Interprocess communication

Vaidė Narváez

Computer Information Systems

June 21th, 2010

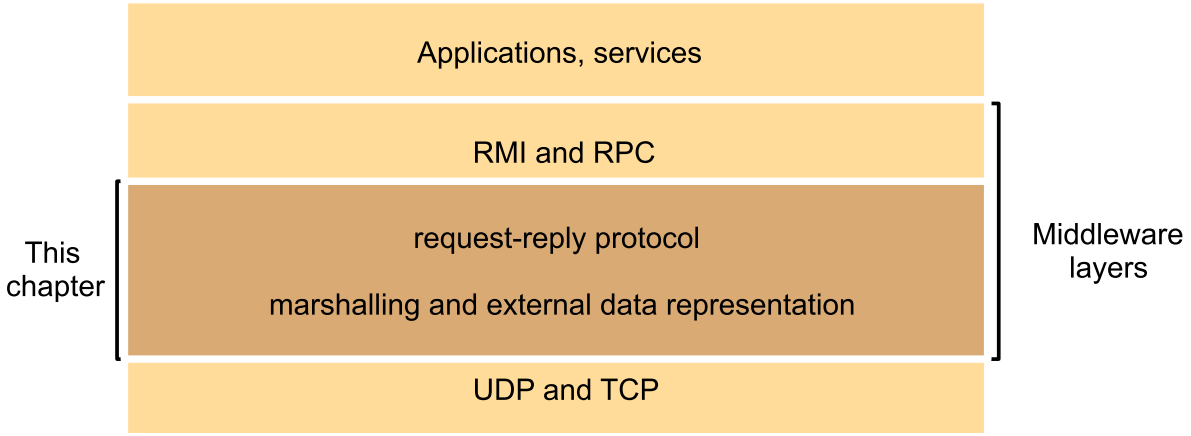
© 2010 Vaidė Narváez

©Pearson Education 2005

protocols for communication

© 2010 Vaidė Narváez

Interprocess communication



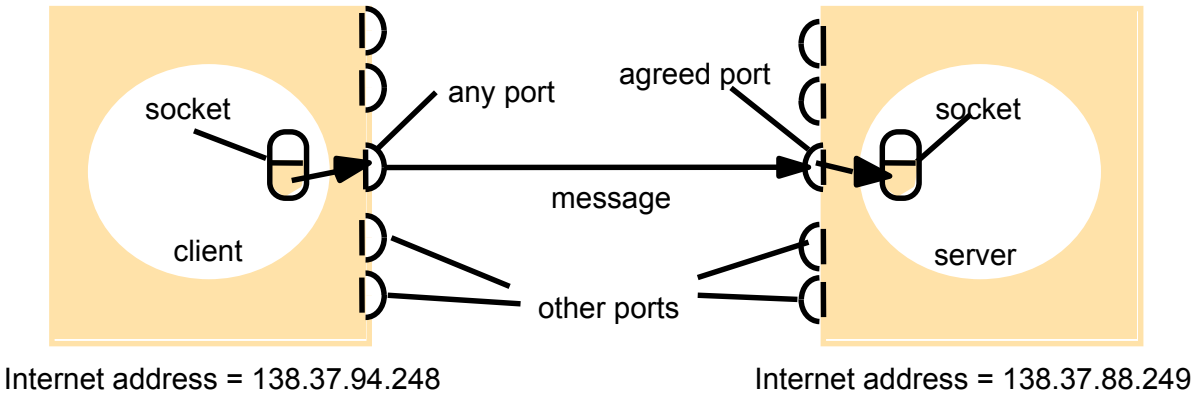
©Addison-Wesley 2005

Interprocess communication

- Message passing: *send* and *receive*
- Synchronous and Asynchronous communication
  - Synchronous: both the sending and the receiving processes synchronize at every message (blocking)
  - Asynchronous: *send* is non-blocking, *receive* can be either, today mostly blocking
- Message destinations
- Reliability
- Ordering

