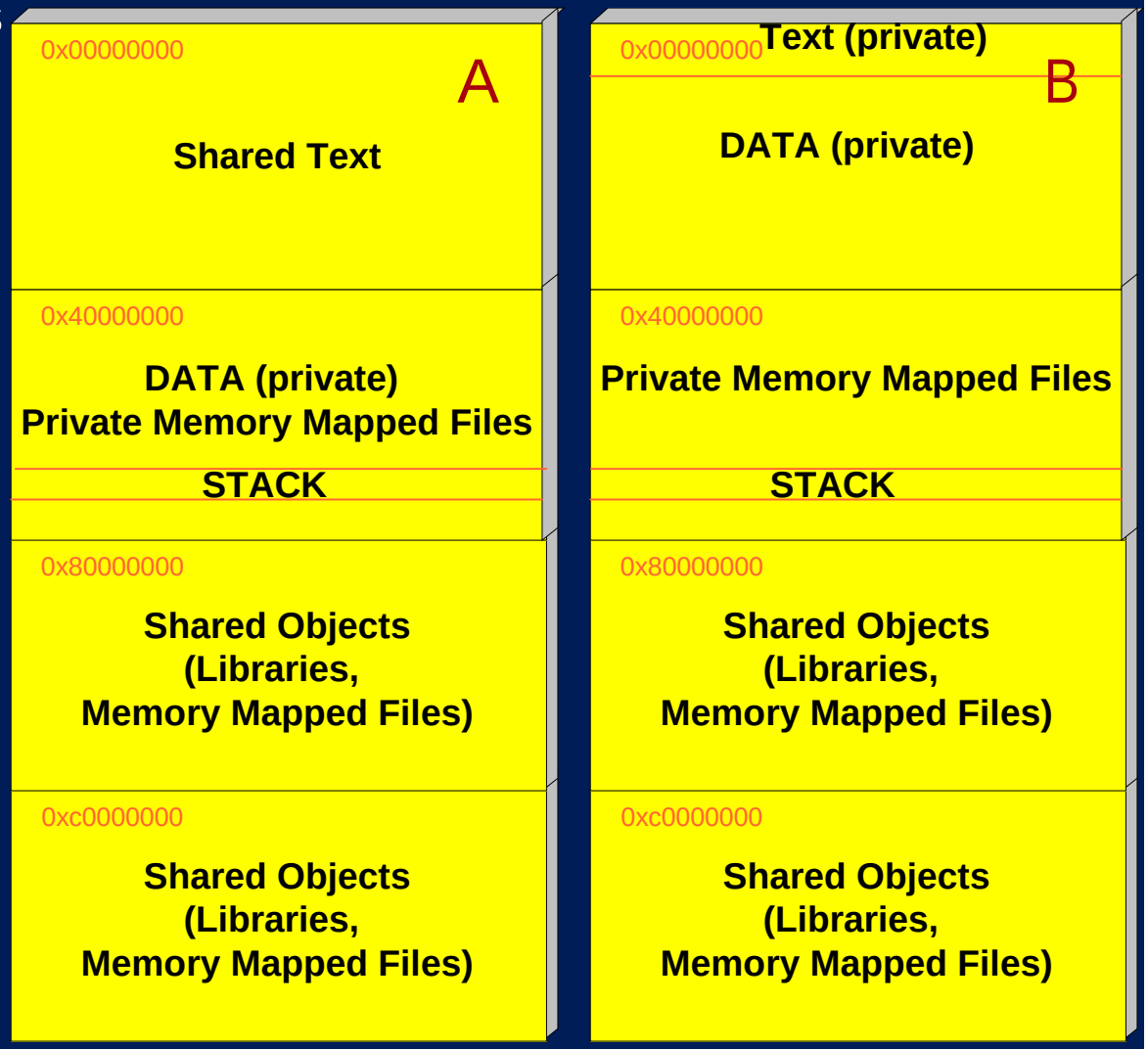
HP Java Labs

HP-UX Virtual Memory Layout

- HP-UX 32 Bit Process
- Four 1 GB Quadrants

A) SHARE_MAGIC

B) EXEC_MAGIC



Type of Executables on HP-UX

There are 3 magic numbers that can be used for a 32-bit executable (11.00 and greater).
/usr/bin/chatr labels the following type of executables in output

