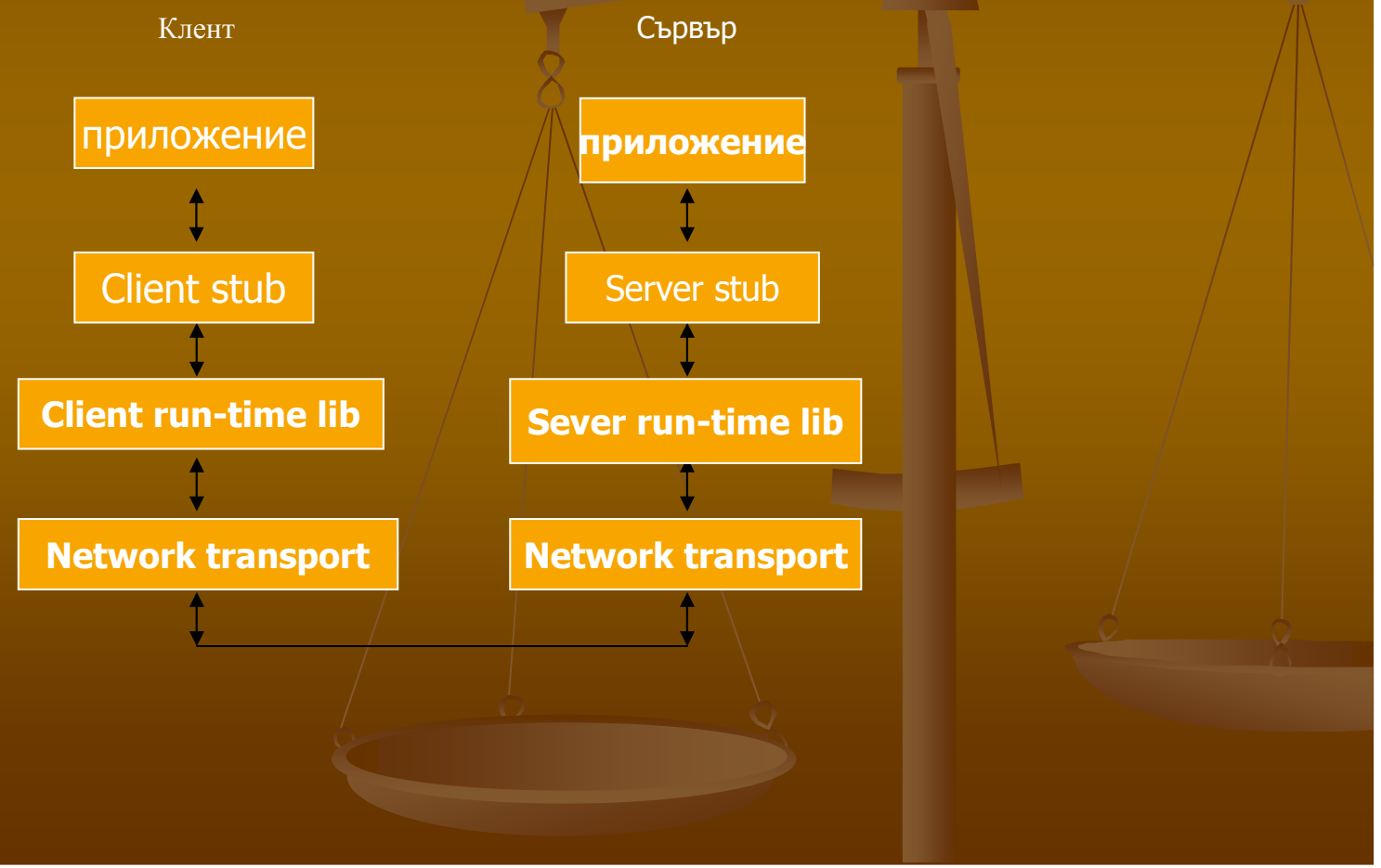
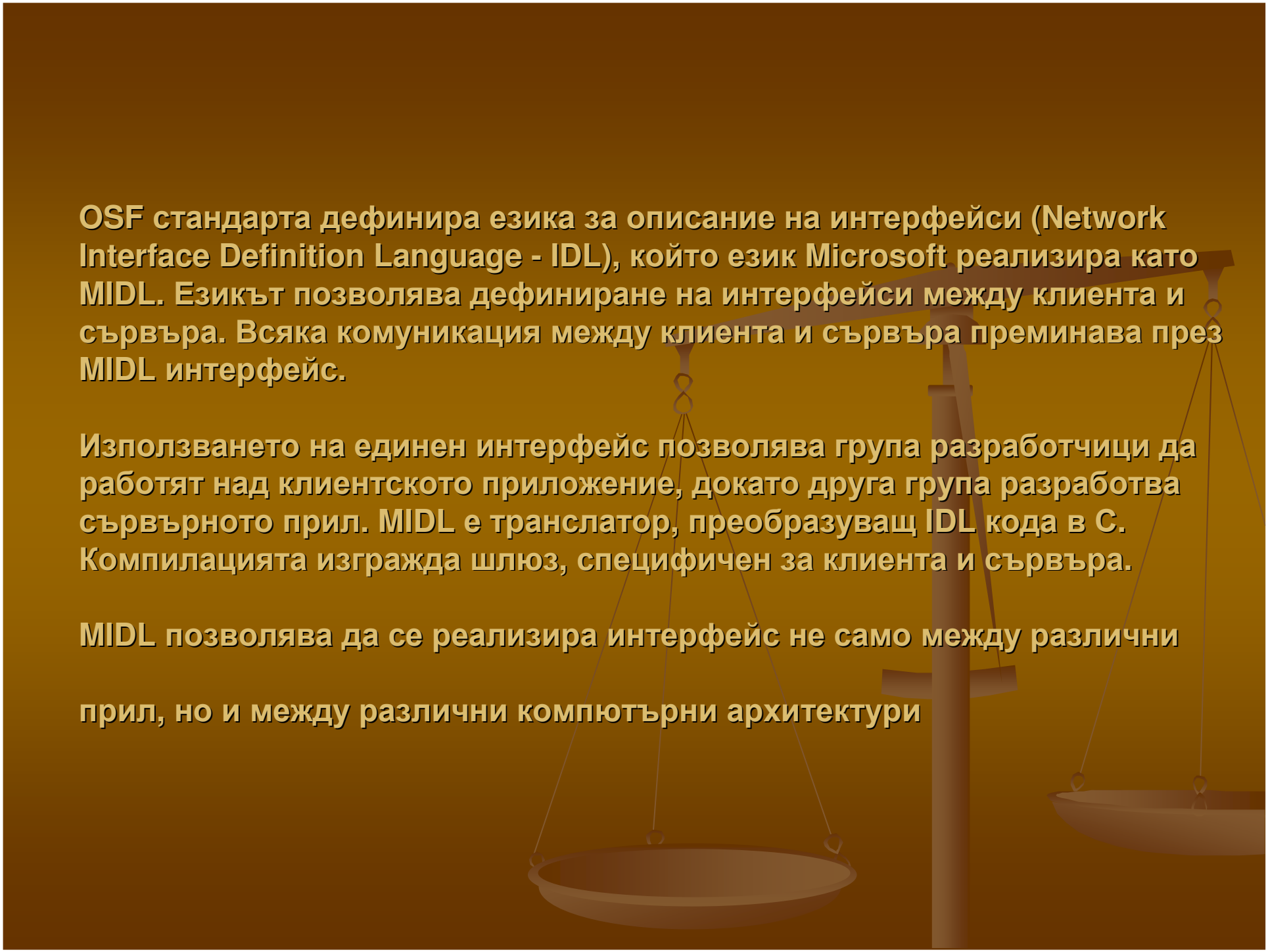
Отдалечено повиквани процедури (Remote Procedure Calls – RPC)



