Capítulo 8

Comunicação entre tarefas

Muitas implementações de sistemas complexos são estruturadas como várias tarefas interdependentes, que cooperam entre si para atingir os objetivos da aplicação, como por exemplo em um navegador Web. Para que as várias tarefas que compõem uma aplicação possam cooperar, elas precisam comunicar informações umas às outras e coordenar suas atividades, para garantir que os resultados obtidos sejam coerentes. Este módulo apresenta os principais conceitos, problemas e soluções referentes à comunicação entre tarefas.

8.1 Objetivos

Nem sempre um programa sequencial é a melhor solução para um determinado problema. Muitas vezes, as implementações são estruturadas na forma de várias tarefas interdependentes que cooperam entre si para atingir os objetivos da aplicação, como por exemplo em um navegador Web. Existem várias razões para justificar a construção de sistemas baseados em tarefas cooperantes, entre as quais podem ser citadas:

Atender vários usuários simultâneos: um servidor de banco de dados ou de e-mail completamente sequencial atenderia um único cliente por vez, gerando atrasos intoleráveis para os demais clientes. Por isso, servidores de rede são implementados com vários processos ou threads, para atender simultaneamente todos os usuários conectados.

Uso de computadores multiprocessador: um programa sequencial executa um único fluxo de instruções por vez, não importando o número de processadores presentes no hardware. Para aumentar a velocidade de execução de uma aplicação, esta deve ser "quebrada" em várias tarefas cooperantes, que poderão ser escalonadas simultaneamente nos processadores disponíveis.

Modularidade: um sistema muito grande e complexo pode ser melhor organizado dividindo suas atribuições em módulos sob a responsabilidade de tarefas interdependentes. Cada módulo tem suas próprias responsabilidades e coopera com os demais módulos quando necessário. Sistemas de interface gráfica, como os projetos *GNOME* [Gnome, 2005] e *KDE* [KDE, 2005], são geralmente construídos dessa forma.

Construção de aplicações interativas: navegadores Web, editores de texto e jogos são exemplos de aplicações com alta interatividade; nelas, tarefas associadas à interface reagem a comandos do usuário, enquanto outras tarefas comunicam através da rede, fazem a revisão ortográfica do texto, renderizam imagens na janela, etc. Construir esse tipo de aplicação de forma totalmente sequencial seria simplesmente inviável.

Para que as tarefas presentes em um sistema possam cooperar, elas precisam **comunicar**, compartilhando as informações necessárias à execução de cada tarefa, e **coordenar** suas atividades, para que os resultados obtidos sejam consistentes (sem erros). Este módulo visa estudar os principais conceitos, problemas e soluções empregados para permitir a comunicação entre tarefas executando em um sistema. A coordenação entre tarefas será estudada a partir do Capítulo 10.

8.2 Escopo da comunicação

Tarefas cooperantes precisam trocar informações entre si. Por exemplo, a tarefa que gerencia os botões e menus de um navegador Web precisa informar rapidamente as demais tarefas caso o usuário clique nos botões *stop* ou *reload*. Outra situação de comunicação frequente ocorre quando o usuário seleciona um texto em uma página da Internet e o arrasta para um editor de textos. Em ambos os casos ocorre a transferência de informação entre duas tarefas distintas.

Implementar a comunicação entre tarefas pode ser simples ou complexo, dependendo da situação. Se as tarefas estão no mesmo processo, elas compartilham a mesma área de memória e a comunicação pode então ser implementada facilmente, usando variáveis globais comuns. Entretanto, caso as tarefas pertençam a processos distintos, não existem variáveis compartilhadas; neste caso, a comunicação tem de ser feita por intermédio do núcleo do sistema operacional, usando chamadas de sistema. Caso as tarefas estejam em computadores distintos, o núcleo deve implementar mecanismos de comunicação específicos, fazendo uso de mecanismos de comunicação em rede. A Figura 8.1 ilustra essas três situações.

Apesar da comunicação poder ocorrer entre *threads*, processos locais ou computadores distintos, com ou sem o envolvimento do núcleo do sistema, os mecanismos de comunicação são habitualmente denominados de forma genérica como "mecanismos IPC" (*Inter-Process Communication*).

