Aula 13

José A. Cardoso e Cunha
DI-FCT/UNL

Este texto resume o conteúdo da aula teórica.

1 Objectivo

Objectivo da aula: suportes físicos para a comunicação entre processos: arquitecturas de memória partilhada e de memória distribuída. Comunicação por memória partilhada. Realização da memória partilhada ao nível do hardware e do SO. Problemas de interferência entre processos concorrentes. Exemplos. Problema de exclusão mútua e regiões críticas. Classificação de soluções e suas limitações: inibição das interrupções de programa, algoritmo de Dekker, instrução TEST-AND-SET

2 Ficheiros e *pipes*

Até agora só estudámos dois potenciais mecanismos de comunicação entre processos: o ficheiro e o *pipe*, no caso do Unix, que tem sido o nosso SO de referência. As comparações são:

Ficheiro normal:

- tem suporte fisico em zonas de disco
- tem um nome global pathname
- pode ser acedido como um stream, em modo sequencial ou directo, via o byte offset
- a leitura nao é destrutiva
- a leitura - read de um ficheiro vazio devolve 0
- a escrita num ficheiro nunca bloqueia o processo
- a informacao do ficheiro persiste, mesmo que nao haja processos com canais abertos para ele

- nao é um mecanismo de comunicacao entre processos concorrentes pelo que a leitura e escrita nao têm acções de sincronização de processos

Pipe:

- tem suporte fisico em memoria e nao em disco
- nao tem um nome global
- só pode ser acedido de forma fifo, em modo sequencial, e nao se pode manipular o byte offset
- a leitura é destrutiva - consome
- a leitura - read de um pipe vazio bloqueia o processo (desde que haja pelo menos um canal aberto para escrita)
- a escrita num pipe pode bloquear o processo se o pipe estiver cheio
- a informacao do pipe nao persiste, logo que haja processos com canais abertos para ele: o buffer que o representa em memoria é descartado
- é usado para a comunicacao entre processos concorrentes, pelo que a leitura e a escrita têm sincronizacao implicitas, podendo desencadear o desbloqueio de processos que estejam aguardando.

Da comparação anterior ressalta que os ficheiros não são dispositivos adequados à comunicação entre processos concorrentes, embora, como se sabem, sejam habitualmente utilizados como suportes intermédios de informação persistente, produzida pela execução de um programa e, depois, consultada durante a execução de outro.

Quanto aos *pipes*, sabemos que, na sua realização no sistema Unix, só podem ser utilizados por processos da mesma 'famïlia', por serem herdados de pais para filhos e não terem nomes globais como os ficheiros. Esta limitação foi eliminada através de uma variante designada por *named pipes*, os quais têm um nome global associado, que surge listado nas directorias (são casos de 'ficheiros especiais').

Outra característica dos *pipes*, relevante do ponto de vista da comunicação, é que só transmitem fluxos (*streams*) de bytes, sem delimitação de separadores de mensagens. Se um processo escreve 10 bytes num pipe e depois escreve 15, um outro processo pode ler 25 bytes com uma única operação read. Se quisermos interpretar as duas escritas, como duas mensagens separadas, teremos de definir uma convenção adequada e implementar um protocolo que permita aos processos indicar os delimitadores de mensagens.

Na figura 1 ilustra-se outra característica dos *pipes*: a da cópia entre buffers do mapa de memória do processo e os buffers internos ao SO.