•Моделът RPC е дефиниран и стандартизиран в Open Software Foundation’s (OSF) Distributed Computing Environment (DCE).
* освен Microsoft RPC има и Sun RPC;
*RPC е технологията на която се основава DCOM;
*концептуално RPC е технология на C и C++;
* използването на RPC в други езикови среди (Perl, Jscript, Visual Basic и др.) се свежда към COM и DCOM;

* създаване на RPC приложение означава:
- клиентски код;
- сървърен код;
- interface definition language file (.IDL);
- application configuuration file (.ACF).



OSF стандарта дефинира езика за описание на интерфейси (Network Interface Definition Language - IDL), който език Microsoft реализира като MIDL. Езикът позволява дефиниране на интерфейси между клиента и сървъра. Всяка комуникация между клиента и сървъра преминава през MIDL интерфейс.

Използването на единен интерфейс позволява група разработчици да работят над клиентското приложение, докато друга група разработва сървърното прил. MIDL е транслатор, преобразуващ IDL кода в C. Компилацията изгражда шлюз, специфичен за клиента и сървъра.

MIDL позволява да се реализира интерфейс не само между различни

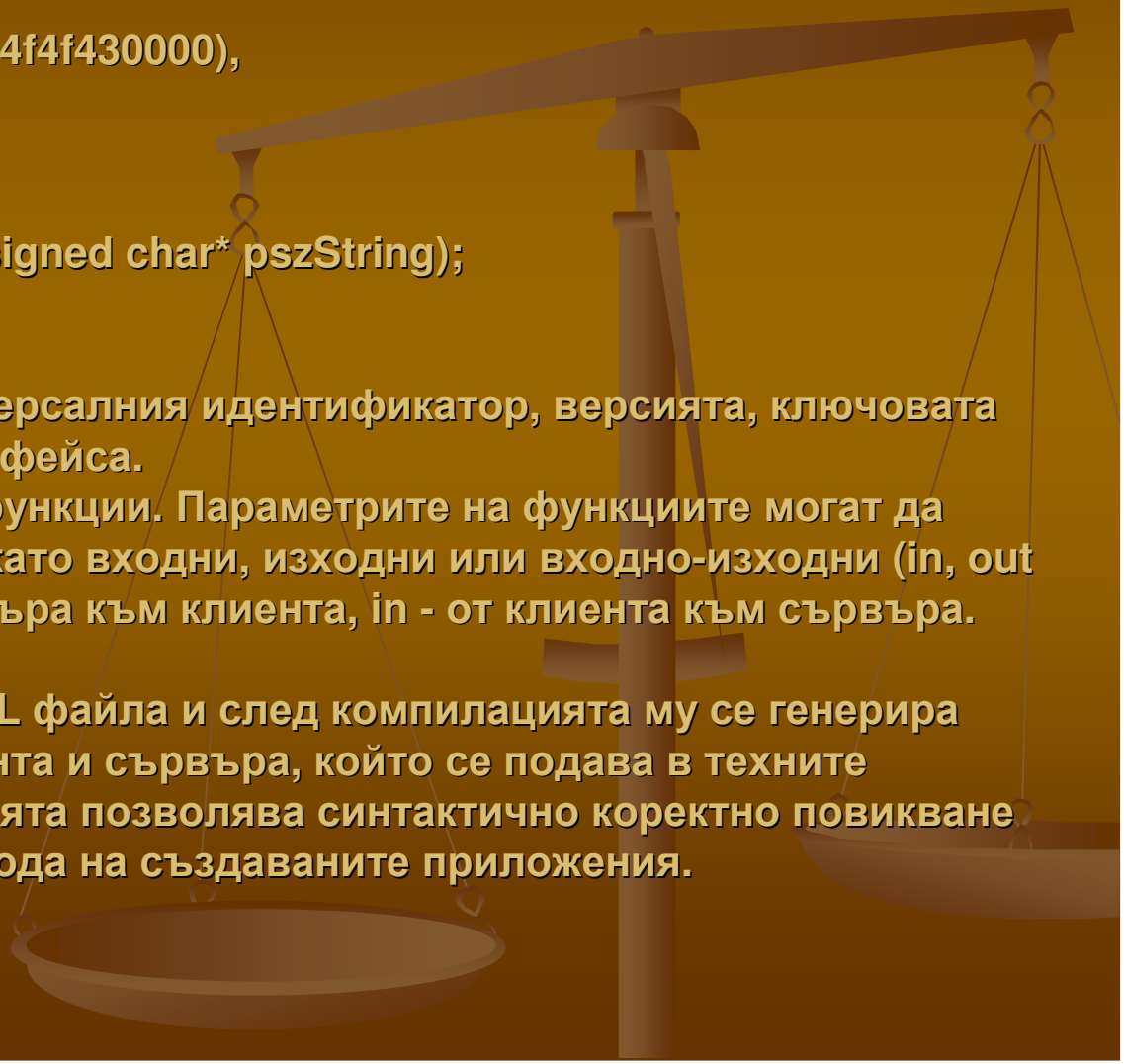
прил, но и между различни компютърни архитектури

