

Sistemas Operacionais

Mensagens e Outros Processos de Comunicação

Norton Trevisan Roman
Marcelo Morandini
Jó Ueyama

Apostila baseada nos trabalhos de Kalinka Castelo Branco, Antônio Carlos Sementille, Paula Prata e nas transparências fornecidas no site de compra do livro "Sistemas Operacionais Modernos"

Semáforos e Monitores

- Projetados para exclusão mútua em processadores que compartilham algum espaço de memória
 - Não funcionam com sistemas distribuídos, em que cada processador (ou grupo de processadores) possui sua própria memória, se comunicando via rede.
 - Como passar informação entre diferentes máquinas?

Mensagens

- Os mecanismos já considerados exigem do S.O. somente a sincronização
 - Asseguram a exclusão mútua, mas não garantem um controle sobre as operações desempenhadas sobre o recurso.
 - Deixam para o programador a comunicação de mensagens através da memória compartilhada
- A troca de mensagens é um mecanismo de comunicação e sincronização
 - Exige do S.O., tanto a sincronização quanto a comunicação entre os processos.
 - É um mecanismo mais elaborados de comunicação e sincronização entre processos

Mensagens

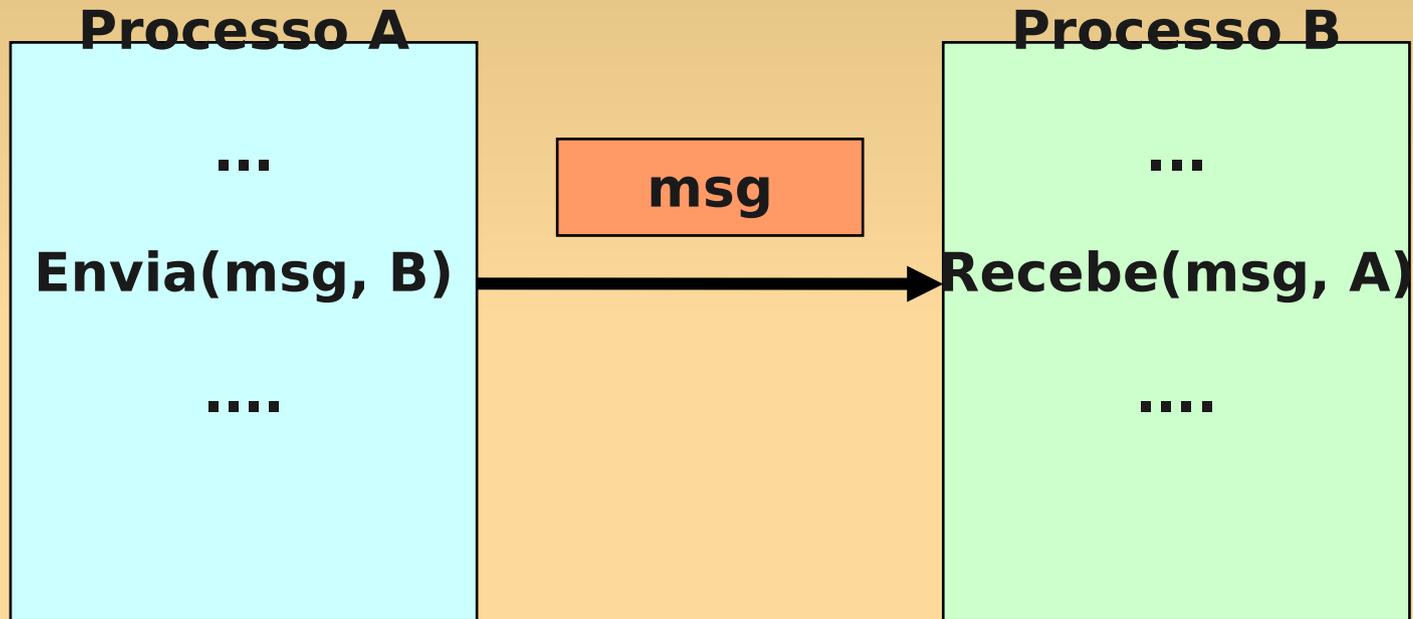
- Esquema de troca de mensagens:
 - Os processos enviam e recebem mensagens, em vez de ler e escrever em variáveis compartilhadas.
 - Sincronização entre processos:
 - Garantida pela restrição de que uma mensagem só poderá ser recebida depois de ter sido enviada.
 - A transferência de dados de um processo para outro, após ter sido realizada a sincronização, estabelece a comunicação.



Mensagens

- Possui duas primitivas:
 - Send
 - `send(destino, &mensagem);`
 - Receive
 - `receive(fonte, &mensagem);`
 - Se não houver mensagem disponível, o receptor pode:
 - Bloquear, até que haja alguma mensagem (Blocked)
 - Retornar com mensagem de erro (Unblocked)
- Implementadas como chamadas ao sistema

Mensagens – Primitivas



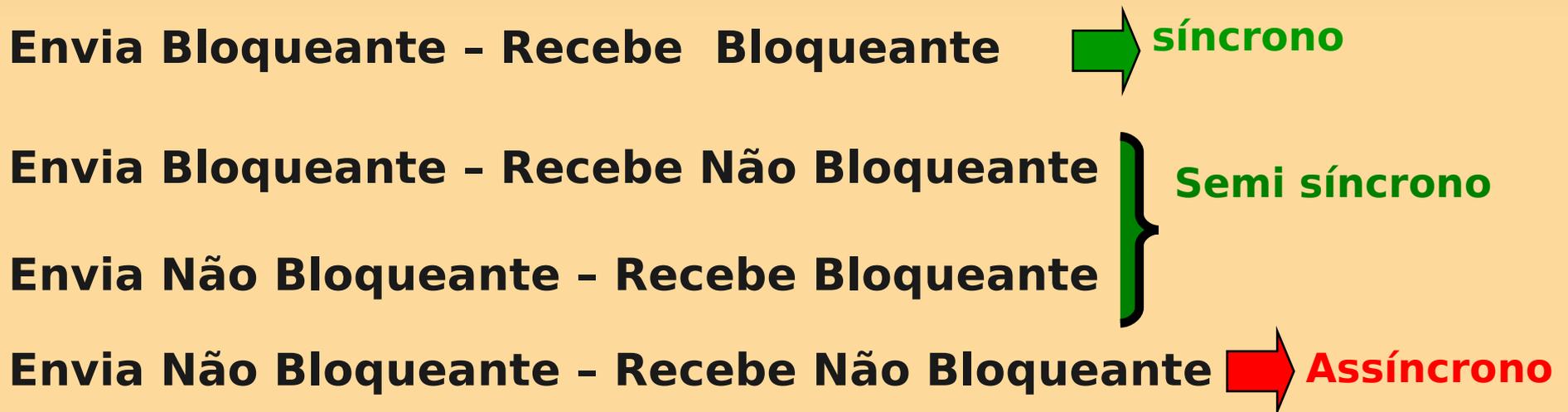
O fato de um somente poder receber a mensagem após o outro enviar garante a sincronia entre os processos

Mensagens – Primitivas

- As primitivas podem ser de dois tipos:
 - Bloqueantes: quando o processo que a executar ficar bloqueado até que a operação seja bem sucedida
 - Quando ocorrer a entrega efetiva da mensagem ao processo destino, no caso da emissão
 - Quando ocorrer o recebimento da mensagem pelo processo destino, no caso de recepção).
 - Não bloqueantes: quando o processo que executar a primitiva, continuar sua execução normal, independentemente da entrega ou do recebimento efetivo da mensagem pelo processo destino.

