Introdução à programação concorrente

Sistemas Operativos

2018 - 2019

Paralelismo e coordenação

Motivação

- Muitos problemas reais não podem ser resolvidos (corretamente) em paralelo sem coordenação!

- Ex: preparar um bolo
  - coordenar acesso a recursos partilhados:
    - num forno só podem caber até N bolos
  - respeitar dependências lógicas dos vários passos da receita
    - podemos decorar um bolo, enquanto outro está no forno
    - mas não podemos decorar um bolo antes de o assar!

Resolver estes problemas impõe alguma forma de coordenação/comunicação!

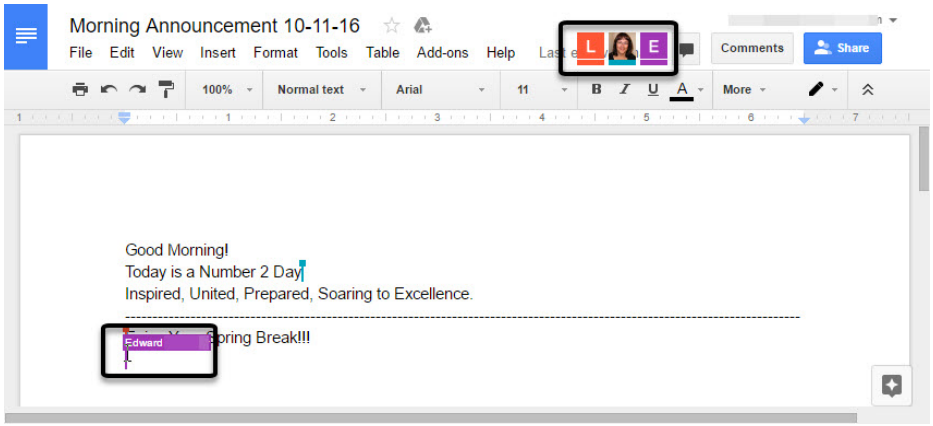
Dois paradigmas para programação concorrente

- Por troca de mensagens
  - Cada tarefa trabalha exclusivamente sobre dados privados
  - Tarefas transmitem dados trocando mensagens
  - Mensagens também servem para sincronizar tarefas
- Por memória partilhada
  - Tarefas partilham dados (no *heap*/amontoado)
  - Troca de dados é feita escrevendo e lendo da memória partilhada
  - Sincronização recorre a mecanismos adicionais (p.e., trincos, semaforos,…).

Analogia:
Edição concorrente de um documento

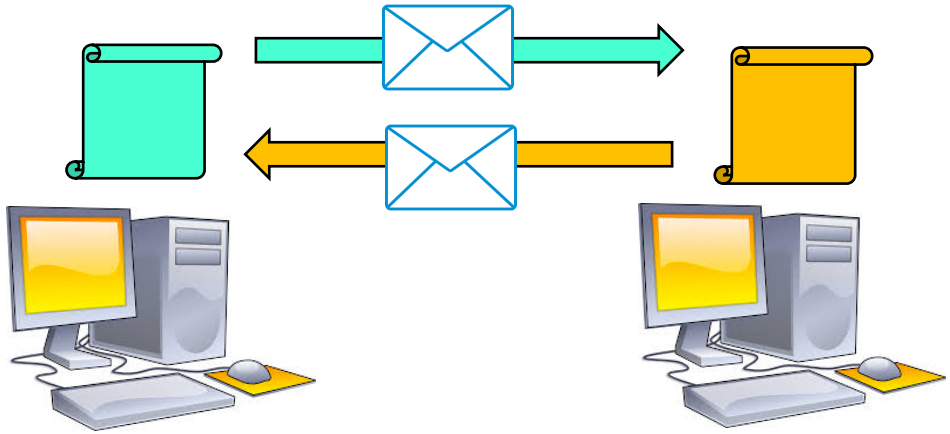
Memória partilhada

- Google docs
  - Única cópia online do documento
  - As alterações de um editor são imediatamente aplicadas ao documento partilhado e visíveis logo aos outros editores



Troca de mensagem

- Cada editor mantem uma cópia privada do documento no seu computador
- Alterações enviadas por email e aplicadas independentemente



Porquê diferentes paradigmas?

