UNIVERSIDADE DE SÃO PAULO

Instituto de Ciências Matemáticas e de Computação

ISSN 0103-2585

SISTEMAS GERADOS DE STUBS – GASP – MANUAL DO USUÁRIO: INSTALAÇÃO E UTILIZAÇÃO

A. H. H. NASCIMENTO
M. J. SANTANA
N. CALOHEGO JR.
R. H. C. SANTANA

Nº 18

NOTAS DIDÁTICAS

São Carlos – SP
1995

Agradecimentos

À CAPES, à FAPESP e ao CNPq, pelo apoio financeiro.

Índice

INTRODUÇÃO

Os sistemas computacionais (*hardware e software*) têm evoluído bastante, principalmente nesta última década. Este trabalho não foge à realidade e ataca uma das áreas em que o mercado vem investindo cada vez mais: o desenvolvimento de aplicações distribuídas, ou seja, o projeto de sistemas utilizando redes de computadores.

O auge hoje é a instalação de redes de computadores, a implantação de servidores de arquivos, servidores de impressão, conexão com a internet, protocolos específicos (para transferência de arquivos, para correio eletrônico, para ligação remota, por exemplo), entre outros.

O sistema Gerador Automático de Procedimentos *Stubs* (GAPS), desenvolvido neste trabalho, é uma ferramenta de auxilio à implementação de aplicações distribuídas, com a qual tem-se um ganho extremamente grande de produtividade e facilidade no desenvolvimento.

O GAPS foi desenvolvido sobre os protocolos de comunicação implementados pelo grupo de Sistemas Distribuídos e Programação Concorrente do LaSD (Laboratório de Sistemas Digitais) do ICMSC (Instituto de Ciências Matemáticas de São Carlos).

O sistema computacional do LaSD é composto por uma rede de microcomputadores PCs sob o sistema operacional DOS e utiliza os protocolos definidos no *packet driver* obtidos da Universidade de Clarkson.

Este trabalho descreve os detalhes de instalação e de utilização do sistema GAPS, através de exemplos, e é composto pelas seguintes seções, além desta seção introdutória:

Na seção 1 são descritos os passos para instalação do sistema GAPS e suas possíveis configurações.

A seção 2 contém a descrição da linguagem de entrada do gerador e são mostrados, através de exemplos, alguns detalhes da linguagem.

Os passos necessários para geração dos procedimentos *stubs* estão descritos na seção 3, dando continuidade ao desenvolvimento de aplicações distribuídas sobre os exemplos mostrados na seção anterior.

Na seção 4 são mostrados os procedimentos para compilação e execução dos processos cliente e servidor.

Finalizando, encontram-se a bibliografia e os apêndices.

SEÇÃO 1

INSTALAÇÃO

Nesta seção encontram-se os passos necessários para instalação do sistema GAPS e suas possíveis configurações para executar os processos cliente e servidor.

1.1- Equipamento necessário para instalação e utilização

Existem duas possibilidades de exigências para instalar e trabalhar com o sistema GAPS: a primeira relacionada à geração dos procedimentos *stubs* e a segunda, utilizada para execução dos processos cliente ou servidor.

Com relação a primeira hipótese, as exigências mínimas são:

1. PC ou compatível;
2. 1 (uma) unidade de disco flexível;
3. 1 (uma) unidade de disco rígido; e
4. MS-DOS versão 3 ou superior.

Para execução da aplicações, exigem-se, pelo menos:

1. PC ou compatível;
2. 1 (uma) unidade de disco flexível;
3. 1 (uma) unidade de disco rígido;

4. MS-DOS versão 3 ou superior; e

5. Placa ethernet ou compatível, para interligação dos equipamentos.

Além destas exigências, é necessário espaço disponível no disco rígido, o que será comentado posteriormente, pois depende do tipo de configuração escolhida.

1.2- O disquete de instalação

O disquete de instalação é composto pelos seguintes programas e arquivos:

1. O programa de instalação (INSTALA.EXE);
2. O programa gerador de *stubs* (GAPS.EXE);
3. O protocolo de rede *Packet Driver* (NE1000 e NE2000);
4. Os *softwares* de conexão (LOGIN.EXE) e desconexão (LOGOUT.EXE) do cliente com o servidor;
5. Os arquivos de utilização no cliente (ESTACAO.CFG e SERVIDOR.ES) e no servidor (USUARIOS);
6. O diretório FONTES contendo os arquivos fontes do programa de instalação e do gerador;
7. O diretório LOG contendo os arquivos fontes dos programas de conexão e desconexão;
8. O diretório REDE contendo os arquivos fontes dos protocolos de rede PKT, SPP e SES; e
9. O diretório EXEMPLOS contendo vários subdiretórios com exemplos de aplicações distribuídas desenvolvidas utilizando-se do gerador.

Estes arquivos e programas serão explicados mais adiante.

1.3- Executando o programa de instalação

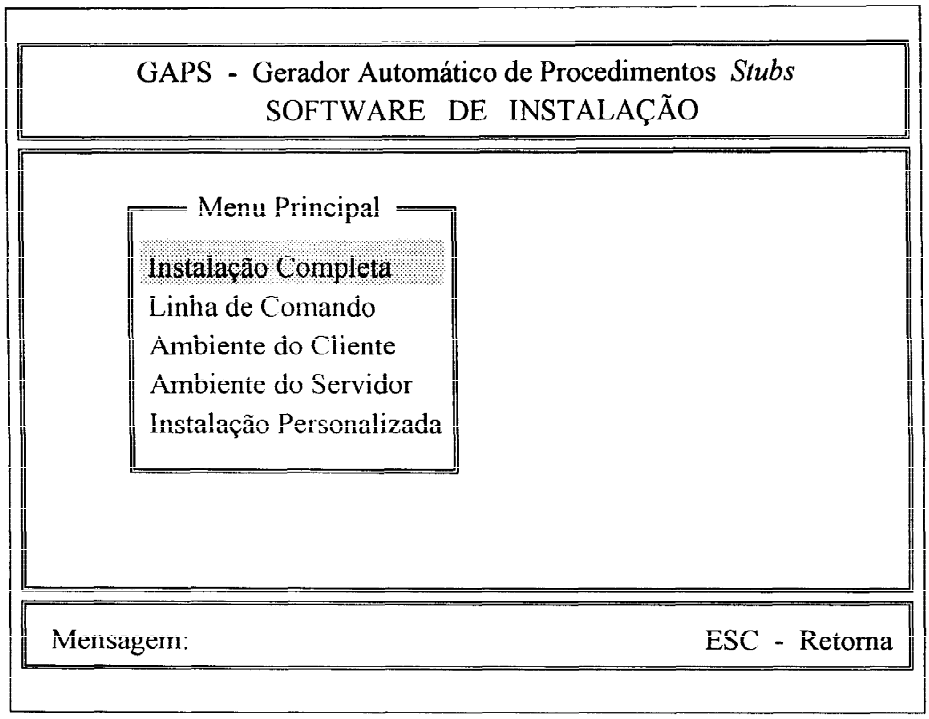
Para iniciar a instalação do sistema GAPS, devem ser executados os seguintes passos:

1. colocar o disquete de instalação na unidade de disco A: ou B:;
2. mudar para a unidade de disco, a qual contém o disquete de instalação; e
3. executar o programa INSTALA.EXE, da seguinte forma:

   A:\> instala <enter>

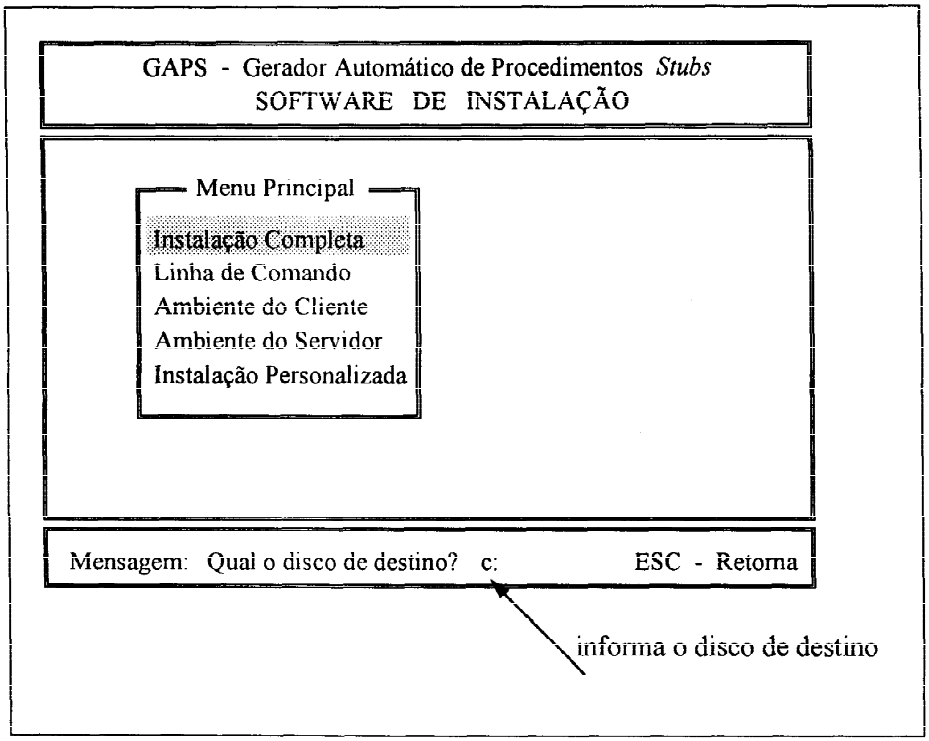
   Caso seja executado da unidade A:.

Com isto, a seguinte tela irá aparecer:



Como pode ser verificado, existem cinco tipos de instalação dos *softwares* do sistema gerador de *stubs*.

Após a solicitação de algum tipo de instalação, o sistema perguntará qual o disco rígido que será instalado o sistema, colocando o default na tela (C:).



O subdiretório \gaps será automaticamente criado e o programa "autoexec.bat" é atualizado com o seguinte comando:

PATH=c:\gaps;%path%

Caso seja confirmada a instalação no disco C:.

A primeira, chamada Intalação Completa, realiza a instalação de todos os programas e arquivos dos sistema, sem exceção. Essa opção precisa de 1,5 Mbytes de disco disponível (Será comentada mais adiante).

A segunda possibilidade, Linha de Comando, só instala no disco rígido o progama gerador de *stubs* e os protocolos de rede. Com esse tipo de instalação não há possibilidade de executar o processo cliente nem o servidor, mas apenas criar o módulo de definição da linguagem de interface, submetê-la ao programa gerador para criar os módulos que conterão os procedimentos stubs e, no máximo, compilar os processos cliente e servidor. Essa instalação consome cerca de 250 Kbytes do disco rígido.

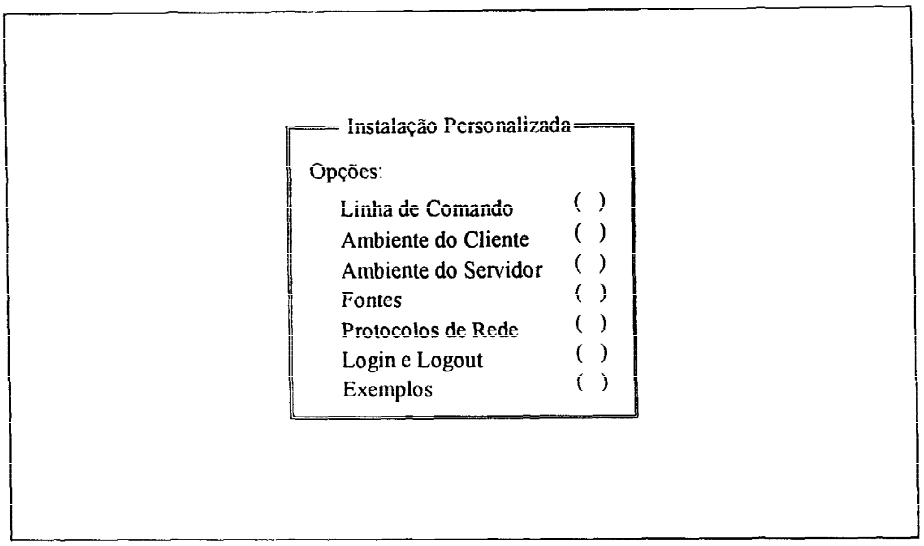
A terceira opção, Ambiente do Servidor, é o módulo responsável por configurar o microcomputador para apenas executar o processo servidor, ou seja, essa instalação só carrega os softwares necessários para a execução do servidor, o qual já deve estar compilado. Consome aproximadamente 20 Kbytes de disco.

A opção quatro, Ambiente do Cliente, é, semelhante à opção anterior, responsável apenas por instalar os programas necessários à execução do processo cliente. Necessita de aproximadamente 120 Kbytes de disco.

A última opção, Instalação Personalizada, é uma instalação mais delhada com a qual pode-se escolher quais componentes serão instalados.

Com essa opção poderá ser instalada algumas porções do sistema, dependendo das opções, assinaladas na tela, feita pela pessoa responsável pela instalação.

Assim, uma tela (igual a mostrada a seguir) com as possíveis opções irá aparecer, pondendo-se marcar com um "x" cada item que desejar instalar.



As três primeiras opções são idênticas às da tela do menu principal.

Com a quarta opção ativada, será instalado o diretório com os fontes de todos os programas do gerador e da instalação, consumindo cerca de 200 Kbytes de disco.

A opção cinco instala o diretório com os programas fontes dos protocolos de rede do sistema (120 Kbytes).

A opção Login e Logout instala o diretório com os arquivos fontes dos programas de *login* e *logout* do cliente com o servidor (10 Kbytes).

E a opção Exemplos instala vários subdiretórios com exemplos de desenvolvimento de aplicações distribuídas, utilizando o gerador, inclusive os exemplos que serão abordados neste manual (70 Kbytes).

Fazendo uma relação com a primeira opção do menu principal, Instalação Completa, é o mesmo que pedir Instalação Personalizada e assinalar a instalação de todos os componentes.

1.4- Considerações Finais

Nesta seção foi comentado um ponto importante: a instalação dos programas necessários à utilização do sistema GAPS e suas possíveis opções para geração dos procedimentos, para configuração dos ambientes do cliente e do servidor e para carregar os programas fontes e os exemplos de desenvolvimento de aplicações distribuídas.

É importante salientar que este programa de instalação é apenas uma ferramenta de apoio para copiar partes indispensáveis à necessidade do usuário, facilitando o trabalho de instalação.

Na próxima seção será mostrado a definição do módulo de especificação através de alguns exemplos.

SEÇÃO 2

LINGUAGEM DE INTERFACE

Esta seção é dedicada à primeira parte da geração dos procedimentos *stubs*: a definição da linguagem de interface.

Os detalhes da linguagem de especificação são mostrados aqui através de três exemplos concretos.

2.1- Definição das aplicações

O primeiro exemplo corresponde a um servidor de nomes bastante simples, o qual é composto por quatro serviços, listados a seguir:

CriaNameServer() - responsável por iniciar a base de dados do servidor;

CriaObjeto(nome,tipo,caract,endereco) - responsável por armazenar as infomações de um objeto no servidor;

RemoveObjeto(nome) - remove um objeto do servidor; e

LocalizaObjeto(nome,endereco) - responsável por retornar o endereco de localização do objeto.

O segundo é um servidor de processamento numérico, o qual tem o seguinte serviço:

SomaVet(Vetor1,Vetor2,Vetor3) - responsável pela soma do Vetor1 com o Vetor2 e retornar o resultado no Vetor3.

E por último um pequeno servidor de arquivos com dois comandos:

RDir(caminho,opcoes,tam_lista,lista) - mostra o diretório remoto; e

RCopy(tam_lista,lista,caminho) - copia um arquivo para o disco remoto.

Antes de se partir para criar os processos distribuídos, deve-se criá-los utilizando os procedimentos locais, para, então, separá-los.

Os processos cliente e servidor de cada exemplo encontram-se no apêndice A.

2.2- Definição da linguagem de interface

Com os processos cliente e servidor criados, estes podem se separar, sem qualquer modificação, para serem executados em máquinas diferentes, e partir para a definição do módulo de especificação da linguagem de interface. A identificação dos módulos da linguagem de interface possuem uma extensão .LGS (Linguagem do Gerador de *Stubs*)

A linguagem de interface do gerador é semelhante à linguagem C com algumas características adicionais e algumas restrições.

