4.7 Resumo

A primeira seção deste capítulo mostrou que os protocolos de transmissão da Internet fornecem dois blocos básicos a partir dos quais os protocolos em nível de aplicativos podem ser construídos. Há um compromisso interessante entre os dois protocolos: o protocolo UDP fornece um recurso simples de passagem de mensagem, que sofre de falhas por omissão, mas não tem penalidades de desempenho incorporadas. Por outro lado, em boas condições, o protocolo TCP garante a entrega das mensagens, mas à custa de mensagens adicionais e com latência e custos de armazenamento mais altos.

A segunda seção mostrou três estilos alternativos de empacotamento. O CORBA e seus predecessores optam por empacotar os dados para uso pelos destinatários que têm conhecimento anterior dos tipos de seus componentes. Em contraste, quando a linguagem Java serializa dados, ela inclui informações completas sobre os tipos de seu conteúdo, permitindo ao destinatário reconstruí-los puramente a partir do conteúdo. A linguagem XML, assim como a Java, inclui informações completas sobre os tipos de dados. Outra grande diferença é que o CORBA exige uma especificação dos tipos dos dados a serem empacotados (no IDL) para gerar os métodos de empacotamento e desempacotamento, enquanto a linguagem Java usa reflexão para serializar e desserializar objetos. Uma variedade de meios é usada para gerar código XML, dependendo do contexto. Por exemplo, muitas linguagens de programação, incluindo a Java, fornecem processadores para fazer a transformação entre objetos em nível de XML e de linguagem.

Mensagens *multicast* são usadas na comunicação entre os membros de um grupo de processos. O *multicast* IP é disponibilizado tanto para redes locais como para a Internet e tem a mesma semântica de falhas que os datagramas UDP; porém, apesar de sofrer de falhas por omissão, é uma ferramenta útil para muitas aplicações *multicast*. Algumas aplicações têm requisitos mais restritos – em particular, o de que a distribuição *multicast* deva ser atômica; isto é, ela deve ter uma distribuição do tipo tudo ou nada. Outros requisitos do *multicast* estão relacionados ao ordenamento das mensagens, o mais exigente, que impõe que todos os membros de um grupo devem receber todas as mensagens na mesma ordem.

Multicast também pode ser suportado pelas redes de sobreposição nos casos em que, por exemplo, *multicast* IP não é suportado. Mais geralmente, as redes de sobreposição oferecem um serviço de virtualização da arquitetura da rede, permitindo que serviços especializados sejam criados sobre a infraestrutura de interligação em rede subjacente, por exemplo, UDP ou TCP. As redes de sobreposição resolvem parcialmente os problemas associados ao princípio fim-a-fim de Saltzer, permitindo a geração de abstrações de rede mais específicas para o aplicativo.

O capítulo terminou com um estudo de caso da especificação MPI desenvolvida pela comunidade de computação de alto desempenho e que apresenta suporte flexível para a passagem de mensagens, junto a suporte adicional para passagem de mensagens por comunicação coletiva.

Exercícios

4.1 É concebivelmente útil que uma porta tenha vários receptores? *páginas 148*

4.2 Um servidor cria uma porta que ele utiliza para receber pedidos dos clientes. Discuta os problemas de projeto relativos ao relacionamento entre o nome dessa porta e os nomes usados pelos clientes. *páginas 148*

4.3 Os programas das Figuras 4.3 e 4.4 estão disponíveis no endereço [www.cdk5.net/ipc] (em inglês). Utilize-os para fazer uma série de testes para determinar as condições nas quais os datagramas, às vezes, são descartados. Dica: o programa cliente deve ser capaz de variar o número de mensagens enviadas e seus tamanhos; o servidor deve detectar quando uma mensagem de um cliente específico é perdida. *páginas 150*

4.4 Use o programa da Figura 4.3 para fazer um programa cliente que leia repetidamente uma linha de entrada do usuário, a envie para o servidor em uma mensagem datagrama UDP e receba uma mensagem do servidor. O cliente estabelece um tempo limite em seu soquete para que possa informar o usuário quando o servidor não responder. Teste este programa cliente com o servidor da Figura 4.4. *páginas 150*

4.5 Os programas das Figuras 4.5 e 4.6 estão disponíveis no endereço [www.cdk5.net/ipc] (em inglês). Modifique-os de modo que o cliente leia repetidamente uma linha de entrada do usuário e a escreva no fluxo. O servidor deve ler repetidamente o fluxo, imprimindo o resultado de cada leitura. Faça uma comparação entre o envio de dados em mensagens de datagrama UDP e por meio de um fluxo. *página 153*