- Historicamente:
  - Algumas arquiteturas só permitiam que programas a correr em CPUs distintos trocassem mensagens
    - Cada CPU com a sua memória privada, interligados por alguma rede
  - Outras suportavam memória partilhada
    - E.g. CPUs podiam aceder à mesma memória RAM através de protocolo de coerência de cache

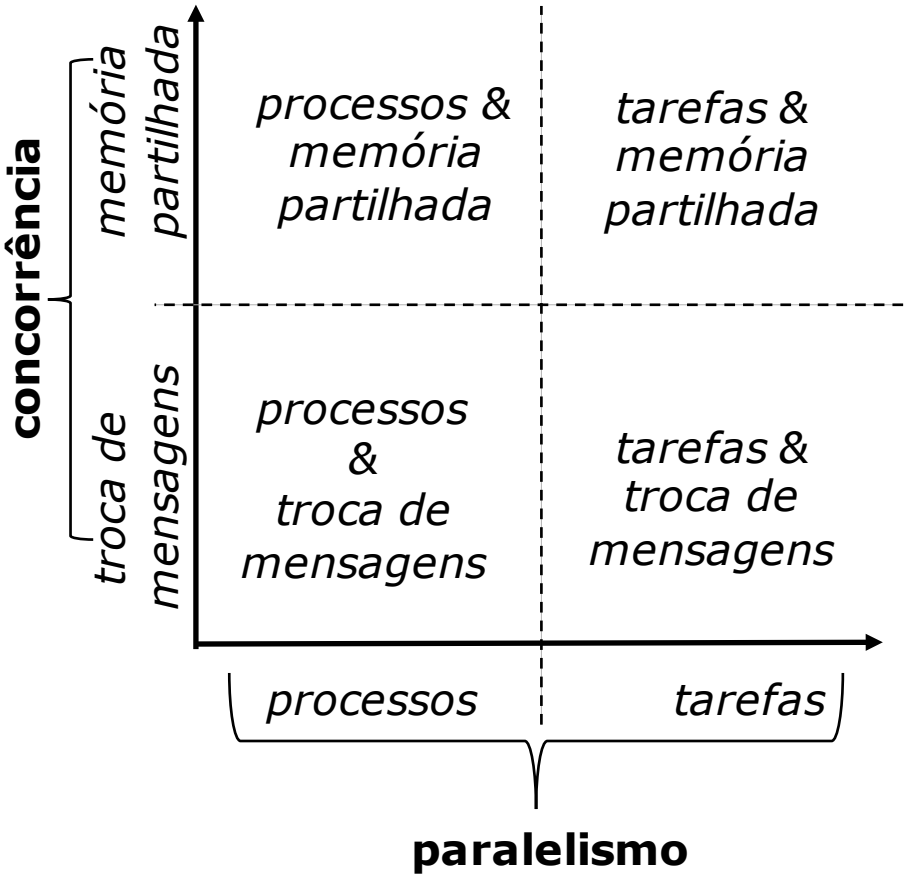
Porquê diferentes paradigmas?

- Estilos diferentes de programação, com virtudes e defeitos:
  - Diferentes ambientes de programação mais apropriados para cada paradigma
  - Preferências de cada programador
  - Alguns problemas mais fáceis de resolver eficientemente num paradigma que noutro

- Voltaremos a esta discussão daqui a umas semanas

Combinações de modelos de paralelismo e coordenação

- Dois modelos de paralelismo:
  1. por tarefa
  2. por processo
- Dois modelos de concorrência:
  1. por troca de mensagens
  2. por memória partilhada
- Os modelos de paralelismo e concorrência podem ser combinados!
  - resultado: 4 alternativas