Uma primeira característica adicionada é um tipo *string* o qual corresponde a um tipo *char* de tamanho 255, que não existe na linguagem C.

Outra característica adicional é a distinção feita entre parâmetros de entrada, de saída e de entrada e saída. Com isso, todos os parâmetros recebem uma declaração informando seu tipo: *in* (*default*) ou *out* ou *inout*, respectivamente.

Com relação às restrições, o gerador não aceita ponteiros, conseqüentemente, listas encadeadas, nem matrizes bidimensionais ou maiores.

Um ponto importante a ressaltar é que, na linguagem de especificação, não existem estruturas de atribuição, nem de repetição, nem de condição, apenas os protótipos dos procedimentos que serão executados remotamente e algumas definições do tipo *define*, constante, enumeração e estrutura.

A partir dos procedimentos criados localmente, podem-se copiar os protótipos dos procedimentos que serão executados remotamente para o módulo de entrada do gerador e alterá-los de acordo com as mudanças na gramática da linguagem. De preferêcia deve-se criar um subdiretório, para cada exemplo, para colocar os programas cliente e servidor e o módulo da linguagem de interface, isso por questão de organização, pois neste subdiretório serão, também, gerados os módulos com os procedimentos *stubs*.

Assim, o primeiro exemplo possuirá o seguinte módulo de definição (colocado no arquivo NAMESERV.LGS):

```
definition NAMESERV
{
  int CriaNameServer();
  int CriaObjeto(string nome, string tipo, string caract, string endereco);
  int RemoveObjeto(string nome, out string endereco);
}
```

O exemplo do servidor numérico terá o seguinte módulo (colocado no arquivo SOMAVET.LGS):

```
definition SOMAVETOR
{
  struct REGISTRO
    { int VETOR[100];
      int ELEMENTOS; };
  int SomaVetor(struct REGISTRO x, struct REGISTRO y,
                out struct REGISTRO z);
}
```

E o módulo do servidor de arquivos será o seguinte (colocado no arquivo FILESERV.LGS):

```
definition FILESERV
{
  int RDir(char CAMINHO[250], char OPCOES[2],
           out int TAM_LISTA, out char LISTA[2000]);
  int Rcopy(char ARQ[250], char CAMINHO[250],
           int TAM_LISTA, char LISTA[2000])
}
```

2.3 Considerações Finais

Nesta seção foi caracterizada a linguagem do gerador e foram definidas as linguagens de interface de três exemplos: de um servidor de nomes, de um servidor numérico e de um servidor de arquivos.

Na próxima seção será mostrada a geração dos procedimentos stubs desses exemplos.

SEÇÃO 3

GERAÇÃO DOS PROCEDIMENTOS *STUBS*

Nesta seção serão vistos os passos necessários para a criação dos procedimentos *stubs*, utilizando o sistema GAPS, dando continuidade aos exemplos da seção anterior.

Para criar os procedimentos através do gerador, precisa-se ter instalado no computador, no mínimo, a opção Linha de Comando. Com essa opção, é instalado o programa do gerador que pode ser executado a partir do *prompt* do DOS.

3.1- Executando o GAPS

Após a definição da linguagem de interface, feita na seção anterior, pode-se submetê-la ao gerador através da linha de comando, como mostrado abaixo:

C:\NAMESERV> gaps NAMESERV <enter> para o primeiro exemplo;

C:\NUMERICO> gaps SOMAVET <enter> para o segundo exemplo; e

C:\FILESERV> gaps FILESERV <enter> para o último exemplo.

Com isso, o gerador (programa GAPS) irá criar cinco módulos para cada exemplo, descritos a seguir:

XXXX_CLT.C - um arquivo fonte com todos os *stubs* do cliente e alguns procedimentos adicionais;

XXXX_SRV.C - um arquivo fonte com todos os *stubs* do servidor e alguns procedimentos adicionais;

XXXX_HDR.H - um arquivo *header* com as estruturas que são comuns aos procedimentos *stubs* do cliente e do servidor;

XXXX_CLT.MAK - um módulo *make* para a compilação do processo cliente; e

XXXX_SRV.MAK - um módulo make para a compilação do processo servidor,

onde XXXX são os quatro primeiros dígitos do nome do módulo de definição, ou seja, o nome atribuído à linguagem de interface, através da cláusula *definition*.

Esses arquivos *make* (.MAK) são produzidos, pelo gerador, para facilitar ainda mais o trabalho do programador, sendo necessário, apenas submeter o módulo ao compilador.

Observe-se, no entanto, que no módulo *make* do cliente, precisa-se substituir o campo <CLIENTE> pelo nome do processo cliente.

Executando o GAPS com a linguagem de especificação de cada exemplo, foram gerados os seguintes módulos:

Servidor de Nome:

NAME_CLT.C

NAME_SRV.C

NAME_HDR.H

NAME_CLT.MAK

NAME_SRV.MAK

Servidor Numérico:

SOMA_CLT.C

SOMA_SRV.C

SOMA_HDR.H

SOMA_CLT.MAK

SOMA_SRV.MAK

Servidor de Arquivos:

FILE_CLT.C

FILE_SRV.C

FILE_HDR.H

FILE_CLT.MAK

FILE_SRV.MAK

O conteúdo desses módulos gerados estão no apêndice B.

3.2 Considerações Finais

Nesta seção foi vista a segunda parte do desenvolvimento de aplicações distribuídas: a geração dos procedimentos *stubs*, utilizando o gerador.

Foram definidos, também, outros módulos produzidos pelo gerador.

A próxima seção entrará em detalhes sobre a execução dos processos cliente e servidor.

SEÇÃO 4

EXECUÇÃO DAS APLICAÇÕES

Nesta seção serão vistos os passos necessários para a compilação e execução do cliente e do servidor.

Para compilar os processos, tanto o cliente quanto o servidor, precisa-se instalar, no mínimo, a opção Linha de Comando. Para a execução do cliente, precisa-se intalar o Ambiente do Cliente e para a execução do servidor, o Ambiente do Servidor.

4.1- Compilando os Processos

Com todos os módulos gerados, pode-se partir para a compilação dos processos distribuídos.

Para compilar, precisa-se do programa *make* do turbo-C e, então, digitar o seguinte comando:

C:\NAMESERV> make -fname_clt <enter> para compilar o processo cliente e

C:\NAMESERV> make -fname_srv <enter> para compilar o processo servidor para o primeiro exemplo.

C:\NUMERICO> make -fsoma_clt <enter> para compilar o processo cliente e

C:\NUMERICO> make -fsoma_srv <enter> para compilar o processo servidor refe-rentes ao segundo exemplo.

C:\FILESERV> make -ffile_clt <enter> para compilar o processo cliente e

C:\FILESERV> make -ffile_srv <enter> para compilar o processo servidor, no caso do último exemplo.

Com isso, são produzidos os programas executáveis.

Em cada exemplo, obtêm-se:

Servidor de Nomes:

cria.exe, criaobj.exe, rmobj.exe, seekobj.exe e servicos.exe;

Servidor Numérico:

cliente.exe e servicos.exe; e

Servidor de Arquivos:

rdir.exe, rcopy.exe e servicos.exe.

4.2- Executando os Processos

Criados os programas executáveis, esses podem ser transportados para os equipamentos e executados independentemente.

Vale Lembrar que os equipamentos devem estar configurados de acordo com a opção Ambiente do Cliente ou Ambiente do Servidor, dependendo do que for executar, o cliente ou o servidor, respectivamente.

Antes de executar os processos, entretanto, deve-se ativar o *packet driver*, disponibilizando o meio de comunicação para os processos. O *packet driver* é ativado através de um dos módulos de execução: o NE1000 ou NE2000, dependendo do tipo de placa. Esse protocolo é carregado na instalação Ambiente do Cliente, Ambiente do Servidor ou Configuração Personalizada.

É importante lembrar que a pessoa que iniciará os processos cliente ou servidor deve-se estar familiarizado com os tipos de placas que está usando, bem como, conhecer seu endereço e seu número de interrupção.

Com essas informações, pode-se executar o *driver* de rede da seguinte forma:

C:\> *driver int_soft int_hard end*

onde, *driver* é o NE1000 ou NE2000, *int_soft* é número da interrupçào de software. *int_hard* é o número de interrupção de hardware e *end* é o endereço utilizado.

