


Secções Críticas

Sistemas Operativos – DEI - IST

1



Concorrência

- Vimos na ultima aula o programa que incrementa uma variável concorrentemente
- ***O programa estava errado!!!***
- Relembrando o que se passava

Sistemas Operativos – DEI - IST

2

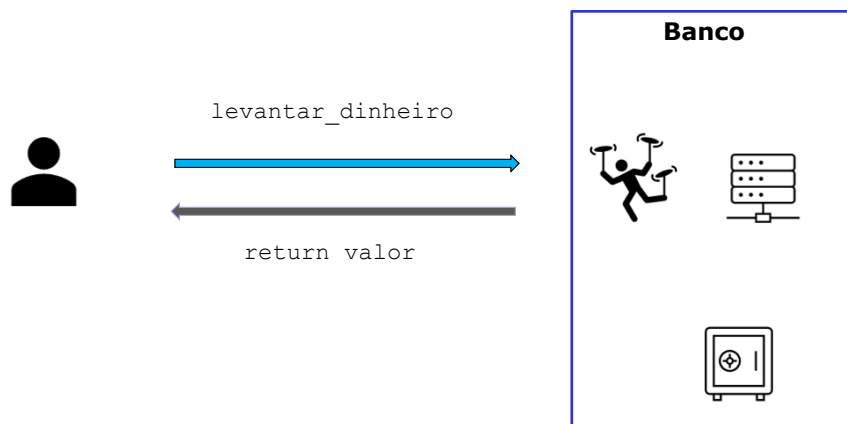


Exemplo do Mundo Real

- Para visualizar melhor o problema temos um exemplo de um sistema de actualizações de contas bancárias
- Suponhamos que temos um servidor que recebe pedidos para debito e créditos numa conta bancária
- Para tornar o servidor eficiente **cada pedido é servido por uma tarefa independente**

Sistemas Operativos – DEI - IST

3



Sistemas Operativos – DEI - IST

4



Função que debita um valor numa conta bancária

```
struct {
    int saldo;
    /* outras variáveis, ex. nome do titular, etc. */
} conta_t;

int levantar_dinheiro(conta_t* conta, int valor) {
    if (conta->saldo >= valor) {
        conta->saldo = conta->saldo - valor;
    }
    else {
        valor = -1;
        return valor;
    }
}
```

