Chat sobre anel

Redes de Computadores e Internet
2º Semestre 2023/2024
Projeto de Laboratório, versão 1

1. Introdução

Pretende-se desenvolver uma *aplicação chat*, simples, para a troca de mensagens entre nós de uma rede. A aplicação *chat* é suportada numa *camada de encaminhamento* que direciona mensagens das suas origens para os seus destinos ao longo de caminhos mais-curtos na rede. Por sua vez, a camada de encaminhamento é suportada numa *camada topológica* que mantem a rede interligada aquando da entrada de novos nós e a saída de nós.

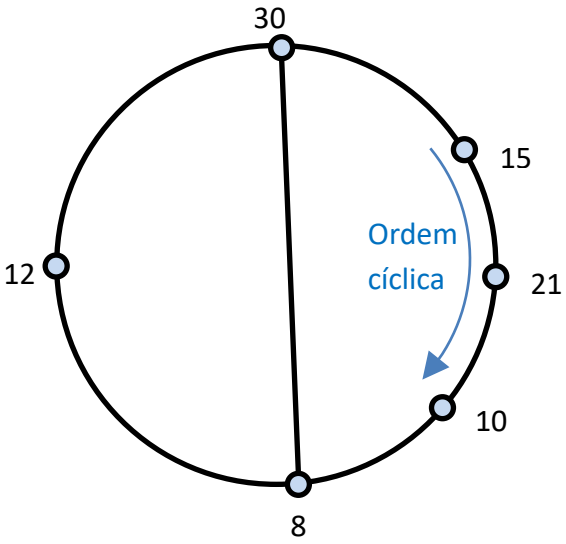
- Camada topológica. A rede que interliga os nós tem como base um anel que circula sobre todos. A este anel podem adicionar-se cordas. O protocolo topológico mantém o anel aquando da entrada de um novo nó e da saída de um nó.
- Camada de encaminhamento. O encaminhamento é por caminho mais-curto. O protocolo de encaminhamento mantém em cada nó uma tabela de expedição que indica para cada destino qual dos seus vizinhos está mais próximo desse destino. Ele reage à adição de uma adjacência e à remoção de uma adjacência.
- Aplicação *chat*. A aplicação consiste simplesmente na troca de mensagens de origens para destinos.

2. Camada topológica: anel com cordas

Cada nó tem um identificador único. Para efeitos de manutenção do anel, este está orientado, por exemplo no sentido dos ponteiros do relógio. O *sucessor* de um nó é o próximo nó do anel percorrido no sentido dos ponteiros do relógio e o *predecessor* de um nó é o nó que o tem como sucessor. O *segundo-sucessor* de um nó é o sucessor do seu sucessor. Na rede da figura, o sucessor do nó 30 é o nó 15, o seu predecessor é o nó 12 e o seu segundo-sucessor é o nó 21. Cada nó mantém atualizados os identificadores e contactos do seu sucessor e segundo-sucessor, bem como o identificador do seu predecessor. Há um servidor de nós (fornecido pelo corpo docente) onde estão registados os identificadores e os contactos de todos os nós pertencentes ao anel a cada momento.

2.1 Entrada de um nó

Suponha-se que o nó i pretende entrar no anel. Ele interroga o servidor de nós, escolhendo para seu futuro sucessor no anel um nó j arbitrário. Seja k e l o sucessor e o predecessor do nó j, respetivamente. O nó i tem de atualizar o seu sucessor para j, o seu segundo-



sucessor para k e o seu predecessor l. O nó de l tem que atualizar o seu sucessor para i e o seu segundo-sucessor para j. Finalmente, o predecessor do nó l tem de atualizar o seu segundo-sucessor para i. Concretamente, a entrada do nó i pede as seguintes ações.

1. O nó i abre uma adjacência com o nó j e estabelece-o como seu sucessor.
2. O nó j informa o nó i da existência do nó k e informa o nó l da entrada do nó i.
3. O nó i atualiza o seu segundo-sucessor para k. Concorrentemente, o nó l fecha a adjacência com o nó j, abre uma adjacência com o nó i, informa-o de que será seu predecessor, informa o seu predecessor da entrada do nó i, atualiza o seu sucessor para i e o seu segundo-sucessor para j.
4. O predecessor do nó l estabelece este nó como seu segundo-sucessor.

