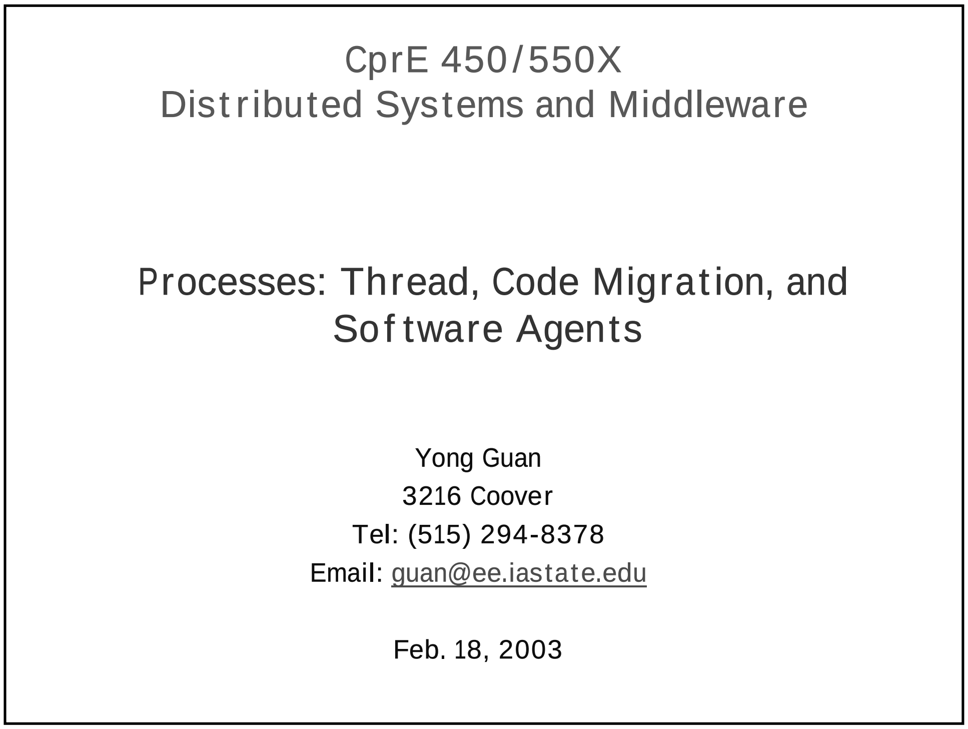
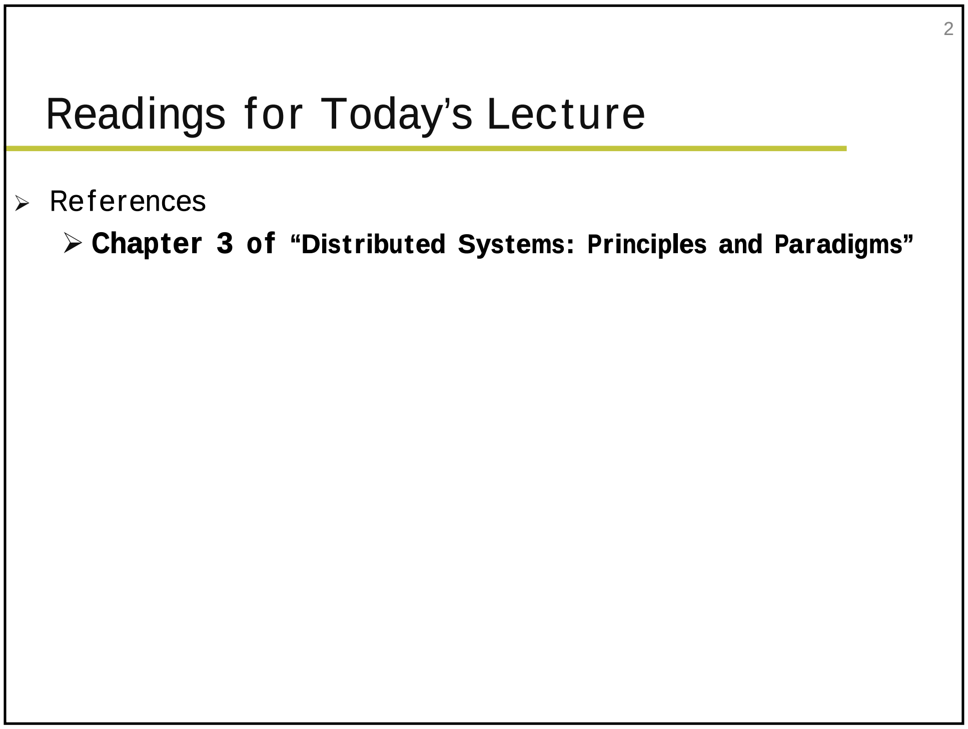


