EECS 768 – Virtual Machines – Spring 2015
Project Report – 3
Names:Bharath Chandra Elluru, Mahitha Doddala, Mohammad Ziauddin

1 Project Goals

The purpose of this project is to understand and identify the code that generates root set during garbage collection in JVM(Java Virtual Machine). Root set generation is a process which is done by JVM to identify live objects in heap. Once this root set is generated, garbage collector is used to collect all the dead objects in the heap. We were intrigued by the fact how a garbage collector manages memory by detecting live and dead objects and freeing the memory for dead objects. We wanted to learn practically how this process happens in hotspot JVM. Software Requirements : Hotspot JVM(Gamma version), Linux Fedora OS Final Deliverable : Understanding the root set generation, four phases of Mark and sweep garbage collector.

2 Project Timeline

The steps that we follow are:
1. Try to identify the code where garbage collection starts in JVM, so that we can try to search for root set generation code assuming this would follow the start up code in garbage collector.
2. Use "find command to search for files where mark sweep string is used assuming that this would produce a list of files where mark sweep algorithm i.e; root set generation algorithm is implemented.
3. Use a java program to invoke garbage collector explicitly using system.gc( ) command and debug the hotspot code.
4. Analyze how Mark and sweep algorithm is implemented in four phases.
5. Use " -XX:+PrintGCDetails" flag to run our program assuming that this flag would examine through the garbage collection code.

3 Current Status

1. We wrote a java program "Hello.java" from which we are explicitly calling garbage collector using System.gc( ) system call.
2. We searched for mark and sweep string using "find" command and discovered it in genMarkSweep.cpp file where mark and sweep garbage collection is implemented in four different phases.
3. Using gdb in hotspot, we have set up the break points in genMarkSweep.cpp to see if this file is reached during garbage collection and we could see the file is reached wlhile execution.
4. The following is the backtrace from mark and sweep implementation. We can see that vmGCOperations.cpp file is used to decode the arguments,GC Operation and redirect it to appropriate file. Then "do_full_collection" method is called from which we are calling "collect" method, which inturn invokes mark and sweep phase1 at a safepoint.
#0 ReferenceProcessor::oops_do (f=0xf7bb2e10 ¡MarkSweep::follow_root_closure¿)
at /home/belluru/vm/OpenJDK/openjdk/hotspot/src/share/vm/memory/referenceProcessor.cpp:171
#1 0xf7725ab4 in SharedHeap::process_strong_roots (this=0xf6e1ece8, activate_scope=4148661249, collecting_perm_gen=4155844353, so=SO_SystemClasses, roots=0xf7bb2e10 MarkSweep::follow_root_closure, code_roots=0xe0a92bc4, perm_blk=0xf7bb2e10 ¡MarkSweep::follow_root_closure¿)

at /home/belluru/vm/OpenJDK/openjdk/hotspot/src/share/vm/memory/sharedHeap.cpp:149
#2 0xf74da8fd in GenCollectedHeap::gen_process_strong_roots (this=0xf6e1ece8,
level=1, younger_gens_as_roots=4148326656, activate_scope=3769182977,
collecting_perm_gen=513, so=SO_SystemClasses,
not_older_gens=0xf7bb2e10 ¡MarkSweep::follow_root_closure¿,
do_code_roots=3769183233,
older_gens=0xf7bb2e10 ¡MarkSweep::follow_root_closure¿)
at /home/belluru/vm/OpenJDK/openjdk/hotspot/src/share/vm/memory/genCollectedHeap.cpp:732
#3 0xf74ddde9 in GenMarkSweep::mark_sweep_phase1 (level=1,
clear_all_softrefs=False)
at /home/belluru/vm/OpenJDK/openjdk/hotspot/src/share/vm/memory/genMarkSweep.cpp:275
#4 0xf74dd685 in GenMarkSweep::invoke_at_safepoint (level=1, rp=0xf6e93060,
clear_all_softrefs=4148572928)
at /home/belluru/vm/OpenJDK/openjdk/hotspot/src/share/vm/memory/genMarkSweep.cpp:102
#5 0xf74e7095 in OneContigSpaceCardGeneration::collect (this=0xf6e217c0,
full=True, clear_all_soft_refs=False, size=0, is_tlab=False)
at /home/belluru/vm/OpenJDK/openjdk/hotspot/src/share/vm/memory/generation.cpp:473
#6 0xf778e4de in TenuredGeneration::collect (this=0xf6e217c0,
full=4146921473, clear_all_soft_refs=3769183488, size=0, is_tlab=False)
at /home/belluru/vm/OpenJDK/openjdk/hotspot/src/share/vm/memory/tenuredGeneration.cpp:311
#7 0xf74da37e in GenCollectedHeap::do_collection (this=0xf6e1ece8, full=256,
clear_all_soft_refs=4155855240, size=0, is_tlab=False, max_level=1)
at /home/belluru/vm/OpenJDK/openjdk/hotspot/src/share/vm/memory/genCollectedHeap.cpp:610
#8 0xf74db4bf in GenCollectedHeap::do_full_collection (this=0xf6e1ece8,
clear_all_soft_refs=4148259328, max_level=1)
at /home/belluru/vm/OpenJDK/openjdk/hotspot/src/share/vm/memory/genCollectedHeap.cpp:937
#9 0xf77cf4e7 in VM_GenCollectFull::doit (this=0xf6fecebc)
at hotspot/src/share/vm/gc_implementation/shared/vmGCOperations.cpp:180In mark_sweep_phase1