© 2010 Vaidė Narváez

Sockets



©Addison-Wesley 2005

```
InetAddress aComputer =
InetAddress.getByName("ic.payap.ac.th");
```

UDP datagrams communication

- Message size: up to 2^{16} bytes, usually 8K
- Blocking: non-blocking *send* and blocking *receive*
- Timeouts: forbid blocking receive operation wait indefinitely
- Receive from any: *receive* does not specify origin for messages
- Failure model:
  - omission failures: can be dropped
  - ordering: can be out of order
- Use of UDP
  - DNS
  - less overhead

© 2010 Vaidė Narváez

Java API for UDP

© 2010 Vaidė Narváez

TCP stream communication

- Message size: unlimited
- Lost messages: sequence no, ack, retransmit after timeout of no ack
- Flow control: sender can be slowed down or blocked by the receiver
- Message duplication and ordering: sequence no
- Message destination: establish a connection, one sender-one receiver, high overhead for short communication

© 2010 Vaidė Narváez

TCP stream communication

- Message size: unlimited
- Lost messages: sequence no, ack, retransmit after timeout of no ack
- Flow control: sender can be slowed down or blocked by the receiver
- Message duplication and ordering: sequence no
- Message destination: establish a connection, one sender-one receiver, high overhead for short communication
- Matching of data items: two processes need to agree on format and order (protocol)
- Blocking: non-blocking send, blocking receive (send might be blocked due to flow control)
- Concurrency: one receiver, multiple senders, one thread for each connection
- Failure model: checksum, sequence #, ack

© 2010 Vaidė Narváez

Java API for TCP

© 2010 Vaidė Narváez

External Data Representation

- Different ways to represent int, float, char... (internally)
- Byte ordering for integers
  - big-endian: most significant byte first
  - small-endian: least significant byte first
- Standard external data representation
  - marshal before sending, unmarshal before receiving
- Send in sender's format and indicates what format, receivers translate if necessary
- External data representation
  - CORBA's Common Data Representation (CDR)
  - Java's object serialization
  - ASCII (XML, HTTP)

© 2010 Vaidė Narváez

External Data Representation: CDR

- support both big-endian and little-endian
- transmitted in sender's ordering and the ordering is specified
- receiver translates if needed

Type	*Representation*
sequence	length (unsigned long) folllowed by elements in order
string	length (unsigned long) followed by characters in order (can also can have wide characters)
array	array elements in order (no length specified because it is fixed)
struct	in the order of declaration of the components
enumerated	unsigned long (the values are specified by the order declared)
union	type tag followed by the selected member

©Addison-Wesley 2005

CDR: example

index in sequence of bytes	← 4 bytes →	*notes on representation*
0–3	5	*length of string*
4–7	"Smit"	*'Smith'*
8–11	"h "	
12–15	6	*length of string*
16–19	"Lond"	*'London'*
20-23	"on "	
24–27	1934	*unsigned long*

The flattened form represents a *Person* struct with value: {'Smith', 'London', 1934}

©Addison-Wesley 2005

Java serialization

- serialization and de-serialization are automatic in arguments and return values of Remote Method Interface (RMI)

- flattened to be transmitted or stored on the disk
  - write class information, types and names of instance variables
  - new classes, recursively write class information, types, names...
  - each class has a handle, for subsequent references
  - values are in Universal Transfer Format (UTF)

© 2010 Vaidė Narváez

Java serialization

```
public class Person implements Serializable {
        private String name;
         private String place;
         private int year;

         public Person(String aName, String aPlace, int aYear){
            name = aName;
            place = aPlace;
            year = aYear;
        }
}
```

Serialized values				*Explanation*
Person	8-byte version number		h0	*class name, version number*
3	int year	java.lang.String name:	java.lang.String place:	*number, type and name of instance variables*
1934	5 Smith	6 London	h1	*values of instance variables*

The true serialized form contains additional type markers;
h0 and h1 are handles/references to other objects within the serialized form

©Addison-Wesley 2005