За да бъдат изградени правилно шлюзовете в клиента и сървъра MIDL се нуждае от дефиниционен файл (Interface Definition File) - съдържа името, версията, локалния идентификатор на интерфейса и списък на процедурите, изграждащи конкретния интерфейс:

```
[uuid (ea981110-as3f-11df-9aa4-d4f4f430000),
 version (1.0)]
interface sample
{
void SampleProc([out,string] unsigned char* pszString);
}
```

Заглавната част съдържа универсалния идентификатор, версията, ключовата дума interface и името на интерфейса.
Тялото съдържа прототип на функции. Параметрите на функциите могат да бъдат допълнително описани като входни, изходни или входно-изходни (in, out или in-out)- out посока от сървъра към клиента, in - от клиента към сървъра.

На база на информацията от IDL файла и след компилацията му се генерира шлюз в синтаксис на C за клиента и сървъра, който се подава в техните заглавни файлове. Информацията позволява синтактично коректно повикване на отдалечената функция в C кода на създаваните приложения.



Стандартът DCE изисква наличие и на Application Configuration File (ACF), който съдържа RPC данни, които няма да бъдат прехвърлени през мрежата. Основната задача е изработването на т.нар. **binding хендъли** (хендъли, използвани при връзка с външно описани обекти).

Привързването в RPC е термин, използван по подобие на известния в класическото програмиране термин свързване (linking). Проблемът при реализиране на свързване в RPC механизма е, че не се знае точния адрес в паметта на процедурата, всъщност тя е разположена на сървъра. По тази причина на никакъв етап не е възможно попълване с конкретна за повикването информация. Очевидно свързване в класическото разбиране (linking) е невъзможно.

Клиентското повикване в Runtime режим на функции изисква изработване на **фиктивен хендъл за привързване** към тях, който ще се използва през цялото време докато се получи резултата.

