CprE 450/550x
Distributed Systems and Middleware

Processes: Thread, Code Migration, and Software Agents

Yong Guan
3216 Coover
Tel: (515) 294-8378
Email: guan@ee.iastate.edu
March 4, 2004

Announcements

- The second project will be announced next Tuesday
- Mid-term Exam: Closed-form
  When: **Thursday, March 4, 2004**
  Time: **3:40-5pm**

3

Readings for Today's Lecture

- References
  - **Chapter 3 of "Distributed Systems: Principles and Paradigms"**

4

Outline

- Threads
- Code Migration
  - **What is code migration?**
  - **Approaches to code migration**
  - **Local resources**
  - **Code migration in heterogeneous systems**
  - **Implementation issues**
- Software Agents
  - **What is software agents?**
  - **Agent Technology**

5

Threads

- Finer granularity in the form of multiple threads of control per process
- Process: a program in execution
  - Program -> virtual processor -> process table (CPU register values, memory maps, opened files, etc)
  - OS ensures independent processes cannot maliciously affect the correctness of each other's behavior –separation
  - Independent address space for each process
  - Switching CPU between processes
  - Swap processes between main memory and disk
- Thread: can also be thought as a (part of a) program in execution on a virtual processor
  - Performance gain, concurrency
    - When a blocking system call is executed, the process as a whole is blocked.
    - Increase parallelism when executing the program on multiprocessor systems
  - Threads are not protected against each other like processes. So it needs propoer design
- Implementation:
  - Construct a thread library that is executed entirely in user mode
    - Cheap to create and destroy threads
    - User-level threads can be switched in a few instructions (CPU register values need be stored)
    - Invocation of a blocking system call will immediately block the entire process to which the thread belongs and all other threads in that process
  - Have the kernel be aware of threads and schedule them
  - Lightweight Process (LWP): A hybrid form of user-level and kernel-level threads

6

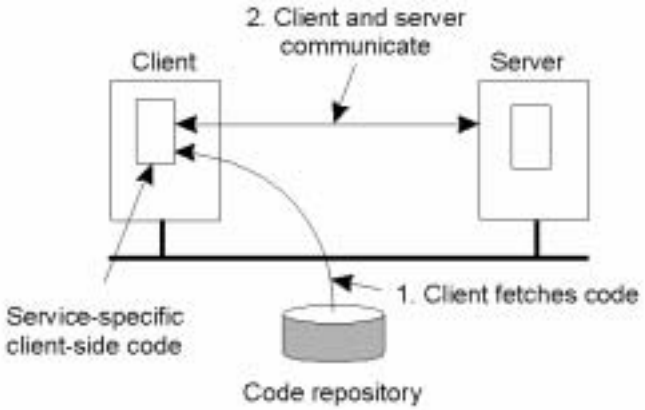
Code Migration

- Communication in distributed systems
  - **Passing data**
  - **Passing program**
    - **Code**
    - **Process**

Reasons for Migrating Code

- Performance
  - **Process migration: An entire process is moved from one machine to another**
    - **Costly and intricate**
    - **Overall system performance can be improved if processes on heavily-loaded machine are moved to lightly-loaded machine.**
  - **Migrating part of the client/server to the server/client**
    - **Database operations involve large quantities of data (C->S)**
    - **Interactive data applications: form processing (S->C)**
  - **Support for code migration can help improve performance by exploiting parallelism**
    - **Web searching: Having several small mobile programs move from site to site ->Linear speedup**
- Flexibility
  - **Traditional approach to building distributed applications is to partition the application into different parts during the design phase.**
  - **Code Migration makes it possible to dynamically configure distributed systems at run-time.**
    - **Dynamic downloading client software**
- Security?

Reasons for Migrating Code (cont.)



The principle of dynamically configuring a client to communicate to a server. The client first fetches the necessary software, and then invokes the server.