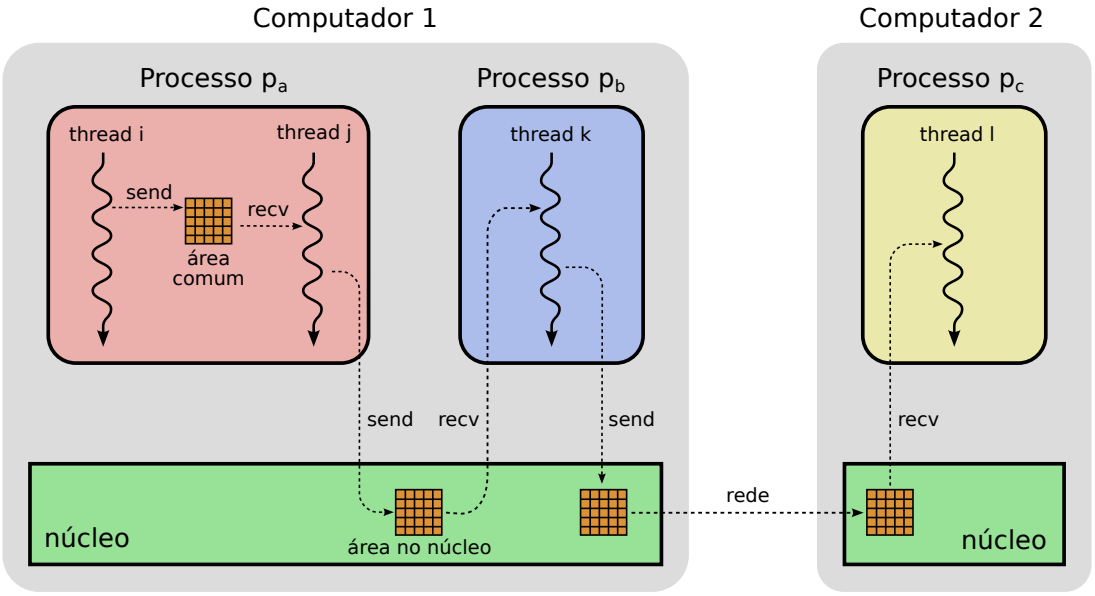

Figura 8.1: Comunicação intraprocesso ($t_i \rightarrow t_j$), interprocessos ($t_j \rightarrow t_k$) e intersistemas ($t_k \rightarrow t_l$).

8.3 Aspectos da comunicação

A implementação da comunicação entre tarefas pode ocorrer de várias formas. Ao definir os mecanismos de comunicação oferecidos por um sistema operacional, seus projetistas devem considerar muitos aspectos, como o formato dos dados a transferir, o sincronismo exigido nas comunicações, a necessidade de *buffers* e o número de emissores/receptores envolvidos em cada ação de comunicação. As próximas seções analisam alguns dos principais aspectos que caracterizam e distinguem entre si os vários mecanismos de comunicação.

8.3.1 Comunicação direta ou indireta

De forma mais abstrata, a comunicação entre tarefas pode ser implementada por duas primitivas básicas: *enviar (dados, destino)*, que envia os dados relacionados ao destino indicado, e *receber (dados, origem)*, que recebe os dados previamente enviados pela origem indicada. Essa abordagem, na qual o emissor identifica claramente o receptor e vice-versa, é denominada **comunicação direta**.

Poucos sistemas empregam a comunicação direta; na prática são utilizados mecanismos de **comunicação indireta**, por serem mais flexíveis. Na comunicação indireta, emissor e receptor não precisam se conhecer, pois não interagem diretamente entre si. Eles se relacionam através de um **canal de comunicação**, que é criado pelo sistema operacional, geralmente a pedido de uma das partes. Neste caso, as primitivas de comunicação não designam diretamente tarefas, mas canais de comunicação aos quais as tarefas estão associadas: *enviar (dados, canal)* e *receber (dados, canal)*. A Figura 8.2 ilustra essas duas formas de comunicação.

