PROGRAMAÇÃO CONCORRENTE

INF009 – SISTEMAS OPERACIONAIS

INTRODUÇÃO – PROGRAMAÇÃO SEQUENCIAL

Um programa que é executado por apenas um processo, é chamado de programa sequencial.

A grande maioria dos programas é sequencial.

Programas sequenciais possuem apenas um fluxo de controle.

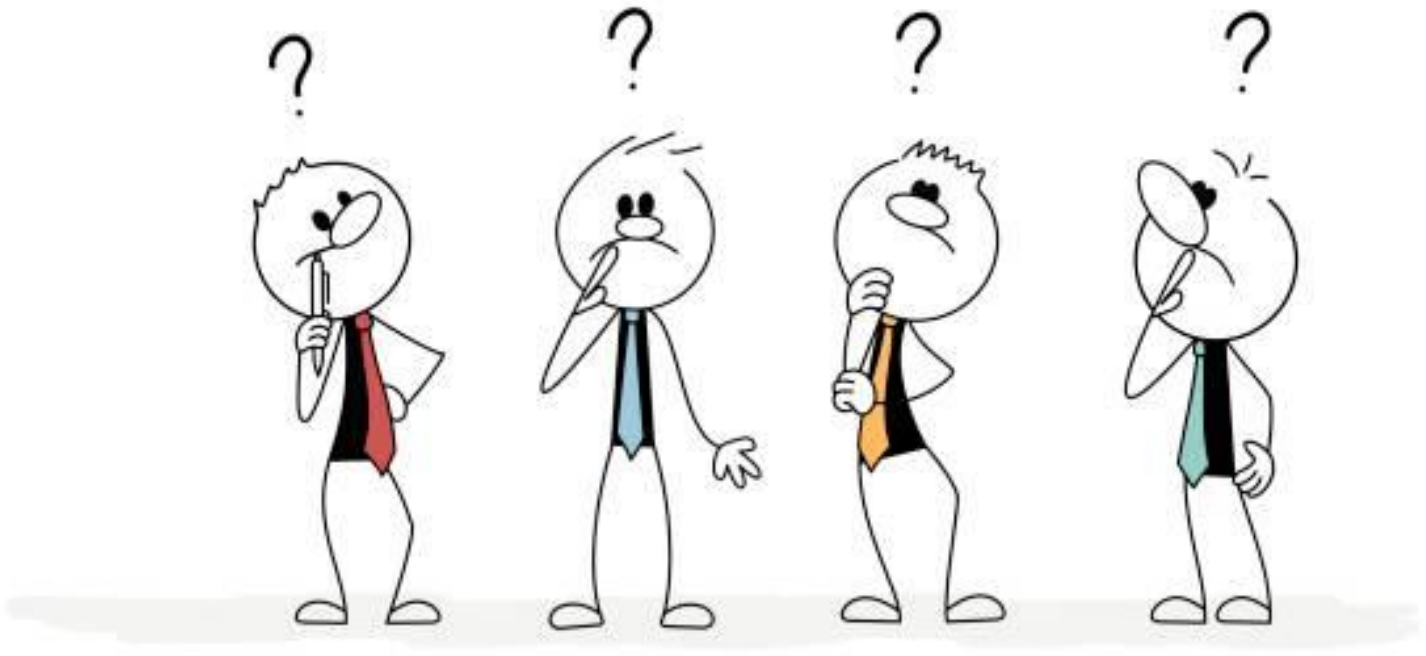
INTRODUÇÃO – PROGRAMAÇÃO CONCORRENTE

Um programa concorrente é executado simultaneamente por diversos processos que cooperam entre si.

INTRODUÇÃO – PROGRAMAÇÃO CONCORRENTE

Mas o que é cooperar?

Porque se chama programação concorrente?

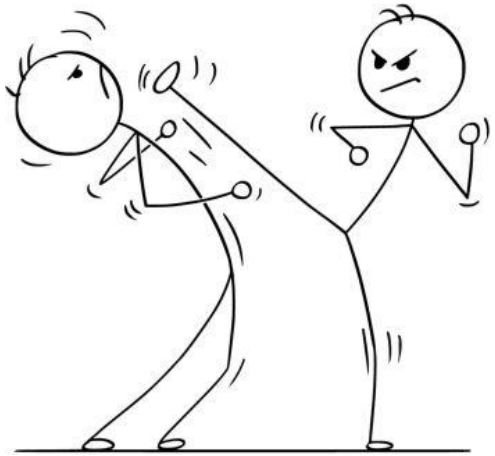


INTRODUÇÃO – PROGRAMAÇÃO CONCORRENTE

Origem: *concurrent programming* com o significado de "programação que ocorre ao mesmo tempo".

