

Remote Procedure Call

Outline

- Protocol Stack
- Presentation Formatting

Spring 2002
CS 461
1

RPC Timeline

```

sequenceDiagram
    participant Client
    participant Server
    Note over Client: Blocked
    Client->>Server: Request
    Note over Server: Blocked
    Note over Server: Computing
    Server-->>Client: Reply
    Note over Client: Blocked
  
```

Spring 2002
CS 461
2

RCP Components

- Protocol Stack
 - BLAST: fragments and reassembles large messages
 - CHAN: synchronizes request and reply messages
 - SELECT: dispatches request to the correct process
- Stubs

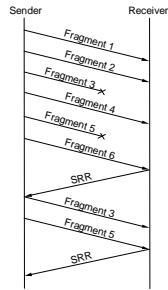
```

graph TD
    subgraph Client_Side [Client]
        C_Caller((Caller client))
        C_Arguments[Arguments]
        C_Return[Return value]
        C_ClientStub[Client stub]
        C_RPC[RPC protocol]
        C_Arguments --> C_ClientStub
        C_ClientStub --> C_RPC
        C_RPC --> C_Return
    end
    subgraph Server_Side [Server]
        S_Caller((Caller server))
        S_Arguments[Arguments]
        S_Return[Return value]
        S_ServerStub[Server stub]
        S_RPC[RPC protocol]
        S_Arguments --> S_ServerStub
        S_ServerStub --> S_RPC
        S_RPC --> S_Return
    end
    C_RPC --- Net((CS 461)) --- S_RPC
  
```

Spring 2002
CS 461
3

Bulk Transfer (BLAST)

- Unlike AAL and IP, tries to recover from lost fragments
- Strategy
 - selective retransmission
 - aka partial acknowledgements



Spring 2002

CS 461

4

BLAST Details

- Sender:
 - after sending all fragments, set timer DONE
 - if receive SRR, send missing fragments and reset DONE
 - if timer DONE expires, free fragments

Spring 2002

CS 461

5

BLAST Details (cont)

- Receiver:
 - when first fragments arrives, set timer LAST_FRAG
 - when all fragments present, reassemble and pass up
 - four exceptional conditions:
 - if last fragment arrives but message not complete
 - send SRR and set timer RETRY
 - if timer LAST_FRAG expires
 - send SRR and set timer RETRY
 - if timer RETRY expires for first or second time
 - send SRR and set timer RETRY
 - if timer RETRY expires a third time
 - give up and free partial message

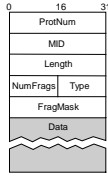
Spring 2002

CS 461

6

BLAST Header Format

- MID must protect against wrap around
- TYPE = DATA or SRR
- NumFrag indicates number of fragments
- FragMask distinguishes among fragments
 - if Type=DATA, identifies this fragment
 - if Type=SRR, identifies missing fragments



Spring 2002

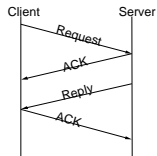
CS 461

7

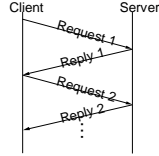
Request/Reply (CHAN)

- Guarantees message delivery
- Synchronizes client with server
- Supports *at-most-once* semantics

Simple case



Implicit Acks



Spring 2002

CS 461

8

CHAN Details

- Lost message (request, reply, or ACK)
 - set RETRANSMIT timer
 - use message id (MID) field to distinguish
- Slow (long running) server
 - client periodically sends “are you alive” probe, or
 - server periodically sends “I’m alive” notice
- Want to support multiple outstanding calls
 - use channel id (CID) field to distinguish
- Machines crash and reboot
 - use boot id (BID) field to distinguish

Spring 2002

CS 461

9

CHAN Header Format

```

typedef struct {
    u_short Type; /* REQ, REP, ACK, PROBE */
    u_short CID; /* unique channel id */
    int MID; /* unique message id */
    int BID; /* unique boot id */
    int Length; /* length of message */
    int ProtNum; /* high-level protocol */
} ChanHdr;

typedef struct {
    u_char type; /* CLIENT or SERVER */
    u_char status; /* BUSY or IDLE */
    int retries; /* number of retries */
    int timeout; /* timeout value */
    XkReturn ret_val; /* return value */
    Msg *request; /* request message */
    Msg *reply; /* reply message */
    Semaphore reply_sem; /* client semaphore */
    int mid; /* message id */
    int bid; /* boot id */
} ChanState;
    
```

Spring 2002

CS 461

10

Synchronous vs Asynchronous Protocols

- Asynchronous interface
 - `send(Protocol llp, Msg *message)`
 - `deliver(Protocol llp, Msg *message)`
- Synchronous interface
 - `call(Protocol llp, Msg *request, Msg *reply)`
 - `upcall(Protocol hlp, Msg *request, Msg *reply)`
- CHAN is a hybrid protocol
 - synchronous from above: **call**
 - asynchronous from below: **deliver**

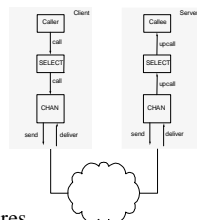
Spring 2002

CS 461

11

Dispatcher (SELECT)