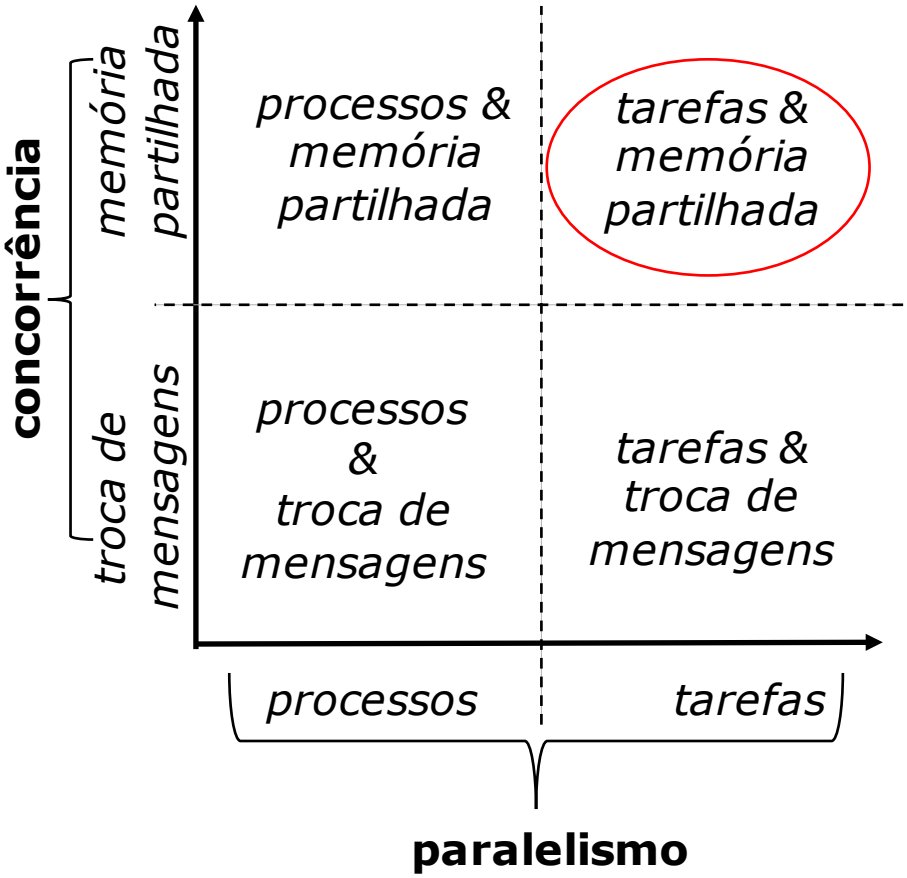
Programação concorrente multi-tarefa por memória partilhada

Sistemas Operativos

2018 / 2019

Combinações de modelos de paralelismo e coordenação

- Dois modelos de paralelismo:
  1. por tarefa
  2. por processo
- Dois modelos de concorrência:
  1. por troca de mensagens
  2. por memória partilhada
- Os modelos de paralelismo e concorrência podem ser combinados!
  - resultado: 4 alternativas