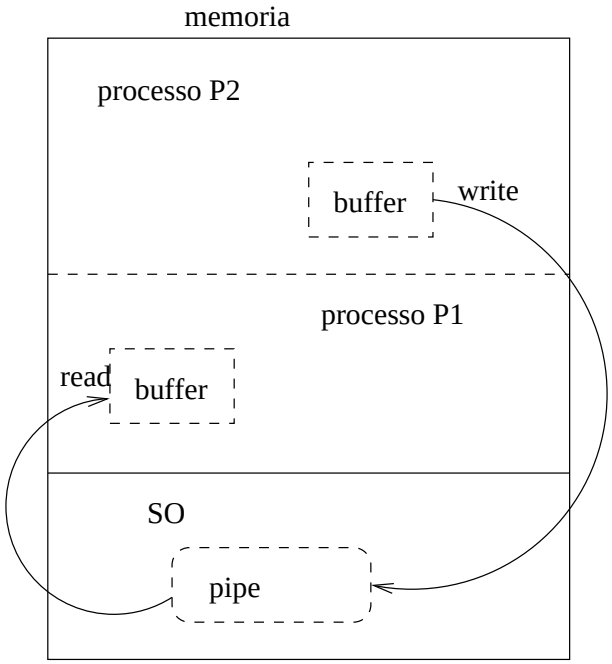

Figura 1: Implementação de *Pipes*.

Na figura, os buffers indicados no mapa dos processos são os que são especificados como argumentos das operações read e write, conforme estudámos anteriormente. A zona indicada para o pipe no SO é criada quando se cria o pipe e gerida pelo SO, para guardar temporariamente os bytes enviados para o pipe por um processo e ainda não recebidos por outro.

Esta forma de comunicação tem, claramente, uma vantagem: os programadores não têm de se preocupar com a sincronização de processos, quando os pipes estão cheios ou vazios, ou ainda quando dois processos concorrentes tentam aceder a um mesmo pipe. Tudo isto é gerido pelo SO. O preço que se paga é a sobrecarga de tempo e de espaço de memória adicional, associada à cópia de bytes ilustrada na figura.

3 Arquitecturas de sistemas de computadores

Para vermos alternativas de comunicação entre processos, consideremos as duas principais classes de arquitecturas de computadores, no que se refere ao mecanismo físico de comunicação suportado pelo *hardware* (2).

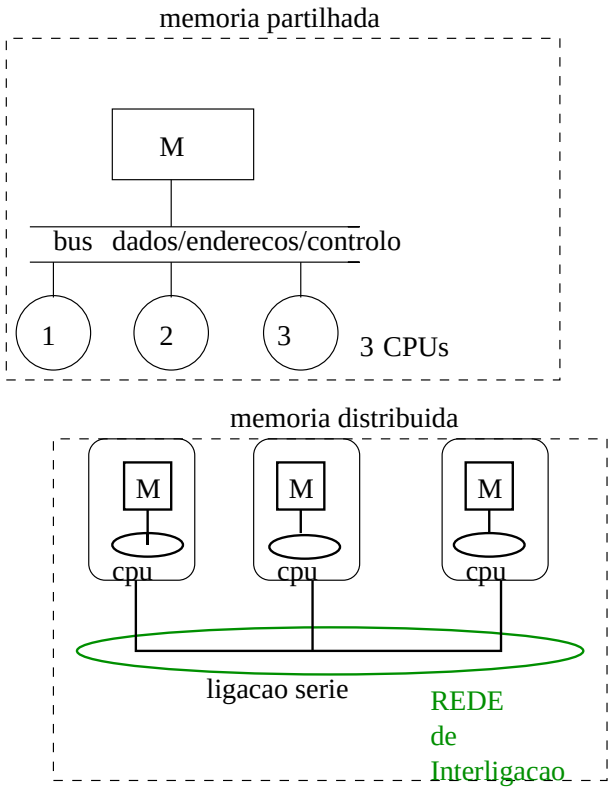

Figura 2: Arquitecturas de memória partilhada e de memória distribuída.

Na configuração de *memória partilhada*, é possível uma comunicação muito eficiente entre os programas possivelmente executando-se em distintos CPUs, em simultâneo. Esta comunicação faz-se através do acesso à memória central comum (RAM), através do bus de sistema. Qualquer CPU pode gerar um endereço real, colocá-lo no bus e pedir a leitura de uma célula de memória, num tempo da ordem das dezenas de nanosegundos. O controlo do acesso ao bus é sequencial, isto é, um CPU de cada vez, e é totalmente realizado por *hardware*.

