Remote Procedure Call

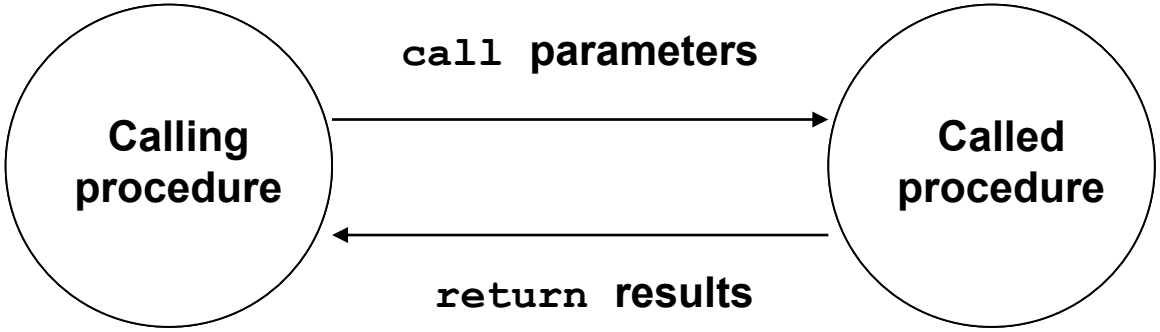
Основна идея

Процес от машина А извиква процедура от машина В. Извикващият процес се блокира докато процедурата се изпълни на другата машина и резултатите се върнат обратно.

<u>Основна цел</u>: прозрачност

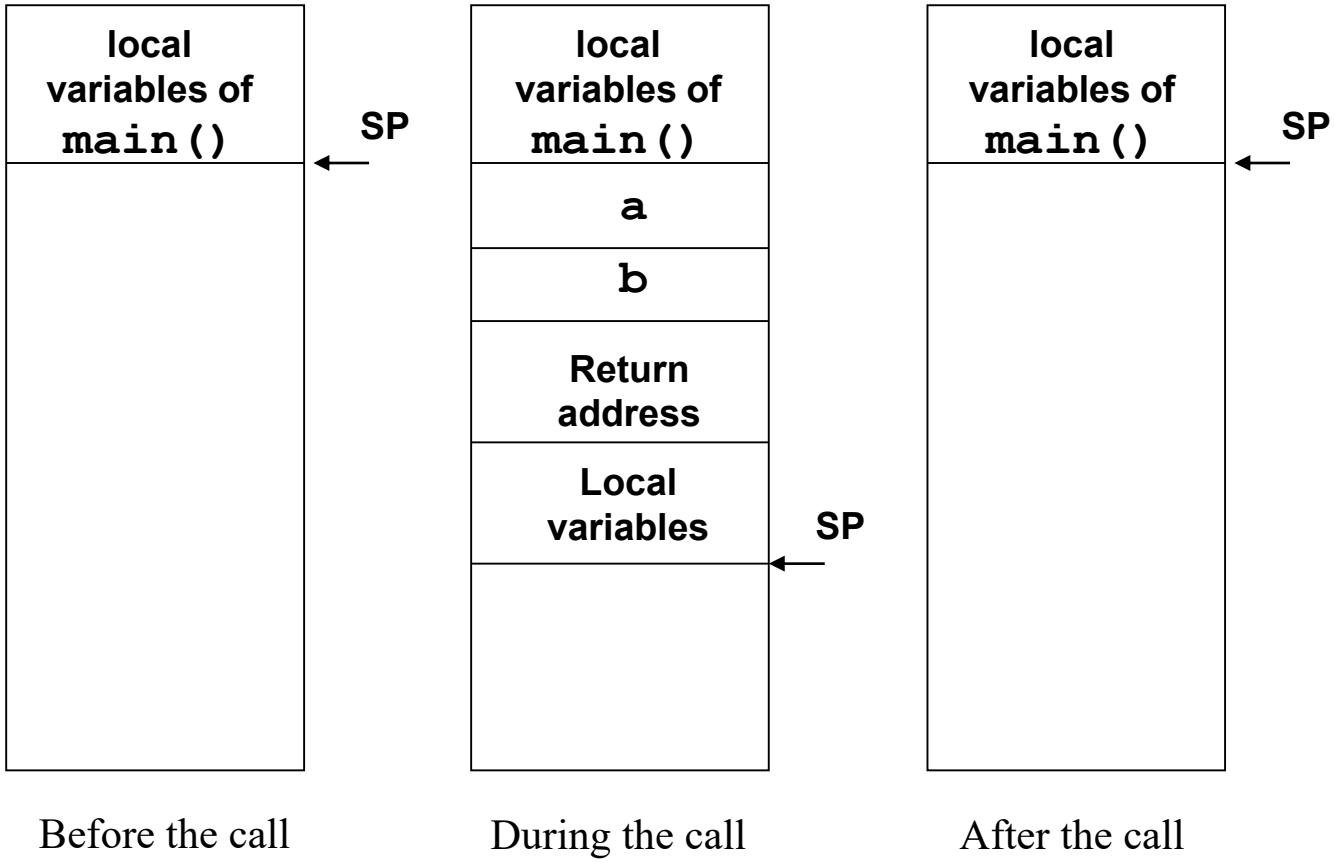
Локално извикване на процедура (LPC)

Процес изпълнява процедура в неговото адресно пространство.



Изпращане на параметри

```
s = mull ( a, b );
```



Предаване на параметри

- ***Call-by-value*** (по стойност) - Променливите се копират в стека. Модифицирането им от извиканата процедура не променя стойността на оригиналните променливи в извикващата процедура;
- ***Call-by-reference*** (по адрес) - В стека се предават само адресите на променливите. Модифицирането им от извиканата процедура води до промяна на стойността на оригиналните променливи в извикващата процедура;
- ***Call-by-copy/restore*** (копиране/възстановяване) – Комбинация от предходните методи.

Call-by-value

```
#include <stdio.h>
int a;
int main() {
        a = 3;
        f( a, 4);
        printf(“%d\n", a);
        return 0;
}
```

```
f(int x, int y) {
        // x will be 3 as passed argument
        x += a; // now a is added to x so x will be 6
        // but now nothing is done with x anymore
        a += 2*y; // a is still 3 so the result is 11
 }
```

Call-by-reference

```
#include <stdio.h>
int a;
int main() {
        a = 3;
        f( &a, 4);
        printf(“%d\n", a);
        return 0;
}
```

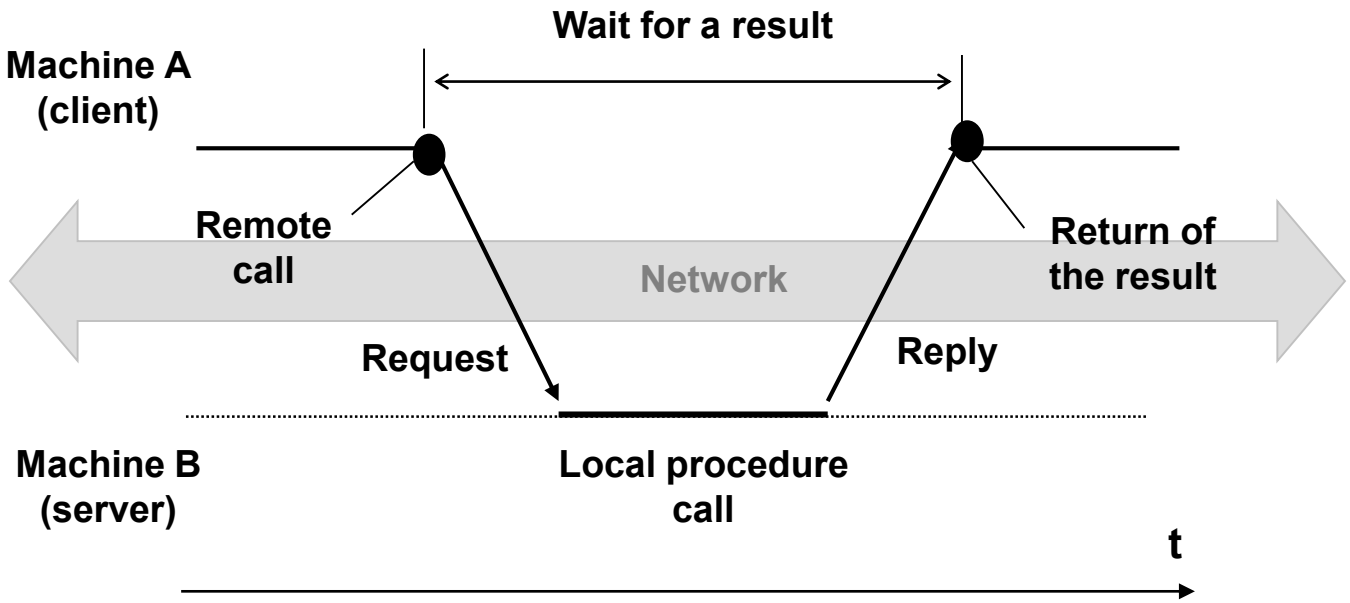
```
f(int *x, int y) {
        // x will be 3 as passed argument
        *x += a; // a is 6 now
        a += 2*y; // so the result is 14
 }
```