Como exemplo, caso esteja utilizando uma placa NE1000, configurada para 0x60 como a interrupção de *software*, 0x03 a interrupção de *hardware* e 0x300 o endereço de entrada/saída, o comando será:

C:\> NE1000 0x60 0x03 0x300 <enter>

A partir daí, pode-se executar o processo cliente ou servidor, como a seguir:

C:\> servicos <enter> para o processo servidor e

C:\> rdir \dos /p <enter> para o processo cliente.

É importante ressaltar que o processo servidor deve ser executado primeiro, pois, caso contrário, o processo cliente não teria resultado, visto que os serviços não estariam disponíveis.

4.3 Considerações Finais

Nesta seção foi completada o desenvolvimento de aplicações distribuídas, utilizando o gerador de stubs; foram mostradas a compilação e execução dos processos cliente e servidor.

BIBLIOGRAFIA

Clarkson University, "**The Packet Driver Specification**", User Documentation for the Packet Driver Collection, version 1.09, U.S., 1989.

Stankelly-Bootle, "**Dominando o Turbo C**", *Ed. Ciência Moderna*, 2a edição, 1989.

Turbo C, "**User's Guide**", *Borland International, Inc.*, versão 2.0.

Turbo C, "**Programmer's Guide**", *Borland International, Inc.*, versão 2.0.

Veloso, P., Santos, C. dos, Azeredo, P. e Furtado, A., "**Estruturas de Dados**", *Ed. Campus*, 4a edição, 1986.

APÊNDICE A

Exemplos de Processos Cliente e Servidor

Neste apêndice estão listadas os arquivos fontes dos processos cliente e servidor dos três exemplos deste trabalho, implementadas em turbo-C:

Servidor de Nomes

1- Processos Clientes

```
#include <stdio.h>
#include <ctype.h>
/* programa principal */
int main()
 {
  if (! CriaNameServer())
    printf("Servidor inicializado.\n");
   else
    printf("Servidor ja inicializado.\n");
 }

#include <stdio.h>
#include <ctype.h>
/* programa principal */
void main (int argc, char *argv[])
 {
  int resp;
  if (argc != 5)
   {
    printf("Erro no numero de parametros\n");
    printf("   nome tipo caracteristica endereco\n");
   exit(0);
   }
  resp = CriaObjeto(argv[1],argv[2],argv[3],argv[4]);
  if (! resp)
    printf("Objeto %s foi inserido.\n",argv[1]);
   else
    printf("Objeto %s ja foi inserido.\n",argv[1]);
```

```
 }

#include <stdio.h>
#include <ctype.h>
/* programa principal */
void main (int argc, char *argv[])
 {
  if (argc != 2)
   {
    printf("Erro no numero de parametros\n");
    printf("   nome_do_objeto\n");
    exit(0);
   }
  if (! RemoveObjeto(argv[1]))
    printf("Objeto %s foi removido.\n",argv[1]);
   else
    printf("Objeto %s nao foi encontrado.\n",argv[1]);
 }

#include <stdio.h>
#include <ctype.h>
/* programa principal */
void main (int argc, char *argv[])
 {
  char endereco[15];
  if (argc != 2)
   {
    printf("Erro no numero de parametros\n");
    printf("   nome_do_objeto\n");
    exit(0);
   }
  if (! LocalizaObjeto(argv[1],endereco))
   {
    printf("Objeto %s foi encontrado.\n",argv[1]);
    printf("endereco: %s.\n",endereco);
   }
   else
    printf("Objeto %s nao foi encontrado.\n",argv[1]);
 }
```

2- Processo Servidor

```
#define MaxObj 100  /* numero maximo de objetos na base de dados */
#include "stdio.h"
#include "string.h"
#include "stdlib.h"
#include "ctype.h"
struct OBJETO { char NOME[11];
                char TIPO[21];
                char CARACT[21];
                char ENDERECO[17];
              };
struct OBJETO DataBase[MaxObj]; /* base de dados */
```

```
int  NumObj = 0;              /* numero de objetos na base de dados */
/* inicializa base de dados */
int CriaNameServer()
 {
  static int I = 1;
  if (I == 1)
   {
    I = 0;
   }
   else
   {
    return 1;
   }
  NumObj = 0;
  return 0;
 }
/* cria objeto */
int CriaObjeto(char nome[], char tipo[], char caract[],
            char endereco[])
 {
  int i;
  for (i=0; i<NumObj; i++)
    if (! strcmp(nome,DataBase[i].NOME))
      return 1;
  strcpy(DataBase[NumObj].NOME,nome);
  strcpy(DataBase[NumObj].TIPO,tipo);
  strcpy(DataBase[NumObj].CARACT,caract);
  strcpy(DataBase[NumObj].ENDERECO,endereco);
  NumObj++;
  printf("objeto inserido: %s\n",nome);
  return 0;
 }
/* remove objeto */
int RemoveObjeto(char nome[])
 {
  int i;
  for (i=0; i<NumObj; i++)
    if (! strcmp(nome,DataBase[i].NOME))
      {
       NumObj--;
       strcpy(DataBase[i].NOME,DataBase[NumObj].NOME);
       strcpy(DataBase[i].TIPO,DataBase[NumObj].TIPO);
       strcpy(DataBase[i].CARACT,DataBase[NumObj].CARACT);
       strcpy(DataBase[i].ENDERECO,DataBase[NumObj].ENDERECO);
       printf("objeto removido: %s\n",nome);
       return 0;
      }
  return 1;
 }
/* localiza objeto */
int LocalizaObjeto(char nome[], char endereco[])
 {
  int i;
  for (i=0; i<NumObj; i++)
    if (! strcmp(nome,DataBase[i].NOME))
      {
```

```
        strcpy(endereco,DataBase[i].ENDERECO);
        return 0;
       }
  return 1;
 }
```

Servidor Numérico

1- Processos Clientes

```
/* soma  duas matrizes */
main()
{
 struct REGISTRO
  {
   int VETOR[100];
   int ELEMENTOS;
  };
 struct REGISTRO a,b,c;
 int i, RESP;
 a.ELEMENTOS = 100;
 b.ELEMENTOS = 100;
/* c = (struct REGISTRO *)malloc(sizeof(struct REGISTRO)); */
 clrscr();
 for (i=0; i<a.ELEMENTOS; i++)
    a.VETOR[i] = i;
 for (i=0; i<b.ELEMENTOS; i++)
    b.VETOR[i] = 2*i;
 printf("Elementos do Vetor 1: ");
 for (i=0; i<a.ELEMENTOS; i++)
   printf("%3d ",a.VETOR[i]);
 printf("\n");
 printf("Elementos do Vetor 2: ");
 for (i=0; i<b.ELEMENTOS; i++)
   printf("%3d ",b.VETOR[i]);
 printf("\n");
 RESP = SomaVet(a,b,&c);
 if (RESP)
   printf("Numero de elementos diferentes.\n");
  else {
   printf("Elementos do Vetor R: ");
   for (i=0; i<c.ELEMENTOS; i++)
      printf("%3d ",c.VETOR[i]);
   printf("\n");
  }
}
```

2- Processo Servidor

```
/* soma duas matrizes */
 struct REGISTRO
```

```
{
 int VETOR[100];
 int ELEMENTOS;
};
int SomaVet(struct REGISTRO x,struct REGISTRO y,struct REGISTRO *z)
{
 int i;
 printf("Elementos do Vetor 1: ");
 for (i=0; i<x.ELEMENTOS; i++)
    printf("%3d ",x.VETOR[i]);
 printf("\n");
 printf("Elementos do Vetor 2: ");
 for (i=0; i<y.ELEMENTOS; i++)
    printf("%3d ",y.VETOR[i]);
 printf("\n");
 if (x.ELEMENTOS != y.ELEMENTOS) return (1);
 for (i=0; i<x.ELEMENTOS; i++)
    z->VETOR[i] = x.VETOR[i] + y.VETOR[i];
 z->ELEMENTOS = x.ELEMENTOS;
 printf("Elementos do Vetor R: ");
 for (i=0; i<z->ELEMENTOS; i++)
    printf("%3d ",z->VETOR[i]);
 printf("\n");
 return(0);
}
```

