



Programação concorrente multi-tarefa por memória partilhada

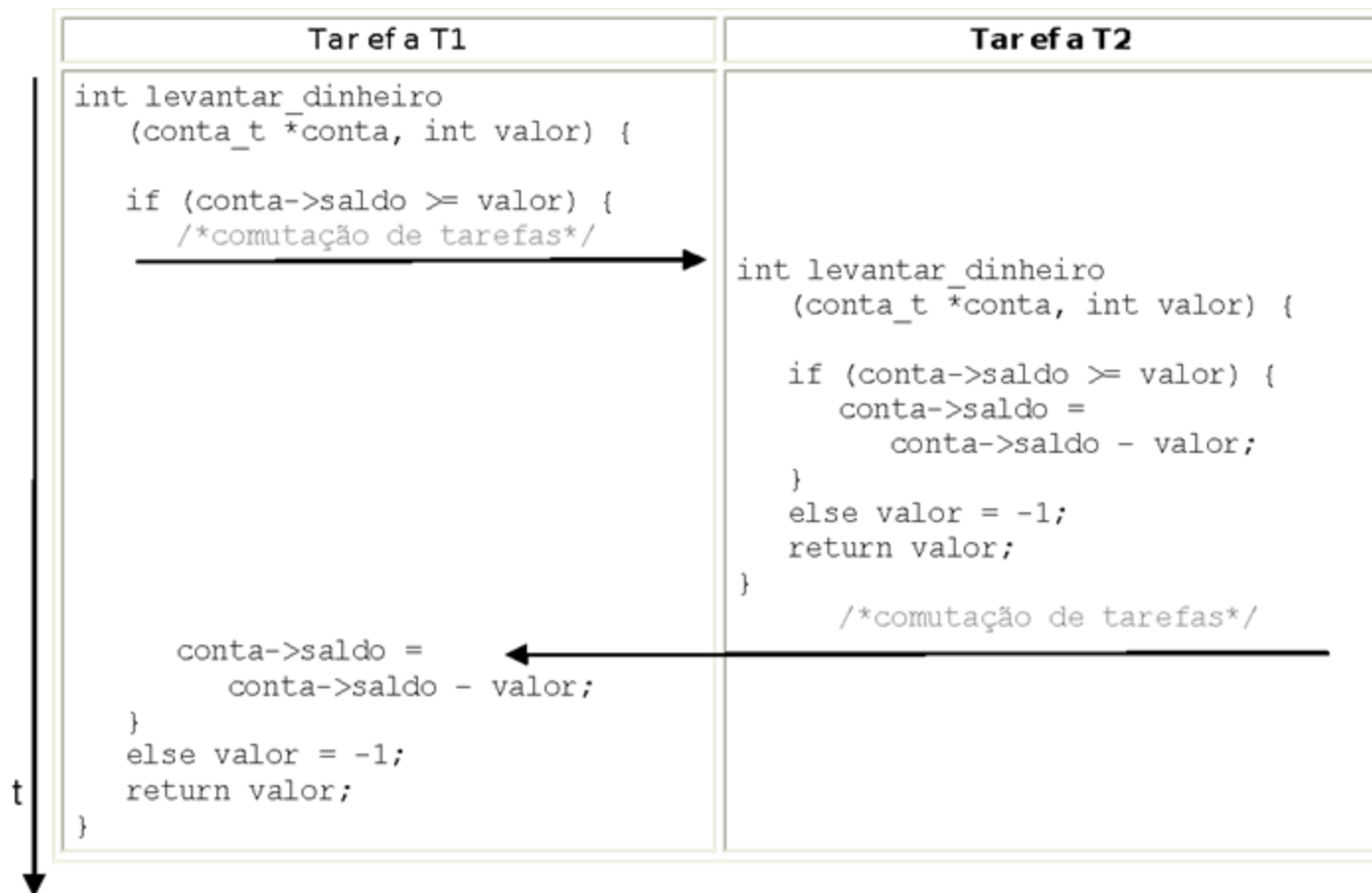
Sistemas Operativos

Execução Concorrente

```
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;  
  
    assert(conta->saldo >= 0);  
  
    return valor;  
}
```

Problema se for multi-tarefa?