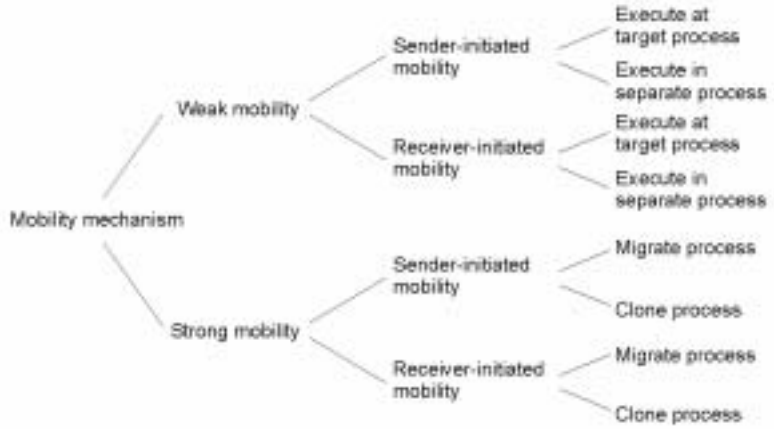
9

Models for Code Migration

- Code migration deals with moving program (process) between machines
  - **Execution status of a program**
  - **Pending signals**
  - **Other parts of the environements**
- Process
  - **Code segment**
  - **Resource segment**
  - **Execution segment**
- Weak Mobility and Strong Mobility
- Initiated party: sender or receiver

10

Models for Code Migration (cont.)



Alternatives for code migration.

11

Migration and Local Resources

- Resource segment cannot always be simply transferred along with the other segments without being changed.
  - **Socket binding to a specific TCP port**
  - **Open file descriptors**
- Process-to-resource bindings
  - **Binding by identifier**
  - **Binding by value**
  - **Binding by type**
- Resource-to-machine bindings
  - **Unattached**
  - **Fastened**
  - **Fixed**

12

Migration and Local Resources (cont.)

Resource-to machine binding

Process-to-resource binding	Unattached	Fastened	Fixed
By identifier	MV (or GR)	GR (or MV)	GR
By value	CP (or MV, GR)	GR (or CP)	GR
By type	RB (or GR, CP)	RB (or GR, CP)	RB (or GR)

Actions to be taken with respect to the references to local resources when migrating code to another machine.

13

Migration in Heterogeneous Systems

- Migrated code can be easily executed at the target machine when dealing with homogeneous systems
- However, for heterogeneous systems,
  - **Each has its own OS and machine architecture**
  - **Migration requires platform-level support**

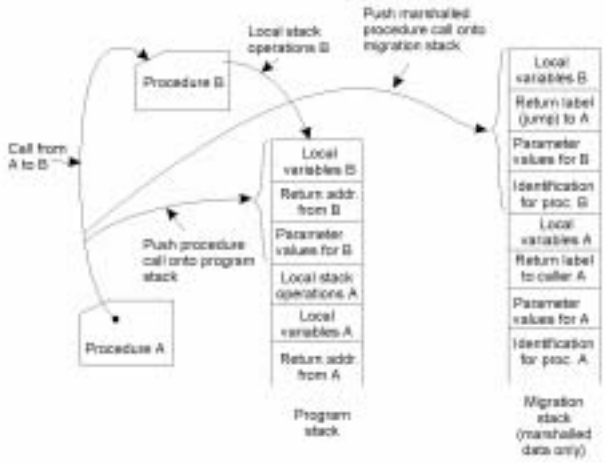
14

Migration in Heterogeneous Systems (cont.)

- Recompile the program at target machine
- Generate different code segments for each potential target platform
- Restrict to specific points in the execution of a program, e.g., next subroutine call.

  Migration stack
- Virtual Machine: Machine-independent intermediate code: Java

Migration in Heterogeneous Systems (cont.)



The principle of maintaining a migration stack to support migration of an execution segment in a heterogeneous environment

Overview of Code Migration in D'Agents (1)

```
proc factorial n {
   if ($n ≤ 1) { return 1; }                  # fac(1) = 1
  expr $n * [ factorial [expr $n – 1] ]       # fac(n) = n * fac(n – 1)
}
set number …        # tells which factorial to compute
set machine …       # identify the target machine

agent_submit $machine –procs factorial –vars number –script {factorial $number }

agent_receive …     # receive the results (left unspecified for simplicity)
```

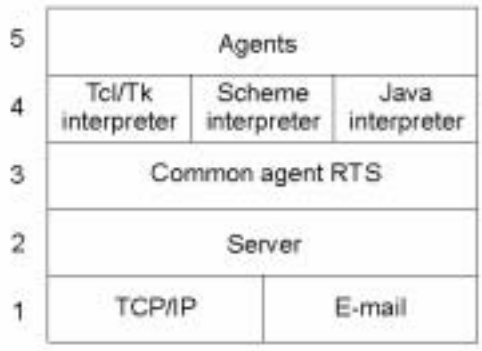
A simple example of a Tel agent in D'Agents submitting a script to a remote machine (adapted from [gray.r95])