Mensagens – Primitivas

- Combinação de Primitivas
 - Existem quatro maneiras de se combinar as primitivas de troca de mensagens:



Mensagens – Primitivas

- Exemplo de Comunicação usando Troca de Mensagens

```
Program emissor_receptor;
Type msg=...;
Var mensagem : msg;
Begin /* inicio do programa principal */
  Cobegin /* inicio dos processos concorrentes */
    Begin /* processo emissor - E */
      repeat
        ...;
        produz uma mensagem;
        Send(mensagem, R);
      until false
    End;

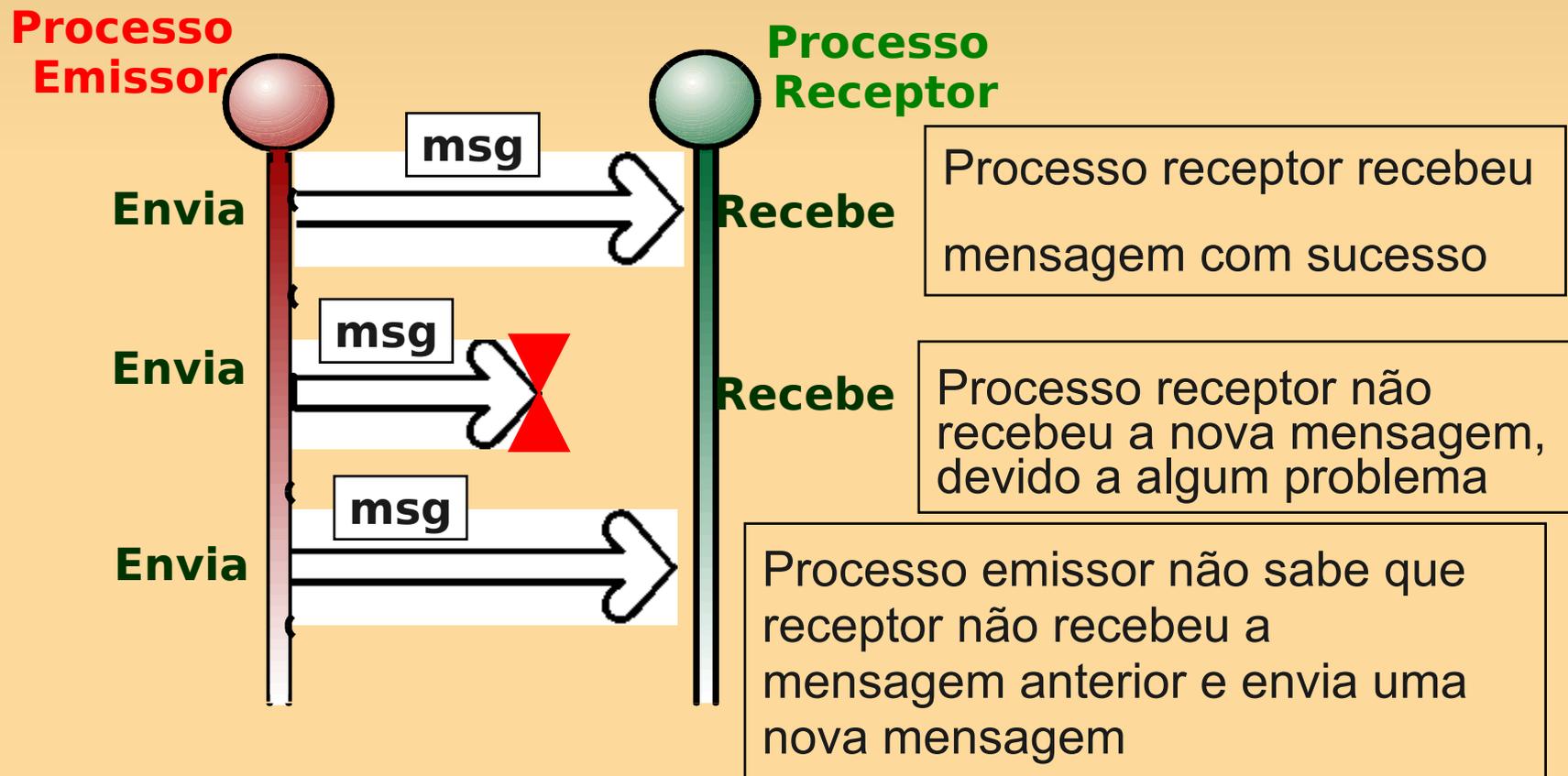
    Begin /* processo receptor - R */
      repeat
        Receive(mensagem, E);
        Consume a mensagem;
        ...;
      until false
    End
  Coend
End.
```

Mensagens

- Os sistemas de troca de mensagens possuem alguns problemas e estudos de projetos interessantes
 - Principalmente quando os processos comunicantes estão em máquinas diferentes, conectadas por uma rede de comunicação.
 - Os principais são:
 - Perda de mensagens
 - Perda de reconhecimento
 - Nomeação de Processos
 - Autenticação
 - Estudos de Projeto para quando emissor e receptor estiverem na mesma máquina

