RPC

UDIS 2011/2012

RPC (Remote procedure call)

- RPC tehnologija omogućava razvoj mrežno distribuiranih aplikacija kod kojih proces na jednom računalu može pozvati proceduru (funkciju, metodu) procesa na drugom računalu.
- Procedure na udaljenom računalu izvode se prividno kao lokalne procedure kroz RPC međusloj.

Razvoj RPC-a

- RPC tehnologija se pojavljuje u 80-tima sa ciljem implementacije imperativne programske paradigme kojom se funkcionalnosti aplikacije enkapsuliraju u procedurama.
- Najčešće korišteni RPC API-ji su:
  1. *Open Network Computing* ONC RPC (Sun)
  2. *Open Group Distributed Computing Envinronment* DCE RPC
     - MS-RPC je Microsoftova verzija DCE RPC-a.

RPC IDL

- Sve RPC implementacije imaju jezik kojim se apstraktno, neovisno o jeziku u kojem će biti implementirana, opisuje RPC usluga.
- Taj jezik se generički naziva definicijski jezik sučelja IDL (*interface definition language*) (npr. Microsoft je implementirao DCE RPC IDL s određenim proširenjima pod imenom MIDL – *Microsoft Interface Definition Language*).
- Udaljena usluga (funkcije koje se mogu pozvati) se definira preko sučelja (*interface*).

IDL

- Programer koristi IDL jezik da bi opisao sučelja udaljenih procedura.
- IDL datoteka je slična C header datotekama proširenim s dodatnim ključnim riječima i konstrukcijama.

Primjer MIDL datoteke

```
import "oaidl.idl";
import "ocidl.idl";
[
  // Jedinstveni identifikator sučelja
  uuid(d803ceb6-86de-49c2-933f-9df366763e4a),
  // Verzija sučelja.
  version(1.0),
  implicit_handle(handle_t MatematikaBinding)
]
interface Matematika // Ime sučelja
{
      void PosaljiBrojeveioperator([in] handle_t
      MatematikaBinding, [in ] int prvi,[in ] char
      oper,[in ] int drugi);

      void VratiRez ([in] handle_t MatematikaBinding,
      [out] int  *rez);
}
```

IDL

- IDL je atributivni programski jezik (*attributed programming language*) kojim se mogu opisati parametri procudura i sve ostalo što je potrebno znati za pozvati udaljenu proceduru.
- IDL jezik ima točno definiranu sintaksu i semantiku.
- IDL-om se definira sučelje koje sadrži funkcije (imena, parametare, tipove parametara) koje onda klijent može pozvati.

MIDL jezik

MIDL

- Sučelje definiranu u MIDL uvijek ima isti format.
- Započinje sa zaglavljem koje sadrži listu atributa sučelja i ime sučelja.
- Svi atributi su zatvoreni u uglatim zagradama.
- Nakon zaglavlja sučelja dolazi tijelo sučelja koje je omeđeno vitičastim zagradama.

MIDL

```
import "oaidl.idl";
import "ocidl.idl";
[
  // Jedinstveni identifikator sučelja
  uuid(d803ceb6-86de-49c2-933f-9df366763e4a),
  // Verzija sučelja.
  version(1.0),
  implicit_handle(handle_t MatematikaBinding)
]
interface Matematika // Ime sučelja
{
      void PosaljiBrojeveioperator([in] handle_t
      MatematikaBinding, [in ] int prvi,[in ] char
      oper,[in ] int drugi);

      void VratiRez ([in] handle_t
      MatematikaBinding, [out] int  *rez);
}
```

- Uključi drugi IDL ili h file
- [uuid] atribut sučelja definira unverzalni jedinstveni identifikator (universally unique identifier (UUID)) sučelja kojim se sučelje razlikuje od drugih sučelja
- [version] atribut definira verziju sučelja (da biste bili sigurni u kompatibilnost klijenta i servera)

UUID

- UUID (universally unique identifier) je standard za jedinstvenu identifikaciju entiteta (proces, objekt, ...) bez centralne koordinacije. To je dio DCE RPC standarda i početno je definiran za identificiranje entiteta na mreži .
- Prema standardu UUID je 128-bitni broj koji se obično prikazuje u heksadecimalnom zapisu (32 znamenke):

