Improving of fragmentation problem in memory management for embedded systems

Tsuyoshi Takahashi (910060)

School of Information Science,
Japan Advanced Institute of Science and Technology

February 15, 2002

Keywords: embedded system, memory management, fragmentation、Viewer、algorithm .

1 Background

The complexity and the scale of embedded systems become larger and larger. Whereas it was once confined to the realm of home electric appliances and numerical controlled machine tools, etc. to limited purpose, today embedded systems are multi-purpose such as PDA and cellular phone. However, development environments for embedded systems are still quite poor; their main programming languages are often only assembly languages and /or C. Thus, technological innovation for develop environment is highly required. Java is one of the answers.

Java has many advantages such as good portability, reusability and *garbage collector*(GC). However, conventional Java VM is too big to port embedded systems. KVM copes with this problem and enable us to apply Java to embedded systems. KVM is designed for products with approximately 128KB of available memory.

Despite the rapid growth in memory size of even the embedded system, the supply of storage is not enough. KVM design of GC is mark-sweep algorithm which is non-moving and non-incremental. This design has two problems. First, memory *fragmentation* can cause it to run out of heap, since GC design is non-moving. Second, user programs have longer GC pause time as the size of used heap memory increases, since the GC is non-incremental and all objects used in the heap are traversed at a time.

2 Purpose

The purpose of this paper is to improve memory fragmentation problem for embedded systems. Embedded systems are often require to run long time. Thus we aim to construct an allocator that bears a long run situation, reduces memory fragmentation. Second purpose of this paper is to develop a framework to customize the behavior of the allocator. We consider two functions to construct the framework. First, this framework can change parameters such as boundary of first fit and two-level allocation, block size, etc. Second, we can change parameter theoretically, intuitively so that we can check inside of memory visually.

Two ways reduce memory fragmentation: introduction of compaction to GC and introduction of allocation algorithm which reduce the fragmentation to the memory management mechanism. We choose the second way, because:

1. KVM is implemented by C. Thus, GC cannot identify pointers and objects.
2. If simple compaction is added to GC, pause time may become long. Incremental compaction can result on much longer execution time.
3. If we can build an allocator that reduces memory fragmentation, the number of compactions may be reduced.
4. Even simple algorithm may have good performance such as *Slab allocation* used in Linux. Thus we try to find allocation algorithm that reduce the memory fragmentation.

3 Algorithms

We propose two memory allocation algorithms as follows.

- Block buddy system
- Separate first fit

Block buddy system has the fixed number of objects as a block. For example, a block has 8KB×10 or 16KB×10, etc. This way deal with the block with Binary buddy system. This algorithm makes heap memory highly reusable. Because this algorithm accept internal fragmentation. Thus heap memory can save large area. But this algorithms is faster than other algorithms to exhaust heap memory. Thus, the number of GC increases.

Separate first fit algorithm, which divides the entire heap memory, and decides a area of allocating objects, allocates an object to certain area with first fit algorithm. This algorithm is more effective in heap memory usage than conventional first fit algorithm,

since the difference between the sizes of allocated objects is small. But if we cannot determine where to divide, efficiency goes down.

4 Preliminary evaluation

There are two ways to measure the memory fragmentation as follows.

- experiment on real-world applications
- experiment on experimental applications that (de)allocates randomly

If we experiment on real-world applications, we give suitable allocation algorithm and parameters for working environment and application. However, this way takes long time, since we must compile again whenever we change an allocation algorithm or parameters. If we experiment on experimental applications that (de)allocate randomly, we give an average performance of each allocation algorithm. And we can implement easily than real-world applications. We choose the second way. Because, our purpose is to investigate general behaviors of the algorithms.

Procedure of this experimental test using framework is as follows.

1. A list of sizes of allocated objects requested by applications working on KVM is recorded. By random selection from this list, object sizes to be allocated are decided.
2. When heap memory is exhausted, a certain size of memory is freed.
3. Objects are allocated with the sizes randomly decided.

We had experimented 5 algorithms with this procedure. As a result of this experiment, Plurality first fit algorithm has the highest performance.

5 Our measurement of fragmentation

Measurement of fragmentation is important in this paper. In general, fragmentation is represented as follows.

$$\frac{\text{realSize} - \text{requestedSize}}{\text{realSize}}$$

What is important is percentage for this way. We proposed another way as follows.

$$\frac{\text{AmountOfFreedSize} - \text{AmountOfAllocatedSize}}{\text{AmountOfFreedSize}}$$

And we draw up graphs by this way. We calculates least-squares method and uptilt. What is important is uptilt for this way. Because uptilt indicates increase of fragmentation in terms of time.

6 Conclusion and Future Works

The conclusion of this study is as follows.

- We propose two memory allocation algorithms to reduce fragmentation, and construct a framework to tune parameters.
- By using this framework, application programmers can easily choose efficient memory allocation algorithms and their parameters.
- We showed even easy algorithms may show good performance.

Future works of this study is as follows.

- The framework calculates appropriate allocation algorithm and it parameters automatically.