COP 6611 Advanced Operating System

Processes

Chi Zhang
czhang@cs.fiu.edu

Outline

- Processes and Threads
- Clients
- Servers
- Code Migration
- Software Agents

2

Introduction to Threads

- A process is a program in execution
  - Program counter, CPU registers, memory maps …
  - Requires hardware support
  - High cost of creation and switching
- A thread has less overhead (CPU context)
  - Efficient Inter-thread communication.
    - Protect inappropriate accesses of shared data
  - Overlap blocking and non-blocking threads
  - Parallelism with multiple CPUs
  - Better programming structure

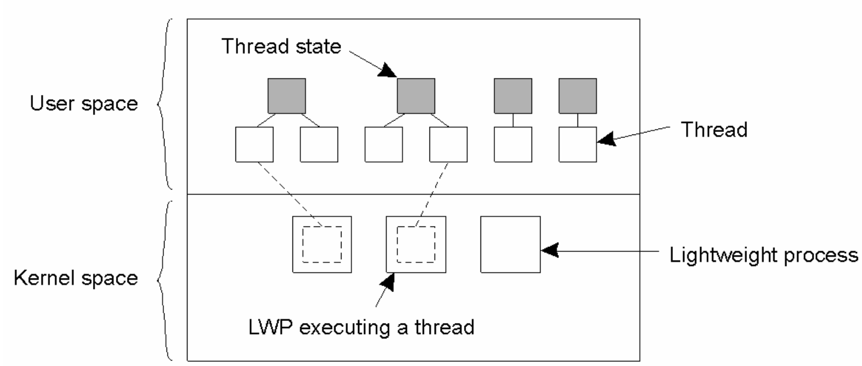
3

Thread Implementation (1)

- User-level threads
  - Cheap to create and destroy threads
  - Cheap to switch threads
    - Occurs through synchronization
  - Blocking system call blocks all threads.
  - Can't utilize multiple CPUs
- Kernel-level threads
  - System call is expensive!
- Hybrid form: Lightweight Process (LWP)
  - Kernel is aware of LWPs, but not threads
  - LWPs search for runnable threads

4

Thread Implementation (2)



Combining kernel-level lightweight processes and user-level threads.

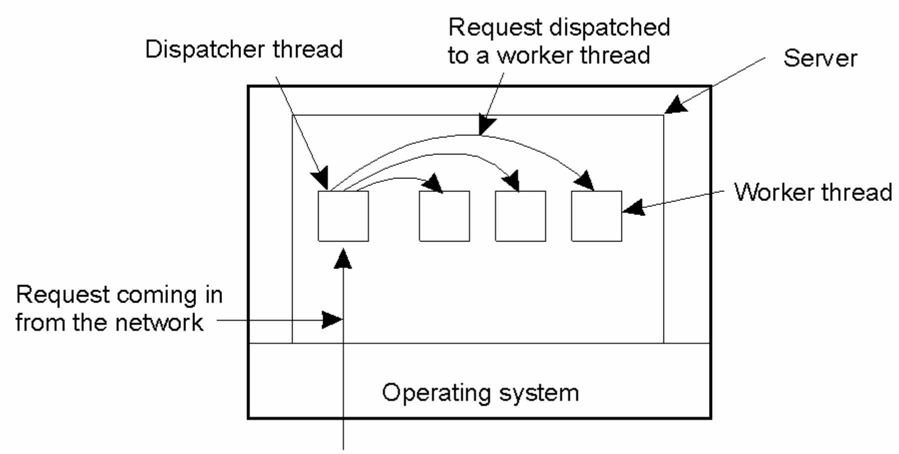
5

Multithreaded Clients

- Display the data before the communication completes
  - Hide communication latencies
- Separate threads for fetching different parts of the HTML page
  - Faster
  - TCP Connections may be set up to different replicas
  - Simple programming

6

Multithreaded Servers (1)



A multithreaded server organized in a dispatcher/worker model.

