Distributed Systems
ECEG-6504

Inter-process Communication

Surafel Lemma Abebe (Ph. D.)

Topics

- Introduction
- APIs for Internet protocols
- External data representation and marshalling
- Client-server communication
- Group communication

Introduction

- Inter-process communication is at the heart of all DSs
- Communication in DSs is based on message passing offered by the underlying network
- Modern DSs consist of thousands of processes scattered across an unreliable network
- Unless the primitive communication facilities of the network are replaced by more advanced ones, development of large scale DSs becomes extremely difficult

Introduction…

- Communication models for DSs:
  - Remote Procedure Calls
  - Remote Method Invocation
  - Message-oriented communication
  - Stream-oriented communication

APIs for Internet Protocols

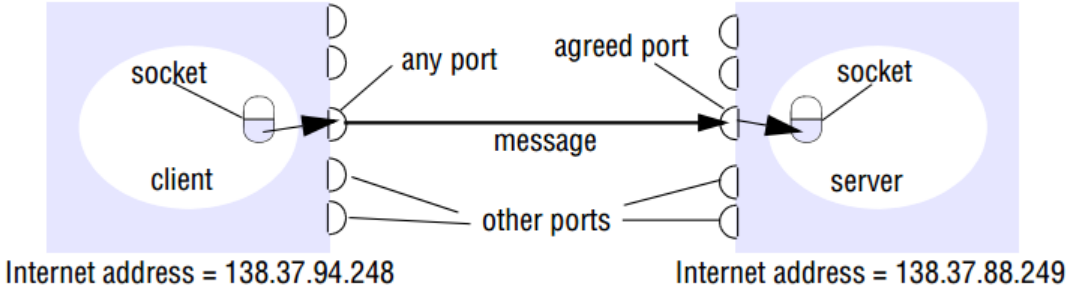
- Characteristics of inter-process communication
  - Message communication operations
    - Send
    - Receive
  - Synchronous communication
    - Sending and receiving processes synchronize at every message
    - Both send and receive are blocking operations
    - Send
      - Sending process is blocked until the corresponding receive is issued
    - Receive
      - Blocks until a message arrives, when a receive is issued
  - Asynchronous communication
    - Send operation is non-blocking
    - Receive operation can be blocking or non-blocking
      - Non-blocking variant
        - » Issues receive operation and proceeds with its program

APIs for Internet Protocols…

- Characteristics of inter-process communication…
  - Message destination
    - Messages are sent to (internet address, local port) pairs
    - Usually, port has one receiver but could have many senders
    - Fixed addresses to services => not location transparent
      - Soln.
        - » Client programs refer to services by name and use a name server or binder to translate their names into server location
  - Reliability
    - Validity and integrity
  - Ordering
    - Some applications require that messages be delivered in sender order

APIs for Internet Protocols…

- Socket
  - Endpoint for communication between processes
  - Could be used to send and receive messages by processes
  - Each socket is associated with a particular protocol
  - For a process to receive a message, its socket must be bound to a local port and internet address of the computer on which it runs
  - Port cannot be shared by processes
  - A process could use multiple ports to receive message



APIs for Internet Protocols…

- UDP datagram communication
  - A datagram sent by UDP is transmitted from a sending process to a receiving process without ack or retries
  - Socket must be created to send or receive a message
  - Issues related to UDP datagram communication
    - Message size
      - If message is too big, its truncated
      - IP protocol restricts message size (e.g. 8kb) => fragmentation
    - Blocking
      - Sockets usually provide non-blocking send and blocking receive for datagram communication
      - Messages are discarded at destination, if no process has a socket bound to the destination port

APIs for Internet Protocols…

- UDP datagram communication…
  - Issues…
    - Timeouts
      - Receive that blocks is suitable for use by a server that is waiting to receive requests from its clients
      - What if the sending process crashed or expected message is lost?
        - » Set timeouts on sockets
    - Receive from any
      - Receive method does not specify an origin for messages
      - Receive method returns the Internet address and local port of the sender
      - Its also possible to send/receive to/from a particular address and port only

APIs for Internet Protocols…

- Failure model for UDP datagrams
  - Reliable communication
    - Integrity is assured using checksum
  - UDP datagrams suffer from the following failures:
    - Omission failures
    - Ordering
  - Applications using UDP should provide their own checks to achieve reliable communication
- Uses of UDP
  - UDP is sometimes preferred
    - No overhead associated with guaranteed message delivery, i.e.,
      - No need to store state information at the source and destination
      - No transmission of extra messages
      - No latency for sender