Esta configuração abrange, como um caso particular, a arquitectura de von Neumann, de um único CPU. Neste caso, podemos ter múltiplos processos, sob o controlo de um SO de multiprogramação (como o Unix), acedendo também a regiões comuns de memória, que utilizam para comunicação, conforme veremos em pormenor, mais adiante.

No caso da configuração de *memória distribuída*, como se vê, temos

múltiplos pares de CPU e Memória, ligados entre si através de uma *rede* de comunicação. Esta última pode ser suportada pelas mais variadas tecnologias, pode ter topologias muito variadas, e pode abranger maiores ou menores escalas de distâncias, mas, em qualquer dos casos, a comunicação entre dois CPU só se pode fazer através do envio de sequências de bits, organizados em bytes, organizados em mensagens, que são necessariamente transmitidas através do suporte físico da rede. Este suporte não permite o acesso 'directo' de um CPU à memória que está associada a outro qualquer CPU: cada CPU só tem acesso à sua memória (dita 'local'), através do seu bus próprio. Existem múltiplas variantes desta configuração, desde as arquitecturas encapsuladas numa só caixa física, até às formadas por computadores individuais, ligados à escala de um edificio, ou até à de computadores ligados à escala internacional. O estudo dessas variantes faz-se na disciplina de Redes de Computadores.

Por agora, só nos interessa compreender que existem dois tipos radicalmente diferentes de suporte físicos para a comunicação: por memória fisicamente partilhada ou por memória distribuída.

4 Modelos de comunicação por memória partilhada

Atendendo ao dito antes, não admira que haja duas principais categorias de modelos de comunicação entre processos: os baseados em memória partilhada e os baseados em memória distribuída.

Nos modelos de memória partilhada, podemos ter o esquema da figura 3.

Na figura, a estrutura partilhada é acedida directamente pelos dois processos, como se fizesse parte integrante dos seus espaços de endereçamento. Dependendo do nível a que o modelo é implementado, a estrutura partilhada pode ser uma variável global, como em programas Pascal, ou pode ser um objecto de mais alto nível, até habitualmente designado por *blackboard* (quadro negro), para dar a ideia de que é acessível de igual modo por todos os processos.

Este modelo parece, à partida, muito interessante para realizar a comunicação entre processos concorrentes, por exemplo, seja através de buffers entre produtores e consumidores, seja através de bases de dados entre leitores e escritos, ou até através de filas de pedidos entre clientes e servidores, ou reservatórios de recursos (buffers) partilhados.

Como será realizado tal modelo? A figura 4 adianta mais alguns pormenores sobre isto.

Note que a figura representa os mapas de memória virtual dos processos, isto é, as regiões ilustradas podem não estar fisicamente contíguas em memória real. Vê-se que, nos mapas de memória dos processos, existem duas zonas

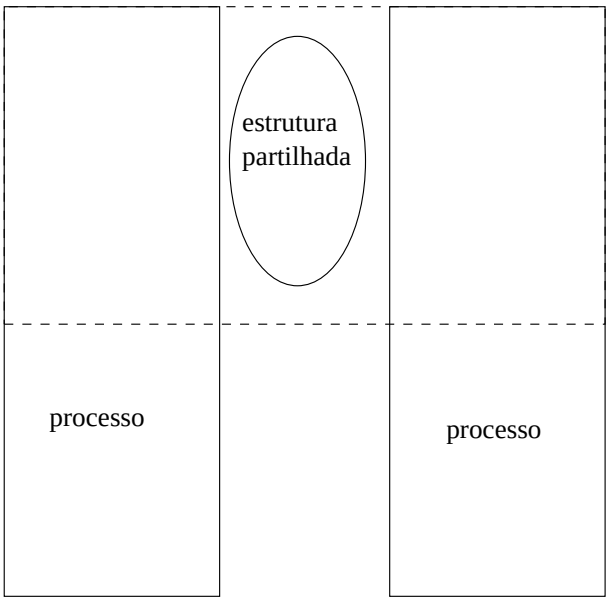

Figura 3: Modelos de comunicação por memória partilhada

bem distintas: uma zona privada, à qual só o respectivo processo pode aceder (como é habitual quando se cria um processo, por exemplo, através de *fork*); uma zona partilhada, que é realizada, fazendo corresponder, em cada um dos mapas de memória dos processos, uma zona de endereços que estão associados, através do mapa *hardware* de transformação de endereços (cf. tabela de páginas), aos endereços reais que representam, em memória real, a estrutura partilhada.