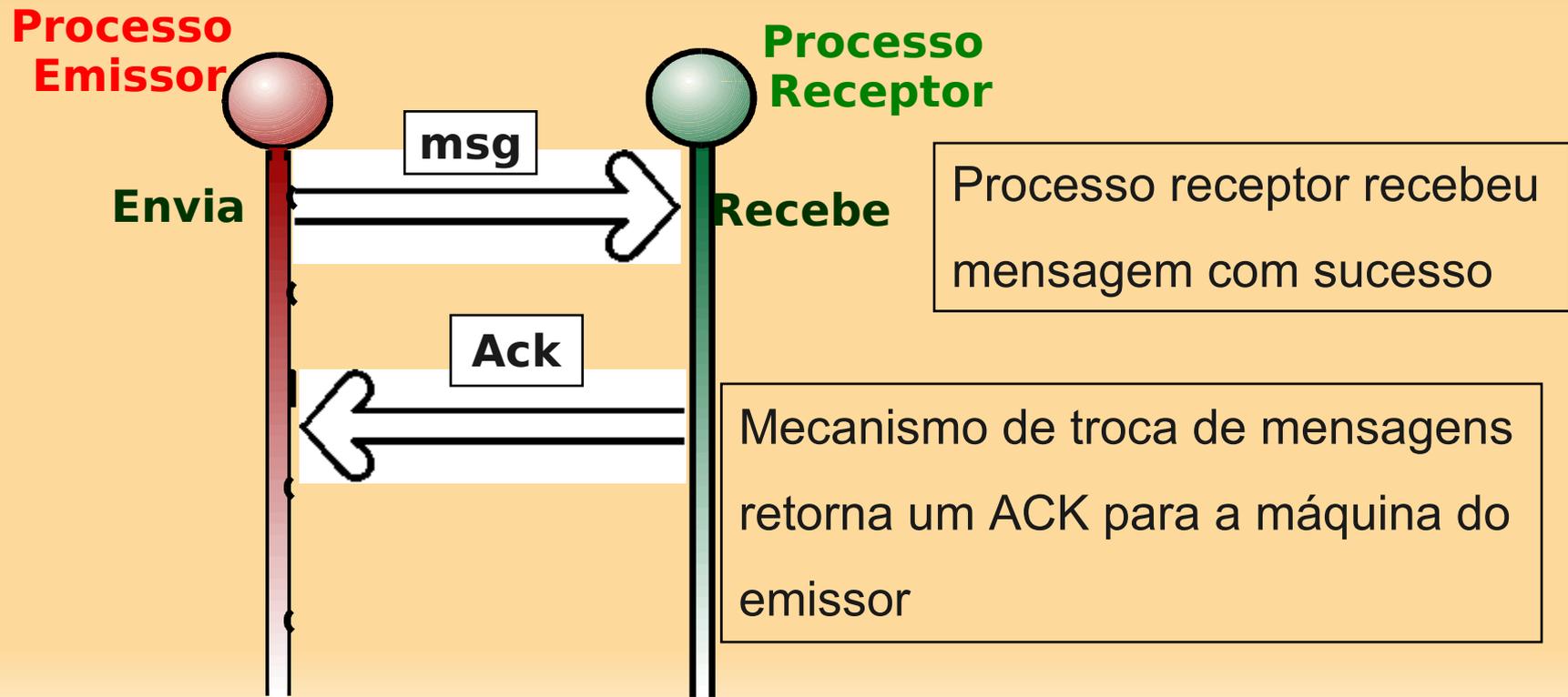
Mensagens – Problemas

- Perda de Mensagens
 - Podem ocorrer por falhas na rede, por exemplo



Perda de Mensagens

- Possível solução:
 - O receptor, ao receber uma nova mensagem, envia uma mensagem especial de reconhecimento (ACK).
 - Se o emissor não receber um ACK dentro de um determinado intervalo de tempo, deve retransmitir a mensagem.



Perda de Mensagens

- Possível solução:
 - O que acontece se a mensagem é recebida corretamente, mas o ACK é perdido?

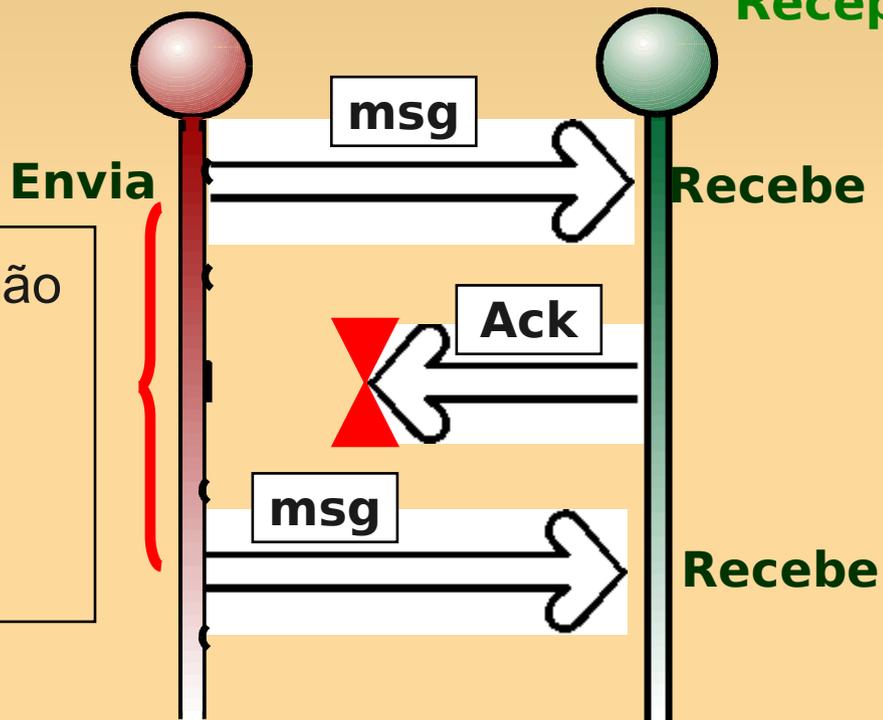
Perda de Reconhecimento

- Possível solução:
 - O que acontece se a mensagem é recebida corretamente, mas o reconhecimento (ACK) é perdido?
 - O emissor retransmitirá a mensagem
 - O receptor irá recebê-la duas vezes

Perda de Reconhecimento

Processo Emissor

Processo Receptor



Processo emissor não recebeu o ACK e enviou novamente a mesma mensagem (esgotamento do tempo de espera do ACK)