- SHARE_MAGIC: shared executable
- EXEC_MAGIC: normal executable
- SHMEM_MAGIC: normal SHMEM_MAGIC executable

For 64 bit (11.00 and greater) executables, there is currently no need to have different magic numbers available as the standard one allows up to 4TB for the program text, another 4TB for its private data and a total of 8TB for shared areas.

Type of Executables on HP-UX

- SHARE_MAGIC is the default on 11.0. SHARE_MAGIC is also called DEMAND_MAGIC. With SHARE_MAGIC, quadrant 1 is used for program text, quadrant 2 is used for program data, and quadrants 3 and 4 are for shared items.

- EXEC_MAGIC allows a greater process data space by allowing text and data to share quadrant 1. Quadrant 2 is still solely used for data, and quadrants 3 and 4 are also the same as with SHARE_MAGIC executables. EXEC_MAGIC applications are created by linking the application with the -N option.

- SHMEM_MAGIC makes 2.75 GB of shared memory available to an application. With SHMEM_MAGIC all of the text and data is in quadrant 1 freeing up quadrant 2 for shared items. The SHMEM_MAGIC processes on the system will share quadrant 2 for shared memory, as well as sharing quadrants 3 and 4 with other processes on the system.

EXEC_MAGIC vs SHARED_MAGIC

	EXEC_MAGIC	Quad 3 Private (q3p)	Quad 4 Private (q4p)	SHARED_MAGIC
Quadrant 1 0x00000000-0x3FFFFFFF	Text starts at the beginning of this space and data starts immediately after the end of the text.	Same	Same	Text only and read only.
Quadrant 2 0x40000000-0x7FFFFFFF	Data and Stack	Same	Same	Data and Stack
Quadrant 3 0x80000000-0xBFFFFFFF	Shared objects	Private Data	Private Data	Shared objects
Quadrant 4 0xC0000000-0xC0000FFF	Kernel gateway page	Same	Same	Kernel gateway page
Quadrant 4 0xC0003000-0xEFFFFFFF	Shared objects	Same	Private Data	Shared objects
Quadrant 4 0xF0000000-0xFFFFFFFF	PDC I/O address space	Same	Same	PDC I/O address space

Enabling 3rd and 4th quadrants for private data

PA-RISC

- ‘chatr +q3p enable <program>’ - an extra 1Gb of private data is made available to a process (Both SHARED_MAGIC and EXEC_MAGIC program can have quadrant 3 and quadrant 4 private). You cannot access shared objects available to other programs in their quadrant 3 when you enable this option.

- ‘chatr +q4p enable <program>’ - this changes quadrants 3 and 4 to be private. You cannot access any shared memory or shared mmap'ed files available to other programs.

Before using q3 or q4 private programs check and see if there are patches that are needed.

ITANIUM (11.23)

- `chatr +as mpas <program>’ will enable all quadrants to be private.

Patches for q3p q4p functionality

HP-UX 11.0 PA-RISC

- Required Patches: PHKL_27282, PHKL_23409, PHKL_28766, PHKL_26136
- 11.0 supports only q3p. It does not support q4p functionality.

HP-UX 11i (11.11) PA-RISC

- Required Patch: PHKL_28428 (or its superseded patch)
- 11.11 supports both q3p and q4p.

HP-UX 11i v1.5 (11.22) ITANIUM

- Does not support q3p, q4p functionality

HP-UX 11i v2 (11.23) ITANIUM

No patches are required

Kernel tunables

- maxdsiz, maxdsiz_64bit

  Controls the size of the DATA region. We can call this the C-heap to differentiate this from the JAVA-heap. sbrk(),malloc(), etc. allocate memory in this region.