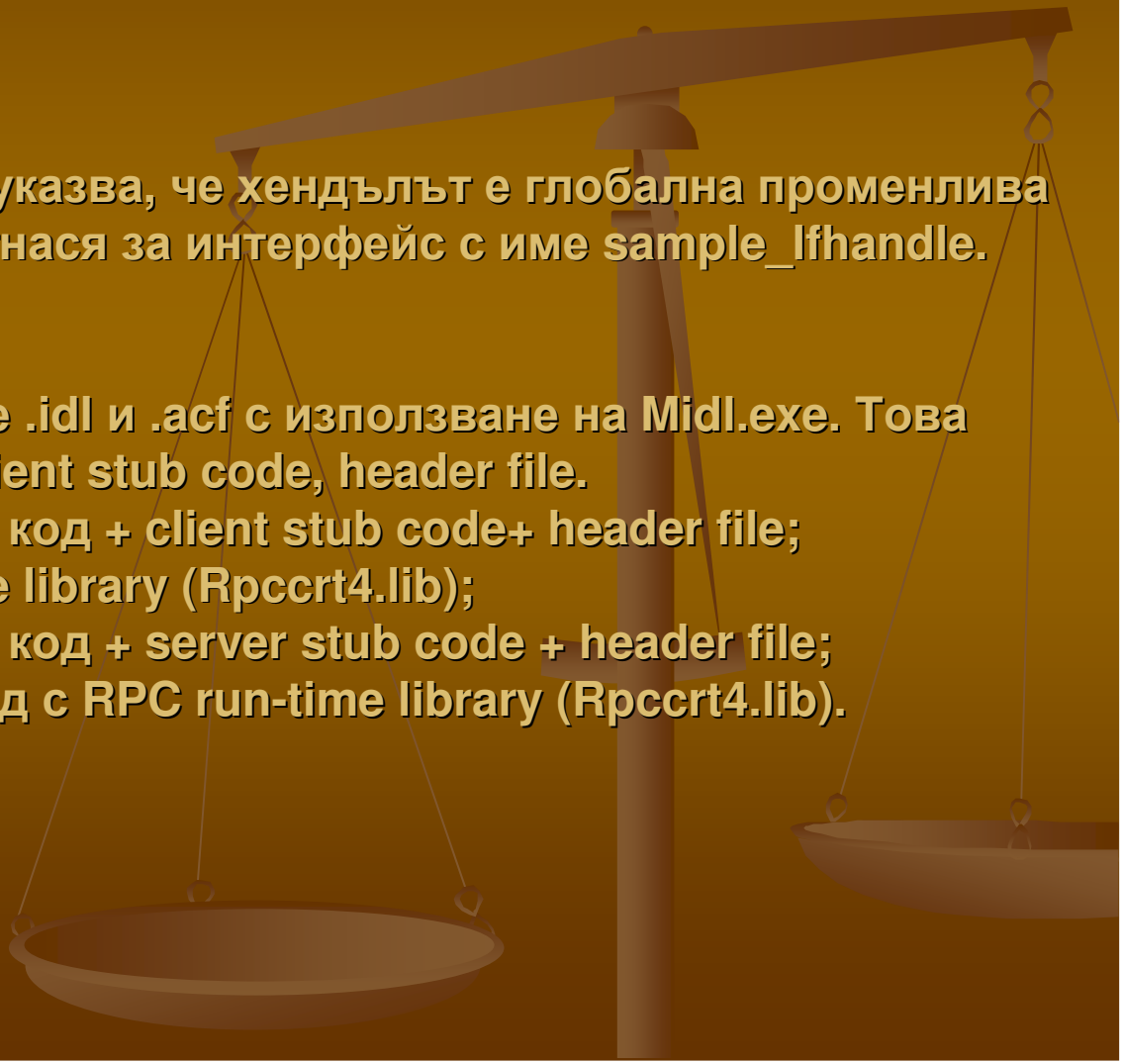
Всъщност хендълът за привързване не е параметър в прототип на функцията и той не е нужно да се предава по мрежата. Неговата роля е да опише глобално името и типа на определен параметър, чиято стойност ще се намери в друг компютър. Ето пример:

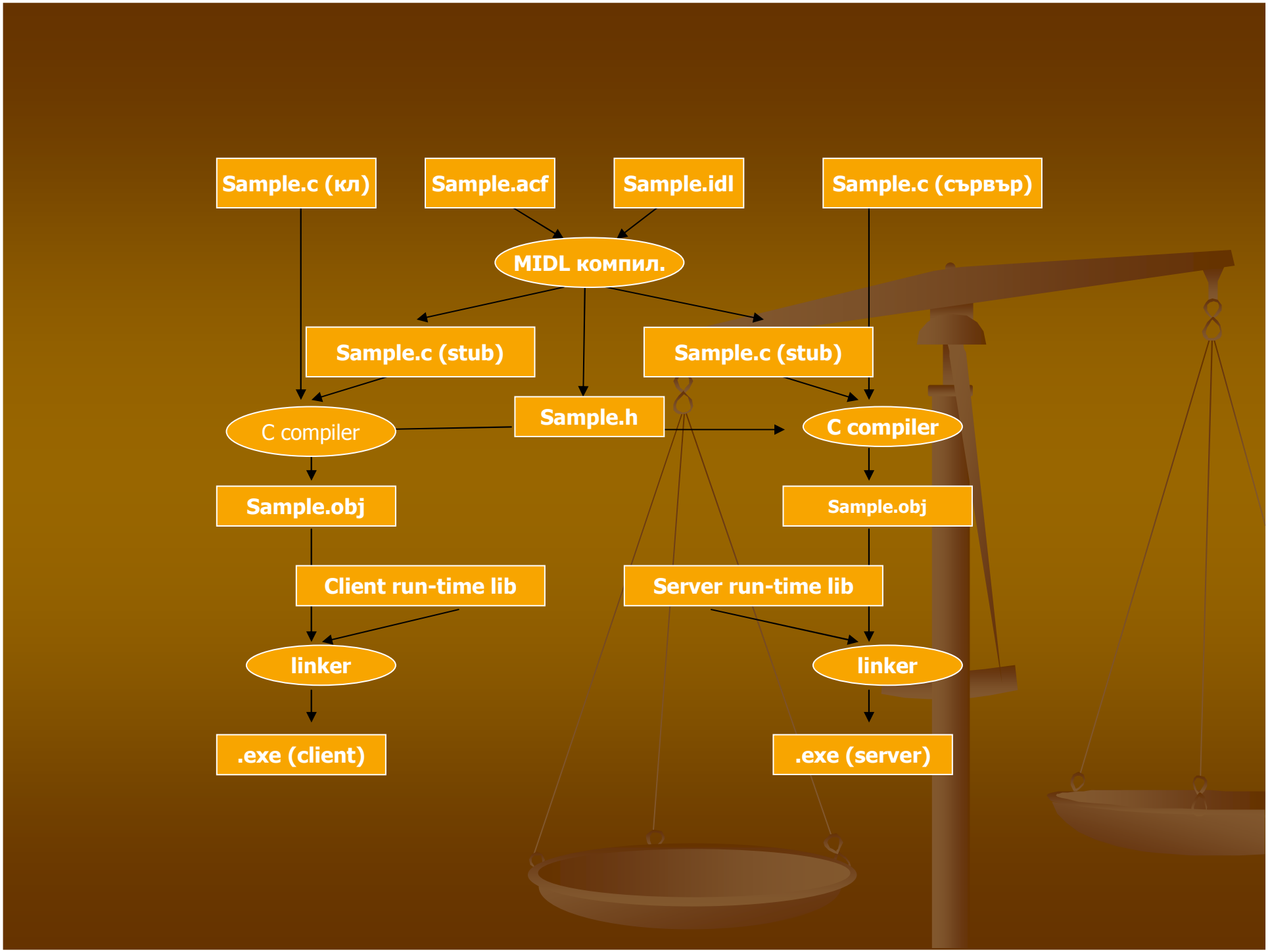
```
/*sample.acf*/
[implicit_handle (handle_t sample_lfhandle)]
interface sample
{}
```

