# Understanding the Performance of the Java Operating System JX using Visualization Techniques

Michael Golm, Christian Wawersich, Meik Felser, Jürgen Kleinöder
Dept. of Computer Science
University of Erlangen-Nuremberg
{golm, wawersich, felser, kleinoeder}@informatik.uni-erlangen.de

In the JX project[1] we build a complete operating system using the type-safe, object-oriented language Java. One of the major challenges is achieving a performance that is competitive to mainstream operating systems, such as Linux or Solaris. It is in the nature of our microkernel-based system, that most time is spent executing Java code. Achieving a good performance therefore requires an optimizing bytecode-to-nativecode translator and instrumentation/visualization tools to locate performance problems. In this paper we concentrate on the second requirement. We instrumented the microkernel and the bytecode-to-nativecode translator to get information about nearly every aspect of the systems behavior. The gathered data can be visualized to show CPU time consumption of methods, thread scheduling, object allocation behavior, and object ages.

**Time related information.** To improve the code, it is necessary for the programmer to identify the most time-consuming parts of a program. These often indicate poor code or bottlenecks and are the parts of the code that benefit most from optimizations. We have two possibilities to identify such parts: (1) method timing and (2) sampling.

**Method timing.** For method timing we instrumented our bytecode-to-nativecode translator to insert code that measures the execution times of methods. The elapsed time during a method is recorded using the time stamp counter (TSC) of the Pentium processor family. The result is totalized for each invocation pair consisting of caller and callee method. The sum is saved together with the number of invocations and the caller and callee address in a data structure per thread. Especially in nested methods the measurement overhead falsifies the result. To get more accurate results, we accumulate the quantified drift by identifying the needed extra cycles. The results can be visualized using a call graph display, or a time diagram, where the time consumed by a method is proportional to the size of a box. Furthermore the execution of extra code during the measurement has noticeable impact on the memory access, the caches, the buffers, and the branch prediction of the processor.

**Sampling.** Sampling is used to find hotspots in the system, but with almost no measurement overhead. While timing measures the exact execution times of methods, sampling can only produce a statistical view of the system. The timer interrupt service routine (ISR) logs the instruction pointer of the interrupted instruction. The distribution of the instruction pointers indicates where the program spends most of its time.

**Event tracing.** JX allows Java components to log component-specific events to an kernel internal event log. The events can be specified by using a system interface. Additionally our bytecode translator may insert calls to the log system to record specific method calls. Time stamps and event numbers are continuously stored in main memory.
The microkernel also uses this logging mechanism to log some internal events. For example, the scheduling behavior of the system can be logged and visualized in a thread activity diagram.

**Memory related information.** Java uses an automatic memory management called garbage collection. During performance debugging we want to know exactly when objects are allocated and destroyed. We collect this information and generate several diagrams.
A time diagram of object allocation can be used to see when objects are created or destroyed and in which library, class, or method they are created. Additionally color code informs about either the size of the allocated objects or the type (class).
For better analysis of object aging a aggregated diagram can be generated displaying the lifetime of objects sorted by their type. The time used in this diagram is measured in allocated memory, that means an object grows older when memory is allocated.
Another interesting information is the cache behavior. We use the Pentium III performance counters to measure the number of lines evicted from the L2 cache and the number of L2 requests. This information can be best evaluated in combination with a thread activity diagram.

**Conclusion.** A comprehensive analyses of a Java-based OS needs several instrumentation and visualization techniques. Instrumentation techniques range from very invasive method timing and object aging analysis to low overhead application generated events.

## Categories and Subject Descriptors

D.4.8 [**Operating Systems**]: Performance - *measurements, monitors*.

## General Terms

Measurement, Documentation, Performance.

## Keywords

Java Operating System, Performance Visualization Techniques.

1. for detailed information visit www.jxos.org

Copyright is held by the author/owner(s).
*JGI'02*, November 3-5, 2002, Seattle, Washington, USA.
ACM 1-58113-599-8/02/0011.