7

Multithreaded Servers (2)

Model	Characteristics
Threads	Parallelism, blocking system calls
Single-threaded process	No parallelism, blocking system calls
Finite-state machine	Parallelism, nonblocking system calls

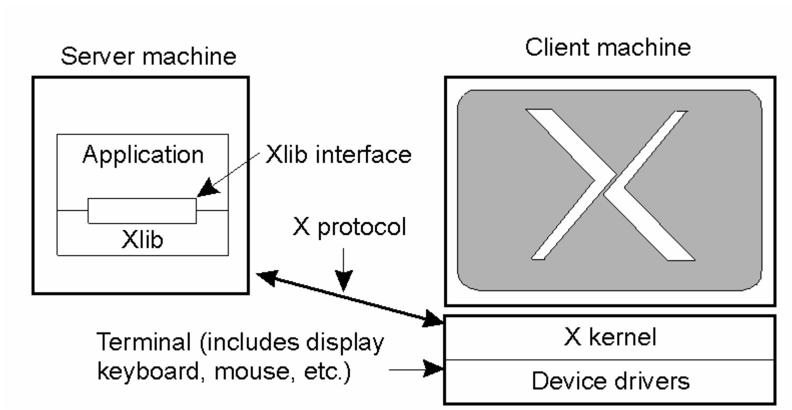
Three ways to construct a server.

Blocking system calls ⇨ make programming easier

Parallelism ⇨ improve performance

8

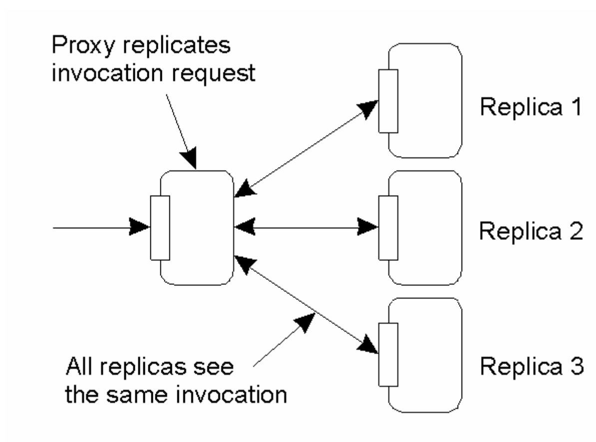
The X-Window System



The basic organization of the X Window System

9

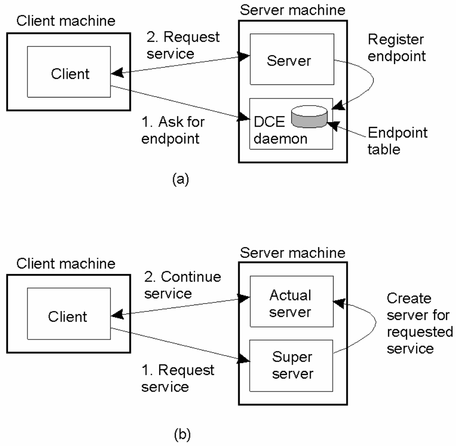
Client-Side Software for Distribution Transparency



A possible approach to transparent replication of a remote object using a client-side solution.

10

Servers: General Design Issues (1)



a) Client-to-server binding using a daemon as in DCE
b) Client-to-server binding using a superserver as in UNIX

Servers: General Design Issues (2)

- Handle communication interrupts
  - Exit the client application
  - Send out-of-band data to a separate control endpoint
  - Send out-of-band data with request data
- Stateful Server
  - Need to recover the states after a crash
  - Cookies in WWW

Object Servers

- Object Creation
  - At the first invocation request (destroy it if no clients are bound to it)
  - At the server initialization time.
- Threads for Objects
  - One for each object (No concurrent data access)
  - One for each request
- Threads Creation
  - Create on-demand
  - Thread pool

13

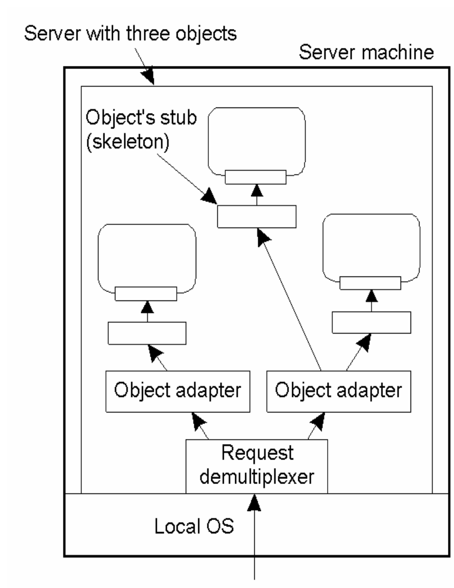
Object Adaptor (1)