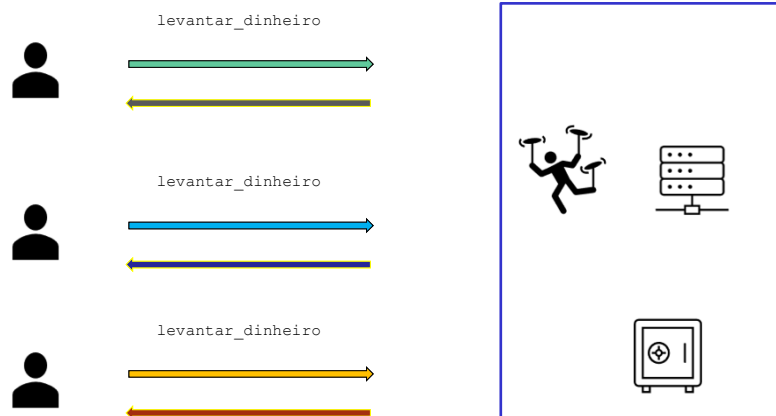
Versão muito simplificada

Sistemas Operativos – DE1 - IST

5

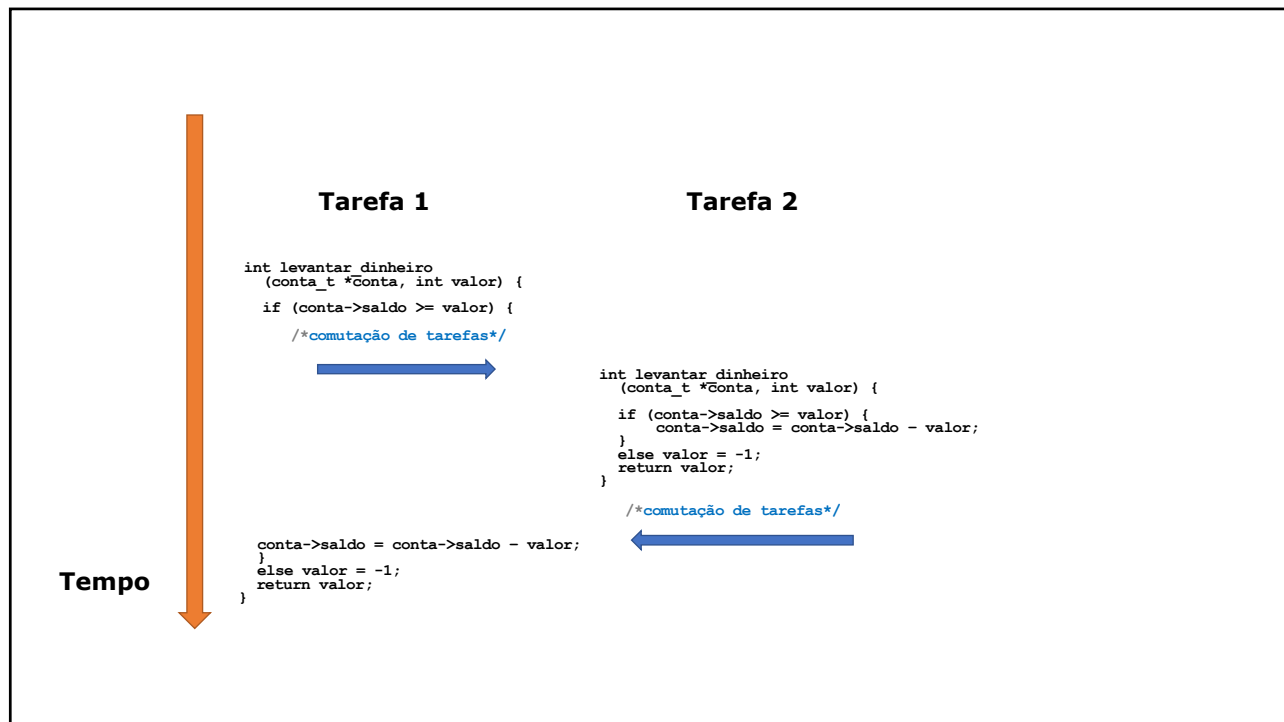


Exemplo com vários clientes em simultâneo

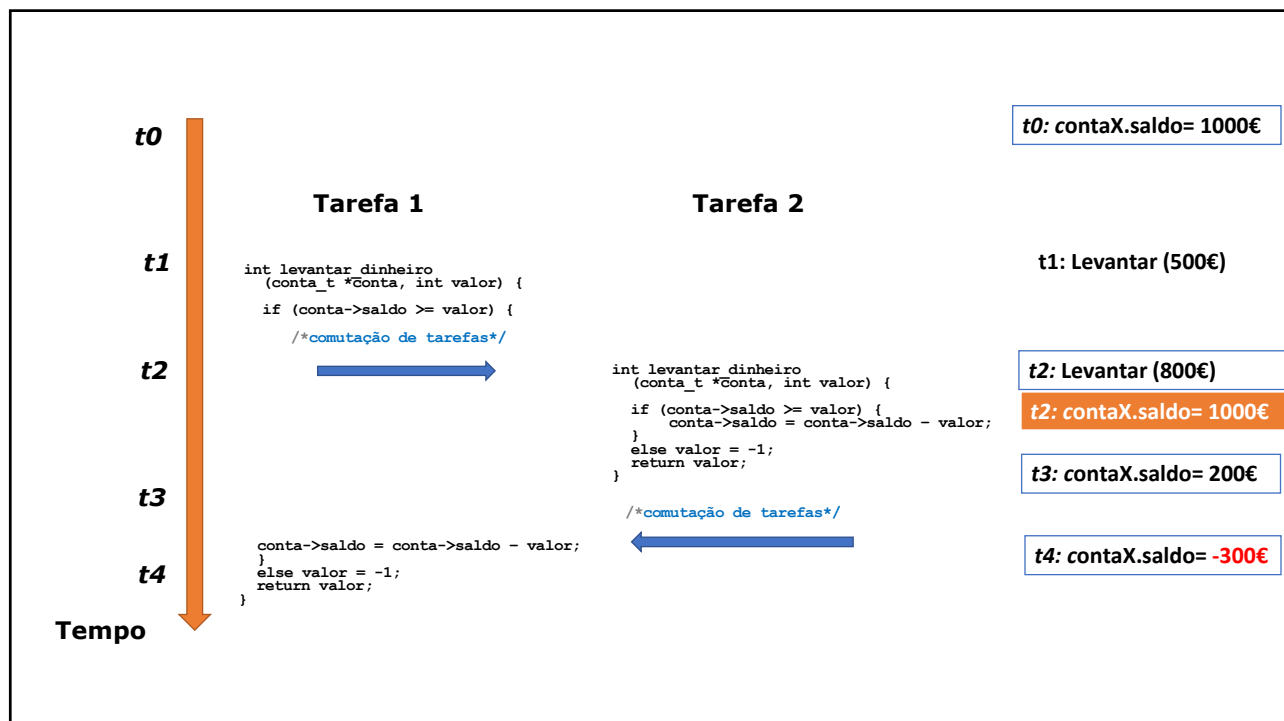


Sistemas Operativos – DE1 - IST

6



7



8



Secção Crítica

- O programa está errado
- Porque está errado?
- **Porque a sequência de testar o valor do saldo da conta e actualizá-lo tem de ser indivisível (atómica)**
- Não podemos testar, perder o processador e depois continuar com o valor anterior porque este pode já ter sido alterado

Sistemas Operativos – DEI - IST

10



Mas quando pode ocorrer a comutação?

- A comutação do núcleo aparece quando o Despacho é chamado na sequência de uma interrupção, normalmente do *timer*
- As interrupções só são aceites no final da execução de uma instrução máquina
- **Se a operação fosse só uma instrução não haveria possibilidade do núcleo comutar a tarefa**

11



Mas não é uma única instrução

```
if (conta->saldo >= valor) {
    conta->saldo = conta->saldo - valor;
```

;assumindo que a variável conta->saldo está na posição SALDO da memória

;assumindo que variável valor está na posição VALOR da memória

```
mov AX, SALDO ;carrega conteúdo da posição de memória
               ;SALDO para registo geral AX
mov BX, VALOR ;carrega conteúdo da posição de memória
               ;VALOR para registo geral BX
sub AX, BX    ;efectua subtracção AX = AX - BX