Em português, o verbo concorrer admite também o significado de **cooperar**.

Não vamos entender que na programação concorrente os processos estão em desacordo!



PROGRAMAÇÃO SEQUENCIAL VS. PROGRAMAÇÃO CONCORRENTE

É comum em sistemas multiusuário, que um programa seja executado simultaneamente por mais de um usuário.

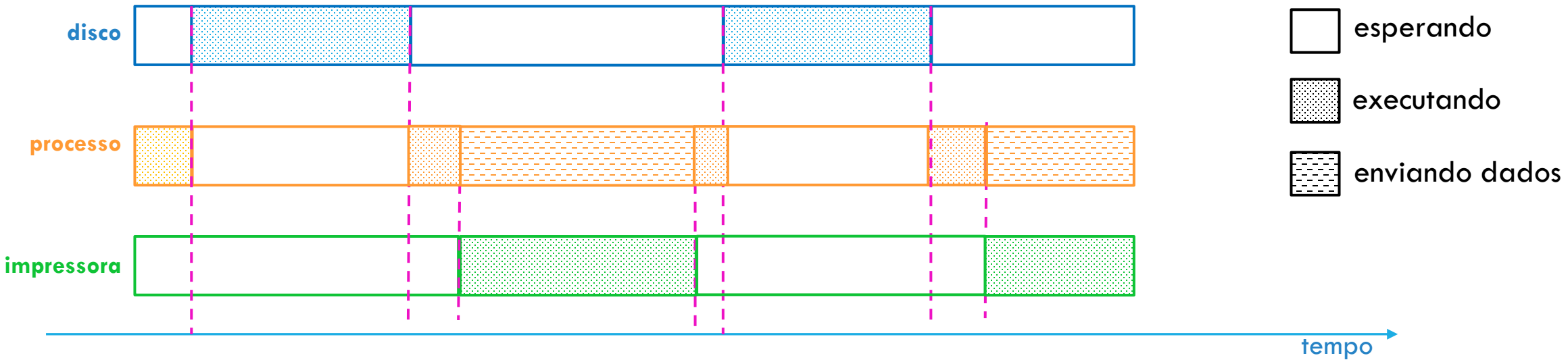
CUIDADO!!

Executar simultaneamente 10 instâncias de um programa, não faz dele um programa concorrente.

PROGRAMAÇÃO SEQUENCIAL VS. PROGRAMAÇÃO CONCORRENTE

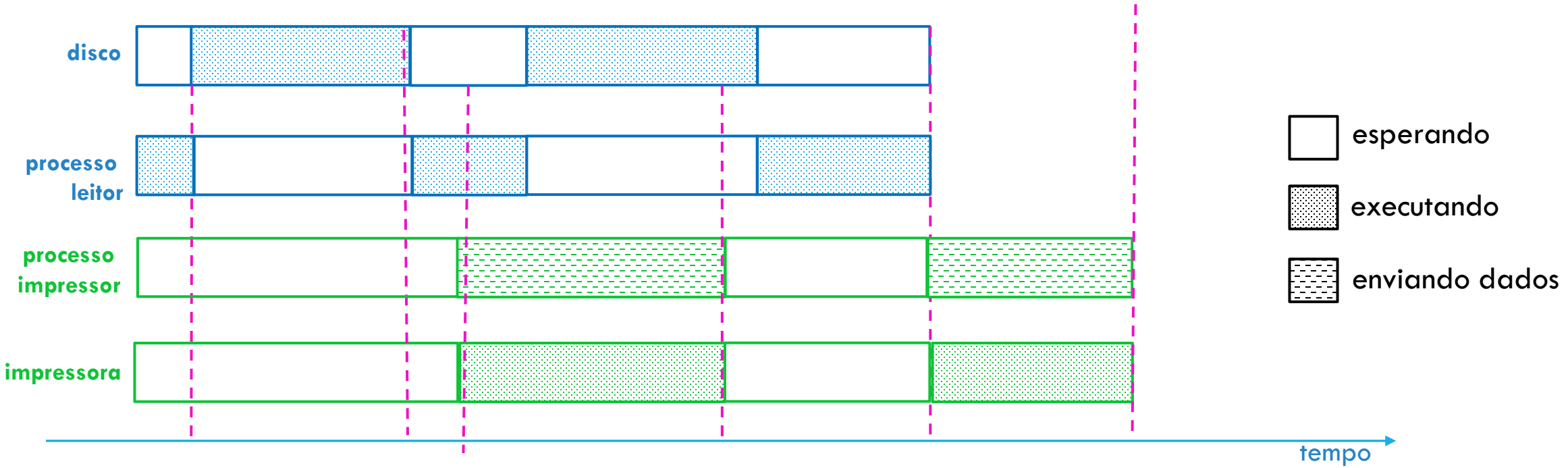
Neste caso, os processos não cooperam, ou seja, não trocam informações!