8.3.2 Sincronismo

Em relação aos aspectos de sincronismo do canal de comunicação, a comunicação entre tarefas pode ser:

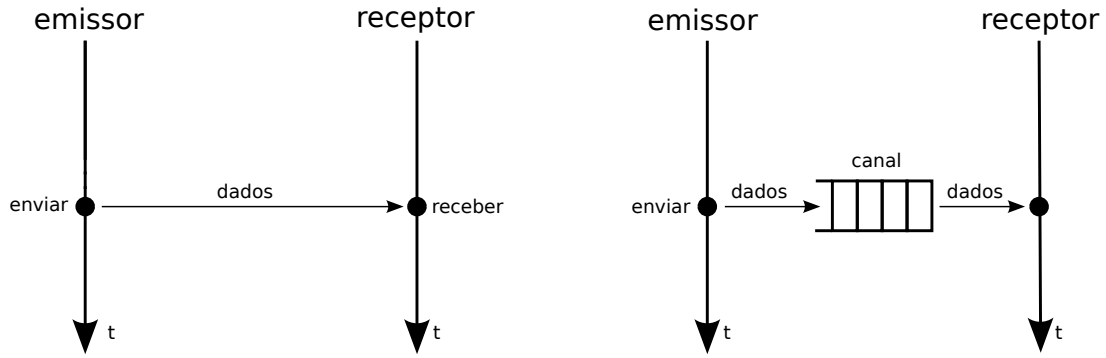

Figura 8.2: Comunicação direta (esquerda) e indireta (direita).

Síncrona (ou bloqueante): quando as operações de envio e recepção de dados bloqueiam (suspendem) as tarefas envolvidas até a conclusão da comunicação: o emissor será bloqueado até que a informação seja recebida pelo receptor, e vice-versa. A Figura 8.3 apresenta os diagramas de tempo de duas situações frequentes em sistemas com comunicação síncrona.

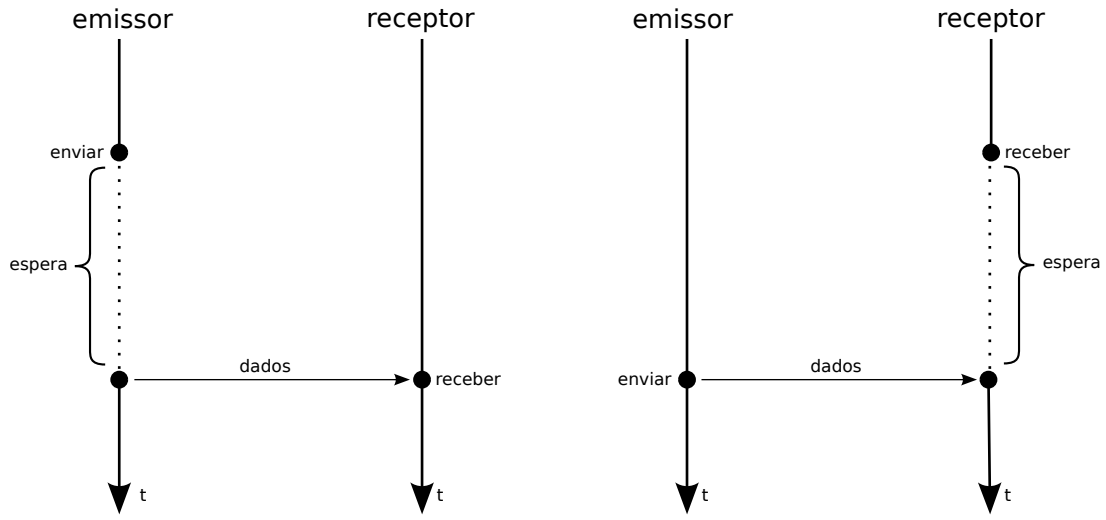

Figura 8.3: Comunicação síncrona.

Assíncrona (ou não-bloqueante): em um sistema com comunicação assíncrona, as primitivas de envio e recepção não são bloqueantes: caso a comunicação não seja possível no momento em que cada operação é invocada, esta retorna imediatamente com uma indicação de erro. Deve-se observar que, caso o emissor e o receptor operem ambos de forma assíncrona, torna-se necessário criar um canal ou *buffer* para armazenar os dados da comunicação entre eles. Sem esse canal, a comunicação se tornará inviável, pois raramente ambos estarão prontos para comunicar ao mesmo tempo. Esta forma de comunicação está representada no diagrama de tempo da Figura 8.4.

