Laboratório de Pesquisa em Redes e Multimídia

Sincronização de Processos (5)

Troca de Mensagens

Universidade Federal do Espírito Santo
Departamento de Informática

Passagem (Troca) de Mensagens

- Motivação:
  - Semáforos e algoritmos de exclusão mútua são baseados no compartilhamento de variáveis. Isso implica no compartilhamento de memória física. Entretanto, em sistemas distribuídos, as máquinas formam unidades independentes (nós de uma rede), não existindo memória compartilhada entre os vários processos.
- Solução:
  - Novo paradigma de programação ("*message passage*"), onde a sincronização e comunicação entre processos é baseada em (duas) novas primitivas: *send* e *receive*.

Primitivas

- *send (destination, message_buffer)*
  - Envia a mensagem armazenada em "*message_buffer*" para um processo destino ou para uma caixa postal.
- *receive (source, message_buffer)*
  - Recebe uma mensagem de uma fonte específica, ou de qualquer fonte, e armazena-a em "*message_buffer*".

Questões de Projeto

- Uma série de novas questões surgem no cenário de troca de mensagens:
  - Sincronização entre os processos;
  - Endereçamento;
  - Formato das mensagens;
  - Disciplinas de filas;
  - Etc.

Sincronização (1)

- "*Blocking send, blocking receive*":
  - Emissor e receptor são bloqueados até que a mensagem seja entregue.
  - Mecanismo conhecido como "*rendezvous*" (encontro).
- *"Nonblocking send, blocking receive"*:
  - Emissor continua processando normalmente (p.ex., enviando novas mensagens).
  - Receptor é bloqueado até a recepção da mensagem (ex: servidor).

Sincronização (2)

- *"Nonblocking send, blocking receive"* (cont.)
  - Esquema mais usado.
  - Erros podem levar a uma situação em que um processo gera mensagens repetidamente, consumindo os recursos do sistema, incluindo tempo do processador e memória.
  - "*Nonblocking send*" coloca no programador a responsabilidade de determinar se a mensagem foi ou não recebida (uso de mensagens de reconhecimento – "*acknowledgment*").
  - Com "*blocking receive*" o processo receptor pode ficar bloqueado eternamente se a mensagem enviada se perder (soluções: uso de *timeout*, uso de mais de uma fonte, etc.)

Sincronização (3)

- *“ Nonblocking send, nonblocking receive”*
  - Nenhuma parte envolvida na comunicação precisa esperar.
  - Com *“nonblocking receive”* o receptor nunca fica bloqueado eternamente, entretanto, existe o perigo da mensagem ser enviada após o processo receptor ter executado o *receive* correspondente, ou seja, a mensagem será perdida.

Endereçamento (1)

- Endereçamento Direto:
  - A primitiva *send* inclui um identificador específico do processo destino.
  - A primitiva *receive* pode operar de duas formas:

    *(i) r*equerendo, a priori, a identificação do processo transmissor; ou

    (ii) usando um parâmetro (“*source*”) para retornar a identificação do processo transmissor após o término da operação (ex: servidor de impressão, que aceita pedidos de impressão de quaisquer outros processos).

Endereçamento (2)

- Endereçamento Indireto:
  - As mensagens não são enviadas diretamente do transmissor para o receptor mas sim um local intermediário (filas) que temporariamente armazena as mensagens enviadas.
  - Tal local é conhecido como *mailbox*. Assim, processos enviam mensagens para *mailboxes* e outros retiram mensagens dos *mailboxes*.
  - Uma vantagem do endereçamento indireto é que, ao desacoplar o transmissor e receptor, uma grande flexibilidade no uso das mensagens é conseguida.

Endereçamento (3)

- Endereçamento Indireto: relacionamentos entre transmissor e receptor
  - *Um-para-um*: permite comunicação privativa.
  - *Muitos-para-um*: útil para interação cliente-servidor. Neste caso, o *mailbox* é geralmente referenciado como "*port*".
  - *Um-para-muitos*: útil para aplicações de grupo (*broadcast* e *multicast*).
- Outras questões:
  - *Mailbox ownership*: no caso de um *port*, ele é tipicamente criado e tem como dono o processo receptor. Para um *mailbox* genérico, o S.O. pode oferecer um serviço do tipo *create_mailbox*. Nesse caso, o criador é o dono.