  F6D90F14-9C73-11D3-B32E-00C04F990BB4

ONC RPC standard

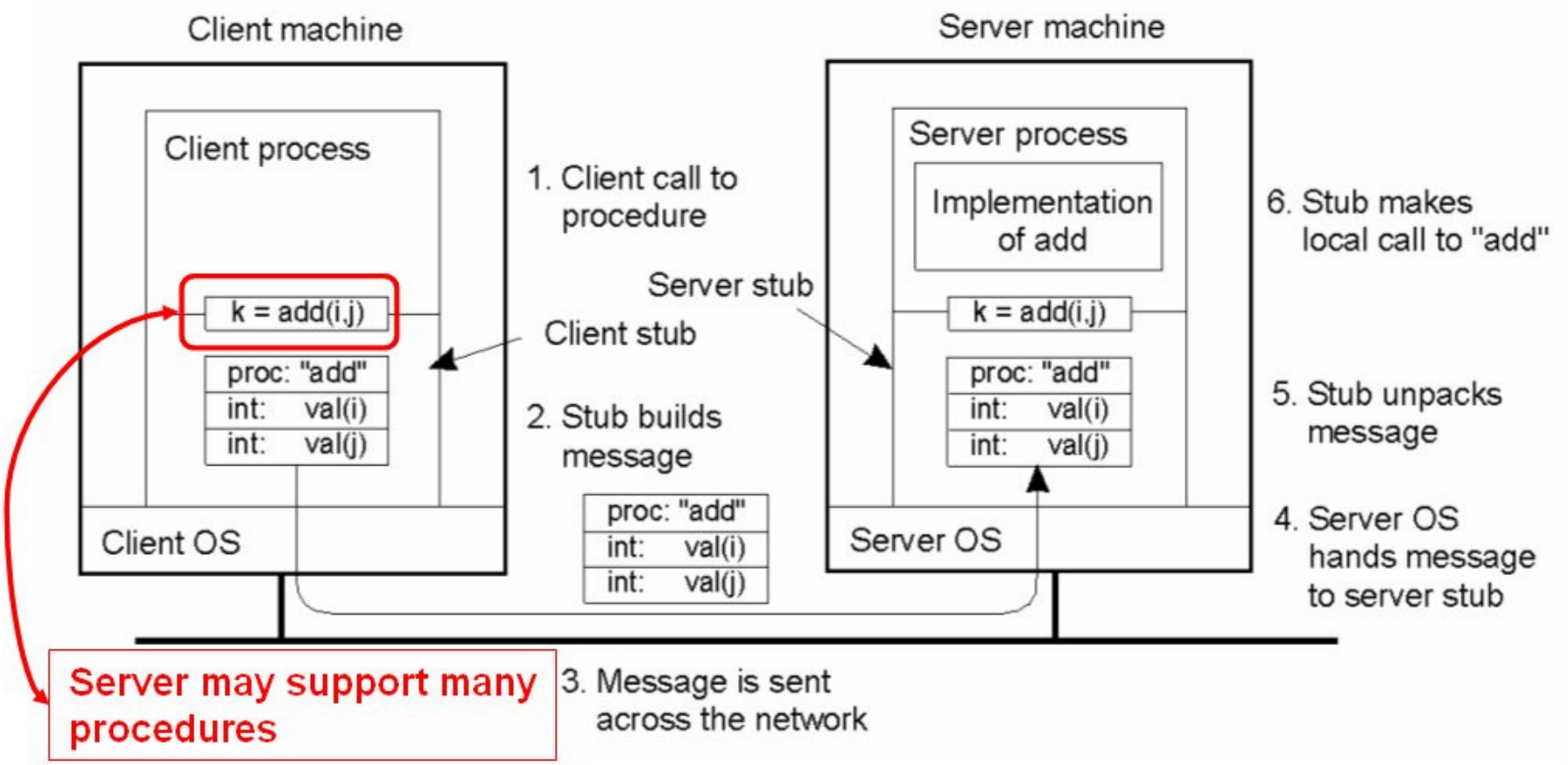
GUID

- GUID (globally unique identifier) je Microsftova implementacija ovog standarda.
- Obzirom na veličinu GUID-a vjerojatnost da se slučajnim generiranjem GUID jednom entitetu dodjeli isti GUID kao i drugom je stvarno minimalna (ali postoji). Češća situacija je da se GUID ne generira slučajno nego da se oslanja ne neku postavku operacijskog sustava (npr. IP adresu računala) da bi se osiguralo da GUID koji se koristi u vlastitom softveru bude jedinstven (ako se radi o javnoj IP adresi nitko ne smije imati istu IP adresu).

IDL

- Nakon pisanja IDL koda koristi se IDL kompajler (*rpcgen* za SUN RPC, *midl.exe* za Microsoft Visual Studio,...) koji taj IDL (Sun-ov ili DCE-ov ili MIDL) zna prevesti u tzv. zamjenske programske elemente ili komunikacijske umetke (*communication stubs*) u programskom jeziku (midl u C-u) koji se koristi za pisanje aplikacije.
- *Stubovi* postoje i na strani klijenta i na strani servera. Ti dijelovi koda omogućavaju da klijent poziva udaljene funkcije na serveru kao da su lokalne.