Processo receptor recebeu mensagem com sucesso, e enviou um ACK

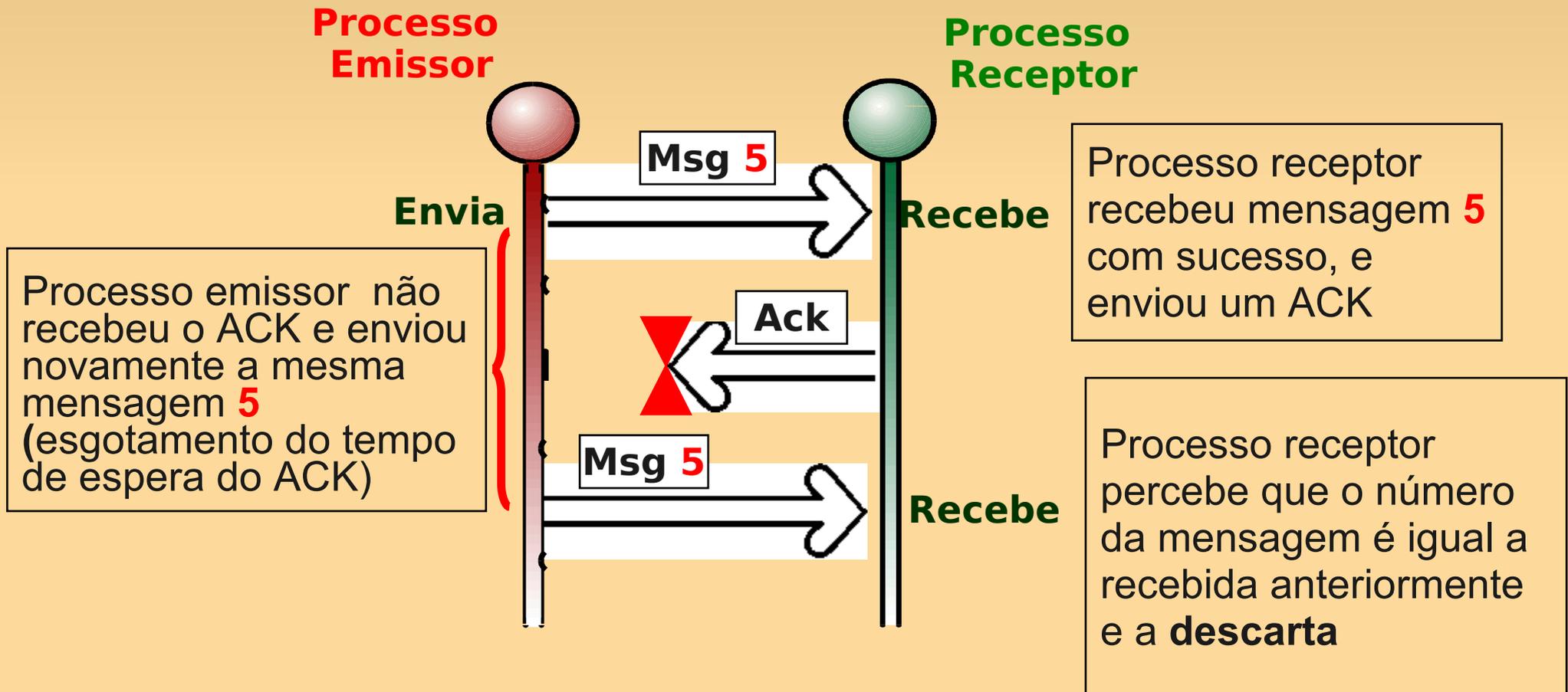
Processo receptor não sabe que o emissor não recebeu o ACK e considera a mensagem recebida como nova (duplicação)

Perda de Reconhecimento

- Possível solução:
 - É essencial que o receptor seja capaz de distinguir uma nova mensagem de uma retransmissão
 - Numerar as mensagens:
 - Se o receptor receber mensagem com o mesmo número que alguma anterior, sabe que é duplicata, ignorando-a.

Perda de Reconhecimento

- Solução:



Mensagens – Nomeação de Processos

- Os processos devem ser nomeados de maneira única, para que o nome do processo especificado no Send ou Receive não seja ambíguo.
 - Ex: processo@máquina (normalmente existe uma autoridade central que nomeia as máquinas).
- Quando o número de máquinas é muito grande:
 - processo@máquina.domínio.

Mensagens – Autenticação

- Como o cliente pode dizer que está se comunicando com o servidor real, e não um impostor?
 - Devemos permitir a comunicação e acessos apenas dos usuários autorizados.
 - Uma solução é criptografar as mensagens com uma chave conhecida apenas por usuários autorizados.

Mensagens

- Questões de estudo de projeto para quando emissor e receptor estão na mesma máquina
 - Desempenho:
 - Copiar mensagens de um processo para o outro é mais lento do que operações com semáforos e monitores;
 - Sugestão:
 - Limitar o tamanho das mensagens ao dos registradores
 - Passar as mensagens usando os registradores

Mensagens

Produtor/Consumidor

Assumimos que mensagens enviadas e não lidas são guardadas pelo S.O.

Usamos um número de mensagens igual ao do buffer

```
#define N 100                                     /* number of slots in the buffer */

void producer(void)
{
    int item;
    message m;                                     /* message buffer */

    while (TRUE) {
        item = produce_item();                    /* generate something to put in buffer */
        receive(consumer, &m);                    /* wait for an empty to arrive */
        build_message(&m, item);                  /* construct a message to send */
        send(consumer, &m);                       /* send item to consumer */
    }
}

void consumer(void)
{
    int item, i;
    message m;

    for (i = 0; i < N; i++) send(producer, &m); /* send N empties */
    while (TRUE) {
        receive(producer, &m);                    /* get message containing item */
        item = extract_item(&m);                  /* extract item from message */
        send(producer, &m);                       /* send back empty reply */
        consume_item(item);                       /* do something with the item */
    }
}
```