Problema 1



E assim?

```
struct {  
    int saldo;  
    /* outras variáveis, ex. nome do titular, etc. */  
} conta_t;  
  
int levantar_dinheiro(conta_t* conta, int valor) {  
    conta->saldo = conta->saldo - valor;  
    return valor;  
}
```

Se a função for chamada N vezes, o que pode
correr mal?

Problema 2

;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
```

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

Secção Crítica

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

} Secção crítica

O que gostaríamos que estas funções fizessem?



O que devemos impor quando uma tarefa entra numa secção crítica?

- Parar o resto do sistema?
- Barrar outras tarefas (do mesmo processo) que tentem entrar em qualquer secção crítica?
- Barrar outras tarefas (do mesmo processo) que tentem entrar em alguma secção crítica que aceda aos mesmos dados partilhados?

Objeto trinco lógico também chamado *mutex*

- Pode ser fechado ou aberto
- Uma vez fechado, outra tarefa que tente fechar espera até ser aberto
- Esta propriedade chama-se exclusão mútua
- Usar trincos diferentes para secções críticas independentes:
 - maximizar paralelismo!





```
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; /* -1 indica erro ocorrido */  
    return valor;  
}
```

Mesmo exemplo com mutex

```
trinco_t t;

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

int levantar_dinheiro(conta_t* conta, int valor) {
    fechar(t);
    if (conta->saldo >= valor)
        conta->saldo = conta->saldo - valor;
    else
        valor = -1; /* -1 indica erro ocorrido */
    abrir(t);
    return valor;
}
```

Solução ainda melhor!...

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

int levantar_dinheiro(conta_t* conta, int valor) {
    fechar(conta->t);
    if (conta->saldo >= valor)
        conta->saldo = conta->saldo - valor;
    else
        valor = -1; /* -1 indica erro ocorrido */
    abrir(conta->t);
    return valor;
}
```



Interface POSIX para mutexes

```
int pthread_mutex_init(pthread_mutex_t *mutex,  
                        pthread_mutexattr_t *attr);  
int pthread_mutex_lock(pthread_mutex_t *mutex);  
int pthread_mutex_unlock(pthread_mutex_t *mutex);  
int pthread_mutex_trylock(pthread_mutex_t *mutex);  
int pthread_mutex_timedlock(pthread_mutex_t *mutex,  
                             const struct timespec *timeout);
```

Exemplo:

```
//no início do main  
pthread_mutex_t count_lock;  
pthread_mutex_init(&count_lock, NULL);  
...  
//todas as secções críticas do programa  
pthread_mutex_lock(&count_lock);  
count++;  
pthread_mutex_unlock(&count_lock);
```



Preciso mesmo usar mutex?



Preciso mesmo usar mutex?

Exemplo

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

int levantar_dinheiro(conta_t* conta, int valor) {
    pthread_mutex_lock(&conta->mutex);
    if (conta->saldo >= valor) {
        conta->saldo = conta->saldo - valor;
        conta->numLevantamentos++;
        pthread_mutex_unlock(&conta->mutex);
    }
    else {
        pthread_mutex_unlock(&conta->mutex);
        valor = -1;
    }
    return valor;
}