APIs for Internet Protocols…

- TCP stream connection
  - API to the TCP protocol provides abstraction of a stream of bytes to which data may be written and from which data may be read
  - Characteristics hidden by the stream abstraction
    - Message sizes
      - Application can choose how much data it writes to a stream or reads from it
    - Lost messages
      - TCP uses an acknowledgement scheme
    - Flow control
      - TCP attempts to match the speed of the processes that read from and write to a stream
    - Message duplication and ordering
      - Message identifiers are associated with each IP packet

APIs for Internet Protocols…

- TCP stream connection …
  - Characteristics …
    - Message destination
      - A pair of communicating processes establish a connection before they can communicate over a stream
      - To send a message, no need to specify address and port after a connection is established
      - Establishing a connection involves
        - » A connect request from a client to a server
        - » An accept request from server to client

APIs for Internet Protocols…

- TCP stream connection…
  - When establishing a connection, API for stream communication assumes:
    - One process plays the client role
      - Creates a socket bound to any port
      - Make a connection request to a server port
    - Another process plays the server role
      - Create a listening socket bound to a server port
      - Wait for clients to request connections
      - Listening socket maintains a queue of incoming connections requests
  - When a server accepts a connection, a new stream socket is created for the server to communicate with the client
  - Pair of sockets in the client and server are connected by a pair of streams, one in each direction (input stream and output stream)

APIs for Internet Protocols…

- TCP stream connection…
  - Issues related to stream communication
    - Matching of data items
      - Two communicating processes need to agree as to the contents of the data transmitted over a stream
    - Blocking
      - Data written to a stream is kept in a queue at the destination socket
      - While reading data, process blocks until data is available in the queue
      - Process reads/writes data from/to a queue
    - Threads
      - When a server accepts a connection, it generally creates a new thread in which to communicate with the new client
      - Advantage of using a separate thread
        - » Server can block when waiting for input without delaying other clients
      - Alternative method, if thread is not supported
        - » Test whether input is available from a stream before attempting to read it

APIs for Internet Protocols…

- TCP stream connection…
  - Failure model
    - Reliable communication
      - Integrity
        - » Checksums
        - » Sequence numbers
      - Validity
        - » Timeouts and retransmission
    - If packet loss over a connection passes some limit, or network is severely congested or detached
      - No ack is received
      - It declares the connection is broken

APIs for Internet Protocols…

- TCP stream connection…
  - Failure model…
    - Effects of a broken connection
      - Process using the connection cannot
        - » Distinguish between network failure and failure of the process at the other end of the connection
        - » Tell whether the message they have sent recently have been received or not

External data representation and marshalling

- Passing parameters or messages when calling a method or procedure may be problematic
  - Why?
    - Messages consists of sequences of bytes, i.e., data structures must be flattened
    - Problem
      - Individual primitive data items transmitted in messages can be data values of many different types
      - Different computers might store primitive values in different order
  - Interoperability issue
    - Different set of codes in different machines
      - ASCII character coding – one byte per character
      - Unicode – two bytes per character

External data representation and marshalling…

- Interoperability issues…
  - Example: 3 (0000 0000 0000 0000 0000 0000 0000 0011) in memory

    big-endian vs Little-endian

Little-Endian approach	Big-Endian Approach
00000011	0000000
00000000	0000000
00000000	0000000
00000000	0000011

    => Not all computers store primitive values in the same order
  - Representation of floating-point numbers also differs between architectures

External data representation and marshalling…

- How to enable two computers exchange data?
  - Values are transmitted in the sender's format, together with an indication of the format used, and the recipient converts the values if necessary
  - Convert values to an agreed external format before transmission and convert to local format on receipt

- External data representation
  - Agreed standard for the representation of data structures and primitive values

External data representation and marshalling…

- Marshalling
  - Process of taking a collection of data items and assembling them into a form suitable for transmission in a message

    => Translate **to** external data representation
  - Requires consideration of all the finest details of the representation
    - Could be error prone if carried out by hand

- Unmarshalling
  - Process of disassembling them on arrival to produce an equivalent collection of data items at the destination

    => Translate **from** external data representation

External data representation and marshalling…

- Approaches to external data representation and marshalling
  - CORBA's common data representation
  - Java's object serialization
  - XML (Extensible Markup Language)

External data representation and marshalling…

- CORBA's common data representation (CDR)
  - Defined with CORBA 2.0
  - Can represent all of the data types that can be used as arguments and return values in remote invocations in CORBA
  - Primitive types
    - 15 primitive types
    - Defines representation for both big-endian and little-endian
    - Values are transmitted in the sender's ordering
  - Constructed types