- Атрибутът implicit_handle указва, че хендълът е глобална променлива и в конкретния случай се отнася за интерфейс с име sample_lfhandle.

Компилация:

1. компилират се файловете .idl и .acf с използване на Midl.exe. Това създава server stub code, client stub code, header file.
2. компилира се клиентския код + client stub code+ header file;
3. свързва се и RPC run-time library (Rpccrt4.lib);
4. компилира се сървърния код + server stub code + header file;
5. свързва се сървърния код с RPC run-time library (Rpccrt4.lib).

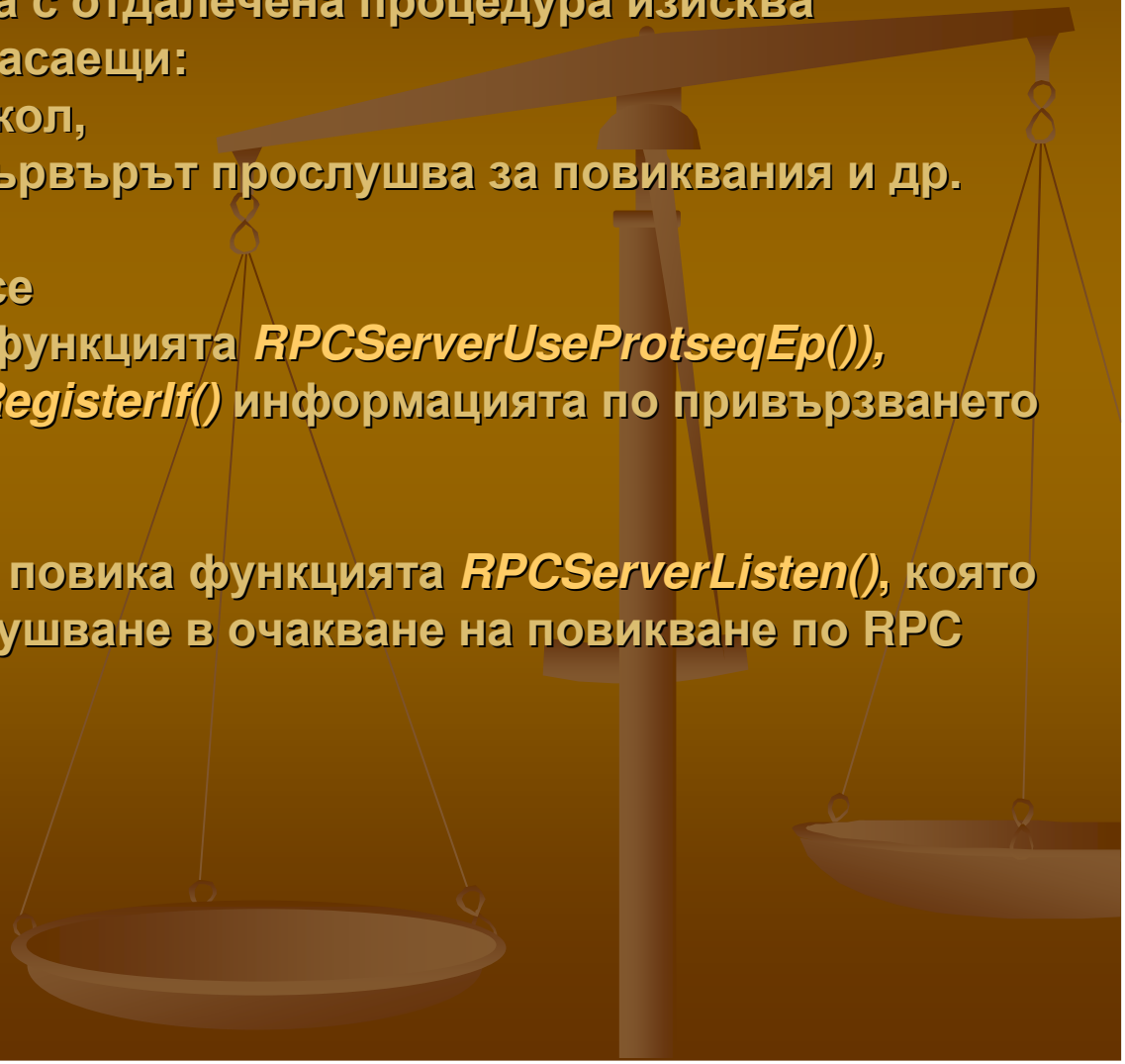



Sample.c (кл)
Sample.acf
Sample.idl
Sample.c (сървър)
MIDL компил.
Sample.c (stub)
Sample.c (stub)
C compiler
Sample.h
C compiler
Sample.obj
Sample.obj
Client run-time lib
Server run-time lib
linker
linker
.exe (client)
.exe (server)