Semissíncrona (ou semibloqueante): primitivas de comunicação semissíncronas têm um comportamento síncrono (bloqueante) durante um prazo pré-definido. Caso esse prazo se esgote sem que a comunicação tenha ocorrido, a primitiva se encerra com uma indicação de erro. Para refletir esse comportamento, as

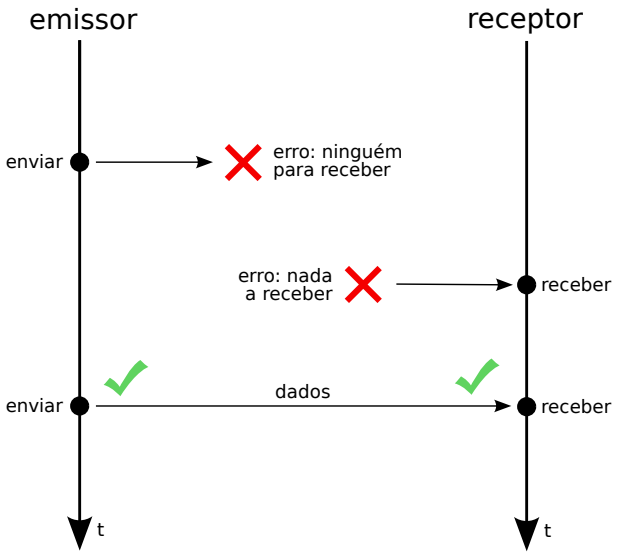

Figura 8.4: Comunicação assíncrona.

primitivas de comunicação recebem um parâmetro adicional, o *prazo*: *enviar (dados, destino, prazo)* e *receber (dados, origem, prazo)*. A Figura 8.5 ilustra duas situações em que ocorre esse comportamento.

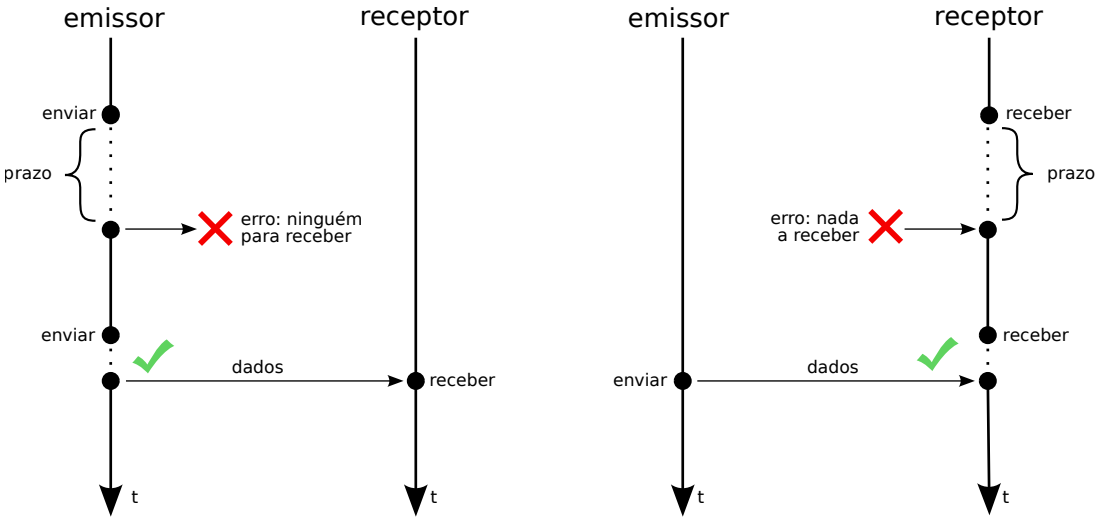

Figura 8.5: Comunicação semissíncrona.

8.3.3 Formato de envio

A informação enviada pelo emissor ao receptor pode ser vista basicamente de duas formas: como uma **sequência de mensagens** independentes, cada uma com seu próprio conteúdo, ou como um **fluxo sequencial** e contínuo de dados, imitando o comportamento de um arquivo com acesso sequencial.

Na abordagem baseada em mensagens, cada mensagem consiste de um pacote de dados que pode ser tipado ou não. Esse pacote é recebido ou descartado pelo receptor em sua íntegra; não existe a possibilidade de receber "meia mensagem" (Figura 8.6). Exemplos de sistema de comunicação orientados a mensagens incluem as *message queues* do UNIX e os protocolos de rede IP e UDP, apresentados na Seção 9.