Overview of Code Migration in D'Agents (2)

```
all_users $machines

proc all_users machines {
   set list ""                    # Create an initially empty list
   foreach m $machines {          # Consider all hosts in the set of given machines
      agent_jump $m               # Jump to each host
      set users [exec who]        # Execute the who command
      append list $users          # Append the results to the list
   }
   return $list                   # Return the complete list when done
}

set machines …                    # Initialize the set of machines to jump to
set this_machine                  # Set to the host that starts the agent

# Create a migrating agent by submitting the script to this machine, from where
# it will jump to all the others in  $machines.

agent_submit $this_machine –procs all_users
                                  -vars  machines
                                  -script { all_users $machines }

agent_receive …                   #receive the results  (left unspecified for simplicity)
```

An example of a Tel agent in D'Agents migrating to different machines where it executes the UNIX *who* command (adapted from [gray.r95])

Implementation Issues (1)



The architecture of the D'Agents system.

Implementation Issues (2)

Status	Description
Global interpreter variables	Variables needed by the interpreter of an agent
Global system variables	Return codes, error codes, error strings, etc.
Global program variables	User-defined global variables in a program
Procedure definitions	Definitions of scripts to be executed by an agent
Stack of commands	Stack of commands currently being executed
Stack of call frames	Stack of activation records, one for each running command

The parts comprising the state of an agent in D'Agents.

Software Agents

- Software agents: autonomous units capable of performing a task in collaboration with other remote agents.
  - **Reacting to/initiating changes in its environment**
  - **Collaborating with users and other agents**
- Mobile agents
- Interface agents
- Information agents

21

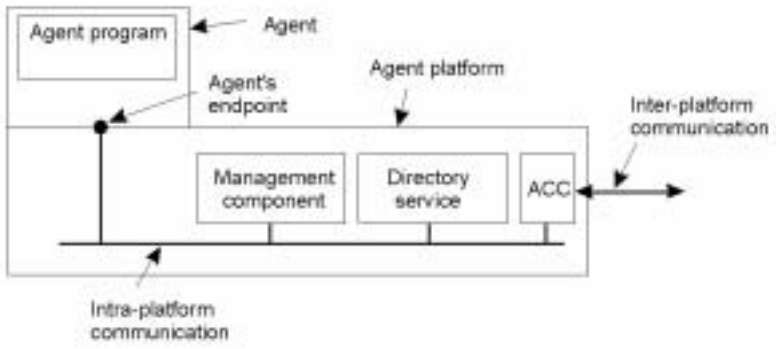
Software Agents in Distributed Systems

Property	Common to all agents?	Description
Autonomous	Yes	Can act on its own
Reactive	Yes	Responds timely to changes in its environment
Proactive	Yes	Initiates actions that affects its environment
Communicative	Yes	Can exchange information with users and other agents
Continuous	No	Has a relatively long lifespan
Mobile	No	Can migrate from one site to another
Adaptive	No	Capable of learning

Some important properties by which different types of agents can be distinguished.

22

Agent Technology



The general model of an agent platform (adapted from [fipa98-mgt]).

23

Agent Communication Languages (1)

Message purpose	Description	Message Content
INFORM	Inform that a given proposition is true	Proposition
QUERY-IF	Query whether a given proposition is true	Proposition
QUERY-REF	Query for a give object	Expression
CFP	Ask for a proposal	Proposal specifics
PROPOSE	Provide a proposal	Proposal
ACCEPT-PROPOSAL	Tell that a given proposal is accepted	Proposal ID
REJECT-PROPOSAL	Tell that a given proposal is rejected	Proposal ID
REQUEST	Request that an action be performed	Action specification
SUBSCRIBE	Subscribe to an information source	Reference to source

Examples of different message types in the FIPA ACL [fipa98-acl], giving the purpose of a message, along with the description of the actual message content.

24

Agent Communication Languages (2)

Field	Value
Purpose	INFORM
Sender	max@http://fanclub-beatrix.royalty-spotters.nl:7239
Receiver	elke@iiop://royalty-watcher.uk:5623
Language	Prolog
Ontology	genealogy
Content	female(beatrix),parent(beatrix,juliana,bernhard)

A simple example of a FIPA ACL message sent between two agents using Prolog to express genealogy information.

Any Questions?

See you next time.