

## Sistemas Operacionais II N

# Algoritmos para exclusão mútua entre dois processos

## Condições de Corrida

- “O resultado da computação depende da ordem em que as instruções são executadas”
- A ordem não é determinística com T.S.
- Condições de corrida **devem ser evitadas!**

### Exemplo

## Seções ou Regiões Críticas

- “Parte do código que acessa dados compartilhados e os deixa em estados intermediários inconsistentes”
- Garantir **exclusão mútua**: somente um processo pode executar dentro da seção (ou região) crítica ao mesmo tempo

## Condições para Concorrência

1. Dois processos não podem executar dentro da região crítica ao mesmo tempo
2. Não pode haver nenhuma suposição sobre a velocidade de execução ou número de CPUs.
3. Nenhum processo fora da região crítica pode bloquear outro processo
4. Não pode haver espera “infinita” para entrar na região crítica

### Primitivas

- diversos mecanismos para atingir esses objetivos

### Tentativa 1

Variável: em\_uso: boolean initial false;

Código:

```
loop
  ...
  loop
    exit when not em_uso
  endloop
  em_uso := true
  REGIÃO CRÍTICA
  em_uso := false
  ...
endloop
```

### Tentativa 2

Variável: vez: integer initial 1;

Processo 1: eu = 1, outro = 2;

Processo 2: eu = 2, outro = 1;

Código:

```
loop
  ...
  loop
    exit when vez == eu
  endloop
  REGIÃO CRÍTICA
  vez := outro
  ...
endloop
```

### Tentativa 3

**Variável:** quer: array[1..2] of bool initial [false,false];  
**P1:** eu = 1, outro = 2; **P2:** eu = 2, outro = 1;

**Código:**

```
loop
  ...
  loop
    exit when not quer[outro]
  endloop
  quer[eu] := true
  REGIÃO CRÍTICA
  quer[eu] := false
  ...
endloop
```

### Tentativa 4

**Variável:** quer: array[1..2] of bool initial [false,false];  
**P1:** eu = 1, outro = 2; **P2:** eu = 2, outro = 1;

**Código:**

```
loop
  ...
  quer[eu] := true
  loop
    exit when not quer[outro]
  endloop
  REGIÃO CRÍTICA
  quer[eu] := false
  ...
endloop
```

### Tentativa 5

**Variável:** quer: array[1..2] of bool initial [false,false];  
**P1:** eu = 1, outro = 2; **P2:** eu = 2, outro = 1;

**Código:**

```
...
ini: quer[eu] := true
if quer[outro] then
  begin
    quer[eu] := false
    goto inicio
  end
  REGIÃO CRÍTICA
  quer[eu] := false
```

### Solução de Dekker (60s)

**Variáveis:** quer: array[1..2] of bool initial [false,false];  
**vez:** integer initial 1; **P1:** eu=1, outro=2; **P2:** eu=2, outro=1;

**Código:**

```
...
init: quer[eu] := true
again: if quer[outro] then
  begin
    if vez==eu then goto again
    quer[eu] := false
    loop exit when vez == eu
  end loop
  goto init
end
  REGIÃO CRÍTICA
  quer[eu] := false
  vez := outro
```

### Solução de Peterson (80s) – version 1.0

**Variáveis:** quer: array[1..2] of bool initial [false,false];  
**vez:** integer initial 1;

**P1:** eu=1, outro=2; **P2:** eu=2, outro=1;

**Código:**

```
...
quer[eu] := true
vez := outro
loop
  exit when not quer[outro] or vez==eu
end loop
  REGIÃO CRÍTICA
  quer[eu] := false
  ...
```

### Solução de Peterson (80s) – version 2.0

**Variáveis:** want: array[1..2] of bool initial [false,false];  
**last:** integer;

**P1:** me=1, other=2; **P2:** me =2, other =1;

**Código:**

```
...
want[me] := true
last := me
loop
  exit when not want[other] or last != me
end loop
  REGIÃO CRÍTICA
  want[me] := false
  ...
```

### Algoritmos para exclusão mútua

- Evolução: idéias e 5 tentativas
- Solução de Dekker
- Solução de Peterson

### Características de todos

- Processos cíclicos
- Utilizando “**busy wait**”
- Controle por variáveis globais: **compartilhadas**
- Os dois processos executam o mesmo protocolo de acesso

### Solução com a instrução TAS

### Solução com instrução Test and Set

Variável: livre: boolean initial true;

Código:

```
loop
    ...
    loop
        exit when TAS(livre)
        // was true? livre := false
    endloop
    REGIÃO CRÍTICA
    livre := true
    ...
endloop
```