Depois de estabelecida a adjacência entre o nó i e o seu predecessor, aquele regista-se no servidor de nós.

No caso particular de existir apenas um nó, não há adjacências. Em todos os outros casos, existirão tantas adjacências ao longo do anel quanto nós.

Adicionalmente às adjacências com o sucessor e o predecessor, um nó pode estabelecer um máximo de uma outra adjacência com um qualquer outro nó que não aqueles. A esta adjacência chama-se *corda*. Para esse efeito, o nó consulta o servidor de nós e escolhe o outro extremo da única corda estabelecida por si. Na rede da figura, a adjacência 8-30 é uma corda.

2.2 Saída de um nó

Suponha-se que o nó i com sucessor j, segundo-sucessor k, e predecessor l pretende sair do anel. Ele começa por retirar o seu registo do servidor de nós. O nó l tem de atualizar

o seu sucessor para j e o seu segundo-sucessor para k. O predecessor do nó l tem de atualizar o seu segundo-sucessor para j.

Concretamente, a saída do nó i pede as seguintes ações.

1. O nó i fecha as adjacências com as suas cordas, com o nó j e com o nó l.
2. O nó l abre uma adjacência com o nó j, promove-o a sucessor, informa-o que será seu predecessor e informa o seu predecessor da existência do nó j.
3. O nó j informa o nó l da existência do nó k. Concorrentemente, o predecessor do l atualiza o seu segundo-sucessor para j.
4. O nó l atualiza o seu segundo-sucessor para k.

3. Camada de encaminhamento: caminhos mais-curtos

Para efeitos de encaminhamento, cada nó mantém uma *tabela de encaminhamento* indexada por nó destino e por vizinho que indica qual o caminho do nó ao nó destino através do vizinho. A partir da tabela de encaminhamento, cada nó determina uma *tabela de caminhos mais-curtos* e uma *tabela de expedição*, ambas indexadas por nó destino. A tabela de caminhos mais-curtos indica o caminho mais-curto para chegar a cada destino (casos de empate são quebrados arbitrariamente). Sempre que ela é alterada, o nó anuncia as alterações aos seus vizinhos através de mensagens de encaminhamento. A tabela de expedição indica o vizinho ao longo do caminho mais-curto até cada um dos destinos. Ela direcionará as mensagens da aplicação *chat*.

A tabela abaixo mostra as tabelas de encaminhamento, de caminhos mais-curtos e de expedição do nó 30 da rede da figura anterior. Por exemplo, da tabela de encaminhamento lê-se que o nó 30 tem três caminhos alternativos para alcançar o nó destino 10, a saber: o caminho 30-8-10, através do vizinho 8; o caminho 30-12-8-10, através do vizinho 12; e o caminho 30-15-21-10, através do vizinho 15. O mais-curto destes três é o caminho 30-8-10 (a negrito) que é copiado para a tabela de caminhos mais-curtos na entrada respeitante ao destino 10. O vizinho ao longo do caminho mais-curto é o 8 que é copiado para a tabela de expedição também na entrada respeitante ao destino 10.

Duas notas. Primeira. O caminho mais-curto de um nó para si próprio é o caminho trivial constituído só por si. Por exemplo, embora as entradas na tabela de encaminhamento do nó 30 relativamente ao destino 30 estejam todas vazias, a tabela de caminhos mais-curtos tem o caminho trivial 30 na entrada correspondente ao nó destino 30. Segunda nota. Um nó não tem caminho válido para o destino através de um vizinho se o caminho mais-curto do vizinho ao destino passa por esse nó. Por exemplo, o caminho mais-curto do nó 15 ao

nó 12 é o caminho 15-30-12. Portanto, o caminho do nó 30 ao nó 12 através do vizinho 15 seria o caminho 30-15-30-12 que repete o nó 30 e é considerado inválido.

