Capítulo 6

Introdução aos Sistemas Operacionais de Rede

Sistemas operacionais de rede possuem as seguintes particularidades:

- estendem o alcance dos sistemas operacionais locais de modo a incluir dispositivos disponíveis em rede tais como: impressoras e diretórios de arquivos;
- fornecem a base computacional distribuída capaz de propiciar acesso transparente aos recursos disponíveis na rede;
- suportam a comunicação e coordenação das aplicações espalhadas pela rede (bases de dados, *groupware*, etc.);

Estas novas funcionalidades incorporadas aos sistemas operacionais demandam tecnologias de redes locais que permitam a movimentação de informação de forma rápida e confiável.

6.1 As Pilhas de Transporte

As pilhas de transporte correspondem aos protocolos de transporte como: TCP/IP, NetBIOS, IPX/SPX, DECnet, AppleTalk, OSI e SNA/APPN, protocolos estes que fornecem uma comunicação fim-a-fim confiável nas redes de longa distância e nas redes locais. Estes protocolos operam através da tecnologia de interconexão representada pelos roteadores, pontes e comportas (*gateways*).

A figura 6.1 mostra como a tecnologia de interconexão de redes é utilizada atualmente. As redes empresariais atuais são formadas de várias redes locais do tipo Ethernet e Token Ring, as quais são interconectadas com outras redes através da rede pública suportando X.25 e *Frame-Relay*. Complementarmente à evolução da tecnologia de interconexão de redes, os sistemas operacionais como OS/2, NetWare, Windows NT e UNIX têm se voltado para as redes. Eles têm introduzido características que permitem a inclusão nestes sistemas de pilhas de comunicação e de adaptadores de rede. A atividade dos programadores também tem sido facilitada pela disponibilidade de APIs que são independentes da pilha. A figura 6.2 ilustra como os sistemas operacionais atuais estão acomodando os múltiplos protocolos e APIs. Isto é feito através do envolvimento das pilhas de transporte por uma interface superior independente do transporte, e por uma interface lógica inferior associada aos dispositivos de rede.

O dispositivo lógico de rede fornece uma única interface para todos os adaptadores de rede. Esta interface permite que as empresas que desenvolvem pilhas de protocolos de transporte não necessitem

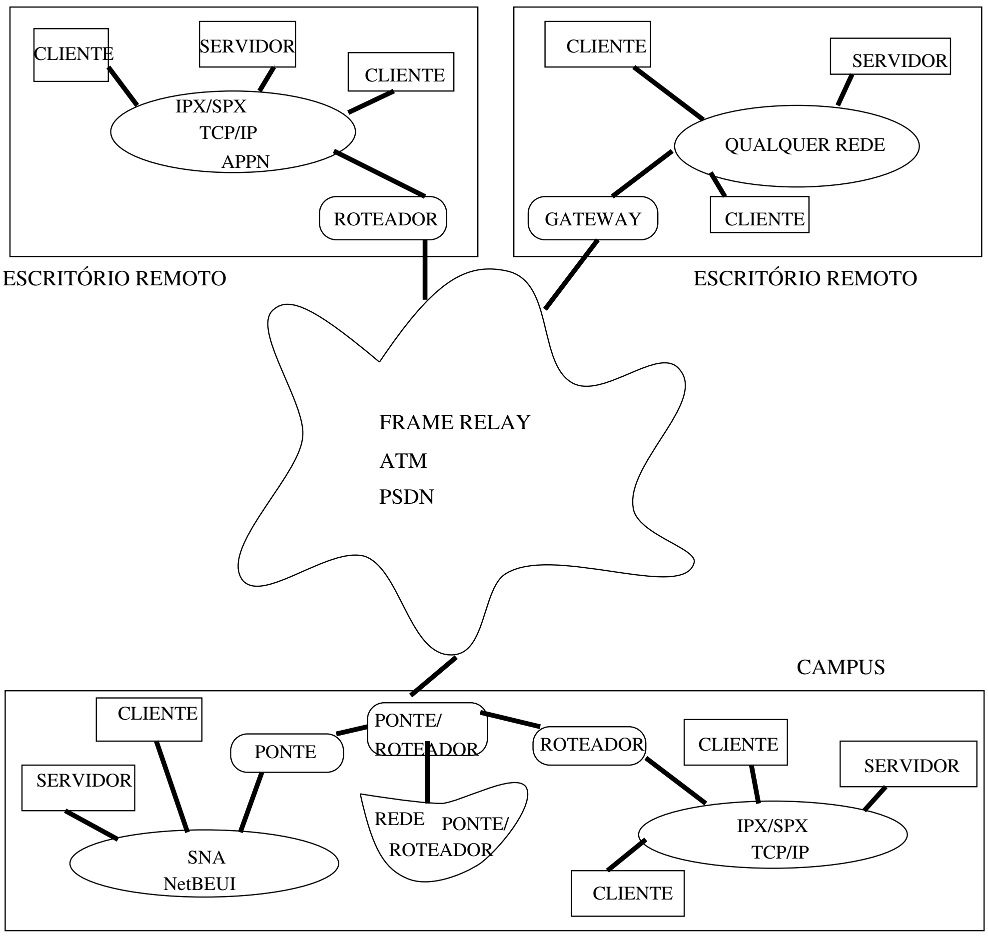

Figura 6.1: Interconexão de Redes

escrever um dispositivo lógico de rede para cada adaptador de rede existente, como também evita que os fabricantes de adaptadores de rede tenham que interfacear com todo tipo de pilha existente. O NDIS da Microsoft/3Com e ODI da Novell são os dois padrões de fato mais utilizados e cuidam do envio e recebimento do dado, como também, do gerenciamento da placa adaptadora de rede.

As interfaces de programação da aplicação (APIs) situam-se no topo da pilha de protocolos permitindo aos programadores "plugarem" os seus programas em uma única interface suportada por múltiplos protocolos. A interface mais difundida neste nível é a interface *socket* de Berkeley, comum nos sistemas UNIX. Outras opções incluem a interface da camada de transporte (TLI) usada no sistema operacional NetWare e muitas implementações UNIX; CPI-C é uma API para protocolo SNA podendo executar sobre pilhas SNA e TCP/IP e "Named Pipes" que interfaceia as pilhas NetBIOS, SPX/IPX e TCP/IP.

Outra tecnologia é representada pelos acopladores de procotolos (*protocol matchmarkers*) os quais permitem que uma aplicação desenvolvida para um protocolo de transporte (ex: SNA) possa executar em outras redes (ex: TCP/IP ou SPX/IPX). Esta estratégia elimina a necessidade de comportas (*gateways*). O nome formal utilizado para esta estratégia é Rede de Transporte Multiprotocolo (MPTN - *Multiprotocol Transport Network*). Após esta discussão, podemos observar que o ambiente heterogêneo das redes vem sendo trabalhado, do ponto de vista da tecnologia de interconexão de redes e das APIs

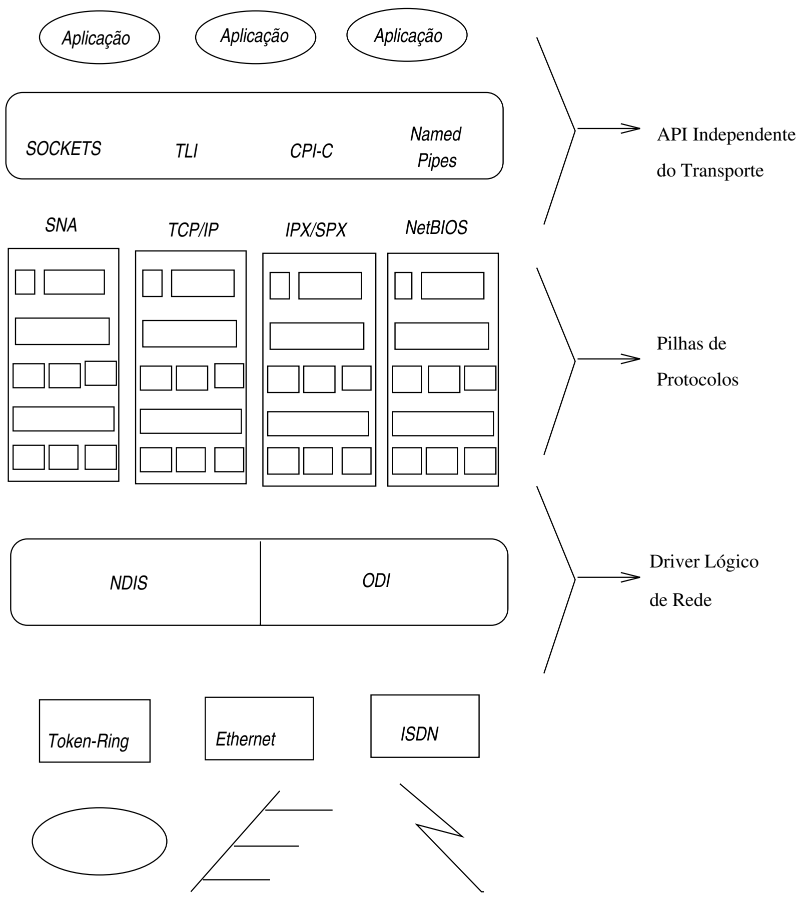

Figura 6.2: Modelo Cliente/Servidor

dos sistemas operacionais, em uma direção que permitirá às aplicações cliente/servidor executarem em pilhas de múltiplos protocolos facilitando bastante os aspectos de portabilidade das aplicações.