CprE 450/550X
Distributed Systems and Middleware

Processes: Thread, Code Migration, and Software Agents

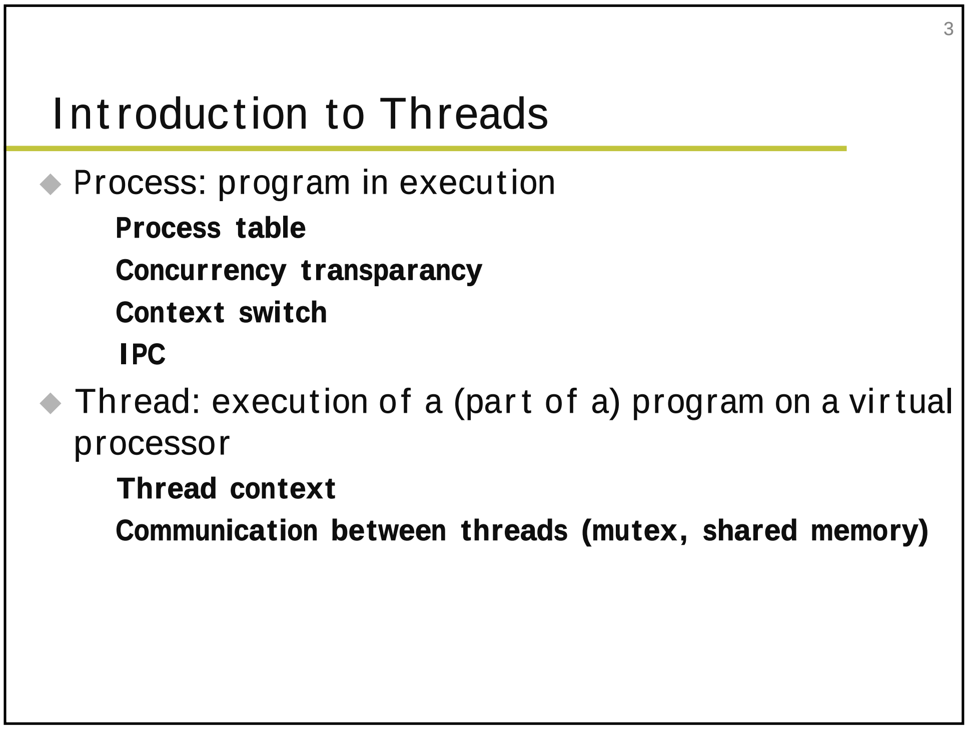
Yong Guan
3216 Coover
Tel: (515) 294-8378
Email: guan@ee.iastate.edu