- Dispatch to appropriate procedure
- Synchronous counterpart to UDP
- Implement concurrency (open multiple CHANS)



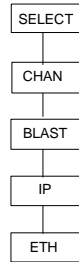
- Address Space for Procedures
 - flat: unique id for each possible procedure
 - hierarchical: program + procedure number

Spring 2002

CS 461

12

Simple RPC Stack



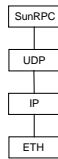
Spring 2002

CS 461

13

SunRPC

- IP implements BLAST-equivalent
 - except no selective retransmit
- SunRPC implements CHAN-equivalent
 - except not at-most-once
- UDP + SunRPC implement SELECT-equivalent
 - UDP dispatches to program (ports bound to programs)
 - SunRPC dispatches to procedure within program



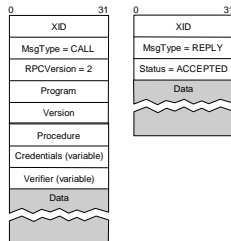
Spring 2002

CS 461

14

SunRPC Header Format

- XID (transaction id) is similar to CHAN's MID
- Server does not remember last XID it serviced
- Problem if client retransmits request while reply is in transit



Spring 2002

CS 461

15

Presentation Formatting

- **Marshalling (encoding)** application data into messages
- **Unmarshalling (decoding)** messages into application data



- **Data types we consider**
 - integers
 - floats
 - strings
 - arrays
 - structs
- **Types of data we do not consider**
 - images
 - video
 - multimedia documents

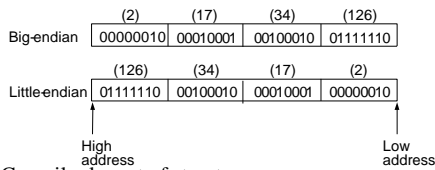
Spring 2002

CS 461

16

Difficulties

- **Representation of base types**
 - floating point: IEEE 754 versus non-standard
 - integer: big-endian versus little-endian (e.g., 34,677,374)



- **Compiler layout of structures**

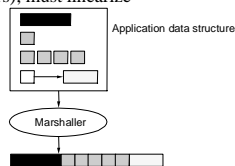
Spring 2002

CS 461

17

Taxonomy

- **Data types**
 - base types (e.g., ints, floats); must convert
 - flat types (e.g., structures, arrays); must pack
 - complex types (e.g., pointers); must linearize



- **Conversion Strategy**
 - canonical intermediate form
 - receiver-makes-right (an $N \times N$ solution)

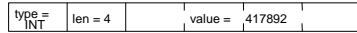
Spring 2002

CS 461

18

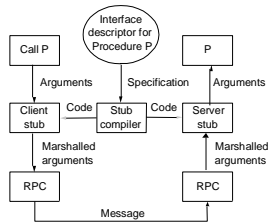
Taxonomy (cont)

- Tagged versus untagged data



- Stubs

- compiled
- interpreted



Spring 2002

CS 461

19

eXternal Data Representation (XDR)

- Defined by Sun for use with SunRPC
- C type system (without function pointers)
- Canonical intermediate form
- Untagged (except array length)
- Compiled stubs

Spring 2002

CS 461

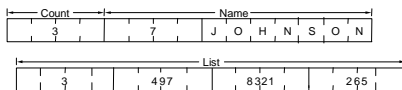
20

```

#define MAXNAME 256;
#define MAXLIST 100;

struct item {
    int    count;
    char  name[MAXNAME];
    int   list[MAXLIST];
};

bool_t
xdr_item(XDR *xdrs, struct item *ptr)
{
    return(xdr_int(xdrs, &ptr->count) &&
           xdr_string(xdrs, &ptr->name, MAXNAME) &&
           xdr_array(xdrs, &ptr->list, &ptr->count,
                     MAXLIST, sizeof(int), xdr_int));
}
    
```



Spring 2002

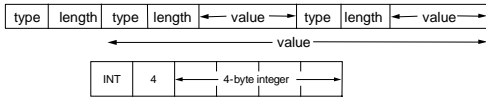
CS 461

21

Abstract Syntax Notation One (ASN-1)

- An ISO standard
- Essentially the C type system
- Canonical intermediate form
- Tagged
- Compiled or interpreted stubs
- BER: Basic Encoding Rules

(tag, length, value)



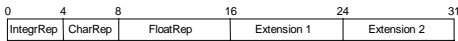
Spring 2002

CS 461

22

Network Data Representation (NDR)

- Defined by DCE
 - Essentially the C type system
 - Receiver-makes-right (architecture tag)
 - Individual data items untagged
 - Compiled stubs from IDL
 - 4-byte architecture tag
- IntegerRep
 - 0 = big-endian
 - 1 = little-endian
 - CharRep
 - 0 = ASCII
 - 1 = EBCDIC
 - FloatRep
 - 0 = IEEE 754
 - 1 = VAX
 - 2 = Cray
 - 3 = IBM



Spring 2002

CS 461

23
