



RPC - Remote Procedure Call

Teoria e prática

Me. Daniel Sucupira Lima



Programa de Pós-Graduação
em Ciência da Computação




Agenda

1. Introdução;
2. Fundamentação teórica;
3. Instalação;
4. Exemplos práticos:
 - a. Exemplo 1: Exibe 1;
 - b. Exemplo 2: Exibe 2;
 - c. Exemplo 3: Pergunta;
 - d. Exemplo 4: Quadrado;
 - e. Exemplo 5: Soma;
 - f. Exemplo 6: Calculadora;
 - g. Exemplo 7: Operação.
5. Conclusão;
6. Referências.

1. Introdução



- Uma das práticas mais comuns em programas é a criação de funções;
- Elas têm como características:
 - Modularização de código;
 - Evitar duplicação de código;
 - Com alterações em um único local pode-se resolver problemas em diversas partes do código;
 - Dentre outros.
- O comportamento dessa função é completamente definido no programa principal ou bibliotecas.



Programa principal com uma outra função local

A função a ser chamada é
completamente definida.

C prog.c

```
prog.c > local_function()
1  #include <stdio.h>
2
3  void local_function()
4  {
5      printf("Executing local code\n");
6  }
7
8  int main()
9  {
10     local_function();
11
12     return 0;
13 }
```

PROBLEMAS

SAÍDA

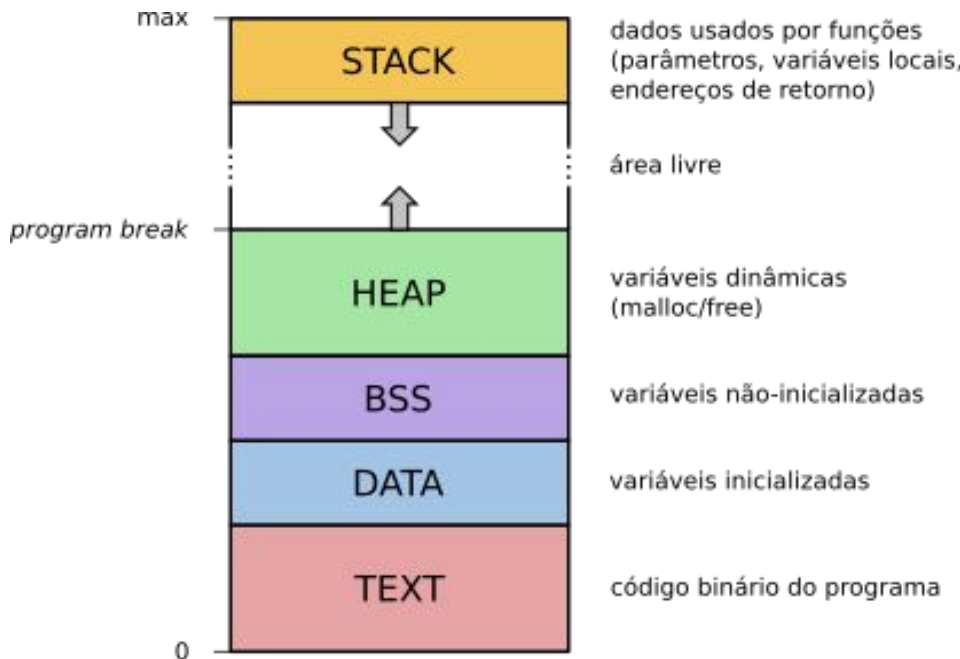
CONSOLE DE DEPURAÇÃO

TERMINAL

```
daniel@daniel-Aspire-A515-55:~/Downloads/Demo$ g++ prog.c -o programa
daniel@daniel-Aspire-A515-55:~/Downloads/Demo$ ./programa
Executing local code
daniel@daniel-Aspire-A515-55:~/Downloads/Demo$
```

Cada programa, na memória principal, tem regiões

As funções locais ficam na seção 'text'



Limites e endereços

- Funções têm endereços;
- Blocos têm tamanhos.

```
C prog.c ×
C prog.c > main()
1  #include <stdio.h>
2
3  void local_function() { }
4
5  int main()
6  {
7      printf("%p, %p, %p, %p\n", main, printf, scanf, local_function);
8      return 0;
9  }
10
```

PROBLEMAS SAÍDA CONSOLE DE DEPURAÇÃO TERMINAL

```
daniel@daniel-Aspire-A515-55:~/Downloads/Demo$ g++ prog.c -o programa
daniel@daniel-Aspire-A515-55:~/Downloads/Demo$ ./programa
0x55a35dfca144, 0x7f5cc2460770, 0x7f5cc2462110, 0x55a35dfca139
daniel@daniel-Aspire-A515-55:~/Downloads/Demo$ size programa
   text    data     bss     dec     hex filename
   1520     560        8    2088     828 programa
daniel@daniel-Aspire-A515-55:~/Downloads/Demo$
```

Comunicações em rede



- Em alguns casos, o código a ser chamado não está na máquina local;
- Exemplos:
 - Obter um status de um serviço;
 - Ativar determinado recurso;
 - Executar um cálculo e obter o resultado;
 - Executar uma transformação sobre dados locais (após upload) e baixar os resultados (com download).

Comunicações em rede



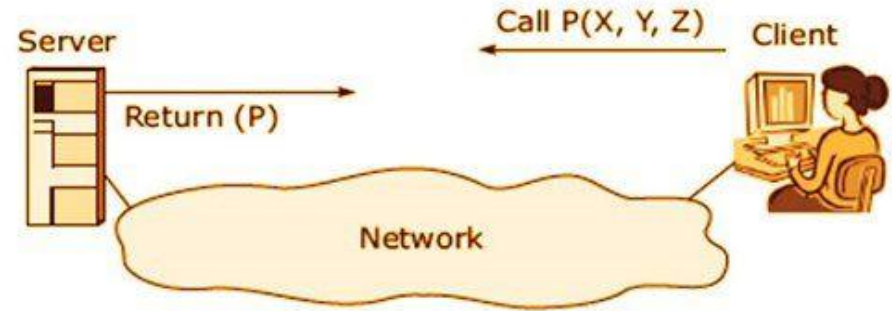
- Pode-se usar, por exemplo, o modelo cliente/servidor;
- Deve-se criar um protocolo para essa comunicação;
- Deve-se fazer sockets para as trocas de mensagens;
- Deve-se empacotar/desempacotar parâmetros/resultados;
- Dentre outros.

Seria interessante houvesse uma forma de simplificar essa comunicação entre clientes e servidores, facilitando as chamadas a métodos em outras máquinas.

Existe uma forma
de facilitar essa
comunicação

Pode-se utilizar o RPC

Remote Procedure Call (RPC)



"A chave por trás do sucesso do RPC é a semântica simples, mas poderosa, de seu modelo de programação."