Call-by-copy/restore

```
int a;
void unsafe(int x) {
    x= 2; //a is still 1
    a= x; //a is now 2
}
//function ends so the value of x is now stored in a -> value of a is now 2

int main() {
    a= 1;
    unsafe(a);
    //when this ends the value of a will be 2
    printf("%d\n", a); //prints 2
}
```

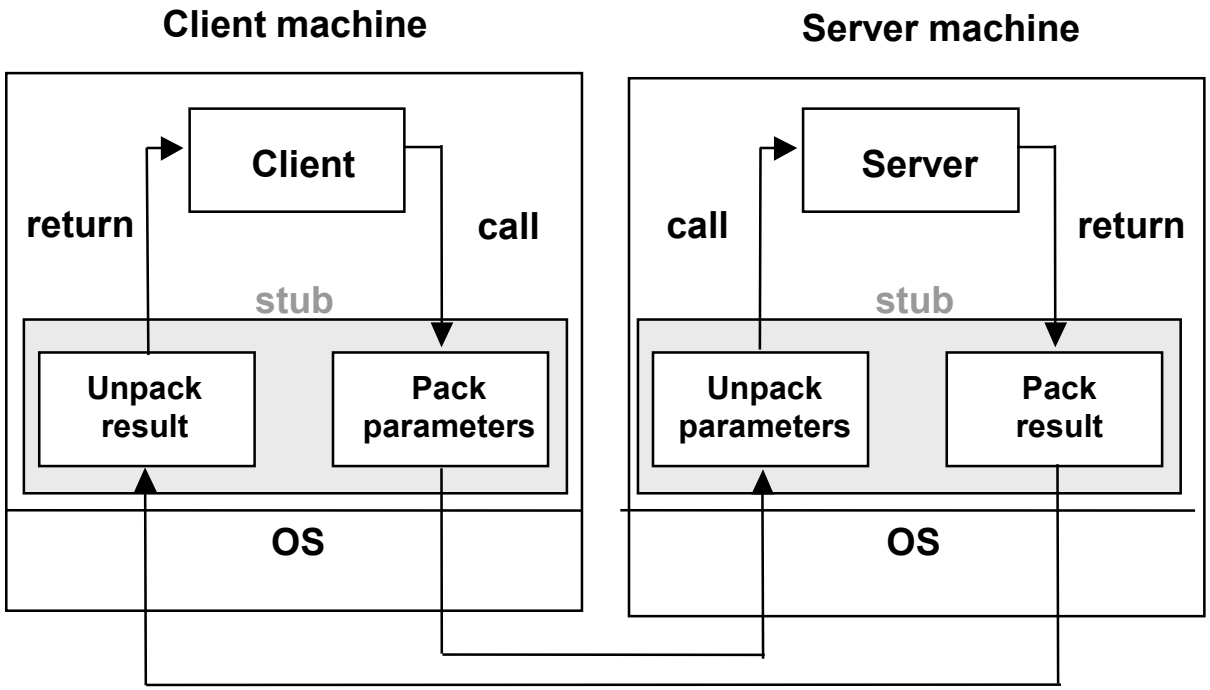
Remote Procedure Call

- Извикващият и извикваният процес са независими един от друг;
- Няма общ стек;
- Използва се комуникационен модел *request-reply*;
- Използват се съобщения между процесите.



RPC комуникационен интерфейс

- Базиран е на библиотечни функции – **стъбове** (***stubs)***;
- Клиентът и сървърът комуникират чрез два стъба, по един за всеки от тях.

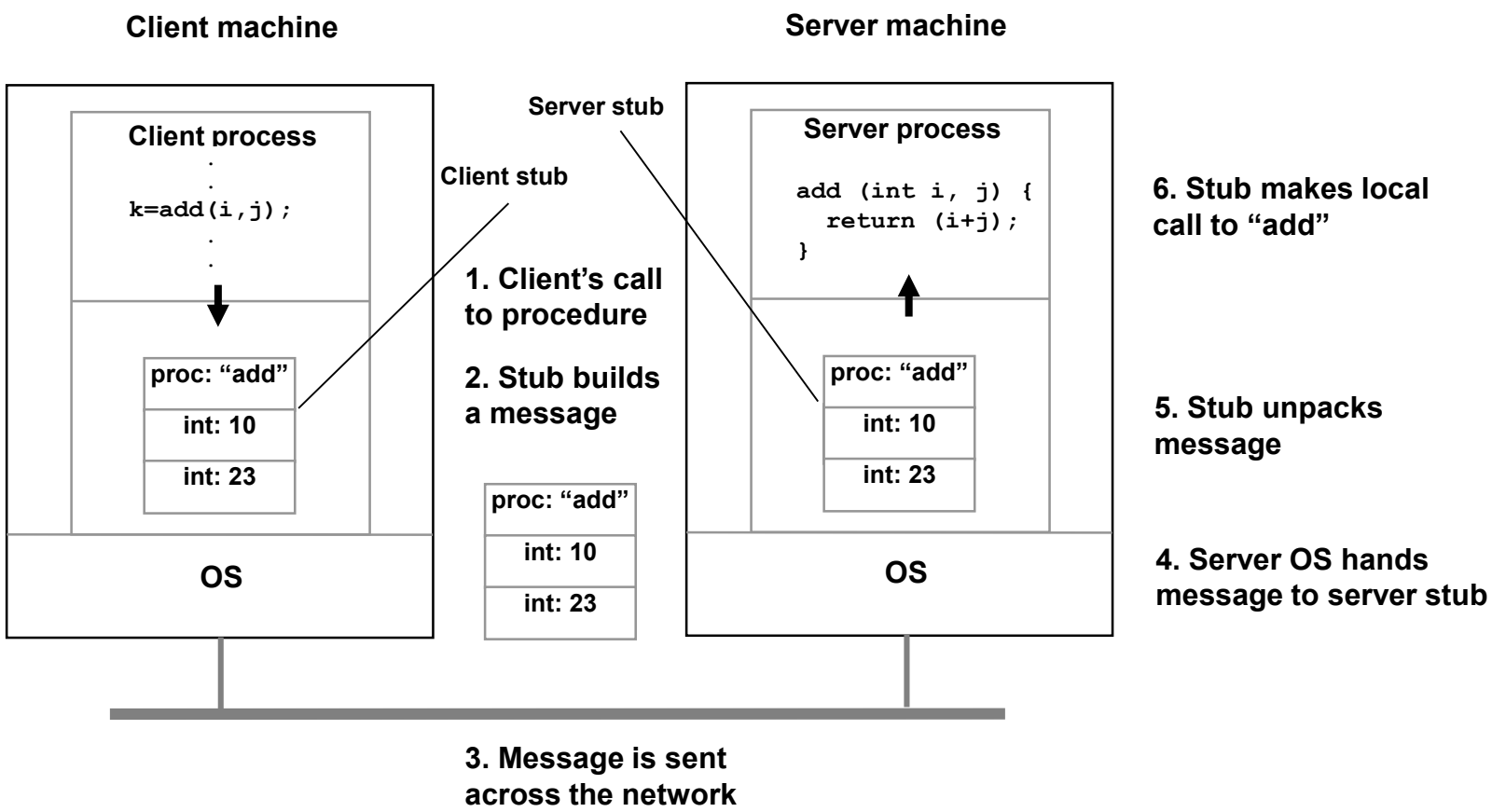


RPC алгоритъм

1. Клиентската процедура извиква клиентския стъб;
2. Клиентския стъб подготвя съобщения и изпълнява системно извикване към локалната ОС;
3. Клиентската ОС изпраща съобщение към сървърната ОС;
4. Отдалечената ОС предава съобщението на сървърния стъб;
5. Сървърният стъб разпакетирва параметрите и извиква сървъра;
6. Сървърният процес обработва данните и връща резултата към сървърния стъб;
7. Сървърният стъб пакетира резултата в съобщение и изпълнява системно извикване към локалната ОС;
8. Сървърната ОС изпраща съобщението към клиентската ОС;
9. Клиентската ОС предава съобщението на клиентския стъб;
10. Клиентският стъб разпакетирва резултата и връща изпълнението към клиентската програма.

RPC пример

```
k = add ( 10 , 23 );
```



RPC пакетиране на параметрите

Пакетирането на параметрите в съобщение– ***parameter marshaling.***

<u>Две важни задачи за решаване:</u>

- Различни формати на данните;
- Изпращане на указатели.

RPC формати на данни

Проблеми:

- Различни архитектури на клиента и сървъра;
- Различни символни таблици;
- Различни числови формати;
- Различно подреждане на байтовете.

Пример за подреждане на данните

Клиентът (Intel) изпраща към сървъра (SPARK) в съобщение integer 5 и string “MARS”. Всеки параметър изисква 4 байта.

3 **0**	2 **0**	1 **0**	0 **5**
7 **S**	6 **R**	5 **A**	4 **M**

Little-endian (Intel) - 5

0 **5**	1 **0**	2 **0**	3 **0**
4 **M**	5 **A**	6 **R**	7 **S**

*Big-endian (SPARC) – 5 * 2 ^{24}*

0 **0**	1 **0**	2 **0**	3 **5**
4 **S**	5 **R**	6 **A**	7 **M**

The inverted message

Решаване на проблема

Използване на унифицирана канонична форма за представяне на данните – *External Data Representation (XDR):*

- Машинно независим формат;
- Big-endian подреждане на данните;
- IEEE представяне на числата с плаваща запетая;
- ASCII представяне на символите.

