Chapter 20

External Data Representation

Problem: Each computer architecture has its own data representation

Bad Solution: Asymmetric Data Conversion
Have one side of a client/server pair perform conversion.
Must write a different version of client/server pair for each of the architecture pairs

Good Solution: Symmetric Data Conversion
Example: Network Standard Byte Order
Both ends perform conversion

Advantage: Flexibility
Neither machine needs to know about the other's architecture

Disadvantage: May perform conversion when it's not necessary

External Data Representation (XDR)

Designed by Sun Microsystems (RFC 1014)
Specifies formats for most data types (Figure 20.2)

Software Support:
library routines to perform conversion

Uses a buffer paradigm to create messages

Building a message:

1) Allocate a buffer (an XDR stream),
specifies encode/decode

`xdrmem_create` initializes a buffer and
returns a pointer to the start of the stream (buffer)

2) Add items (i.e., fields) one at a time:
Call conversion routines, they
a) encode an object
b) append the encoding to the buffer
c) update the buffer

Result: a stream with XDR format.

Sending an XDR Message

Use `write` (TCP) to send the buffer.

or...

You can have XDR conversion routines automatically send data across the TCP connection by:

1) Create a socket

2) Call `fdopen` to attach a UNIX standard I/O stream to the socket

3) Use `xdrstdio_create` (instead of `xdrmem_create`) to create the XDR stream and connect it to the I/O descriptor

4) the conversion routines will automatically perform a buffered write (or read)

Standard I/O library routines can be called (e.g., `fflush`)

Receiving an XDR Message

Reverse the entire encoding process

Call `xdrmem_create` to create a buffer and
specify `XDR_DECODE`.

Use standard I/O functions (`read`) to fill the buffer

Call conversion routines to decode the information

Note: if the buffer is created specifying encode,
the conversion routines encode

if the buffer is created specifying decode,
the conversion routines decode

XDR Details

The following primitive types can be encoded:

bool	`xdr_bool`
char	`xdr_char`
short	`xdr_short`
unsigned short	`xdr_u_short`
int	`xdr_int`
unsigned int	`xdr_u_int`
long	`xdr_long`
unsigned long	`xdr_u_long`
float	`xdr_float`
double	`xdr_double`
void	`xdr_void`
enum	`xdr_enum`

All unsigned are encoded as unsigned integer.

All signed, character, enum are encoded as integer.

Float, double, void, have special formats.

XDR Details

The following composite types can be encoded.

fixed-length array	`xdr_vector`
discriminated unions	`xdr_union`
variable-length arrays	`xdr_array`
variable-length byte arrays	`xdr_bytes`
strings	`xdr_string`
variable-length strings	`xdr_wrapstring`
object references	`xdr_reference`
object references	`xdr_pointer`

Variable-length items have a length integer encode in front of them.

Unions must be discriminated (to figure out what types the components have).

`xdr_reference` recursively follows pointers, but doesn't handle null.

`xdr_pointer` recursively follows pointers and handles null. Warning: garbage pointers are “followed”.

No xdr exists for multiple dimensional arrays.

XDR Example (send)

Pack up a few integers and send them

```
#include <rpc/types.h>
#include <rpc/xdr.h>
#include "connectUDP.c"
main(){
  int sock;
  int test_number_a = 6;
  int test_number_b = 47;
  float test_number_c = -34.5;
  char buffer[80];
  XDR xdrobject;
  XDR *xdrstream = &xdrobject;
  /* XDR a message */
  xdrmem_create(xdrstream, buffer, 80, XDR_ENCODE);
  xdr_int(xdrstream, &test_number_a);
  xdr_int(xdrstream, &test_number_b);
  xdr_float(xdrstream, &test_number_c);
  /* Get a socket (UDP) */
  sock = connectUDP("aardvark", "7654");
  /* send the message */
  write(sock, buffer, 80);
  xdr_destroy(xdrstream);
  close(sock);
}
```

XDR Example (receive)

Unpack the integers and print them

```
#include <rpc/types.h>
#include <rpc/xdr.h>
#include "passiveUDP.c"
main(){
  int sock;
  int test_number_a;
  int test_number_b;
  float test_number_c;
  char buffer[80];
  XDR xdrobject;
  XDR *xdrstream = &xdrobject;
  /* Get a socket (UDP) */
  sock = passiveUDP("7654");
  read(sock, buffer, 80);
  close(sock);
  /* extract the message */
  xdrmem_create(xdrstream, buffer, 80, XDR_DECODE);
  xdr_int(xdrstream, &test_number_a);
  xdr_int(xdrstream, &test_number_b);
  xdr_float(xdrstream, &test_number_c);
  printf("%d, %d, %f\n", test_number_a,
    test_number_b, test_number_c);
  xdr_destroy(xdrstream);
}
```