EXEMPLO :: PROGRAMAÇÃO SEQUENCIAL



Fonte: Sistemas Operacionais – R. Oliveira, A. Carissimi, S. Toscani

EXEMPLO :: PROGRAMAÇÃO CONCORRENTE



Fonte: Sistemas Operacionais – R. Oliveira, A. Carissimi, S. Toscani

PROGRAMAÇÃO CONCORRENTE

Naturalmente, criar processos concorrentes é mais complicado do que lidar com processos independentes.

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 mecanismos de 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

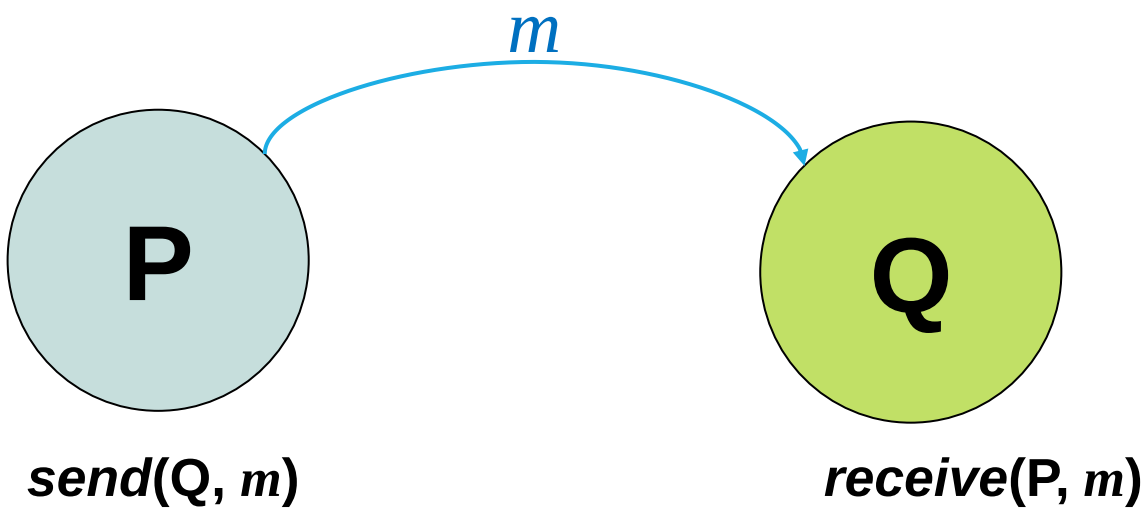
A troca de mensagens é feita baseada em duas ***primitivas***:

- *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

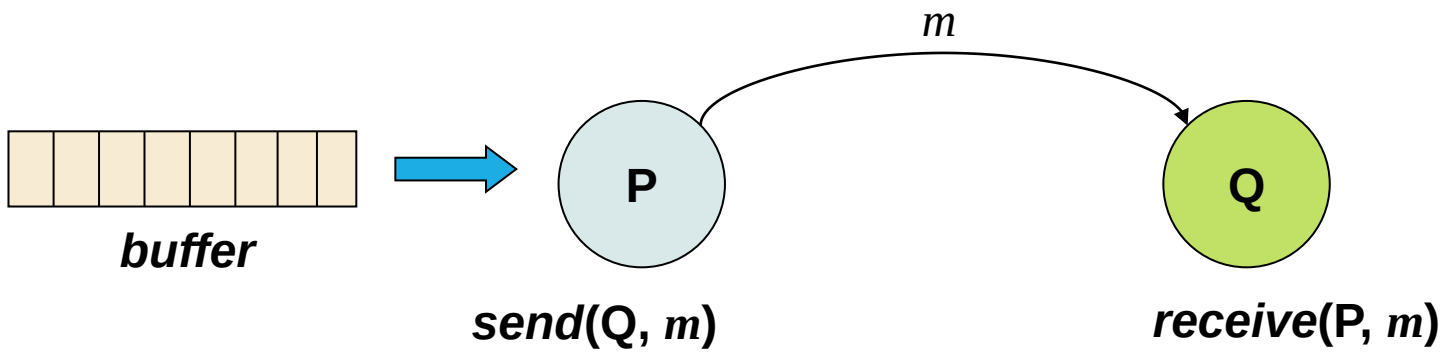
Troca de mensagens por sincronização:

- 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*
- *Bounded-capacity*
- *Unbouded-capacity*



COMUNICAÇÃO ENTRE PROCESSOS :: COMPARTILHAMENTO DE MEMÓRIA

Processos trocam informações através de leituras e escritas numa área compartilhada;

O que os processos precisam garantir??

PARA PENSAR UM POUCO...

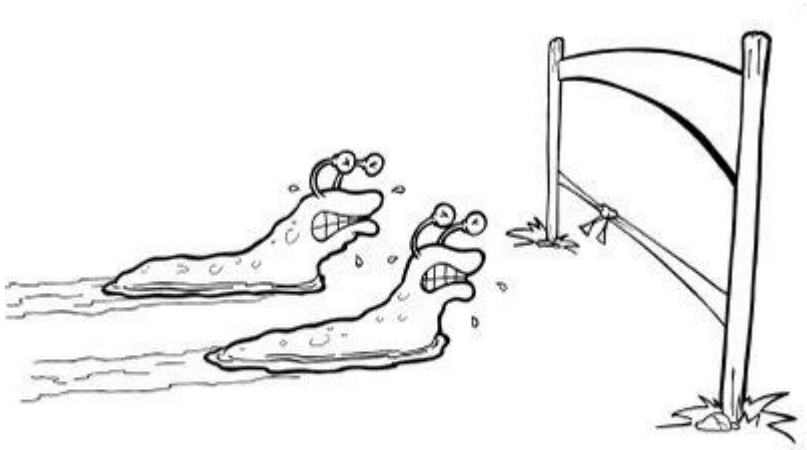
O que acontece quando dois processos querem escrever na mesma área de memória no mesmo instante?

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

Suponha dois processos, que alteram o valor da variável compartilhada *x*

P_1: $x := x + 1$
P_2: $x := x + 2$

Considere $x = 2$

P_1 ➔ P_2 : $x = 5$
P_2 ➔ P_1 : $x = 5$

P_1: $x := x + 1$
P_2: $x := x * 2$

Considere $x = 2$

P_1 ➔ P_2 : $x = 6$
P_2 ➔ P_1 : $x = 5$

COMUNICAÇÃO ENTRE PROCESSOS :: REGIÃO CRÍTICA

- Como evitar condições de corrida?
  - Sincronizando os processos

ou seja

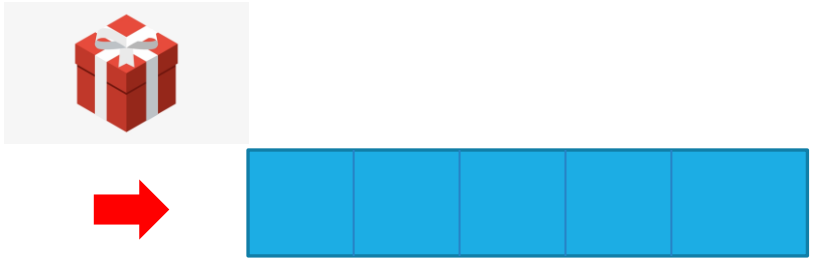
  - Proibindo que mais de um processo possa ler ou escrever numa área compartilhada ao mesmo tempo.

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??