Servidor de Arquivos

1- Processos Clientes

```
/* cliente: DIR remoto */
#include <stdio.h>
#include <ctype.h>
/* programa principal */
void main (int argc, char *argv[])
 {
  char *LISTA;
  int TAM_LISTA;
  int MAX_LISTA;
  int i;
  MAX_LISTA = 1000*sizeof(char);
  LISTA = (char *)malloc(MAX_LISTA);
  for (i=0; i<MAX_LISTA; i++)
     *(LISTA+i) = ' ';
  printf("\n Remote Dir versao 1.0. LaSD/ICMSC/USP.");
  /* se so ha um parametro e se eh /? (ajuda) */
  if ((argc == 2) && (! strcmp(argv[1],"/?")))
   {
    printf("\n Sintaxe: rdir [caminho] [opcoes]\n\n");
    printf("    [opcoes]:  /p - pause a cada preenchimento da tela\n");
    printf("               /w - listagem horizontal\n");
    exit(0);
```

```
    }
    else
     /* se nao ha caminho nem opcao, ou seja, eh o diretorio corrente */
     if (argc == 1) {
       if (ListaDiretorio("","",&TAM_LISTA,LISTA)) {
          printf("Erro na procura do diretorio.\n");
          exit(0); } }
      else
       /* se nao ha opcao */
       if (argc == 2) {
          if (ListaDiretorio(argv[1],"",&TAM_LISTA,LISTA)) {
             printf("Erro na procura do diretorio.\n");
             exit(0); }}
        else
          if (argc == 3) {
             if (ListaDiretorio(argv[1],argv[2],&TAM_LISTA,LISTA)) {
                printf("Erro na procura do diretorio.\n");
                exit(0); }}
           else
             /* se numero de parametros eh maior que 4 */
             if (argc > 3)
              {
               printf("\n Numero de parametros excedido.\n\n");
               printf("\n Sintaxe: rdir [caminho] [opcoes]\n\n");
               printf("    [opcoes]:  /p - pause a cada preenchimento da tela\n");
               printf("               /w - listagem horizontal\n");
               exit(0);
              }
             else
              /* caso contrario */
              {
               printf("\n Parametros incorretos.\n\n");
               printf("\n Sintaxe: rdir [caminho] [opcoes]\n\n");
               printf("    [opcoes]:  /p - pause a cada preenchimento da tela\n");
               printf("               /w - listagem horizontal\n");
               exit(0);
              }
   for (i=0; i<TAM_LISTA; i++)
      printf("%c",*(LISTA+i));
   exit(0);
  }

/* cliente: COPY remoto */
 #include <stdio.h>
 #include <ctype.h>
 #define STRING    250
 #define LEN_LINHA  81
 FILE *ARQUIVO;
 /* le arquivo */
 int LeArquivo(char ARQ[STRING], int *TAM_LIST, char *LIST)
  {
   char LINHA[LEN_LINHA];
   int i = 0;
   int j = 0;
   if ((ARQUIVO=fopen(ARQ,"r")) == NULL)
     return(1);
```

```
  fgets(LINHA,LEN_LINHA,ARQUIVO);
  while (! feof(ARQUIVO))
   {
    for (i=0; LINHA[i] != '\n'; i++)
      {
       *(LIST+j) = LINHA[i];
       j++;
      };
    *(LIST+j) = '\n';
    j++;
    fgets(LINHA,LEN_LINHA,ARQUIVO);
   }
  fclose(ARQUIVO);
  *TAM_LIST = j;
  return(0);
 }
/* programa principal */
void main (int argc, char *argv[])
 {
  char *LISTA;
  int TAM_LISTA;
  int MAX_LISTA;
  int i;
  MAX_LISTA = 1000*sizeof(char);
  LISTA = (char *)malloc(MAX_LISTA);
  for (i=0; i<MAX_LISTA; i++)
     *(LISTA+i) = ' ';
  printf("\n Remote Copy versao 1.0. LaSD/ICMSC/USP.");
  /* se so ha um parametro e se eh /? (ajuda) */
  if ((argc == 2) && (! strcmp(argv[1],"/?")))
   {
    printf("\n Sintaxe: rcopy [arquivo] [caminho]\n\n");
    exit(0);
   }
   else
    if (argc == 3) {
      if (LeArquivo(argv[1],&TAM_LISTA,LISTA)) {
        printf("Arquivo nao encontrado.\n");
        exit(0); }
      if (CopiaArquivo(argv[1],argv[2],&TAM_LISTA,LISTA)) {
        printf("Diretorio nao encontrado.\n");
        exit(0); } }
     else
      /* caso contrario */
      {
       printf("\n Parametros incorretos.\n\n");
       printf("\n Sintaxe: rcopy [arquivo] [caminho]\n\n");
       exit(0);
      }
 }
```

2- Processo Servidor

```
/* servico de DIR remoto */
```

```
#include <stdio.h>
#include <ctype.h>
#define STRING    250
#define LEN_LINHA  81
/* lista diretorio */
int ListaDiretorio(char CAMINHO[STRING], char OPCOES[2],
                   int *TAM_LIST, char *LIST)
 {
  FILE *ARQUIVO;
  char COMANDO[STRING] = {"\0"};
  char LINHA[LEN_LINHA];
  int i = 0;
  int j = 0;
  printf("\n\nComando: rdir %s %s\n",CAMINHO,OPCOES);
  strcat(COMANDO,"dir ");
  strcat(COMANDO,CAMINHO);
  strcat(COMANDO," ");
  strcat(COMANDO,OPCOES);
  strcat(COMANDO," > temporar.io~");
  printf("COMANDO = %s\n",COMANDO);
  system(COMANDO);
  if ((ARQUIVO=fopen("temporar.io~","r")) == NULL)
    {
     printf("Erro na abertura do arquivo temporario");
     return(1);
    }
  fgets(LINHA,LEN_LINHA,ARQUIVO);
  while (! feof(ARQUIVO))
    {
     for (i=0; LINHA[i] != '\n'; i++)
       {
        *(LIST+j) = LINHA[i];
        j++;
       };
     *(LIST+j) = '\n';
     j++;
     fgets(LINHA,LEN_LINHA,ARQUIVO);
    }
  fclose(ARQUIVO);
  system("del temporar.io~");
  *TAM_LIST = j;
  return(0);
 }


/* servidor: COPY remoto */
#include <stdio.h>
#include <ctype.h>
#define STRING    250
#define LEN_LINHA  81
FILE *ARQUIVO;
int CopiaArquivo(char ARQ[STRING],
                 char CAMINHO[STRING],
                 int TAM_LIST, char LIST[2000])
 {
  char DEST[LEN_LINHA] = {"\0"};
```

```
int i;
strcpy(DEST,CAMINHO);
if (strchr(ARQ,':'))
  {for (i=2; i<STRING; i++)
      ARQ[i-2] = ARQ[i]; }
if (ARQ[0] != '\\')
   strcat(DEST,"\\\0");
strcat(DEST,ARQ);
if ((ARQUIVO=fopen(DEST,"w")) == NULL)
   return(1);
for (i=0;i<*TAM_LIST;i++)
    fprintf(ARQUIVO,"%c",*(LIST+i));
return(0);
}
```

APÊNDICE B

Código dos Módulos Gerados pelo Sistema GAPS

Neste apêndice estão listadas os arquivos gerados pelo GAPS dos três exemplos deste trabalho, implementadas em turbo-C:

Servidor de Nomes

1- NAME_HDR.H

```
/* header */
#define LEN_MAX_STR  256
/* definicao da variavel do tipo union */
union TIPO_PRE TAM_TIPO_PRE;
/* codigos de identificacao dos procedimentos (primitivas) */
#define PROC_01  0x02
#define PROC_02  0x03
#define PROC_03  0x04
#define PROC_04  0x05
#define PROC_05  0x06
/* prototipos dos procedimentos */
int CriaNameServer();
int CriaObjeto(char nome[],char tipo[],char caract[],char endereco[]);
int RemoveObjeto(char nome[]);
int LocalizaObjeto(char nome[],char endereco[]);
int ImprimeObjetos();
```