RPC


Client machine
Server machine
Client process
Server process
Implementation of add
k = add(i,j)
k = add(i,j)
proc: "add"
int: val(i)
int: val(j)
proc: "add"
int: val(i)
int: val(j)
Client OS
Server OS
1. Client call to procedure
Server stub
Client stub
2. Stub builds message
proc: "add"
int: val(i)
int: val(j)
6. Stub makes local call to "add"
5. Stub unpacks message
4. Server OS hands message to server stub
Server may support many procedures
3. Message is sent across the network

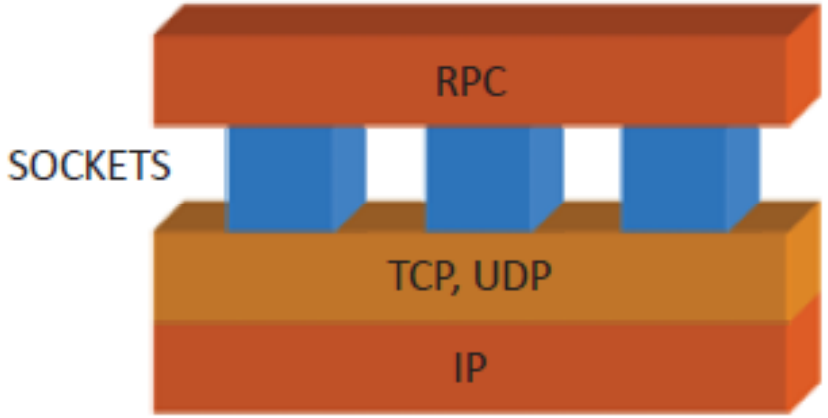
RPC

- Pojednostavljeno rpc se odvija na sljedeći način:

1. Klijent poziva udaljenu proceduru
2. Poziv se prosljeđuje klijentskom *stubu*
3. Klijentski *stub* formatira poziv procedure u oblik koji se može prosljediti preko mreže
4. Predaje tako formatirane podatke operacijskom sustavu koji ih šalje na mrežu
5. Na strani servera operacijski sustav dobiva rpc podatke
6. Prosljeđuje ih serverskom *stubu*
7. Serverski *stub* poziva proceduru servera

RPC vezivanje (*binding*)

- Kod RPC-a se *bindingom* naziva spajanje klijenta na server.
- Opet su klijent i server jedinstveno identificirani IP adresom i TCP/UDP portom.



RPC vezivanje

- Postoji nekoliko načina vezivanja klijenta i servera:

1. Automatsko vezivanje (*automatic binding*) kod kojeg se povezivanje radi automatski od strane klijentskog *stuba*. Klijentski *stub* dobiveni handle na vezu prosljeđuje RPC međusloju.

RPC vezivanje

2. Implicitno vezivanje (*implicit binding*) kod kojeg klijentska aplikacija treba dohvatiti podatke za vezivanje, a vezivanje se obavi unutar klijentskog *stuba*.
3. Eksplicitno vezivanje (*explicit binding*) kod kojeg se eksplicitno prilikom svakog RPC poziva treba prosljeđivati handle na vezu (znači postoji dio aplikacije koje eksplicitno otvara vezu i dobavlja vraćeni handle na vezu).

MIDL

```
import "oaidl.idl";
import "ocidl.idl";
[
  // Jedinstveni identifikator sučelja
  uuid(d803ceb6-86de-49c2-933f-9df366763e4a),
  // Verzija sučelja.
  version(1.0),
  implicit_handle(handle_t MatematikaBinding)
]
interface Matematika // Ime sučelja
{
      void PosaljiBrojeveioperator([in] handle_t
      MatematikaBinding, [in ] int prvi,[in ] char
      oper,[in ] int drugi);

      void VratiRez ([in] handle_t
      MatematikaBinding, [out] int  *rez);
}
```

- [implicit_handle] atribut specificira globalnu varijablu koja sadrži handle koji koristi svaka funkcija koja treba implicitni handle.
- interface ključna riječ specificira ime sučelja

