# COMUNICAÇÃO ENTRE PROCESSOS



## Comunicação entre processos

- Memória Compartilhada:
  - os processo compartilham variáveis e trocam informações através do uso dessas variáveis compartilhadas


- Sem Memória Compartilhada:
  - os processos compartilham informações através de troca de mensagens
  - o S.O é responsável pelo mecanismo de comunicação entre os processos




## Comunicação entre processos - troca de mensagens

- Aspectos importantes para um sistema de troca de mensagens (1)
  - Simplicidade: construção de novas aplicações para interoperar com já existentes deve ser facilitada
  - Semântica Uniforme: comunicação local (processos no mesmo nodo) e comunicação remota (processos em nodos diferentes) através de funções tão próximas quanto possível
  - Eficiência: reduzir número de mensagens trocadas tanto quanto possível
    - economizar fechamento e abertura de conexões;
  - Confiabilidade: garantir entrega da mensagem - confirmação, eliminação de duplicatas, ordenação
  - Corretude: relacionada principalmente a comunicação em grupo
    - garantia de aspectos como Atomicidade; Ordenação; “Survivability”



## Comunicação entre processos - troca de mensagens

- Aspectos importantes para um sistema de troca de mensagens (2)
  - Flexibilidade: possibilidade de utilizar somente funcionalidade requerida (em prol de desempenho)
    - necessidade ou não de entrega garantida, ordenada, atomica, etc
  - Segurança: suporte a autenticação, privacidade
  - Portabilidade: disponibilidade do mecanismo de IPC em plataformas heterogêneas



## Comunicação entre processos - troca de mensagens

- Operações: send/receive - sincronização
  - Send bloqueante:
    - processo enviador fica bloqueado até recepção de confirmação do receptor
    - problema: ficar bloqueado para sempre - mecanismo de *time-out*
  - Send não-bloqueante:
    - processo enviador pode proceder assim que conteúdo (dados a enviar) for copiado para buffer de envio

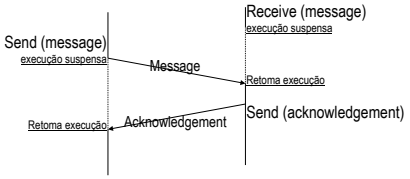


## Comunicação entre processos - troca de mensagens

- Operações: send/receive - sincronização
  - Receive bloqueante:
    - processo receptor fica bloqueando na operação de receive até chegada da mensagem
    - problema: ficar bloqueado para sempre - mecanismo de *time-out*
  - Receive não-bloqueante:
    - processo receptor informa ao núcleo onde armazenar mensagem e procede
    - processo receptor fica sabendo da chegada da mensagem por:
      - polling - teste periódico
      - interrupção

## Comunicação entre processos - troca de mensagens

- Operações: send/receive - sincronização
  - quando send e receive são bloqueantes a comunicação é dita síncrona
  - senão é dita assíncrona






## Comunicação entre processos - troca de mensagens

- Operações: send/receive – tipos de endereçamento
  - explícito: processo com o qual se quer comunicar é explicitamente identificado através de um parâmetro
  - implícito: processo com o qual se quer comunicar não é identificado
    - ex.: sender mandando para qualquer receiver que desempenhe uma função/serviço - sender nomeia serviço ao invés de processo - qualquer processo servidor (de um grupo) que desempenhe esta função pode receber a mensagem
    - receiver quer receber independente do sender: modelo cliente/servidor - servidor quer poder servir qualquer cliente



## Comunicação entre processos - troca de mensagens

- Endereçamento explícito:
  - o remetente identifica o recebedor
  - o recebedor identifica o remetente

```
/ * task 0 * /
main ( ) {
    -
    -
    -
  send ( t1, m );
}
```

```
/ * task 1 * /
main ( ) {
    -
    -
    -
  receive ( t0, m );
}
```