Представянето в канонична форма - ***serialization.***

Обратното представяне – ***deserialization***.

RPC и указатели

Проблем: указателите са валидни само в локалното адресно пространство на процеса.

Решения:

- Изпращане само на стойности, не на указатели;
- Изпращане на указатели към сървърния стъб и генериране на специален код за обработка на тези указатели.

Локализиране на сървъра

- Адресът на сървъра е твърдо зададен в кода;
- Динамично свързване (dynamic binding).

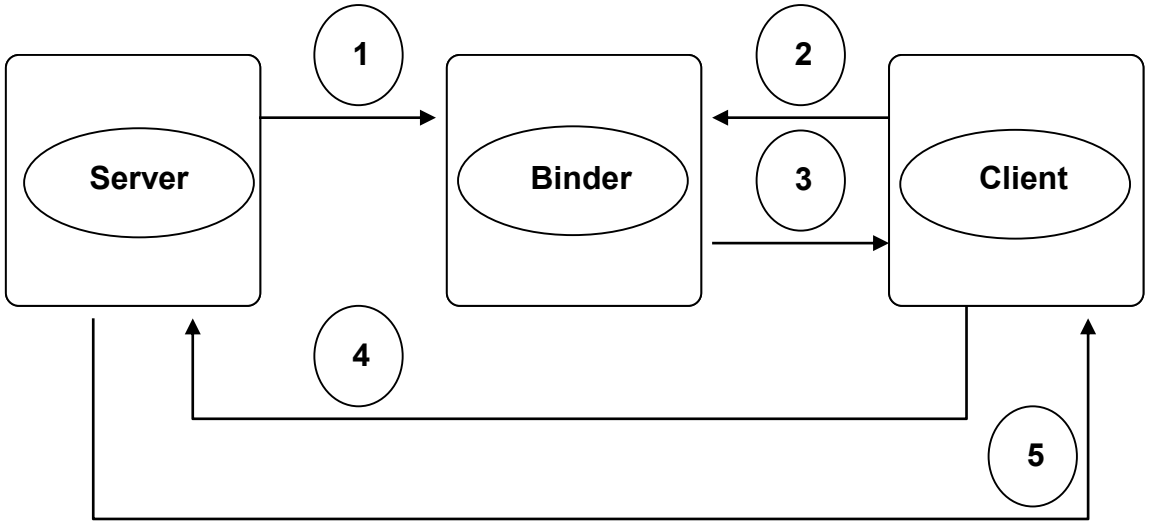
Динамично свързване

- Създава се формална спецификация на сървъра и генериране на стъба.

```
#define MAX_PATH    128  // max size of a filename
#define BUF_SIZE    256  // max size of data buffer


Specification of file_server, version 2.1;

  int  read (in char name [MAX_PATH], out char buf [BUF_SIZE],
          in long bytes, in long position);       // read from a file
  int  write (in char name [MAX_PATH], in char buf [BUF_SIZE],
          in long bytes, in long position);       // write into a file
  int  create (in char name [MAX_PATH], in int mode); // create a file
  int  delete (in char name [MAX_PATH]);          // delete a file

end; // end of specification
```

Динамично свързване

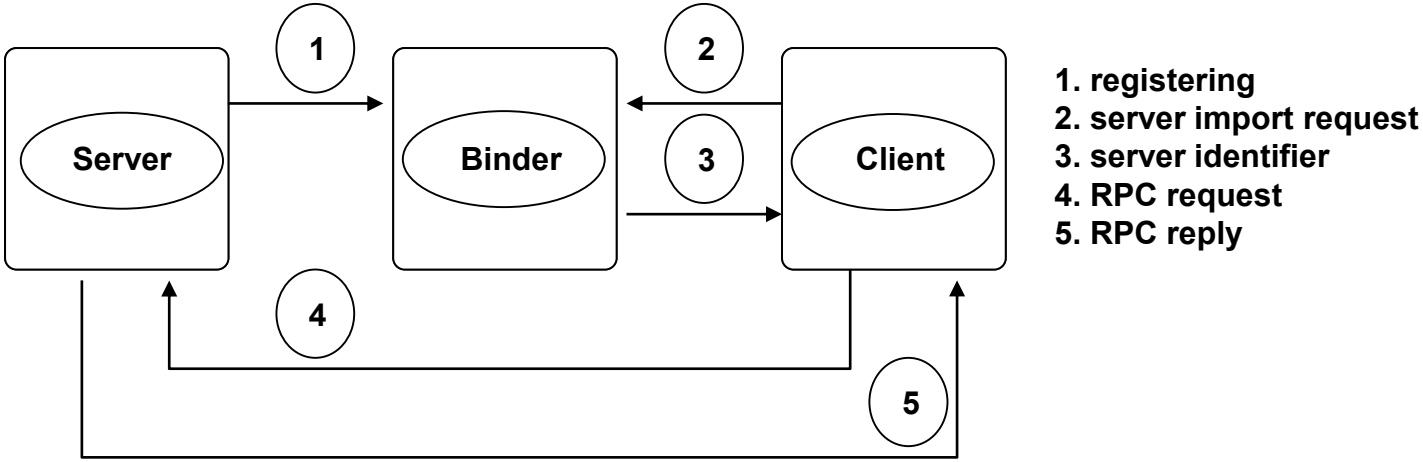
- Регистрация на сървъра:
  - Име;
  - Идентификатор;
  - Номер на версия;
  - Дескриптор.



1. registering
2. server import request
3. server identifier
4. RPC request
5. RPC reply

Динамично свързване

- Локализиране на сървъра:
  - Клиентът изпраща заявка към **binder** за получаване на информация за сървъра;
  - B**inder** връща дескриптора на сървъра и уникален идентификатор;
  - Клиентът използва този дескриптор за адресиране на сървъра.



Динамично свързване

Предимства:

- Висока степен на надеждност;
- Мигриране на сървъра;
- Балансирано натоварване на сървъра;
- Автентикация.

Недостатъци:

- Натоварване на **binder** при много заявки;
- Необходимост от синхронизиране между множество **binders**.

Семантика на грешките при RPC

<u>RPC типове грешки</u>:

- RPC клиентът не може да локализира сървъра;
- Загуба на заявки;
- Загуба на отговор;
- Отказ на сървъра;
- Отказ на клиента.

RPC сървър не може да бъде локализиран

<u>Възможни причини:</u>

- Отказ на сървъра;
- Пререгистиране на сървъра с нова версия или интерфейс.

Грешките трябва да се обработят с използване на изключения (signal handlers в C). Но се губи прозрачноста).

Загуба на заявки

Решение:

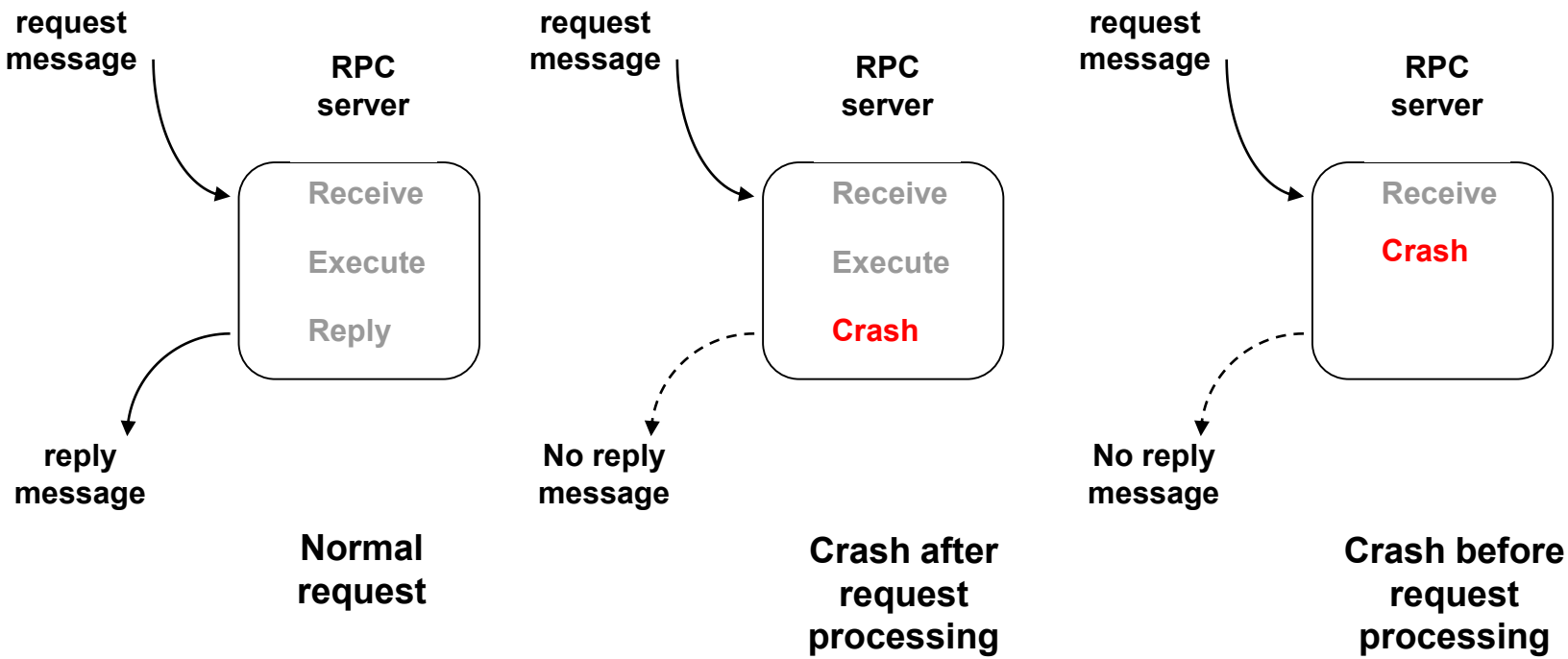
- Използване на таймер. След изтичането му се изпраща повторно заявката;
- В случай на много изгубени заявки – генериране на изключение.