Para implementar um tal mecanismo, admite-se (por agora... isto é, em ASC2) que a arquitectura do sistema é da categoria *memória fisicamente partilhada*. Pois, caso contrário, não seria viável suportar a transformação de endereços, acima mencionada, completamente por *hardware*.

A figura 5 dá um exemplo mais concreto da implementação da memória partilhada.

No mapa de memória virtual de cada processo 'algures' entre o *heap* e a pilha de execução, o SO reserva uma página (ou mais) virtual, correspondendo aos endereços do mapa que se pretendem fazer corresponder a endereços reais de uma (ou mais) página física de memória. Esta correspondência é, como se sabe, efectuada com base no hardware de transformação de endereços que suporta as tabelas de páginas.

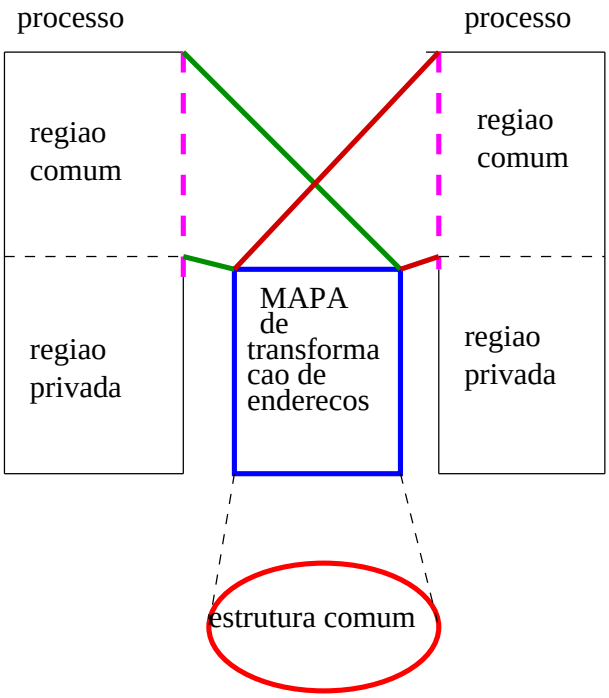

Figura 4: Comunicação por memória partilhada

Para dar acesso, a mais do que um processo, a uma mesma página física de memória, tudo o que o SO tem de fazer é inicializar, nas respectivas entradas das duas tabelas de páginas, o mesmo endereço real, base dessa página física. Os endereços virtuais nos mapas dos dois processos não precisam de ser os mesmos.

Uma vez inicializadas estas estruturas, qualquer referência a um endereço de uma página virtual partilhada, em qualquer dos processos, é automaticamente transformada, pelo *hardware* de endereçamento, no endereço real da posição correspondente da página física comum em RAM.

Num sistema com múltiplos CPU, o acesso pode ser realmente simultâneo! Pelo contrário, num sistema com um só CPU e um SO com multiprogramação, o acesso é 'virtualmente instantâneo': quando um processo está em execução, pode aceder à memória partilhada. Quando esse processo terminar a execução ou perder o controlo do CPU (por se bloquear ou expirar a sua fatia de tempo atribuída), outro processo, que tenha acesso à mesma página física, pode aceder-lhe, observando a informação lá entretanto colocada pelo primeiro