- Object Adaptors
  - Group objects per policy
  - Unaware of the specific interfaces of the objects they control
- Now consider the policy of “one thread for each object”
  - Communications between threads takes place by means of buffers

14

Object Adapter (2)

Organization of an object server supporting different activation policies.



Object Adapter (3)

XXXX_invoke(unsigned in_size, char in_args[], unsigned* out_size, char* out_args[])

```
/* Definition of general message format */
struct message {
   long  source                  /* senders identity */
   long  object_id;              /* identifier for the requested object */
   long  method_id;              /* identifier for the requested method  */
   unsigned  size;               /* total bytes in list of parameters */
   char  **data;                 /* parameters as sequence of bytes */
};

/* General definition of operation to be called at skeleton of object */
typedef void (*METHOD_CALL)(unsigned, char* unsigned*, char**);

long register_object (METHOD_CALL call);        /* register an object  */
void unrigester_object (long object)id);        /* unrigester an object */
void invoke_adapter (message *request);         /* call the adapter */
```

The *header.h* file used by the adapter and any program that calls an adapter.

16

Object Adapter (4)

```
typedef struct thread THREAD;                  /* hidden definition of a thread */

thread *CREATE_THREAD (void (*body)(long tid), long thread_id);
/* Create a thread by giving a pointer to a function that defines the actual  */
/* behavior of the thread, along with a thread identifier  */

void get_msg (unsigned *size, char **data);
void put_msg(THREAD *receiver, unsigned size, char **data);
/* Calling get_msg blocks the thread until of a message has been put into its  */
/* associated buffer.  Putting a message in a thread's buffer is a nonblocking */
/* operation. */
```

The *thread.h* file used by the adapter for using threads.

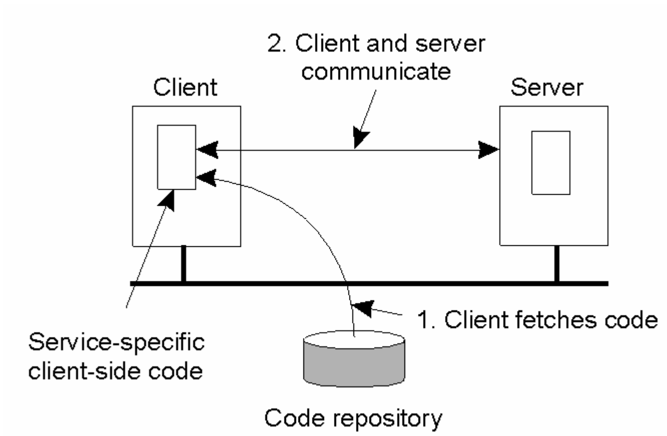
17

Object Adapter (5)

The main part of an adapter that implements a thread-per-object policy.

```
#include <header.h>
#include <thread.h>
#define MAX_OBJECTS        100
#define NULL               0
#define ANY                –1

METHOD_CALL invoke[MAX_OBJECTS];     /* array of pointers to stubs     */
THREAD *root;                        /* demultiplexer thread           */
THREAD *thread[MAX_OBJECTS];         /* one thread per object          */

void thread_per_object(long object_id) {
    message      *req, *res;                  /* request/response message  */
    unsigned     size;                        /* size of messages          */
    char         **results;                   /* array with all results    */

    while(TRUE) {
        get_msg(&size, (char*) &req);         /* block for invocation request */

        /* Pass request to the appropriate stub. The stub is assumed to      */
        /* allocate memory for storing the results.                          */
        (invoke[object_id]*)(req→size, req→data, &size, results);

        res = malloc(sizeof(message)+size);   /* create response message    */
        res→object_id = object_id;            /* identify object            */
        res→method_id = req.method_id;        /* identify method            */
        res→size = size;                      /* set size of invocation results */
        memcpy(res→data, results, size);      /* copy results into response */
        put_msg(root, sizeof(res), res);      /* append response to buffer  */
        free(req);                            /* free memory of request     */
        free(*results);                       /* free memory of results     */
    }
}

void invoke_adapter(long oid, message *request) {
    put_msg(thread[oid], sizeof(request), request);
}
```