Загуба на отговор

Решение:

- Използване на таймер. След изтичането му се изпраща повторно заявката;
- Създаване на заявки, които са *idempotent* и използване на таймер;
- Номериране на заявките с цел избягване на повторното им обслужване.

Отказ на сървъра



Отказ на сървъра

Семантика на решенията:

- *At-least-once*. Клиентската ОС се опитва а получи отговор. Чака или се свързва към друг сървър;
- *At-most-once*. След изтичане на таймер ОС връща грешка на клиента; може и изобщо да не се изпълни.
- *No guarantee*. Клиентската заявка може да бъде обработена 0 или безкраен брой пъти. Опростена семантика;
- *Exactly once*. Изисква гарантиране; Сложна семантика.

Отказ на клиента

Изпраща заявка за обработка и отказва - orphan computation

Решения:

- *Extermination*. Преди изпращане на заявка се проверява лог файла и клиентският стъб записва информация в него след изпращане; Използва се след рестартиране на клиента.
- *Reincarnation*. Времето за работа се разделя на т.н. **епохи** между отказите. След рестартиране се изпраща броадкаст към всички машини за начало на нова епоха;
- *Gentle reincarnation*. След получаване на съобщение за нова епоха, всяка машина се опитва да се свърже с източника. Ако не успее – клиентския процес се унищожава;
- *Timer expiration*. За всяка заявка се дефинира фиксиран интервал от време. Ако не може да завърши - изисква се нов.

Проблем:

Унищожаването на **orphan** процес - ами ако има заключване на файлове?

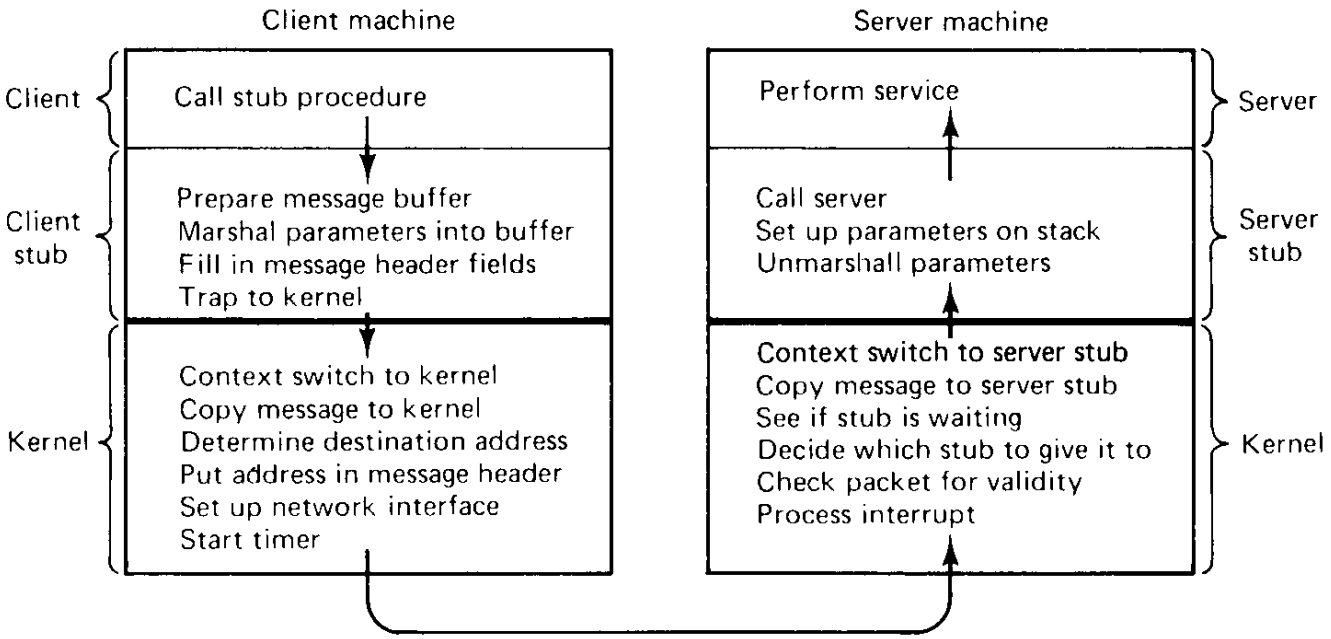
Реализация

1. Какъв протокол да се използва- conneclion или connectionless.
2. Стандартен или специално проектиран за RPC (IP, TCP).
3. Потвърждения. Ако е разбит на части, трябва ли за всяка част да се потвърждава? Стратегии:

- **Stop-and-wait** – изпраща, чака потвърждение (като TCP).
- **Blast protocol** – изпраща всички пакети и получава общ отговор. Ако се изгуби пакет - сървърът буферира получените до пристигане на цялото съобщение – указва кой пакет да се препрати - **selective repeat**.
- **Overrun error** – невъзможност да се приемат пакети без времеви интервал между тях (изчакване на изпращача след изпращане).

Критичен път

Последователност от действия за всяка RPC влияе на производителността.



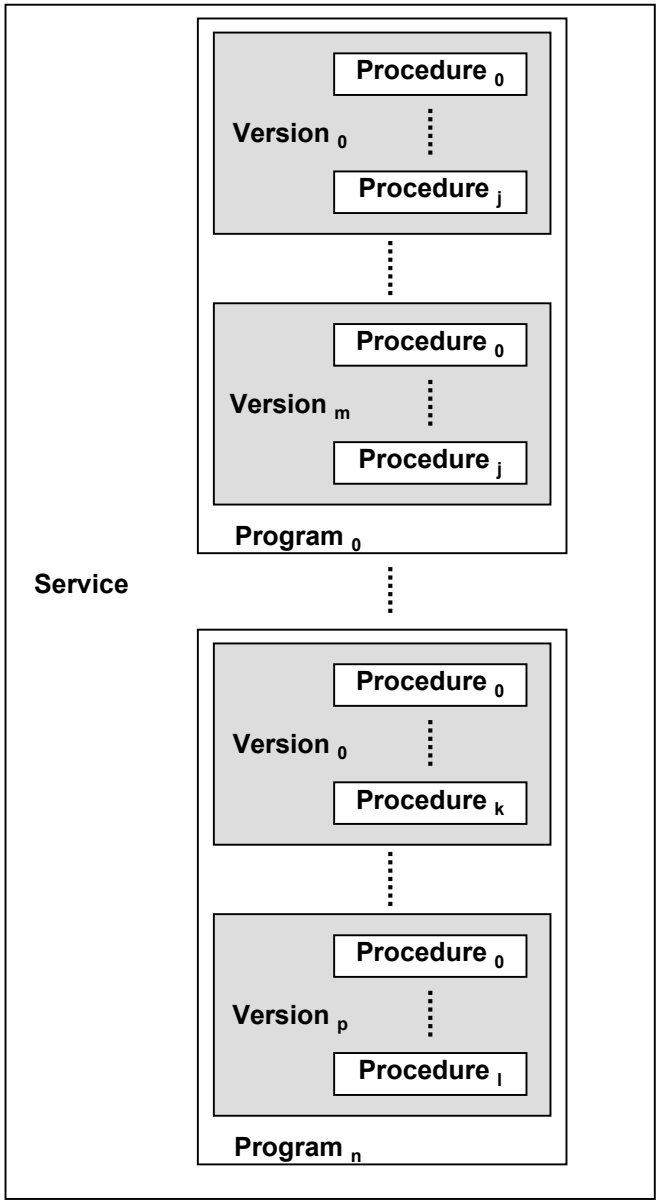
Основен проблем е редица копирания (от 1 до 8 в различни системи). В най-добрия са 2; от памет в мрежов контролер и при сървъра от мрежов контролер в паметта.

RPC реализация - SunRPC

Идентификация на RPC процедура:

- *Program number*. Програма е множество от процедури;
- *Version number*;
- *Procedure number*.