6.2 A Arquitetura Cliente/Servidor

O modelo cliente/servidor serve de base para o desenvolvimento de serviços no âmbito dos sistemas operacionais de rede.

6.2.1 Características do Servidor

O papel básico de um programa servidor consiste em atender vários clientes que desejam requisitar serviços sobre um recurso compartilhado gerenciado pelo servidor. As características de um programa servidor encontram-se listadas a seguir:

- espera pelas requisições dos clientes: o servidor passa a maior parte do tempo aguardando passivamente a chegada da requisição dos clientes. O servidor deve estar preparado para a situação onde vários clientes solicitam serviços simultaneamente;

- execução de várias requisições ao mesmo tempo: o servidor deve responder prontamente às requisições dos clientes. Para isto, deve ser capaz de atender múltiplos clientes concorrentemente e, ao mesmo tempo, proteger a integridade dos recursos compartilhados;
- prioridade para clientes importantes: o servidor deve ser capaz de atribuir níveis diferentes de prioridade aos clientes. Por exemplo, um servidor deve ser apto a atender uma requisição de serviço envolvendo uma atividade de elaboração de um relatório e, ao mesmo tempo, atender uma requisição do tipo OLTP (*On-Line Transaction Processing*) para clientes de alta prioridade;
- execução de atividades em *background*: um programa servidor deve poder executar tarefas em *background*. Estas últimas executam funções menos prementes do ponto de vista da atividade principal do servidor como, por exemplo, descarregar registros de dados para uma base de dados em momentos fora da demanda de pico dos serviços;
- robustez: como o servidor é uma entidade que normalmente realiza serviços críticos para os clientes, o programa servidor deve estar disponível permanentemente, o que demanda uma grande robustez, tanto do programa como do sistema no qual ele se insere;
- extensibilidade: pela própria natureza das suas funções, o programa servidor tende a demandar continuamente, mais memória e capacidade de processamento. O ambiente deve ser capaz de atender as necessidades de expansão dos servidores.

Um sistema operacional de rede estende os serviços básicos de um sistema nativo através da introdução servidores acessíveis de qualquer nó da rede. Clientes e servidores se comunicam via pilhas de protocolos de rede. Examinaremos a seguir os serviços básicos que um sistema operacional nativo deve oferecer à construção de servidores bem como os serviços estendidos propiciados por estes servidores.

6.2.2 Serviços básicos

Como visto anteriormente, um servidor deve exibir um alto grau de concorrência. O ideal é que uma tarefa seja alocada para cada requisição atendida pelo servidor. Neste caso, é importante que o sistema operacional seja multitarefa e suporte, consequentemente, os mecanismos de sincronização e comunicação entre tarefas. Do ponto de vista da eficiência é importante que o servidor possa, também, ter um alto grau de concorrência internamente a um único programa. O código do servidor pode executar de forma mais eficiente se tarefas forem alocadas a partes de um mesmo programa ao invés de programas separados. Estas tarefas são denominadas *threads*. Tarefas dentro de um mesmo programa são rápidas de serem criadas, possuem um chaveamento de contexto também rápido e possuem um acesso fácil à informação compartilhada. A figura 6.3 mostra o tipo de suporte que um servidor requer de um sistema operacional.

A seguir são discutidos os principais requisitos de um sistema operacional básico para suporte ao modelo cliente/servidor:

Preempção de tarefa

Um sistema operacional multitarefa e preemptivo deve fixar pedaços (*slots*) de tempo para a execução de cada tarefa. Sem um esquema multitarefa e preemptivo, uma tarefa cederá a vez para execução de uma outra tarefa somente se assim o desejar. Por outro lado, é mais seguro e fácil escrever programas servidores multitarefas em ambientes onde o sistema operacional realiza, automaticamente, o chaveamento entre tarefas;

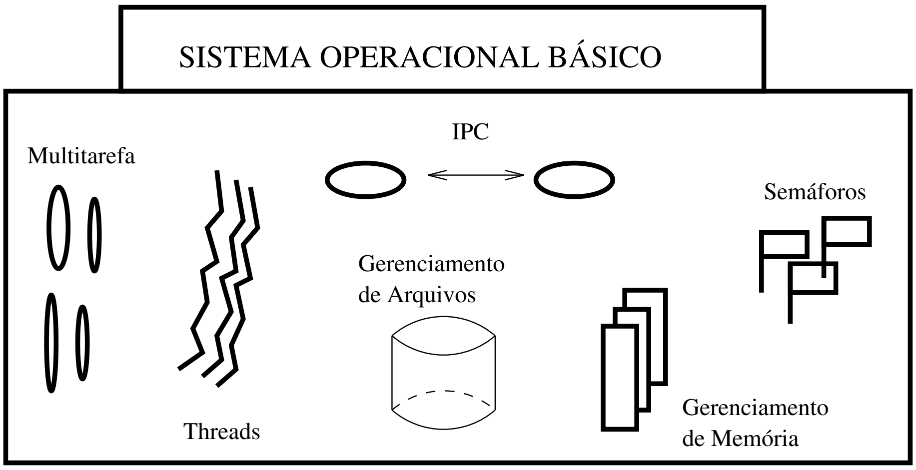

Figura 6.3: Serviços do Sistema Operacional necessários ao Modelo Cliente/Servidor

Prioridade das tarefas

O sistema operacional deve despachar as tarefas para execução baseado nas suas prioridades. Esta característica permite aos servidores diferenciarem o nível dos serviços baseados na prioridade dos seus clientes;

Semáforos

Um sistema operacional deve fornecer mecanismos simples de sincronização de modo a preservar a consistência dos recursos compartilhados pelas tarefas concorrentes;

Comunicação entre processos - IPC

O sistema operacional deve prover mecanismo que permita os processos se comunicarem. É desejável que o sistema redirecione, transparentemente, as chamadas a processos remotos. Este requisito é fundamental para o desenvolvimento de aplicações onde recursos e processos possam mover-se entre máquinas;

Threads

São unidades de concorrência dentro do próprio programa. As *threads* são utilizadas para criar programas servidores implicitamente concorrentes e dirigidos por eventos. Cada evento esperando pode ser associado a uma *thread* que fica bloqueada até a ocorrência do evento, enquanto outras *threads* executam de forma produtiva as atividades do servidor;

Proteção entre tarefas

O sistema operacional deve proteger as tarefas da interferência das outras tarefas com relação aos seus recursos. Uma única tarefa não deve ser capaz de comprometer todo o sistema. Estes requisitos de proteção também se estendem ao sistema de arquivos e às chamadas do sistema operacional;

Sistema de arquivo multiusuário de alto desempenho

O sistema de arquivos deve suportar múltiplas tarefas e oferecer os mecanismos de *lock* para proteção da integridade dos dados. Programas servidores costumam trabalhar sobre vários arquivos simultaneamente. Desta forma, o sistema de arquivos deve permitir que um número elevado de arquivos encontrem-se abertos simultaneamente sem que isto provoque uma degradação elevada no desempenho do sistema;

Gerenciamento eficiente de memória

O sistema de gerenciamento de memória deve suportar, eficientemente, programas grandes e objetos de dado também grandes. Estes programas e objetos de dado devem ser transferidos da memória para o disco e vice-versa (*swapping*) sem grandes dificuldades e, se possível, em blocos de granularidade pequena;

Ligação dinâmica

Os serviços do sistema operacional devem ser expansíveis. Deve ser fornecido mecanismo que permita a expansão dos serviços sem recompilação do sistema operacional.