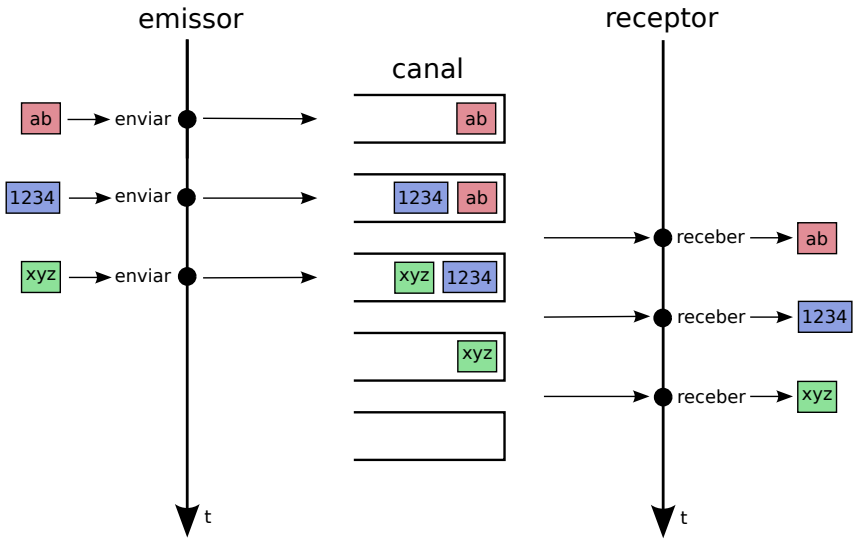

Figura 8.6: Comunicação baseada em mensagens.

Caso a comunicação seja definida como um fluxo contínuo de dados, o canal de comunicação é visto como o equivalente a um arquivo: o emissor "escreve" dados nesse canal, que serão "lidos" pelo receptor respeitando a ordem de envio dos dados. Não há separação lógica entre os dados enviados em operações separadas: eles podem ser lidos byte a byte ou em grandes blocos a cada operação de recepção, a critério do receptor. A Figura 8.7 apresenta o comportamento dessa forma de comunicação.

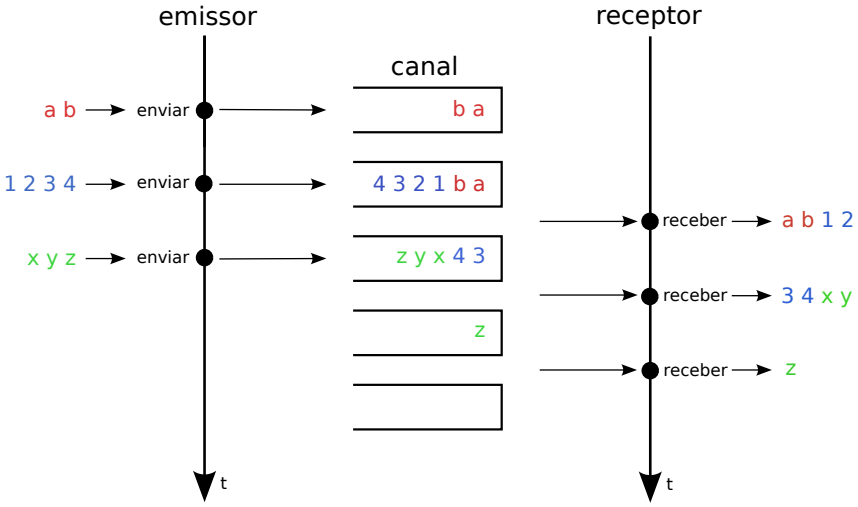

Figura 8.7: Comunicação baseada em fluxo de dados.

Exemplos de sistemas de comunicação orientados a fluxo de dados incluem os *pipes* do UNIX e o protocolo de rede TCP/IP (este último é normalmente classificado como *orientado a conexão*, com o mesmo significado). Nestes dois exemplos a analogia com o conceito de arquivos é tão forte que os canais de comunicação são identificados por descritores de arquivos e as chamadas de sistema `read` e `write` (normalmente usadas com arquivos) são usadas para enviar e receber os dados. Esses exemplos são apresentados em detalhes na Seção 9.

8.3.4 Capacidade dos canais

O comportamento síncrono ou assíncrono de um canal de comunicação pode ser afetado pela presença de *buffers* que permitam armazenar temporariamente os dados em trânsito, ou seja, as informações enviadas pelo emissor e que ainda não foram recebidas pelo receptor. Em relação à capacidade de *buffering* do canal de comunicação, três situações devem ser analisadas:

Capacidade nula ($n = 0$): neste caso, o canal não pode armazenar dados; a comunicação é feita por transferência direta dos dados do emissor para o receptor, sem cópias intermediárias. Caso a comunicação seja síncrona, o emissor permanece bloqueado até que o destinatário receba os dados, e vice-versa. Essa situação específica (comunicação síncrona com canais de capacidade nula) implica em uma forte sincronização entre as partes, sendo por isso denominada *Rendez-Vous* (termo francês para "encontro"). A Figura 8.3 ilustra dois casos de *Rendez-Vous*. Por outro lado, a comunicação assíncrona torna-se inviável usando canais de capacidade nula (conforme discutido na Seção 8.3.2).