Type	*Representation*
sequence	length (unsigned long) followed by elements in order
string	length (unsigned long) followed by characters in order (can also 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

External data representation and marshalling…

- CORBA's CDR…
  - Example:

    struct person{'Smith', 'London', '1984'}

    (each character occupies one byte – assumption)

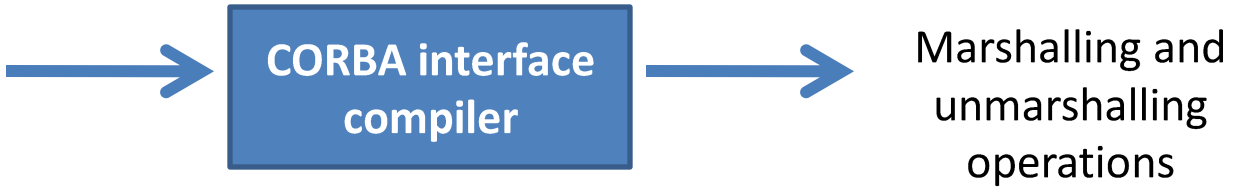
index in sequence of bytes	← 4 bytes →	*notes on representation*
0–3	5	*length of string*
4–7	"Smit"	*'Smith'*
8–11	"h___"	
12–15	6	*length of string*
16–19	"Lond"	*'London'*
20–23	"on__"	
24–27	1984	*unsigned long*

  - Sender and recipient have common knowledge of the order and types of the data items to be transmitted in a message
    - Type of data items is not given

External data representation and marshalling…

- CORBA's CDR…
  - Marshalling in CORBA
    - Marshalling operations are generated from the specification of the types of data items to be transmitted in a message
    - Types of the data structures and basic data items are described **in CORBA IDL**

```
struct Person {
    string name;
    string place;
    long year;
};
```



External data representation and marshalling…

- Java object serialization
  - Refers to the activity of flattening an object or a connected set of objects into a serial form
  - Platform specific

```
public class Person implements Serializable {
   private String name;
   private String place;
   private int year;
   public Person(String nm, place, year) {
      nm = name; this.place = place; this.year = year;
   }
   // more methods
}
```

External data representation and marshalling…

- Java object serialization…
  - Deserialization
    - Consists of restoring the state of an object or set of objects from their serialized form
    - Has no prior knowledge of the types of the objects in the serialized form
      - Information about the class of each object is included in the serialized form
        - » Name of the class and a version number
  - All objects referenced in an object are also serialized
    - References are serialized as handles
    - There must be a 1 to 1 correspondence between object references and handles
    - Each object must be written only once

External data representation and marshalling…

- Java object serialization…
  - The serialized object holds Class information as well as object instance data
  - There is enough class information passed to allow Java to load the appropriate class at runtime
  - It may not know before hand what type of object to expect
  - Example:
    - struct person{‘Smith’, ‘London’, ‘1984’}

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*
1984	5 Smith	6 London	h1	*values of instance variables*

External data representation and marshalling…

- Extensible markup language (XML)
  - Defined by W3C
  - Readable by humans
  - Designed for writing structured documents for the web
  - XML data items are tagged with ‘markup’ strings
  - Use of tags in XML is different from that of in HTML
  - Tags are used to
    - describe logical structure of the data
    - Associate attribute-value pairs with logical structures

```
<person id="12345678">
    <name>Smith</name>
    <place>London</place>
    <year>1984</year>
</person>
```

External data representation and marshalling…

- Extensible markup language (XML)…
  - Extensible
    - Users can define their own tags
  - To be used by more than one application
    - There has to be agreement on the names of tags
  - XML namespaces
    - Used for scoping names
  - Example:

```
<p:person p:id="123456789" xmlns:p="http://www.aait.edu.et/nsOne">
    <p:name>Smith</p:name>
    <p:place>London</p:place>
    <p:year>1934</p:year>
</p:person>
```

External data representation and marshalling…

- Extensible markup language (XML)…
  - XML schemas
    - Defines
      - Elements and attributes that can appear in a document
      - How the elements are nested
      - Order and number of elements
      - Type and default value
    - May be shared by many different documents
    - Example:

```
<xsd:schema  xmlns:xsd = URL of XML schema definitions >
    <xsd:element name= "person" type ="personType" />
        <xsd:complexType name="personType">
            <xsd:sequence>
                <xsd:element name = "name"  type="xs:string"/>
                <xsd:element name = "place"  type="xs:string"/>
                <xsd:element name = "year"  type="xs:positiveInteger"/>
            </xsd:sequence>
            <xsd:attribute name= "id"   type = "xs:positiveInteger"/>
        </xsd:complexType>
</xsd:schema>
```