6.2.3 Serviços Estendidos

Estes serviços devem ser suportados por software de sistema que irá explorar o aspecto distribuído oferecido pelas redes, fornecerá acesso flexível à informação compartilhada e permitirá que o sistema operacional de rede seja fácil de gerenciar e de ser mantido.

A figura 6.4 ilustra alguns dos serviços presentes nos sistemas operacionais de rede.

A seguir são apresentados os principais serviços que permitem estender um sistema operacional convencional para suportar aplicações cliente/servidor.

Comunicação

A extensão do sistema operacional deve oferecer um conjunto amplo de pilhas de protocolos de comunicação de modo a permitir que o servidor comunique-se com o maior número possível de clientes. Além disto, o servidor pode necessitar comunicar-se com outras plataformas servidoras no caso de que ele necessite de suporte para realização dos seus serviços;

Acesso a dispositivos

As extensões ao sistema operacional devem permitir estender os serviços de arquivo e impressão para a rede. O ideal, é que as aplicações acessem os dispositivos de forma transparente do fato deles estarem localizados na própria estação ou remotamente;

Grandes objetos binários - BLOBS (*Binary Large Objects*)

Grandes objetos, normalmente denominados BLOBs, como imagens, vídeo, gráficos, documentos multimídia, etc., necessitam de extensões do sistema operacional que permitam o transporte destes BLOBs em velocidades elevadas. Bases de dado e arquivos devem estar preparados para armazenar e recuperar estes objetos;

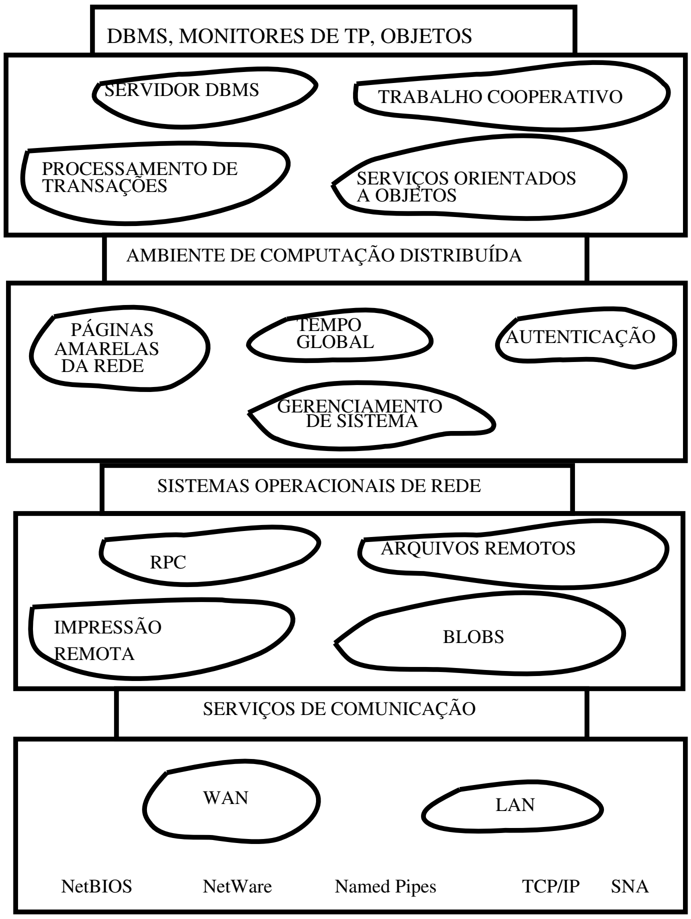

Figura 6.4: Serviços propiciados pelos sistemas operacinais de rede

Diretórios globais e páginas amarelas de rede

A extensão ao sistema operacional devem fornecer meios ao cliente de localizar servidores e seus serviços na rede através de algum serviço do tipo páginas amarelas de rede. Recursos de rede devem ser encontrados pelo nome. Os servidores devem ser capazes de registrar dinamicamente os seus serviços no responsável pelo diretório;

Serviços de autenticação e autorização

Os clientes devem ser capazes de provar aos servidores que eles são realmente quem eles dizem que são. O serviço de autorização determina se o cliente autenticado tem a permissão de obter o serviço remoto;

Gerenciamento do sistema

As extensões ao sistema operacional devem suportar a presença de uma plataforma de gerenciamento do sistema. Serviços de gerenciamento incluem: configuração, monitoramento de desempenho de todos os elementos, geração de alarmes quando algo não se comporta adequadamente, distribuição e gerenciamento de pacotes de software para as estações clientes, deteção de vírus e de invasores e, por último, capacidade de tarifação dos recursos utilizados;

Base de tempo de rede

Os clientes e servidores devem ser capazes de sincronizar seus relógios. Este tempo deve ser coordenado através de alguma autoridade universal de tempo;

Serviços orientados a objeto

Trata-se de uma área onde os serviços estendidos ainda demorarão um certo tempo para se tornarem comuns. Os serviços estão se orientando cada vez mais para a forma de objetos. Isto demandará serviços que realizarão um trabalho de intermediação dos objetos de modo a que cada objeto possa interagir com qualquer objeto na rede. Neste sentido serão necessários serviços para repositório e intercâmbio de objetos. As aplicações cliente/servidor do futuro serão entre objetos comunicantes e grupos de objetos constituirão associações de objetos para fornecimento de serviço. O objetivo consiste em extensões que permitam que objetos comuniquem-se através de redes, plataformas de hardware e sistemas operacionais.

6.3 O Conceito de Transparência

Os sistemas operacionais de rede estão deixando de ser um conjunto de estações de trabalho independentes, capazes de se comunicarem via um sistema de arquivos compartilhados, para se tornarem um ambiente de computação distribuída que torna a rede transparente para os usuários. Alguns dos aspectos de transparência que farão com que a "rede desapareça" são listados a seguir:

- transparência de localização: o usuário não deve necessitar conhecer a localização de um recurso;
- transparência de nome: o usuário deve ser capaz de utilizar a mesma convenção de nome para localizar um recurso na rede;
- transparência de conteúdo: o sistema deve ser capaz de mover recursos nos servidores sem que haja necessidade de mudanças no cliente;
- transparência de acesso: o sistema deve fornecer um único *password* (ou autenticação) que seja válido para todos os servidores da rede;
- transparência de replicação: o usuário não deve ser capaz de identificar quantas cópias existem de um determinado recurso;
- transparência de acesso distribuído: o usuário deve ser capaz de trabalhar com qualquer recurso na rede como se este fosse local. O sistema operacional da rede deve manipular os controles de acesso e fornecer os serviços de diretório;
- transparência de tempo distribuído: o usuário não deve perceber qualquer diferença de tempo entre os servidores de rede;
- transparência de falha: o usuário deve estar protegido de falhas na rede. O sistema operacional deve manipular as tentativas e reconexões de sessões. Deve ser fornecido, ainda, níveis de redundância de modo a oferecer tolerância a falhas;

- transparência de administração: o usuário deve interagir com uma única interface de gerenciamento de sistema. O sistema operacional de rede deve estar integrado com os serviços de gerenciamento locais.

O objetivo do sistema operacional de rede consiste em alcançar todas estas transparências sem, no entanto, interferir na autonomia de cada sistema operacional local. Um dos aspectos básicos associados à questão da transparência nos sistemas distribuídos relaciona-se à questão do nome. Nestes sistemas os nomes devem ser únicos no contexto no qual eles são usados ou resolvidos. O contexto de um nome pode ser imaginado como uma autoridade autônoma de nomeação. Pode ser comparado ao código de área de um sistema telefônico. Um espaço de nomes federado é formado por um conjunto de autoridades autônomas de nomeação. Neste caso, cada nome deve incluir a sua autoridade de nomeação. Por exemplo, caso estejamos dentro da autoridade de nomeação do sistema telefônico brasileiro, somente podemos chamar alguém na França utilizando o código do país, no caso França, seguido do número da pessoa. Este esquema de nomeação pode ser representado através de uma forma hierárquica (árvore) onde é possível introduzir-se vários níveis de hierarquia.