Feb. 18, 2003



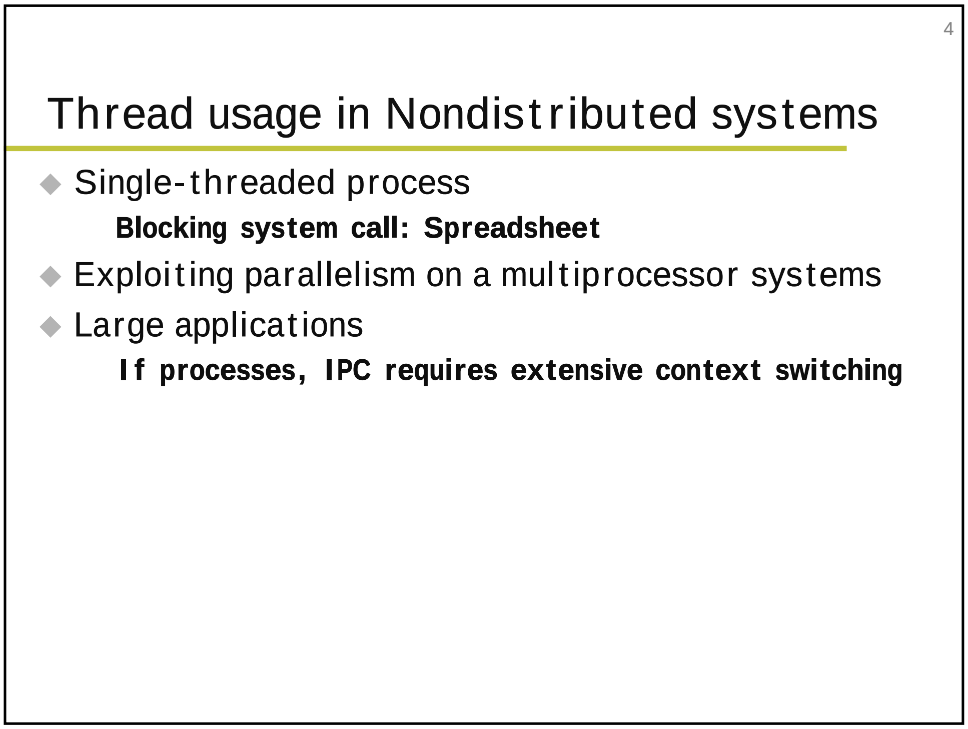
Readings for Today’s Lecture

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



Introduction to Threads

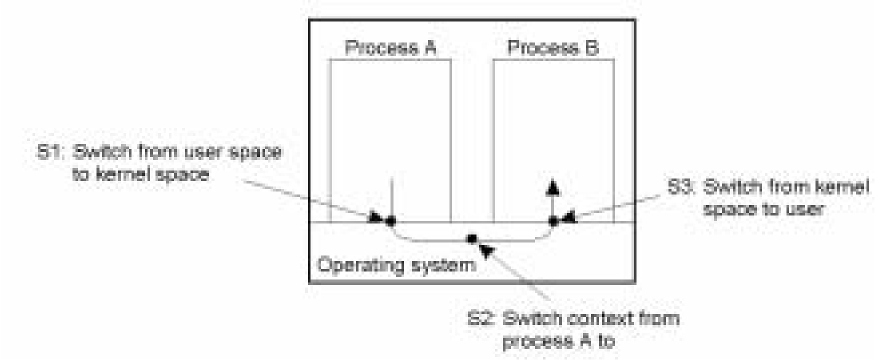
- Process: program in execution
  - **Process table**
  - **Concurrency transparancy**
  - **Context switch**
  - **IPC**
- Thread: execution of a (part of a) program on a virtual processor
  - **Thread context**
  - **Communication between threads (mutex, shared memory)**



Thread usage in Nondistributed systems

- Single-threaded process
  - **Blocking system call: Spreadsheet**
- Exploiting parallelism on a multiprocessor systems
- Large applications
  - **If processes, IPC requires extensive context switching**

Thread Usage in Nondistributed Systems

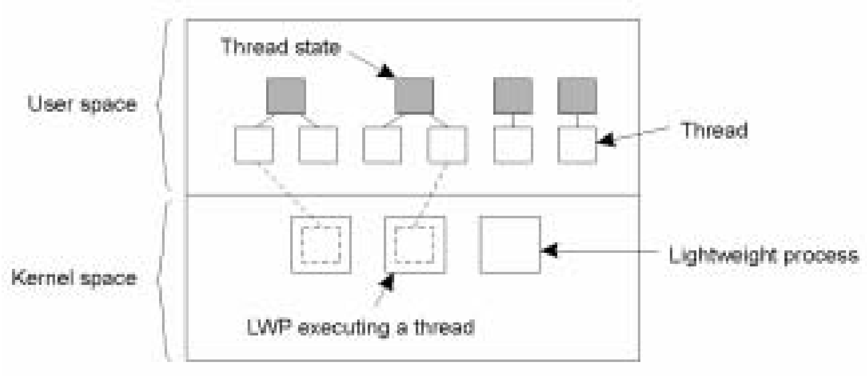

Process A
Process B
S1: Switch from user space to kernel space
S3: Switch from kernel space to user
Operating system
S2: Switch context from process A to