Client-server communication

- Described in terms of the send and receive operations
- Request-reply communication is
  - Synchronous
    - Because the client process blocks until the reply arrives from the server
  - Reliable
    - Because the reply from the server is effectively an ack to the client
- Asynchronous request-reply communication
  - An alternative that may be useful in situations where clients can afford to retrieve replies later

- A protocol built over datagrams avoids unnecessary overheads associated with the TCP stream protocol
  - Acknowledgements are redundant, since requests are followed by replies
  - Establishing a connection involves two extra pairs of messages in addition to the pair required for a request and a reply
  - Flow control is redundant for the majority of invocations, which pass only small arguments and results

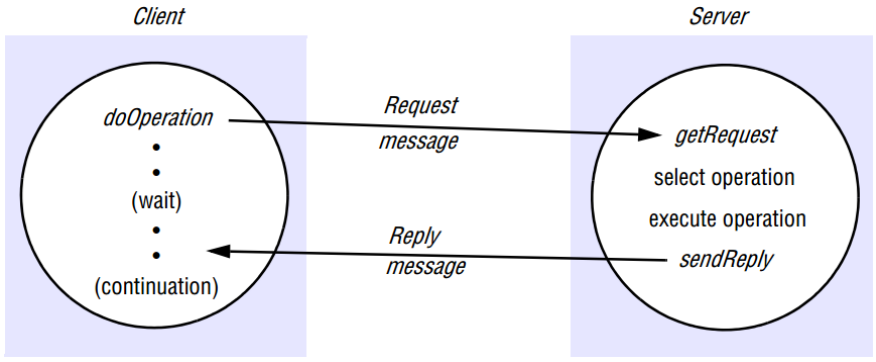
Client-server communication…

- Request-reply protocol
  - Based on

public byte[] doOperation (RemoteRef s, int operationId, byte[] arguments)
Sends a request message to the remote server and returns the reply.
The arguments specify the remote server, the operation to be invoked and the arguments of that operation.

public byte[] getRequest ();
Acquires a client request via the server port.

public void sendReply (byte[] reply, InetAddress clientHost, int clientPort);
Sends the reply message *reply* to the client at its Internet address and port.



Request-reply message structure

messageType	*int (0=Request, 1= Reply)*
requestId	*int*
remoteReference	*RemoteRef*
operationId	*int or Operation*
arguments	*// array of bytes*

Client-server communication…

- Request-reply protocol…
  - Management of message using message identifier
  - Message identifier consists of
    - A requestId
      - Taken from an increasing sequence of integers by the sending process
      - Makes the identifier unique to the sender
    - An identifier for the sender process
      - E.g., its port and internet address
      - Makes the identifier unique to the distribution system
  - What happens when requestId reaches the max value for an unsigned integer?
    - It is reset to zero
    - Lifetime of a message identifier should be much less than the time taken to exhaust the values in the sequence of integers

Client-server communication…

- Request-reply protocol…
  - Failure model
    - If implemented over UDP, *doOperation*, *getRequest* and *sendReply*
      - Would suffer from omission failures
      - Would not guarantee messages to be delivered in sender order
    - A process could also crash
    - doOperation uses timeout, in case of a server failure or message lost
      - What doOperation can do upon timeout
        - » Return immediately from doOperation with a failure indication
        - » doOperation sends the request message repeatedly until either it gets a reply or is reasonably sure that the delay is due to lack of response from the server

      => Duplicate messages

Client-server communication…

- Request-reply protocol…
  - Failure model…
    - Discard duplicate request messages
      - The protocol is designed to recognize successive messages (from the same client) with the same request identifier and to filter out duplicates
    - Lost reply messages
      - Server receives a duplicate request
      - Some servers can execute their operations more than once
        - » Idempotent operation
      - Some need to take special measures to avoid executing its operation more than once

Client-server communication…

- Request-reply protocol…
  - Failure model…
    - History
      - Used by servers that require retransmission of replies without re-execution of operation
      - Contains a record of (reply) messages that have been transmitted
      - Entry contains
        - » Request identifier,
        - » A message, and
        - » An identifier of a client to which its sent
      - Has a memory cost
        - » Request of a client could be interpreted as an ack
      - Problems
        - » Large number of clients
          => big volume of reply messages
        - » Client process terminates without acknowledging the last reply
          - Messages must be discarded after a limited period of time