## Comunicação entre processos - troca de mensagens

- Endereçamento implícito do lado recebedor:
  - remetente identifica o recebedor
  - recebedor recebe de qualquer remetente

```
/ * task 0 * /
main ( ) {
    -
    -
    -
  send ( t2, m );
}
```

```
/ * task 1 * /
main ( ) {
    -
    -
    -
  send ( t2, m );
}
```

```
/ * task 2 * /
main ( ) {
    -
    -
    -
  receive ( -1, m );
}
```



## Comunicação entre processos - troca de mensagens

- Operações: send/receive - identificação de processos
  - machine_id@local_id
    segunda parte usada localmente na máquina para achar processo
  - e se processo move ?
  - machine_id , local_id, machine_id
    nome de criação , máquina onde está
  - a máquina onde o processo foi criado deve manter tabela dizendo onde processo se encontra
  - máquinas que um processo visita têm que manter entrada dizendo a próxima máquina para onde o processo migrou;
  - overhead; mensagem alcançar destino pode depender de vários nodos (falhas?)



## Comunicação entre processos - troca de mensagens

- Operações: send/receive - identificação de processos
  - métodos não transparentes: especifica identificador de máquina
  - identificador único do processo não deve ter embutida informação de localização do processo
  - nomeação em dois níveis:
    - nome de alto nível (string) independente de localização
    - nome de baixo nível: como machine_id@local_id
    - *servidor de nomes* traduz de um para outro
    - processos usam nomes de alto nível para endereçar outros processos
    - durante o send o servidor de nomes é consultado para achar nome de baixo nível (localização)

## Comunicação entre processos - troca de mensagens

- Operações: send/receive - bufferização
  - transmissão da mensagem: copiar corpo da mensagem do espaço de endereçamento do processo enviador para espaço de endereçamento do receptor
  - processo recebedor pode não estar pronto para receber
    - SO salva mensagem (**SO do lado receptor implementa bufferização**)
  - relação com o sincronismo da comunicação
    - síncrona - *null buffer* (um extremo)
    - assíncrona - *buffer* de capacidade ilimitada (outro extremo)
    - tipos intermediários de *buffers*
      - *single-message*
      - *finite-bound or multiple-message*



## Comunicação entre processos - troca de mensagens

- Operações: send/receive - característica: bufferização
  - ***null buffer*** - não há espaço temporário para armazenar mensagem
  - para comunicação síncrona
  - estratégia 1:
    - mensagem permanece no espaço de endereçamento do processo enviador, que fica bloqueado no send
    - quando receptor faz receive, uma mensagem de confirmação para o enviador é mandada, e o send pode enviar os dados
  - estratégia 2:
    - enviador manda a mensagem e espera confirmação
    - se processo receptor não está em receive, kernel descarta mensagem
    - se processo receptor está em receive, manda confirmação
    - se não há confirmação dentro de um tempo (*time-out*) é sinal de que receptor não estava em receive, e a mensagem é repetida



## Comunicação entre processos - troca de mensagens

- Operações: send/receive - bufferização
  - ***single message buffer***
  - para comunicação síncrona
  - idéia: manter a mensagem pronta para uso no local de destino
  - *request message* é bufferizada no nodo receptor, caso o processo receptor não esteja pronto para recepção



## Comunicação entre processos - troca de mensagens

- Operações: send/receive - bufferização
  - ***unbounded-capacity buffer***
  - para comunicação assíncrona
  - enviador não espera receptor - podem existir várias mensagens pendentes ainda não aceitas pelo receptor
  - para garantir entrega de todas mensagens, um buffer ilimitado é necessário



## Comunicação entre processos - troca de mensagens

- Operações: send/receive - bufferização
  - ***finite-bound buffer***
  - *unbounded capacity buffer*: impossível na prática
  - estratégia necessária para buffer *overflow*
    - comunicação sem sucesso: send retorna código de erro
    - comunicação com controle de fluxo: o enviador fica bloqueado no send até que o receptor aceite alguma(s) mensagem(s);
      - introduz sincronização entre originador e receptor
      - send assíncrono não é assíncrono para todas mensagens



