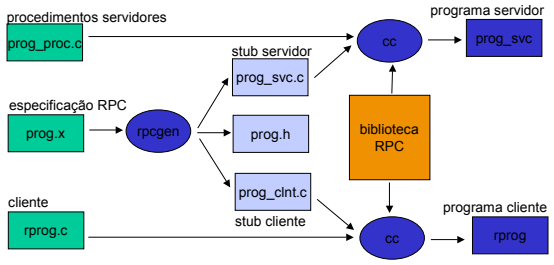
rpcgen - Funcionamento



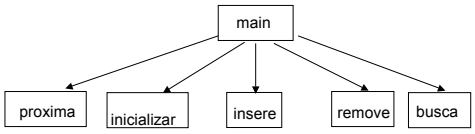
rpcgen

- Exemplo de arquivo de especificação:

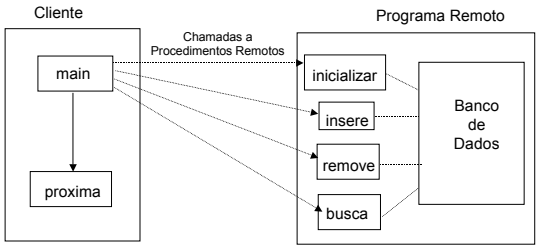
```
/* date.x
   especificação de serviços remotos de data e hora */

program DATE_PROG{
    version DATE_VERS{
        long BIN_DATE(void) = 1;
        string STR_DATE(long) = 2;
    } = 1;
} = 0x31234567;
```

Passo 1: Construir uma Aplicaçao Convencional



Passo 2: Dividir o programa em duas partes



Passo 3: Criar uma Especificação Rpcgen

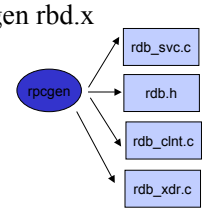
```
/* rbd.x
   especificação rpc para um programa de banco de dados
   que oferece os procedimentos INSERE, REMOVE e BUSCA
*/

struct example {   /* estrutura não usada, declarada para ilustrar como rpcgen */
        int exfield1;  /* constrói rotinas XDR para converter estruturas */
        char exfield2;
};

program RBDPROG{  /* nome do programa remoto */
     version RDBVERS{  /* declaração da versão */
         int INICIALIZAR(void) = 1; /* primeiro procedimento deste programa */
         int INSERE(string) = 2; /* segundo procedimento deste programa */
        int REMOVE(string) = 3; /* terceiro procedimento deste programa */
        int BUSCA(string) = 4; /* quarto procedimento deste programa */
    } = 1; /* definição da versão do programa */
} = 0x30090949; /* número do programa remoto (deve ser único) */
```

Passo 4: Rodar o Rpcgen

- rpcgen rbd.x



Rpcgen
Arquivo .h

```
/* rbd.h */

struct example {
        int exfield1;
        char exfield2;
};

typedef sruct example example;
bool_t xdr_example();

#define RBDPROG (u_long) 0x30090949)
#define RDBVERS ((u_long) 1)
#define INICIALIZAR ((u_long) 1)
extern int *inicializar_1();
#define INSERE ((u_long) 2)
extern int *insere_1();
#define REMOVE ((u_long) 3)
extern int *remove_1();
#define BUSCA ((u_long) 4)
extern int *busca_1();
```

Rpcgen
Arquivo de Conversão XDR

```
/* rbd_xdr.c */
#include <rpc/rpc.h>
#include "rbd.h"

bool_t
xdr_example(xdrs, objp)
        XDR *xdrs;
        example *objp;
{
        if (!xdr_int(xdrs, &objp->exfield1)) {
                return(FALSE);
        }
        if (!xdr_char(xdrs, &objp->exfield2) {
                return(FALSE);
        }
        return(TRUE);
}
```