- maxssiz, maxssiz_64bit

  Controls the size of the primordial thread (main thread) stack. By default, the JVM restricts the size of this stack to 2MB.

- maxtsiz, maxtsiz_64bit

  Controls the size of the TEXT region. This contains the executable.

Kernel tunables

- Even though maxdsiz may be set to a large value, the actual available heap space (DATA) might be much lower because the memory mapped (mmap) segments that are mapped private, STACK, TEXT (EXEC_MAGIC case), Java heap, Java threads, etc. also share this address space.

- Similarly, even though maxtsiz might be a large value, it consumes only as much physical/virtual memory as the executable requires.

- On the contrary, maxssiz consumes as much virtual space as the value it is set to. In other words , raising maxssiz may cause user processes which use all (or nearly all) of the previously available data area to fail allocation with the [ENOMEM] error, even with maxdsiz set above the current amount of memory allocated for data by this process.

Kernel tunables

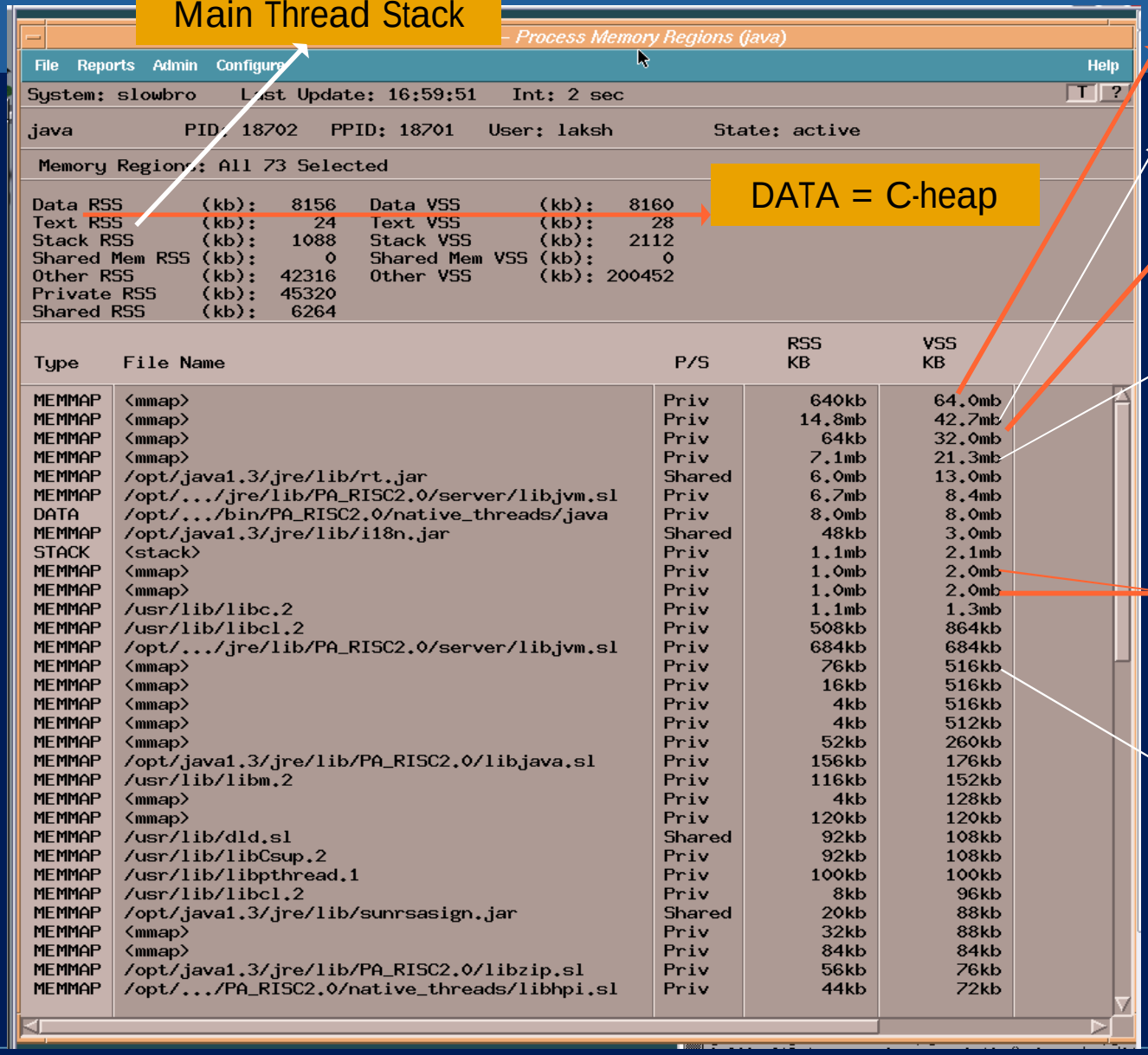
SWAP

- swapinfo –mt (displays swap space usage on the system)

TYPE	Mb AVAIL	Mb USED	Mb FREE	PCT USED	START/ LIMIT	Mb RESERVE	PRI	NAME
dev	4096	0	4096	0%	0	-	1	/dev/vg00/lvol2
reserve	-	266	-266					
memory	4089	1313	2776	32%				
total	8185	1579	6606	19%	-	0	-	

Swap is reserved at the time virtual memory is allocated for a process. But when the lazy-swap option is enabled, swap is allocated at the time of actual use of memory.

Glance Memory Regions (/opt/perf/bin/gpm)



Glance Memory Regions

- RSS (Resident Set Size) - The size (in KB unless otherwise indicated) of the resident memory occupied by a memory region
- VSS (Virtual Set Size) - The size (in KB unless otherwise indicated) of the virtual memory occupied by a memory region