6.4 Comunicação nos Sistemas Operacionais de Rede

Esta seção fornece uma descrição das opções de transporte disponíveis para a implementação de serviços no âmbito dos sistemas operacionais de rede. As principais opções são:

- TCP/IP (com *sockets*);
- IPX/SPX (com TLI - *Transport Layer Interface*);
- NetBEUI (com NetBIOS);
- APPC/SNA (com CPI-C);
- *Named Pipes*.

As pilhas de transporte representam o **sistema nervoso** dos sistemas operacionais de rede e são elas que permitem agrupar os diferentes ambientes físicos. A figura 6.5 ilustra como as opções listadas anteriormente situam-se relativamente ao Modelo de Referência OSI/ISO.

6.5 TCP/IP e *sockets*

Trata-se de uma solução particularmente poderosa nos ambientes que requerem interconexão entre diferentes redes. O protocolo TCP/IP é disponível praticamente para todos os sistemas operacionais.

6.5.1 Protocolo Internet (IP)

A camada de rede do TCP/IP é suportada pelo protocolo IP o qual fornece o mecanismo básico para roteamento dos pacotes entre redes. Trata-se de um protocolo não confiável que depende da camada superior para fornecimento de um serviço confiável.

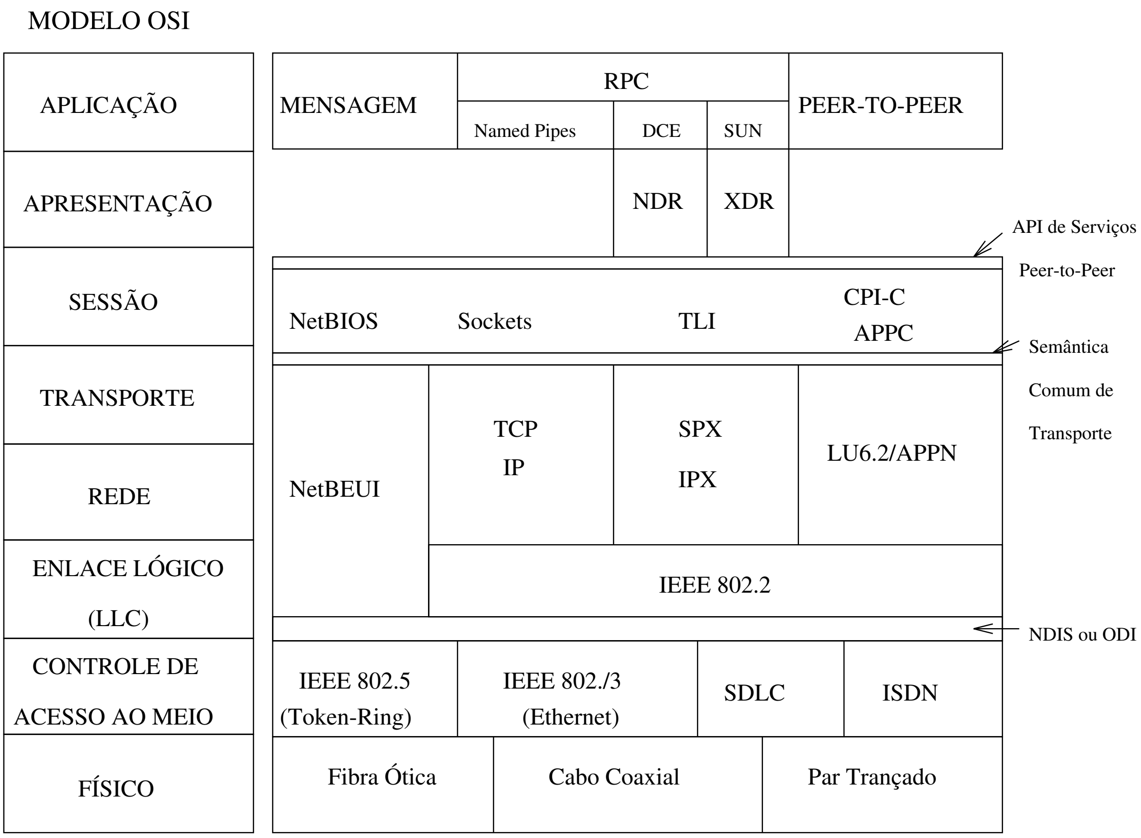

Figura 6.5: As pilhas de protocolo e o Modelo OSI

6.5.2 TCP e UDP

A camada de transporte do TCP/IP consiste de 2 protocolos que fornecem serviços de transporte fim-a-fim:

- TCP (*Transmission Control Protocol*): fornece um serviço confiável baseado em conexão;
- UDP (*User Datagram Protocol*): fornece serviço datagrama. Datagramas não são confiáveis mas, por outro lado, são rápidos.

O protocolo TCP/IP não especifica uma interface de aplicação. Entretanto, a interface denominada *sockets* surgiu como a API mais importante associada a esta pilha de protocolos.

6.5.3 *Sockets*

Foram introduzidos no UNIX BSD em 1981 como uma interface genérica para suportar comunicação via rede entre sistemas UNIX. Em 1985 a Sun introduziu os protocolos NFS (*Network File System*) e RPC (*Remote Procedure Call*) sobre a interface de *sockets*. Em 1986, a AT&T introduziu a interface TLI (*Transport Layer Interface*), interface esta que possui basicamente a mesma funcionalidade dos *sockets* só que de uma forma mais independente da rede.

Suporte para a interface *sockets* existe praticamente para todos os sistemas operacionais, incluindo: MS-DOS, Windows, OS/2, UNIX, e muitos *mainframes*. A API denominada WinSock é uma especificação que define o uso do TCP/IP no Windows. No sistemas derivados do UNIX, o *socket* faz parte do núcleo de modo a oferecer um serviço de comunicação entre processos. Nos sistemas que não são do tipo UNIX BSD, tais como: MS-DOS, Windows, Mac e OS/2, a interface de *sockets* é fornecida como uma biblioteca. Os três tipos de *sockets* disponíveis são: datagrama, *stream* e *raw*.

- *socket* do tipo datagrama: fornece a interface para os serviços de datagrama baseados no protocolo UDP. O UDP trata a comunicação na forma de pacotes independentes e não fornece garantias com relação à comunicação;

- *socket* do tipo *stream*: fornece a interface para o protocolo TCP que é um protocolo baseado em conexão e, portanto, confiável. O *socket* do tipo *stream* garante que os pacotes são enviados sem erros ou duplicação e são recebidos na mesma ordem em que são enviados;

- *socket* do tipo *raw*: fornece uma interface para os protocolos IP e ICMP (*Internet Control Message Protocolo*). Esta interface é utilizada normalmente para testar novos protocolos.

O endereço de um socket no TCP/IP consiste de 2 partes: um endereço internet (*IP-address*) e um número de porta conforme indicado na figura 6.6.

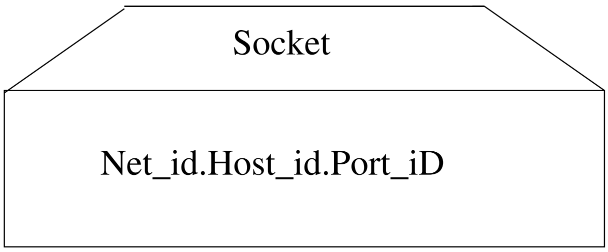

Figura 6.6: Endereço de *socket* no TCP/IP

Um endereço internet é um número de 32 bits, geralmente representado por quatro números decimais separados por pontos, que devem ser únicos para cada conexão de rede utilizada pelo TCP/IP. Um *host* TCP/IP pode ter tantos endereços internet quantas interfaces de rede ele possui. Uma porta é um ponto de entrada para uma aplicação residente no *host* sendo representada por um inteiro de 16 bits. As portas são normalmente utilizadas para definir pontos de entrada para acesso a servidores. Importantes programas comerciais como ORACLE e DBMS Sysbase possuem portas definidas e conhecidas por todos. Muitas implementações de *sockets* suportam o serviço denominado de *portmapper* desenvolvido pela Sun. Trata-se de um serviço de diretório para as portas. Quando da iniciação, os servidores menos conhecidos requisitam portas disponíveis no sistema e registram os seus serviços e os números das portas correspondentes no *portmapper*. Os processos clientes podem questionar o *portmapper*, através de qualificadores, para obter o número da porta associada a uma determinada aplicação servidora. A questão que se coloca é como acessar o *portmapper*? Trata-se da porta de número 111!!!