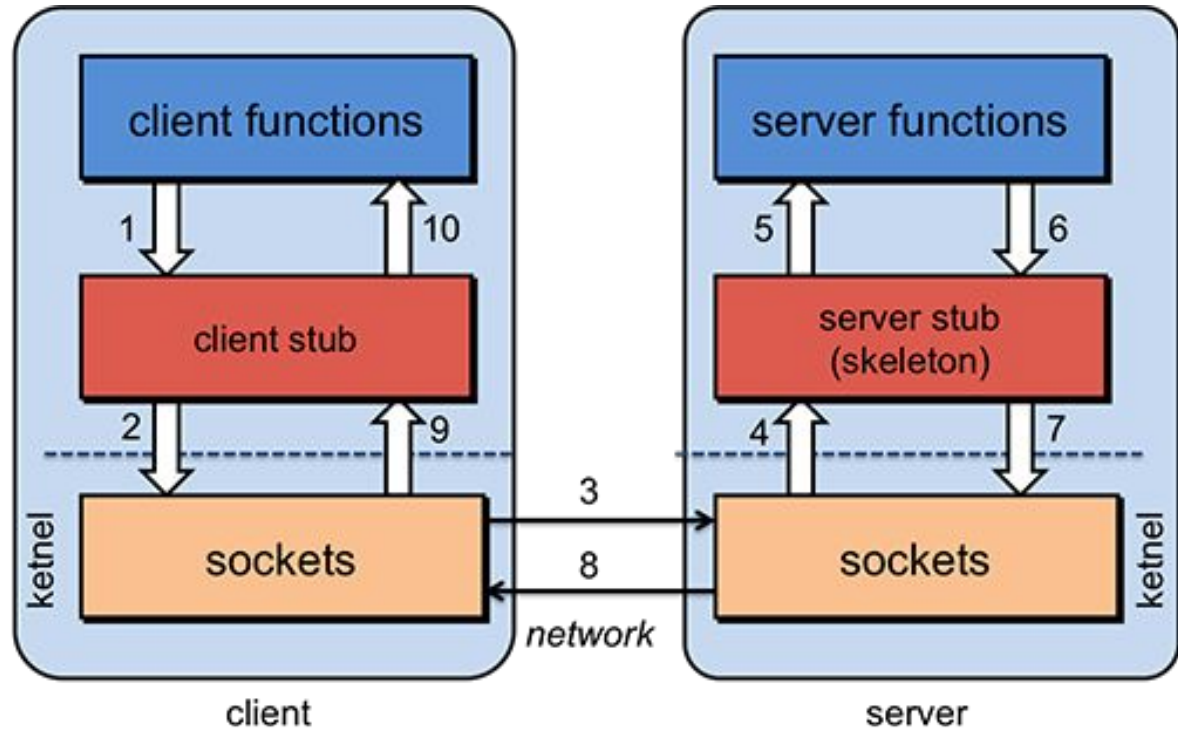
- Autores: Stephanie Wang, Benjamin Hindman e Ion Stoica

2. Fundamentação teórica



- O RPC permite a chamada à procedimentos remotos;
- Ele cria uma interface simples e elegante para essa chamada;
- Essa interface é feita através dos **stubs**;
- Existe um stub para o cliente e outro para o servidor.

Visualização dos stubs

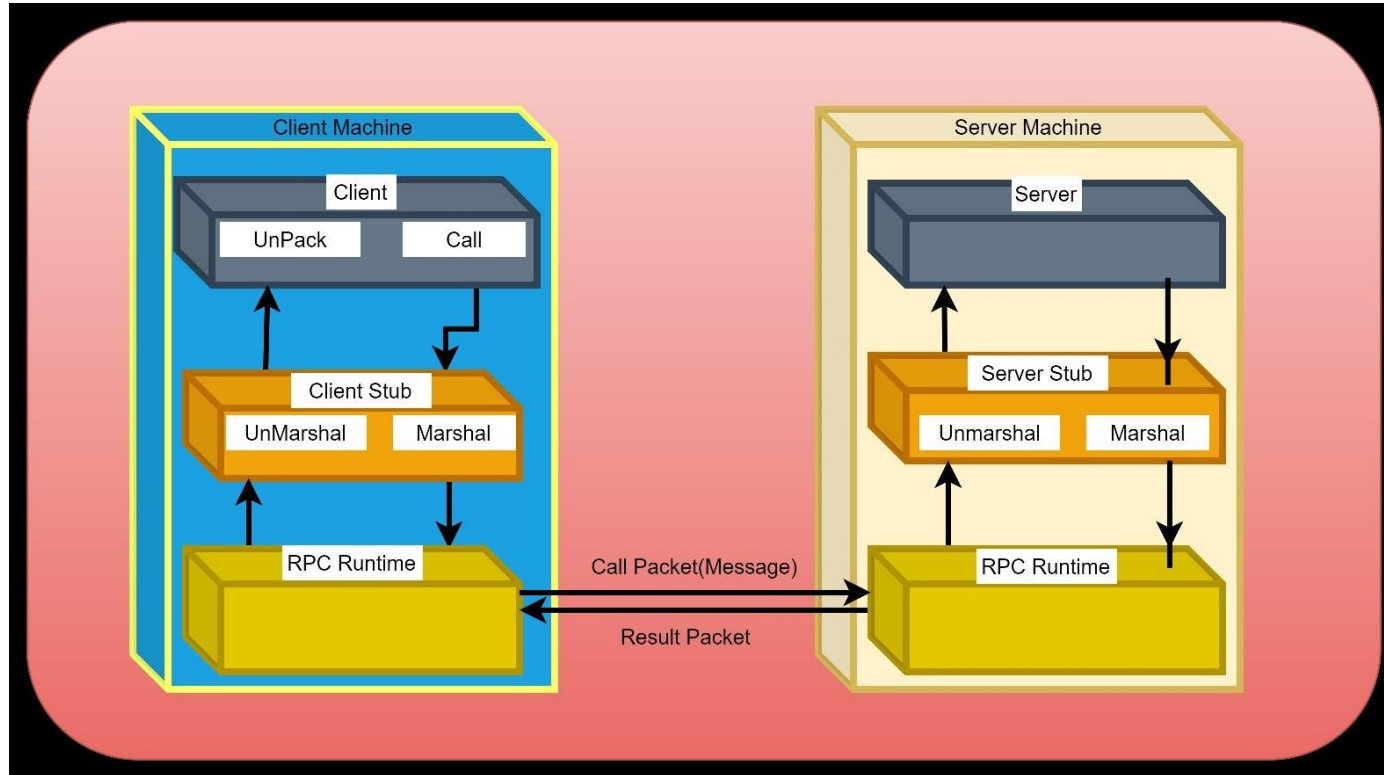


2. Fundamentação teórica



- Observe que os stubs recebem os dados da camada superior como parâmetros e monta uma mensagem a ser enviada como cadeia de bytes;
- Esse processo de empacotamento é chamado de Marshalling ou packaging;
- Observe ainda que os stubs devem entregar dados para a camada superior como parâmetros, desmontando uma mensagem que veio como cadeia de bytes;
- Esse processo de desempacotamento é chamado de unmarshalling ou unpacking.

Empacotamento e desempacotamento



3. Instalação



- Exibe-se neste trabalho como instalar duas versões do RPC a serem usadas pela linguagem de programação C;
- Dependendo do sistema operacional e da sua versão os procedimentos mudam;
- Características da máquina:
 - Sistema operacional: Ubuntu;
 - Versão: Ubuntu 22.04.2 LTS - Jammy Jellyfish;
 - Tipo: 64 bits.

3. Instalação



- Passo 1:
 - As novas versões de ubuntu não tem, por padrão, as ferramentas essenciais de compilação de programas C/C++;
 - Caso sua máquina não tenha, instale-as.

```
daniel@daniel-Aspire-A515-55:~$ sudo apt install build-essential
Lendo listas de pacotes... Pronto
Construindo árvore de dependências... Pronto
Lendo informação de estado... Pronto
Os seguintes pacotes foram instalados automaticamente e já não são necessários:
  libntirpc3.5 liburcu8
Utilize 'sudo apt autoremove' para os remover.
```

3. Instalação



- Passo 2:
 - Verifique se sua máquina já tem as ferramentas de RPC.

```
daniel@daniel-Aspire-A515-55:~$ rpcinfo
Comando 'rpcinfo' não encontrado, mas poder ser instalado com:
sudo apt install rpcbind
daniel@daniel-Aspire-A515-55:~$
```

3. Instalação



- Passo 2:
 - Como minha máquina ainda não tinha, eu instalei.

```
daniel@daniel-Aspire-A515-55:~$ sudo apt install rpcbind
Lendo listas de pacotes... Pronto
Construindo árvore de dependências... Pronto
Lendo informação de estado... Pronto
Os seguintes pacotes foram instalados automaticamente e já não são necessários:
  libntirpc3.5 liburcu8
```