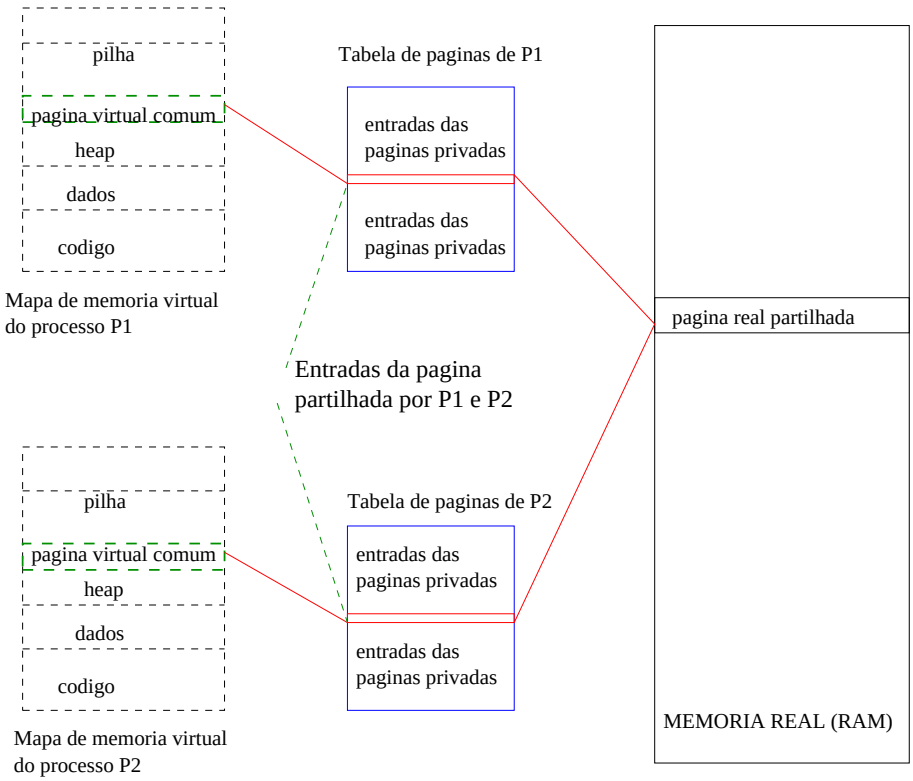

Figura 5: Implementação da memória partilhada, a nível do SO

processo.

A grande vantagem deste mecanismo é que, sendo praticamente totalmente suportada pelo hardware de transformação de endereços e não envolvendo cópias de bytes (ao contrário dos pipes), é muito eficiente para suportar a comunicação entre processos.

A principal desvantagem ver-se-á mais adiante.

5 Modelos de comunicação por memória distribuída

Por comunicação por *memória distribuída* entendemos aqueles modelos nos quais existe uma transferência explícita de informação, do mapa de memória de um processo, para o mapa de memória de outro. Isto significa que há sempre cópia de bytes entre os dois espaços. O esquema abstracto destes modelos ilustra-se na figura 6.

Como se ilustra, este modelo é uma espécie de 'generalização' das operações de entrada e saída, e envolve, como intermediário, alguma meio de comunicação, gerido pelo SO, que pode armazenar dados temporariamente, em zonas de memória próprias, como acontecia, por exemplo, no caso dos *pipes*. Dependendo do tipo de meio de comunicação e do modelo suportado, a interacção entre os processos pode ser de diversos tipos: modelo de tipo de 'conversa telefónica', modelo de tipo de 'envio de telegramas', ou modelo de tipo de 'difusão de mensagens'.

Na figura 7 ilustra-se um cenário de implementação de comunicação por mensagens, num sistema de memória física partilhada.

Na figura 8 ilustra-se um cenário de implementação de comunicação por mensagens, num sistema de memória física distribuída.

Os problemas associados à comunicação e à execução de programas em redes de computadores são o objecto da disciplina de Redes de Computadores. Nestes sistemas, a comunicação faz-se por mensagens, pois não se dispõe de memória física partilhada.

Em ASC2, admite-se sempre uma arquitectura de memória partilhada, tenha um ou múltiplos CPU. Nota-se que, para estas arquitecturas, o programador pode optar por modelos de comunicação entre processos, baseados em memória partilhada ou em mensagens, desde que o SO os disponibilize através das suas chamadas ao sistema.

