Today
- Code/Process Migration
  - Motivation
  - Approaches and techniques
  - Secure Code Migration

Process/Code Migration
- Pass programs to be run remotely
- Migration could be
  - Weak (Code Migration)
  - Strong (Process Migration)
- Examples:
  - Java applets
  - Condor, OpenMOSIX
  - SETI@home
  - VM migration

Why Code Migration?
- Performance
  - Load balancing and parallelism
  - Utilization of idle resources
  - Localized computation (close to data)
- Flexibility
  - Easier to deploy and configure a DS
  - Do not need preinstalled software
  - Can change actual implementation
Process Structure

- Process consists of
  - Code segment: Instructions
  - Execution segment: Private data, stack, PC
  - Resource segment: External resources
- Can migrate one or more of the segments

Code Migration Models

- Weak mobility
  - Only transfer code segment
  - Can be executed by target process or new process
- Strong mobility
  - Transfer execution and resource segments as well
  - Move the process or clone the process

Initiating the Migration

- Sender-initiated
  - Sender sends the code to a remote machine
  - The receiver should trust the sender
- Receiver-initiated
  - Receiver asks for code to be executed on it
  - Does the sender need to trust the receiver?

Resource Migration

- How do we migrate resources?
  - Files, printers, sockets, etc.
- Process-to-resource binding
  - By-identifier: URL, file descriptor
  - By-value: Libraries
  - By-type: Printers, local devices
Resource Mobility

- Unattached
  - Can be easily moved
  - E.g.: data files
- Fastened
  - Can be moved, but high cost
  - E.g.: local database
- Fixed
  - Cannot be moved
  - E.g.: local devices, sockets

Resource Migration Techniques

- Global reference:
  - Set up a global name
  - Can be accessed remotely
  - Move/copy the resource
    - E.g.: copy small data files
  - Rebind to a local resource
    - E.g.: to a local socket

Migration in Heterogeneous Systems

- Machines can have
  - Different architecture/OS
  - Different runtime systems, libraries, utilities
- Weak Mobility
  - Require code to be compiled for target machine
- Strong Mobility
  - Also need to carry execution state

Scripting and Language Support

- Use a machine-independent architecture (virtual machine)
- Code runs on VM
  - Interpreted source code (Perl, Tcl, etc.)
  - Interpreted intermediate code (Java)
System Virtual Machines

- A private machine running on top of a physical machine
- Can migrate whole computing environment
  - OS, processes, runtime state
  - Local resource bindings move with the environment

Virtual Machine Live Migration

- Local clusters: Assume shared storage system
- Need to:
  - Copy CPU state and memory pages
  - Suspend VM at source and resume at destination
- Two main approaches:
  - Pre-copy and Post-copy

VM Live Migration Approaches

- Pre-copy:
  - VM runs on sources, copy memory pages iteratively (dirty pages in each round)
  - Downtime: Suspend VM, copy CPU state and remaining dirty pages
  - Resume: Restart VM at destination
- Post-copy:
  - Downtime: Suspend VM, copy CPU state and any essential pages
  - Resume: Restart VM at destination
  - Copy remaining memory pages
  - Post-copy: On-demand, active pushing, pre-paging

Pre-copy vs. Post-copy

- Pros and Cons?
Secure Code Migration

- Protecting machine from downloaded code
- How do we protect our local resources?
- Approaches:
  - Code-signing: Authenticate the code being downloaded
  - sandboxing
  - Playgrounds

Sandboxing

- Run the downloaded code in a controlled/isolated environment
- Example: Java Virtual Machine
  - Uses trusted class loaders
  - Uses byte code verification
  - Uses a security manager at runtime

JVM: Enforcing Security Policies

- Restricting access to local resources
  - Access only through references to objects
  - Hide references to local resources
- Stack Introspection
  - Every method call is preceded by checking caller’s authorization
  - Caller granted temporary privileges during call
- Name space management
  - Same classname can be mapped to different classes

Sandboxing Example: Native Client

- Chrome Native Client
  - Allows running native code inside browser. Why?
- Sandbox mechanisms:
  - Need to compile code via NaCl compiler
  - Code verified before execution
  - Restricted API and resource access
  - Utilizes browser’s sandboxing (e.g., runs inside a process with lower privileges)
Playground

- Separate machine exclusive for running migrated code
  - All resources of other machines are isolated
- Honeypots: for attracting potential viruses