3. Instalação



- Passo 2:
 - Verifique a instalação.

```
daniel@daniel-Aspire-A515-55:~$ rpcinfo
  program version netid  address          service  owner
  100000    4    tcp6   :::0.111         portmapper superuser
  100000    3    tcp6   :::0.111         portmapper superuser
  100000    4    udp6   :::0.111         portmapper superuser
  100000    3    udp6   :::0.111         portmapper superuser
  100000    4    tcp    0.0.0.0.0.111   portmapper superuser
  100000    3    tcp    0.0.0.0.0.111   portmapper superuser
  100000    2    tcp    0.0.0.0.0.111   portmapper superuser
  100000    4    udp    0.0.0.0.0.111   portmapper superuser
  100000    3    udp    0.0.0.0.0.111   portmapper superuser
  100000    2    udp    0.0.0.0.0.111   portmapper superuser
  100000    4    local  /run/rpcbind.sock portmapper superuser
  100000    3    local  /run/rpcbind.sock portmapper superuser
daniel@daniel-Aspire-A515-55:~$
```

4. Exemplos práticos



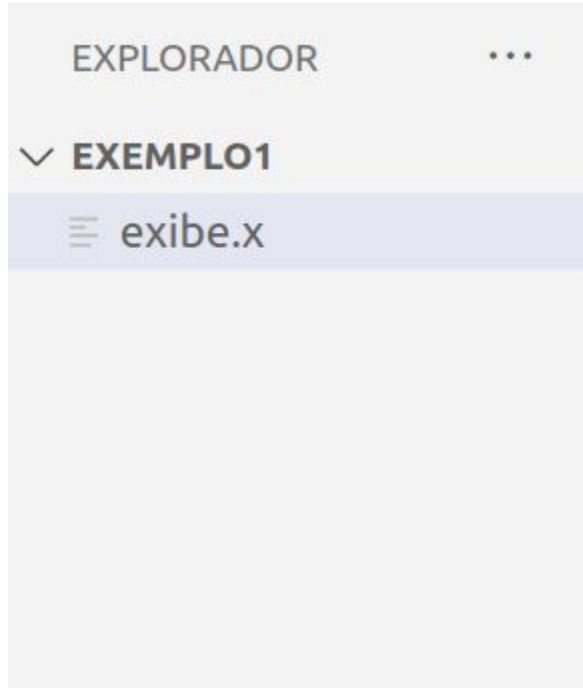
- Em cada um dos exemplos deve-se criar o arquivo que definirá:
 - Funções a serem chamadas;
 - Estruturas de dados a serem usadas;
 - Versões das definições;
 - Esse arquivo tem a extensão .x.

4. Exemplos práticos: exemplo 1



- Demonstro a interface `exibe.x`, do exemplo 1;
- Crio uma pasta em `~/Downloads/rpc/exemplo1`;
- Crio um único arquivo, que é o `exibe.x`;
- Deve-se gerar os stubs de cliente e servidor;
- Para gerar os stubs deve-se executar o comando: `rpcgen -a -C exibex.X`:
 - Flag `a`: cria todos os arquivos de cliente e servidor;
 - Flag `C`: cria códigos no modo `ansi c`.

4. Exemplos práticos: exemplo 1



≡ *exibe.x* ×

≡ *exibe.x*

```
1
2  program EXIBE_PROG{
3      version EXIBE_VERS{
4          void exibe()=1;
5      }=1;
6  }=0x23451111;
7
```

4. Exemplos práticos: exemplo 1



```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$ ls
exibe.x
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$ rpcgen -a -C exibex
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$ ls
exibe_client.c  exhibe_clnt.c  exhibe.h  exhibe_server.c  exhibe_svc.c  exhibe.x  Makefile.exibe
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$
```


4. Exemplos práticos: exemplo 1



- O código gerado já está pronto para uso;
- Foram criados códigos de cliente e servidor;
- Define-se funções que chamam os códigos de stub;
- Tudo isso já está pronto, até exibindo em quais locais personalizar a funcionalidade.

4. Exemplos práticos: exemplo 1



- Também foi fornecido um arquivo Makefile;
- Dependendo do seu sistema operacional, já está operacional a utilização da biblioteca de rpc;
- Nas versões de Ubuntu como 14.04 o Makefile já estava finalizado;
- Nessa versão do 22.04 não está pronto ainda;
- O próximo slide ilustra isso.

4. Exemplos práticos: exemplo 1



```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$ make -f Makefile.exibe
cc -g -c -o exhibe_clnt.o exhibe_clnt.c
In file included from exhibe_clnt.c:7:
exibe.h:9:10: fatal error: rpc/rpc.h: Arquivo ou diretório inexistente
   9 | #include <rpc/rpc.h>
     |           ^~~~~~
compilation terminated.
make: *** [<embutido>: exhibe_clnt.o] Erro 1
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$
```

4. Exemplos práticos: exemplo 1



- Esse erro ocorreu pois não se sabe em qual pasta estão os arquivos de cabeçalho e ainda não se está indicando quais arquivos de implementação linkar;
- Resolvi procurar em qual pasta estão esses conteúdos;
- O slide seguinte exhibe o resultado.

4. Exemplos práticos: exemplo 1



```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$ find /usr/include -type f -iname '*rpc*'
/usr/include/linux/rxrpc.h
/usr/include/misc/fastrpc.h
/usr/include/tirpc/rpc/rpc_msg.h
/usr/include/tirpc/rpc/rpcb_prot.x
/usr/include/tirpc/rpc/rpc_com.h
/usr/include/tirpc/rpc/rpc.h
/usr/include/tirpc/rpc/rpcsec_gss.h
/usr/include/tirpc/rpc/rpcb_prot.h
/usr/include/tirpc/rpc/rpcb_clnt.h
/usr/include/tirpc/rpc/rpcent.h
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$
```

4. Exemplos práticos: exemplo 1



- A imagem anterior mostrou que o comando `rpcbind` instalou a versão do `rpc` chamada TIRPC;
- Pode-se instalar outras versões, como o NTIRPC:

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$ sudo apt install libntirpc-dev
[sudo] senha para daniel:
Lendo listas de pacotes... Pronto
Construindo árvore de dependências... Pronto
Lendo informação de estado... Pronto
Os NOVOS pacotes a seguir serão instalados:
 libntirpc-dev
```

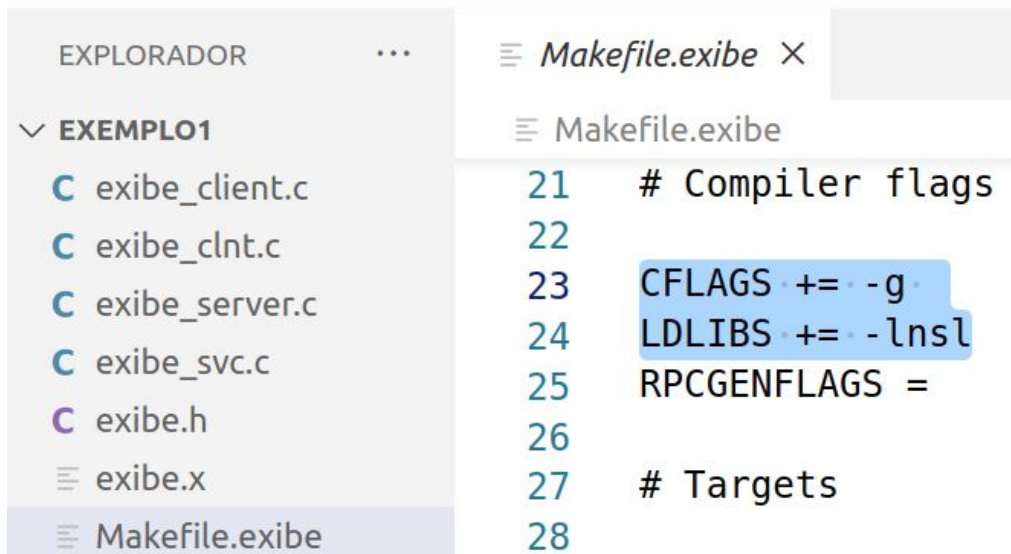
4. Exemplos práticos: exemplo 1

- Agora tem-se as seguintes bibliotecas:

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$ find /usr/include -type f -iname '*rpc*'
/usr/include/linux/rxrpc.h
/usr/include/misc/fastrpc.h
/usr/include/ntirpc/ltnng/rpcping.h
/usr/include/ntirpc/rpc/rpc_msg.h
/usr/include/ntirpc/rpc/rpcb_prot.x
/usr/include/ntirpc/rpc/rpc_com.h
/usr/include/ntirpc/rpc/tirpc_compat.h
/usr/include/ntirpc/rpc/rpc_err.h
/usr/include/ntirpc/rpc/rpc.h
/usr/include/ntirpc/rpc/rpcb_prot.h
/usr/include/ntirpc/rpc/rpcb_clnt.h
/usr/include/ntirpc/rpc/rpc_cksum.h
/usr/include/ntirpc/rpc/rpcent.h
/usr/include/tirpc/rpc/rpc_msg.h
/usr/include/tirpc/rpc/rpcb_prot.x
/usr/include/tirpc/rpc/rpc_com.h
/usr/include/tirpc/rpc/rpc.h
/usr/include/tirpc/rpc/rpcsec_gss.h
/usr/include/tirpc/rpc/rpcb_prot.h
/usr/include/tirpc/rpc/rpcb_clnt.h
/usr/include/tirpc/rpc/rpcent.h
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$
```