6.5.4 Utilização dos *sockets*

A figura 6.7 ilustra uma interação entre cliente e servidor envolvendo a interface de *socktes*.

As fases envolvidas na figura 6.7 são:

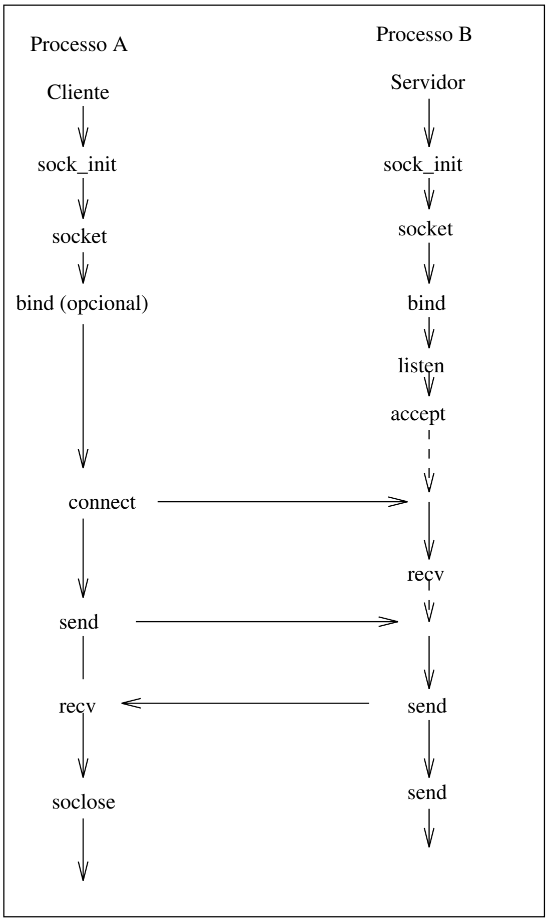

Figura 6.7: Interação entre cliente e servidor via *socket* do tipo *stream*

1. criação dos *sockets*: os processos que utilizam *sockets* devem iniciar o ambiente de tempo de execução vinculado aos *sockets* através da chamada *socket_init*. Na sequência deve ser criado o *socket* através da chamada *socket*;

2. estabelecimento de uma porta de serviço: os processos servidores devem associar (*bind*) os seus *sockets* a uma porta única de modo a se tornar conhecido na rede;

3. aguardar pedidos de conexão: os processos servidores que utilizam o *socket* do tipo *stream* (com conexão) devem evocar a função *listen* de modo a indicar que o processo encontra-se pronto para aceitar conexões solicitadas pelos clientes;

4. conexão ao servidor: o cliente emite uma chamada de *connect* no *socket* to tipo *stream* com a finalidade de iniciar uma conexão em uma porta associada a um determinado serviço de um processo servidor. O processo cliente deve localizar o endereço internet do servidor a partir de um servidor de nomes. Após o retorno com sucesso, o *socket* do cliente estará vinculado à conexão estabelecida;

5. aceitação da conexão: o servidor aceita uma requisição de conexão através da chamada *accept*. O servidor pode ficar ou não (opcional) bloqueado caso não existam requisições de conexão pen-

dentes. Caso várias requisições estejam pendentes, normalmente trabalha-se com uma FIFO, ou seja, a requisição mais antiga será atendida primeiro. Quando do retorno da chamada, no caso de existir solicitação de requisição, é criado um novo *socket* o qual será utilizado daquele momento em diante para receber as solicitações emitidas por aquele cliente, assim como, quando do retorno dos resultados após a realização do serviço. O *socket* original permanece livre para aceitar evocações de conexão emitidas por outros clientes. No caso da conexão estabelecida com um determinado cliente, o servidor pode decidir criar uma nova *thread* para gerenciar a comunicação com o cliente conectado ou utilizar a *thread* original para realizar o serviço solicitado. No primeiro caso, a *thread* original é utilizada para ficar aguardando novos clientes;

6. interação cliente/servidor: várias chamadas podem ser utilizadas pelos clientes e servidores para interação. As chamadas *readv, writev, send* e *recv* podem ser empregadas somente no caso de *sockets* que se encontram no estado conectado. As chamadas *sendto* e *recvfrom* podem ser utilizadas em qualquer momento mas necessitam de informação adicional relativamente aos endereços;

7. eliminação do *socket*: o cliente e o servidor utilizam a chamada *soclose* para encerramento da sessão TCP. Deve ser observado que o *socket* original continua apto para iniciar outras sessões. Caso o servidor tenha criado uma *thread* específica para atender o serviço, esta *thread* é encerrada;

8. aceitar novas interações ou encerrar as operações: o processo servidor pode evocar a chamada *soclose* relativamente ao *socket* original conhecido pelos clientes encerrando as suas atividades ou então aceitar novos pedidos de conexão por parte dos clientes;

6.6 IPX/SPX

Em função da penetração das soluções de rede oferecidas pela Novell, o IPX/SPX, que é a pilha de comunicação nativa do NetWare, é uma das soluções mais difundidas na prática. O IPX/SPX é uma implementação do protocolo de transporte e de rede XNS (*Xerox Network Services*).

A camada de rede do IPX/SPX é suportada pelo protocolo IPX (Internal Packet Exchange). Trata-se de um protocolo do tipo datagrama sem garantias. O SPX da Novell fornece um protocolo confiável sobre o IPX. O NetWare possui 12 chamadas de API que podem ser utilizadas para realização de serviços de datagrama suportados pelo IPX.

A camada de transporte do SPX/IPX é suportada pelo protocolo SPX (Sequence Packet Exchange), o qual fornece um serviço garantido e confiável através de um esquema de conexão. O protocolo é implementado em cima do IPX e oferece 16 chamadas na sua API.

6.6.1 TLI (*Transport Layer Interface*)

A pilha NetWare suporta quatro protocolos no topo da pilha SPX/IPX:

1. NetBios;

2. *Named Pipes*;

3. TLI;

4. APIs IPX/SPX.

Dentre estes protocolos, o TLI é utilizado pela Novell como padrão para as aplicações que necessitam da camada de transporte. Uma aplicação desenvolvida em TLI é, teoricamente, independente da pilha de protocolos. Com pouquissimas modificações, é possível que uma mesma aplicação execute em uma pilha IPX/SPX ou TCP/IP. TLI é uma implementação atualizada dos *sockets* de Berkeley e a sua API consiste de 25 chamadas.

6.6.2 NetBIOS e NetBEUI

NetBIOS foi o primeiro protocolo de redes voltado para comunicação programa-programa. É uma interface utilizada sobre várias pilhas de protocolo como TCP/IP, XNS, Vines, OSI, SPX/IPX e redes IBM/Microsoft (figura 6.8). NetBEUI é a pilha de protocolo que acompanha os produtos IBM e Microsoft. Originalmente foi introduzido como o transporte utilizado pelos comandos NetBIOS. Em geral, a literatura referencia NetBIOS englobando a interface NetBIOS e a pilha NetBEUI. Algumas vezes isto provoca confusão pois NetWare não tem nada a haver com NetBEUI mas, por outro lado, utiliza NetBIOS como interface tanto para o SPX/IPX como para o TCP/IP.

NetBEUI suporta serviços de datagrama e orientado a conexão. Também suporta um serviço dinâmico de nomes. Os comandos do NetBIOS são especificados em uma estrutura denominada Network Control Block (NCB), a qual contém também os parâmetros associados com o comando e os campos nos quais o NetBIOS retornará informação para o programa.

