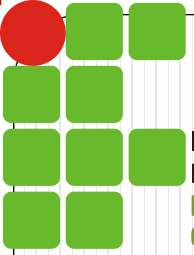


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
BAHIA
Campus Salvador

Aula 6: Comunicação entre processos

Instituto Federal da Bahia

INF009 - Sistemas Operacionais

Profª Flávia Maristela

Comunicação entre processos (-- motivação --)

- Processos em execução no sistema operacional podem ser:
  - Independentes:
    - Quando não podem ser afetados pela execução de outro processo
  - Cooperantes
    - Quando podem ser afetados pela execução de outro processo
- Já sabendo que compartilhamento causa problemas, é mais fácil criar processos independentes!!

Comunicação entre processos (-- motivação --)

- Entretanto, é extremamente desejável criar um ambiente com processos cooperantes!

- Porque?
  - Compartilhamento de informações
  - Aumento da velocidade de computação
  - Modularidade
  - Dar suporte a execução de várias tarefas

- Processos cooperantes requerem comunicação entre processos (*Interprocess communication – IPC*)

Comunicação entre processos (-- definição --)

- Mecanismo que permite aos processos trocarem dados ou informações.

- Comunicação entre processos não usa interrupção!

- Frequentemente é feita de duas formas:
  - Troca de mensagens
  - Compartilhamento de memória

Comunicação entre processos (-- troca de mensagens --)

- Se pensarmos numa arquitetura centralizada, os processos estão na mesma máquina.
  - Diferentes processos têm acesso aos mesmos recursos.

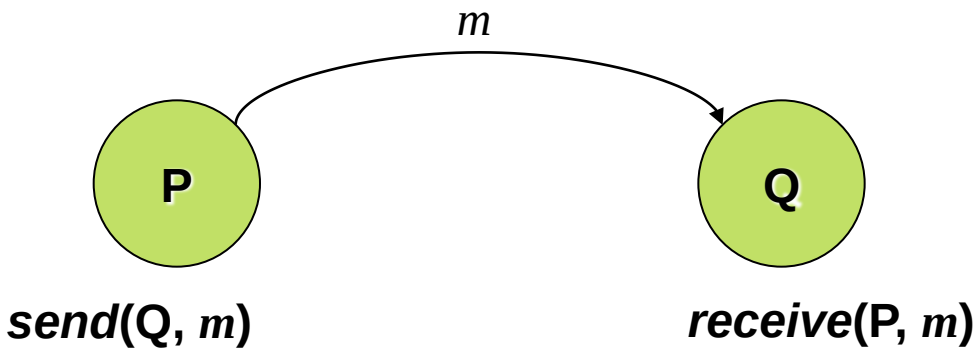
- O que acontece se a arquitetura do sistema for distribuída? (um *chat*, por exemplo)

- Como os processos podem se comunicar?

Comunicação entre processos (-- troca de mensagens --)

- Processos podem se comunicar por troca de mensagens.
  - Frequentemente quando estão em diferentes máquinas e precisam compartilhar dados
- A troca de mensagens é feita baseada em duas ***primitivas***:
  - *send()*
  - *receive()*
- Mensagens podem ter tamanho fixo ou variável
- Se dois processos precisam se comunicar, deve haver um *link* entre eles.

Comunicação entre processos (-- troca de mensagens --)