Client-server communication…

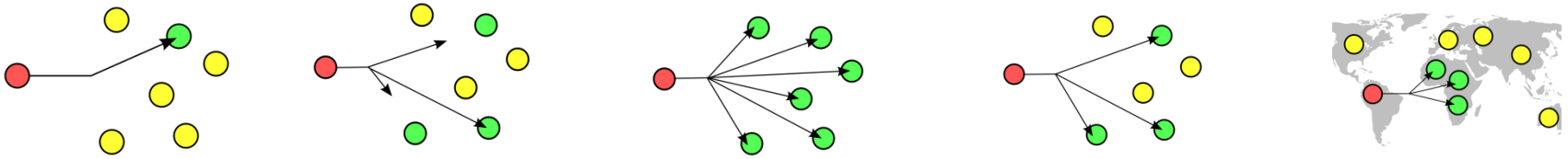
- Request-reply protocol…
  - Failure model…
    - Styles of exchange protocols
      - Produce differing behaviors in the presence of communication failures
      - Request (R) protocol
        - » Implemented over UDP
        - » Suffers from same communication failures
      - Request-reply (RR) protocol
        - » Communication failure could be masked by the retransmission of requests with duplicate filtering and saving of replies in a history for retransmission
      - Request-reply-acknowledge reply (RRA) protocol
        - » Enables to discard entries (lower than requestId) from its history
        - » Loss of an acknowledgement message is harmless

Name	*Messages sent by*		
	Client	*Server*	*Client*
R	*Request*		
RR	*Request*	*Reply*	
RRA	*Request*	*Reply*	*Acknowledge reply*

Group communication

- The pair wise exchange of messages is not the best model for communication from one process to a group of other processes

- Multicast operation
  - Sends a single message from one process to each of the members of a group of processes
  - Membership of a group is transparent to the sender
  - A computer belongs to a multicast group if one or more processes have sockets that belong to the multicast group
  - Simplest form provides no guarantees about message delivery or ordering



Group communication…

- Multicast messages provide a useful infrastructure for constructing DS with the following characteristics:
  - Fault tolerance based on replicated services
    - A replicated service consists of a group of servers
    - Client requests broadcasted to all members to perform the same operation
  - Discovering services in spontaneous (ad-hoc) networking
    - Multicast messages can be used to
      - Locate available discovery services in order to register their interfaces
      - Look up the interfaces of other services
  - Better performance through replicated data
    - Data is replicated to increase the performance of a service
  - Propagation of event notifications
    - Notify processes when something happens

Group communication…

- IP multicast
  - Built on top of the Internet Protocol (IP)
  - Allows the sender to transmit a single IP packet to a set of computers that form a multicast group
  - The sender is unaware of the identities of the individual recipients and of the size of the group
  - Specified by a class D internet address
    - 224.0.0.0 to 239.255.255.255
  - IP packets can be multicast both on local network and on the Internet
    - Local multicast uses local network such as Ethernet
    - Internet uses multicast routers
      - Time to live (TTL)
        - » Limits the distance of propagation of a multicast datagram

Group communication…

- IP multicast…
  - Multicast address allocation
    - Managed by Internet Assigned Numbers Authority (IANA)
    - Permanent addresses
      - Exist when there are no members
    - Temporary addresses
      - Cease to exist when all the members have left the group
      - Requires a free multicast address to avoid accidental participation in an existing group
      - To avoid accidental participation in an existing group, it sets TTL to a small value
  - Datagrams multicast over IP multicast have the same failure characteristics as UDP datagrams
    - Unreliable multicast

Group communication…

- Failure in IP multicast
  - Multicast router failure prevents all recipients beyond it from receiving the message
    - A datagram sent may be lost
    - Process on the router could fail
  - On LAN, recipient may drop the message b/c its buffer is full
  - Members of a group could receive the same array of messages in different orders

Group communication…

- Effects of reliability and ordering
  - Fault tolerance based on replicated services
    - Requires either all of the replicas or none of them should receive a request to perform an operation in the same order
    - Consistency
  - Discovering services in spontaneous networking
    - An occasional lost request is not an issue when discovering services
  - Better performance through replicated data
    - Effect of lost messages and inconsistent ordering would depend on
      - Method of replication
      - Importance of all replicas being totally up-to-date
  - Propagation of event notifications
    - Application determines the qualities required for muticast
- Reliable multicast or unreliable multicast?
  - Depends on the application's requirements