ijecs open access www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 4 Issue 3 March 2015, Page No. 11119-11125

Memory Management In C++ And Java

Gayathri Kandasamy Sengottaiyan

gayathri.sengottain@gmail.com

Professor: Tarik El Taeib

teltaeib@my.bridgeport.edu

University Of Bridgeport -Computer Science

ABSTRACT

Memory management is basic for all languages because it is the important factor to determine the efficiency of the language. Language like Java have own garbage collector hence the programmer no need to do memory management. But in C++ the programmer has to release the memory through new and delete functions. In this paper, we are going to discuss how the memory is managed in both the languages and what are all the issues are there in memory in both the languages.

Keywords: Call stack, Activation Record, Smart Pointer and Garbage Collector

1.INTRODUCTION

Memory management in C++ and Java. Both the program has memory management techniques. This paper compare the differences between their memory management approach and garbage collectors. Basically C++ does not have efficient garbage collector but it has some means to achieve this through smart pointer, RAII. Then the paper will analyses the issues in garbage collector. To define the memory function we should understand about the call stack, this is very important for every programming language. Every program has function the main job of the function has to return the value to the main function the main function will return value to the memory about the status. So call stack has to be maintained throughout the program. This paper will discuss how call stack is works in C++ and Java. Another memory in the system which is most powerful and large area of memory has been hold by this memory called heap memory. When the program needs memory as a block it will be allocated from the heap memory by operating system. This will be more efficient than stack memory but it should be used efficiently to prevent deadlock condition. Both C++ and Java uses dynamic memory allocation to use heap memory, this paper also analyses how this heap memory is used by the C++ and Java. Here the smart pointer and garbage collector plays a major role.

2.MEMORY MODEL

Before discuss about each type of memory we will have brief note to know what are the memory used by these languages and their purpose of using the particular memory for particular task.

2.1.CODING AREA

The area where all the code will be loaded before the execution.

2.2.STACK

This is also called execution stack after loading execution will be done here like all the computation work, function call, recursive call etc.

2.3.HEAP

This is dynamic memory allocation this is a large pool of memory so all the objects will be reside here.

3.CALL STACK

Call stack is the memory where the code of the program will be stored and starts the execution then returns the value if it has done its work. The call stack is also known by the "Execution Stack", "Machine Stack"[4]. But it will be simply referred as "Stack". The subroutine is alive unless it knows where the value should be returned or where the control to be transferred after execution. In order to full fill it task the call stack has to keep track of each function or task and the call stack will be monitoring for all the subroutine which are in the stack. Operation of the stack is "push" and "pop". So the subroutine has to push the address when it want to perform its task and take out the address when it has done its task. The important function of the call stack is maintain the address to retain the function control as it is mentioned above the address can be stored indifferent ways. But most

convenient way is to store on same task where the program function is loaded for execution, this will pay more benefit like recursion function will be done more easily since it has the address on the same location where the function resides.

All the function are reside in the "stack frame" it is also called as "Activation Record". Let see how this activation record work for different function.

3.1ACTIVATION RECORD

Activation record has the information which is needed for the execution is placed on the stack. The information is known as "Activation Record". This record will be allocate the memory for the function fromtop to bottom. As soon as the computation is done for the function or "method call", the memory allocated for this particular record will be destroyed. For example we will see simple function call how it is allocated in the "frame".

```
voidrkr()
{
ga();
}
voidga()
{
dr();
}
void gr()
{
gk();
}main(){gk();}[4]
```

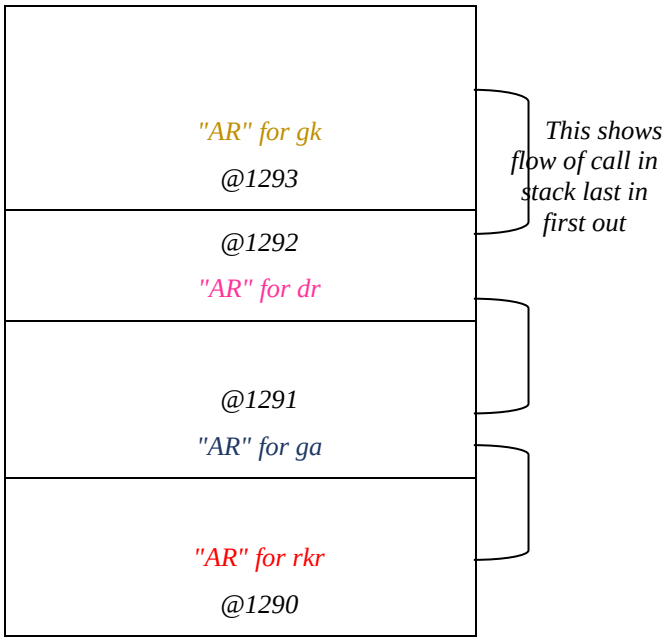


FIG1:Memory allocation diagram of above program

The "call Activation Record" contain some of the information which need for the proper execution