2- NAME_CLT.C

```
/* Client Stub */
#include "\gaps\rede\globais.h"
#include "\gaps\rede\union.h"
#include "NAME_hdr.h"
extern int GNumPrim;
void CompletaTabPri()
 {
  int I;
```

```
  /* carrega a tabela de primitivas */
  AtribueTabPrim(0,PROC_01,10,"2\0");
  AtribueTabPrim(1,PROC_02,10,"2\0");
  AtribueTabPrim(2,PROC_03,10,"2\0");
  AtribueTabPrim(3,PROC_04,10,"2\0");
  AtribueTabPrim(4,PROC_05,10,"2\0");
  GNumPrim = 5;
 }
void Inicializa()
 {
  static int PRIM_VEZ = 1;
  if (PRIM_VEZ)
   {
    PRIM_VEZ = 0;
    CarregaServidores();
    CarregaConfiguracao();
    CompletaTabPri();
   }
 }
int CriaNameServer()
 {
  int TAM_MSG = 0;
  char *SRV = "2\0";
  BYTE MENSAGEM[3000];
  union TIPO_PRE ARGUMENTO;
  int I_1=0, J_1=0;
  int RETORNO;
  Inicializa();
  ExecutarProcedimento(SRV,10,PROC_01,MENSAGEM,&TAM_MSG);
  DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
  RETORNO = ARGUMENTO.INT;
  return(RETORNO);
 }
int CriaObjeto(char nome[LEN_MAX_STR],char tipo[LEN_MAX_STR],
        char caract[LEN_MAX_STR],char endereco[LEN_MAX_STR])
 {
  int TAM_MSG = 0;
  char *SRV = "2\0";
  BYTE MENSAGEM[3000];
  union TIPO_PRE ARGUMENTO;
  int I_1=0, J_1=0;
  int RETORNO;
  Inicializa();
  strcpy(ARGUMENTO.STRING,nome);
  MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"string");
  strcpy(ARGUMENTO.STRING,tipo);
  MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"string");
  strcpy(ARGUMENTO.STRING,caract);
  MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"string");
  strcpy(ARGUMENTO.STRING,endereco);
  MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"string");
  ExecutarProcedimento(SRV,10,PROC_02,MENSAGEM,&TAM_MSG);
  DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
  RETORNO = ARGUMENTO.INT;
  return(RETORNO);
 }
```

```
int RemoveObjeto(char nome[LEN_MAX_STR])
 {
  int TAM_MSG = 0;
  char *SRV = "2\0";
  BYTE MENSAGEM[3000];
  union TIPO_PRE ARGUMENTO;
  int I_1=0, J_1=0;
  int RETORNO;
  Inicializa();
  strcpy(ARGUMENTO.STRING,nome);
  MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"string");
  ExecutarProcedimento(SRV,10,PROC_03,MENSAGEM,&TAM_MSG);
  DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
  RETORNO = ARGUMENTO.INT;
  return(RETORNO);
 }
int LocalizaObjeto(char nome[LEN_MAX_STR],char endereco[LEN_MAX_STR])
 {
  int TAM_MSG = 0;
  char *SRV = "2\0";
  BYTE MENSAGEM[3000];
  union TIPO_PRE ARGUMENTO;
  int I_1=0, J_1=0;
  int RETORNO;
  Inicializa();
  strcpy(ARGUMENTO.STRING,nome);
  MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"string");
  ExecutarProcedimento(SRV,10,PROC_04,MENSAGEM,&TAM_MSG);
  DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
  RETORNO = ARGUMENTO.INT;
  DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"string");
  strcpy(endereco,ARGUMENTO.STRING);
  return(RETORNO);
 }
int ImprimeObjetos()
 {
  int TAM_MSG = 0;
  char *SRV = "2\0";
  BYTE MENSAGEM[3000];
  union TIPO_PRE ARGUMENTO;
  int I_1=0, J_1=0;
  int RETORNO;
  inicializa();
  ExecutarProcedimento(SRV,10,PROC_05,MENSAGEM,&TAM_MSG);
  DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
  RETORNO = ARGUMENTO.INT;
  return(RETORNO);
 }
```

3- NAME_SRV.C

```
/* Server Stub */
#include "\gaps\rede\globais.h"
#include "\gaps\rede\union.h"
#include "NAME_hdr.h"
```

```
int Procedimento_01(int *TAM_MSGent, BYTE MENSAGEM[])
 {
  int TAM_MSG = 0;
  union TIPO_PRE ARGUMENTO;
  int RETORNO;
  /* argumentos de entrada */
  int I_1=0, J_1=0;
  TAM_MSG = *TAM_MSGent;
  /* malloc */
  RETORNO = CriaNameServer();
  ARGUMENTO.INT = RETORNO;
  MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
  *TAM_MSGent = TAM_MSG;
  /* free */
  return(0);
 }
int Procedimento_02(int *TAM_MSGent, BYTE MENSAGEM[])
 {
  int TAM_MSG = 0;
  union TIPO_PRE ARGUMENTO;
  int RETORNO;
  /* argumentos de entrada */
  char nome[LEN_MAX_STR];
  char tipo[LEN_MAX_STR];
  char caract[LEN_MAX_STR];
  char endereco[LEN_MAX_STR];
  int I_1=0, J_1=0;
  TAM_MSG = *TAM_MSGent;
  /* malloc */
  DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"string");
  strcpy(nome,ARGUMENTO.STRING);
  DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"string");
  strcpy(tipo,ARGUMENTO.STRING);
  DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"string");
  strcpy(caract,ARGUMENTO.STRING);
  DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"string");
  strcpy(endereco,ARGUMENTO.STRING);
  RETORNO = CriaObjeto(nome,tipo,caract,endereco);
  ARGUMENTO.INT = RETORNO;
  MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
  *TAM_MSGent = TAM_MSG;
  /* free */
  return(0);
 }
int Procedimento_03(int *TAM_MSGent, BYTE MENSAGEM[])
 {
  int TAM_MSG = 0;
  union TIPO_PRE ARGUMENTO;
  int RETORNO;
  /* argumentos de entrada */
  char nome[LEN_MAX_STR];
  int I_1=0, J_1=0;
  TAM_MSG = *TAM_MSGent;
  /* malloc */
  DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"string");
  strcpy(nome,ARGUMENTO.STRING);
```

```
 RETORNO = RemoveObjeto(nome);
 ARGUMENTO.INT = RETORNO;
 MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
 *TAM_MSGent = TAM_MSG;
 /* free */
 return(0);
}
int Procedimento_04(int *TAM_MSGent, BYTE MENSAGEM[])
{
 int TAM_MSG = 0;
 union TIPO_PRE ARGUMENTO;
 int RETORNO;
 /* argumentos de entrada */
 char nome[LEN_MAX_STR];
 char endereco[LEN_MAX_STR];
 int I_1=0, J_1=0;
 TAM_MSG = *TAM_MSGent;
 /* malloc */
 DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"string");
 strcpy(nome,ARGUMENTO.STRING);
 RETORNO = LocalizaObjeto(nome,endereco);
 ARGUMENTO.INT = RETORNO;
 MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
 strcpy(ARGUMENTO.STRING,endereco);
 MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"string");
 *TAM_MSGent = TAM_MSG;
 /* free */
 return(0);
}
int Procedimento_05(int *TAM_MSGent, BYTE MENSAGEM[])
{
 int TAM_MSG = 0;
 union TIPO_PRE ARGUMENTO;
 int RETORNO;
 /* argumentos de entrada */
 int I_1=0, J_1=0;
 TAM_MSG = *TAM_MSGent;
 /* malloc */
 RETORNO = ImprimeObjetos();
 ARGUMENTO.INT = RETORNO;
 MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
 *TAM_MSGent = TAM_MSG;
 /* free */
 return(0);
}
void IniciaVariaveis()
{
 AtribuePrimitiva(PROC_01,Procedimento_01);
 AtribuePrimitiva(PROC_02,Procedimento_02);
 AtribuePrimitiva(PROC_03,Procedimento_03);
 AtribuePrimitiva(PROC_04,Procedimento_04);
 AtribuePrimitiva(PROC_05,Procedimento_05);
}
void main()
{
 IniciaVariaveis();
```

```
  Servidor();
 }
```

4- NAME_CLT.MAK

```
#*********************************************************
#  MakeFile para o CLIENTE
#  Obs: coloque o nome do arquivo (.c) que contem o
#       programa principal em <cliente>
#*********************************************************
OBJECTS = cliente.obj ether.obj pkt_clt.obj spp_clt.obj lib_clt.obj NAME_clt.obj
CFLAGS = -c -ml -B
cliente.exe: $(OBJECTS)
  tcc -ml $(OBJECTS)
ether.obj: \gaps\rede\ether.c
  tcc $(CFLAGS) \gaps\rede\ether.c
pkt_clt.obj: \gaps\rede\pkt_clt.c
  tcc $(CFLAGS) \gaps\rede\pkt_clt.c
spp_clt.obj: \gaps\rede\spp_clt.c
  tcc $(CFLAGS) \gaps\rede\spp_clt.c
lib_clt.obj: \gaps\rede\lib_clt.c
  tcc $(CFLAGS) \gaps\rede\lib_clt.c
NAME_clt.obj: NAME_clt.c
  tcc $(CFLAGS) NAME_clt.c
cliente.obj: cliente.c
  tcc $(CFLAGS) cliente.c
```