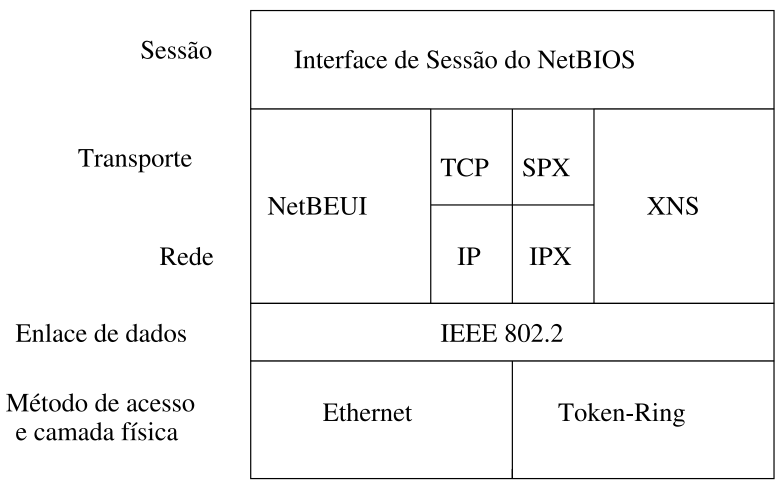

Figura 6.8: Interface NetBIOS

6.6.3 Canais Nomeados (*Named Pipes*)

São mecanismos IPC usados no OS/2 e UNIX (onde são também denominados FIFOS). O objetivo é tratar o canal de comunicação envolvendo dois processos como se fosse um arquivo. Desta forma, através de canais nomeados, os processos podem trocar dados como se eles estivessem escrevendo ou lendo em/de um arquivo sequencial. No lugar de colocar no disco os dados gravados no canal, os dados são convertidos para uma mensagem e enviados para outro processo. Semelhantemente, uma operação de leitura retira os dados da mensagem recebida e não do disco. No caso do UNIX os canais nomeados somente são utilizados por dois processos executando na mesma máquina. No caso do OS/2 os canais nomeados podem ser utilizados para comunicação entre processos via rede, especificando-os como canais remotos. O OS/2 redirecionan as operações relativas ao canal nomeado remoto usando o NetBIOS para alcançar o servidor especificado.

6.7 Protocolos SNA

APPC (*Application Program to Program Communication*) é o protocolo da IBM para comunicação entre processos em um ambiente distribuído suportado por rede SNA. A IBM está evoluindo a SNA na direção de um sistema operacional distribuído verdadeiro suportando serviços de diretório de rede, acesso transparente a recursos da rede (servidores, aplicações, *displays*, impressoras e dados), fluxos de dado e gerenciamento integrado de rede. A infraestrutura responsável por este processamento distribuído real denomina-se APPN (*Advanced Peer-to-Peer Network*). O objetivo da APPN é criar uma internet SNA sem o papel central do *mainframe* na hierarquia tradicional das configurações SNA. O *mainframe* é um nó como outro qualquer na rede. APPN permite que aplicações LU6.2 (denomina-se da sessão na qual programas comunicam-se na forma de pares, isto é, programa a programa) através das APIs APPC ou CPI-C (*Common Programming Interface for Communications*).

Através do uso da APPC ou CPI-C, um programa pode conversar através das 50.000 redes SNA intaladas no mundo, com qualquer outro programa. A arquitetura APPC é uma solução proposta pela IBM para comunicação programa-programa, processamento de transações distribuídas e acesso a base de dados remota. A interface de aplicação com o APPC denomina-se CPI-C. Trata-se de um grupo de chamadas do sistema correspondentes aos comandos, também chamados de verbos, da APPC. Ativar uma chamada do sistema CPI-C resulta na formatação automática de um bloco de controle APPC usando os parâmetros da chamada e o bloco de controle sendo passado à LU6.2. A IBM está expandindo o papel da interface CPI-C para ambientes não-SNA como, por exemplo, TCP/IP e OSI. A API CPI-C possui em torno de 40 chamadas e a APPC consiste de aproximadamente 60 chamadas. Muitas destas chamadas relacionam-se com a função de configuração.

A arquitetura APPC é otimizada para suportar conversações de longa duração. No caso, o custo para criação de uma sessão e alocação de conversações é elevado se apenas algumas mensagens forem enviadas. No caso de uma conversação demorada, este *overhead* é amortizado.

6.8 RPC da SUN Microsystems

A Sun Microsystems desenvolveu o RPC na década de 80 com o objetivo de permitir o desenvolvimento de aplicações cliente/servidor sem necessidade de programar com interfaces sob a camada de transporte como, por exemplo, *sockets*.

Este mecanismo de interação cliente/servidor serve de base para o desenvolvimento de serviços estendidos em sistemas operacionais de rede e encontra-se disponível em praticamente todos os sistemas operacionais derivados do UNIX.

Exemplo de servidores acessados via RPC:

- servidores NFS (Network File System): permitem acesso transparente à sistemas de arquivos remotos;
- portmapper: diretório de servidores RPC;

- servidor de NIS (Network Information System): serviço de nomes e passwords;
- servidores de estatísticas.

Nas sessões seguintes o protocolo RPC da Sun será referenciado simplesmente como RPC.

6.8.1 O cliente RPC

O lado do cliente do protocolo RPC é bastante simples. O cliente ativa **callrpc**, passando-lhe os oito parâmetros seguintes:

1. nome do nó remoto;
2. nome do servidor;
3. número da versão do servidor;
4. procedimento a ser ativado;
5. tipo do parâmetro de entrada enviado ao procedimento remoto;
6. parâmetro de entrada ou uma estrutura no caso de mais de um parâmetro de entrada;
7. tipo do parâmetro de saída retornado pelo procedimento remoto;
8. parâmetro de saída ou estrutura de dado no caso de mais de um parâmetro de saída.

Como o RPC é um mecanismo genérico, é necessário indicar qual servidor será contactado e qual procedimento dentro deste servidor deverá ser evocado. O parâmetro correspondente ao nó remoto é usado pelo RPC para obter o endereço IP do *host* onde se encontra executando o servidor. O parâmetro do nome do programa é, na realidade, um número atribuído por um administrador da rede para designar processos servidores individuais. O RPC especifica que para cada programa remoto executando em um computador deve ser associado um inteiro de 32 bits que o cliente deve usar para evocar o servidor. O RPC também atribui um inteiro para cada procedimento dentro do programa remoto. Os procedimentos são numerados sequencialmente a partir de 1. Por convenção, o número 0 é reservado para um procedimento de *echo* que pode ser utilizado para testar se o programa remoto pode ser alcançado. Desta forma, um procedimento remoto específico em um dado computador pode ser identificado pelo par: (prog, proc), onde prog identifica o programa remoto e proc identifica um procedimento dentro do programa remoto. Para evitar conflitos entre programas de diferentes organizações, a Sun divide os números dos programas em 4 grupos:

A RPC facilita a expansão do servidor ao permitir a atribuição de um número de versão para cada nova versão do servidor. Consequentemente, versões novas e antigas de um servidor poderão existir.

6.8.2 XDR - *External Data Representation*

Como máquinas diferentes possuem formatos diferentes, a RPC utiliza um protocolo chamado *External Data Representation* (XDR) para traduzir dados de/para um formato de intercâmbio pré-definido. Várias rotinas de serviço são fornecidas com a RPC para conversão de tipos básicos de dado para XDR. Estas incluem **xdr_int**, **xdr_double**, **xdr_float**, **xdr_long**, **xdr_string**, etc. O parâmetro tipo na evocação do procedimento remoto corresponde ao nome da rotina de serviço XDR que o RPC deve

De	Até	Valores atribuídos por
0x00000000	0x1FFFFFFF	Sun
0x20000000	0x3FFFFFFF	Gerente local
0x40000000	0x5FFFFFFF	Transiente
0x60000000	0xFFFFFFFF	Reservado

Figura 6.9: Números de programa do RPC da Sun

chamar para converter o parâmetro correspondente a ser enviado/recebido para/do procedimento. Por exemplo, suponha que um procedimento remoto, chamado LOOKUP, converta um nome em endereço (trata-se de uma função típica do servidor de nome). O comando **callrpc** poderia ser o seguinte:

callrpc(ESTACAO,SERVIDOR,MINHAVERSAO,
LOOKUP,xdr_string,nome,xdr_long,endereco).