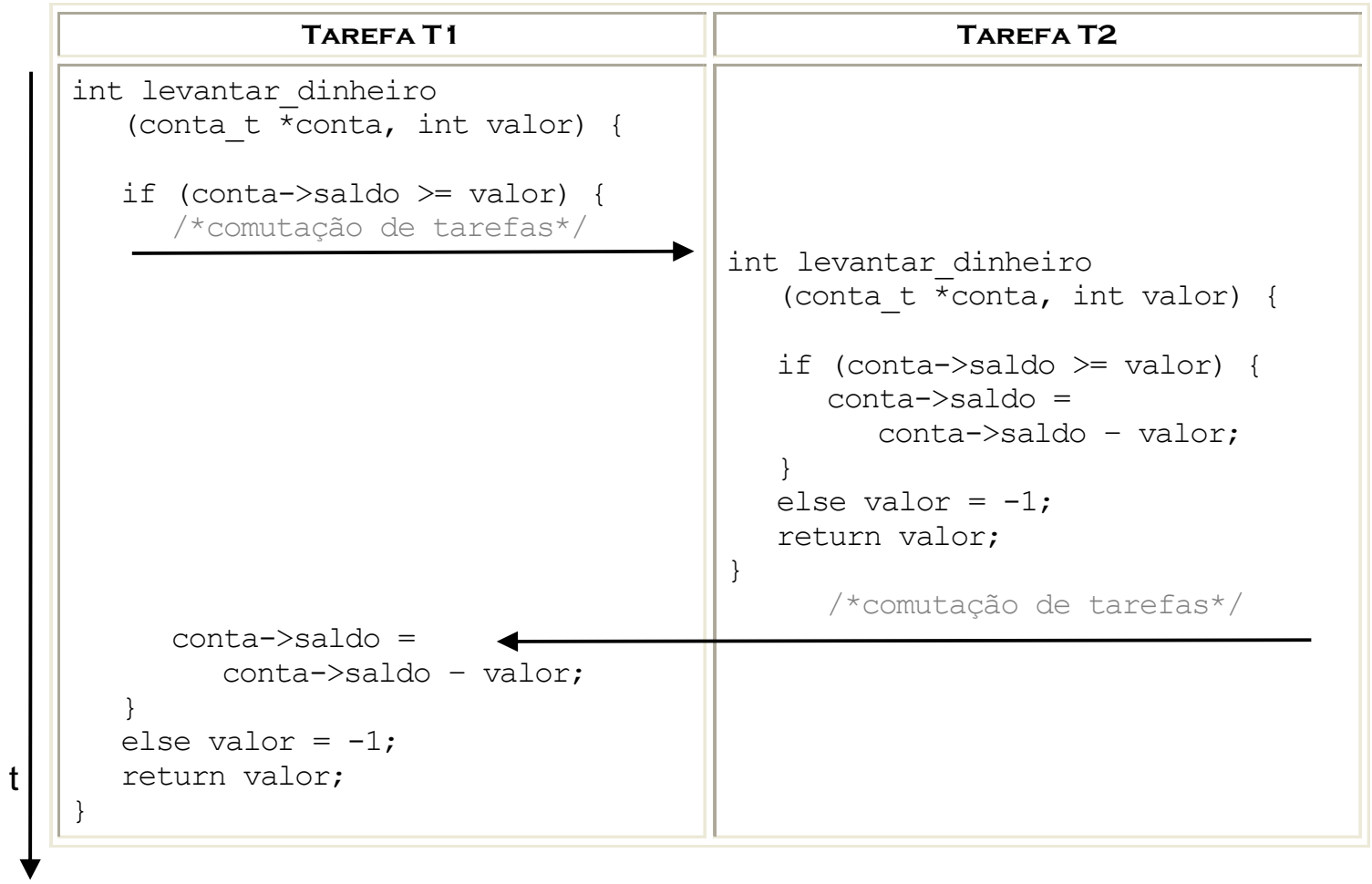


Execução Concorrente

```
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;
   return valor;
}
```

Problema se for multi-tarefa?

Problema 1


TAREFA T1
TAREFA T2
int levantar_dinheiro
(conta_t *conta, int valor) {
if (conta->saldo >= valor) {
/*comutação de tarefas*/
int levantar_dinheiro
(conta_t *conta, int valor) {
if (conta->saldo >= valor) {
conta->saldo =
conta->saldo - valor;
}
else valor = -1;
return valor;
}
/*comutação de tarefas*/
conta->saldo =
conta->saldo - valor;
}
else valor = -1;
return valor;
}
t

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

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 nesta secção crítica?

```
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;
}
```

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!

Mesmo exemplo com trincos

```
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 trincos (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:

```
pthread_mutex_t count_lock;

pthread_mutex_init(&count_lock, NULL);
pthread_mutex_lock(&count_lock);
count++;
pthread_mutex_unlock(&count_lock);
```

Desafio de hoje:

Como implementar um trinco lógico?

Propriedades desejáveis num trinco

- Propriedade de correção (*safety)*
  - Exclusão mútua
    - no máximo uma tarefa detém o trinco
- Propriedades de progresso (*liveness*)
  - Ausência de interblocagem (deadlock)
    - Se pelo menos uma tarefa tenta obter o trinco, então alguma o obterá (dentro de um tempo finito)
  - Ausência de míngua (starvation)
    - Se uma dada tarefa tenta obter o trinco, essa tarefa conseguirá obtê-lo (dentro de um tempo finito)
  - Eficiência

Secção Crítica: Implementações

- Algorítmicas
- Hardware
- Sistema Operativo

Como implementar um trinco?

Primeira tentativa: soluções algorítmicas

Tentativa de Solução #1

```
int trinco = ABERTO;

Fechar () {
   while (trinco == FECHADO)   {/* instrução vazia */};
   trinco = FECHADO;
}

