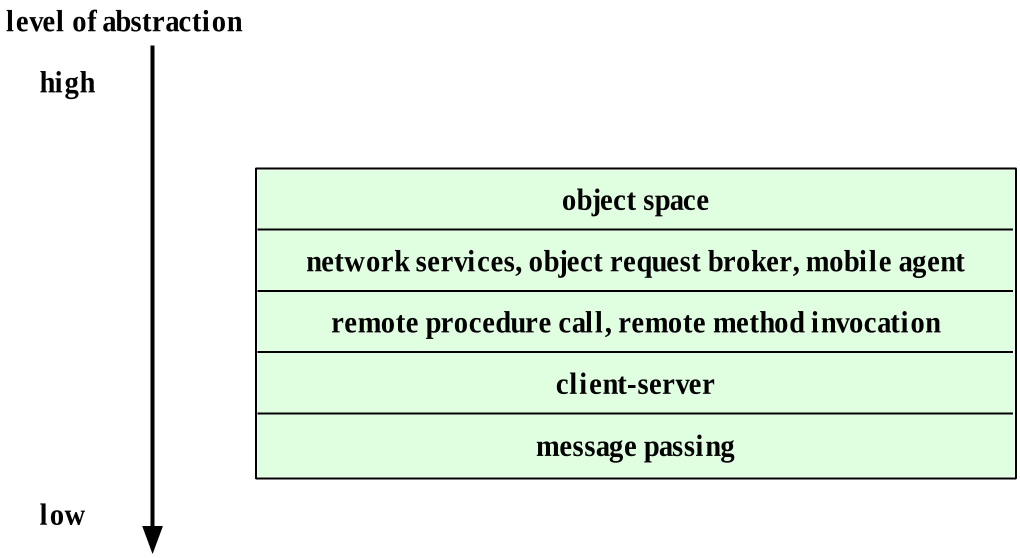
SFDV3006
Concurrent Programming

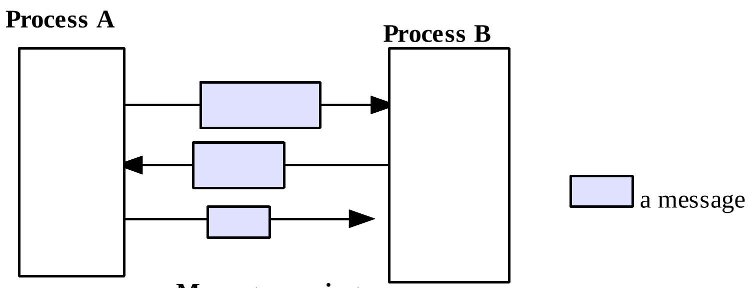
Lecture 9 – Message Passing

Distributed Application Paradigms



The Message Passing Paradigm

- Message passing is the most fundamental paradigm for distributed applications.
  - A process sends a message representing a request.
  - The message is delivered to a receiver, which processes the request, and sends a message in response.
  - In turn, the reply may trigger a further request, which leads to a subsequent reply, and so forth.



Message passing

Message Passing Paradigm

- The basic operations required to support the basic message passing paradigm are *send*, and *receive*.
- For connection-oriented communication, the operations *connect* and *disconnect* are also required.
- With the abstraction provided by this model, the interconnected processes perform input and output to each other, in a manner similar to file I/O. The I/O operations encapsulate the detail of network communication at the operating-system level.
- The *socket* application programming interface is based on this paradigm.

Absence of shared memory

- In previous lectures interaction between threads is via shared memory
  - In Java, object references to shared memory
- Usually encapsulated in monitors
- In a distributed environment shared memory does not exist
- Communication is achieved via passing messages between concurrent / parallel threads

Message Passing overview

- Main operations
  - send
  - receive
- Synchronization
  - Synchronous
  - Asynchronous
  - Rendezvous
- Multiplicity
  - one-one
  - many-one
  - many-many

- *Anonymity*
  - *anonymous message passing*
  - *non-anonymous message passing*

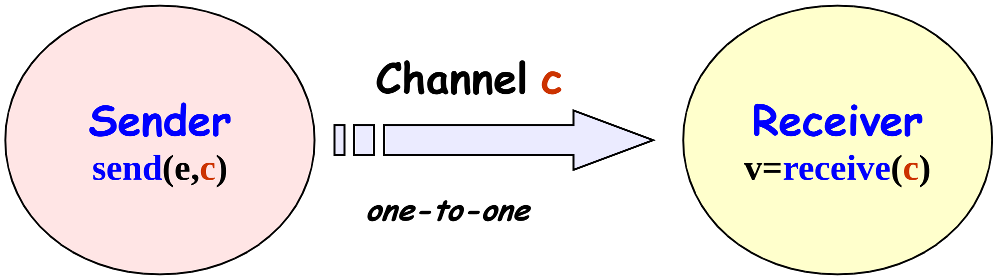
Synchronous and Asynchronous Message Passing

- Synchronous Message Passing: sender of a message blocks until it has been received
- Asynchronous Message Passing: sender does not wait and messages which have been sent but not received are buffered
- Synchronous and Asynchronous are both one-way form of communication – messages are transmitted in one direction only from sender to receiver.
- Rendezvous – two way message passing protocol used for client-server interaction