4. Exemplos práticos: exemplo 1

- Deve-se indicar estas pastas para o Makefile;
- A versão padrão está da seguinte forma:



The screenshot shows a code editor with two panels. The left panel is a file explorer titled 'EXPLORADOR' with a menu icon and three dots. It shows a directory structure under 'EXEMPLO1' containing files: 'exibe_client.c', 'exibe_clnt.c', 'exibe_server.c', 'exibe_svc.c', 'exibe.h', 'exibe.x', and 'Makefile.exibe'. The right panel shows the content of 'Makefile.exibe' with line numbers 21 through 28. The content includes comments for compiler flags and targets, with specific flag settings highlighted in blue.

```
21  # Compiler flags
22
23  CFLAGS += -g
24  LDLIBS += -lnsl
25  RPCGENFLAGS =
26
27  # Targets
28
```

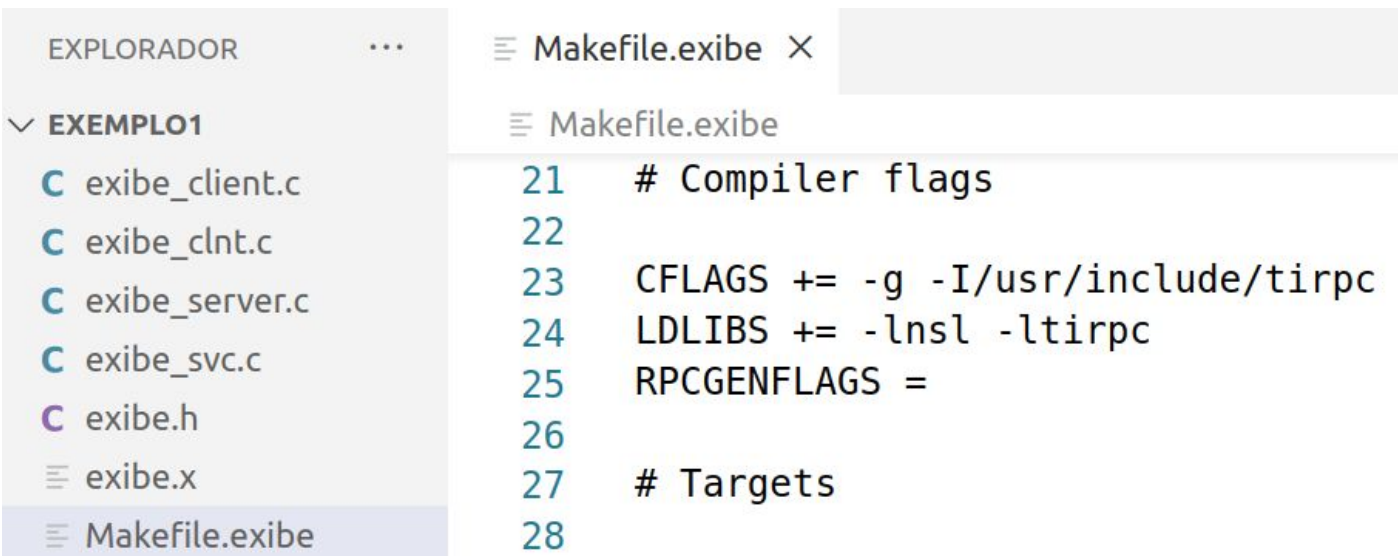

4. Exemplos práticos: exemplo 1



- Para indicar o tirpc:
 - Cabeçalhos: `/usr/include/tirpc/`;
 - Linkar: `-l tirpc`.
- Para indicar ntirpc:
 - Cabeçalhos: `/usr/include/ntirpc/`;
 - Linkar: `-l ntirpc`.

4. Exemplos práticos: exemplo 1

- Escolhi a biblioteca tirpc, por ser a instalação padrão que é obtida com o rpcbind.



The screenshot shows a code editor interface. On the left is a file explorer with the title 'EXPLORADOR'. It contains a folder 'EXEMPLO1' which is expanded to show several files: 'exibe_client.c', 'exibe_clnt.c', 'exibe_server.c', 'exibe_svc.c', 'exibe.h', 'exibe.x', and 'Makefile.exibe'. The 'Makefile.exibe' file is selected and highlighted in blue. On the right side of the editor, the content of 'Makefile.exibe' is displayed. It starts with a tab icon and the filename 'Makefile.exibe' followed by a close icon. The file content includes comments for compiler flags and targets, with line numbers 21 through 28 visible on the left margin of the code area.

```
21  # Compiler flags
22
23  CFLAGS += -g -I/usr/include/tirpc
24  LDLIBS += -lnsl -ltirpc
25  RPCGENFLAGS =
26
27  # Targets
28
```

4. Exemplos práticos: exemplo 1



- Já é possível ter sucesso ao chamar o comando make.

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$ make -f Makefile.exibe
cc -g -I/usr/include/tirpc -c -o exhibe_clnt.o exhibe_clnt.c
cc -g -I/usr/include/tirpc -c -o exhibe_client.o exhibe_client.c
cc -g -I/usr/include/tirpc -o exhibe_client exhibe_clnt.o exhibe_client.o -lnsl -ltirpc
cc -g -I/usr/include/tirpc -c -o exhibe_svc.o exhibe_svc.c
cc -g -I/usr/include/tirpc -c -o exhibe_server.o exhibe_server.c
cc -g -I/usr/include/tirpc -o exhibe_server exhibe_svc.o exhibe_server.o -lnsl -ltirpc
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$
```

4. Exemplos práticos: exemplo 1



- A implementação padrão não vem com código;
- Executa-se primeiramente o servidor;
- Executa-se então o cliente.

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$ ls
exibe_client      exhibe_clnt.c  exhibe_server  exhibe_svc.c  Makefile.exibe
exibe_client.c    exhibe_clnt.o  exhibe_server.c  exhibe_svc.o
exibe_client.o    exhibe.h       exhibe_server.o  exhibe.x
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$ ./exibe_server
```

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$ ls
exibe_client      exhibe_clnt.c  exhibe_server  exhibe_svc.c  Makefile.exibe
exibe_client.c    exhibe_clnt.o  exhibe_server.c  exhibe_svc.o
exibe_client.o    exhibe.h       exhibe_server.o  exhibe.x
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$ ./exibe_client
usage: ./exibe_client server_host
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$ ./exibe_client localhost
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$
```

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$ ls
exibe_client      exhibe_clnt.c  exhibe_server  exhibe_svc.c  Makefile.exibe
exibe_client.c    exhibe_clnt.o  exhibe_server.c  exhibe_svc.o
exibe_client.o    exhibe.h       exhibe_server.o  exhibe.x
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo1$ ./exibe_server
```