"RV" this is return value after execution it return s control to the specified location sometimes value.

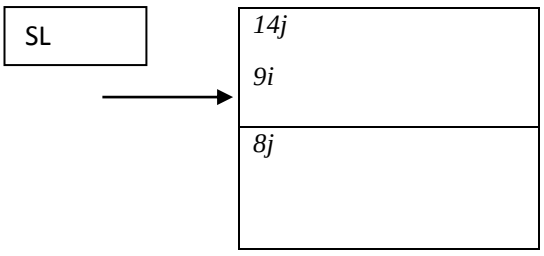
"RA" this in turn will return the address of the function that is location of the control where it return after execution. when the program starts suppose if you take the above example the program will call the function "rkr" then the stack pointer address will be "1290" when it finish the execution it return to "ga" function so it return the address to stack pointer "1291" now the pointer will points to "ga". Till then we have seen about the activation record at function level next thing we are going to see how the activation record going to handle the function attributes that is variables.

In all the programming language we are talking about the scope of the variable we know how the scope is working but in activation record how it detects and how it allow as to have a two different variable with same name in two different function.

To do this task the "stack" is having something calls "scope activation record" this holds two different type of information, information about the local variable and the "static Link " information which is a pointer for the "SAR" it gives the information where the scope of the variable ends and how to access the global variable[7].

Let us see through this example code

```
voidrkr()
{
int j=8;
{
i=9;
j=14;}[7]
```



\

FIG2:Diagram shows the variables in memory

4.STACK IN C++

When the program starts its execution the activation record will be created in the stack inorder to store the variables and method to compute. The activation record contains values of the variable, return value of the function, register's content. While the function has called it has been interrupted by another function. But the function's job has not finished so it need some memory to store its address on the other hand some data structure. That is "stack" functions are invoked the way it interrupted and it has been

manipulated during the run time "so it is called Run time stack". The most advantage in stack is "object on stack are fully managed by the compiler" the programmer no need to allocate and de allocate memory. Since the memory is managed by the compiler the object which has the scope or in other words which know its life time can only be in stack. "In fact every object has a static size which is known at compile time (determined using the sizeof operator). As a result, every object can be placed on the stack. But this does not mean that all memory requirements for this object can be satisfied by the stack: if you place a string on the stack, the implementation of the string would need to allocate memory from the heap to full fill it's duty."

Let see how the call stack is utilized by the C++. This has explained with the following example C++ code

```
int main(){
        int a=3;
        rkr(a);
        cout<<endl;
}
Void rkr(int x){
        cout<<ga(x+1);
}
Intga(int p){
        int q=gk(p/2);
        return 2*q;
}
Intgk(int n){
        return n*n+1;

}[7]
```

p4		*Q*	*B"AR" for ga*
x3			*A"AR" for rkr*
		a3	*os"AR" for main*

FIG3: Activation Record of above program

5. CALL STACK IN JAVA

Like in C++ java is also using the call stack to load the run able subroutine. java is fully object oriented so it have several class . In C++compiler will be responsible for stack allocation, here it takes care by "Java Virtual Machine (JVM)". The Java will load the corresponding byte code which is a run able code and it allocate memory as structured memory. Several run time area is there in "JVM". Here we are going to see about the java stack.

java stack consider all the method as a thread and it will do "pop" an "push" operations on stack. The thread which is running on the stack is "current method" of the "current class". The thread will compute with the local variable or other member with the current frame. Java stack will "push" the frame in to stack when the thread call the method. This method can either completes the operation by returning the value or control which states it completes normally, if it throws exception then it turns "abrupt completion". In either way after completion the corresponding stack memory will be deleted. Unlike the stack in C++ here the memory need not to be contiguous. It is based on the programmer they are allowed to specify the size of the stack.