int consultar(conta_t* conta, int *levantamentos) {
    int s;
    pthread_mutex_lock(&conta->mutex);
    if (levantamentos)
        *levantamentos = conta->numLevantamentos;
    s = conta->saldo;
    pthread_mutex_unlock(&conta->mutex);
    return s;
}
```

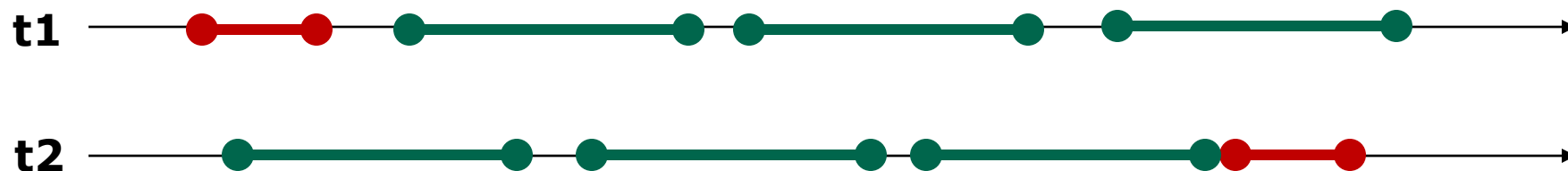
**Se esta função só
lê, é mesmo
preciso mutex?**

**Não usar trinco:
arriscamos resultado
inconsistente!**

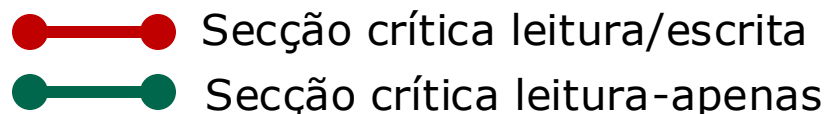
**Usar trinco: proíbe
que duas tarefas
executem a função em
paralelo**

O custo da sincronização (com mutex)

- Sem sincronizar
 - Bom desempenho mas **incorreto!**



- Sincronizando com mutex



Trincos de leitura-escrita

Podem ser fechados de duas formas:

fechar para ler ou fechar para escrever

Semântica:

Os escritores só podem aceder em exclusão mútua

Os leitores podem aceder simultaneamente com outros leitores mas em exclusão mútua com os escritores

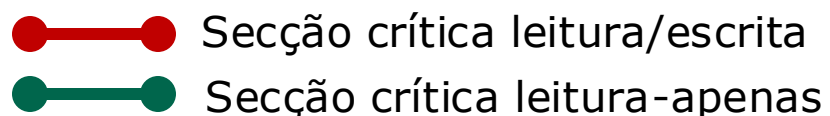
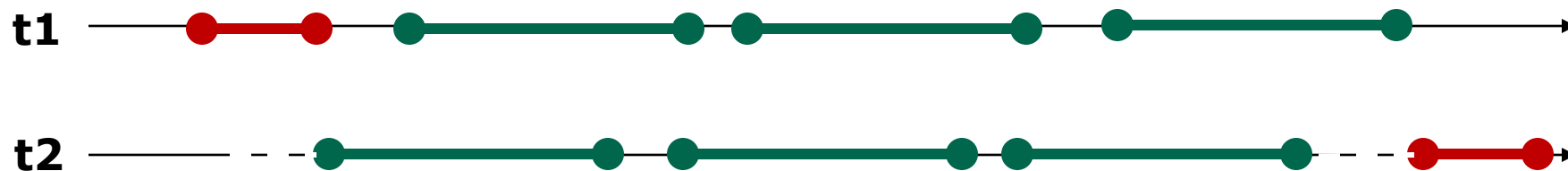
Vantajoso quando acessos a secções críticas de leitura são dominantes

O custo da sincronização (com mutex)

- Sincronizando com mutex



- Sincronizando com trinco de leitura-escrita





Trincos de leitura-escrita em POSIX

Tipo de dados: `pthread_rwlock_t`

Fechar para ler:

```
int pthread_rwlock_rdlock(pthread_rwlock_t *lock);
```

Fechar para escrever:

```
int pthread_rwlock_wrlock(pthread_rwlock_t *lock);
```

Abrir:

```
int pthread_rwlock_unlock(pthread_rwlock_t *lock);
```

Mais variantes disponíveis

`trylock`, `timedlock`, etc. Ver *man pages*.



Mesmo exemplo com trincos de leitura-escrita

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

int levantar_dinheiro(conta_t* conta, int valor) {
    pthread_rwlock_wrlock(&conta->rwlock);
    if (conta->saldo >= valor) {
        conta->saldo = conta->saldo - valor;
        conta->numLevantamentos ++;
    }
    else valor = -1;
    pthread_rwlock_unlock(&conta->rwlock);
    return valor;
}

int consultar (conta_t* conta, int *levantamentos) {
    int s;
    pthread_rwlock_rdlock(&conta->rwlock);
    if (levantamentos)
        *levantamentos = conta->numLevantamentos;
    s = conta->saldo;
    pthread_rwlock_unlock(&conta->rwlock);
    return s;
}
```