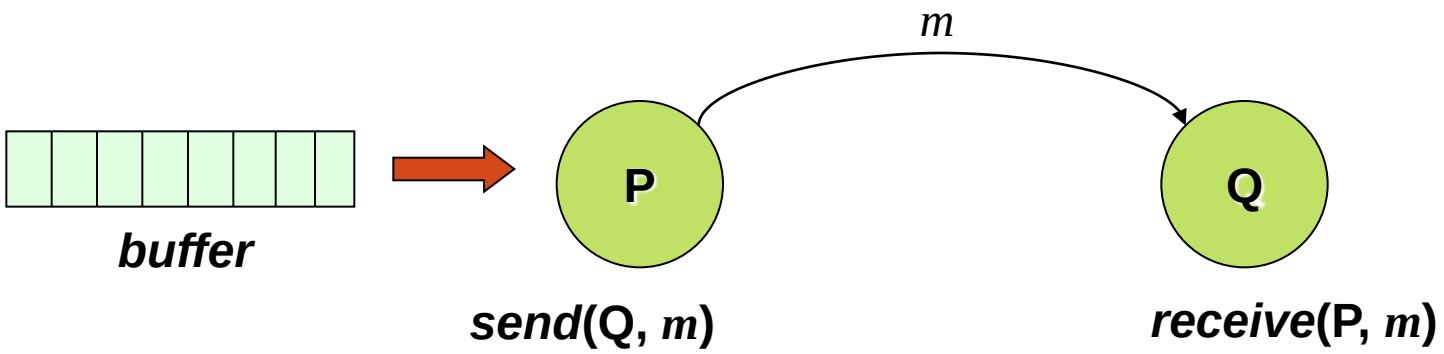


Comunicação entre processos (-- troca de mensagens --)

- Troca de mensagens por sincronização:
  - Blocking send: processo que envia a mensagem fica bloqueado até a confirmação do recebimento
  - Nonblocking send: processo envia a mensagem e vai executar a próxima instrução
  - Blocking receive: receptor fica bloqueado até que a mensagem esteja disponível
  - Nonbloking receive: o receptor devolve uma mensagem válida ou nula.

Comunicação entre processos (-- troca de mensagens --)

- Troca de mensagens por bufferização:
  - *Zero capacity*
  - *Bouded-capacity*
  - *Unbouded-capacity*



Comunicação entre processos (-- compartilhamento de memória --)

- Processos devem definir uma área de memória que será compartilhada;
- Por padrão o sistema operacional não permite que um processo acesse outro processo!
- Como resolver???

Comunicação entre processos (-- compartilhamento de memória --)

- Caso dois processos desejem compartilhar memória, ambos precisam assumir as consequências de não considerar as restrições do sistema operacional

Comunicação entre processos (-- compartilhamento de memória --)

- Processos trocam informações através de leituras e escritas numa área compartilhada;
- O sistema operacional não controla esta operação!
- O que os processos precisam garantir??

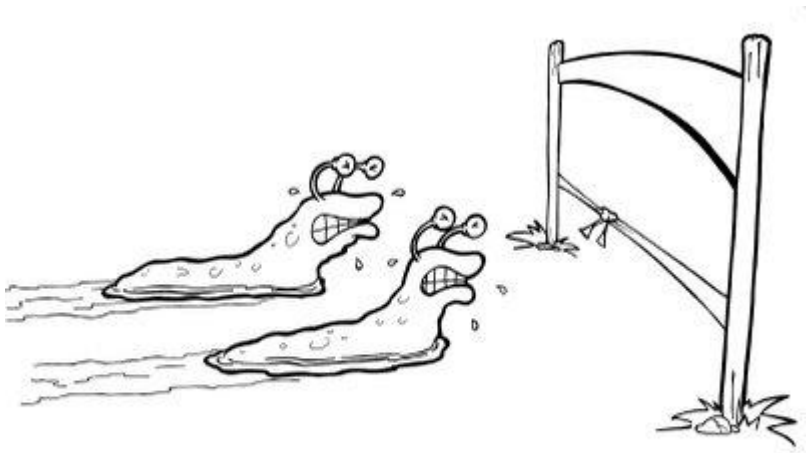
Para pensar um pouco...

O que acontece quando dois processos querem escrever na mesma área de memória no mesmo instante?

Comunicação entre processos (-- *Race condition* --)

- Em alguns sistemas operacionais, processos cooperantes frequentemente compartilham algum dispositivo de armazenamento.
  - Arquivos
  - Memória
  - Disco

Comunicação entre processos (-- *Race condition* --)



- Dois processos podem tentar ler ou escrever dados num espaço compartilhado, e o resultado final depende de quem está executando naquele momento.