5- NAME_SRV.MAK

```
#*********************************************************
#  MakeFile para o SERVIDOR
#*********************************************************
OBJECTS = servicos.obj ether.obj pkt_srv.obj spp_srv.obj lib_srv.obj servidor.obj
CFLAGS = -c -ml -B
servicos.exe: $(OBJECTS)
  tcc -ml $(OBJECTS)
ether.obj: \gaps\rede\ether.c
  tcc $(CFLAGS) \gaps\rede\ether.c
pkt_srv.obj: \gaps\rede\pkt_srv.c
  tcc $(CFLAGS) \gaps\rede\pkt_srv.c
spp_srv.obj: \gaps\rede\spp_srv.c
  tcc $(CFLAGS) \gaps\rede\spp_srv.c
lib_srv.obj: \gaps\rede\lib_srv.c
  tcc $(CFLAGS) \gaps\rede\lib_srv.c
servidor.obj: servicos.c
  tcc $(CFLAGS) -oservidor servicos.c
servicos.obj: NAME_srv.c
  tcc $(CFLAGS) -oservicos NAME_srv.c
```

Servidor Numérico

1- SOMA_HDR.H

```
/* header */
#define LEN_MAX_STR  256
/* definicao da variavel do tipo union */
union TIPO_PRE TAM_TIPO_PRE;
/* codigos de identificacao dos procedimentos (primitivas) */
#define PROC_01  0x02
/* estruturas */
struct REGISTRO { int  VETOR[100];
                  int  ELEMENTOS; };
/* prototipos dos procedimentos */
int SomaVet(struct REGISTRO x,struct REGISTRO y,struct REGISTRO *z);
```

2- SOMA_CLT.C

```
/* Client Stub */
#include "\gaps\rede\globais.h"
#include "\gaps\rede\union.h"
#include "veto_hdr.h"
extern int GNumPrim;
void CompletaTabPri()
 {
  int I;
  /* carrega a tabela de primitivas */
  AtribueTabPrim(0,PROC_01,10,"2\0");
  GNumPrim = 1;
 }
void Inicializa()
 {
  static int PRIM_VEZ = 1;
  if (PRIM_VEZ)
   {
    PRIM_VEZ = 0;
    CarregaServidores();
    CarregaConfiguracao();
    CompletaTabPri();
   }
 }
int SomaVet(struct REGISTRO x,struct REGISTRO y,struct REGISTRO *z)
 {
  int TAM_MSG = 0;
  char *SRV = "2\0";
  BYTE MENSAGEM[3000];
  union TIPO_PRE ARGUMENTO;
  int I_1=0, J_1=0;
  int RETORNO;
  Inicializa();
  for (J_1=0; J_1<100; J_1++)
    {
     ARGUMENTO.INT = x.VETOR[J_1];
     MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
    };
  ARGUMENTO.INT = x.ELEMENTOS;
  MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
  for (J_1=0; J_1<100; J_1++)
```

```
  {
   ARGUMENTO.INT = y.VETOR[J_1];
   MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
  };
 ARGUMENTO.INT = y.ELEMENTOS;
 MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
 ExecutarProcedimento(SRV,10,PROC_01,MENSAGEM,&TAM_MSG);
 DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
 RETORNO = ARGUMENTO.INT;
 for (J_1=0; J_1<100; J_1++)
   {
    DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
    z->VETOR[J_1] = ARGUMENTO.INT;
   };
 DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
 z->ELEMENTOS = ARGUMENTO.INT;
 return(RETORNO);
}
```

3- SOMA_SRV.C

```
/* Server Stub */
 #include "\gaps\rede\globais.h"
 #include "\gaps\rede\union.h"
 #include "veto_hdr.h"
 int Procedimento_01(int *TAM_MSGent, BYTE MENSAGEM[])
  {
   int TAM_MSG = 0;
   union TIPO_PRE ARGUMENTO;
   int RETORNO;
   /* argumentos de entrada */
   struct REGISTRO x;
   struct REGISTRO y;
   struct REGISTRO *z;
   int I_1=0, J_1=0;
   TAM_MSG = *TAM_MSGent;
   /* malloc */
   z = (struct REGISTRO *)malloc(sizeof(struct REGISTRO));
   for (J_1=0; J_1<100; J_1++)
     {
      DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
      x.VETOR[J_1] = ARGUMENTO.INT;
     };
   DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
   x.ELEMENTOS = ARGUMENTO.INT;
   for (J_1=0; J_1<100; J_1++)
     {
      DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
      y.VETOR[J_1] = ARGUMENTO.INT;
     };
   DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
   y.ELEMENTOS = ARGUMENTO.INT;
   RETORNO = SomaVet(x,y,z);
   ARGUMENTO.INT = RETORNO;
   MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
```

```
  for (J_1=0; J_1<100; J_1++)
    {
     ARGUMENTO.INT = z->VETOR[J_1];
     MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
    };
  ARGUMENTO.INT = z->ELEMENTOS;
  MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
  *TAM_MSGent = TAM_MSG;
  /* free */
  free(z);
  return(0);
 }
void IniciaVariaveis()
 {
  AtribuePrimitiva(PROC_01,Procedimento_01);
 }
void main()
 {
  IniciaVariaveis();
  Servidor();
 }
```

4- SOMA_CLT.MAK

```
#*************************************************************
#  MakeFile para o CLIENTE
#  Obs: coloque o nome do arquivo (.c) que contem o
#       programa principal em <cliente>
#*************************************************************
OBJECTS = cliente.obj ether.obj pkt_clt.obj spp_clt.obj lib_clt.obj veto_clt.obj
CFLAGS = -c -ml -B
cliente.exe: $(OBJECTS)
  tcc -ml $(OBJECTS)
ether.obj: \gaps\rede\ether.c
  tcc $(CFLAGS) \gaps\rede\ether.c
pkt_clt.obj: \gaps\rede\pkt_clt.c
  tcc $(CFLAGS) \gaps\rede\pkt_clt.c
spp_clt.obj: \gaps\rede\spp_clt.c
  tcc $(CFLAGS) \gaps\rede\spp_clt.c
lib_clt.obj: \gaps\rede\lib_clt.c
  tcc $(CFLAGS) \gaps\rede\lib_clt.c
veto_clt.obj: veto_clt.c
  tcc $(CFLAGS) veto_clt.c
cliente.obj: cliente.c
  tcc $(CFLAGS) cliente.c
```

5- SOMA_SRV.MAK

```
#*************************************************************
#
#  MakeFile para o SERVIDOR
#*************************************************************
OBJECTS = servicos.obj ether.obj pkt_srv.obj spp_srv.obj lib_srv.obj servidor.obj
CFLAGS = -c -ml -B
servicos.exe: $(OBJECTS)
```

```
  tcc -ml $(OBJECTS)
ether.obj: \gaps\rede\ether.c
  tcc $(CFLAGS) \gaps\rede\ether.c
pkt_srv.obj: \gaps\rede\pkt_srv.c
  tcc $(CFLAGS) \gaps\rede\pkt_srv.c
spp_srv.obj: \gaps\rede\spp_srv.c
  tcc $(CFLAGS) \gaps\rede\spp_srv.c
lib_srv.obj: \gaps\rede\lib_srv.c
  tcc $(CFLAGS) \gaps\rede\lib_srv.c
servidor.obj: servicos.c
  tcc $(CFLAGS) -oservidor servicos.c
servicos.obj: veto_srv.c
  tcc $(CFLAGS) -oservicos veto_srv.c
```

Servidor de Arquivos

1- FILE_HDR.H

```
/* header */
 #define LEN_MAX_STR  256
 /* definicao da variavel do tipo union */
 union TIPO_PRE TAM_TIPO_PRE;
 /* codigos de identificacao dos procedimentos (primitivas) */
 #define PROC_01  0x02
 #define PROC_02  0x03
 /* prototipos dos procedimentos */
 int RDir(char CAMINHO[],char OPCOES[],int *TAM_LIST,char LIST[]);
 int RCopy(char ARQ[],char CAMINHO[],int TAM_LIST,char LIST[]);
```