mov SALDO, AX ;escreve resultado da subtracção na
               ;posição de memória SALDO
```

Sistemas Operativos – DEI - IST

12



Não sabemos o resultado da compilação

- O problema, é como já vimos com o exemplo de `glob++`, uma instrução de C pode gerar várias instruções de *assembler* pelo que o problema não pode ser resolvido desta forma

Precisamos de um novo conceito

Sistemas Operativos – DEI - IST

13



Secção Crítica

```
int levantar_dinheiro (ref *conta, int valor)
{
    if (conta->saldo >= valor) {
        conta->saldo = conta->saldo - valor;
    } else valor = -1;
    return valor;
}
```

} Secção crítica

Sistemas Operativos – DEI - IST

14



Trinco Lógico

- Pode ser fechado ou aberto
- Uma vez fechado, outra tarefa que tente fechar espera até ser aberto
- Esta propriedade designa-se **Exclusão Mútua**



Frequentemente chamado *mutex*

Sistemas Operativos – DEI - IST

15



Secção Crítica

```
int levantar_dinheiro (ref *conta, int valor)
{
    lock(trinco);
    if (conta->saldo >= valor) {
        conta->saldo = conta->saldo - valor;
    } else valor = -1;
    unlock (trinco);
    return valor;
}
```

} Secção crítica

Sistemas Operativos – DEI - IST

16



Conclusões

A concorrência na execução das tarefas levanta o problema de que acesso a variáveis partilhadas tem de ser controlado pelos programas através da utilização de secções críticas

Sistemas Operativos – DEI - IST

17