Nas aulas seguintes, vamos estudar os problemas de cada um destes modelos, para a programação de aplicações compostas por processos concorrentes.

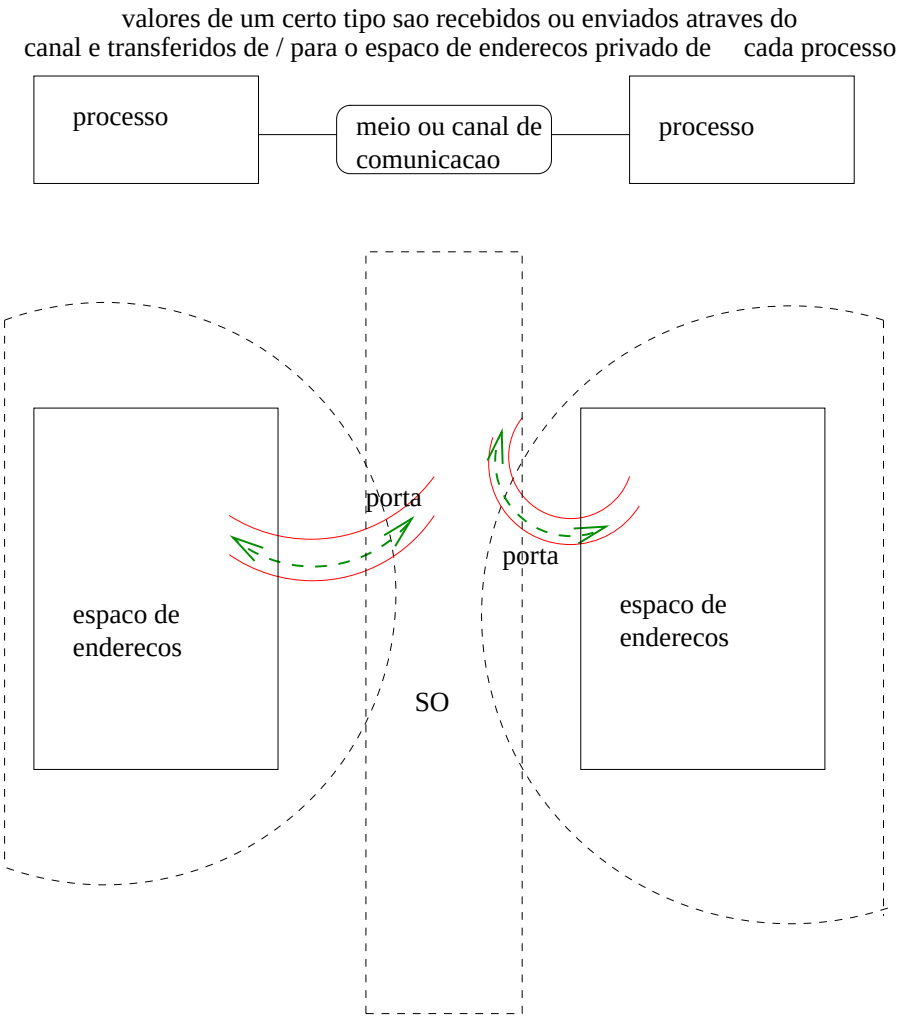