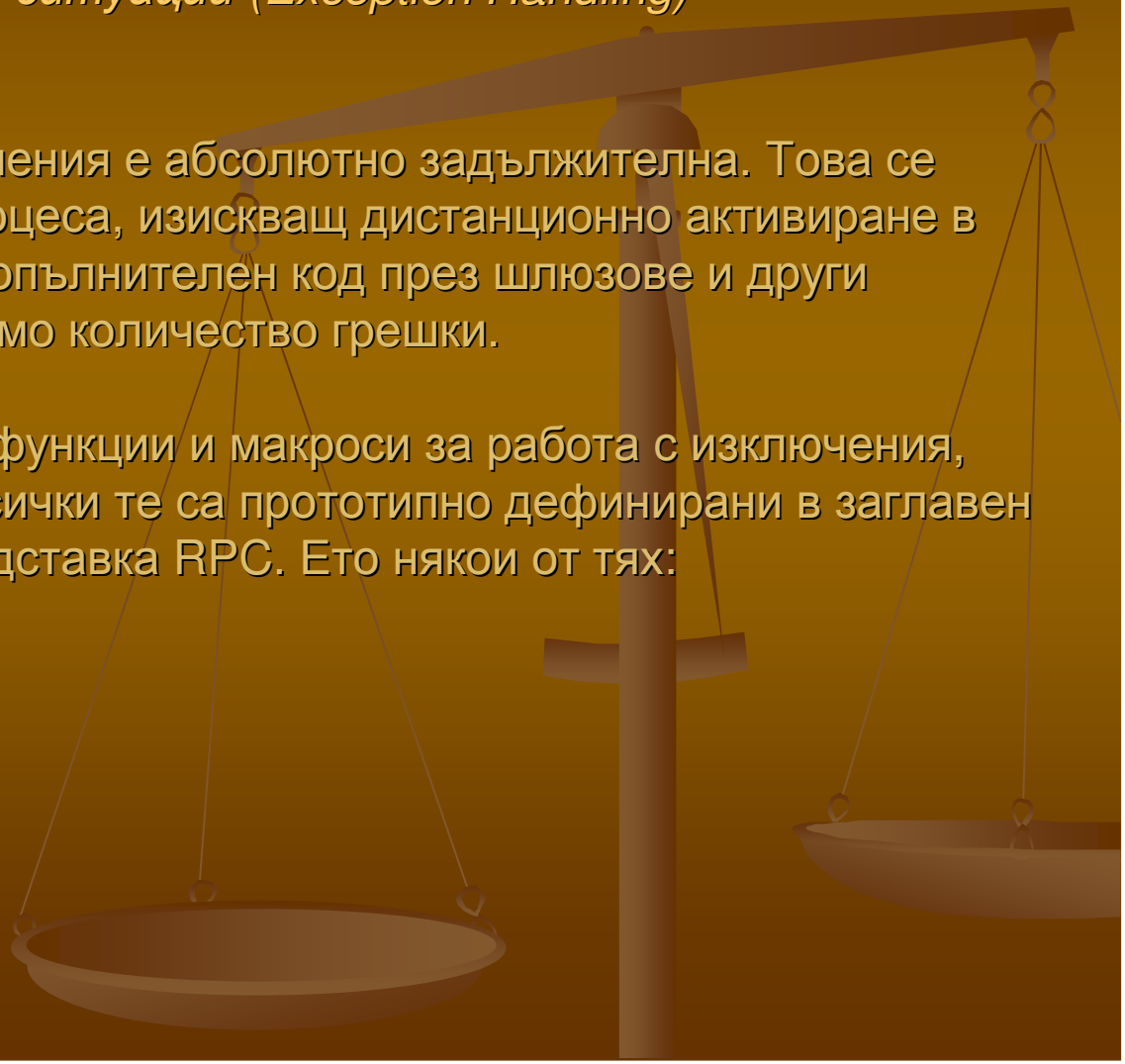
Осъществяването на връзка с отдалечена процедура изисква предварителни настройки касаещи:
- използван мрежови протокол,
- мрежови адрес на който сървърът прослушва за повиквания и др.

След като тези параметри се регистрират в системата (функцията *RPCServerUseProtseqEp()*), с повикване на *RPCServerRegisterIf()* информацията по привързването става достъпна за клиента.

В края сървърът следва да повика функцията *RPCServerListen()*, която го вкарва в режим на прослушване в очакване на повикване по RPC протокол.



Обработка на изключителни ситуации (Exception Handling)



В RPC обработката на изключения е абсолютно задължителна. Това се налага от спецификата на процеса, изискващ дистанционно активиране в друга среда, активиране на допълнителен код през шлюзове и други процеси, съпътствани от голямо количество грешки.

Microsoft дефинира набор от функции и макроси за работа с изключения, съпътстващи RPC обмена. Всички те са прототипно дефинирани в заглавен файл RPC.h и започват с представка RPC. Ето някои от тях:
RpcTryExcept;
RpcEndExcept;
RrcExcept();

Приложение, включващо клиент и сървър и използване на RPC повиквания

Представен е код на пример с вградени две приложения.

Клиентското приложение реализира поръчки на пица;

сървърното изпълнява всички описани до тук служебни действия по инициализиране и коректно изпълнение на RPC, а именно активира сървър, поставя го в прослушващ режим и с помощта на интерфейсни описания в IDL и ACF файлове експонира определена сървърна функционалност.

