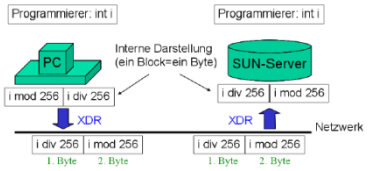
VSY - SUN RPC

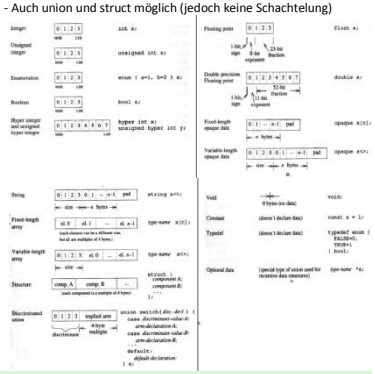
eXternal Data Representation

- XDR bietet Maschinenunabhängigkeit bei der Übermittlung der Daten



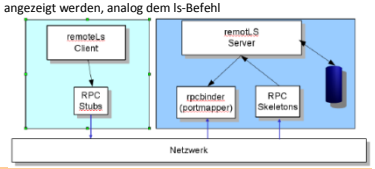
XDR-Eigenschaften

- Bytes auf allen Architekturen gleich
- Alle Datentypen werden als Vielfaches von 4 Bytes übertragen
- Ähnliche Typen wie in C, jedoch wird char* als string deklariert
- Auch union und struct möglich (jedoch keine Schachtelung)



XDR-Beispiel: Remote ls

- Es soll der Inhalt eines Verzeichnis auf einem entfernten Rechner angezeigt werden, analog dem ls-Befehl



Checkfragen RemoteLs

- Welche Teile im vorstehenden Bild werden von uns programmiert und welche Teile werden durch RPC zur Verfügung gestellt?
  - Interface zwischen Clients
  - XDR-File
  - remoteLs Client
  - remoteLs Server
- Was müssen wir als erstes Erstellen und wie könnte dieses etwas aussehen?
  - ???

XDR-Beispiel: Remote ls div.x

```
 *
 * The result of a READDIR operation.
 */
union readdir_res switch (int errno) {
case 0:
     namelist list; /* no error: return directory listing */
default:
     void;      /* error occurred: nothing else to return */
};

/*
 * The directory program definition
 */
program DIRPROG {
   version DIRVERS {
        readdir_res
        READDIR(nametype) = 1;
   } = 1;
} = 0x20150976;
const MAXNAMELEN = 255;          /* maximum entry length */

typedef string nametype<MAXNAMELEN>;    /* a directory entry */

typedef struct namenode *namelist;  /* a link in the listing */

/* A node in the directory listing */

struct namenode {
nametype name;          /* name of directory entry */
namelist next;          /* next entry */
```

XDR-Beispiel: Codeteile

- Aufruf: rpcgen generiert nun zusätzlich eine XDR-Datei (hier dir_xdr.c) welche mit Client und Server gelinkt werden muss
- Der Aufbau wird nur minimal geändert
- Die vordefinierten structures können in C wie gewohnt verwendet werden

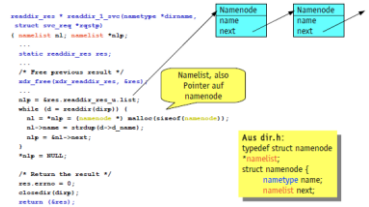
Generiertes dir.h

```
void dirprog_1(char *host) {
/* ...*/
result = readdir_1(&dir, cl);
 if (result->errno == 0) {
for ( nl = result->readdir_res_u.list;
      nl != NULL;
                  nl = nl->next) {
                        printf("%s\n", nl->name);
       }
/* ... */
}
int main(int argc, char *argv[]) {
   ...
   host= argv[1];
   dirprog_1(host);
   exit(0);
}
```

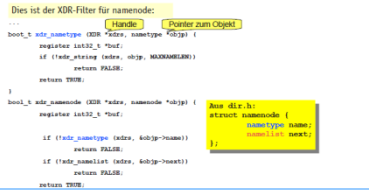
Aus dir.h:

```
struct readdir_res {
  int errno;
  union{
    namelist list;
  } readdir_res_u;
};
```

Inhalt von dir_proc.c (Server-Implementation)



XDR-Interna: Generiertes dir_xdr.h



Diverses

Spezielle RPC Calls

- Nonblocking RPC
  - Setzt der Client den Timeout auf 0, kehrt clnt_call nach dem Senden sofort mit einem Timeout-Fehler zurück
  - Der Server übergibt NULL als return-Wert. Dadurch weiss der Skeleton, dass keine Antwort zu senden ist
  - Dies funktioniert mit allen Transportmethoden, also egal ob TCP oder UDP
- Callback
  - Client registriert sich als temporärer Server und geht später in die Server-Methode
- Broadcast
  - Der Request wird über Broadcast verteilt
  - Client kann mehrere Antworten verarbeiten

Broadcast

```
enum clnt_stat clnt_broadcast(unsigned long prognum,
        unsigned long versnum, unsigned long procnum,
        xdrproc_t inproc, char *in,  xdrproc_t outproc, char *out,
        resultproc_t eachresult);
```

- Der Call wird als Broadcast auf alle lokale Netze durchgeführt. Jede zurückkehrende Antwort wird über die übergebene Methode eachresult abgewickelt:

```
eachresult(char *out, struct sockaddr_in *addr);
```

- In das out wird die Antwort dekodiert, addr enthält die Adresse des Rechners, der die Antwort liefert
- Mit dem Rückgabewert von eachresult können Sie definieren, ob clnt_broadcast auf weitere Resultate warten soll (return 0)
- Broadcasts können über Ethernet max. 1500 octets übertragen

Zusammenfassung RPC

- Ermöglicht verteilte Applikationen zu schreiben
- Abstrahierung von low-level Kommunikation (Sockets)
- Kommunikationsdetails versteckt [] weniger Kontrolle
- Verschiedene Abstraktionslevels vorhanden
- rpcgen generiert aus Protokoll-Spezifikation die Stub und Skeletons
- XDR ermöglicht es Daten unabhängig von der Systemarchitektur zu codieren
- Vorgesehene Authentisierung mit DES vorhanden

RPC im Vergleich

- Basiert auf Methoden-Aufruf, der statt lokal nun remote erfolgt
- Keine Vererbung möglich
- Parameter sind Basis-Typen und Strukturen, keine Objekte
- Protokoll in RFC definiert, praktisch aber nur für C vorhanden
- Server-Dienste sind Rechnerorientiert, d.h. keine Ortstransparenz
- In der Regel auf LAN beschränkt
- Broadcasts sind möglich