Mensagens – Mecanismos de Comunicação Síncronos

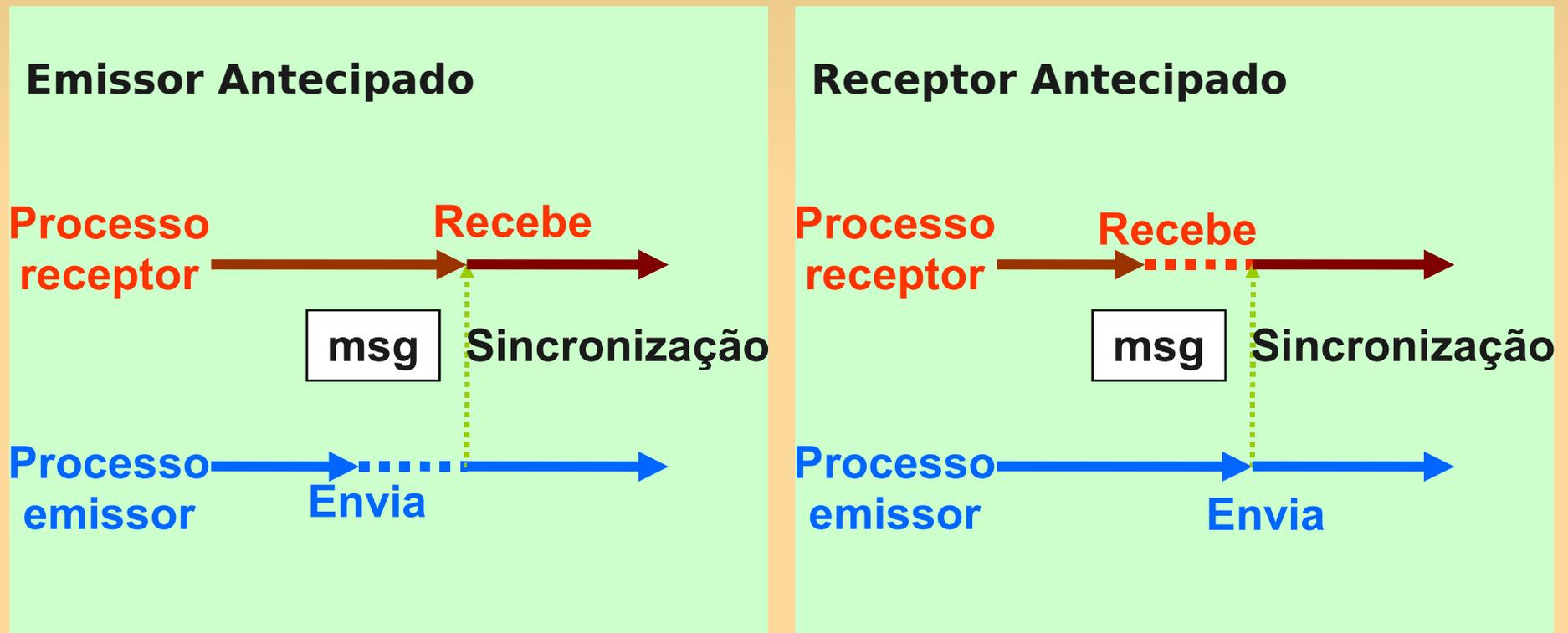
- Existem três mecanismos de comunicação síncrona mais importantes:
 - Rendez-vous 
 - Rendez-vous Estendido 
 - Chamada Remota de Procedimento
 - Ok, ok 

Comunicação Síncrona: Mecanismos

- Rendez-vous
 - Obtido através de primitivas Envia e Recebe bloqueantes colocadas em processos distintos;
 - A execução destas primitivas em tempos diferentes faz com que o processo que executar a primitiva antes do outro fique bloqueado até que ocorra a sincronização entre os dois processos, e a consecutiva transferência da mensagem;
 - Em seguida, ambos os processos continuarão seu andamento em paralelo.
 - Ex.: linguagem CSP.

Comunicação Síncrona: Mecanismos

- Rendez-vous



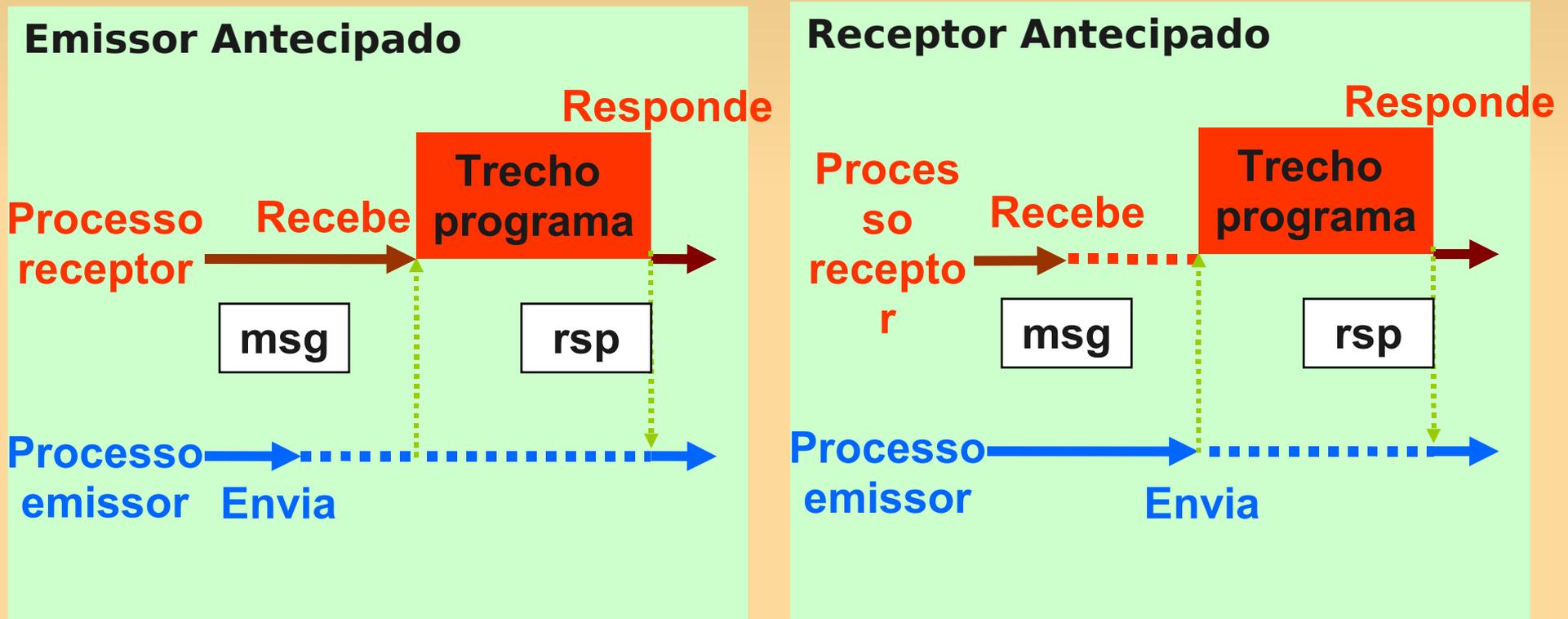
A transferência da mensagem é copiada diretamente do emissor ao receptor, sem uso de buffer intermediário.

Comunicação Síncrona: Mecanismos

- Rendez-vous Estendido
 - Caracteriza-se por apresentar uma estrutura de comunicação onde um processo consegue comandar a execução de um trecho de programa previamente estabelecido, pertencente a outro processo, envolvendo sincronização e, eventualmente, troca de mensagem.
 - O processo emissor deve esperar que o receptor chegue na parte do código desejada
 - Ex: linguagem ADA.