- Context switching as the result of IPC

Thread Implementation

- User-level thread library
  - **Easy to create and detroy**
  - **Easy to context switch: only a few instructions**
  - **Invoking a blocking system call blocks the entire process**
- Kernel thread implementation
  - **Benefits of thread disappears**
- Lightweight Process (LWP)
- Scheduler Activations (upcall to the thread package)
  - **Less elegant: violates the layered structure of the system**

Thread Implementation

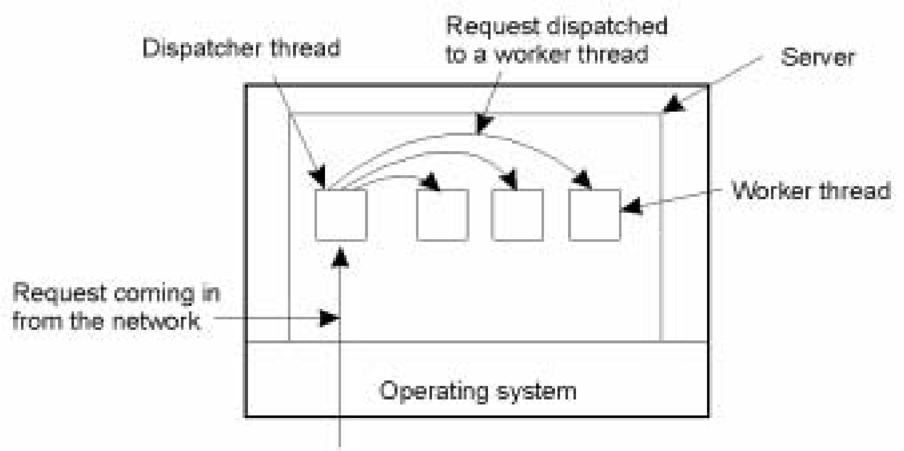


- Combining kernel-level lightweight processes and user-level threads.

Threads in Distributed Systems

- Multithread Clients
  - **Web browsers**

Multithreaded Servers (1)


Dispatcher thread
Request dispatched to a worker thread
Server
Worker thread
Request coming in from the network
Operating system

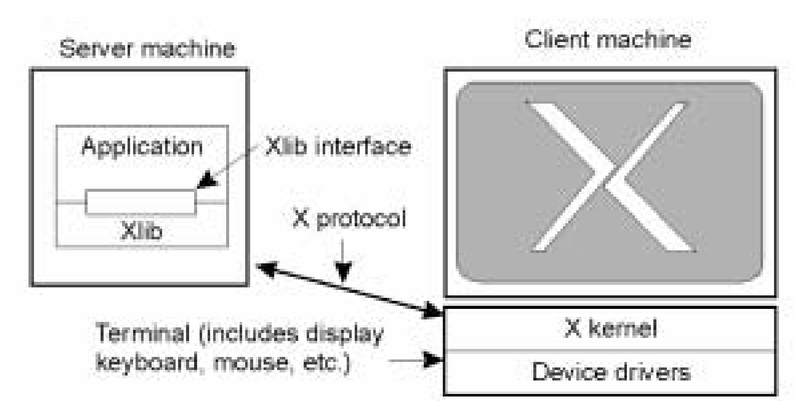
- A multithreaded server organized in a dispatcher/worker model.