Capacidade infinita ($n = \infty$): o emissor sempre pode enviar dados, que serão armazenados no *buffer* do canal enquanto o receptor não os consumir. Obviamente essa situação não existe na prática, pois todos os sistemas de computação têm capacidade de memória e de armazenamento finitas. No entanto, essa simplificação é útil no estudo dos algoritmos de comunicação e sincronização, pois torna menos complexas a modelagem e análise dos mesmos.

Capacidade finita ($0 < n < \infty$): neste caso, uma quantidade finita (n) de dados pode ser enviada pelo emissor sem que o receptor os consuma. Todavia, ao tentar enviar dados em um canal já saturado, o emissor poderá ficar bloqueado até surgir espaço no *buffer* do canal e conseguir enviar (comportamento síncrono) ou receber um retorno indicando o erro (comportamento assíncrono). A maioria dos sistemas reais opera com canais de capacidade finita.

Para exemplificar esse conceito, a Figura 8.8 apresenta o comportamento de duas tarefas trocando dados através de um canal de comunicação com capacidade para duas mensagens e comportamento bloqueante.

8.3.5 Confiabilidade dos canais

Quando um canal de comunicação transporta todos os dados enviados através dele para seus receptores, respeitando seus valores e a ordem em que foram enviados, ele é chamado de **canal confiável**. Caso contrário, trata-se de um **canal não-confiável**. Há várias possibilidades de erros envolvendo o canal de comunicação, ilustradas na Figura 8.9:

Perda de dados: nem todos os dados enviados através do canal chegam ao seu destino; podem ocorrer perdas de mensagens (no caso de comunicação orientada a mensagens) ou de sequências de bytes, no caso de comunicação orientada a fluxo de dados.

Perda de integridade: os dados enviados pelo canal chegam ao seu destino, mas podem ocorrer modificações em seus valores devido a interferências externas.

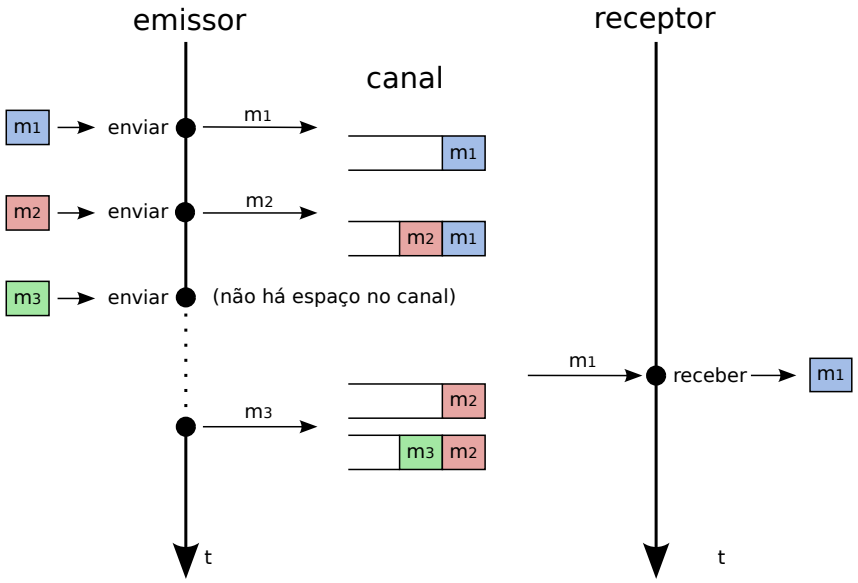

Figura 8.8: Comunicação bloqueante usando um canal com capacidade 2.

Perda da ordem: todos os dados enviados chegam íntegros ao seu destino, mas o canal não garante que eles serão entregues na ordem em que foram enviados. Um canal em que a ordem dos dados é garantida é denominado **canal FIFO** ou **canal ordenado**.