Структура на услугите



SunRPC – области на програмите

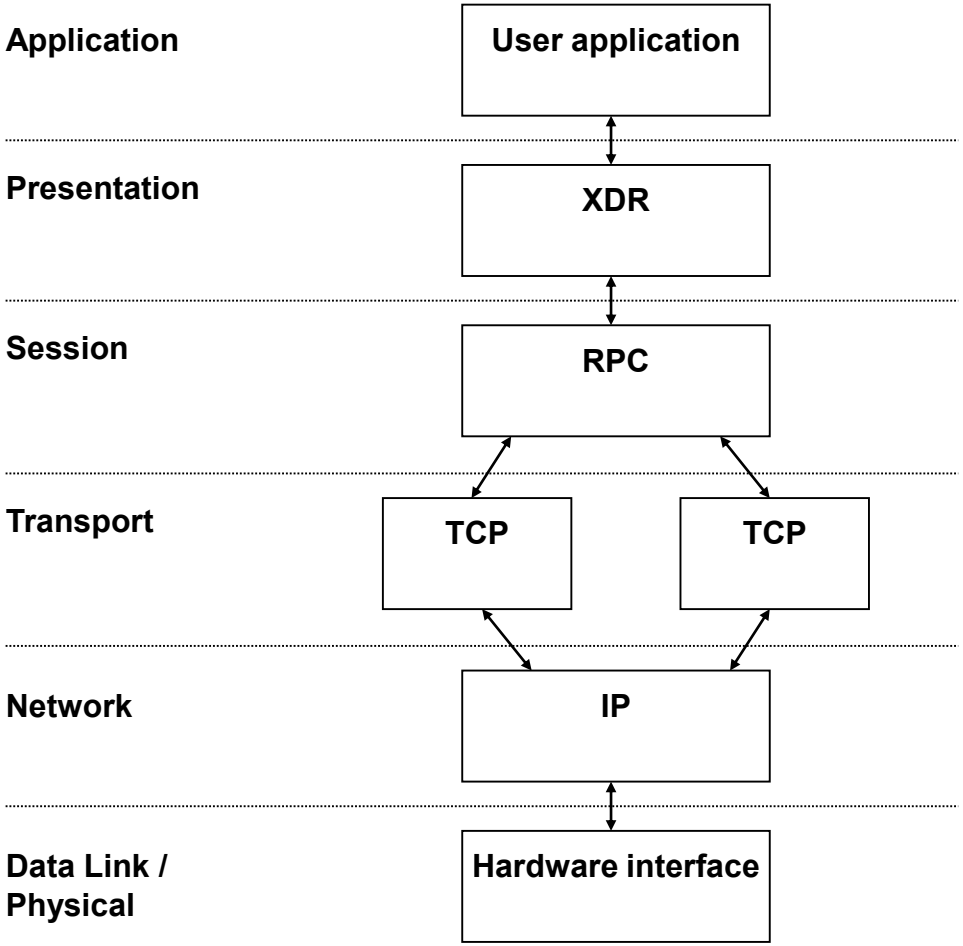
Range (Hexadecimal)	Administrator	Lifetime	Distribution
00000000 - 1fffffff	Sun	Permanent	Public
20000000 - 3fffffff	User	Unknown	Local
40000000 - 5fffffff	Developer	Transient	Local
60000000 - 7fffffff	Reserved		
80000000 - 9fffffff	Reserved		
a0000000 - bfffffff	Reserved		
c0000000 - dfffffff	Reserved		
e0000000 - ffffffff	Reserved		

SunRPC – файлът */etc/rpc*

linux.tu-varna.acad.bg - PuTTY

```
#
# rpc 88/08/01 4.0 RPCSRC; from 1.12   88/02/07 SMI
#
portmapper      100000  portmap sunrpc
rstatd          100001  rstat rstat_svc rup perfmeter
rusersd         100002  rusers
nfs             100003  nfsprog
ypserv          100004  ypprog
mountd          100005  mount showmount
ypbind          100007
walld           100008  rwall shutdown
yppasswdd       100009  yppasswd
etherstatd      100010  etherstat
rquotad         100011  rquotaprog quota rquota
sprayd          100012  spray
3270_mapper     100013
rje_mapper      100014
selection_svc   100015  selnsvc
database_svc    100016
rexd            100017  rex
alis            100018
sched           100019
llockmgr        100020
```

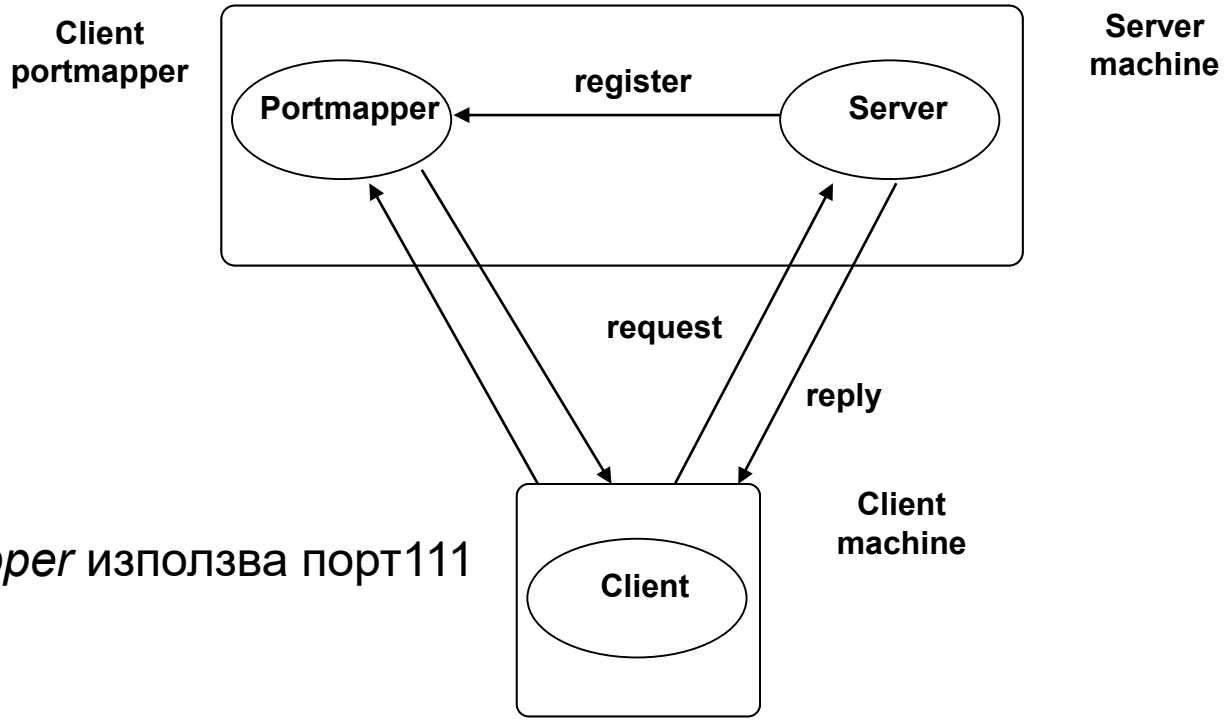
SunRPC - протоколи



SunRPC – семантика на протоколите

- **В случай на UDP**. При отговор се използва семантика *at-least-once*. Отказ на отговор означава 0 или повече пъти изпълнение;
- **В случай на TCP**. При отговор се използва семантика *exactly one*. Отказ на отговор не означава, че процедурата не е била изпълнена.

SunRPC – свързване към сървър



Portmapper използва порт111

SunRPC – свързване към сървър

Алгоритъм:

1. Сървърът създава порт при стартиране;
2. Сървърният стъб регистрира към локалния *portmapper* неговият номер и версия;
3. Portmapper съхранява сървърния номер и версия;
4. Клиентът се свързва към *portmapper* за получаване на номера и порта на сървъра;
5. След получаването им, клиентът изпраща заявка.

SunRPC – регистриране

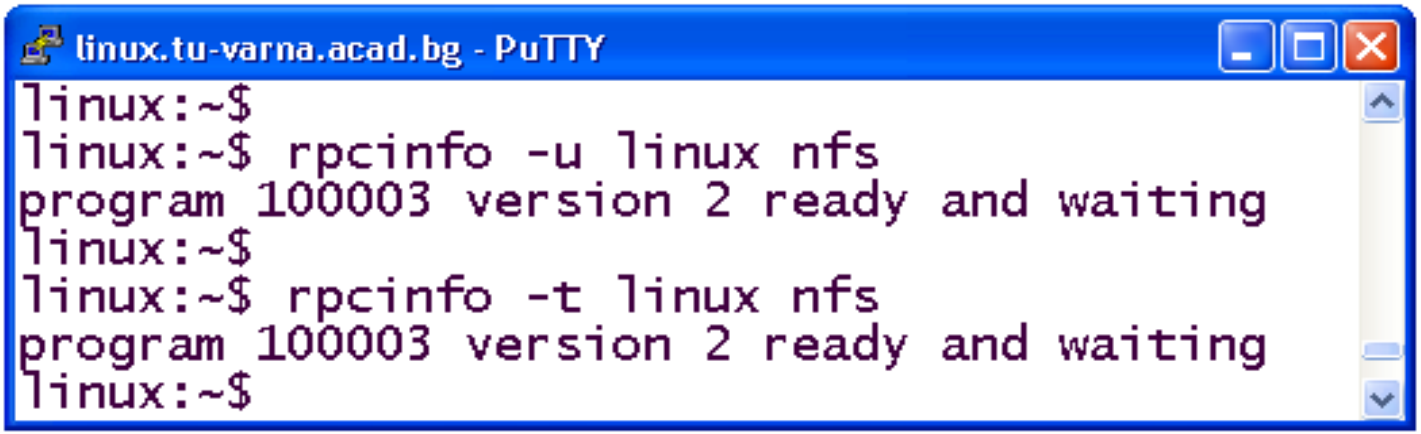
 # rpcinfo –p [host]