Tabela de encaminhamento 30

	8	12	15
8	**30-8**	30-12-8	-
10	**30-8-10**	30-12-8-10	30-15-21-10
12	30-8-12	**30-12**	-
15	-	-	**30-15**
21	30-8-10-21	30-12-8-10-21	**30-15-21**
30	-	-	-

Tabela de caminhos mais curtos 30

8	30-8
10	30-8-10
12	30-12
15	30-15
21	30-15-21
30	30

Tabela de expedição 30

8	8
10	8
12	12
15	15
21	15
30	-

3.1 Receção de uma mensagem de encaminhamento

Uma mensagem de encaminhamento é da forma (i, n, ϕ) em que ϕ é um caminho com origem em i e destino em n. Quando o nó j recebe do vizinho i a mensagem de encaminhamento (i, n, ϕ), ele toma as seguintes ações:

1. Calcula o caminho para n através do vizinho i. Se j é um nó de ϕ, então não há caminho válido para n através de i. Caso contrário, se j não é um nó de ϕ, então o caminho de j para n através de i é a concatenação de j com ϕ, $j - \phi$.
2. Atualiza a entrada indexada por n e i na tabela de expedição com o cálculo anterior.
3. Atualiza a entrada indexada por n na tabela de caminhos mais-curtos. Se, efetivamente, a nova entrada for diferente da anterior, ela é anunciada a cada um dos vizinhos em mensagens de encaminhamento.
4. Atualiza a entrada indexada por n na tabela de expedição.

3.2 Adição de uma adjacência

Quando uma adjacência é adicionada entre dois nós, cada um deles envia ao outro uma mensagem de encaminhamento por cada entrada na tabela de caminhos mais-curtos.

Inicialmente, um nó que acabe de entrar na rede só se tem a si próprio como destino na sua tabela de caminhos mais-curtos.

3.3 Remoção de uma adjacência

Quando uma adjacência entre dois nós é removida, cada um deles remove as entradas indexadas pelo outro da sua tabela de encaminhamento, atualiza a tabela de caminhos mais-curtos, anuncia as entradas alteradas desta aos seus vizinhos em mensagens de encaminhamento e atualiza a sua tabela de expedição.

4. Especificação

Cada grupo de dois alunos deve concretizar a aplicação **`COR (ChatOnRing)`** correspondendo a um nó num anel. As adjacências são concretizadas com sessões TCP, formando uma *rede de sobreposição* à rede de computadores subjacente. Um nó é cliente na sessão TCP com o seu sucessor e servidor na sessão TCP com o seu predecessor. Numa corda, o nó que a criou é cliente e o seu vizinho é o servidor. Sem perda de generalidade, assumimos que não haverá mais do que 16 nó na rede e que cada nó cria no máximo uma corda. O registo e cancelamento de registo no servidor de nós é concretizado por UDP. A aplicação **`COR`** é composta pelos elementos seguintes:

- comando de invocação da aplicação;
- interface de utilizador;
- protocolo topológico;
- protocolo de encaminhamento;
- protocolo *chat*.

4.1 Comando de invocação da aplicação

A aplicação **`COR`** é invocada com o comando

`COR` ***`IP TCP regIP regUDP`***

em que ***`IP`*** e ***`TCP`*** constituem o *contacto* do nó criado pela aplicação: ***`IP`*** é o endereço da máquina que aloja a aplicação e ***`TCP`*** é o porto TCP servidor da aplicação (porto de escuta). Os parâmetros ***`regIP`*** e ***`regUDP`*** são o contacto, endereço IP e porto UDP, respetivamente, do servidor de nós fornecido pelo corpo docente. Por omissão, estes parâmetros tomam os valores **`193.136.138.142`** e **`59000`**.

4.2 Interface de utilizador

A interface de utilizador consiste nos seguintes comandos, as abreviaturas dos quais estão entre parêntesis.

- **`join (j)`** ***`ring id`***
  Entrada de um nó com identificador ***`id`*** no anel ***`ring`***. Se o identificador ***`id`*** já estiver a ser usado por outro nó nesse anel, então a aplicação escolhe um identificador único, avisando o utilizador dessa alteração. Os valores de ***`ring`*** são

representados por três dígitos, podendo variar entre ***000*** e ***999***; os valores de ***id*** são representados por dois dígitos, podendo variar entre ***00*** e ***99***.