Java Memory Regions

- The JAVA threads are private mmap segments. The default size for this mmap is 512KB(32bit), 1MB(64bit).
- JVM CodeCache (holds compiled JAVA methods) is a private mmap segment. The default size is 32MB.
- The JAVA heap is a private mmap (Use -XheapInitialSizes to determine sizes of different generations) segment. The three regions in HotSpot JVM heap (new, old and permanent) are allocated as three different mmap regions in 32bit mode in 1.3.1 or greater JVMs.

The JAVA heap is mapped MAP_NORESERVE (lazy swap). When multiple processes are spawned, memory and swap have to be estimated carefully, otherwise running processes may abort in the middle of a run due to insufficient swap space, instead of processes aborting at startup time.

java -XheapInitialSizes

Defaults when no options are specified-

```
NewRatio:          3
SurvivorRatio:     8
MaxTenuringThreshold: 32
Survivor size:     589824
Eden size:         5177344
New Size reserved:    22347776   initial:  6356992
Old Size reserved:    44761088   initial: 12779520
Perm Size reserved:   67108864   initial:  1048576
```

New size will default to around 1/3rd the total heap size if –Xmn is not specified. –Xmn is an alias for –XX:NewSize. If this value is higher than MaxNewSize, MaxNewSize will be set to this value as well. New generation will be resized to 1/3rd the total heap as the heap grows from –Xms to –Xmx.

Large Heap Size with 32-bit Java

For Java invoked from the command line, Java will automatically choose an appropriate executable.

PA-RISC

- For heaps less than 1500MB, the executable is ‘java’ (EXEC_MAGIC executable).

- For heaps greater than or equal to 1500MB, and less than 2400MB the executable is ‘java_q3p’ (HP-UX 11.00 or greater).

- For heaps of 2400MB to 3800MB, the executable is ‘java_q4p’ (HP-UX 11.11 or greater).