Comunicação entre processos (-- *Race condition:* exemplo ilustrativo --)

- Um exemplo ilustrativo:
  - Suponha duas *threads*, que alteram o valor da variável *x*

T_1: x := x + 1
T_2: x := x + 2

Considere x = 2

T_1 ➔ T_2 : x = 5
T_2 ➔ T_1 : x = 5

T_1: x := x + 1
T_2: x := x * 2

Considere x = 2

T_1 ➔ T_2 : x = 6
T_2 ➔ T_1 : x = 5

Comunicação entre processos (-- *Race condition:* exemplo clássico --)

- PRODUTOR *vs*. CONSUMIDOR
  - Dois processos compartilham um buffer de tamanho fixo. Um deles (processo produtor) coloca dados no buffer. O outro (processo consumidor), remove estes dados.
  - Você consegue ver o problema?

Comunicação entre processos (-- *Race condition* e região crítica --)

- Como evitar condições de corrida?
  - Sincronizando os processos

ou seja

  - Proibindo que mais de um processo possa ler ou escrever numa área compartilhada ao mesmo tempo.

Comunicação entre processos (-- Exclusão Mútua --)

- Definição:
  - Mecanismo que garante que cada processo que usa uma área compartilhada terá acesso exclusivo a mesma.

Qual é o problema da exclusão mútua??

Para pensar...

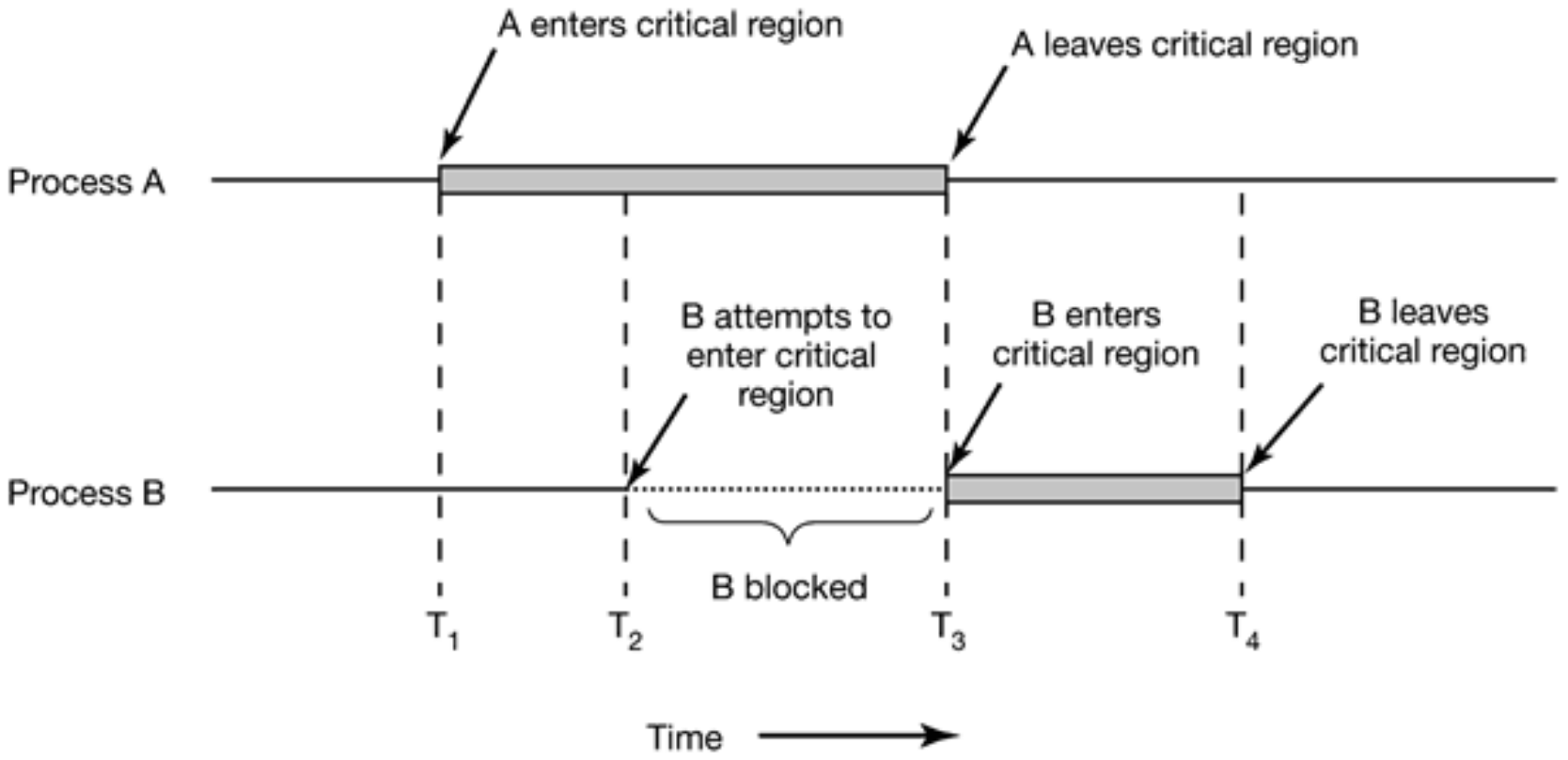
- Pense no problema do PRODUTOR *vs*. CONSUMIDOR.