PRODUTOR



CONSUMIDOR

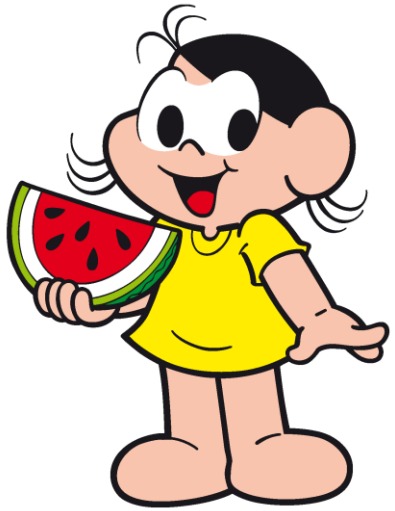
PRODUTOR *VS.* CONSUMIDOR

Condição de corrida: se dá na disputa pelo “prato”

Tainan :: Produtor

Região crítica

Variável compartilhada



Liane :: Consumidora

Exclusão Mútua: serve para garantir que Liane só vai pegar o prato quando ele estiver pronto.
E que Isaac vai aguardar o prato ficar vazio para fazer a reposição

Para pensar...

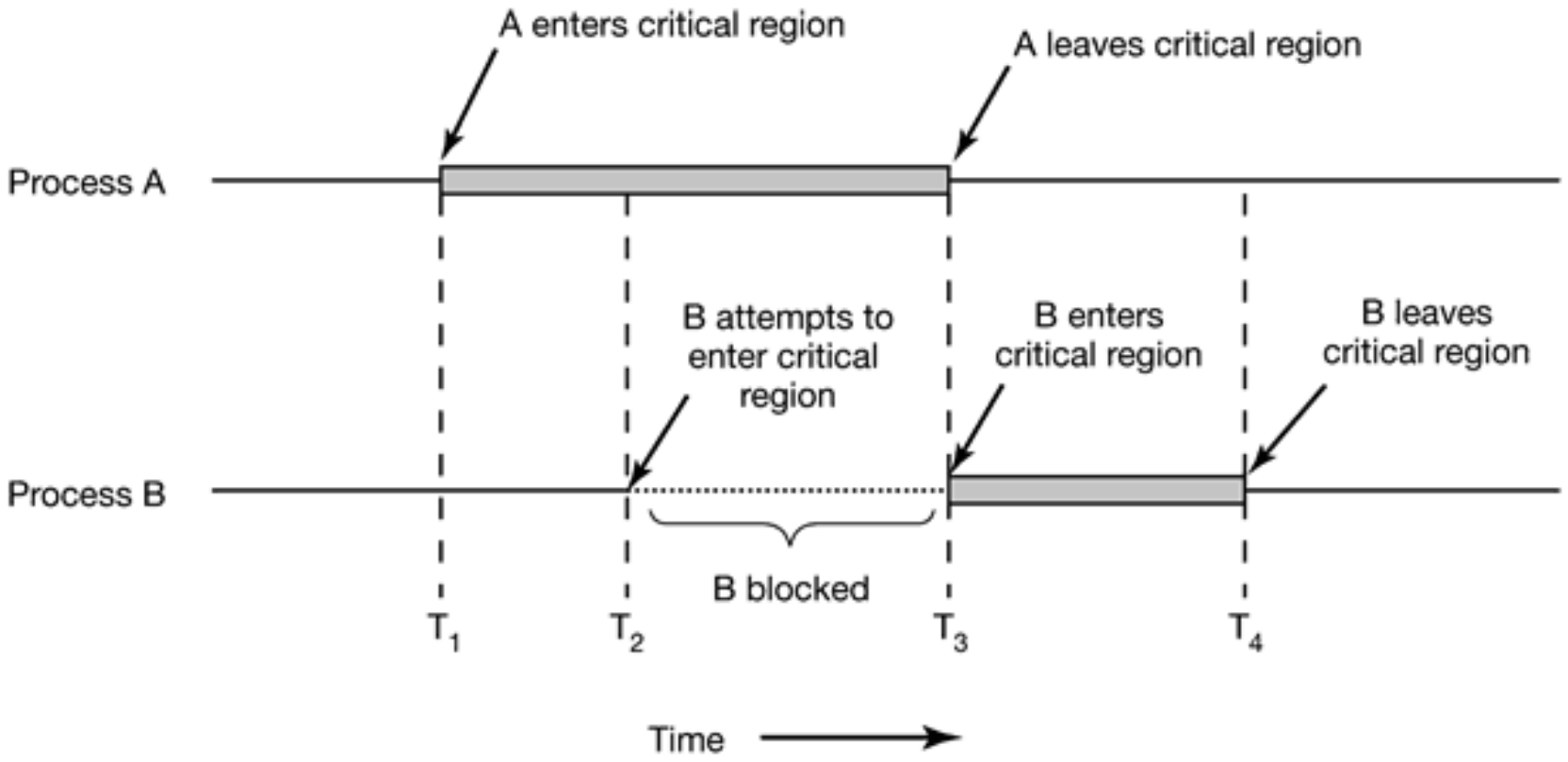
- Pense no problema do PRODUTOR *vs*. CONSUMIDOR.