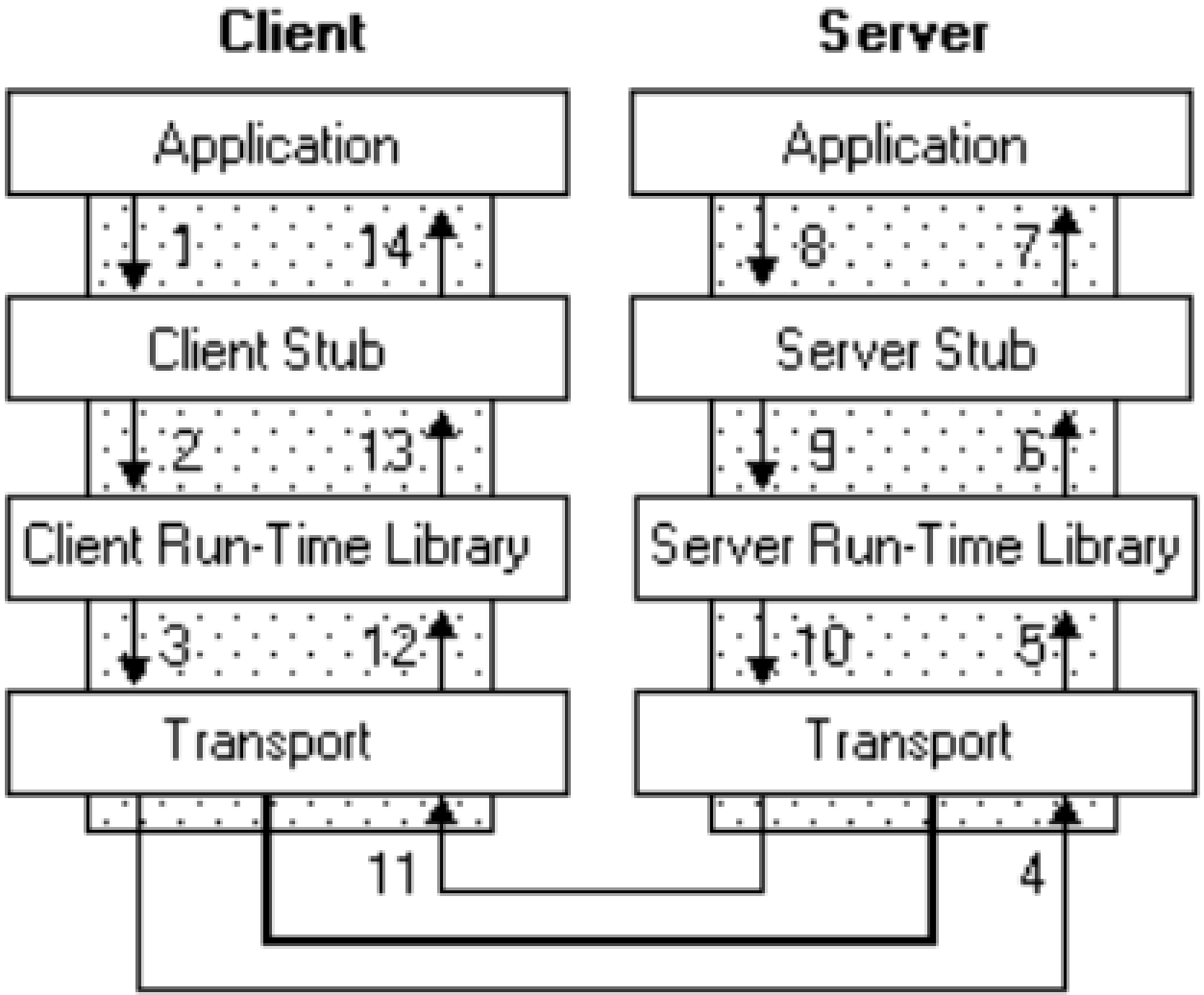
MIDL

```
import "oaidl.idl";
import "ocidl.idl";
[
  // Jedinstveni identifikator sučelja
  uuid(d803ceb6-86de-49c2-933f-9df366763e4a),
  // Verzija sučelja.
  version(1.0),
  implicit_handle(handle_t MatematikaBinding)
]
interface Matematika // Ime sučelja
{
      void PosaljiBrojeveioperator([in] handle_t
      MatematikaBinding, [in ] int prvi,[in ] char
      oper,[in ] int drugi);

      void VratiRez ([in] handle_t
      MatematikaBinding, [out] int  *rez);
}
```

- [in] atribut pokazuje da se parametra prosljeđuje od pozivatelja procedure
- [out] atribut identificira parametar tipa pokazivača koji se vraća nazad pozivatelju procedure (tj. od servera nazad klijentu).

RPC


Client
Server
Application
Application
1
14
8
7
Client Stub
Server Stub
2
13
9
6
Client Run-Time Library
Server Run-Time Library
3
12
10
5
Transport
Transport
11
4

Formatiranje

- *Stub* je zadužen za formatiranje parametara koji se razmjenjuju između dva procesa.
- Formata podataka ovisi o cijelom nizu fakotra. Npr. različiti operacijski sustavi koriste različite načine predstavljanja podataka (ASCII, EBCDCIC, 16-bitni, 32-bitni *integer, big-endian*, *little-endian*).
- Stoga se dva distribuirana procesa trebaju "dogovoriti" oko načina formatiranja podataka.

Formatiranje

- Jedno rješenje je vanjski (eksterni) prijelazni način formatiranja podataka.
- U tom slučaju prilikom razmjene podataka potrebno je napraviti konverziju podataka.
- Postupak *marshallinga* je konverzija podataka iz lokalnog formata u eksterni format, a postupak *unmarshalling* je konverzija podataka u obrnutom smjeru iz eksternog formata u lokalni format.