2- FILE_CLT.C

```
/* Client Stub */
 #include "\gaps\rede\globais.h"
 #include "\gaps\rede\union.h"
 #include "FILE_hdr.h"
 extern int GNumPrim;
 void CompletaTabPri()
  {
   int i;
   /* carrega a tabela de primitivas */
   AtribueTabPrim(0,PROC_01,10,"2\0");
   AtribueTabPrim(1,PROC_02,10,"2\0");
   GNumPrim = 2;
  }
 void Inicializa()
  {
   static int PRIM_VEZ = 1;
   if (PRIM_VEZ)
    {
     PRIM_VEZ = 0;
     CarregaServidores();
     CarregaConfiguracao();
```

```
    CompletaTabPri();
   }
 }
int RDir(char CAMINHO[250],char OPCOES[2],int *TAM_LIST,char LIST[2000])
 {
  int TAM_MSG = 0;
  char *SRV = "2\0";
  BYTE MENSAGEM[3000];
  union TIPO_PRE ARGUMENTO;
  int I_1=0, J_1=0;
  int RETORNO;
  Inicializa();
  for (I_1=0; I_1<250; I_1++)
    {
     ARGUMENTO.CHAR = CAMINHO[I_1];
     MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"char");
    };
  for (I_1=0; I_1<2; I_1++)
    {
     ARGUMENTO.CHAR = OPCOES[I_1];
     MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"char");
    };
  ExecutarProcedimento(SRV,10,PROC_01,MENSAGEM,&TAM_MSG);
  DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
  RETORNO = ARGUMENTO.INT;
  DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
  *TAM_LIST = ARGUMENTO.INT;
  for (I_1=0; I_1<2000; I_1++)
    {
     DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"char");
     LIST[I_1] = ARGUMENTO.CHAR;
    };
  return(RETORNO);
 }
int RCopy(char ARQ[250],char CAMINHO[250],int TAM_LIST,char LIST[2000])
 {
  int TAM_MSG = 0;
  char *SRV = "2\0";
  BYTE MENSAGEM[3000];
  union TIPO_PRE ARGUMENTO;
  int I_1=0, J_1=0;
  int RETORNO;
  inicializa();
  for (I_1=0; I_1<250; I_1++)
    {
     ARGUMENTO.CHAR = ARQ[I_1];
     MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"char");
    };
  for (I_1=0; I_1<250; I_1++)
    {
     ARGUMENTO.CHAR = CAMINHO[I_1];
     MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"char");
    };
  ARGUMENTO.INT = TAM_LIST;
  MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
  for (I_1=0; I_1<2000; I_1++)
```

```
    {
     ARGUMENTO.CHAR = LIST[I_1];
     MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"char");
    };
   ExecutarProcedimento(SRV,10,PROC_02,MENSAGEM,&TAM_MSG);
   DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
   RETORNO = ARGUMENTO.INT;
   return(RETORNO);
  }
```

3- FILE_SRV.C

```
/* Server Stub */
 #include "\gaps\rede\globais.h"
 #include "\gaps\rede\union.h"
 #include "FILE_hdr.h"
 int Procedimento_01(int *TAM_MSGent, BYTE MENSAGEM[])
  {
   int TAM_MSG = 0;
   union TIPO_PRE ARGUMENTO;
   int RETORNO;
   /* argumentos de entrada */
   char CAMINHO[250];
   char OPCOES[2];
   int *TAM_LIST,
   char LIST[2000];
   int I_1=0, J_1=0;
   TAM_MSG = *TAM_MSGent;
   /* malloc */
   TAM_LIST = (int *)malloc(sizeof(int));
   for (I_1=0; I_1<250; I_1++)
     {
      DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"char");
      CAMINHO[I_1] = ARGUMENTO.CHAR;
     };
   for (I_1=0; I_1<2; I_1++)
     {
      DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"char");
      OPCOES[I_1] = ARGUMENTO.CHAR;
     };
   RETORNO = RDir(CAMINHO,OPCOES,TAM_LIST,LIST);
   ARGUMENTO.INT = RETORNO;
   MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
   ARGUMENTO.INT = *TAM_LIST;
   MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
   for (I_1=0; I_1<2000; I_1++)
     {
      ARGUMENTO.CHAR = LIST[I_1];
      MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"char");
     };
   *TAM_MSGent = TAM_MSG;
   /* free */
   free(TAM_LIST);
   return(0);
  }
```

```
int Procedimento_02(int *TAM_MSGent, BYTE MENSAGEM[])
 {
  int TAM_MSG = 0;
  union TIPO_PRE ARGUMENTO;
  int RETORNO;
  /* argumentos de entrada */
  char ARQ[250];
  char CAMINHO[250];
  int TAM_LIST;
  char LIST[2000];
  int I_1=0, J_1=0;
  TAM_MSG = *TAM_MSGent;
  /* malloc */
  for (I_1=0; I_1<250; I_1++)
    {
     DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"char");
     ARQ[I_1] = ARGUMENTO.CHAR;
    };
  for (I_1=0; I_1<250; I_1++)
    {
     DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"char");
     CAMINHO[I_1] = ARGUMENTO.CHAR;
    };
  DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
  TAM_LIST = ARGUMENTO.INT;
  for (I_1=0; I_1<2000; I_1++)
    {
     DesmontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"char");
     LIST[I_1] = ARGUMENTO.CHAR;
    };
  RETORNO = RCopy(ARQ,CAMINHO,TAM_LIST,LIST);
  ARGUMENTO.INT = RETORNO;
  MontarMensagem(MENSAGEM,&TAM_MSG,&ARGUMENTO,"int");
  *TAM_MSGent = TAM_MSG;
  /* free */
  return(0);
 }
void IniciaVariaveis()
 {
  AtribuePrimitiva(PROC_01,Procedimento_01);
  AtribuePrimitiva(PROC_02,Procedimento_02);
 }
void main()
 {
  IniciaVariaveis();
  Servidor();
 }
```

4- FILE_CLT.MAK

```
#*********************************************************
#  MakeFile para o CLIENTE
#  Obs: coloque o nome do arquivo (.c) que contem o
#       programa principal em <cliente>
#*********************************************************
```

```
OBJECTS = cliente.obj ether.obj pkt_clt.obj spp_clt.obj lib_clt.obj FILE_clt.obj
CFLAGS = -c -ml -B
cliente.exe: $(OBJECTS)
  tcc -ml $(OBJECTS)
ether.obj: \gaps\rede\ether.c
  tcc $(CFLAGS) \gaps\rede\ether.c
pkt_clt.obj: \gaps\rede\pkt_clt.c
  tcc $(CFLAGS) \gaps\rede\pkt_clt.c
spp_clt.obj: \gaps\rede\spp_clt.c
  tcc $(CFLAGS) \gaps\rede\spp_clt.c
lib_clt.obj: \gaps\rede\lib_clt.c
  tcc $(CFLAGS) \gaps\rede\lib_clt.c
FILE_clt.obj: FILE_clt.c
  tcc $(CFLAGS) FILE_clt.c
cliente.obj: cliente.c
  tcc $(CFLAGS) cliente.c
```

5- FILE_SRV.MAK

```
#*************************************************************
#  MakeFile para o SERVIDOR
#*************************************************************
OBJECTS = servicos.obj ether.obj pkt_srv.obj spp_srv.obj lib_srv.obj servidor.obj
CFLAGS = -c -ml -B
servicos.exe: $(OBJECTS)
  tcc -ml $(OBJECTS)
ether.obj: \gaps\rede\ether.c
  tcc $(CFLAGS) \gaps\rede\ether.c
pkt_srv.obj: \gaps\rede\pkt_srv.c
  tcc $(CFLAGS) \gaps\rede\pkt_srv.c
spp_srv.obj: \gaps\rede\spp_srv.c
  tcc $(CFLAGS) \gaps\rede\spp_srv.c
lib_srv.obj: \gaps\rede\lib_srv.c
  tcc $(CFLAGS) \gaps\rede\lib_srv.c
servidor.obj: servicos.c
  tcc $(CFLAGS) -oservidor servicos.c
servicos.obj: FILE_srv.c
  tcc $(CFLAGS) -oservicos FILE_srv.c
```