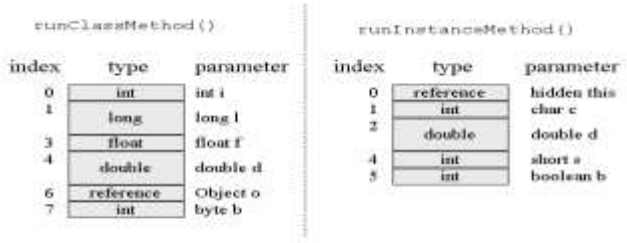
java stack has three parts "local variable", "operand stack", "frame data"

5.1LOCAL VARIABLE

It has "zero size array of words" it will store value of all the data type, references and return address as a single entry in the array. Since it is a char array it will consider everything as a char so before storing in to array value of float, double will be converted in to integer and then it will be stored in the array.

```
"class Example3a {

public static intrunClassMethod(inti, long l, float f,
double d, Object o, byte b) {

return 0;
   }

publicintrunInstanceMethod(char c, double d, short s,
boolean b) {

return 0;
   }
}"[7]
```

This is an example program how the local variable of the frame get stored. Here the index are used to retrieve the variable. This is well defined in the following picture.



[7]

FIG4*: stack of class method and instance method memory picture*

5.2.OPERAND STACK

This is also create "array of words" like local variable but it will not access by its "indices", it will do "push" and "pop" operation in the stack.

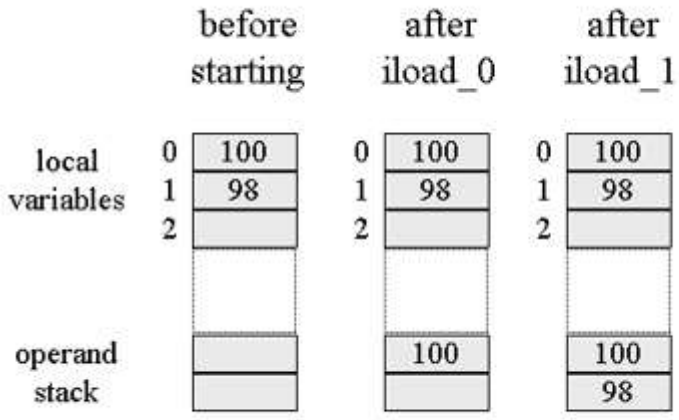
Just like the local variable it will also convert the int,float,double values to integer before pushing in to the operand stack."JVM" uses the operand stack as the work space it will pop the variable from the stack and it again push the result in to the stack. Let see this through following code

"iload_0 // push the int in local variable 0

iload_1 // push the int in local variable 1

iadd // pop two ints, add them, push result

istore_2 // pop int, store into local variable 2"



[4]

FIG 5:operation of JVM stack

5.3.FRAME DATA

If the data is constant pool of information than the frame data is used to refer the data, it use pointer for the frame to access the data. The data types, address and all type of references will be like symbols. So frame data will assist the virtual machine to complete the stack operation if the stack completes the execution normally it will resume the current method from the stack and it will make the pointer to point the register for further information and makes it to complete the method. If it returns the value make it store in the operand stack of the current method.

6.DYNAMIC MEMORY MANAGEMENT

We shall discuss how the dynamic allocation works in C++ and Java. Since this is most important form of memory it should be used efficiently in order to avoid shortage of memory. Here we are going to discuss how the garbage collector is working in C++ and Java. As it said before heap is large pool of memory which is slower than stack but it is an efficient since the value reside here long time than in the stack.

7.DYNAMIC MEMORY MANAGEMENT IN C++

The Dynamic memory is achieved using pointers let us see the brief introduction about the pointers. In the dynamic memory we are not determine the size of the memory at compile time rather by run time. Here only reference of the variable will be present since the memory is dynamic.

Pointer is the concept used for dynamic allocation in C++, this holds the address of another variable and also used to allocate dynamic memory.

This is dynamic allocation of single variable

```
int * p = new int; // dynamic integer, pointed to by
*p = 10;           // assigns 10 to the dynamic integer
cout<< *p;         // prints 10
```

This is for set of array

```
double * numList = new double[size];   // dynamic array

for (inti = 0; i< size; i++)
numList[i] = 0;                        // initialize array elements to 0

numList[5] = 20;                       // bracket notation
*(numList + 7) = 15;                   // pointer-offset notation
                                       //
means same as numList[7]
```

" There is a set of operators which are relate to dynamic memory management lists six variations of operator new and a matching operator delete for each. This paper will only use the most basic variant."