- **direct join (dj)** ***id succid succIP succTCP***
  Entrada de um nó com identificador ***id*** diretamente num anel, sem consulta ao servidor de nós. A adjacência é estabelecida com o nó do anel de indentificador ***succid*** e contacto ***succIP*** e ***succTCP***. Este comando permite criar um anel sem recurso ao servidor de nós. Contudo, o utilizador terá de saber que o identificador escolhido será único no anel e terá de saber o identificador e contacto de um outro nó no anel. O nó não ficará registado no servido de nós. Se ***id*** for igual ***succid***, então cria-se um anel com um só nó.
- **chord (c)**
  Estabelecimento de uma corda para um nó do anel que não o sucessor ou o predecessor. Cada nó estabelece no máximo uma corda.
- **remove chord (rc)**
  Eliminação da corda previamente estabelecida.
- **show topology (st)**
  Mostra o estado do protocolo topológico, consistindo em: (i) o identificador dele próprio; (ii) o identificador e contacto do seu sucessor; (iii) o identificador e o contacto do seu segundo-sucessor; (iii) o identificador do seu predecessor; (iv) os identificadores dos vizinhos em cordas, se os houver
- **show routing (sr)** ***dest***
  Mostra a entrada da tabela de encaminhamento de um nó relativa ao destino ***dest***.
- **show path (sp)** ***dest***
  Mostra o caminho mais curto de um nó para o destino ***dest***.
- **show forwarding (sf)**
  Mostra a tabela de expedição de um nó.
- **message (m)** ***dest message***
  Envio da mensagem ***message*** ao nó ***dest***.
- **leave (l)**
  Saída do nó do anel.
- **exit (x)**
  Fecho da aplicação.

4.3 Protocolo topológico

O contacto de um nó é constituído pelo seu endereço IP e pelo seu porto TCP servidor. Na comunicação entre um nó e o servidor de nós são usadas seis mensagens protocolares.

- **NODES** ***ring***
  Um nó pede ao servidor de nós o identificador e respetivos contactos dos nós existentes no anel ***ring***.
- **NODESLIST** ***ring<LF>***
  id1 IP1 TCP1<LF>
  id2 IP2 TCP2<LF>

…

O servidor de nós envia uma lista com uma linha por nó no anel ***ring***. Cada linha dessa lista contém o identificador e contacto de um nó.

- **REG** ***ring id IP TCP***
  Um nó regista-se no anel ***ring***.
- **OKREG**
  O servidor de nós confirma o registo de um nó.
- **UNREG** ***ring id***
  O nó retira o seu registo no anel ***ring***.
- **OKUNREG**
  O servidor de nós confirma a retirada do registo de um nó.

Para a manutenção do anel são usadas quatro mensagens protocolares.

- **ENTRY** ***new new.IP new.port<LF>***
  Um nó informa outro sobre a entrada no anel do nó com identificador ***new*** e contacto ***new.IP*** e ***new.port***. Mensagens deste tipo têm origens nos nós que entram no anel e passam pelos seus sucessores para serem entregues aos predecessores destes.
- **SUCC** ***succ succ.IP succ.port<LF>***
  Um nó informa o seu predecessor sobre o identificador ***succ*** e contacto ***succ.IP*** e ***succ.port*** do seu sucessor.
- **PRED** ***i<LF>***
  Um nó informa o seu sucessor sobre a sua identidade.
- **CHORD** ***i<LF>***
  Um nó que estabelece uma corda informa o outro extremo da corda sobre a sua identidade.

4.4 Protocolo de encaminhamento

Existe apenas um tipo de mensagem de encaminhamento.

- **ROUTE** ***i n path<LF>***
  O nó ***i*** anuncia a um seu vizinho o caminho ***path*** para alcançar o destino ***n***. O caminho ***path*** é uma sequência de identificadores de nós separados por "-", começando em ***i*** e terminando em ***n***.

4.5 Protocolo de *chat*

Existe apenas um tipo de mensagem de *chat*.

- **CHAT** ***i n chat<LF>***
  O nó origem ***i*** envia ao nó destino ***n*** a mensagem ***chat***. Estas mensagens têm 128 carateres no máximo.

5. Desenvolvimento

Cada grupo de alunos deve adquirir a destreza necessária sobre programação em redes para realizar a aplicação proposta.

5.1 Etapas recomendadas

As etapas seguintes são recomendadas na concretização do projeto. Os números em parêntese retos são a duração de referência para cada etapa.