- O que acontece se quando o produtor estiver armazenando um item, o consumidor não puder consumir nada?

Comunicação de Processos
(-- Exclusão mútua e região crítica --)

- Dois processos não podem estar simultaneamente em suas regiões críticas
- Nada pode ser assumido com relação a velocidade dos processos ou quantidade de processadores disponível
- Nenhum processo fora de sua região crítica pode bloquear um processo que esteja na região crítica
- Nenhum processo deve esperar indefinidamente para entrar na região crítica.

Comunicação de Processos
(-- Exclusão mútua e região crítica --)



Tanenbaum, Capítulo 2

Comunicação de Processos
(-- Como implementar exclusão mútua --)

- Espera ocupada
- *Sleep* and *wakeup*
- Semáforos
- Mutex
- Monitores

Comunicação de Processos
(-- Exclusão mútua + espera ocupada --)

- Premissa da espera ocupada:
  - Enquanto um processo executa na região crítica, o outro apenas espera.

- Formas de implementar:
  - Interrupção:
    - Problema: não é ideal que processos tenham controle sobre as interrupções

Comunicação de Processos
(-- Exclusão mútua + espera ocupada --)

- Formas de implementar:
  - Alternância Obrigatória

```
while (TRUE) {
    while (turn != 0)    /* loop */ ;
    critical_region();
    turn = 1;
    noncritical_region();
}
```

(a)

```
while (TRUE) {
    while (turn != 1);   /* loop */ ;
    critical_region();
    turn = 0;
    noncritical_region();
}
```

(b)

Comunicação de Processos
(-- Sleep e Wakeup --)

- Primitivas (chamadas de sistemas)
- *sleep()*
  - Bloqueia um processo enquanto aguarda um recurso
- *wakeup()*
  - Ativa o processo quando o recurso foi liberado

Comunicação de Processos
(-- *Sleep* e *Wakeup* --)

```
#define N 100        /* number of slots in the buffer */
int count = 0;       /* number of items in the buffer */

void producer (void)
{
    int item;

    while (TRUE) {
        item = produce_item();
        if (count == N) sleep();
        insert_item(item);
        count = count + 1;
        if (count == 1) wakeup(consumer);
    }
}

void consumer(void)
{
    int item;

    while (TRUE) {
        if (count == 0) sleep();
        item = remove_item();
        count = count - 1;
        if (count == N - 1) wakeup(producer);
        consume_item(item);
    }
}
```

Exercícios:

- Defina:
  - Programação concorrente
  - Programação paralela
  - Condição de corrida
  - Exclusão Mútua
  - Região Crítica
- O que é necessário para garantir para que seja possível implementar uma boa solução de exclusão mútua?
- O que são primitivas?
- Como pode ocorrer a comunicação entre processos?
- Em qual situação devemos usar cada um dos mecanismos de IPC? Justifique.