Abrir () {
   trinco = ABERTO;
}
```

Tentativa de Solução #2

```
int trinco_t1 = ABERTO;
int trinco_t2 = ABERTO;
```

Tarefa T1

```
t1_fechar () {
   while (trinco_t2 == FECHADO);
   trinco_t1 = FECHADO;
}

t1_abrir() {trinco_t1 = ABERTO;}
```

Tarefa T2

```
t2_fechar ( ) {
   while (trinco_t1 == FECHADO);
   trinco_t2 = FECHADO;
}

t2_abrir() {trinco_t2 = ABERTO;}
```

Tentativa de Solução #3
(igual à #2 mas com linhas trocadas)

```
int trinco_t1 = ABERTO;
int trinco_t2 = ABERTO;
```

Tarefa T1

```
t1_fechar () {
   trinco_t1 = FECHADO;
   while (trinco_t2 == FECHADO);
}

t1_abrir() {trinco_t1 = ABERTO;}
```

Tarefa T2

```
t2_fechar ( ) {
   trinco_t2 = FECHADO;
   while (trinco_t1 == FECHADO);
  }

t2_abrir() {trinco_t2 = ABERTO;}
```

Tentativa de Solução #4

```
int trinco_vez = 1;
```

Tarefa T1

```
t1_fechar () {
   while (trinco_vez == 2);
}

t1_abrir () {trinco_vez = 2;}
```

Tarefa T2

```
t2_fechar () {
   while (trinco_vez == 1);
}

t2_abrir () {trinco_vez = 1;}
```

Ainda não conseguimos cumprir as propriedades todas...

Algoritmo da Padaria
Versão intuitiva

- Cada cliente tem:
  - Senha com inteiro
    - Com número positivo caso esteja à espera da sua vez (ou a ser atendido)
    - Com zero caso contrário
  - Caneta
    - Sem tampa (caso o cliente esteja a escrever na sua senha)
    - Com tampa (caso o cliente não esteja a escrever na sua senha)
- Qualquer cliente pode observar os elementos acima dos outros clientes, mas só observa um de cada vez

Algoritmo da Padaria
Versão intuitiva

- Quando um cliente quer ser atendido:
  - **Fase 1 (obtenho número para a minha senha)**
    - Tiro tampa da minha caneta
    - Olho para as outras senhas, 1 por 1, para determinar máximo
    - Escrevo na minha senha: máximo+1
    - Coloco tampa na minha caneta
  - **Fase 2 (espero até ser sua vez de ser servido)**
    - Olho para a senha de cada cliente, 1 por 1
    - Para cada outro cliente com senha positiva, espero enquanto:
      - Outro cliente tem tampa fora da caneta
      - Senha do outro tem número inferior à minha
      - Em caso de empate, caso o id do outro cliente seja inferior ao meu
  - **Fase 3** (posso ser atendido em exclusão mútua!)
  - **Fase 4:** coloca senha a 0 (já fui atendido)

Algoritmo de Lamport (Bakery)

```
int senha[N];   // Inicializado a 0
int escolha[N];   // Inicializado a FALSE

Fechar (int i) {
  int j;
  escolha[i] = TRUE;
  senha [i] = 1 + maxn(senha);
  escolha[i] = FALSE;

  for (j=0; j<N; j++) {
    if (j==i) continue;
    while (escolha[j]) ;
    while (senha [j] && (senha [j] < senha [i]) ||
           (senha [i] == senha [j] && j < i)));
  }
}