linux.tu-varna.acad.bg - PuTTY

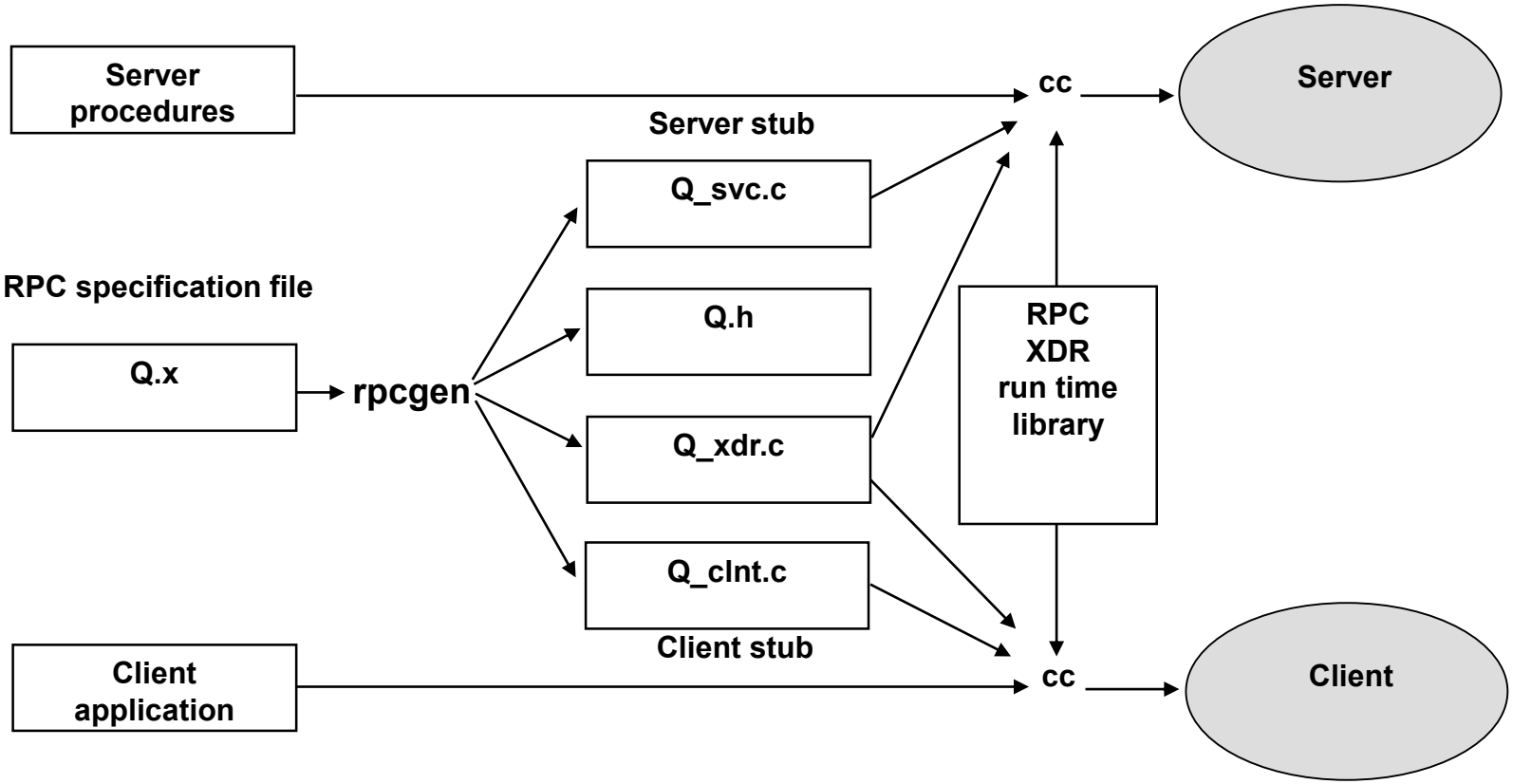
```
linux:~$
linux:~$ rpcinfo -p
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100005    1   udp    704  mountd
    100005    2   udp    704  mountd
    100005    1   tcp    707  mountd
    100005    2   tcp    707  mountd
    100003    2   udp   2049  nfs
    100003    2   tcp   2049  nfs
linux:~$ rpcinfo -p dpx20
   program vers proto   port
    100000    2   tcp    111  portmapper
    100000    2   udp    111  portmapper
    100068    2   udp   1036
    100068    3   udp   1036
    100068    4   udp   1036
    100003    2   udp   2049  nfs
    100005    1   udp    768  mountd
    100005    1   tcp    778  mountd
    100024    1   udp    789  status
    100024    1   tcp    791  status
    100021    1   tcp    986  nlockmgr
    100021    1   udp    988  nlockmgr
    100021    3   tcp    991  nlockmgr
    100021    3   udp    993  nlockmgr
    100020    1   udp   1001  llockmgr
    100020    1   tcp   1003  llockmgr
    100021    2   tcp   1006  nlockmgr
linux:~$
```

SunRPC – регистриране

```
# rpcinfo -u host program [version]
# rpcinfo -t host program [version]
```



SunRPC – генериране на стъб (*rpcgen*)

