COMP90015 Distributed Systems

Protocols

School of Computing and Informati

2022 Semester II

1. Exchange Protocols
   - Request Protocol
   - Request/Reply Protocol
   - Re

2. Remote Invocation
   - Remote Procedure Call
   - Remote Method Invocation

Communication Failure

Let's say we have a client and a server and we use a reliable stream communication protocol like TCP to send requests from the client to the server. Consider the case where the client writes a request (e.g. in the form of a JS ters) to the socket ocket however a an the client kno e server?

What should the client do in this case?

> *Under extreme failure conditions, e.g. network ou nite period of time, every communication protocol either blocks for an indefinite period of time or eventually times out and fails by raising an exception to the application.*

Requests, Responses and Acknowledgements

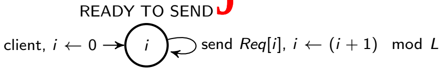
- Exchange protocols are fundamental building blocks of more complicated protocols. They describe how a sender and receiver, or e.g. a client and server, can exchange messages in a systematic way. Usually we talk about the client sendin … he server providi … ase when the server re

- For the p … otocols we will use *sequence numbers* $0, 1, 2, \ldots, L-1$. The value of L can be deter … we may only need two sequence numbers, 0 and 1, or we may n … .

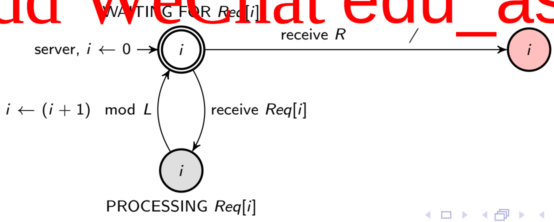
- Request with sequence number i from the clie … $[i]$.
- Response to $Req[i]$ will be written as $Rsp[i]$
- Acknowledgement of $Rsp[i]$ will be written as $Ack[i]$.

Send a sequence of requests without expecting replies

The client's sender protocol is modelled as below, which is a FSM with L states (shown in compact form), each representing the current message identifier i that is being sent.

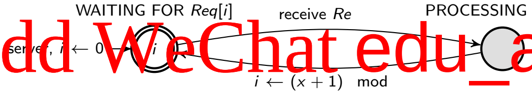


The serve … error state is ent … ge is received. The error state raises an exception to the serve … the communication protocol has failed to operate as expe…



Handling errors

With the previous protocol example there is no way for the sender to know that the receiver is in error. The sender will simply continue to send new requests. ol to recover or tolerate cessing the next rece e x may or may not eq



In fact in this case we really do not care about sequence numbers at all.

"Maybe" semantics

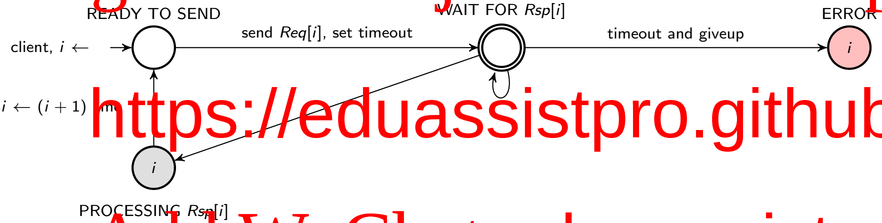
The simple Request protocol provides no guarantees to the client which is called *maybe* semantics. Maybe the request was processed by the server, maybe it w ... If the sequence ... eds to assume th ... uests, in any order, e.g.

$$Req[0], Req[1], Req[3], Req[4], Req[2], \ldots$$

The distributed system must be able to operate correct ... eak guarantees, otherwise it must use a different protocol.

Ensuring requests are processed in sequence

To ensure the sequence of requests is processed in the same order as sent, the client needs a response to each request and cannot send the next request until the response for the currently sent request has been received.



Ensuring sequence is *synchronous* – it does not be sent until it has received a response for the current request. Ensuring that the request has been processed may be impossible. It may eventually give up and the protocol is then in error (exception raised to the client), or it may continue to timeout and retry forever, which *blocks* the client from sending more requests.

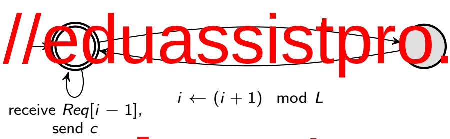
"At least once" semantics and idempotent requests

- Waiting for a response and retrying if no response is received within a certain time is guaranteeing that the request was processed by the receiver *at least once* or else an error occurs.
- If the server's receiver protocol is the same as earlier, the server may process the same request more than once. This may lead to an error. There are genera
  - state ... e is the answ
  - state ... ount. State
    - E.g. *withdraw*$(accountId, 5)$ and the response is *newBalance*. This modifies the state of the account.
  - For stateless requests, processing the same request multiple t ... o an error but will waste resources at the server.
  - For stateful requests, if the request is *idempote* ... not lead to an error. E.g. a request like *setBala* ... le times does not lead to an error, but a request like *deposit*$(accountId, 5)$ will lead to an error if executed erroneously multiple times.

For non-idempotent requests we would like the protocol to ensure that each request is processed only once.

"Exactly once" semantics

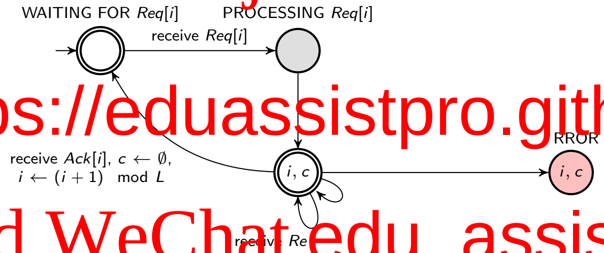
Since the protocol has introduced the possibility of duplicate requests, the receiver must be able to remove duplicate requests. Note that the sender will not send $Req[i+1]$ until it has received $Rsp[i]$ – the server will never "miss" requests, and so such an error condition never arises.