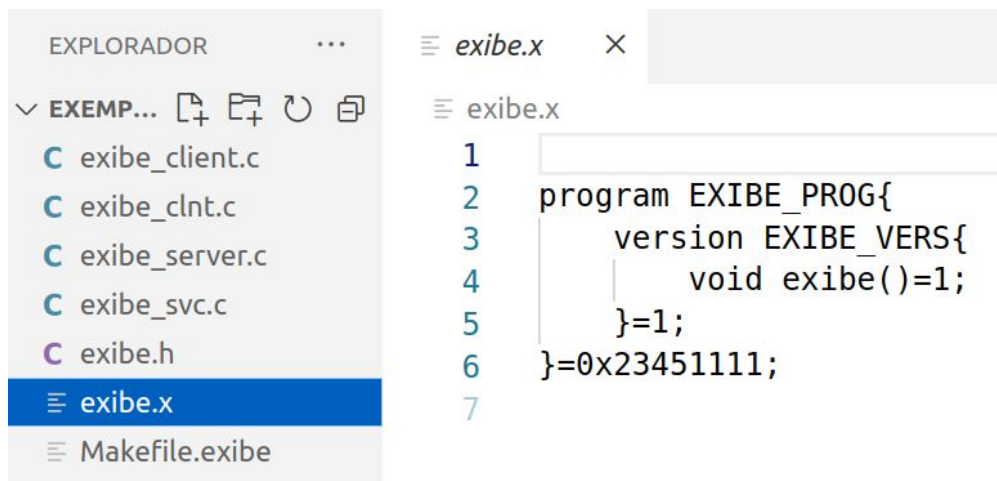
4. Exemplos práticos: exemplo 1



- O slide anterior mostrou como usar o mecanismo;
- Nada foi exibido;
- A partir de então serão construídos exemplos de minha autoria a fim de mostrar diferentes recursos do rpc:
 1. Exibe 1;
 2. Exibe 2;
 3. Pergunta;
 4. Quadrado;
 5. Soma;
 6. Calculadora;
 7. Operação.

4. Exemplos práticos: exemplo 2 - exhibe 2

- O método invocado no servidor exhibe uma mensagem;
- Devem ser repetidos cada um dos passos;
- Eles foram executados na pasta ~/Downloads/rpc/exemplo2;

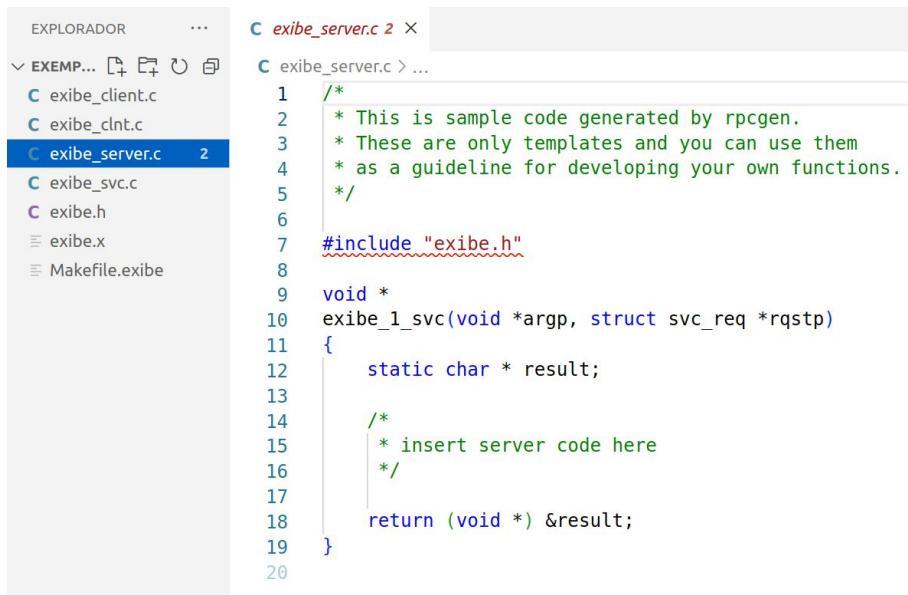


The image shows a file explorer window on the left and a code editor window on the right. The file explorer, titled 'EXPLORADOR', shows a directory structure with files: `exibe_client.c`, `exibe_clnt.c`, `exibe_server.c`, `exibe_svc.c`, `exibe.h`, `exibe.x` (highlighted), and `Makefile.exibe`. The code editor, titled 'exibe.x', shows the following code:

```
1  
2 program EXIBE_PROG{  
3     version EXIBE_VERS{  
4         void exhibe()=1;  
5     }=1;  
6 }=0x23451111;  
7
```

4. Exemplos práticos: exemplo 2 - exhibe 2

- Código padrão:



The screenshot shows a code editor interface. On the left is a file explorer pane titled 'EXPLORADOR' with a list of files: 'EXEMP...', 'exibe_client.c', 'exibe_clnt.c', 'exibe_server.c' (highlighted with a blue bar and the number '2'), 'exibe_svc.c', 'exibe.h', 'exibe.x', and 'Makefile.exibe'. On the right is the code editor window titled 'C exhibe_server.c 2'. It displays the source code for 'exibe_server.c' with line numbers 1 through 20. The code is a C file generated by rpcgen, containing a comment block, an include statement for 'exibe.h', and a function definition for 'exibe_1_svc'.

```
1  /*
2  * This is sample code generated by rpcgen.
3  * These are only templates and you can use them
4  * as a guideline for developing your own functions.
5  */
6
7  #include "exibe.h"
8
9  void *
10 exhibe_1_svc(void *argp, struct svc_req *rqstp)
11 {
12     static char * result;
13
14     /*
15      * insert server code here
16      */
17
18     return (void *) &result;
19 }
20
```


4. Exemplos práticos: exemplo 2 - exhibe 2

- Código alterado:

```
void *  
exibe_1_svc(void *argp, struct svc_req *rqstp)  
{  
    static char * result;  
  
    printf("A função foi chamada no servidor !!!\n");  
  
    return (void *) &result;  
}
```

4. Exemplos práticos: exemplo 2 - exhibe 2

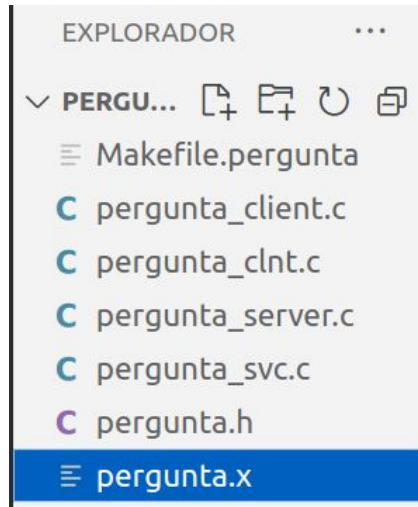
```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo2$ ./exibe_server
```

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo2$ ./exibe_client localhost  
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo2$
```

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/exemplo2$ ./exibe_server  
A função foi chamada no servidor !!!
```

4. Exemplos práticos: exemplo 3 - pergunta

- O método invocado no servidor entrega uma resposta;
- Devem ser repetidos cada um dos passos;
- Eles foram executados na pasta ~/Downloads/rpc/pergunta;



`pergunta.x` ✕

`pergunta.x`

```
1
2 program PERGUNTA_PROG{
3     version PERGUNTA_VERS{
4         int pergunta()=1;
5     }=1;
6 }=0x23451111;
7
```

4. Exemplos práticos: exemplo 3 - pergunta



- Código padrão:

```
int *
pergunta_1_svc(void *argp, struct svc_req *rqstp)
{
    static int  result;

    /*
     * insert server code here
     */

    return &result;
}
```

4. Exemplos práticos: exemplo 3 - pergunta



- Código alterado:

```
int *  
pergunta_1_svc(void *argp, struct svc_req *rqstp)  
{  
    static int  result;  
  
    result = 7;  
  
    return &result;  
}
```

