Distributed Systems Course 335

Unassessed Coursework No. 2

Lecturer: M. Sloman

1a The following message passing primitives are supported by a set of library calls:

send (dest, msg) – an *asynchronous* send message primitive, where dest is the name of the process to which the message msg is to be sent.

receive (source, msg) – this causes the receiving process to block waiting for a message from the process with name source. msg is a buffer into which the incoming message is copied.

receiveany (source, msg) – the process is blocked waiting for a message from any source. The name of the sender is received in source and the incoming message is received in msg.

i) Explain what is meant by an *asynchronous* send message primitive and why it may lead to buffer exhaustion at the receiver.
ii) Explain why both the above receive and receiveany primitives are needed.

b Using the above message primitives, design a simple printer service for a distributed system with multiple printers, each controlled by a process called *printer*. There are also multiple users. When a *user* process wants to print a document it sends a message containing its process type (i.e. user) to a single *coordinator* process which allocates free printers. The coordinator replies with the name of a free printer when one is available, and the user send a page at a time to the printer. When a printer is free it sends a message containing its process type (i.e. printer) to the coordinator to indicate it is now available for printing.

Give pseudocode outlines for the *user*, *coordinator* and *printer* processes, using the above message primitives. Your solution should describe any datastructures needed by the coordinator process.

Assume the printer process has sufficient buffer space for a single message containing one page to print, and that communication is reliable so timeouts and retransmissions can be ignored.

2a What is the *exported interface* and *imported interface* of a component in a distributed system. Briefly outline the functions performed by a Remote Procedure Call (RPC) support system when a server *exports* an interface and when a *binding* takes place between a client and a server?

b Explain why the implementation of a Remote Procedure Call (RPC) requires a stub *procedure* for both client and server and explain the functions performed by both client and server stubs.

Distributed Systems Course 335
Unassessed Coursework No. 2 Solution

Lecturer: M. Sloman

1a i) Asynchronous send: this is an unblocked send in which the sender process sends the message and continues once the message has been copied out of its address space. It does not know when or if the message is received by the destination. One particular sender (or multiple processes) may send messages to a receiver at a rate faster than the receiver can process the messages. Each message has to be buffered at the receiver while waiting to be processed – this can lead to buffer overflow if the receiver cannot process messages fast enough as there will be a finite number of buffers.

ii) The receive permits the receiver to selectively receive messages from a single source. A server process such as a file server does not know which sources will be sending requests so needs the ReceiveAny to be able to receive a message from any source.

1b) User: // when ready to print

```
send (coordinator, userName, userProc)
receive (coordinator, pn)  // get name (pn) of free printer
loop
        send (pn, nextpage)
        receive(pn, ack)
until EOF  //end of file
send (pn, EOF)
```

Coordinator

```
procnames:   a linked list of process names
//either user processes waiting for a printer or available printers
// assume 2 procedures addtail (name), removehead (name)
printersavailable:  Boolean  // true indicates   printers on procnames

set procnames to null
printersavailable := false
loop {
        receiveany (source, proctype)
        if (proctype = printer)  then {  // printer request
             if  (printersavailable = false) & (procnames != null)  then {
                  removehead (name)  //users Q'd
                  send (source, name)  //send UserNname to printer
                  send (name, source) // send printerName to 1st user on Q }
             else {  addtail (source) ; printersavailable := true}/  // no users waiting so Q printer
        }
        if (proctype != printer) & (printersavailable = true)  then { // message from user
                removehead (name)  // remove printer from Q
                if procnames = null then printersavailable := false
                send (source, name) // send printerName to user  }
                send (name, source) // send userName to 1st printer on Q }
        if (proctype != printer) & (printersavailable != true)  then // message from user
                {addtail (source)  }  // Q user
        }
```

Printer

```
loop {
        send (coordinator, printerName, printerProc)   //printer available
        receive ( coordinator, name)  //get user name
        loop {
                receive (name, msg)
                print (msg)
                send (name, ack)
                } until msg = eof
        }
```

2a **Exported interface** defines parameters and procedure names of set of procedures implemented by server.

Imported interface defines parameters and procedure names of set of remote procedures called by client.

Export: Server registers itself with a name server as offering service defined by exported interface.
Provides its address and exported interface type.

Bind: Client queries name server to find suitable instance of server which has exported interface corresponding to imported interface type.
Check for type compatibility, & that server still available & exporting interface.
Obtain address of server to use for RPC calls.

2b RPC stub: This is the outline procedure which acts as the local representative of the remote procedure. There is a stub for each remote procedure called at the client and one at the server for each procedure it provides which can be called by a remote client.

Client stub marshalls call parameters into a message, performs any transformation to cater for heterogeneity and accesses the transport layer communication primitive to send the message to the remote node.

Eventually receives response message, performs transformations, unpacks message into return parameters and performs local return to client process.

Server stub receives message from local dispatcher, transforms representations, unpacks message into calling parameters, makes local call on client procedure. When client returns, it marshals return parameters, performs transformations and sends message to remote client.