No caso onde vários parâmetros devem ser transferidos, estes terão que ser colocados em uma única estrutura de dado. Por exemplo, caso dois inteiros tenham que ser enviados a um procedimento remoto, eles podem ser combinados na estrutura mostrada na figura 6.10.

```
struct    argumentos{
                int x;
                int y;
};
struct argumentos meusdados;
...
meusdados.x = primeiro inteiro;
meusdados.y = segundo inteiro;
...
```

Figura 6.10: Estrutura a ser enviada como parâmetro da RPC

Neste caso, o usuário deverá fornecer a sua própria rotina de serviço XDR para o comando **callrpc**. Escrever uma rotina deste tipo significa percorrer a estrutura de dados chamando a rotina XDR apropriada para cada elemento na estrutura. A figura 6.11 ilustra a rotina para a chamada seguinte:

callrpc(ESTACAO,SERVIDOR,VERSAO,media,
xdr_argumentos,meusdados,xdr_float,resultado).

```
struct argumentos    *pointer;        /*aponta para a minha estrutura*/
XDR      *xdrsp;                        /*aponta para o fluxo de dados XDR*/


{xdr_int(xdrsp,&pointer  ->  x);  /*converte o primeiro elemento*/
 xdr_int(xdrsp,&pointer  ->  y);  /*converte o segundo elemento*/
return;
}
...
```

Figura 6.11: Exemplo de rotina para conversão de estrutura de dados para XDR

6.8.3 Servidor de RPC

O servidor deve registrar os seus procedimentos que poderão ser acessados remotamente através da rotina **registerrpc**. Posteriormente, ativa a rotina da bilbioteca RPC denominada **svc_run**. Esta rotina permite ao servidor esperar as chamadas aos procedimentos registrados.

A rotina **registerrpc** utiliza os seis parâmetros discriminados a seguir:

1. númereo do programa a ser registrado;
2. número da versão;
3. número do procedimento que está sendo registrado;
4. nome do procedimento a ser evocado;
5. rotina de serviço XDR a ser evocada para os parâmetros de entrada;
6. rotina de serviço XDR a ser evocada para os parâmetros de saída.

A figura 6.12 ilustra como registrar o exemplo anterior e a figura 5.9 mostra o uso da RPC.

```
#define  UM   1
registerrpc(NUMERO,SERVIDOR,VERSAO,
              UM,media,xdr_argumentos,xdr_float);
svc_run();  /*nao retorna*/
```

Figura 6.12: Registro do servidor do exemplo da figura 5.7

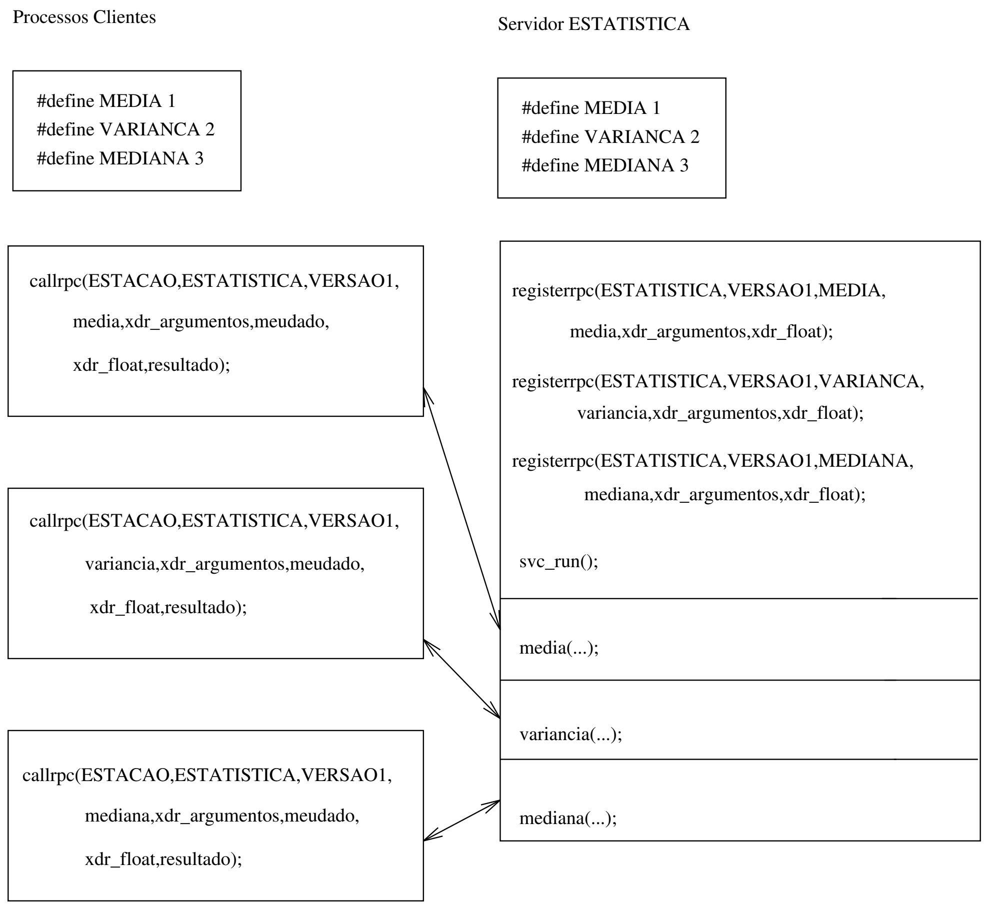

Figura 6.13: Utilização da RPC

6.8.4 *Portmapper*

Conforme mencionado anteriormente, os números de programa RPC são divididos em grupos de 20000000(hex) como mostrado na figura 6.9. A faixa definida pela Sun é usada para processos servidores de interesse geral para usuários da Internet como, por exemplo, *Network File System - NFS*, *Yellow Pages*, etc. A faixa seguinte pode ser usada para servidores específicos da empresa e os números são alocados pelos administrador da rede. A terceira faixa é alocada dinamicamente por rotinas de serviço RPC. Os números de programa são ligados dinamicamente a números de porta UDP e TCP por meio do processo RPC denominado *portmapper*. O processo *portmapper* é executado na porta 111.

A figura 6.14 apresenta um exemplo que ilustra o uso do *portmapper*[1]. Quando um servidor evoca **registerrpc** {1}, a rotina da biblioteca cria um *socket* e envia uma mensagem para o *portmapper* informando que o *socket* corresponde a um determinado procedimento remoto {2}. O *portmapper* utiliza esta informação para atualizar uma tabela interna contendo entradas constituídas de **programa/versão/procedimento** associadas a descritores de *socket*. A chamada **registerrpc** retorna {3} e o servidor chama **svc_run** {4} passando a aguardar uma evocação do procedimento registrado. Quando um cliente ativa **callrpc** {5}, a rotina da biblioteca RPC usa o parâmetro correspondente ao nome da

[1]Exemplo extraído do livro: Introdução aos Sistemas Cliente/Servidor: Guia prático para profissionais de sistemas. Paul E. Renaud. IBPI Press. 2a. tiragem.

estação onde se encontra o servidor para obter o endereço IP da estação. Em seguida envia uma mensagem para o *portmapper* rodando nesta máquina {6}. O *portmapper* pesquisa a tabela usando como entrada o número de **programa/versão/procedimento** fornecido pelo cliente e determina o descritor de *socket* que o servidor está usando. O *portmapper* responde à rotina da biblioteca RPC que executa do lado do cliente enviando o descritor de *socket* que o servidor está usando. Na sequência, **callrpc** envia os parâmetros da chamada de procedimento para o processo servidor {7} e o procedimento é ativado {8}. Quando o procedimento retorna {9}, os resultados são enviados de volta a **callrpc** {10}, que depois retorna os resultados para o cliente {11}.