Comunicação Síncrona: Mecanismos

- Rendez-vous Estendido

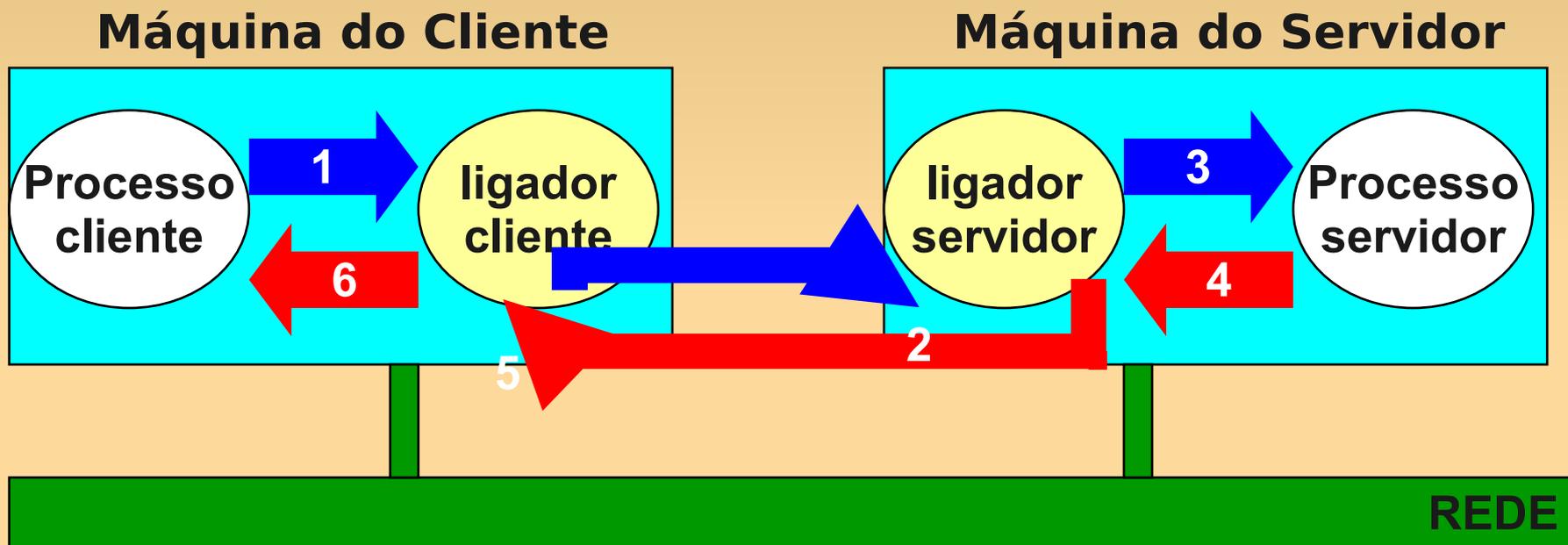


Comunicação Síncrona: Mecanismos

- Chamada Remota de Procedimento (RPC - Remote Procedure Call)
 - Apresenta uma estrutura de comunicação na qual um processo pode comandar a execução de um procedimento situado em outro processador.
 - O processo chamador deverá ficar bloqueado até que o procedimento chamado termine.
 - Tanto a chamada quanto o retorno podem envolver troca de mensagem, conduzindo parâmetros.
 - Ex: linguagem DP, C (via aplicativo rpcgen).

Comunicação Síncrona: Mecanismos

- Chamada Remota de Procedimento (RPC)



- (1) e (3) são chamadas de procedimento comuns
- (2) e (5) são mensagens
- (4) e (6) são retornos de procedimento comuns

Comunicação Síncrona: Mecanismos

- Vantagem da Chamada Remota de Procedimento
 - Cliente e servidor não precisam saber que as mensagens são utilizadas.
 - Eles as vêem como chamadas de procedimento locais.

Comunicação Síncrona: Mecanismos

- Problemas da Chamada Remota de Procedimento
 - Dificuldade da passagem de parâmetros por referência:
 - Se servidor e cliente possuem diferentes representações de informação (necessidade de conversão).
 - Diferenças de arquitetura:
 - As máquinas podem diferir no armazenamento de palavras.
 - Ex: long do C tem tamanhos diferentes, dependendo se o S.O. For 32 ou 64 bits

Comunicação Síncrona: Mecanismos

- Problemas da Chamada Remota de Procedimento
 - Falhas semânticas:
 - Ex: o servidor pára de funcionar quando executava uma RPC. O que dizer ao cliente?
 - Se disser que houve falha e o servidor terminou a chamada logo antes de falhar, o cliente pode pensar que falhou antes de executar a chamada.
 - Ele pode tentar novamente, o que pode não ser desejável (ex: atualização de BD).
 - Principais abordagens: “no mínimo uma vez”, “exatamente uma vez” e “no máximo uma vez”.

Comunicação Síncrona: Mecanismos

- Falhas Semânticas da Chamada RPC:
 - No mínimo uma vez
 - O cliente fica retransmitindo o pedido até que tenha a resposta desejada
 - Exatamente uma vez
 - Toda chamada é executada exatamente uma vez
 - Se o cliente transmite e o servidor cai, ele não sabe se o servidor processou o pedido antes de cair
 - No máximo uma vez
 - Se o servidor cair, o cliente saberá do erro, mas não saberá se a operação foi executada

Comunicação – Outros Mecanismos

- RPC – Remote Procedure Call
 - Rotinas que permitem comunicação de processos em diferentes máquinas;
 - Chamadas remotas;
- MPI – Message-passing Interface;
 - Sistemas paralelos;
- RMI Java – Remote Method Invocation
 - Permite que um objeto ativo em uma máquina virtual Java possa interagir com objetos de outras máquinas virtuais Java, independentemente da localização dessas máquinas virtuais;

Mensagens

- Outros Mecanismos de Comunicação Baseados na Troca de mensagens
 - Existem diversos mecanismos atuais baseados na troca de mensagens, além dos discutidos. Os mais representativos são:
 - Caixas Postais (Mailboxes)
 - Portos (Ports)

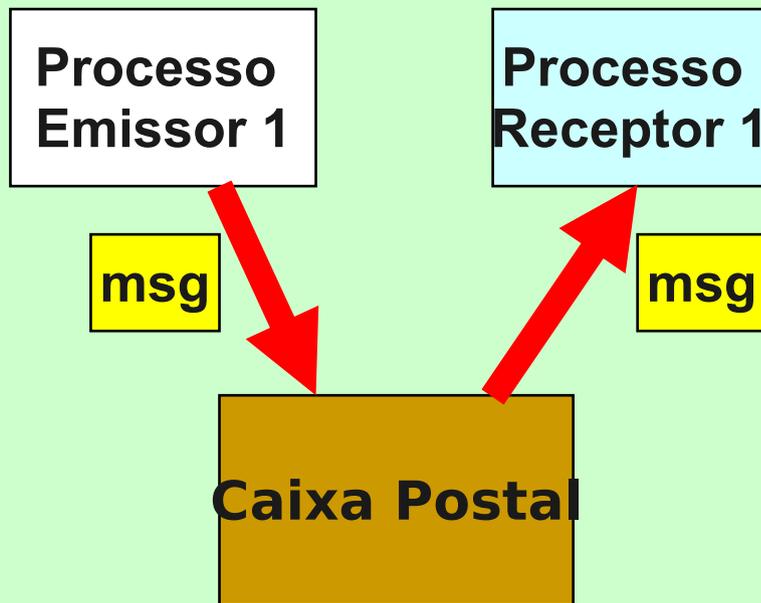
Caixas Postais

- Estrutura de dados:
 - São filas de mensagens não associadas, a princípio, com nenhum processo.
- Lugar para se colocar um certo número de mensagens
 - Mensagens são enviadas ou lidas da caixa postal, e não diretamente dos processos
 - Quando um processo tenta enviar para uma caixa cheia, ele é suspenso até que uma mensagem seja removida da caixa