1. Leitura atenta do enunciado do projeto. [um dia]
2. Criação de um anel com apenas um nó, registo e cancelamento de registo, comandos **`join`** e **`leave`**. [uma semana, em paralelo com o seguinte]
3. Criação de um anel com apenas dois nós, sem registo no servidor de nós, comandos **`direct join`**, **`show topology`**, **`message`**. [uma semana, em paralelo com o anterior]
4. Criação de um anel com vários nós, comandos **`join`** e **`direct join`**. [uma semana]
5. Manutenção do anel com saída de nós, comando **`leave`**. [meia semana]
6. Implementação do protocolo de encaminhamento, comandos **`show routing`**, **`show forwarding`**. [uma semana]
7. Inclusão de cordas, comandos **`chord`** e **`remove chord`**, e integração das três camadas. [uma semana]

5.2 Chamadas de sistema

O código da aplicação fará uso das seguintes primitivas:

- leitura de informação do utilizador para a aplicação: `fgets()`;
- conversão entre *strings* e tipos de dados: `sscanf()`, `sprintf()`;
- gestão de um cliente UDP: `socket()`, `close()`;
- gestão de um servidor UDP: `socket()`, `bind()`, `close()`;
- comunicação UDP: `sendto()`, `recvfrom()`;
- gestão de um cliente TCP: `socket()`, `connect()`, `close()`;
- gestão de um servidor TCP: `socket()`, `bind()`, `listen()`, `accept()`, `close()`;
- comunicação TCP: `write()`, `read()`;
- multiplexagem de entradas síncronas: `select()`.

5.3 Depuração

Faça uso das estratégias seguintes que permitem uma boa depuração do código:

- comente o código à medida que o desenvolve;
- use um depurador, `gdb` ou `xgdb`, para inspecionar a execução sequencial do código num nó;
- use o comando `nc` (`netcat`) para testar a comunicação com um cliente ou com um servidor, quer seja TCP ou UDP;
- execute o analisador de protocolos Wireshark para visualizar todas as mensagens trocadas com um nó e seus conteúdos;

- tente inter-operar um nó implementado pelo seu grupo com um nó implementado por outro grupo (isto é possível porque o formato das mensagens é o mesmo para todos).

Quer os clientes quer os servidores devem terminar graciosamente, pelo menos nas seguintes situações de falha:

- mensagens mal formatadas;
- sessão TCP fechada abruptamente;
- erro nas chamadas de sistema.

6. Entrega do Projecto

Cada grupo deve submeter no sistema fénix um ficheiro comprimido `.zip` (não é aceite `.rar`) com os três elementos seguintes: (i) código fonte da aplicação **`COR`**; (ii) a respetiva `makefile`; e (iii) uma ficha de auto-avaliação preenchida que será disponibilizada mais tarde. A execução do comando `make` deverá produzir o executável **`COR`** sem quaisquer erros ou avisos. O código será testado apenas no ambiente de desenvolvimento do laboratório.

A data de submissão do projeto é sexta-feira, dia 22 de março, até às 23:59, ou segunda-feira, dia 25 de março, até às 23:59, com um ponto de penalização.

7. Porquê este projeto?

Este projeto tem como objetivo evidente capacitar os alunos a programar aplicações em rede usando a API (*Aplication Programming Interface*) de *sockets*, enfatizando: (i) a partilha da rede e a multiplexagem de informação por meio de endereços IP e portos; (ii) a assimetria na comunicação patente na distinção entre cliente e servidor; (iii) a diferença de serviços fornecidos por TCP e UDP; e (iv) as próprias primitivas da comunicação. No entanto, os objetivos do projeto vão muito para além dessa aprendizagem operacional. O projeto envolve conceitos fundamentais em redes de computadores e levanta questões que iremos encontrar ao longo do semestre em contextos diversos. Por exemplo:

- **Redes de sobreposição**. Como foi referido, a rede em anel construída com sessões TCP é uma rede sobreposta à rede de computadores e à internet subjacentes. Redes deste tipo são usadas, por exemplo, nas chamadas redes de distribuição conteúdos (*Content Distribution Networks*, CDNs) que, como o nome sugere, servem de intermediárias entre consumidores e fornecedores de conteúdos. Para perceber a utilidade das redes de sobreposição, note que um utilizador da internet não tem possibilidade de escolher o caminho seguindo pelos pacotes de dados de um ponto A para um ponto B (veremos isto nas aulas). Pode acontecer que o melhor caminho de A para B, segundo determinado critério, passe por C, conquanto o caminho oferecido pela internet não passe por C. Com uma rede de sobreposição que contenha uma sessão TCP de A para C e outra de C para B é possível

direcionar os pacotes de dados pelo caminho A-C-B. Já agora, em boa verdade se diga que a própria internet pode ser considerada uma rede de sobreposição tendo por base redes de comutação de circuitos em fibra ótica.