4. Exemplos práticos: exemplo pergunta

3 -

- Código padrão:

```
void
pergunta_prog_1(char *host)
{
    CLIENT *clnt;
    int *result_1;
    char *pergunta_1_arg;

#ifdef DEBUG
    clnt = clnt_create (host, PERGUNTA_PROG, PERGUNTA_VERS, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif /* DEBUG */

    result_1 = pergunta_1((void*)&pergunta_1_arg, clnt);
    if (result_1 == (int *) NULL) {
        clnt_perror (clnt, "call failed");
    }

#ifdef DEBUG
    clnt_destroy (clnt);
#endif /* DEBUG */
}
```

4. Exemplos práticos: exemplo pergunta

3 -

- Código alterado:

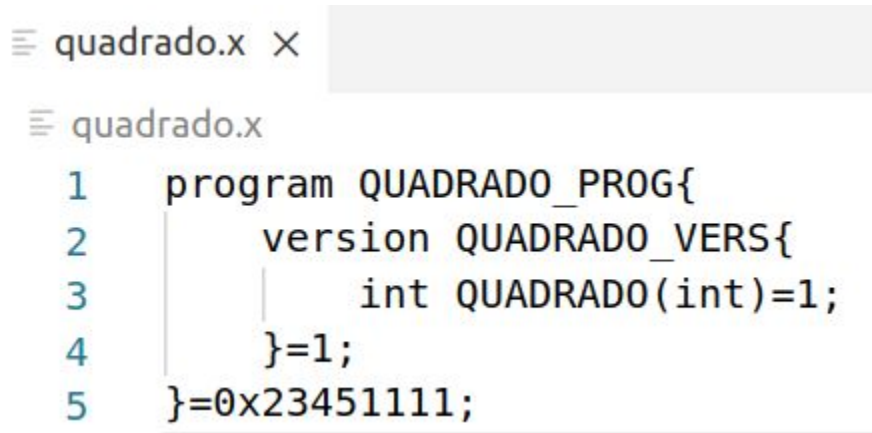
```
void
pergunta_prog_1(char *host)
{
    CLIENT *clnt;
    int *result_1;
    char *pergunta_1_arg;


#ifdef DEBUG
    clnt = clnt_create (host, PERGUNTA_PROG, PERGUNTA_VERS, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif /* DEBUG */

    result_1 = pergunta_1((void*)&pergunta_1_arg, clnt);
    if (result_1 == (int *) NULL) {
        clnt_perror (clnt, "call failed");
    }
    else
    {
        printf("Valor retornado: %d\n", *result_1);
    }
#ifdef DEBUG
    clnt_destroy (clnt);
#endif /* DEBUG */
}
```

4. Exemplos práticos: exemplo 4 - quadrado

- Ilustra como enviar um parâmetro;
- Devem ser repetidos cada um dos passos;
- Eles foram executados na pasta ~/Downloads/rpc/quadrado.






4. Exemplos práticos: exemplo 4 - quadrado

- Código padrão:


```
int *  
quadrado_1_svc(int *argp, struct svc_req *rqstp)  
{  
    static int result;  
  
    /*  
     * insert server code here  
     */  
  
    return &result;  
}
```



4. Exemplos práticos: exemplo 4 - quadrado

- Código alterado:

```
int *  
quadrado_1_svc(int *argp, struct svc_req *rqstp)  
{  
    static int  result;  
  
    printf("Valor recebido: %d\n", *argp);  
  
    result = *argp * *argp;  
  
    return &result;  
}
```



4. Exemplos práticos: exemplo 4 - quadrado


- Código padrão:

```
void
quadrado_prog_1(char *host)
{
    CLIENT *clnt;
    int *result_1;
    int quadrado_1_arg;

#ifdef DEBUG
    clnt = clnt_create (host, QUADRADO_PROG, QUADRADO_VERS, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif /* DEBUG */

    result_1 = quadrado_1(&quadrado_1_arg, clnt);
    if (result_1 == (int *) NULL) {
        clnt_perror (clnt, "call failed");
    }

#ifdef DEBUG
    clnt_destroy (clnt);
#endif /* DEBUG */
}
```



4. Exemplos práticos: exemplo 4 - quadrado

- Código alterado:

```
void
quadrado_prog_1(char *host)
{
    CLIENT *clnt;
    int *result_1;
    int  quadrado_1_arg = 7;

#ifdef DEBUG
    clnt = clnt_create (host, QUADRADO_PROG, QUADRADO_VERS, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif /* DEBUG */

    result_1 = quadrado_1(&quadrado_1_arg, clnt);
    if (result_1 == (int *) NULL) {
        clnt_perror (clnt, "call failed");
    }
    else
    {
        printf("Resultado: %d\n", *result_1);
    }

#ifdef DEBUG
    clnt_destroy (clnt);
#endif /* DEBUG */
}
```

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/quadrado$ ./quadrado_server
```

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/quadrado$ ./quadrado_client localhost  
Resultado: 49  
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/quadrado$ █
```

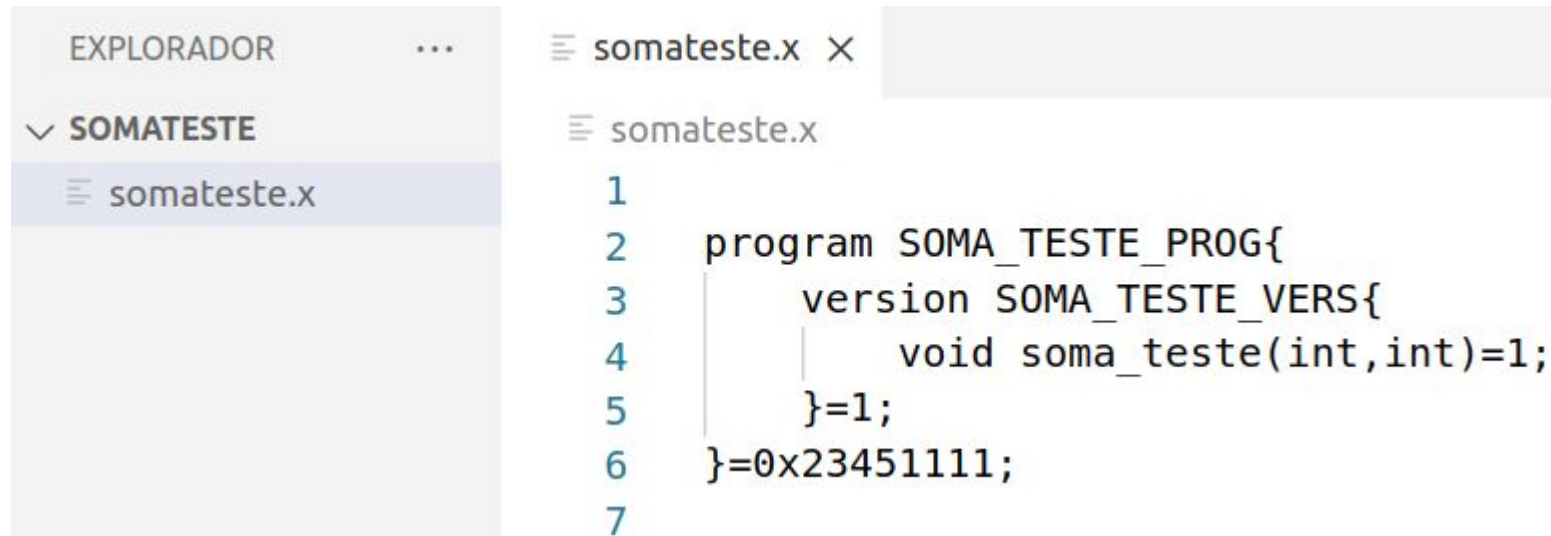
```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/quadrado$ ./quadrado_server  
Valor recebido: 7  
█
```

4. Exemplos práticos: exemplo 5 - soma



- Ilustra como enviar 2 parâmetros;
- Pode-se pensar que é necessário apenas colocar dois atributos int.
- Devem ser repetidos cada um dos passos;
- Eles foram executados na pasta ~/Downloads/rpc/somateste.