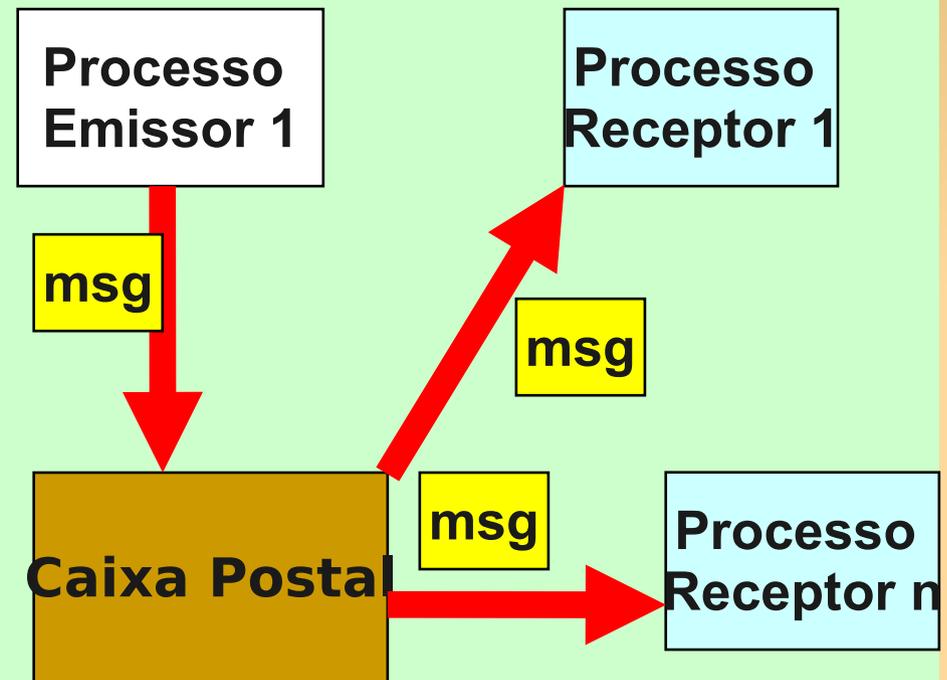
Caixas Postais

- Os processos podem enviar e receber mensagens das caixas postais
 - Mecanismo adequado para comunicar diversos remetentes a diversos receptores.
 - Cada processo pode ter uma caixa exclusiva
- Restrição
 - Necessidade de envio de duas mensagens para comunicar o remetente com o receptor: uma do remetente à caixa postal, e outra da caixa postal para o receptor.

Caixas Postais



**Comunicação
um-para-um**



**Comunicação
um-para-vários**

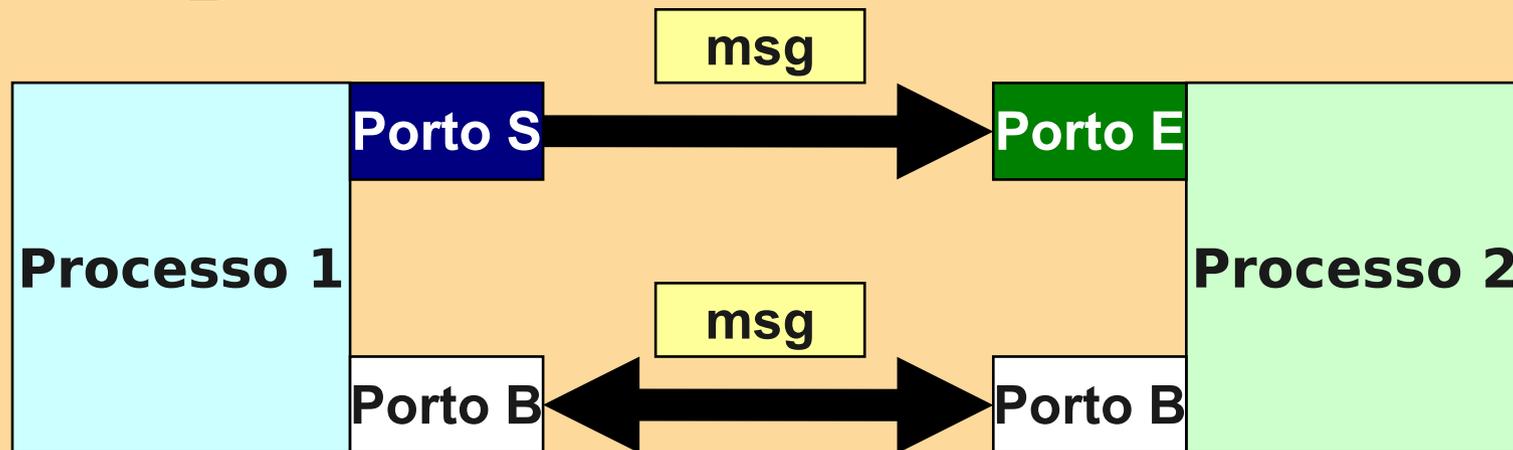
Portos

- Consistem em elementos do sistema que permitem a comunicação entre conjuntos de processos.
 - Conexão de dados virtual ou lógica, usada para que programas troquem informação diretamente
 - Ex: TCP – numerados de 1 a 65535
- Cada porto é como uma caixa postal, porém com um dono, que será o processo que o criar.