Synchronous Message Passing

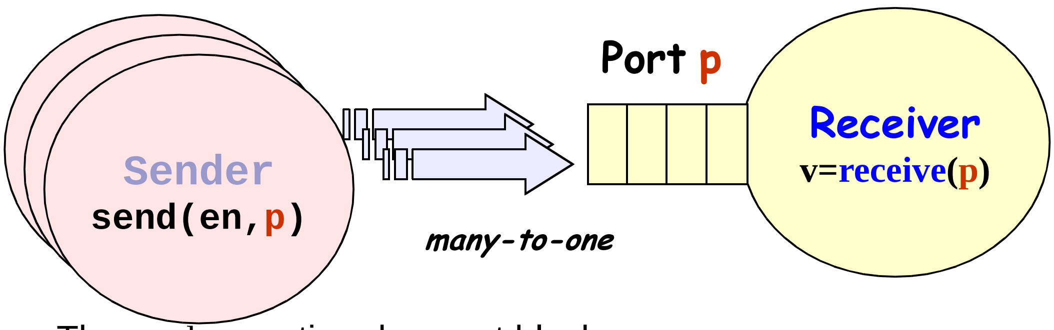
- Messages can be addressed directly to the destination process or indirectly to some intermediate entity.
- Messages are sent to and received from channels
- A channel connects two and only two processes.
- A single process can send to the channel and a single process can receive from the channel. This is one-one communication.

Synchronous Message Passing - contd



- send(e, c): Send e to channel c. Sending process is blocked until channel received e.
- v=receive(c): receive into a local variable v from channel c. The calling process is blocked until a message is sent into the channel
- The above operations do not require messages to be buffered

Asynchronous Message Passing



- The send operation does not block
- Messages which have been sent but not received are held in the message queue.
- Senders add messages to the tail of the queue and receivers remove message from the head.
- Many senders send messages to a **port** but only a single receiver may receive messages from it.
- Many-to-one communication

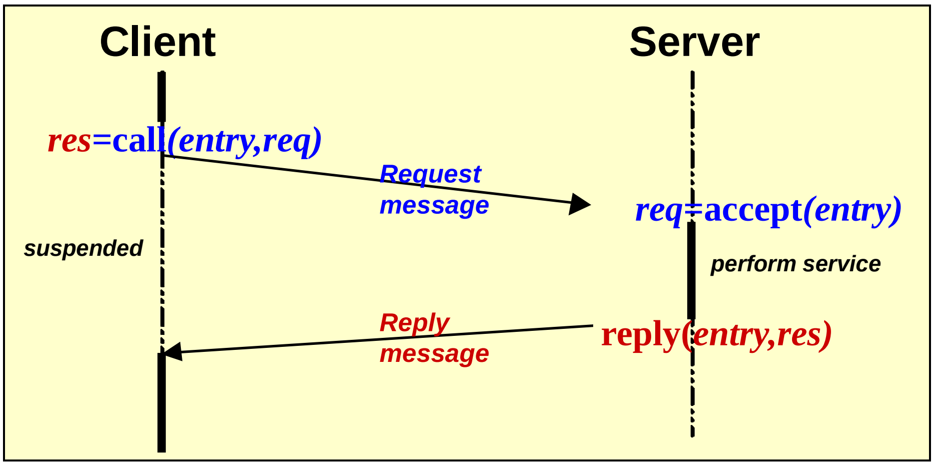
Asynchronous Message Passing - contd

- Port: conceptually an unbounded FIFO queue of messages
- Port is also known as mailbox
- Two operations:
  - send(e, p): send value e to port p. Calling process not blocked.
  - v=receive(p): receive value into variable v from port p. Calling process is blocked if no value is queued to port

Rendezvous Message Passing

- Also called request-reply
- Used to support client-server interaction.
- Client processes send request messages to a server process requesting the server to perform to some service.
- The requests are queued to an **entry** in FIFO order.
- The server accepts requests from an entry and on completion of the requested service sends reply message to client.
- Many-one communication. Many clients may request service from a single server.
- The reply to request is one-one communication.

Rendezvous



Socket programming in Java or any other language is a good example of rendezvous. Other examples include RPC, XML-RPC, RMI etc

Producer Consumer example

- In synchronous message passing the Consumer acts like a server which receives items from Producer
- Notice that there is no buffer
- Producer has to communicate directly with the consumer
- Implementation using sockets – with object serialization to send the item to the consumer
- Consumer blocks until it receives from the producer
- Producer must know consumer host and port
-

Producer Consumer example

- In asynchronous message passing there is no direct communication between the Producer and the Consumer
- Communication through a mailbox/port
- Producer deposits item into the mailbox
- Consumer receives from the mailbox
- Communication between the mailbox and consumer can be implemented using Announcer – Listener architecture
- Producer, consumer and mailbox could be on different machines
- Messages are held in a message queue in the mailbox
- Very widely used and implemented, scalable
- Basis for MOM (Message Oriented Middleware)
- IBM MQ Series is an example of a widely used messaging server