4. Exemplos práticos: exemplo 5 - soma



```
1
2  program SOMA_TESTE_PROG{
3      version SOMA_TESTE_VERS{
4          void soma_teste(int,int)=1;
5      }=1;
6  }=0x23451111;
7
```

4. Exemplos práticos: exemplo 5 - soma



- A prática mostra que, para enviar dois inteiros, não basta colocar apenas dois atributos;

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/somateste$ rpcgen -a -C somateste.x  
void soma_teste(int,int)=1;  
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  
somateste.x, line 4: only one argument is allowed  
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/somateste$
```

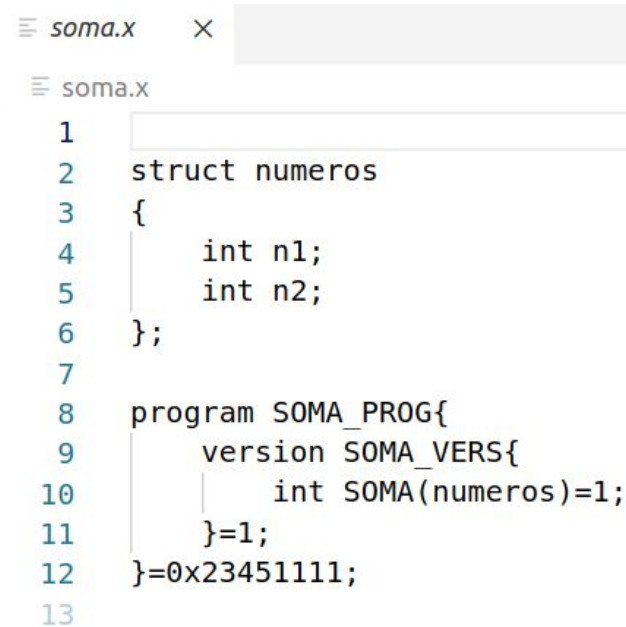
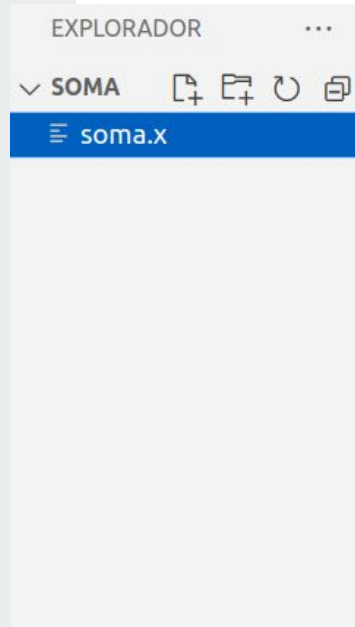

4. Exemplos práticos: exemplo 5 - soma

- Então, o que deve ser feito?



```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/somateste$ rpcgen -a -C somateste.x  
void soma_teste(int,int)=1;  
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  
somateste.x, line 4: only one argument is allowed  
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/somateste$
```

Deve-se criar um tipo de dados com o conteúdo desejado




4. Exemplos práticos: exemplo 5 - soma



- O código foi colocado na pasta ~/Downloads/rpc/soma;
- Devem ser repetidos cada um dos passos.


```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/soma$ rpcgen -a -C soma.x
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/soma$ ls
Makefile.soma  soma_client.c  soma_clnt.c  soma.h  soma_server.c  soma_svc.c  soma.x  soma_xdr.c
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/soma$
```



4. Exemplos práticos: exemplo 5 - soma

- Código padrão:

```
int *  
soma_1_svc(numeros *argp, struct svc_req *rqstp)  
{  
    static int result;  
  
    /*  
     * insert server code here  
     */  
  
    return &result;  
}
```



4. Exemplos práticos: exemplo 5 - soma

- Código alterado:

```
int *  
soma_1_svc(numeros *argp, struct svc_req *rqstp)  
{  
    static int result;  
  
    result = argp->n1 + argp->n2;  
  
    return &result;  
}
```

4. Exemplos práticos: exemplo 5 - soma

- Código padrão:

```
void
soma_prog_1(char *host)
{
    CLIENT *clnt;
    int *result_1;
    numeros soma_1_arg;

#ifdef DEBUG
    clnt = clnt_create (host, SOMA_PROG, SOMA_VERS, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif /* DEBUG */

    result_1 = soma_1(&soma_1_arg, clnt);
    if (result_1 == (int *) NULL) {
        clnt_perror (clnt, "call failed");
    }
#ifdef DEBUG
    clnt_destroy (clnt);
#endif /* DEBUG */
}
```

4. Exemplos práticos: exemplo 5 - soma

- Código alterado:

```
void
soma_prog_1(char *host)
{
    CLIENT *clnt;
    int *result_1;
    numeros soma_1_arg;
    soma_1_arg.n1 = 5;
    soma_1_arg.n2 = 7;

#ifdef DEBUG
    clnt = clnt_create (host, SOMA_PROG, SOMA_VERS, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif /* DEBUG */

    result_1 = soma_1(&soma_1_arg, clnt);
    if (result_1 == (int *) NULL) {
        clnt_perror (clnt, "call failed");
    }
    else
    {
        printf("O resultado foi: %d\n", *result_1);
    }
#ifdef DEBUG
    clnt_destroy (clnt);
#endif /* DEBUG */
}
```

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/soma$ ./soma_server
```

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/soma$ ./soma_client localhost  
0 resultado foi: 12  
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/soma$
```

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/soma$ ./soma_server
```


4. Exemplos práticos: exemplo 6 - calculadora




- Ilustra como ter mais de uma função;
- Devem ser repetidos cada um dos passos;
- Eles foram executados na pasta ~/Downloads/rpc/calculadora.

4. Exemplos práticos: exemplo 6 - calculadora

≡ calculadora.x ×

≡ calculadora.x

```
1
2 struct numeros
3 {
4     int n1;
5     int n2;
6 };
7
8 program CALCULADORA_PROG{
9     version CALCULADORA_VERS{
10         int SOMA(numeros)=1;
11         int SUB(numeros)=2;
12     }=1;
13 }=0x23451111;
14
```



4. Exemplos práticos: exemplo 6 - calculadora

- Código padrão:

```
int *
soma_1_svc(numeros *argp, struct svc_req *rqstp)
{
    static int  result;


    /*
     * insert server code here
     */

    return &result;
}
```

```
int *
sub_1_svc(numeros *argp, struct svc_req *rqstp)
{
    static int  result;

    /*
     * insert server code here
     */

    return &result;
}
```



4. Exemplos práticos: exemplo 6 - calculadora

- Código alterado:

```
int *  
soma_1_svc(numeros *argp, struct svc_req *rqstp)  
{  
    static int result;  
  
    result = argp->n1 + argp->n2;  
  
    return &result;  
}
```

```
int *  
sub_1_svc(numeros *argp, struct svc_req *rqstp)  
{  
    static int result;  
  
    result = argp->n1 - argp->n2;  
  
    return &result;  
}
```


4. Exemplos práticos: exemplo 6 - calculadora

- Código padrão:

```
void
calculadora_prog_1(char *host)
{
    CLIENT *clnt;
    int *result_1;
    numeros soma_1_arg;
    int *result_2;
    numeros sub_1_arg;

#ifdef DEBUG
    clnt = clnt_create (host, CALCULADORA_PROG, CALCULADORA_VERS, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif /* DEBUG */

    result_1 = soma_1(&soma_1_arg, clnt);
    if (result_1 == (int *) NULL) {
        clnt_perror (clnt, "call failed");
    }
    result_2 = sub_1(&sub_1_arg, clnt);
    if (result_2 == (int *) NULL) {
        clnt_perror (clnt, "call failed");
    }
#ifdef DEBUG
    clnt_destroy (clnt);
#endif /* DEBUG */
}
```



4. Exemplos práticos: exemplo 6 - calculadora

- Código alterado parte 1:

```
void
calculadora_prog_1(char *host)
{
    CLIENT *clnt;
    int *result_1;
    numeros soma_1_arg;
    soma_1_arg.n1 = 7;
    soma_1_arg.n2 = 5;
    int *result_2;
    numeros sub_1_arg;
    sub_1_arg.n1 = 7;
    sub_1_arg.n2 = 5;

#ifdef DEBUG
    clnt = clnt_create (host, CALCULADORA_PROG, CALCULADORA_VERS, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif /* DEBUG */
}
```