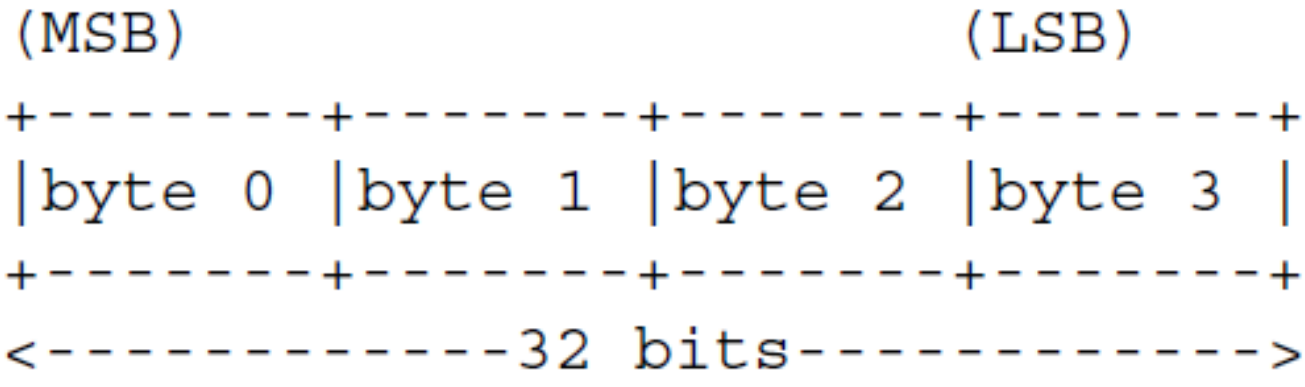
Formatiranje

- ONC RPC koristi XDR (*External Data Representation*) eksterni format podataka.
- DCE RPC koristi NDR (*Network Data Representation*) eksterni format.
- CORBA koristi *Common Data Representation* (CDR)/IDL eksterni format.
- Java RMI koristi serijalizaciju objekata (*object serialization*) za formatiranje podataka za prijenos preko mreže između dva procesa.

Formatiranje

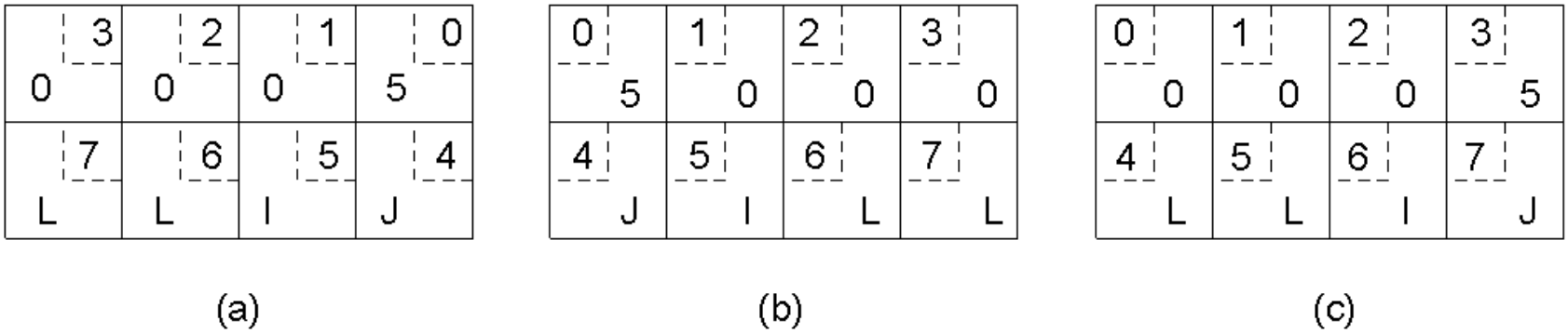
- Npr. XDR unsigned integer je 32-bitna vrijednost u rasponu [0 - 4294967295]. Predstavljen je binarnim brojem sa najznačajnijim bajtom 0 i najmanje značajnim bajtom 3.

Unsigned Integer



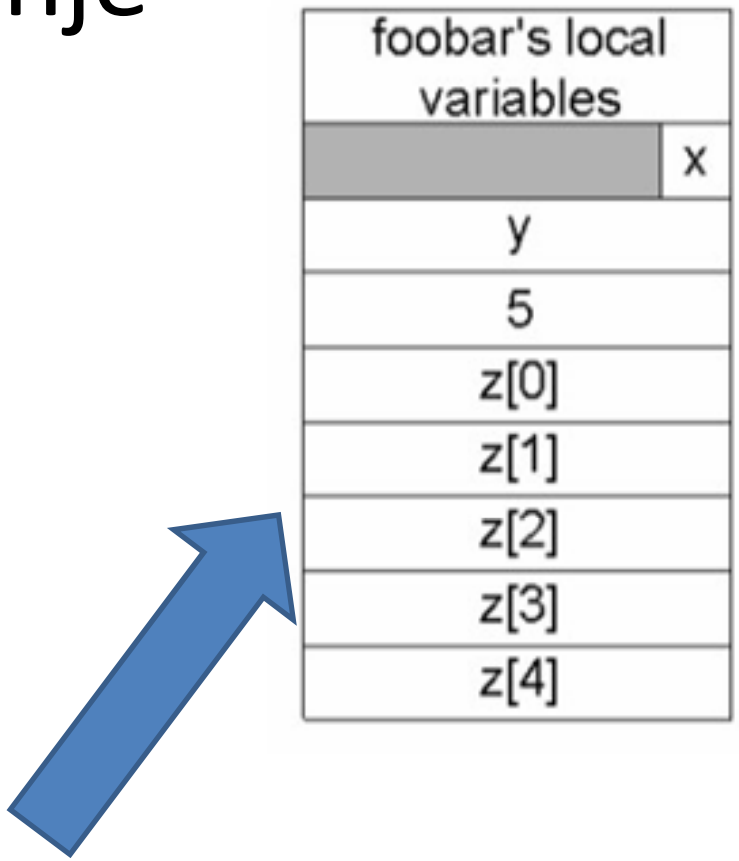
Formatiranje

a) Originalna poruka na Pentium-u
b) Primljena poruka na SPARC-u (Sun-ov RISC procesor)
c) Poruka nakon *unmarshalling-a.*