ITANIUM

- For heaps of 1500MB to 3500MB, the executable is `java_q4p’ (HP-UX 11.23 or greater)

Large Heap Size with 32-bit Java

HP-UX 11.11 (PA-RISC)

- Because of segmentation in the HP-UX virtual address space, when the Java heap is larger than 3000MB, either new space (-Xmn) or old space (-mx minus -Xmn) must be approximately 850MB or less (applicable to 11.11 only).

HP-UX 11.00 or greater

- You do not need to directly invoke any of the q3p or q4p programs. Just invoke 'java' as usual, and the appropriate program will be run for you.

Components in a JAVA program

- Virtual Machine is written in C/C++
- JAVA code
- JAVA code calling native methods
- Native code calling into JAVA code

Memory Allocation

- JAVA heap

  All objects that are created with the ‘new’ keyword in JAVA reside here.
- C heap

  Memory allocated in native code with
  - ‘malloc’ in C
  - ‘new’ in C++

Java Objects

- Necessary to make a distinction between live objects and reachable objects

  Reachable objects- If we can reach an object from the root set through any number of intermediate references, it is termed reachable

  Live objects- These are reachable objects that are currently being used by the program

Java Objects

- When JNI references are not cleaned up properly, they could prevent the collection of some unwanted JAVA objects
- All objects that are reachable may not be live

  - objects that are being referenced by some long living objects. Even though their use in the program is over, they cannot be garbage collected as the long living objects are still alive

Symptoms of Process Memory Growth

- Java Heap Object Retention:
  - Unaccountable growth of the Java Heap
- C Heap Memory Leak:
  - Constantly increasing DATA RSS and VSS
  - System running out of swap space
  - Programs failing with out of memory (ENOMEM) errors

Reasons for Out of Memory Errors

- Virtual address space limitations
- Insufficient java heap
- Low values for kernel parameters

max_thread_proc	Number of threads per process
nkthread	Total number of threads
maxdsiz	Data region size
nfiles	Total number of open files
maxfiles	Soft limit for number of open files per process
maxfiles_lim	hard maximum number of file descriptors per process

Virtual Address Space Usage: Example 1

A) maxtsiz – 1GB (Upper limit for TEXT region)
B) maxdsiz – 1GB (Upper limit for DATA region)
Address space is reserved for TEXT and DATA in incremental amounts as needed.
C) maxssiz – 400MB (Upper limit for STACK region, reserved upfront)
D) Java heap - -Xms1GB –Xmx1GB (Perm gen- 64MB default. Not included in mx value.)

- New Size reserved: 357892096 initial: 357892096
- Old Size reserved: 715849728 initial: 715849728
- Perm Size reserved: 67108864 initial: 1048576

E) JVM Code Cache – 32MB
F) 300 threads in the application (300 * 512KB = 150MB)

Space left for the DATA (C-heap) region

Approximate (only significant, greater than 5MB, regions shown in calculation)

2 GB – C – D – E – F - space consumed by TEXT

Virtual Address Space Usage: Example 2

A) maxtsiz – 1GB
B) maxdsiz – 1GB
C) maxssiz – 400MB
D) Java heap - -Xms500m –Xmx1500m (will invoke java_q3p)

- New Size reserved: 524288000 initial: 174718976
- Old Size reserved: 1048576000 initial: 349569024
- Perm Size reserved: 67108864 initial: 1048576

E) JVM Code Cache – 32MB
F) 300 threads in the application (300 * 512KB = 150MB)

Space available for the Java thread stacks
Approximate (only significant, greater than 5MB, regions shown in calculation)

3 GB – C – D – E – space consumed by TEXT – space consumed by DATA

OutOfMemoryError: Example 3

Throwable: java.lang.OutOfMemoryError: unable to create new native thread

java.lang.OutOfMemoryError: unable to create new native thread

at java.lang.Thread.start(Native Method)

CHECK

- Whether there is enough space for private mmaps for thread stacks.
- The number of threads in glance/gpm and see whether max_thread_proc and nkthread are set appropriately.