Programação Concorrente

Trocas de Mensagens

Prof. Eduardo Alchieri

Trocas de Mensagens

- **Trocas de Mensagens**
  - Para a comunicação/sincronização em processos que estão em máquinas diferentes (ambientes em rede), os conceitos de semáforos e monitores perdem suas funcionalidades, pois as variáveis de controle estariam em algumas das máquinas e seria impossível realizar bloqueios atômicos
  - Assim, para a comunicação entre processos que estão em máquinas diferentes, faz-se necessário a Troca de Mensagens
  - Esse método de comunicação interprocessos usa duas primitivas:

    send(destination, &message); e

    receive(source, &message);

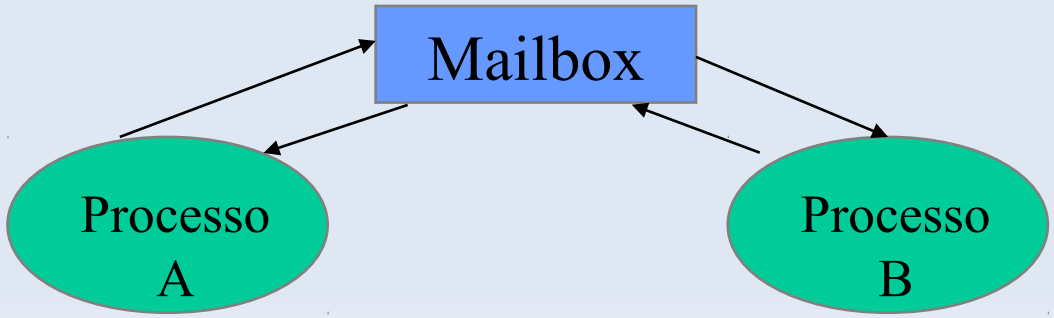
Trocas de Mensagens

- **Trocas de Mensagens**
  - O processo que desejar enviar (ou escrever) uma mensagem, chama *send,* passando o endereço de onde a mensagem está
  - O processo que desejar receber (ou ler) uma mensagem executa o *receive*, indicando uma origem (ou colocando qualquer origem) de onde receberá a mensagem
  - Se não houver mensagem disponível, o processo que executou o *receive* poderá ficar bloqueado até que alguma mensagem chegue. Como alternativa, ele pode retornar imediatamente acompanhado de um código de erro
  - Durante a comunicação por troca de mensagens, pode haver perdas das mensagens.
  - Para resolver esse problema, o receptor pode combinar de enviar uma mensagem especial de confirmação de recebimento (*acknowledgement* - **ACK**)

Trocas de Mensagens

- **Trocas de Mensagens**
  - A comunicação entre processos pode ser feita através de endereçamento direto ou indireto
  - No **endereçamento direto** o processo que deseja enviar ou receber uma mensagem deve endereçar explicitamente o nome do processo receptor ou transmissor

  - O **endereçamento indireto** utiliza uma área compartilhada, na qual as mensagens podem ser colocadas pelo processo transmissor e retiradas pelo receptor



Trocas de Mensagens

- **Trocas de Mensagens**
  - Independente da forma de endereçamento entre os processos, a comunicação entre eles pode bloquear ou não os processos envolvidos
  - Basicamente, há dois tipos de comunicação:
    - Comunicação Síncrona
    - Comunicação Assíncrona
  - Na Comunicação Síncrona, quando um processo envia uma mensagem (SEND) ele fica esperando até que o processo receptor leia a mensagem e vice-versa
  - Na Comunicação Assíncrona, nem o receptor permanece aguardando o envio de uma mensagem, nem o transmissor o seu recebimento

Trocas de Mensagens

- Problema do produtor e consumidor usando trocas de mensagens

```
#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 */
     }
}
```

Trocas de Mensagens

- **MPI (Message Passing Interface)**
  - MPI é uma biblioteca de comunicação que permite a programação baseada em troca de mensagens

MPI_Init(&argc , &argv)

Inicializa uma execução em MPI, é responsável por copiar o código do programa em todos os processadores que participam da execução. Nenhuma outra função MPI pode aparecer antes de MPI_INIT. argc, argv são variáveis utilizadas em C para recebimento de parâmetros.

MPI_Finalize()

Termina uma execução MPI. Deve ser a última função em um programa MPI.

MPI_Comm_Size(*communicator* , *&size*)

Determina o número de processos em uma execução. *communicator* indica o grupo de comunicação e &size contém, ao término da execução da primitiva, o número de processos no grupo.

Trocas de Mensagens

- **MPI (Message Passing Interface)**

MPI_Comm_Rank(*communicator , &pid*)

Determina o identificador do processo corrente. *communicator* indica o grupo de comunicação e *&pid* identifica o processo no grupo.

MPI_Send (*&buf, count, datatype, dest, tag, comm*)

Permite a um processo enviar uma mensagem para um outro. É uma operação não bloqueante. O processo que a realiza continua sua execução. Os parâmetros são:

&buf: endereço do buffer de envio
count: número de elementos a enviar
datatype: tipo dos elementos a serem enviados
dest: identificador do processo destino da mensagem
tag: tipo da mensagem
comm: grupo de comunicação

Trocas de Mensagens

- **MPI (Message Passing Interface)**

MPI_Recv (*&buf, count, datatype, dest, tag, comm*)

Função responsável pelo recebimento de mensagens. É uma operação bloqueante. O processo que a executa fica bloqueado até o recebimento da mensagem. Os parâmetros são:

&buf: endereço do buffer de recebimento
count: número de elementos a enviar
datatype: tipo dos elementos a serem enviados
dest: identificador do processo remetente da mensagem
tag: tipo da mensagem
comm: grupo de comunicação
status: status de operação

Trocas de Mensagens

- Para compilar
  - mpicc (sudo apt-get install libopenmpi-dev)
    - mpicc arquivo.c -o executável
- Para executar
  - mpirun (sudo apt-get install openmpi-bin)
    - mpirun -[argumentos] [executável]
    - O argumento np especifica o número de processos: mpirun -np 2 executável
    -
- Mostrar exemplo de comunicação usando MPI