- **Arquitetura em camadas**. O sistema de comunicação proposto é modular, estando dividido em três camadas, com a camada topológica fornecendo um serviço à camada de encaminhamento e este fornecendo um serviço à camada de aplicação. Consegue caracterizar cada um desses serviços? Note que apesar de todas as adjacências na rede serem constituídas por sessões TCP, sendo, portanto, fiáveis, a camada de encaminhamento não garante em absoluto a entrega de todas as mensagens da aplicação. Porquê? Como faria para garantir que todas as mensagens da aplicação fossem entregues? Seria possível substituir apenas a camada de encaminhamento por outra que usasse um algoritmo diferente para o cálculo dos caminhos mais-curtos? Note que o servidor de nós apenso à camada topológica não necessita de registar todos os nós presentes no anel. Como alteraria a camada topológica para minimizar a informação mantida no servidor de nós e que implicações teria essa alteração na camada de encaminhamento?
- **Padronização dos protocolos**. Para que o equipamento de rede de vendedores diferentes possa inter-operar, é necessário criar normas de comunicação. O presente projeto especifica a sintaxe e a semântica das mensagens que podem ser usadas em cada protocolo de cada camada, fora isso, conferindo liberdade de implementação. Experimente inter-operar o seu projeto com o de outro grupo.
- **Estado de um protocolo**. A informação global do estado de um dado protocolo, a qual lhe permite reagir automaticamente a acontecimentos externos, não se encontra em um local físico. Ao invés, está distribuída pelos vários nós. Por exemplo, no programa a desenvolver, não há uma estrutura de dados com toda a informação do anel. O servidor de nós fornecido pelo corpo docente também não foi programado com uma tal estrutura, tendo capacidade para registar todos os nós do anel, mas não a sua interligação. O anel é uma abstração substanciada na união de informações de estado locais a cada nó, nomeadamente as suas sessões TCP com o sucessor e o predecessor.
- **Concorrência intra-camada**. Os acontecimentos externos a um protocolo são assíncronos; para além disto, em geral, não sabemos quais os atrasos relativos na receção de mensagens protocolares lançadas na rede. Por consequência, haverá sempre condições de corrida na execução de um protocolo. Algumas condições de corrida não afetam a funcionalidade do protocolo, outras poem-na em causa. Por exemplo, no presente projeto, quando um nó sai e fecha as suas adjacências, não sabemos prever se é o sucessor ou o predecessor que primeiro deteta o fecho da sua adjacência com o nó. Em ambos os casos, o anel tem de ser devidamente reposto. É isso que acontece? Numa primeira versão deste projeto, o corpo docente não tinha incluído as mensagens protocolares **`PRED i`** e **`CHORD i`**. Que problemas poderiam surgir sem estas mensagens? O que acontece se dois nós consecutivos no anel resolvem sair mais ou menos em simultâneo?

- **Concorrência inter-camadas.** A questão das condições de corrida também se põe entre camadas e há que garantir que elas não afetam o normal funcionamento de cada uma das camadas. Por exemplo, quando um nó sai, o seu predecessor deve logo atualizar a sua tabela de encaminhamento ou, por hipótese, esperar que o anel seja primeiro restituído?
- **Conetividade.** Uma topologia baseada num anel que percorre todos os nós resiste à falha de um qualquer nó, isto é, mesmo que um qualquer nó falhe, existirá sempre um caminho entre quaisquer outros dois nós. Será o converso verdade? Isto é, uma rede que resista à falha de um qualquer nó tem, necessariamente, que conter um anel que percorre todos os nós? Como se pode caracterizar em termos de anéis uma rede que resista a falha de um qualquer nó? Talvez consiga adquirir intuição para responder a esta última questão, mas um argumento rigoroso está fora do âmbito desta UC!

8. Bibliografia

- José Sanguino, A Quick Guide to Networking Software, 5ª edição, 2020.
- W. Richard Stevens, Unix Network Programming: Networking APIs: Sockets and XTI (Volume 1), 2ª edição, Prentice-Hall PTR, 1998, ISBN 0-13-490012-X, capítulo 5.
- Michael J. Donahoo, Kenneth L. Calvert, TCP/IP Sockets in C: Practical Guide for Programmers, Morgan Kaufmann, ISBN 1558608265, 2000.
- Manual on-line, comando `man`.