4.6 Use os programas desenvolvidos no Exercício 4.5 para testar o efeito sobre o remetente quando o receptor falha e vice-versa. *página 153*

4.7 O XDR da Sun empacota dados convertendo-os para uma forma *big-endian* antes de transmiti-los. Discuta as vantagens e desvantagens desse método em comparação com o CDR do CORBA. *página 160*

4.8 O XDR da Sun alinha cada valor primitivo em um limite de quatro bytes, enquanto o CDR da CORBA alinha um valor primitivo de tamanho *n* em um limite de *n* bytes. Discuta os compromissos na escolha dos tamanhos ocupados pelos valores primitivos. *página 160*

4.9 Por que não há nenhuma tipagem de dados explícita no CDR do CORBA? *página 160*

4.10 Escreva um algoritmo, em pseudocódigo, para descrever o procedimento de serialização mostrado na Seção 4.3.2. O algoritmo deve mostrar quando identificadores são definidos ou substituídos por classes e instâncias. Descreva a forma serializada que seu algoritmo produziria para uma instância da seguinte classe *Couple*.

```
class Couple implements Serializable{
        private Person one;
        private Person two;
        public Couple(Person a, Person b) {
                one = a;
                two = b;
        }
}
```

página 162

4.11 Escreva um algoritmo, em pseudocódigo, para descrever a desserialização da forma serializada produzida pelo algoritmo definido no Exercício 4.10. Dica: use reflexão para criar uma classe a partir de seu nome, um construtor a partir de seus tipos de parâmetro e uma nova instância de um objeto a partir do construtor e dos valores de argumento. *página 162*

4.12 Por que dados binários não podem ser representados diretamente em XML, por exemplo, como valores em Unicode? Os elementos XML podem transportar *strings* representados como *base64*. Discuta as vantagens ou desvantagens de usar esse método para representar dados binários. *página 164*

4.13 Defina uma classe cujas instâncias representem referências de objeto remoto. Ela deve conter informações semelhantes àquelas mostradas na Figura 4.13 e deve fornecer métodos de acesso necessários para os protocolos de nível mais alto (consulte requisição-resposta, no Capítulo 5, para ver um exemplo). Explique como cada um dos métodos de acesso será usado por esse protocolo. Dê uma justificativa para o tipo escolhido para a variável de instância que contém informações sobre a interface do objeto remoto. *página 168*

4.14 O *multicast* IP fornece um serviço que sofre de falhas por omissão. Faça uma série de testes, baseada no programa da Figura 4.14, para descobrir as condições sob as quais uma mensagem *multicast* às vezes é eliminada por um dos membros do grupo *multicast*. O kit de testes deve ser projetado de forma a permitir a existência de vários processos remetentes. *página 170*

4.15 Esboce o projeto de um esquema que utilize retransmissões de mensagem *multicast* IP para superar o problema das mensagens descartadas. Seu esquema deve levar em conta os seguintes pontos:

i) podem existir vários remetentes;

ii) geralmente apenas uma pequena parte das mensagens é descartada;

iii) os destinatários podem não enviar uma mensagem necessariamente dentro de um limite de tempo em particular.

Suponha que as mensagens que não são descartadas chegam na ordem do remetente. *página 173*

4.16 Sua solução para o Exercício 4.15 deve ter superado o problema das mensagens descartadas no *multicast* IP. Em que sentido sua solução difere da definição de *multicast* confiável? *página 173*

4.17 Imagine um cenário no qual mensagens *multicast* enviadas por diferentes clientes são entregues em ordens diferentes a dois membros do grupo. Suponha que esteja em uso alguma forma de retransmissões de mensagem, mas que as mensagens que não são descartadas cheguem na ordem do remetente. Sugira o modo como os destinos poderiam remediar essa situação. *página 173*

4.18 Reveja a arquitetura da Internet apresentada no Capítulo 3 (consulte as Figuras 3.12 e 3.14). Que impacto a introdução das redes de sobreposição tem sobre essa arquitetura, em particular sobre a visão conceitual do programador em relação à Internet? *página 175*

4.19 Quais são os principais argumentos para a adoção de uma estratégia de supernós no Skype? *página 177*

4.20 Conforme discutido na Seção 4.6, o padrão MPI oferece diversas variantes da operação *send*, incluindo *MPI_Rsend*, que presume que o destinatário está pronto para receber no momento do envio. Quais otimizações são possíveis na implementação se essa suposição estiver correta e quais são as repercussões no caso de ser falsa? *página 180*