Abrir (int i) {senha [i] = 0;}
```

- senha contém o número da senha atribuído à tarefa
- escolha indica se a tarefa está a pretender aceder à secção crítica

- Pi indica que está a escolher a senha
- Escolhe uma senha maior que todas as outras
- Anuncia que escolheu já a senha

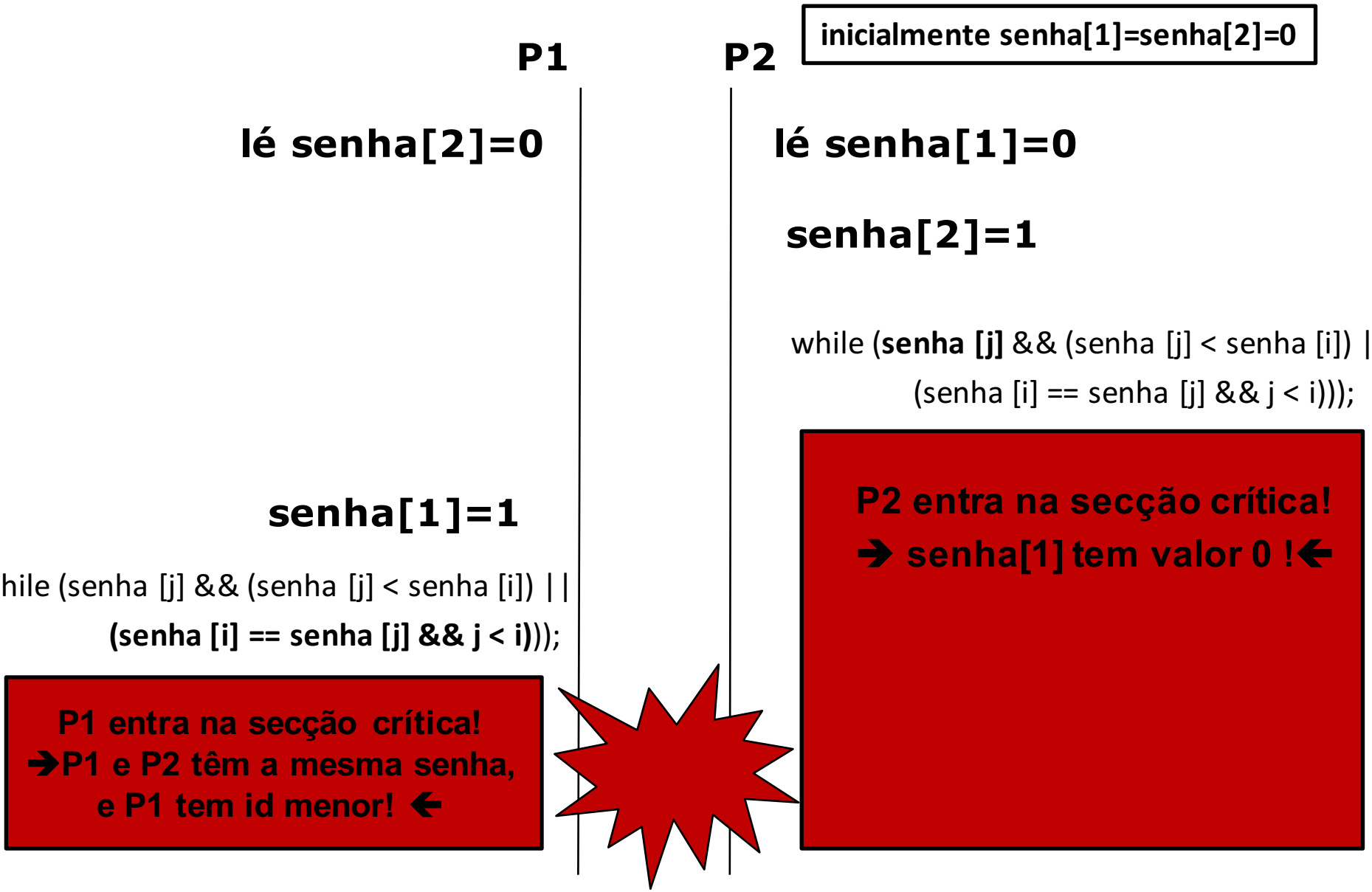
- Pi verifica se tem a menor senha de todos os Pj

- Se Pj estiver a escolher uma senha, espera que termine

- Neste ponto, Pj ou já escolheu uma senha, ou ainda não escolheu
  - Se escolheu, Pi vai ver se é menor que a sua
  - Se não escolheu, vai ver a de Pi e escolher uma senha maior

- Se a senha de Pi for menor, Pi entra
- Se as senhas forem iguais, entra o que tiver o menor identificador

E se não usássemos *escolha*



Soluções Algorítmicas

- Conclusão:
  - Complexas → Latência
  - Só são corretas se não houver reordenação de acessos memória
    - Implica perder otimizações de desempenho que são possíveis por compiladores modernos e caches
  - Só contemplam espera ativa

Como implementar um trinco?

Segunda tentativa: soluções com suporte do hardware

Soluções com Suporte do Hardware

- Abrir( ) e Fechar( ) usam instruções especiais oferecidas pelos processadores:
  - Inibição de interrupções:
    - só iremos estudar mais à frente!
  - Exchange (xchg no Intel)
  - Test-and-set (cmpxchg no Intel)

Aproveitar instruções hw atómicas
Exemplo: Test-and-set

