

Sistemas de Tempo Real

Exclusão Mútua em sistemas
multithread



Exclusão Mútua

» Quando dois ou mais programas têm suas instruções entrelaçadas, ao executar trechos críticos, podemos ter um comportamento instável, com resultados que dependem da ordem de execução destas instruções;

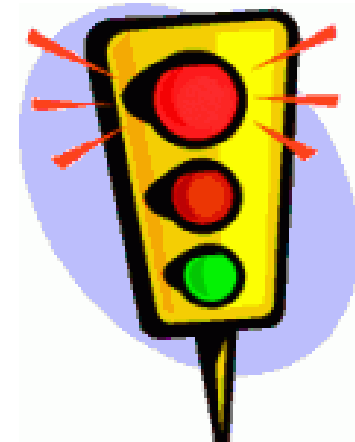
A collage of images related to engineering and education. On the left, a whiteboard with the text 'I LOVE TRANSISTORS' and a stick figure. In the center, a young child looking at something. On the right, a snippet of C++ code and a yellow circuit board.

Exclusão Mútua

- » Situações onde dois ou mais processos estão lendo ou escrevendo algum dado compartilhado e o resultado final depende de uma determinada ordem de execução, são chamadas de condições de disputa.
- » **exclusão mútua:** Forma de certificar que se um processo (ou thread) está dentro de sua seção crítica (trecho do código onde condição de disputa por dados compartilhados ocorre), o outros processos (ou thread) vão ser impedidos de fazerem o mesmo;

Semáforo

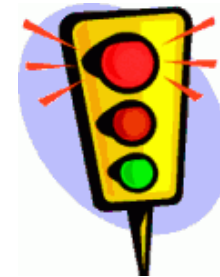
- » Um dos recursos para a resolução do problema da exclusão mútua;
- » Hoje, os semáforos são capazes de resolver muitos outros problemas além da exclusão mútua.



Semáforo

» Um semáforo é um objeto de uma classe que possui um membro de dado inteiro “S”(contador do semáforo) e dois métodos que definem as únicas operações permitidas:

- Aguarda / Bloqueia (primitiva P)
- Continua / Libera (primitiva V)



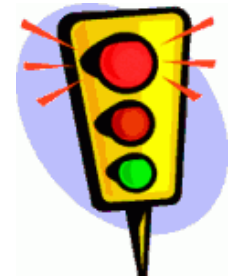
» Semáforos podem ser usados para implementar guardas no código de acesso a dados compartilhados.

Semáforo – variável

```
int semaforo = 0; //global
```

```
while(1){  
    if(semaforo == 0){  
        semaforo = 1;  
        atuadores;  
        semaforo = 0;  
    }  
}
```

```
while(1){  
    if(semaforo == 0){  
        semaforo = 1;  
        sensores;  
        semaforo = 0;  
    }  
}
```



Controle realizado pelo programador!

Semáforo – Lego NXT

```
mutex semaforo;
```

```
task1:
```

```
while(true){
```

```
    Acquire(semaforo);
```

```
    atuadores;
```

```
    Release(semaforo);
```

```
}
```

```
task2:
```

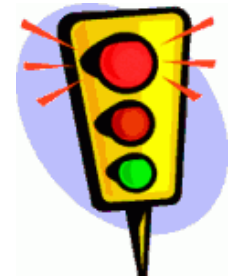
```
while(true){
```

```
    Acquire(semaforo);
```

```
    sensores;
```

```
    Release(semaforo);
```

```
}
```



Controle realizado pelo Sistema Operacional!

Threads – Lego NXT

```
Task main(){
```

```
//definição dos sensores
```

```
Precedes(task1, task2, ...);
```

```
}
```

→ main() encerra execução após o Precedes.



Questões importantes em Multithreading



- » Desempenho: a mudança de contexto dos diversos threads provoca queda de desempenho (overhead).
- » Segurança: como sincronizar threads para que elas não interfiram com uma outra.
- » Longevidade: como evitar situações de deadlock ou livelock para garantir que todos os threads farão progresso.



Deadlock x Livelock

» Deadlock:

- » Situação na qual duas ou mais unidades concorrentes não conseguem prosseguir a execução por que cada uma está aguardando que alguma das outras faça alguma coisa.

» Livelock

- » Situação na qual uma unidade concorrente não consegue terminar a execução ou entrar em uma seção crítica por excesso de trabalho ou falta de velocidade. Difere de deadlock por estar ativa e não bloqueada ou aguardando algo.



Comunicação entre tarefas

- » Tarefas podem exigir comunicação para fins de:
- cooperação: T1 necessita um serviço de T2 para prosseguir execução (aguarda serviço).
 - competição: T1 precisa um recurso que T2 está usando (aguarda recurso).