Os canais de comunicação usados no interior de um sistema operacional para a comunicação entre processos ou *threads* locais são geralmente confiáveis, ao menos em relação à perda ou corrupção de dados. Isso ocorre porque a comunicação local é feita através da cópia de áreas de memória, operação em que não há risco de erros. Por outro lado, os canais de comunicação entre computadores distintos envolvem o uso de tecnologias de rede, cujos protocolos básicos de comunicação são não-confiáveis (como os protocolos *Ethernet*, IP e UDP). Mesmo assim, protocolos de rede de nível mais elevado, como o TCP, permitem construir canais de comunicação confiáveis.

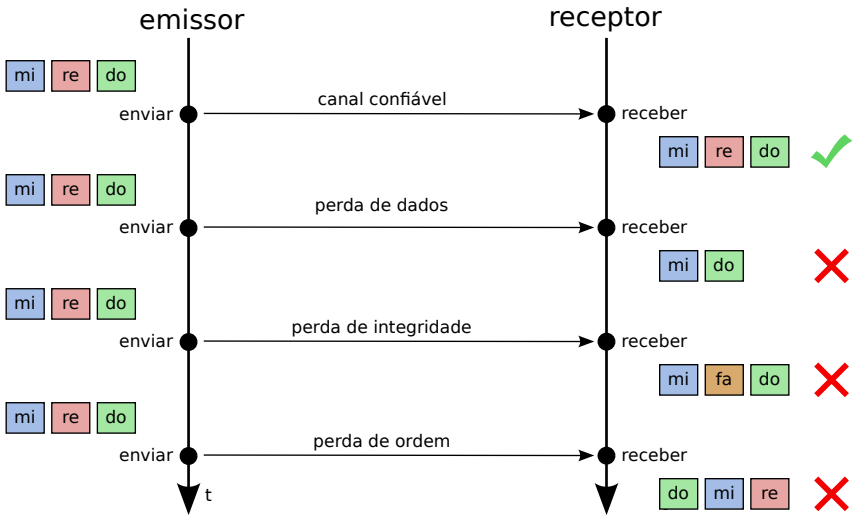

Figura 8.9: Comunicação com canais não confiáveis.

8.3.6 Número de participantes

Nas situações de comunicação apresentadas até agora, cada canal de comunicação envolve apenas um emissor e um receptor. No entanto, existem situações em que uma tarefa necessita comunicar com várias outras, como por exemplo em sistemas de *chat* ou mensagens instantâneas (IM – *Instant Messaging*). Dessa forma, os mecanismos de comunicação também podem ser classificados de acordo com o número de tarefas participantes:

1:1: quando exatamente um emissor e um receptor interagem através do canal de comunicação; é a situação mais frequente, implementada por exemplo nos *pipes* UNIX e no protocolo TCP.

M:N: quando um ou mais emissores enviam mensagens para um ou mais receptores. Duas situações distintas podem se apresentar neste caso:

- Cada mensagem é recebida por **apenas um receptor** (em geral aquele que pedir primeiro); neste caso a comunicação continua sendo ponto-a-ponto, através de um canal compartilhado. Essa abordagem é conhecida como *mailbox* (Figura 8.10), sendo implementada nas *message queues* do UNIX e Windows e também nos *sockets* do protocolo UDP. Na prática, o *mailbox* funciona como um *buffer* de dados, no qual os emissores depositam mensagens e os receptores as consomem.
- Cada mensagem é recebida por **vários receptores** (cada receptor recebe uma cópia da mensagem). Essa abordagem, ilustrada na Figura 8.11, é conhecida como *barramento de mensagens* (*message bus*), *canal de eventos* ou ainda *canal publish-subscribe*. Na área de redes, essa forma de comunicação é chamada de *difusão de mensagens* (*multicast*). Exemplos dessa abordagem podem ser encontrados no D-Bus [Free Desktop, 2018], o barramento de mensagens usado nos ambientes de *desktop* Gnome e KDE, e no COM, a infraestrutura de comunicação interna entre componentes nos sistemas Windows [Microsoft, 2018].