Formatiranje

- Klijent i server moraju se dogovoriti i na koji način će pakirati parametre u poruku. Hoće li na prvo mjesto staviti prvi argument ili će argumente slagati nekim drugim redoslijedom.
- Sve ovo treba biti pokriveno u eksternom formatu podataka.




```
foobar(int x, float y, int z[5])
{
.......
}
```

RPC ↔ socket

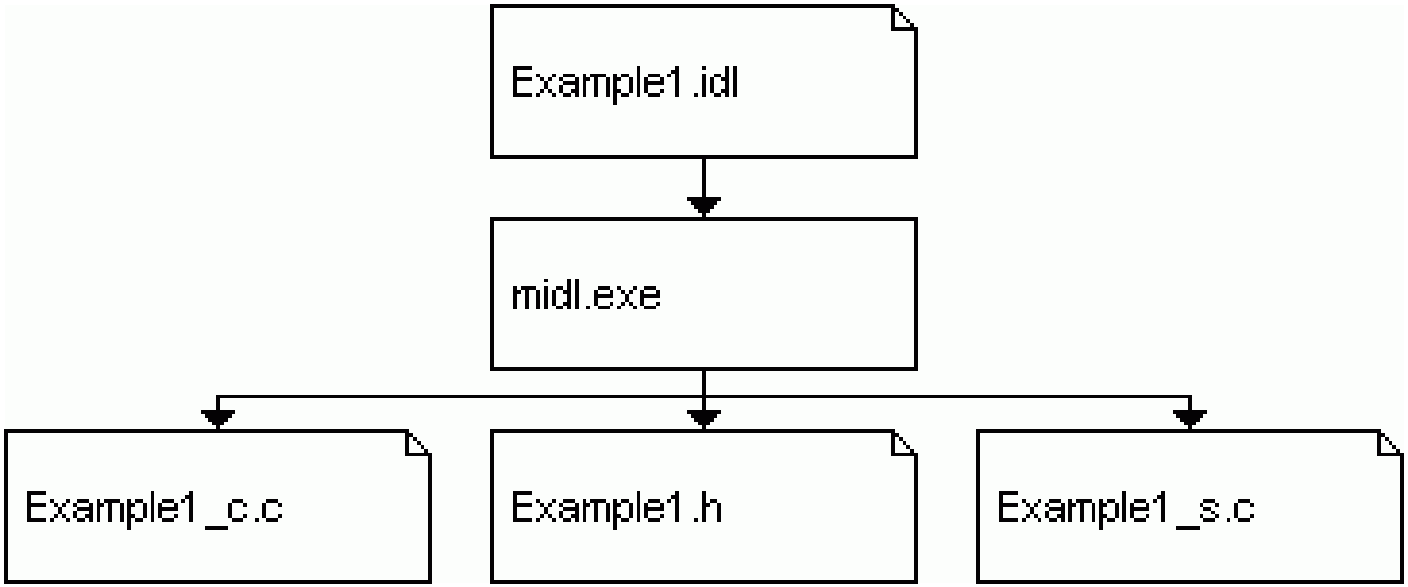
- PREDNOSTI
  - Jednostavnost
  - Lakše programiranje

RPC aplikacije

- NEDOSTATCI
  - Zahtjeva dosta procesorske snage, tipični RPC poziv se izvodi u 24 koraka tj. 10,000~15,000 instrukcija
  - Velika pojasna širina komunikacije

Realizacija MS-RPC aplikacije

- IDL treba iskompajlirati s MIDL kompajlerom (*midl.exe*) koji će prevesti IDL kôd u klijentski odnosno serverski *stub* u C-u koji se onda koristi u serverskoj odnosno klijentskoj aplikaciji.



MIDL

- Napomena: Uključiti u Project Properties→MIDL→Command Line→Additional options→/app_config kako se poziv implicite_handle ne bi trebao odvajati u zasebnu datoteku.

Server

1

```
#include <iostream>
#include "Matematika_h.h"
int oper1=0,oper2=0;
char operznak='+';

void PosaljiBrojeveioperator(/* [in] */ handle_t MatematikaBinding,
    /* [in] */ int prvi, /* [in] */ unsigned char oper, /* [in] */ int drugi)
{
    oper1=prvi;
    oper2=drugi;
    operznak=oper;
    printf("%d\n",oper1);
    printf("%d\n",oper2);
    printf("%c\n",operznak);
}
```

Server

2

```
void VratiRez(
   /* [in] */ handle_t MatematikaBinding,
   /* [out] */ int *rez)
{
     int a;
     if(operznak=='+')
            a=oper1+oper2;
     else if(operznak=='*')
            a=oper1*oper2;
     else if(operznak=='-')
            a=oper1-oper2;
     else if(operznak=='/')
            a=oper1/oper2;

     printf("%d\n",a);
     memcpy(rez, &a, sizeof(int));
}
```