7.1.SMART POINTER

In order to fix this issue the smart pointer is used let we discuss in brief. In C++ memory leak is caused by misuse of the pointers. There may have chance the user may forgot to delete the pointer after using this, it is not wise idea to delete pointer explicitly because it cause overhead to the programmer. So we need a solution which automatically recover the memory after it has done the job. That means is smart pointer.

The smart pointer are look like a pointer it has same properties like dereferencing and referencing and it also had added advantage that it can also be an object, "so it can call constructor and destructor "

```
class Jack
{  int age;
char* name;
public:
Jack(): name(0),age(0)
      { }
Jack(char* name, int age): name(name), age(age)
      {}
      ~Jack()      {}
void Display()
{printf("Name = %s Age = %d \n", name, age); }
void Shout()
      {          printf("Ooooooooooooooooo",);}
};
void main()
{
   Person* ja  = new Person("Scott", 25);
ja->Display();
delete ja;
}
```

In this case we should explicitly remove the pointer after it has done in the above program name is not deleted so it will occupy the memory. So we need the pointer to handle by itself.

The smart pointers are used by using operator overloading.

```
class SPEX
{
private:
   Person1*    pData; // pointer to person class
public:
SPEX(Person1* pValue) : pData(pValue){}

   ~SPEX(){  // pointer no longer requried;  delete pData;}
   Person& operator* ()   {        return *pData;  }
Person* operator-> ()   {          return pData;}};}

   void main()
   {
      SPEX<PERSON1>p(new Person("Scott", 25));
   p->Display();
      {
         SPEX<PERSON1> q = p;
         q->Display();
         // Destructor of Q will be called here..
      }
   p->Display();
   }[7]
```

Here as soon as the scope ends the corresponding destructor will be called and the memory will be destroyed. auto pointer is an efficient smart pointer which holds the address of another object it will de allocate the object when the object goes out of scope.

7.2.ISSUES IN AUTO POINTER

The auto pointer has some limitation it has to transfer the ownership and destruct the copy. To overcome this we can implement the pointer with counter if one pointer points the same object the counter should increment and if any other pointer lost the ownership the counter need to be decrement and make the pointer to point any other object. when the counter reach 0 it should delete the object which it points[2].

7.3.RAII

*"**Resource Acquisition Is Initialization (RAII)**[4] is a programming idiom used in several object-oriented languages, most prominently C++". RAII is first developed in C++ and later it has been used by many programming languages. This will tie with the object scope or "lifetime", it will manage the memory based on the scope. It is not a garbage collector which resides on time, resource is allocated at the time of object created and it will be de allocated when the object scopes ends.*

8.DYNAMIC MEMORY MANAGEMENT IN JAVA

java has very great advantage through garbage collector. So the programmer no need to get worry about the memory problems such as creating the memory and removing it explicitly.

8.1.WORKING OF GARBAGE COLLECTOR

There is a misunderstanding in working of garbage collector. The garbage collector tracks the object which are alive and it stores the rest of the object in "designate garbage". It will not kill the dead object.

In dynamic allocation the memory area used is heap, this has done by the operating system as soon as dynamic allocation starts, this can be easily managed by the JVM and it also have some of the advantages. The first one is it is not needed to synchronize with the operating system

globally at each time of creating object. Because the object creation is followed by the successive pointer. Second thing is if the object is not used for long time it automatically reused so there is no overhead of explicit deleting.

As said in this paper before in heap the objects can only be referred. So as far as the object is referenced in the memory JVM consider the object alive if not the memory can be used by other object that is dead.

Java always create object in heap, let see through some example how it is working

```
public class CoffeeMaker {

publicCoffeeMaker(){} } .. ..

CoffeeMakeraCoffeeMaker; aCoffeeMaker = new CoffeeMaker();

int sugar = 4;

Integer sugarObject = new Integer(3);[6]
```

The memory allocation of the above program, there are two classes coffeemaker is the user defined class and Integer is the default class. So the object of the two classes will resides in the heap and the values will resides in the stack. java variables which are in heap can be access through only the reference variable, Since in heap the values are referenced.