```
BTS varX, 0
```

- De forma indivisível (*):
  - Lê o bit menos significativo de varX
  - Escreve o valor do bit na *carry flag*
  - Coloca esse bit de varX com valor 1

(*) tranca o bus de memória, logo também funciona em multi-processador

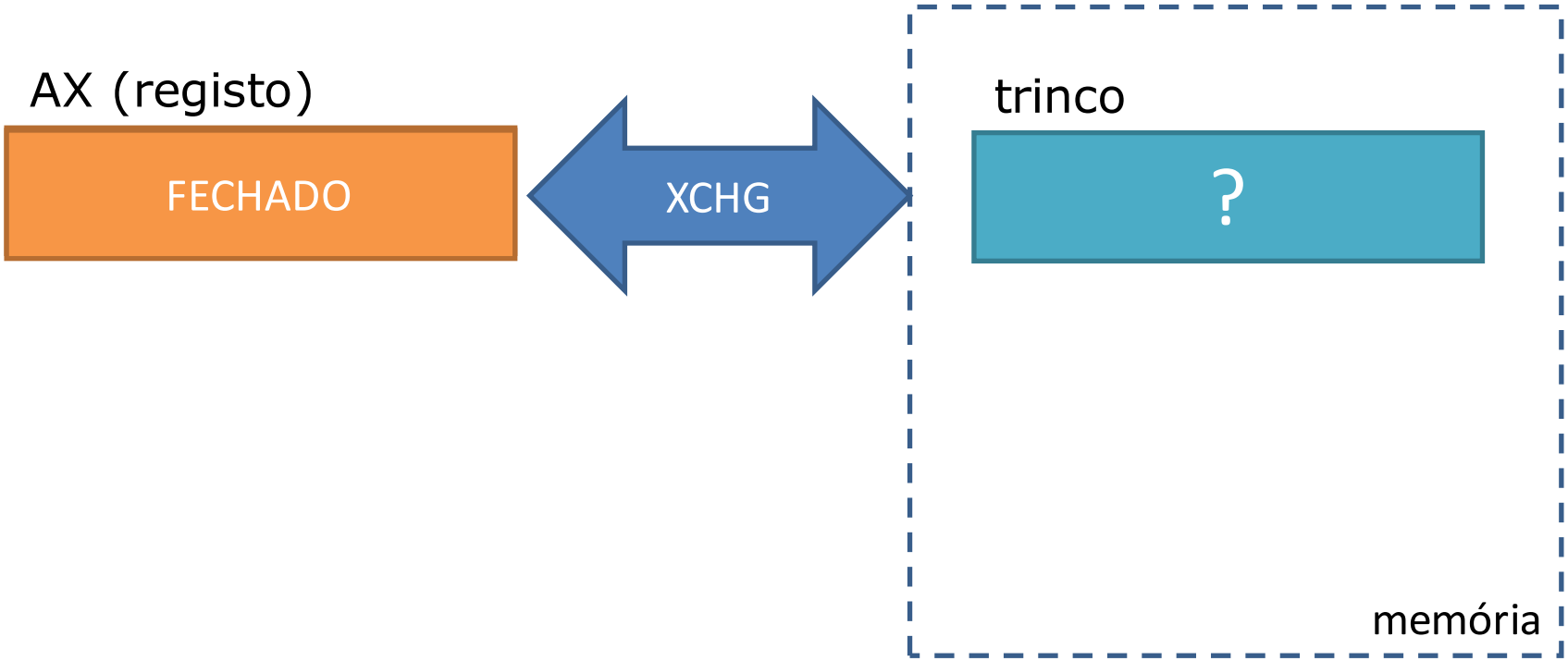
Aproveitar instruções hw atómicas
Exemplo: Test-and-set

```
ABERTO  EQU 0        ; ABERTO equivale ao valor 0
FECHADO EQU 1        ; FECHADO equivale ao valor 1