Server

3

```
int main()
{
   RPC_STATUS status;

   status = RpcServerUseProtseqEp(reinterpret_cast<unsigned
     char*>("ncacn_ip_tcp"),
     RPC_C_PROTSEQ_MAX_REQS_DEFAULT,
     reinterpret_cast<unsigned char*>("4747"), NULL);

if (status)  exit(status);

    status=RpcServerRegisterIf2(Matematika_v1_0_s_ifspec,
     NULL, NULL, RPC_IF_ALLOW_CALLBACKS_WITH_NO_AUTH,
     RPC_C_LISTEN_MAX_CALLS_DEFAULT, -1, NULL);

   if (status) exit(status);
```

Server

4

```
status = RpcServerListen(1,
    RPC_C_LISTEN_MAX_CALLS_DEFAULT , FALSE);

  if (status)  exit(status);
}

void* __RPC_USER midl_user_allocate(size_t size)
{
   return malloc(size);
}
void __RPC_USER midl_user_free(void* p)
{
   free(p);
}
```

Server

- Funkcija RpcServerUseProtseqEp ukazuje RPC run-time biblioteci da koristi navedeni protokol i krajnju točku komunikacije za prihvaćanje udaljenih poziva procedura.

  RPC_STATUS RPC_ENTRY RpcServerUseProtseqEp(unsigned char* *Protseq*, unsigned int *MaxCalls*, unsigned char* *Endpoint*, void* *SecurityDescriptor*);

  PRIMJER:

  status = RpcServerUseProtseqEp(reinterpret_cast<unsigned char*>("ncacn_ip_tcp"), RPC_C_PROTSEQ_MAX_REQS_DEFAULT, reinterpret_cast<unsigned char*>("4747"), NULL);

Server

- Funkcija RpcServerRegisterIf2 registrira naše sučelje s RPC run-time bibliotekom.

  RPC_STATUS RPC_ENTRY RpcServerRegisterIf2(RPC_IF_HANDLE *IfSpec*, UUID* *MgrTypeUuid*, RPC_MGR_EPV* *MgrEpv*, unsigned int *Flags*, unsigned int *MaxCalls*, unsigned int *MaxRpcSize*, RPC_IF_CALLBACK_FN* *IfCallbackFn*);

PRIMJER:

```
status=RpcServerRegisterIf2(Matematika_v1_0_s_ifspec,
NULL, NULL, RPC_IF_ALLOW_CALLBACKS_WITH_NO_AUTH,
RPC_C_LISTEN_MAX_CALLS_DEFAULT, -1, NULL);
```

Server

- Funkcija RpcServerListen signalizira RPC run-time biblioteci da server osluškuje dolazne udaljene pozive procedura.

  RPC_STATUS RPC_ENTRY RpcServerListen(unsigned int *MinimumCallThreads*, unsigned int *MaxCalls*, unsigned int *DontWait*);

  PRIMJER:

  status = RpcServerListen(1, RPC_C_LISTEN_MAX_CALLS_DEFAULT, FALSE);

Server

- midl_user_allocate funkcija i na strani klijenta i na strani server omogućava RPC aplikaciji alokaciju memorije.
- I klijent i server moraju implementirati ovu funkciju, ukoliko se pri kompajliranju IDL datoteke ne koristi opcija /osf.
- Aplikacije i zamjenski programski elementi pozivaju funkciju midl_user_allocate kada rade s pokazivačima. Npr. klijentov stub poziva midl_user_allocate kada radi *unmarshalling* podataka primljenih od server preko out pokazivača.

Server

- midl_user_free funkcija i na strani klijenta i na strani server omogućava RPC aplikaciji dealokaciju alocirane memorije.
- I klijent i server moraju implementirati ovu funkciju, ukoliko se pri kompajliranju IDL datoteke ne koristi opcija /osf.

Server

PRIMJER:

```
void* __RPC_USER midl_user_allocate(size_t size)
{
    return malloc(size);
}
void __RPC_USER midl_user_free(void* p)
{
    free(p);
}
```

Klijent

1

```
#include <iostream>
#include "Matematika_h.h"

int main()
{
  RPC_STATUS status;
  unsigned char* szStringBinding = NULL;
  status = RpcStringBindingCompose(NULL, reinterpret_cast<unsigned
    char*>("ncacn_ip_tcp"), reinterpret_cast<unsigned
    char*>("laris6.fesb.hr"), reinterpret_cast<unsigned char*>("4747"), NULL,
    &szStringBinding);

  if (status)
    exit(status);
  status = RpcBindingFromStringBinding(szStringBinding,
    &MatematikaBinding);
  if (status)
    exit(status);
```

Klijent

2

```
int rezultat;
    RpcTryExcept
    {
            PosaljiBrojeveioperator(MatematikaBinding,24,'*',35);
            VratiRez(MatematikaBinding,&rezultat);
            printf("Rezultat = %d",rezultat);
    }
    RpcExcept(1)
     {
       unsigned long greska=RpcExceptionCode();
              std::cerr << “Greska " << greska  << std::endl;
    }
    RpcEndExcept

    status = RpcStringFree(&szStringBinding);
    if (status)
              exit(status);
  }
```

Klijent

- RpcStringBindingCompose funkcija kreira string za spajanje (*binding*).