- Quais os três cenários mais relevantes deste problema?

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

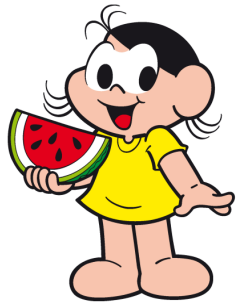
PRODUTOR *VS.* CONSUMIDOR :: BUFFER

Condição de corrida: se dá na disputa pelo “prato”

Isaac :: Produtor

Buffer :: Região crítica

Capacidade 3 posições



Lidia :: Consumidora

N PRODUTORES *VS. M* CONSUMIDORES

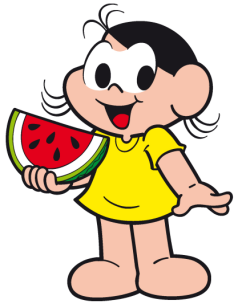
Condição de corrida: se dá na disputa pelo “prato”

Isaac

Jaqueline

Buffer :: Região crítica

Capacidade 3 posições



Lídia

Lucas

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);
    }
}
```

Comunicação de Processos
(-- Problemas clássicos --)

- Jantar dos filósofos
- Escritores e Leitores
- Barbeiro dorminhoco

Comunicação entre processos
(-- O jantar dos filósofos --)

Comunicação entre processos (-- O jantar dos filósofos --)

- Formulado por E. Dijkstra para caracterizar o problema da sincronização e concorrência

- Descrição
  - 5 filósofos numa mesa de jantar circular
  - 5 pratos de espaguete
  - 1 garfo entre cada par de pratos

Comunicação entre processos (-- O jantar dos filósofos --)

- Descrição
  - Cada filósofo pode “comer” ou “pensar”
  - Cada filósofo usa dois garfos para comer
  - Cada filósofo pega um garfo por vez

Jantar dos filósofos
(-- 1ª solução --)

```
#define N 5

void philosopher (int i)
{
    while (TRUE)
    {
        think();
        take_fork (i);
        take_fork ((i+1) % N);
        eat();
        put_fork (i);
        put_fork ((i+1) % N);
    }
}
```

Jantar dos filósofos
(-- 2ª solução --)

```
#define N 5

void philosopher (int i)
{
    while (TRUE)
    {
        think();
        take_fork (i);
        if (fork((i+1) % N) is available)
        {
            take_fork ((i+1) % N);
            eat();
            put_fork (i);
            put_fork ((i+1) % N);
         }
        else
           put_fork (i);
    }
}
```

Jantar dos filósofos
(-- 3ª solução --)