4. Exemplos práticos: exemplo 6 - calculadora

- Código alterado parte 2:

```
result_1 = soma_1(&soma_1_arg, clnt);
if (result_1 == (int *) NULL) {
    clnt_perror (clnt, "call failed");
}
else
{
    printf("Resultado da soma: %d\n", *result_1);
}
result_2 = sub_1(&sub_1_arg, clnt);
if (result_2 == (int *) NULL) {
    clnt_perror (clnt, "call failed");
}
else
{
    printf("Resultado da subtração: %d\n", *result_2);
}
#ifdef DEBUG
    clnt_destroy (clnt);
#endif /* DEBUG */
}
```

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/calculadora$ ./calculadora_server
```

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/calculadora$ ./calculadora_client localhost  
Resultado da soma: 12  
Resultado da subtração: 2  
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/calculadora$
```


4. Exemplos práticos: exemplo 7 - operação




- Ilustra:
 - a. Como ter estruturas compostas;
 - b. Que é possível usar diferentes tipos de dados;
 - c. Que o retorno também pode ser uma estrutura.
- Devem ser repetidos cada um dos passos;
- Eles foram executados na pasta ~/Downloads/rpc/operacao.

4. Exemplos práticos: exemplo 7 - operação

operacao.x X

operacao.x

```
1
2 struct numeros
3 {
4     int n1;
5     int n2;
6 };
7
8 struct partes
9 {
10     struct numeros parte1;
11     struct numeros parte2;
12     int peso1;
13     float peso2;
14     double peso3;
15     char tipo;
16 };
17
18 struct resultado
19 {
20     int status;
21     double retorno;
22 };
23
24 program SOMA_PROG{
25     version SOMA_VERS{
26         resultado OPERATION(partes)=1;
27     }=1;
28 }=0x23451111;
```



4. Exemplos práticos: exemplo 7 - operação

- Código original:

```
resultado *
operation_1_svc(partes *argp, struct svc_req *rqstp)
{
    static resultado result;

    /*
     * insert server code here
     */

    return &result;
}
```

4. Exemplos práticos: exemplo 7 - operação

- Código alterado:

```
resultado *  
operation_1_svc(partes *argp, struct svc_req *rqstp)  
{  
    static resultado result;  
  
    printf("O tipo de operação é: %c", argp->tipo);  
  
    int p1 = argp->parte1.n1 * argp->parte2.n1;  
    int p2 = argp->parte1.n2 * argp->parte2.n2;  
    float v = argp->peso1 * p1 + argp->peso2 * p2;  
    result.retorno = argp->peso3 * v;  
  
    if (argp->tipo == 'a' && result.retorno > 7)  
    |     result.status = 1;  
    else  
    |     result.status = 2;  
  
    return &result;  
}
```

4. Exemplos práticos: exemplo 7 - operação

- Código original:

```
void
soma_prog_1(char *host)
{
    CLIENT *clnt;
    resultado *result_1;
    partes operation_1_arg;

#ifdef DEBUG
    clnt = clnt_create (host, SOMA_PROG, SOMA_VERS, "udp");
    if (clnt == NULL) {
        clnt_pcreateerror (host);
        exit (1);
    }
#endif /* DEBUG */

    result_1 = operation_1(&operation_1_arg, clnt);
    if (result_1 == (resultado *) NULL) {
        clnt_perror (clnt, "call failed");
    }
#ifdef DEBUG
    clnt_destroy (clnt);
#endif /* DEBUG */
}
```

4. Exemplos práticos: exemplo 7 - operação

- Código alterado parte 1:

```
void  
soma_prog_1(char *host)  
{  
    CLIENT *clnt;  
    resultado *result_1;  
    partes operation_1_arg;  
  
    operation_1_arg.parte1.n1 = 1;  
    operation_1_arg.parte1.n2 = 2;  
    operation_1_arg.parte2.n1 = 3;  
    operation_1_arg.parte2.n2 = 4;  
    operation_1_arg.peso1 = 7;  
    operation_1_arg.peso2 = 7.77;  
    operation_1_arg.peso3 = 7.7777777;  
    operation_1_arg.tipo = 'a';  
  
#ifndef DEBUG  
    clnt = clnt_create (host, SOMA_PROG, SOMA_VERS, "udp");  
    if (clnt == NULL) {  
        clnt_pcreateerror (host);  
        exit (1);  
    }  
#endif /* DEBUG */
```

4. Exemplos práticos: exemplo 7 - operação

- Código alterado parte 2:

```
operation_1_arg.peso2 = 7.77;  
operation_1_arg.peso3 = 7.7777777;  
operation_1_arg.tipo = 'a';  
  
#ifndef DEBUG  
    clnt = clnt_create (host, SOMA_PROG, SOMA_VERS, "udp");  
    if (clnt == NULL) {  
        clnt_pcreateerror (host);  
        exit (1);  
    }  
#endif /* DEBUG */  
  
    result_1 = operation_1(&operation_1_arg, clnt);  
    if (result_1 == (resultado *) NULL) {  
        clnt_perror (clnt, "call failed");  
    }  
    else  
    {  
        printf("O retorno é: %lf\n", result_1->retorno);  
        printf("O status é: %d\n", result_1->status);  
    }  
#ifndef DEBUG  
    clnt_destroy (clnt);  
#endif /* DEBUG */  
}
```

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/operacao$ ./operacao_server
```

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/operacao$ ./operacao_client localhost
```

```
0 retorno é: 646.800022
```

```
0 status é: 1
```

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/operacao$
```

```
daniel@daniel-Aspire-A515-55:~/Downloads/rpc/operacao$ ./operacao_server
```




Conclusão

- O RPC é um recurso muito poderoso;
- Ele disponibiliza diversos recursos para uma comunicação;
- Sistemas eficientes podem ser feitos com base em sua utilização.



Dúvidas?





Obrigado!

Contato:

Me. Daniel Sucupira Lima

E-mail:

daniel.lima@aluno.uece.br



Referências



- MAZIERO, Carlos. **Memória processo**. 2021. 1 imagem. 415 x 283 pixels. Disponível em: <https://wiki.inf.ufpr.br/maziero/lib/exe/fetch.php?media=prog2:memoria-processo.png>. Acesso em: 22 de abril de 2023.
- ARORA, Ridze. **Remote procedure call**. 2020. 1 imagem. 500 x 250 pixels. Disponível em: <https://i.pinimg.com/564x/68/7b/2c/687b2cf1cff474b693ec10c97a7a6c0a.jpg>. Acesso em: 2 de abril de 2023.
- KRZYZANOWSKI, Paul. **Steps in executing a remote procedure call**. 2021. 1 imagem. 482 x 303 pixels. Disponível em: <https://people.cs.rutgers.edu/~pxk/417/notes/images/rpc-flow.png>. Acesso em: 2 de abril de 2023.
- WANG, Stephanie; HINDMAN, Benjamin; STOICA, Ion. In reference to RPC: it's time to add distributed memory. In: **HotOS**. 2021. p. 191-198.
- PAWAN, V. **Steps of RPC**. 2021. 1 imagem. 540 x 303 pixels. Disponível em: https://media.licdn.com/dms/image/C5612AQE7IHr8My57Nw/article-inline_image-shrink_1500_2232/0/1624297081904?e=1687392000&v=beta&t=iWAOxDx4MS-9FsHRjiBA2x42-h4BjOZDuM8MgZD71gU. Acesso em: 2 de abril de 2023.
- Emojiterra team. **Rosto pensativo emoji**. 2017. 1 imagem. 303 x 303 pixels. Disponível em: <https://images.emojiterra.com/google/android-12l/512px/1f914.png>. Acesso em: 2 de abril de 2023.
- EV Comunicação. **Conheça seu público alvo**. 2015. 1 imagem. 461 x 303 pixels. Disponível em: <http://varizes.net.br/wp-content/uploads/2019/06/duvida1-970x637.jpg>. Acesso em: 2 de abril de 2023.