RPC_STATUS RPC_ENTRY RpcStringBindingCompose(unsigned char* *ObjUuid*, unsigned char* *ProtSeq*, unsigned char* *NetworkAddr*, unsigned char* *EndPoint*, unsigned char* *Options*, unsigned char** *StringBinding*);

PRIMJER:

```
unsigned char* szStringBinding = NULL;
status = RpcStringBindingCompose(NULL,
reinterpret_cast<unsigned char*>("ncacn_ip_tcp"),
reinterpret_cast<unsigned char*>("laris6.fesb.hr"),
reinterpret_cast<unsigned char*>("4747"), NULL,
&szStringBinding);
```

Podržani *binding* protokoli

Protocol sequence	Network address	Examples
ncacn_nb_tcp	Windows NT/Windows 2000 computer name	myserver
ncacn_nb_ipx	Windows NT/Windows 2000 computer name	myserver
ncacn_nb_nb	Windows NT/Windows 2000 or Windows 95 computer name	myserver
ncacn_ip_tcp	Four-octet Internet address, or host name. On Windows XP, if the IPv6 network stack is installed, Windows XP IPv6 is fully supported and an IPv6 address is also accepted.	128.10.2.30 anynode.microsoft.com
ncacn_np	Windows NT/Windows 2000 server name (leading double backslashes are optional)	myserver \\myotherserver
ncacn_spx	IPX Internet address, or Windows NT/Windows 2000 server name	~00000001080028B30612C myserver
ncacn_dnet_nsp	Area and node syntax	4.120
ncacn_at_dsp	Windows NT/Windows 2000 computer name, optionally followed by @ and the AppleTalk zone name. Defaults to @*, the client's zone, if no zone provided	servername@zonename servername
ncacn_vns_spp	StreetTalk server name of the form item@group@organization	printserver@sdkdocs@microsoft
ncadg_mq	Windows NT/Windows 2000 server name	myserver
ncacn_http	Internet address (either four-octet or friendly name, or local Windows NT/Windows 2000 server name	128.10.2.30 somesvr@anywhere.com mylocalsvr
ncadg_ip_udp	Four-octet Internet address, or host name	128.10.2.30 anynode.microsoft.com
ncadg_ipx	IPX Internet address, or Windows NT/Windows 2000 server name	~00000001080028B30612C myserver
ncalrpc	Machine name	thismachine

Klijent

- RpcBindingFromStringBinding funkcija iz string u kojem je zapisano odredište spajanja generira handle.

  RPC_STATUS RPC_ENTRY RpcBindingFromStringBinding(unsigned char* *StringBinding*, RPC_BINDING_HANDLE* *Binding*);

  PRIMJER:

  status = RpcBindingFromStringBinding (szStringBinding, &MatematikaBinding);

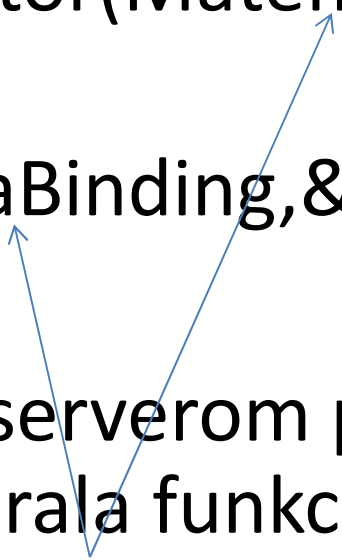
Klijent

- RpcTryExcept naredba se koristi za upravljanje izuzetcima (*exception handling*) kod RPC aplikacija. Ukoliko neka naredba unutar bloka RpcTryExcept generira izuzetak, izvršavaju se naredbe u RpcExcept bloku. RpcTryExcept naredba treba završiti s RpcEndExcept naredbom.

```
RpcTryExcept
  {
       PosaljiBrojeveioperator(MatematikaBinding,24,'+',61);
       VratiRez(MatematikaBinding,&rezultat);
       printf("Rezultat = %d",rezultat);
  }
  RpcExcept(1)
  {
       unsigned long greska=RpcExceptionCode();
      std::cerr << "Runtime reported exception " << greska  << std::endl;
   }
   RpcEndExcept
```

Klijent

```
PosaljiBrojeveioperator(MatematikaBinding,24,'
   +',61);
VratiRez(MatematikaBinding,&rezultat);
```



- Funkcije se sa RPC serverom povezuju preko handle koji je generirala funkcija RpcBindingFromStringBinding iz konekcijskog stringa.

Klijent

- Funkcija RpcStringFree oslobađa string alociran za spajanje na RPC server.

  RPC_STATUS RPC_ENTRY RpcStringFree(unsigned char** *String*);

  PRIMJER:

  status = RpcStringFree(&szStringBinding);

RPC biblioteka

- Vodite računa da u projekte uključite i dodatnu biblioteku Rpcrt4.lib sa implementacijom korištenih funkcija.