Run-Time Program Maagement

1. What is a run-time system? How does it differ from a ”mere” library?

2. List some of the major tasks that may be performed by a run-time system.

3. What is a virtual machine? What distingushes it from interpreters of other sorts?

4. Explain the distinction between system and process VMs. What other terms are sometimes used for system VMs?

5. What is managed code?

6. Why do many virtual machines use a stack-based intermediate form?

7. Give serveral examples of special purpose instructions provided by Java bytecode/

8. Summarize the architecture of the Java Virtual Machine.

9. Summarize the content of a Java class file.

10. Explain the validity checks performed on a class file at load time.

11. What is a just-in-time (JIT) compiler? What are its potential advantages over interpretation or conventional compilation?

12. Why might one prefer bytecode over source code as the input to a JIT compiler?

13. What distinguishes dynamic compilation from just-in-time compilation?

14. What is a hot path? Why is it significant?

15. Under what circumstances can a JIT compiler expand virtual methods in-line?

16. What is deoptimization? When and why is it needed?

17. Explain the distinction between the function and expression tree representations of a lambda expression in C#.

18. Summarize the relationship between compilation and interpretation in Perl.

19. What is binary translation? When and why is it needed?

20. Explain the tradeoffs between static and dynamic binary translation.

21. What is emulation? How is it related to interpretation and simulation.

22. What is dynamic optimization? How can it improve on static optimization?

23. What is binary rewriting? How does it differ from binary translation and dynamic optimization.

24. Describe the notion of a partial execution trace. Why is it important to dynamic optimization and rewriting?

25. What is mobile code?

26. What is sandboxing? When and why is it needed? How can it be implemented?

27. What is reflection? What purpose does it serve?

28. Describe an inappropriate use of reflection.

29. Name an aspect of reflection supported by the CLI but not by the JVM.

30. Why is reflection more difficult to implement in Java or C# than it is in Perl or Ruby?

31. What are annotations (Java) or attributes (C#)? What are they used for?

32. What are `javadoc`, `apt`, `JML`, and `LINQ`, and what do they have to do with annotation?

33. Briefly describe three different implementation strategies for a symbolic debugger.

34. Explain the difference between breakpoints and watchpoints. Why are watchpoints potentially more expensive?

35. Summarize the capabilities provided by the Unix `ptrace` mechanism.

36. What is the principal difference between the Unix `prof` and `gprof` tools?

37. For the purposes of performance analysis, summarize the relative strengths and limitations of statistical sampling, instrumentation, and hardware performance counters. Explain why statistical sampling and instrumentation might profitably be used together.

38. Summarize the architecture of the Common Language Infrastructure. Constrast it with the JVM. Highlight those features intended to facilitae cross-language interoperability.

39. Describe how the choice of just-in-time compilation (and the rejection of interpretation) influenced the structure of the CLI.

40. Describe several different kinds of references supported by the CLI. Why are there so many?

41. What is the purpose of the Common Language Specification? Why is it only a subset of the Common Type System?

42. Describe the CLI's support for unsafe code. How can this support be reconciled with the need for saftey in embedded settings?