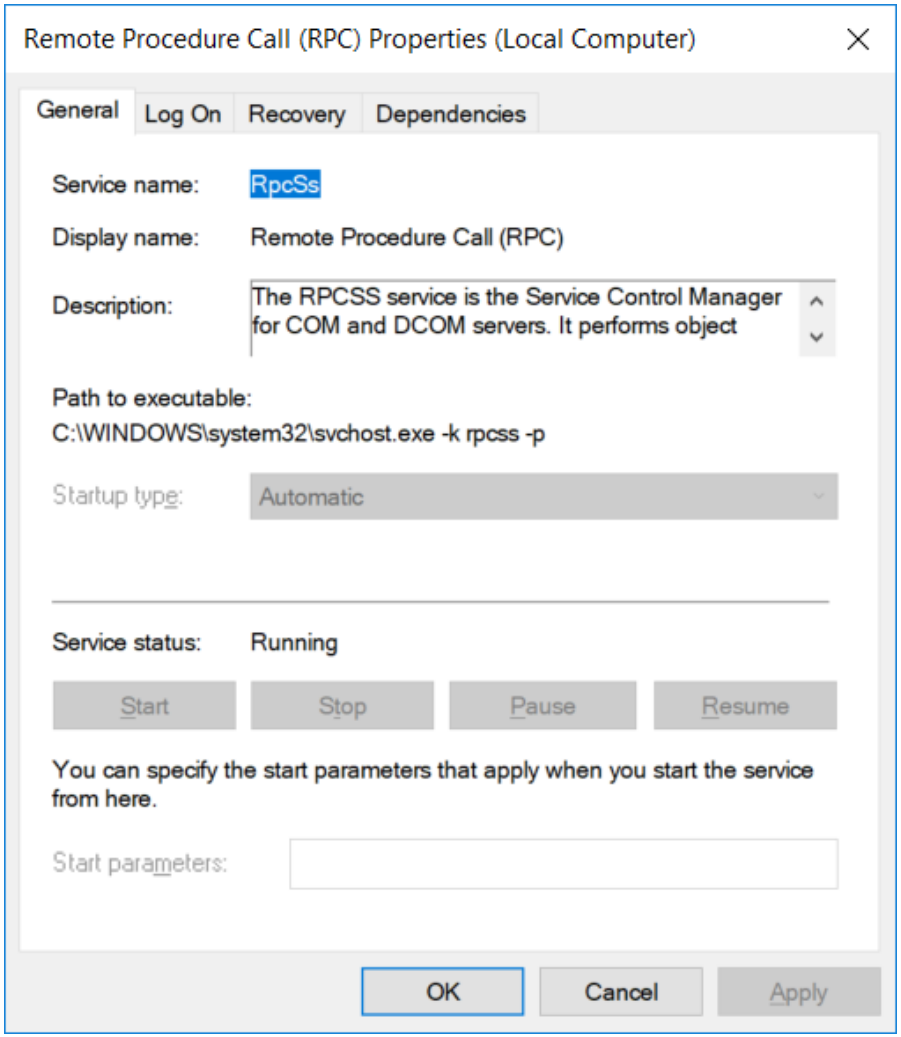


Спецификация на протокола

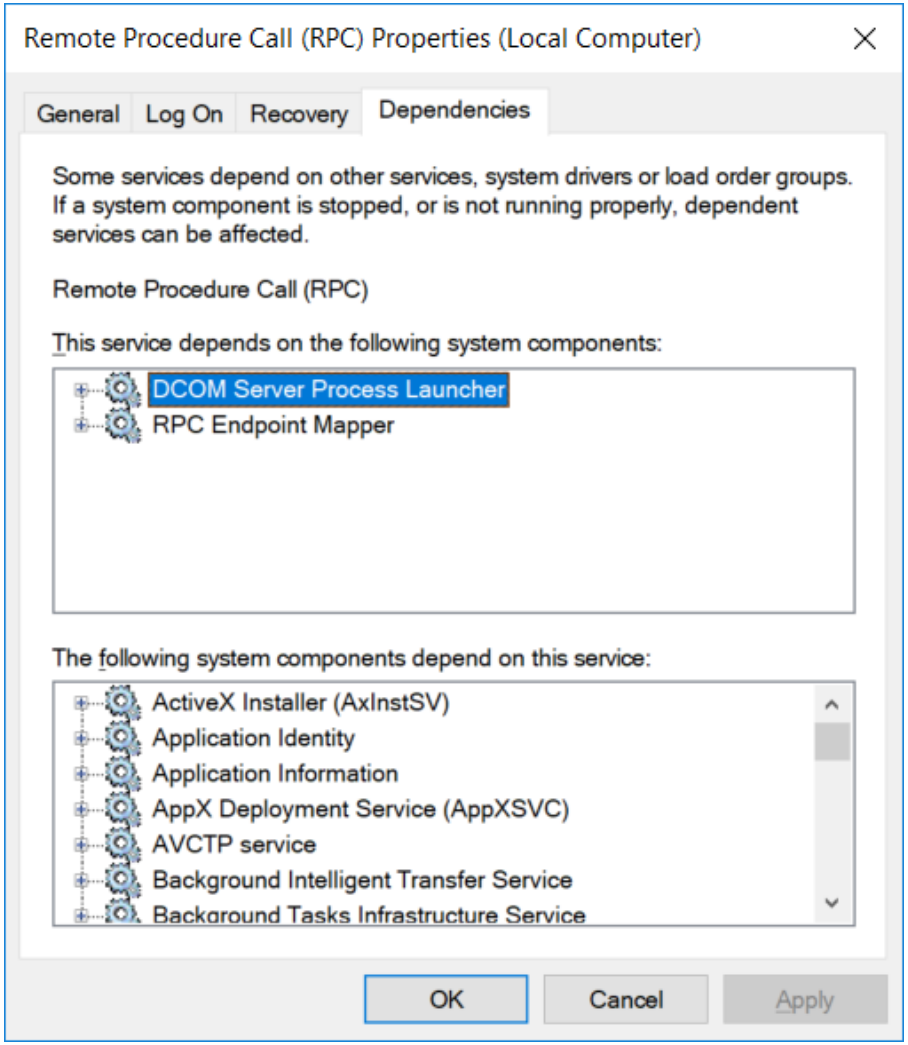
Използва се RPCL (RPC Language)

```
/*  time.x  */
/* Defines two procedures
        gettime_1 - returns the binary date and time
        getstrtime_1 - takes a binary date and returns a string
*/

program TIMEPROG {
        version TIMEVERS {
                long GETTIME(void) = 1;   /* procedure number = 1 */
                string GETSTRTIME(long) = 2;/* procedure number = 2 */
        } = 1;          /* version number = 1 */
} = 0x20000001;         /* program number = 0x20000001 */
```

Windows RPC


Remote Procedure Call (RPC) Properties (Local Computer)
General
Log On
Recovery
Dependencies
Service name:
RpcSs
Display name:
Remote Procedure Call (RPC)
Description:
The RPCSS service is the Service Control Manager for COM and DCOM servers. It performs object
Path to executable:
C:\WINDOWS\system32\svchost.exe -k rpcss -p
Startup type:
Automatic
Service status:
Running
Start
Stop
Pause
Resume
You can specify the start parameters that apply when you start the service from here.
Start parameters:
OK
Cancel
Apply

Windows RPC

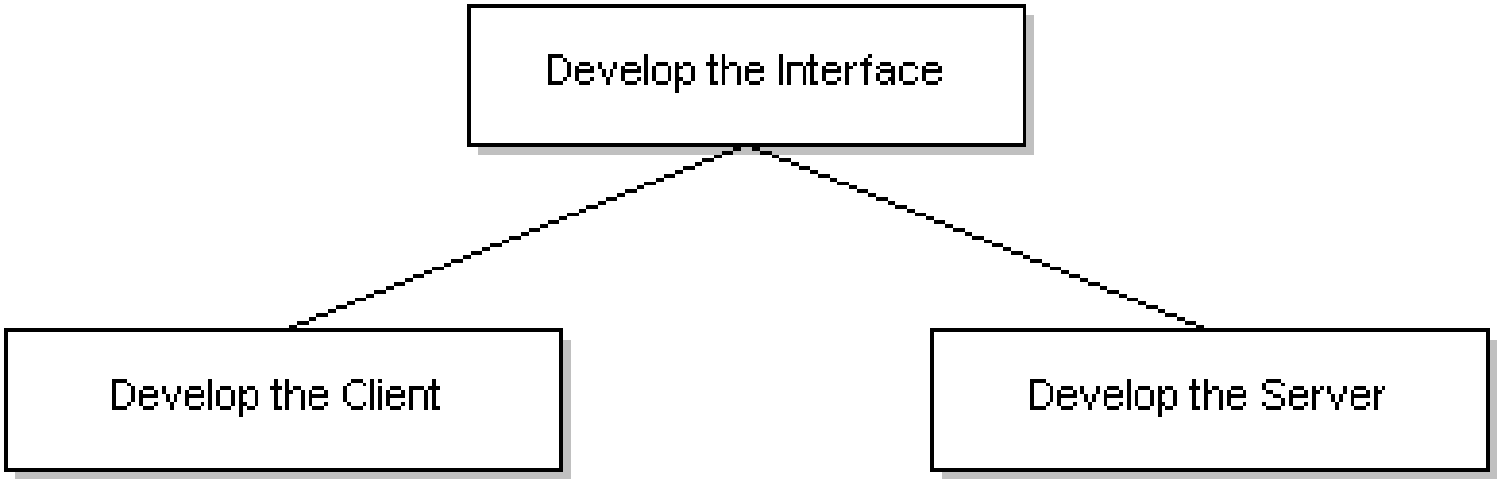

Remote Procedure Call (RPC) Properties (Local Computer)
General
Log On
Recovery
Dependencies
Some services depend on other services, system drivers or load order groups. If a system component is stopped, or is not running properly, dependent services can be affected.
Remote Procedure Call (RPC)
This service depends on the following system components:
DCOM Server Process Launcher
RPC Endpoint Mapper
The following system components depend on this service:
ActiveX Installer (AxInstSV)
Application Identity
Application Information
AppX Deployment Service (AppXSVC)
AVCTP service
Background Intelligent Transfer Service
Background Tasks Infrastructure Service
OK
Cancel
Apply