If a duplicate $Req[i]$ was received, it must be that th receive $Rsp[i]$. Instead of the server reprocessing the duplicate request, keeping a copy of the response any simply resending it can be done by the protocol. The server does not reprocess the duplicate request. In this case the protocol is providing *exactly once* semantics.

Ensuring cached *Rsp*[*i*] can be deleted

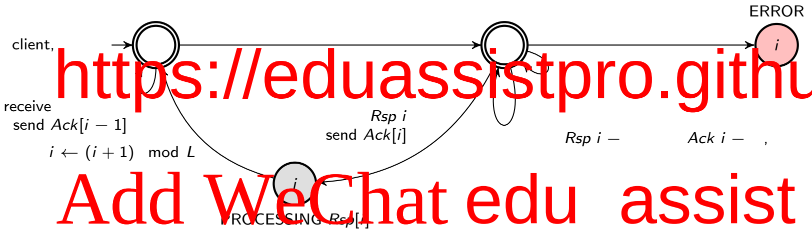
The server does not want to cache *Rsp*[*i*] indefinitely as it consumes resources. The receiving protocol can require the receipt of *Rsp*[*i*] to be acknowledged so that it can safely delete *Rsp*[*i*] from its cache.



For a synchronous Request/Reply the consumption of resource c takes constant space (i.e. there is only ever 1 cached response at a time) and therefore ensuring that it can be deleted is not really so important. But for asynchronous protocols where several requests and responses can be outstanding, resource consumption needs to be managed.

Send acknowledgements

The client will need to send acknowledgements, perhaps multiple times due to acknowledgements being lost.



Since acknowledgements do not represent any cached data, there is no notion of that at the client.

Discussion questions I

Question (1): For each of the Reply, Request/Reply and Request/Reply/Acknowledge protocols, draw sequence diagrams that show all of the relevant communication scenarios, including scenarios involving loss of messages.

Questio

process fai e, the

state of the p ocols be

improved to handle such possibilities?

Question (3): The protocols so far are synchrono

outstanding request is permitted at any one time. Let's s

outstanding requests/acknowledgements at any

Request/Reply/Acknowledge protocol that allows this. What about allowing up to k outstanding requests/acknowledgements at any one time?

Question (4): The protocols so far assumed that there is a sender and a receiver. In a peer-to-peer model, where either peer can make requests of

Discussion questions II

the other peer, we might consider a single protocol that involves states for both sending and receiving. This can be thought of as two protocols operatin … n you design a pr

Questio … communication. Suppose we have a point-to-multipoint protocol, e.g. where 3 peers are communicating such that a request se … er is to be processed by the other 2. Can you design a Request/Reply/Acknowledge protocol for this ca … he case of k peers?

Remote Procedure Call

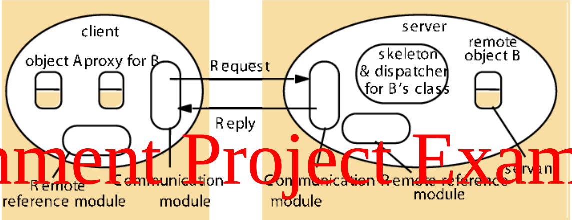
RPCs enable clients to execute procedures in server processes based on a defined service interface.

- **Communication Module** Implements the desi [illegible] retransmission of requests, dealing with duplicates a [illegible] sults.
- **Client Stub Procedure** Behaves like a local proc [illegible] the procedure identifiers and arguments which is hande [illegible] nication module. Unmarshalls the results in the reply.
- **Dispatcher** Selects the server stub based on the procedure identifier and forwards the request to the server stub.
- **Server stub procedure** Unmarshalls the arguments in the request message and forwards it to the service procedure. Marshalls the arguments in the result message and returns it to the client.

Remote Method Invocation

An object that can receive remote invocations is called a remote object. A remote object can receive remote invocations as well as local invocations. Remote objects can invoke methods in local objects as well as other remote ob

A remote object reference is a unique identifier that can b throughout the distributed system for identifying an object. This is used for invoking methods in a remote object and can be passed as arguments or returned as results of a remote method invocation.



- The **C** ... es (reque...
  - Mess...

- The **R** ... t references and maintaining the remote object table which is used for translating between local and remote object reference
- The **Proxy** plays the role of a local object to the invoking o... is a proxy for each remote object which is responsible for:

  - Marshalling the reference of the target object, its own method i... ents and forwarding them to the communication module.
  - Unmarshalling the results and forwarding them to the invoking object
- There is one **Dispatcher** for each remote object class. It is responsible for mapping to an appropriate method in the skeleton based on the method ID.
- The **Skeleton** is responsible for:
  - Unmarshalling the arguments in the request and forwarding them to the servant.
  - Marshalling the results from the servant to be returned to the client.

Binder or Registry

- Client programs require a way to obtain the remote object reference of the remote
- A **bind** that su
- A binder es to object references.
- Servers register their remote objects (by name) with the b ok them up by name

Discussion Question

Questio r
is a centrali gister their
remote o t happens
if the numb arge for a
single binder service to support? What can be done to solv is a
problem for RPC?

Garbage collection and Exceptions

- Garba and must count re opy of the remote

- Except ptions are RMI specific such as time out exceptions if there is network f