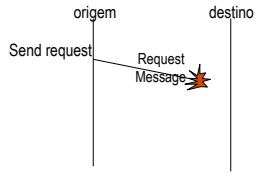
## Comunicação entre processos - troca de mensagens

- Operações: send/receive - codificação e decodificação
  - mensagens entre processos rodando em diferentes arquiteturas
  - transferência de valores de ponteiros para memória perdem significado
  - identificação necessária para dizer tipo de dado sendo transmitido

  - uso de formato comum de transferência (sintaxe de transferência)
    - representação com tags (tagged): mensagem carrega tipo dos dados transferidos.
      Ex.: ASN.1 (abstract syntax notation - CCITT); sistema operacional Mach
    - representação sem tags: receptor tem que saber decodificar mensagem.
      Ex.: XDR (eXternal Data Representation - Sun); Courier (Xerox)

## Comunicação entre processos - troca de mensagens

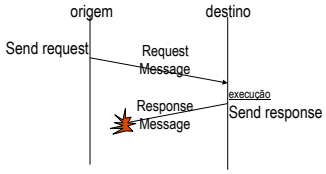
- Operações: send/receive - tratamento de falhas
  - perda de mensagem de pedido






## Comunicação entre processos - troca de mensagens

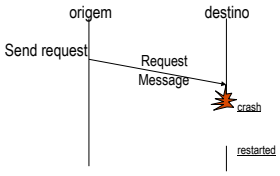
- Operações: send/receive - tratamento de falhas
  - perda de mensagem de resposta






## Comunicação entre processos - troca de mensagens

- Operações: send/receive - tratamento de falhas
  - execução do pedido no destinatário não tem sucesso





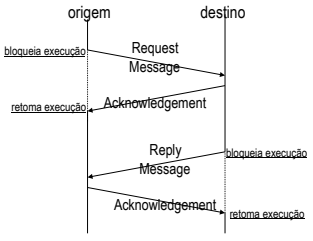

## Comunicação entre processos - troca de mensagens

- Operações: send/receive - tratamento de falhas
  - estratégias
    - four message: confirmação das mensagens de pedido e resposta
    - three message: confirmação da resposta
    - two message: resposta é confirmação



## Comunicação entre processos - troca de mensagens

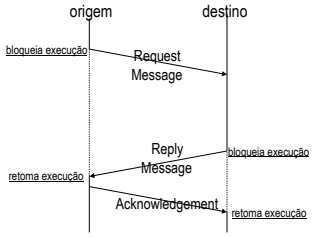
- Operações: send/receive - tratamento de falhas
  - estratégia
    - four message: confirmação das mensagens de pedido e resposta





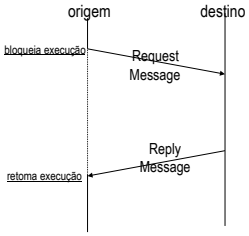

## Comunicação entre processos - troca de mensagens

- Operações: send/receive - tratamento de falhas
  - estratégia
    - three message: confirmação da resposta



## Comunicação entre processos - troca de mensagens

- Operações: send/receive - característica: tratamento de falhas
  - estratégia
    - two message: resposta é confirmação






## Comunicação entre processos - troca de mensagens

- Operações: send/receive - tratamento de falhas
  - idempotência e tratamento de requests duplicados
    - idempotência: repetibilidade
      - operação idempotente pode ser repetida inúmeras vezes sem causar efeitos colaterais ao servidor
        - ex.: get-time(), sqrt(x)
    - operações não idempotentes
      - necessitam "*exactely-once*" *semantics*
      - garante que somente uma execução no servidor é realizada
      - ex.: conta.deposita(valor); conta.retira(valor)
      - usar identificador único para cada request;
      - lado servidor guarda reply cache para responder mesma resposta a pedido repetido



## Comunicação entre processos - troca de mensagens

- Troca de mensagens: Mecanismos
  - Arquivos (Unix *pipes*)
  - *Sockets*
  - RPC (*Remote Procedure* Call)
    - RMI (Remote Method Invocation) - Java, DCOM, CORBA, …