Първата стъпка е създаване на клиентското приложение. Ето кратък код на клиентска програма, написана на C, която изпълнява повиквания на две дистанционни (RPC) процедури:

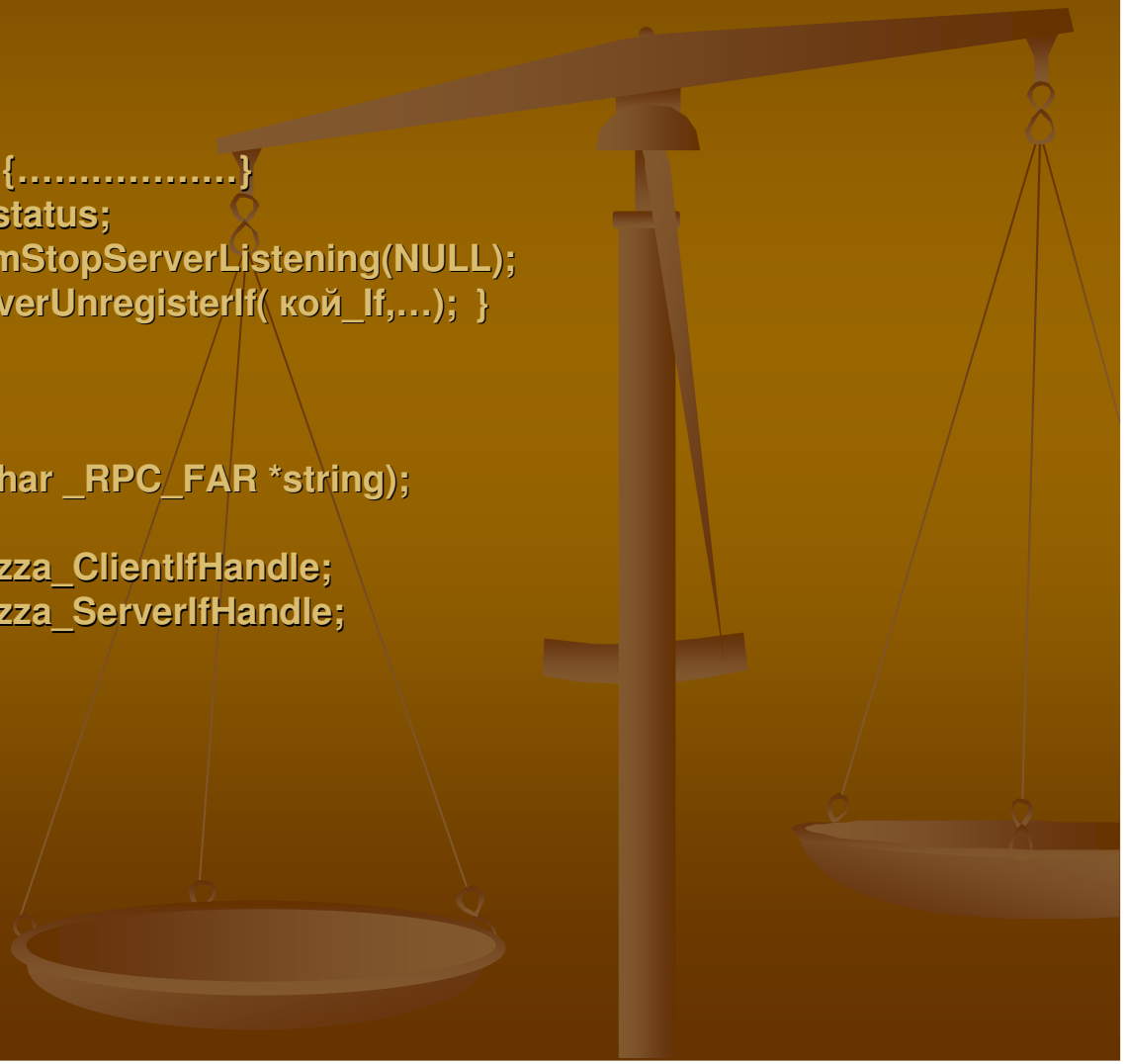
```
void _CRTAPI1 main(void)
{       RpcTryExcept
                pizzaRpc(“ искате да поръчате пица?”);
                Shutdown();
        RpcExcept(1)
                printf(“ има грешка при RPC повикването. \n”);
        RpcEndExcept
}
```

-Описваме двете ф-ии :

```
#include <RPC.H>
include <pizza.h>
…
void pizzaRPC(unsigned char *string) {……..……}
void Shutdown(void) { RPC_STATUS status;
                      status = RpcMgmStopServerListening(NULL);
                      status = RpcServerUnregisterIf( кой_If,…);  }
```

- pizza.h се създава от MIDL:

```
….
void pizzaRPC( unsigned char _RPC_FAR *string);
void ShutDown(void);
extern RPC_IF_HANDLE pizza_ClientIfHandle;
extern RPC_IF_HANDLE pizza_ServerIfHandle;
..
```