Rpcgen
Stub do Cliente

```
/* rbd_clnt.c */
#include <rpc/rpc.h>
#include "rbd.h"

int * inicializar_1(argp, clnt)
    void *argp;
    CLIENT *clnt;
{
    static int res;

    bzero((char *)&res, sizeof(res));
    if (clnt_call(clnt, INICIALIZAR, xdr_void,
    argp,  xdr_int, &res, TIMEOUT) != RPC_SUCCESS)
            return (NULL);
    return (&res);
}

int *insere_1(argp, clnt)
    char **argp;
    CLIENT *clnt;
{
    static int res;
    bzero((char *)&res, sizeof(res));
    if (clnt_call(clnt, INSERE, xdr_wrapstring,
    argp, xdr_int, &res, TIMEOUT) != RPC_SUCCESS)
            return (NULL);
    return (&res);
}
```

```
int * remove_1(argp, clnt)
    char **argp;
    CLIENT *clnt;
{
    static int res;

    bzero((char *)&res, sizeof(res));
    if (clnt_call(clnt, REMOVE, xdr_wrapstring,
    argp, xdr_int, &res, TIMEOUT) != RPC_SUCCESS)
            return (NULL);
    return (&res);
}

int *busca_1(argp, clnt)
    char **argp;
    CLIENT *clnt;
{
    static int res;
    bzero((char *)&res, sizeof(res));
    if (clnt_call(clnt, BUSCA, xdr_wrapstring,
    argp, xdr_int, &res, TIMEOUT) != RPC_SUCCESS)
            return (NULL);
    return (&res);
}
```

Rpcgen
Stub do Servidor

```
/* rbd_svc.c */
#include <rpc/rpc.h>
#include "rbd.h"

static void rbdprog_1();

main()
{
    SVCXPRT *transp;
    (void)pmap_unset(RBDPROG, RBDVERS);

    transp = svcudp_create(RPC_ANYSOCK);
    if (transp == NULL) {
       (void) fprintf("Não pode criar serviço udp\n");
       exit(1);
    }
    if (!svc_register(transp, RBCPROG, RBDVERS, rbdprog_1,
       IPPROTO_UDP)) {
       (void) fprintf("Não pode registrar tal prog.\n");
       exit(1);
    }
    transp = svctcp_create(RPC_ANYSOCK, 0, 0);
    if (transp == NULL) {
       (void) fprintf("Não pode criar serviço TCP\n");
       exit(1);
     }
    if (!svc_register(transp, RBCPROG, RBDVERS, rbdprog_1,
       IPPROTO_TCP)) {
       (void) fprintf("Não pode registrar tal prog.\n");
        exit(1);
    }
```

```
    svc_run();
    (void)fprintf("SVC_RUn retornado \n");
    exit(1);
}

static void rbdprog_1(rqstp, transp)
    struct svc_req *rqstp;
    SVCXPRT *transp;
{
    union {
            char *insere_1_arg;
            char *remove_1_arg;
            char *busca_1_arg;
    } argument;
    char *result;
    bool_t (*xdr_argument) (), (*xdr_result)();
    char *(*local)();

    switch (rqstp->rq_proc) {
    case NULLPROC:
     ( void)svc_sendreply(transp, xdr_void,(char *) NULL);
      return;
    case INICIALIZAR:
      xdr_argument = xdr_void;
      xdr_result = xdr_int;
      local = (char *(*)())inicializar_1;
      break;
    case INSERE:
      xdr_argument = xdr_wrapstring;
      xdr_result = xdr_int;
      local = (char *(*)()) insere_1;
      break;
```

Rpcgen
Continuação Stub do Servidor

```
    case REMOVE:
      xdr_argument = xdr_wrapstring;
      xdr_result = xdr_int;
      local = (char *(*)())remove_1;
      break;
    case BUSCA:
      xdr_argument = xdr_wrapstring;
      xdr_result = xdr_int;
      local = (char *(*)()) busca_1;
      break;
    default:
      svcerr_noproc(transp);
      return;
    }
    bzero((char*)&argument, sizeof(argument));
    if (!svc_getargs(transp, xdr_argument, &argument)) {
       svcerr_decode(transp);
       return;
    }
    result = (*local)(&argument, rqstp);
    if (result != NULL && !svc_sendreply(transp,
         xdr_result, result )) {
            svcerr_systemerr(transp);
    }
    if (!svc_freeargs(transp, xdr_argument, &argument)) {
        (void)fprintf("Problema nos argumentos\n");
        exit(1);
    }
}
```

Passo 5: Escrever procedimentos de Interface com o Stub

- Rotinas de Interface do Cliente

```
/* rbd_cif.c - inicializar, insere, remove, busca */
#include <rpc/rpc.h>
#include "rbd.h"
extern CLIENT *handle;  /* handle para procedimento
      remoto */

int inicializar()
{
      return *inicializar_1(handle);
}
int insere(item)
char *item;
{
      char **arg;
      arg = &item;
      return *insere_1(arg, handle);
}
```

```
int remove(item)
char *item;
{
    char **arg;
    arg = &item;
    return *remove_1(arg, handle);
}
int busca(item)
char *item;
{
    char **arg;
    arg = &item;
    return *busca_1(arg, handle);
}
```