Endereçamento Indireto

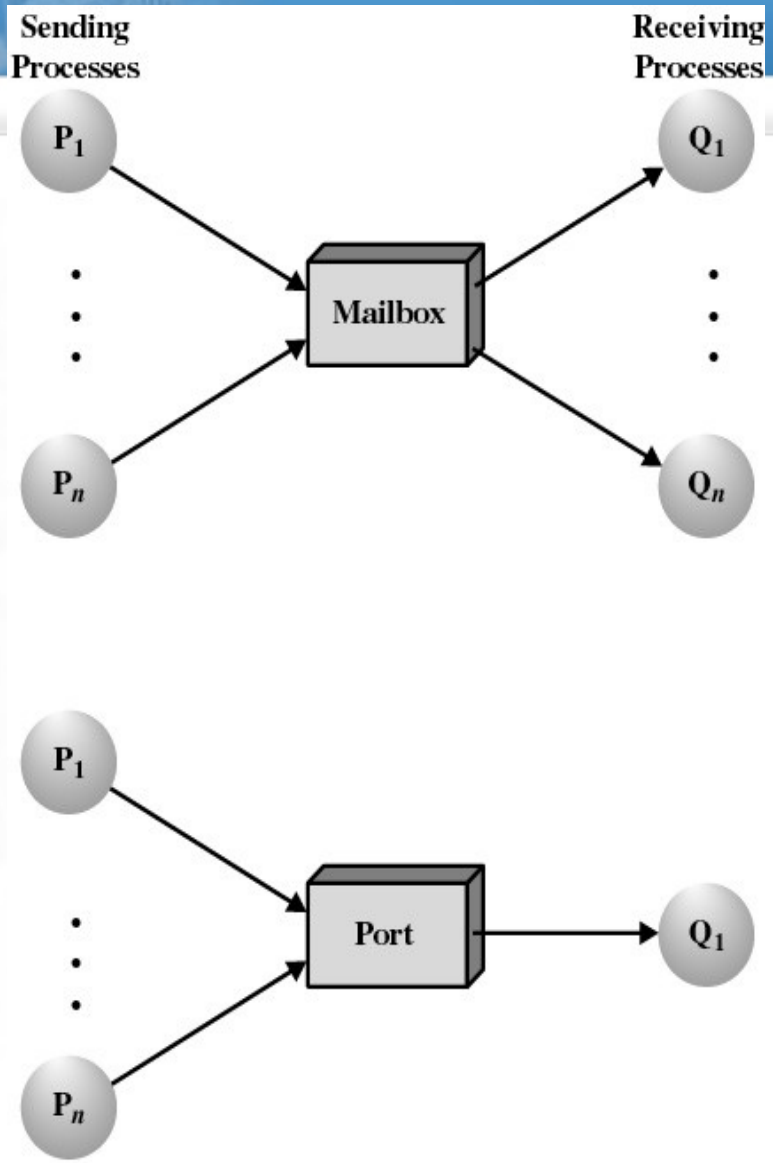

Figure 5.24 Indirect Process Communication

Formato da Mensagem

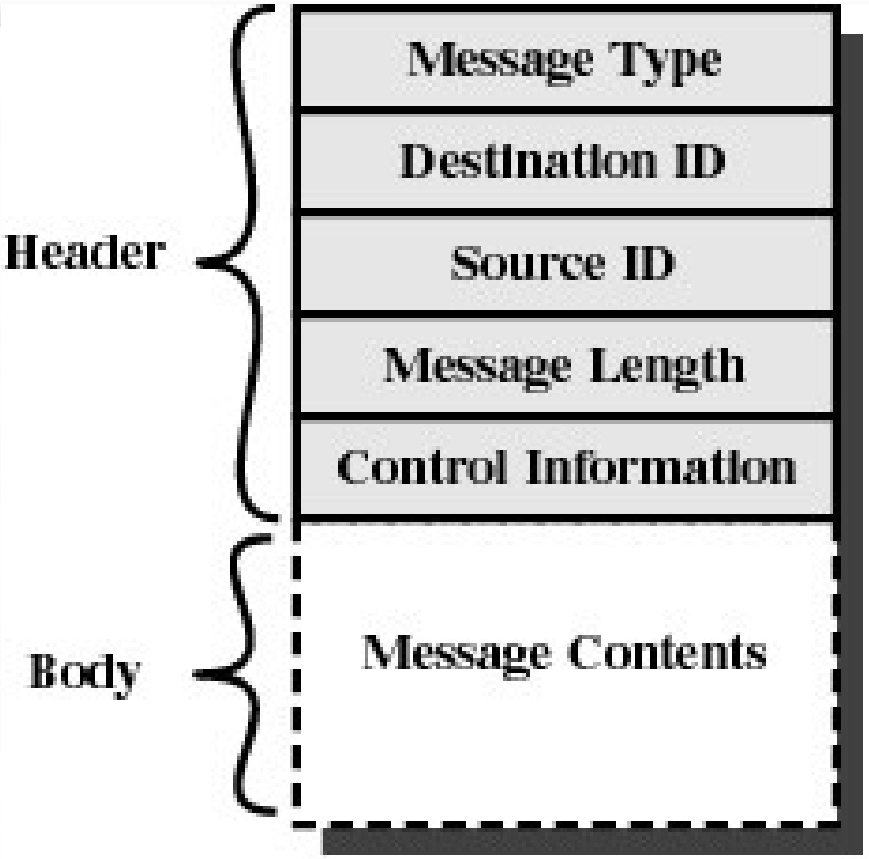

Figure 5.25 General Message Format

```
#define N 100                           /* number of slots in the buffer */
void producer(void) {
    int item;
    message m;                          /* message buffer */
    while (TRUE) {
        produce_item(&item);            * generate someth. 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 */
        extract_item(&m, &item);        /* extract item from message */
        send(producer, &m);             /* send back empty reply */
        consume_item(item);  }}         /* do something with the item */
```

Problema do Produtor-Consumidor com Troca de Mensagens

Referências

- Silberschatz A. G.; Galvin P. B.; Gagne G.; ''Fundamentos de Sistemas Operacionais'', 6a. Edição, Editora LTC, 2004.
  - Seção 4.5
- A. S. Tanenbaum, ''Sistemas Operacionais Modernos'', 2a. Edição, Editora Prentice-Hall, 2003.
  - Seção 2.3.8
- Deitel H. M.; Deitel P. J.; Choffnes D. R.; “Sistemas Operacionais”, 3ª. Edição, Editora Prentice-Hall, 2005
  - Seção 3.5.2