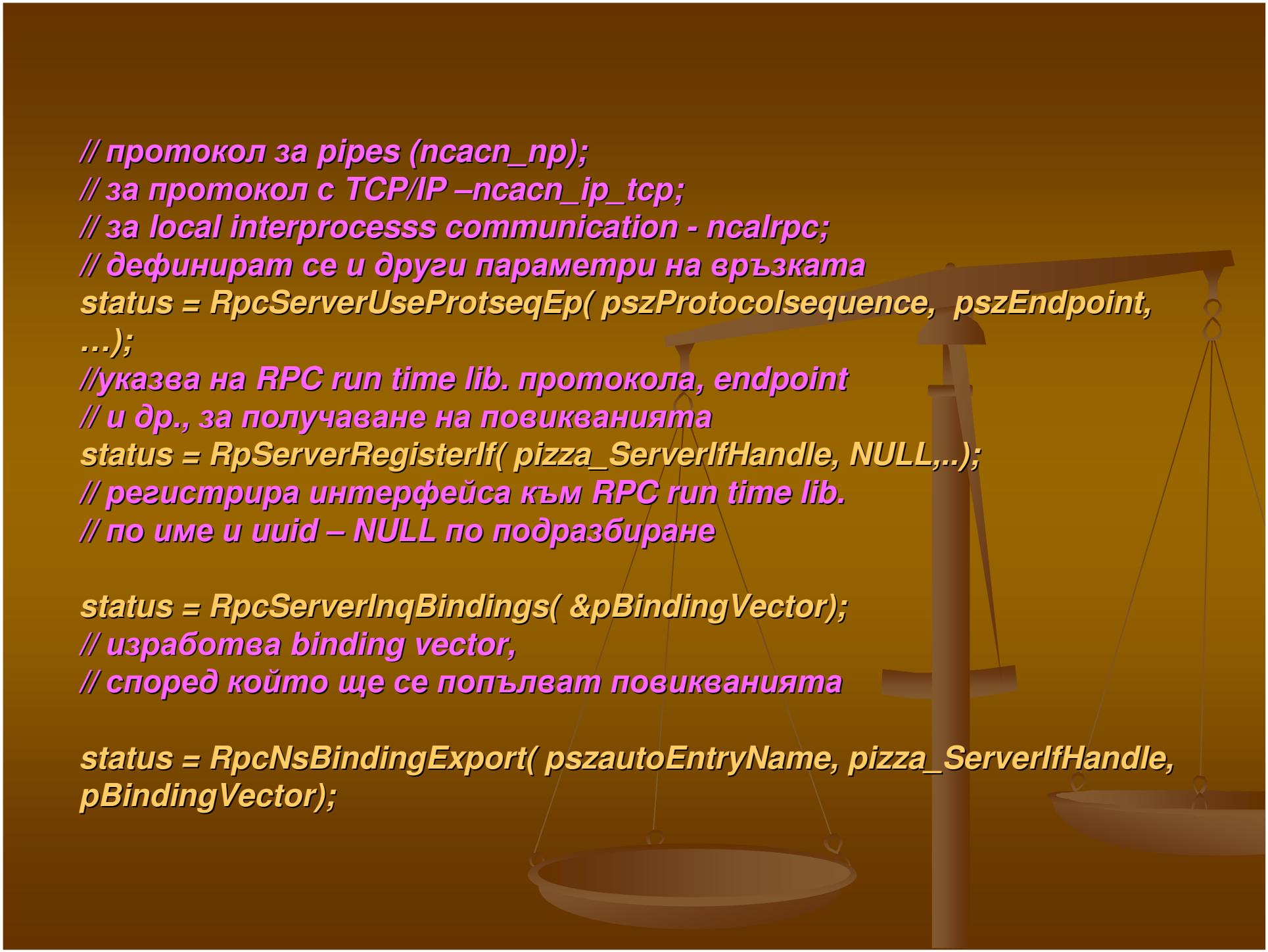
Сървърното приложение е сложната част: установява се вектор на привързване (binding vector), регистрира се хендъл и се експортира информация, необходима за привързването.
Сървърът застава в режим на изчакване на RPC повиквания.

Следва код на сървърната част за приложение Pizza:

```
#include <STDLIB.H>
#include <RPC.H>
#include “pizza.h”

void  main( int argc, char* argv[])
{RPC_STATUS status ;        // тип 32 бита, status code при RPC повиквания
RPC_BINDING_VECTOR * bindingVector;
 /* масив от binding handles в сървъра. Всеки от тях указва откъде
сървърът може да получи RPC повикване */

unsigned char *pszAutoEntryName =
“/.:/Autohandle_sample”;
// името указва къде се експортират binding handles
// и обекти – еднакво в кл. и сърв.
unsigned char *pszEndpint = \\pipe\\auto;
unsigned char *pszProtocolsequence = “ncacn_np”;
```



```
// протокол за pipes (ncacn_np);
// за протокол с TCP/IP –ncacn_ip_tcp;
// за local interprocesss communication - ncalrpc;
// дефинират се и други параметри на връзката
status = RpcServerUseProtseqEp( pszProtocolsequence,  pszEndpoint,
...);
//указва на RPC run time lib. протокола, endpoint
// и др., за получаване на повикванията
status = RpServerRegisterIf( pizza_ServerIfHandle, NULL,..);
// регистрира интерфейса към RPC run time lib.
// по име и uuid – NULL по подразбиране

status = RpcServerInqBindings( &pBindingVector);
// изработва binding vector,
// според който ще се попълват повикванията

status = RpcNsBindingExport( pszautoEntryName, pizza_ServerIfHandle,
pBindingVector);
```

```
// експортира вектора и интерфейса
status = RpcServerListen(  /*указания колко повиквания и колко време се чакат */);

}
```

Остават още- IDL и ACF файловете. За проекта те са достатъчно кратки и прости. В тях е описана експонираната от сървъра функционалност, а именно две процедури с възможност за дистанционни повикване.

pizza.idl

```
[ uuid (.....), version (1.0), pointer_default(unique)]
interface pizza
{
void pizzaRPC([in, string] unsigned char *string);
void ShutDown(void);
}
```

pizza.acf

```
[ auto_handle ]
interface pizza
{
}
```