Process communication lecture 3

Shared resources

Particularly shared memory

- local
- remote

Transparency

Remote Procedure Calls - 16.3

Shared resources

Separate processes can alter the resource.

Need to check the state of the resource to either receive data or know that some event has occurred.

Usually need to explicitly coordinate access to the resource.

What if the information I want isn't there yet?

When do I try again?

Could combine with a semaphore or other event signalling method.

Which reminds me, before semaphores, UNIX had

- Lock files - if the file exists the resource is locked.
- Difficult to clean up if the owning process dies (worse than simple locks or semaphores - files are persistent).

Files: easy to use but slow.

OS may provide synchronization help, e.g. only one writer at a time.

Memory: fastest

Synchronization is usually handled explicitly by the processes.

Shared memory, threads and processes.

Different threads in the same process automatically share memory.

To have shared memory between different heavyweight processes:

Need system calls to

- define sections of shared memory,
- attach the shared memory to the process,
- detach, indicate who can access it etc.

Both processes need to know the name of the area of shared memory.

Must make sure the memory is attached to some unused area of the process's address space.

Usual security checks - can this process attach to this chunk of memory?

Just as we can pass messages with shared memory we can also share memory with messages.

Distributed shared memory

Shared memory between processes running on different machines.

Why?

Shared memory is a natural method to share information.

Processes don't need to be changed to run on a distributed system.

Why not?

Slow.

May need many messages to transfer the shared memory or parts of it across the network.

Extra complications to coordinate use of the shared memory.

How?

Copy the memory to whichever machine has a process sharing it.

Mark it read only.

If the process writes, memory access fault, kernel determines shared memory.

Sends a write request to originating machine, this can broadcast the change to other processors to update their copies.

Maybe only copy some of the shared memory - copy on read.

Simplified if only one process is allowed to write - readers/writers problem.

Shared memory only works with homogeneous processors (distributed or not).

Can get similar benefits with another approach.

Transparency

Shared memory is an attempt to simplify the system genie - remember, your understanding of the system.

A goal in distributed operating system design is to make the system look like one centralised system.

- Names
- Resources
- Processing

The distributed nature is *transparent* to a user. i.e. they can't see it, the user's system genie doesn't include a notion of multiple machines.

Remote Procedure Calls (RPC) - Birrell & Nelson 1984 - 16.3.1

If our message passing system uses blocking sends (until we get an acknowledgement of message receipt and possibly a result) it looks just like a procedure call.

Stubs

Since we want the call inside our program to look like a local call (and the service to look like a local call) something else has to do the work.

Client and Server stubs

The client procedure call goes to a stub associated with the process which packages up the request and sends it via the kernel to the server stub.

The stubs at both ends need to be constructed from the same interface specification - to ensure consistency.

Care has to be taken about different versions of the service. A different version may take slightly different parameters or return different types etc.

Messages are highly structured

- what procedure to execute
- parameters
- version number
- timestamp
- source address
- possibly the type of machine the request comes from

Where is the server end?

- include port numbers at compile time or
- have a binder or rendezvous (or matchmaker) service
  - server end registers
  - client end sends the request to find the server
  - one extra message before the RPC call itself

This allows multiple servers - binder can spread the load

Binder can periodically check on servers - deregistering those that don't respond

Binder could do the security work

- otherwise servers have to check each call

Dealing with parameters

There is nothing to stop us using RPC over heterogeneous networks.

Data formats in the different machines may be different.

- ASCII, Unicode, EBCDIC for strings
- Big or little endian for integers
- Different floating point formats

Stubs need to know about this.

This assembling of the parameters is known as *marshalling*.

Different approaches?

- client stub converts to the server format
- server stub converts from the client format
- convert to and from a canonical format
- no one needs to know other machines formats

(we don't want to have to convert from format A to canonical to A and then back again)

Reference parameter problem

The references no longer make sense.

either disallow reference parameters

or copy the parameter to and fro

- either in one chunk or whenever the server makes a change to it
- (this is just like distributed shared memory)

Copy/restore semantics aren't quite the same.

e.g. pointer as a reference parameter

the same parameter passed twice in the parameter list

(of course these are bad form anyway)

Sometimes we don't need to copy the data both ways

e.g. input buffer (only needs to be copied from the server to the client)

Because we only expect one send and one return message we can

use a connectionless protocol - LAN

- faster
- can fail or be duplicated
- server end should keep a history to prevent duplication

Assignment 3

handling events - override the following methods

```
public boolean mouseDown(Event evt, int x, int y)
public boolean mouseUp(Event evt, int x, int y)
public boolean mouseDrag(Event evt, int x, int y)
```

need to return true

```
public void paint(Graphics g)
```

Normal way of drawing in a Component - Frames are a subclass of Component

Look up class Graphics

```
g.setXORMode(Color.white);
g.drawLine(x0, y0, x1, y1);
g.setPaintMode();
```

To draw outside of paint use:

```
Graphics g = getGraphics();
```

and draw using this Graphics object.