Portos

Cria_porto(S, saída, msg)
conecta_porto(S, E)
envia_porto(S, msg)
desconecta_porto(S, E)
destruir_porto(S)

Cria_porto(E, entrada, msg)
recebe_porto(E, msg)
destruir_porto(E)

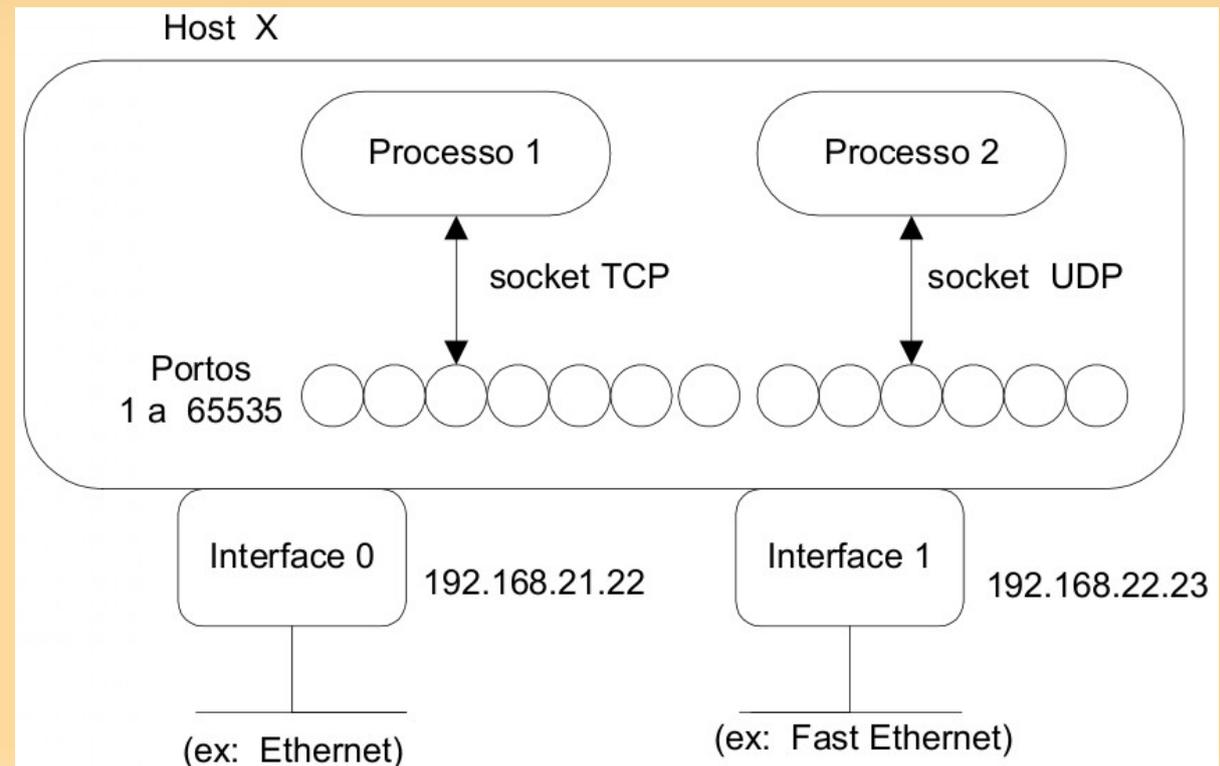


Portos

- Outras Características dos Portos
 - A criação e a interligação de portos e caixas postais pode ser feita de maneira dinâmica.
 - A necessidade de enfileiramento das mensagens enviadas, torna necessário o uso de “buffers”, para o armazenamento intermediário.
 - A comunicação entre processos locais ou remotos, em um sistema estruturado com portos, será feita pela execução de primitivas síncronas (ou assíncronas) do tipo envia e recebe.

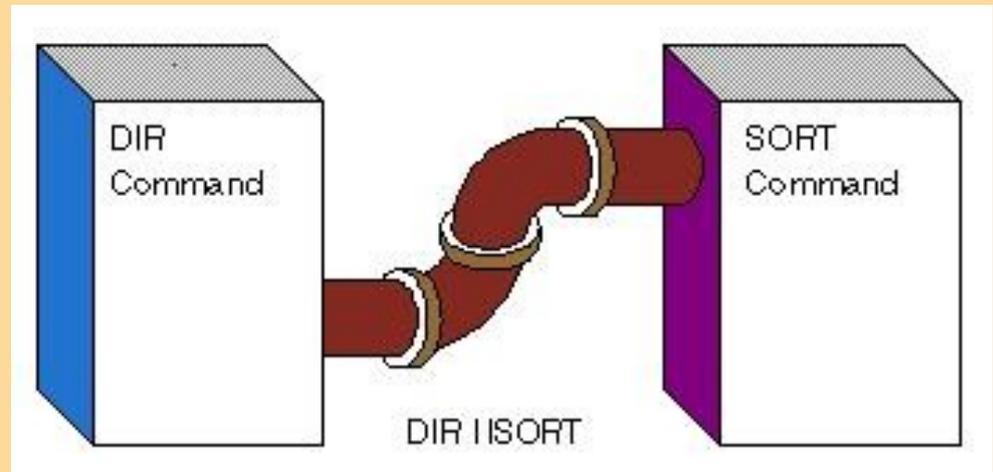
Comunicação – Outros Mecanismos

- Socket:
 - Par endereço IP e porta utilizado para comunicação entre processos em máquinas diferentes;
 - Host X (192.168.1.1:1065) Server Y (192.168.1.2:80);
 - Em unix, sockets são um tipo de arquivo que conectam processos a seus respectivos portos



Comunicação – Outros Mecanismos

- Pipe:
 - Fluxo de dados de mão única entre processos
 - Usados no sistema operacional UNIX para permitir a comunicação entre processos.
 - Um pipe é um modo de conectar a saída de um processo com a entrada de outro processo, sem o uso de arquivos temporários.
 - Todo dado escrito por um processo na pipe é dirigido, pelo kernel, a outro processo, que pode então lê-lo



Comunicação – Outros Mecanismos

- Pipe:
 - Um pipeline é uma conexão de dois ou mais programas ou processos através de pipes.
 - Permite a criação de filas de processos;
 - Ex: `ps -ef | grep alunos | sort`
 - Existe enquanto o processo existir;
 - Exemplos:
 - Sem uso de pipes (usando arquivos temporários)
 - `$ ls > temp`
 - `$ sort < temp`
 - Com uso de pipes
 - `$ ls | sort`

Comunicação – Outros Mecanismos

- Problemas com pipes:
 - Não há como abrir uma pipe já existente
 - Dois processos não compartilham a mesma pipe, a menos que sejam irmãos, e ela tenha sido criada pelo pai desses processos
- Named pipe (ou FIFO):
 - Extensão de pipe que soluciona esse problema
 - É um tipo de arquivo especial em Unix
 - Pode ser aberto por qualquer processo
 - Continua existindo mesmo depois que o processo terminar;
 - Criado com chamadas de sistemas;

Referências Adicionais

- Bovet, D.P., Cesati, M.: Understanding the Linux Kernel. O'Reilly. 1st Ed. 2000.