3.1 Mark and Sweep Phase1

FileName Function Name Useful Line Number
1. genMarkSweep.cpp mark_sweep_phase1 line 268
"gen_process_strong_roots" function is invoked from mark_sweep_phase1 function to process the roots.
2. genColletedHeap.cpp gen_process_strong_roots line 731
"process_strong_roots" function is called
3. sharedHeap.cpp process_strong_roots line 175
The scanning option argument passed to this function is set to systemclass and based on that, we are calling SystemDictionary::always_strong_oops_do.
4. systemDictionary.cpp always_strong_oops_do line 1728
systemDictionary::always_strong_oops_do This function provides GC support. Following the roots during marksweep are seperated into two phases.The first phase follows preloaded classes and all other system classes,since these will never get unloaded anyway.The second phase unloads unreachable classes from the system dictionary and follows the remaining classes contents. We call always_strong_classes_do from this method.
5. systemDictionary.cpp always_strong_classes_do(blk) line 1736
We make a call to oops_do method which Visit extra methods apart from preloaded classes and system classes for marking.
invoke_method_table()¿oops_do(blk);
6. dictionary.cpp line 374

oops_do method is called which uses do_oop() method to iterate through objects in dictionary and mark them.
oop closure is a closure that has a method, do_oop() which takes fields of objects as parameters and does something arbitrary with them like for instance following references to children. Traversing through all the objects and marking is expected to complete here and the control flows back to genMarkSweep.cpp.
7. genMarkSweep.cpp mark_sweep_phase1 line 280
process_discovered_references method is called which is used to process the discovered references during marking
8. referenceProcessor.cpp process_discovered_references line 195
The process_discovered_reflist method is called which is used to process the discovered references during mark phase and remove the dead objects. This processing is repeated for soft, weak and Final and Phantom reference lists. control returns back to genMarkSweep.cpp
9. genMarkSweep.cpp mark_sweep_phase1
Control returns back to genMarkSweep.cpp and all the dead objects are removed using unlink method,i.e; the string tables and symbol tables are cleaned up.

By the end of phase1 all the live objects are marked.

3.2 Mark and Sweep Phase2

In mark_sweep_phase2, we are preparing for compaction. Once the dead objects are removed, we need to do compaction.
1. genMarkSweep.cpp - mark_sweep_phase2 - line 331
gch ¿ prepare_for_compaction() method is called from mark_sweep_phase2 method.
2. genCollectedHeap.cpp - prepare_for_compaction - line 1253
This method makes a call to ContiguousSpace::prepare_for_compaction in space.cpp which calls SCAN_AND_FORW
3. SCAN_AND_FORWARD
This is used to compute the new addresses for the live objects and store it in the mark Used by universe::mark_sweep_phase2().

Compaction is expected to be completed by end of phase2.

3.3 Mark and Sweep Phase3

1. Gets the collection heap using the heap() call
2. From the heap it takes the permanent generation using the perm_gen() function
3. Checks whether it needs to perform mark sweep
4. Adjusts pointers for all the live objects using pre_adjust_pointers()
5. Gets the generation level for all root objects and the non root objects
6. Processes the system dictionary objects using gen_process_strong_roots
7. Processing weak roots and adjusting the pointers for all the objects
8. Iterate the above steps for all the generations

3.4 Mark and Sweep Phase4

1. Gets the collection heap using the heap() call
2. From the heap it takes the permanent generation using the perm_gen() function

3. Compaction of the unused space is done using the compact() function call. The function checks word by word within the memory if they have been assigned to an object and tries to reallocate the live objects to create a contiguous list of memory words. In some cases the objects are not moved. 4. This process is repeated for all the generations.
5. Then compaction is performed over objects that share memory space.