```
#define N 5

void philosopher (int i)
{
   while (TRUE)
   {
      think();
      down(mutex);
      take_fork (i);
      take_fork ((i+1) % N);
      eat();
      put_fork (i);
      put_fork ((i+1) % N);
      up(mutex);
   }
}
```

mutex = mutual exclusion

1) Se **mutex = 0**, a região crítica está indisponível

2) Se **mutex = 1**, a região crítica pode ser acessada;

==========

mutex = 1

Down:
mutex --;

Up:
mutex ++;

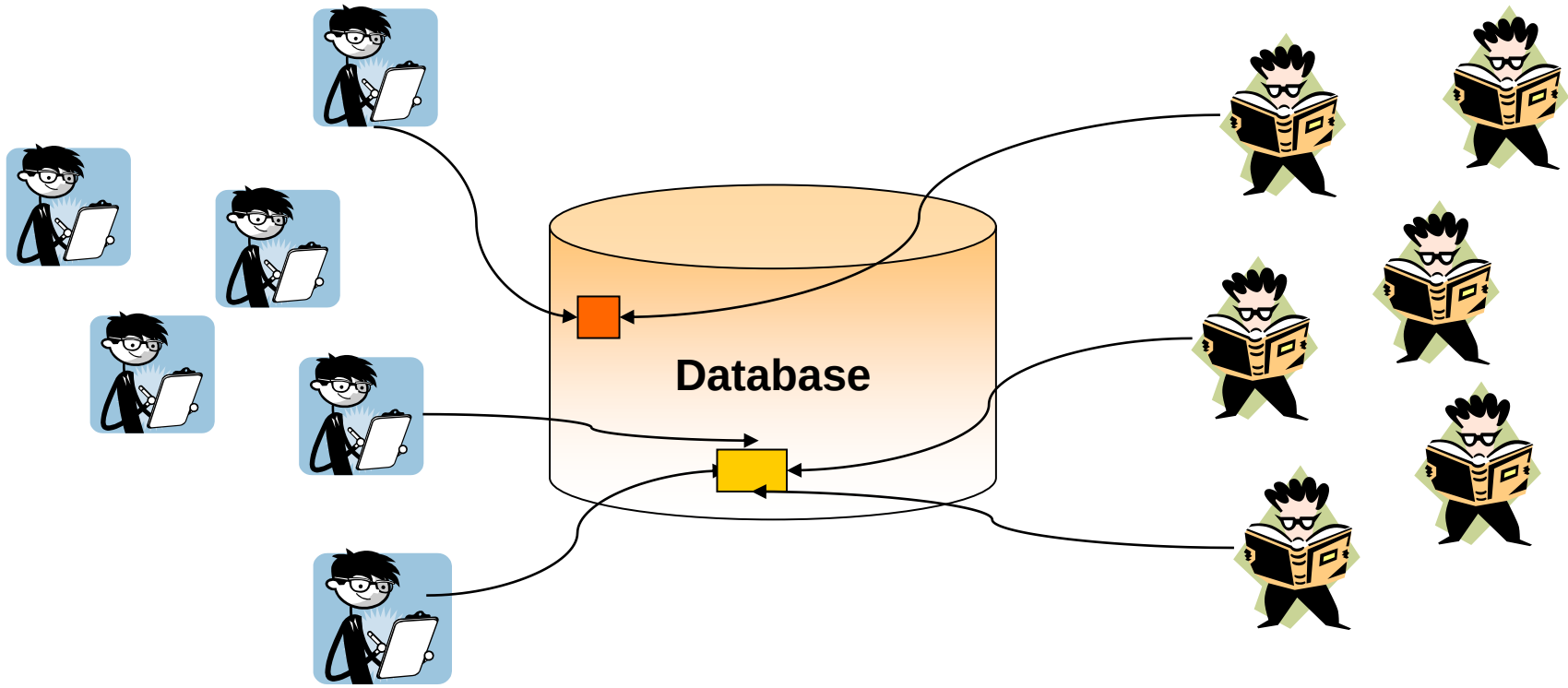
Jantar dos filósofos
(-- 4ª solução --)

- Atribui 3 possíveis estados aos filósofos
  - PENSANDO
  - COMENDO
  - FAMINTO

- Idéia:
  - Um filósofo no estado “faminto” só pode pegar os garfos se os seus vizinhos (esquerda e direita) não estiverem “comendo”.

- Estudar a solução para o problema dos filósofos!

Comunicação entre processos
(-- Os leitores e escritores --)



COMUNICAÇÃO ENTRE PROCESSOS (-- OS LEITORES E ESCRITORES --)

várias threads devem acessar a mesma memória compartilhada.

Algumas para ler e outras para escrever.

Restrições :

- Enquanto uma thread estiver escrevendo, nenhuma outra pode acessar este espaço de memória.
- Enquanto uma thread estiver lendo, somente outras que estejam querendo ler podem acessar este espaço de memória.

Comunicação entre processos
(-- Barbeiro dorminhoco --)

COMUNICAÇÃO ENTRE PROCESSOS (-- BARBEIRO DORMINHOCO --)

Se não há clientes, o barbeiro adormece;

Se a cadeira do barbeiro estiver livre, um cliente pode ser atendido imediatamente;

O cliente espera pelo barbeiro se houver uma cadeira de espera vazia;

Se não tiver onde sentar, o cliente vai embora.