Figura 6: Memória distribuída

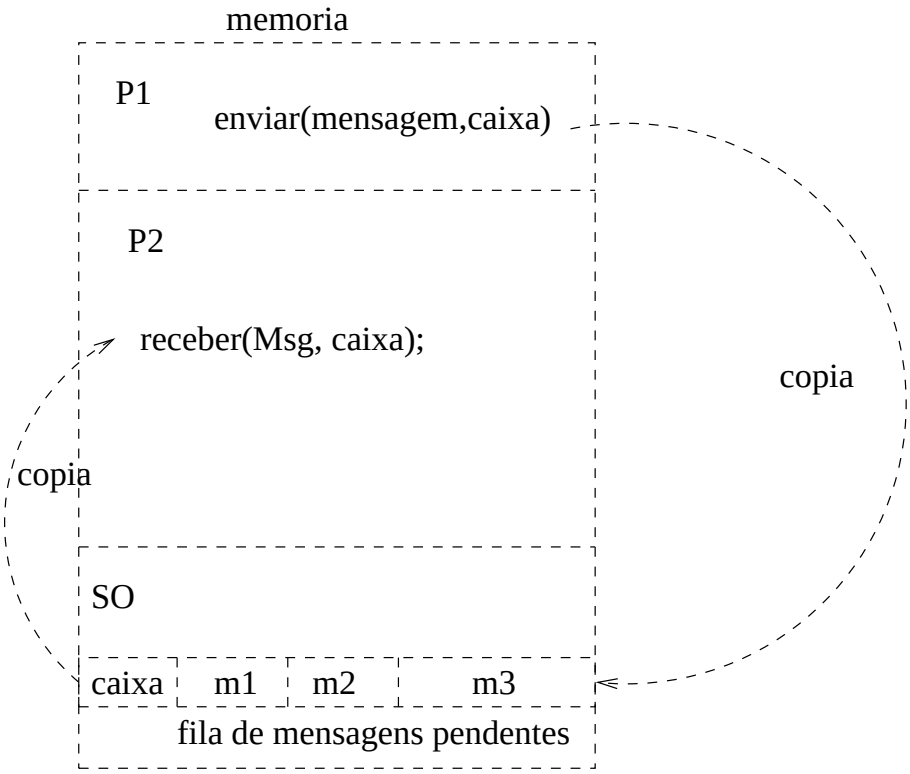

Figura 7: Mensagens

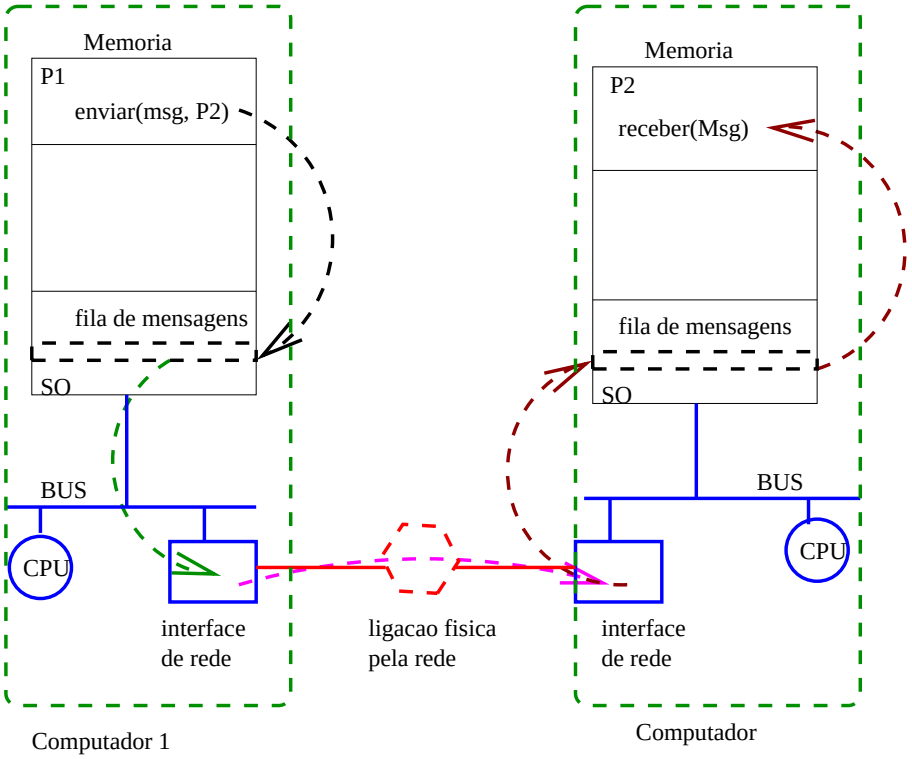

Figura 8: Mensagens