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

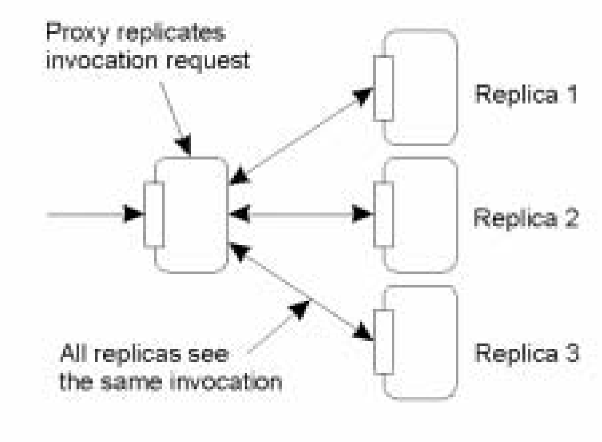
Three ways to construct a server.

The X-Window System



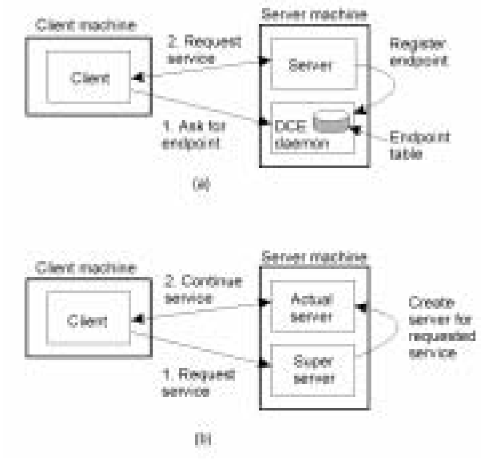
The basic organization of the X Window System

Client-Side Software for Distribution Transparency


Proxy replicates invocation request
Replica 1
Replica 2
Replica 3
All replicas see the same invocation

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

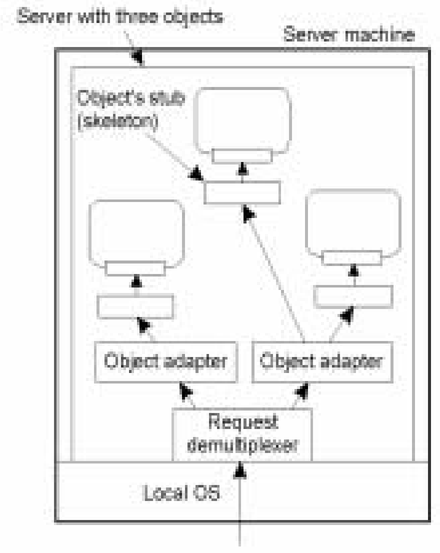
Servers: General Design Issues



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

Object Adapter (1)

Organization of an object server supporting different activation policies.



15

Object Adapter (2)

```
/* Definitions needed by caller of adapter and adapter */
#define TRUE
#define MAX_DATA 65536

/* 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

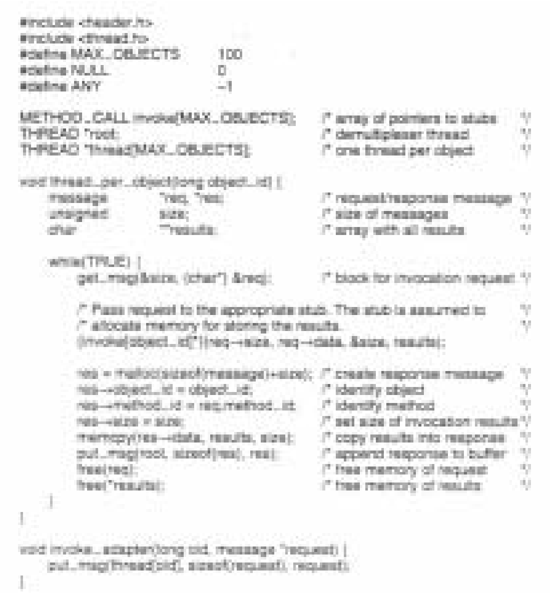
Object Adapter (3)

```
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.

Object Adapter (4)

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);
}
```