Windows RPC

Administrator: Command Prompt

```
C:\WINDOWS\system32>sc enumdepend rpcss 100000
[SC] EnumDependentServices: entriesread = 183

SERVICE_NAME: GoogleChromeElevationService
DISPLAY_NAME: Google Chrome Elevation Service
        TYPE               : 10  WIN32_OWN_PROCESS
        STATE              : 1  STOPPED
        WIN32_EXIT_CODE    : 1077  (0x435)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0

SERVICE_NAME: PrintWorkflowUserSvc_1d07169e
DISPLAY_NAME: PrintWorkflow_1d07169e
        TYPE               : f0   ERROR
        STATE              : 4  RUNNING
                             (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0

SERVICE_NAME: DevicesFlowUserSvc_1d07169e
DISPLAY_NAME: DevicesFlow_1d07169e
        TYPE               : e0  USER_SHARE_PROCESS INSTANCE
        STATE              : 1  STOPPED
        WIN32_EXIT_CODE    : 1077  (0x435)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

Windows RPC - създаване на приложение

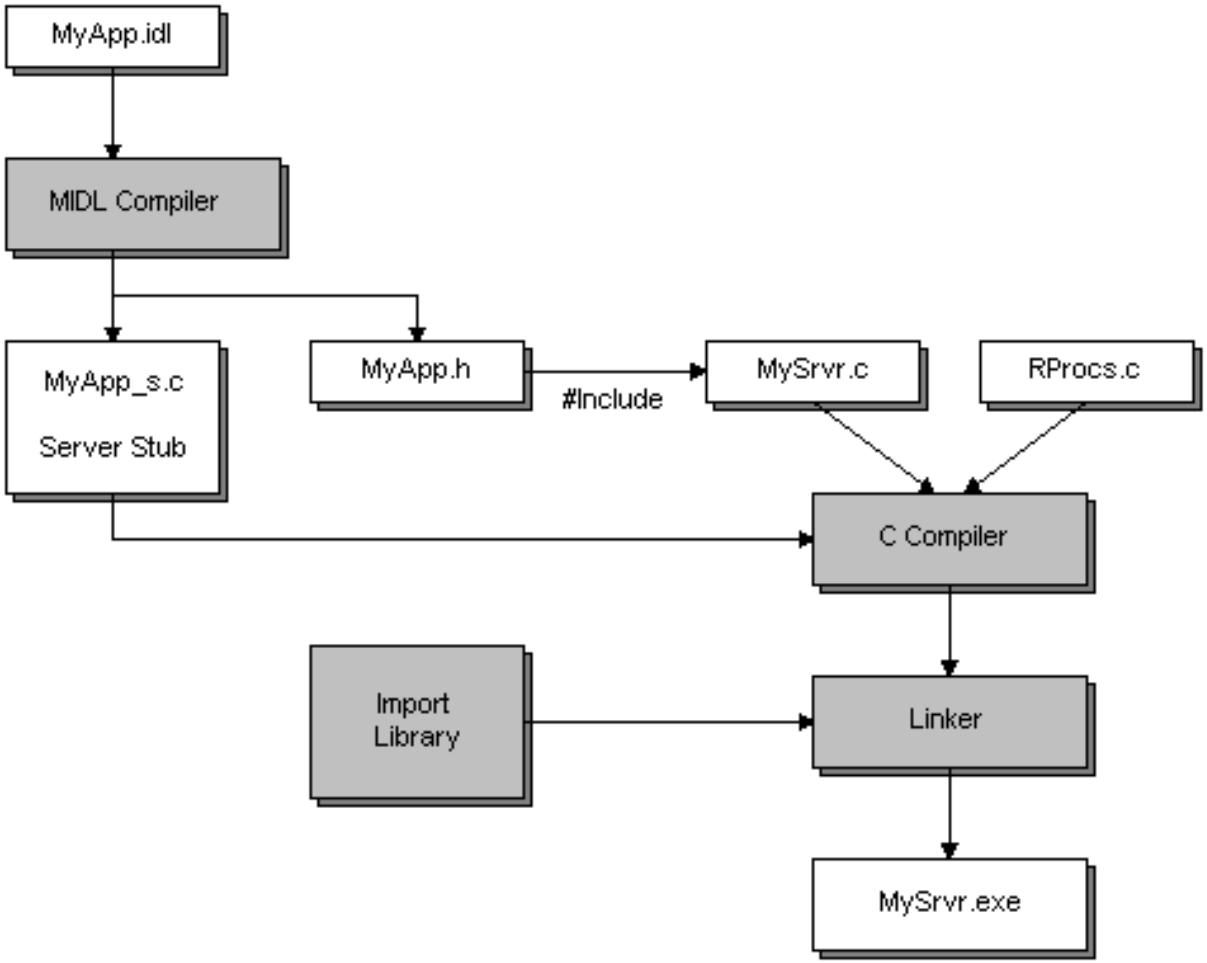


Windows RPC - създаване на интерфейс (MIDL)

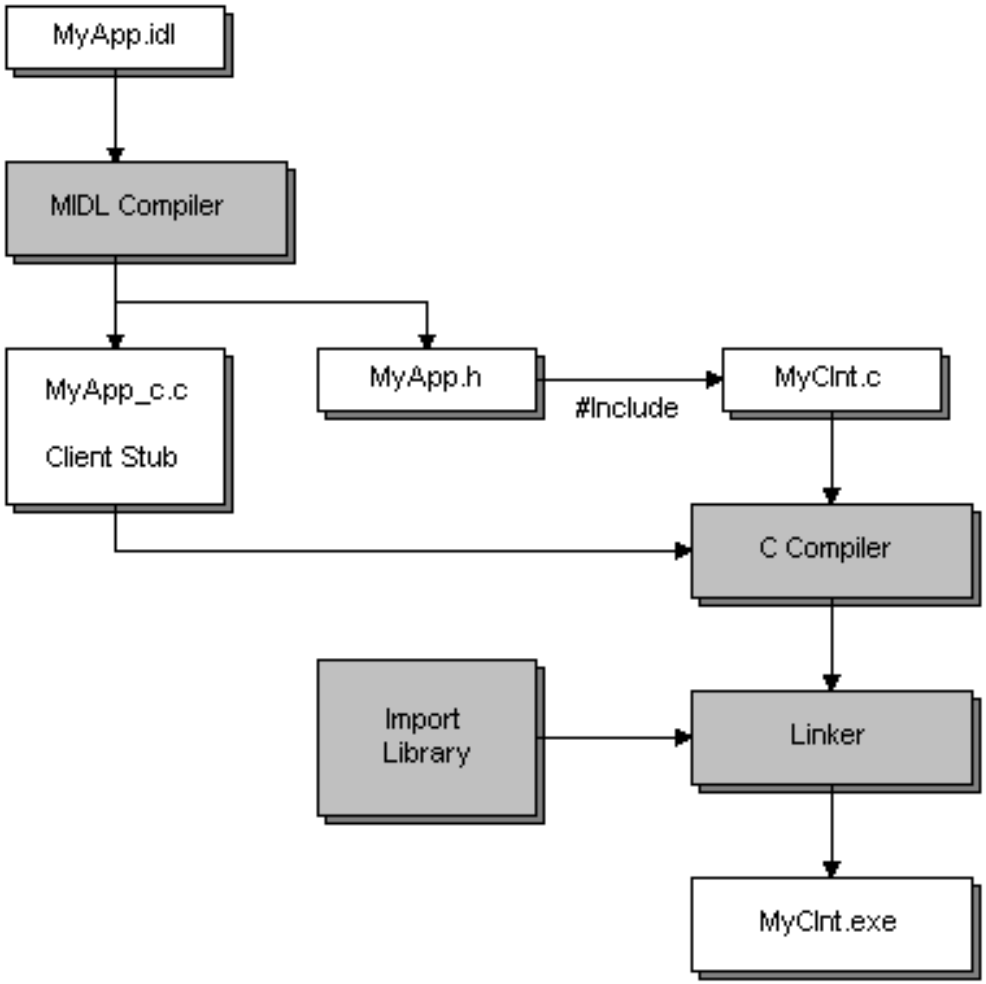
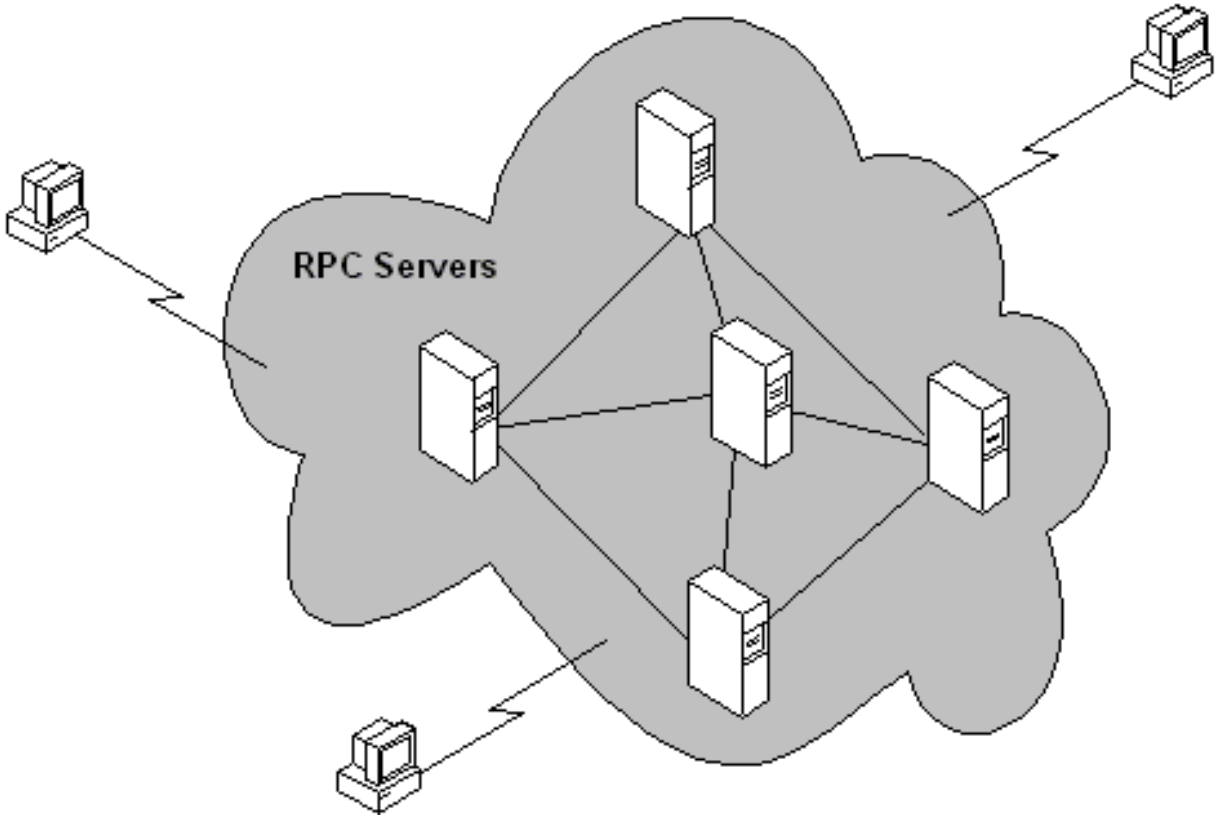
Използва се IDL (Interface Definition Language)

```
[
  uuid(ba209999-0c6c-11d2-97cf-00c04f8eea45),
  version(1.0)
]
interface MyInterface
{
  const unsigned short INT_ARRAY_LEN = 100;

  void MyRemoteProc(
      [in] int param1,
      [out] int outArray[INT_ARRAY_LEN]
  );
}
```

Windows RPC - създаване на сървър



Windows RPC - създаване на клиент



Разпределена облачна система