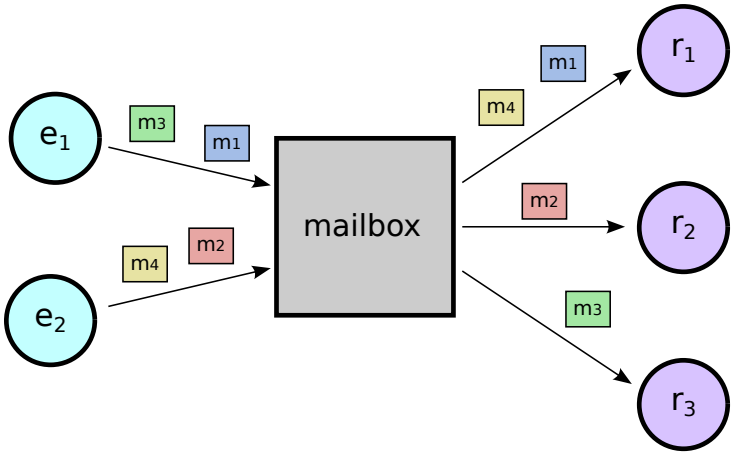

Figura 8.10: Comunicação M:N através de um *mailbox*.

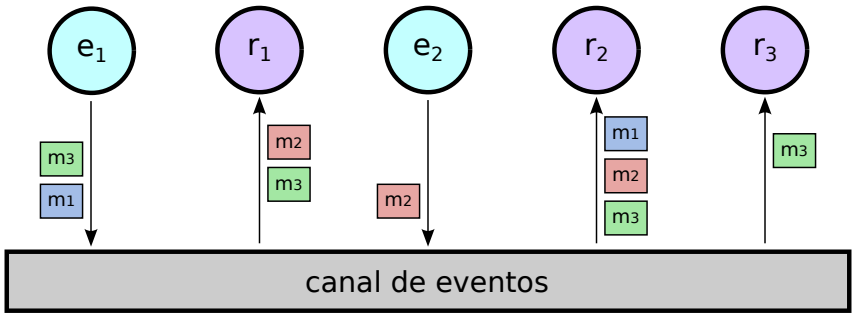

Figura 8.11: Comunicação M:N através de um barramento de mensagens.

Exercícios

1. Quais são as vantagens e desvantagens das abordagens a seguir, sob as óticas do sistema operacional e do programador de aplicativos?

   (a) comunicação bloqueante ou não-bloqueante

   (b) canais com *buffering* ou sem *buffering*

   (c) comunicação por mensagens ou por fluxo

   (d) mensagens de tamanho fixo ou variável

   (e) comunicação 1:1 ou M:N

2. Explique como processos que comunicam por troca de mensagens se comportam em relação à capacidade do canal de comunicação, considerando as semânticas de chamada síncrona e assíncrona.

3. Sobre as afirmações a seguir, relativas mecanismos de comunicação, indique quais são incorretas, justificando sua resposta:

   (a) A comunicação indireta (por canais) é mais adequada para sistemas distribuídos.

   (b) Canais com capacidade finita somente são usados na definição de algoritmos, não sendo implementáveis na prática.

   (c) Na comunicação direta, o emissor envia os dados diretamente a um canal de comunicação.

   (d) Na comunicação por fluxo, a ordem dos dados enviados pelo emissor é mantida do lado receptor.

   (e) Na comunicação por troca de mensagens, o núcleo transfere pacotes de dados do processo emissor para o processo receptor.

4. Sobre as afirmações a seguir, relativas à sincronização na comunicação entre processos, indique quais são incorretas, justificando sua resposta:

   (a) Na comunicação semi-bloqueante, o emissor espera indefinidamente pela possibilidade de enviar os dados.

(b) Na comunicação síncrona ou bloqueante, o receptor espera até receber a mensagem.

(c) Um mecanismo de comunicação semi-bloqueante com prazo $t = \infty$ equivale a um mecanismo bloqueante.

(d) Na comunicação síncrona ou bloqueante, o emissor retorna uma mensagem de erro caso o receptor não esteja pronto para receber a mensagem.

(e) Se o canal de comunicação tiver capacidade nula, emissor e receptor devem usar mecanismos não-bloqueantes.

(f) A comunicação não-bloqueante em ambos os participantes só é viável usando canais de comunicação com *buffer* não-nulo.

Referências

Free Desktop. D-Bus - a message bus system. https://www.freedesktop.org/wiki/Software/dbus/, 2018.

Gnome. Gnome: the free software desktop project. http://www.gnome.org, 2005.

KDE. KDE desktop project. http://www.kde.org, 2005.

Microsoft. Component object model (COM). https://docs.microsoft.com/en-us/windows/desktop/com/component-object-model--com--portal, 2018.