Fechar_hard:
L1:         MOV AX, 0
BTS trinco, AX
JC L1                ; a variável trinco fica fechada (valor 1)
                     ; a carry flag fica com o valor inicial do trinco
                     ; se carry flag ficou a 1, trinco estava FECHADO
                     ; implica voltar a L1 e tentar de novo
                     ; se carry flag ficou a 0, trinco estava ABERTO
RET                  ; trinco fica a 1 (FECHADO)

Abrir_hard:
MOV         AX, ABERTO
MOV         trinco, AX
RET
```

Aproveitar instruções hw atómicas
Exemplo: XCHG

- XCHG [registo], [localização em memória]
- Troca um valor com o outro
- De forma indivisível
  - Bus de memória é trancado durante o XCHG

Trinco com XCHG



Trinco com XCHG
Situação 1

AX (registo)

FECHADO

Entrei em exclusão mútua?

Não! Já estava trancado por outra tarefa.
Tento de novo...

Trinco com XCHG
Situação 2

AX (registo)

ABERTO

Entrei em exclusão mútua?

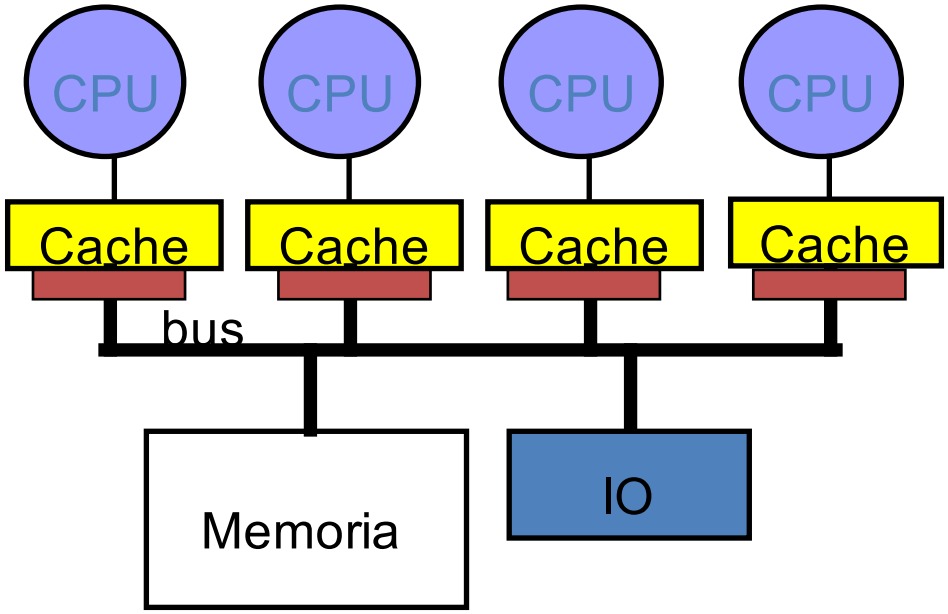
Sim!

Trinco com XCHG
Agora em pseudo-assembly

```
trinco = ABERTO;

Fechar_hardware:
MOV  AX, FECHADO
L1:   XCHG AX, trinco
CMP  AX, ABERTO
JNZ  L1
RET

Abrir_hard:
MOV AX, ABERTO
MOV trinco, AX
RET
```

XCHG em Multiprocessadores

	P1	P2
Instante 1	P1 inicia exchange e tranca o bus	
Instante 2	P1 completa exchange e tranca a secção crítica	P2 tenta fazer exchange mas bloqueia-se a tentar obter o bus
Instante 3	P1 entra secção crítica	P2 verifica que o trinco está trancado e fica em espera activa



Soluções com Suporte do Hardware

- Conclusão:
  - Oferecem os mecanismos básicos para a implementação da exclusão mútua, mas...
  - Algumas não podem ser usadas directamente por programas em modo utilizador
    - e.g., inibição de interrupções
  - Outras só contemplam espera activa
    - e.g., exchange, test-and-set

Como implementar um trinco?

Terceira tentativa:
Trincos como objetos geridos pelo núcleo do Sistema Operativo

VEREMOS MAIS À FRENTE NO SEMESTRE

Como implementar um trinco?

Terceira tentativa:
Trincos como objetos geridos pelo núcleo do Sistema Operativo

VEREMOS MAIS À FRENTE NO SEMESTRE