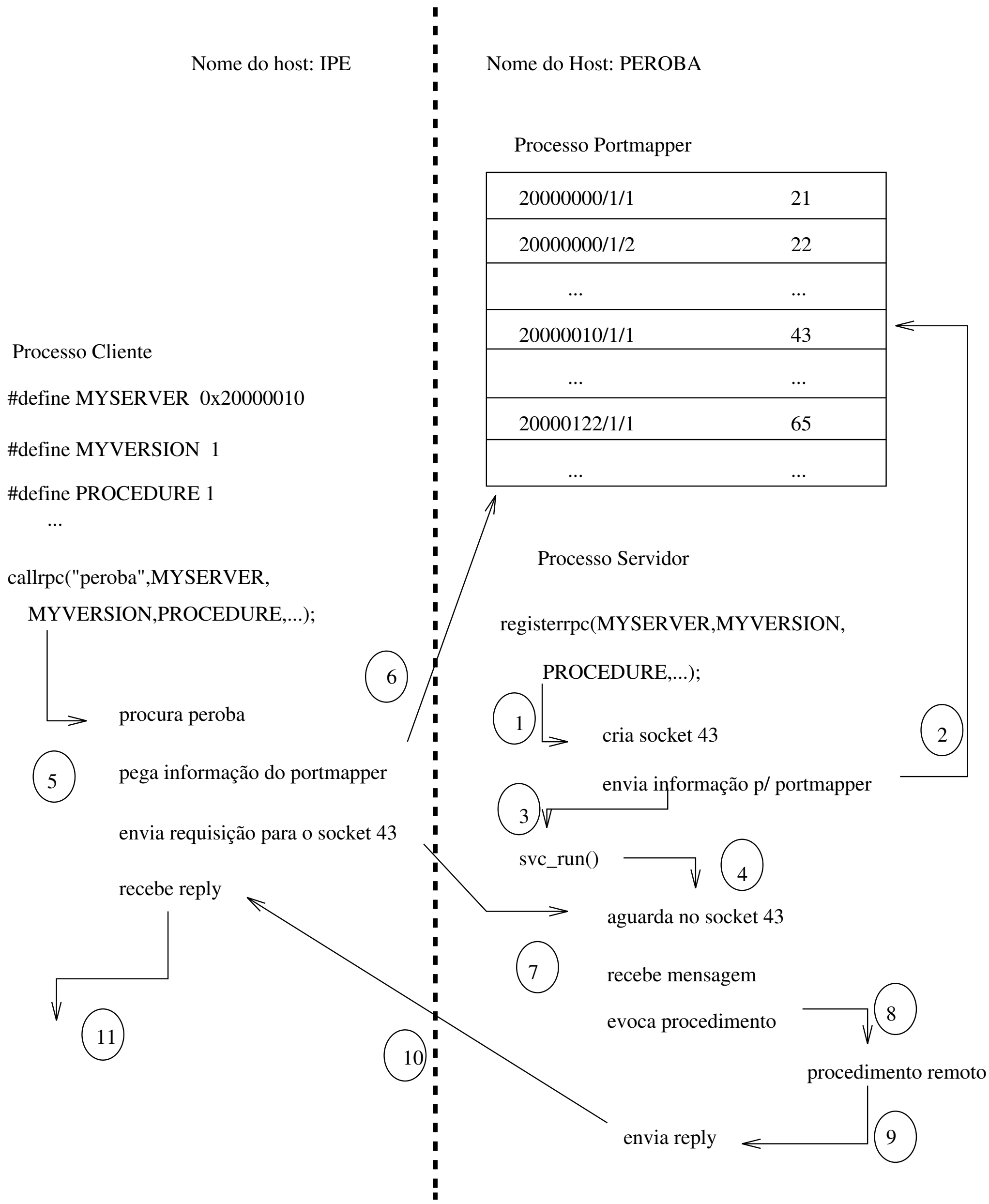

Figura 6.14: Uso do *Portmapper*

No caso do exemplo anterior, somente um procedimento remoto foi registrado por parte do servidor.

Caso mais de um procedimento seja registrado, quando da evocação de **svc_run** o servidor ficará aguardando a evocação de um procedimento remoto em um conjunto de *sockets*.

6.9 Exemplo de Serviço Estendido

O serviço NFS (Network File System) foi desenvolvido pela SUN Microsystems para compartilhamento *on-line* de arquivos. O acesso on-line permite a manipulação de arquivos remotos da mesma forma que arquivos locais. Como o protocolo que implementa o serviço não é proprietário da SUN, praticamente todos os fabricantes de computadores o adotaram para fins de interoperabilidade em termos de compartilhamento de arquivos.

O protocolo utiliza RPC sobre pilha TCP/IP. O protocolo permite a servidores *exportar* sistemas de arquivos para um conjunto de clientes (ou para toda a Internet). Clientes podem *montar* estes sistemas de arquivos criando referências locais para os mesmos (sem copiá-los). A montagem de um sistema de arquivos remoto exige uma interação cliente-servidor para fins de autenticação.

A implementação do protocolo NFS exige alteração do sistema operacional tanto no lado cliente quanto no lado servidor. O protocolo supõe a existência de um *sistema de arquivos virtual* (ver figura 6.15). Quando um arquivo é referenciado o sistema de arquivos virtual examina se o mesmo faz parte do sistema de arquivos local ou pertence a um sistema de arquivos remoto (montado localmente). Se o arquivo for local o sistema de arquivos virtual deixa a cargo do sistema de arquivos local a sua manipulação. Caso contrário, se o arquivo pertencer a um sistema de arquivos remoto, o sistema de arquivos virtual interage com o servidor NFS para a sua operação.

A interação cliente-servidor NFS permite que todas as operações sobre arquivos locais sejam conduzidas sobre arquivos remotos de forma totalmente transparente. Por exemplo, um cliente NFS pode:

- abrir um arquivo remoto;
- ler dados de um arquivo remoto;
- gravar dados num arquivo remoto;
- remover um arquivo remoto.

A figura 6.15 ilustra a arquitetura necessária à implementação do protocolo NFS.

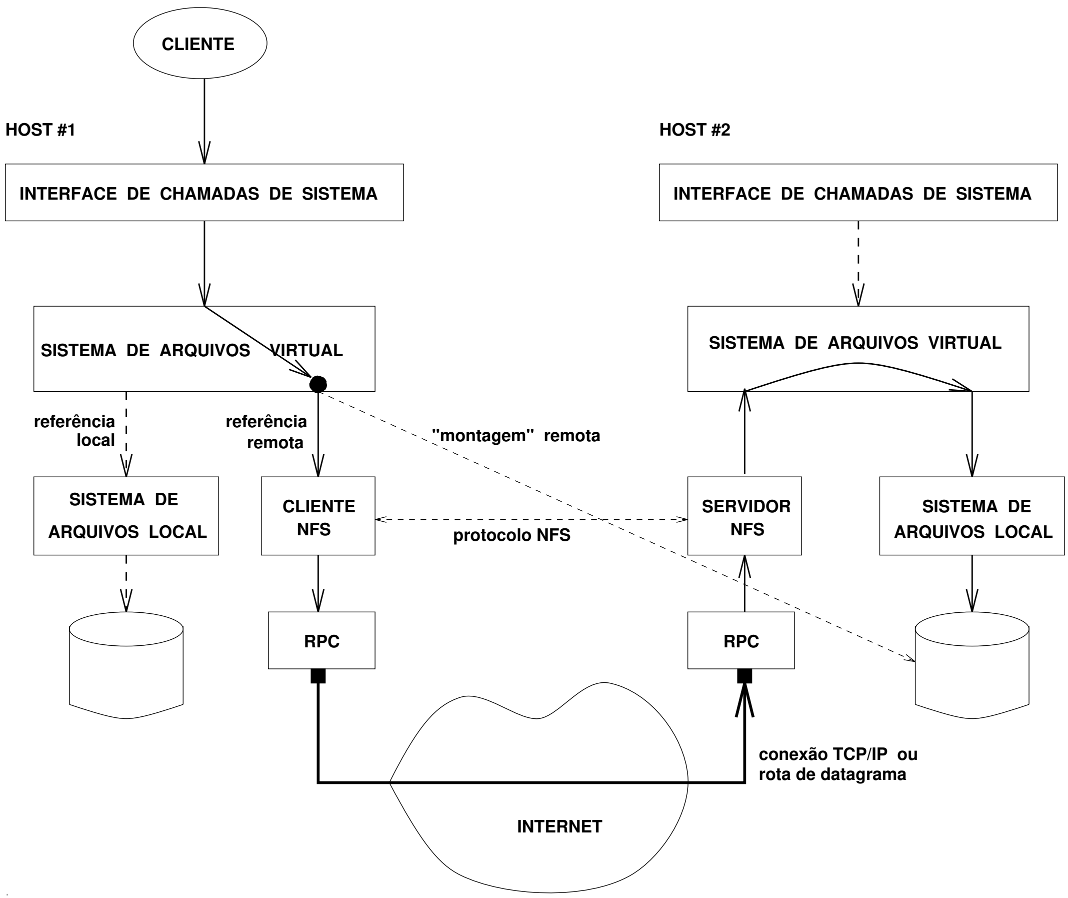

Figura 6.15: Arquitetura do NFS (Network File System).