Memory Management in a Java-written Operating System

Why write an OS in Java?

- The operating system can benefit from:
  - Portability
  - Type safety
  - Automatic memory management provided to the applications
  - Easy installation of downloaded code
  - No language barriers between the applications and the system

Issues with Java memory management

- Memory management requires access to memory addresses, low-level structures, and to the hardware
- Java is not a low-level language:
  - How to write memory management policies in Java?
  - How to access the physical memory in Java?

Previous attempts to build Java environments

- Virtual machines:
  - jalapeño
  - joeg
- Operating systems:
  - JavaOS
  - JX

Presence of native code !

How do we manage memory in Java?

Entities introduced to manipulate memory in Java:

- Managing memory implies actions at two levels:
  - Activating the low-level devices
  - Using a set of management policies
- Two notions have been introduced to allow this in Java:
  - A Java bytecode processor and physical objects

Structure of our system:

- The upper layers of the execution environment are similar to traditional approaches (programs and JDK classes)
- The services provided by the JVM and the OS are here written in Java and lay on the Java bytecode processor and various physical objects

Details of the memory architecture:

- A basic version of our system is currently running using a software version of the Java bytecode processor, but the memory is not completely managed in Java
- A similar work is also being applied to other JVM and OS services:
  - Exception handling
  - I/O handling
  - Memory management
  - Process scheduling
  - At the same time, the Java bytecode processor and the physical objects are also being enhanced

Status: Current work; future plans