8.2.NULL VALUE

In java if the object is not pointing to any of the object then it is initialized to null which throws some exception. So the object which is not used has to be nullify by its reference. But in case of C++ it is considered as a macro and it does not point to any object.

8.3.SWEEPING GARBAGE

To detect which dead object JVM has the algorithm to collect the dead object, the algorithm is mark and sweep algorithm. The algorithm start its search from the "GC" root and mark the memory which are referenced by the object and the memory which are free collect as memory for reuse.

8.4.ISSUES IN GARBAGE COLLECTOR

The garbage collector collects the memory which is not used but it does not delete reference in heap. The garbage collector is successful in deleting object from memory but the value of the pointer remains it results in dangling pointers some time. This situation occurs in multithread condition. "Real-Time Java therefore supports two kinds of threads: real-time threads, which may access and refer to objects stored in the garbage-collected heap, and no-heap real-time threads, which may not access or refer to these objects. No-heap real-time threads execute asynchronously with the garbage collector; in particular, they may execute concurrently with or suspend the garbage collector at any time. On the other hand, the garbage collector may suspend real-time threads at any time and for unpredictable lengths of time"[3].This technique try to give memory control to the programmer Real Time Java has implementation model you can refer this paper for more details.

9.DESTRUCTOR AND FINALIZER

we know both the language has similar functions in most cases since both are object oriented language in order to clear the function or value from memory before it removed automatically by operating system we have destructor in C++ and finalizer in java. Both are responsible for clear the memory but still it has some difference. Let we see through example

```
"classCPPdeallocate {
public :
CPPdeallocate (): _p( new int ())
{
};
~ CPPdeallocate ()
{
delete _p;
}
private :
int * _p;
};"[1]
```

Here the pointer is used for dynamic allocation when the cppdeallocte() has been called the object will call the destructor and it delete the memory and other resource which was hold by this object.

If in case of java the use of finalizer is not as efficient as destructor since it is "undefined". If you see this code.

```
"public class JavaFinalize {
private static void work () {
myresource t = new myresource ();
}
public static void ptime ( String info ) {
System .out .println ( info + " : " +
System .currentTimeMillis ());
}
private static void sleepwell ( long m ) {
try {
Thread .sleep (m);
} catch ( Exception e ) { }
}
public static void main ( String args []) {
ptime (" Start ");
work ();
ptime (" sleep ");
sleepwell (2000);
ptime (" sleep done ");
System .gc ();
ptime (" End ");
}
}
classmyresource {
protected void finalize () throws Throwable
{
JavaFinalize .ptime (" finalizer ");
super .finalize ();
}
}"[1]
```

In this program the finalizer that is system.gc() called after the sleep. so the object is delete after some time mean while the program will terminate so there is no use of finalizer here. So the finalizer is not used as destructor in C++. In other words java finalizer is not reliable.

10.CONCLUSION

When compare the two languages such as C++ and JAVA, both have some sort of advantages and did advantages. C++ is said to have more memory leaks it can controlled by some point with smart pointer and RAII. These techniques are implemented manually or it is semi automated but there should be a technique which is fully automated like Java.

As we see in Java garbage collector is fully automated but sometimes like in multi threading it creates "Dangling pointer" reference. The next thing the garbage collector has to search and collect the memory from heap it will slow down the process and that hence Java is not as powerful like C++ which is faster than Java. So the technique has to improve in java as in performance level and some of the research like "RTJ(Real Time Java)" has try to solve this real time problems. Still in C++ automation is needed for memory management which should not compromise language efficiency.

11.Conflict of Interest

There is no conflict of interest

REFERENCES

[1]. Aspects of Memory Management in Java and C++ Emil Vassev, Joey Paquet Department of Computer Science and Software Engineering Concordia University Montreal, Quebec, H3G 1M8, Canada {i_vassev, paquet}@cse.concordia.ca[1]

[2]. A practical comparison of the Java and C++ resource management facilitiesMarkusWinand 2003-02-08

[3]. An Implementation of Scoped Memory for Real-Time Java William S. Beebee, Jr. and Martin Rinard
[4].http://en.wikipedia.org/wiki/Call_stack

[5].http://www.artima.com/insidejvm/ed2/jvm8.html

[6].http://javabook.compuware.com/content/memory/impact-of-garbage-collection-on-performance.aspx

[7].Object Oriented Memory Management (Java and C++)
http://bd-things.net/object-oriented-memory-management/