Reasons for Migrating Code



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

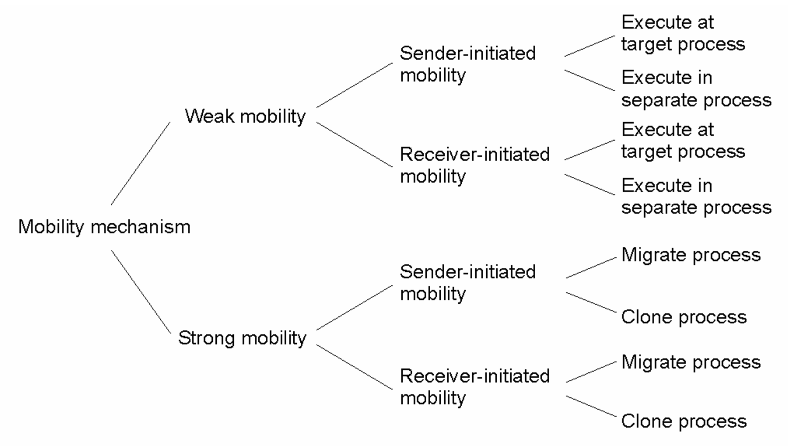
19

Migration and Local Resources

- A process consists of three segments
  - Code, resource and execution
- Three resource-to-machine bindings
  - Unattached
  - Fastened
  - Fixed
- Three process-to-resource bindings
  - Identifier
  - Value
  - Type

20

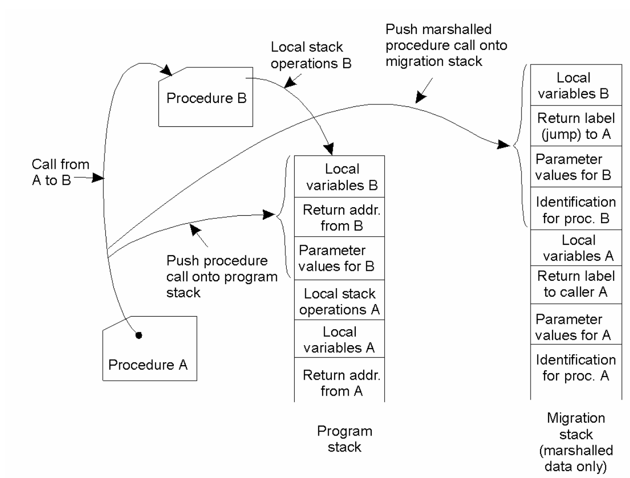
Models for Code Migration



Alternatives for code migration.

21

Migration in Heterogeneous Systems



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

22

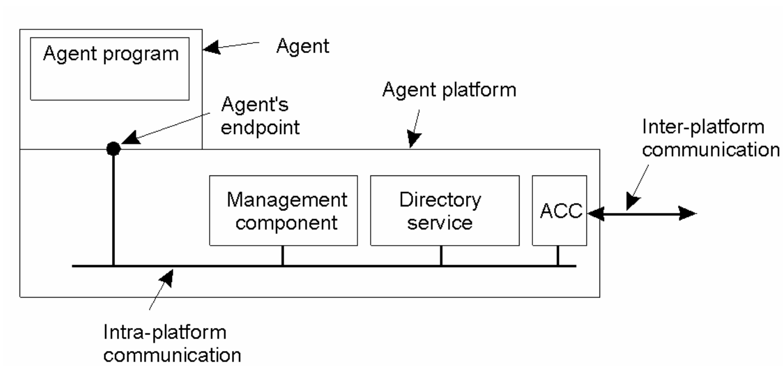
Software Agents in Distributed Systems

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.

23

Agent Technology



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

24

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.

25

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.

26