Rotinas de Interface do Servidor

```
/* rbd_sif.c - inicializar_1, insere_1, remove_1, busca_1
      */
#include <rpc/rpc.h>
#include "rbd.h"
static int retcode;

int *inicializar_1()
{
      retcode = inicializar();
      return &retcode;
}
int *insere_1(i)
char **i;
{
      retcode = insere(*i);
      return &retcode;
}
```

```
int *remove_1(i)
char **i;
{
    retcode = remove(*i);
    return &retcode;
}
int *busca_1(i)
char **i;
{
    retcode = busca(*i);
    return &retcode;
}
```

Passo 6: Compilar e Linkar o Programa Cliente

- cc -c rbd_cif.c ⟶ rbd_cif.o
- cc -c rbd.c ⟶ rdb.o
- cc -o rbd rbd.o rbd_clnt.o rbd_xdr.o rbd_cif.o

Programa Cliente

```
/* rbd.c - main, proxima*/
#include <rpc/rpc.h>
#include "rbd.h"

#define RMACHINE  "localhost"  /* nome da máquina remota */
CLIENT *handle;   /* handle para um procedimento remoto */

int main(argc, argv)
int argc;
char *argv[];
{
      char palavra[MAXWORD+1];
      char cmd;
      int plvlen; /* tamanho da palavra */

   /* Seta a conexão para uma chamada remota de procedimento */
      handle = clnt_create(RMACHINE, RBDPROG, RBDVERS, "tcp");
      if (handle == 0) {
        printf("Não pode conectar programa remoto\n");
        exit(1);
      }
      while (1) {
        plvlen = proxima(&cmd, palavra);
        if (plvlen < 0)
          exit(0);
```

```
        switch (cmd) {
        case 'i':   /*inicialize */
                    inicializar();
                printf("BD inicializado \n");
                break;
        case 'i':   /*insere */
                    insere(palavra);
                printf("%s Inserida \n", palavra);
                break;
        case 'b':   /*busca */
                    if (busca(palavra))
                      printf("%s foi encontrada \n", palavra);
                else  printf("%s não existe \n", palavra);
                break;
        case 'r':   /*remove */
                    if (remove(palavra))
                     printf("%s removida \n", palavra);
                    else
                        printf("%s não encontrada"\n", palavra);
                break;
       case 'q':    /*quit */
                    printf("programa encerrado \n");
                    exit(0);
         default:   /* entrada ilegal */
                    printf("Comando inválido \n");
                    break;
          }
      }
}
```

Continuação Programa Cliente

```
int proxima(cmd, palavra)
char *cmd, *palavra;
{
        int i, ch;
        ch = getc(stdin);
        while (ch == ' ')
                ch = getc(stdin);
        if (ch == EOF)
                return -1;
        *cmd = (char) ch;
        ch = getc(stdin);
        while (ch = ' ')
                ch = getc(stdin);
        if (ch == EOF)
                return -1;
        if (ch == '\n')
                return 0;
        i = 0;
        while ((ch != ' ') && (ch != '\n')) {
                if (++i > MAXWORD) {
                  printf("Erro, palavra muito longa \n");
                  exit(1);
                }
                *palavra++ = ch;
                ch = getc(stdin);
        }
        return i;
}
```

Passo 7: Compilar e Linkar o Programa Servidor

- cc -c rbd_sif.c → rbd_sif.o
- cc -c rbd_srp.c → rdb_srp.o
- cc -o rbddaemon rbd_svc.o rbd_xdr.o rbd_sif.o rbd_srp.o

Programa Servidor

```
/* rbd_srp.c - inicializar, insere, remove, busca*/
#include <rpc/rpc.h>
#include "rbd.h"

/* Procedimentos remotos do servidor e dados globais */

char bd[BDSIZE][MAXWORD+1] /* armazena o dicionário de palavras */
int   npalavras = 0;  /* número de palavras no dicionário */

int inicializar()
{
        npalavras = 0;
        return 1;
}

int insere(palavra)
char *palavra;
{
        strcpy(bd[npalavras], palavra);
        npalavras++;
        return npalavras;
}
```

```
int remove(palavra)
char *palavra;
{
    int i;
    for (i=0; i<npalavras; i++)
            if (strcmp(palavra, bd[i]) == 0) {
                npalavras--;
                strcpy(bd[i], bd[npalavras]);
                return 1;
            }
    return 0;
}

int busca(palavra)
char *palavra;
{
    int i;
    for (i=0; i<npalavras; i++)
            if (strcmp(palavra, bd[i]) == 0 )
                return 1;
    return 0;
}
```

Passo 8: Iniciar o Servidor e Executar o Cliente

- Iniciar o servidor em background: rbddaemon&
- Executar o Cliente: rbd