



Cooperação entre Actividades Variáveis de condição

Sistemas Operativos





Sincronização baseada em invariantes

- **Ideia base:** para cada estrutura de dados partilhada, definir os invariantes lógicos que devem ser sempre respeitados
- **Invariante:** expressão lógica construída com base em variáveis de estado partilhadas
- Se uma tarefa tentar executar uma operação que torna o invariante falso deve ser bloqueada
- Quando o invariante for verdadeiro e houver tarefas bloqueadas, uma tarefa deve retomar a execução



Exemplos de invariantes: produtor-consumidor

- $NumeroMensagens \geq 0$  Se existem 0 mensagens e uma tarefa executar `LeMsg` tem de ser bloqueada de outra forma a condição ficaria falsa ($NumeroMensagens = -1$)
- $NumeroMensagens < NMAX$  Se os produtores executarem `EscreveMsg` e o numero de mensagem for $NMAX-1$ têm de ser bloqueados

A conjugação das duas condições (“E” lógico) define toda a sincronização necessária



Condições de sincronização

- Apesar de não termos pensado em termos de invariantes, programámos com base em condições lógicas para controlar os semáforos
- Estas condições são uma forma de explicitar as condições de sincronização que devem ser sempre validas para que a estrutura de dados partilhada tenha um funcionamento correcto



Leitores Escritores: programação

- Duas classes de tarefas:
 - Leitores: apenas lêem a estrutura de dados
 - Escritores: modificam a estrutura de dados
- Condições de sincronização
 - Escritores: só podem aceder em **exclusão mútua**. Não pode haver leitores porque poderiam ler valores inconsistentes, ou outros escritores porque estes poderiam modificar valores testados ou usados na computação pelo escritor
 - Leitores: podem aceder **simultaneamente com outros leitores, mas em exclusão mútua com os escritores**



Variável de Condição

- Permite a uma tarefa esperar por uma condição que depende da ação de outra tarefa
 - Condição é **booleano determinado em função do estado de variáveis partilhadas**



Variável de Condição

- Variável de condição **sempre associada a um trinco**
 - O trinco que protege as secções críticas com acessos às variáveis partilhadas que definem a condição da espera
 - Pode haver mais que uma variável de condição associada ao mesmo trinco
- O conjunto trinco + variáveis de condição é normalmente chamado um *monitor*



Variáveis de Condição - POSIX

- `pthread_cond_t`
- Criação/destruição de variáveis de condição;
 - `pthread_cond_init (condition,attr)`
 - `pthread_cond_destroy (condition)`
- Assinalar e esperar nas variáveis de condição:
 - `pthread_cond_wait (condition,mutex)`
 - `pthread_cond_signal (condition)`
 - `pthread_cond_broadcast (condition)`



Variável de Condição: primitivas (semântica Mesa)

- *wait(conditionVar, mutex)*
 - **Atomicamente**, liberta o trinco associado e bloqueia a tarefa
 - Tarefa é colocada na fila de espera associada à variável de condição
 - Quando for desbloqueada, **a tarefa re-adquire o trinco** e só depois é que a função *esperar* retorna

Uma tarefa só pode chamar wait quando detenha o trinco associado à variável de condição



Variável de Condição: primitivas (semântica Mesa)

- *signal(conditionVar)*
 - Se houver tarefas na fila da variável de condição, desbloqueia uma
 - Tarefa que estava bloqueada passa a executável
 - Se não houver tarefas na fila da variável de condição, **não tem efeito**
- *broadcast(conditionVar)*
 - Análogo ao signal mas desbloqueia todas as tarefas na fila da variável de condição

Normalmente estas primitivas são chamadas quando a tarefa ainda não libertou o trinco associado à variável de condição



Exemplo: acesso a parque de estacionamento

```
int vagas = N;
```

```
void entrar() {  
    if (vagas==0)  
        esperar até haver vaga  
    vagas --;  
}
```

```
void sair() {  
    vagas ++;  
}
```



Padrões habituais de programação com variável de condição

```
lock(trinco);  
/* ..acesso a variáveis partilhadas.. */  
while (! condiçãoSobreEstadoPartilhado)  
    wait(varCondicao, trinco);  
/* ..acesso a variáveis partilhadas.. */  
unlock(trinco);
```

Código
que espera
por condição

```
lock(trinco);  
/* ..acesso a variáveis partilhadas.. */  
  
/* se o estado foi modificado de uma forma  
que pode permitir progresso a outras tarefas,  
chama signal (ou broadcast) */  
signal/broadcast(varCondicao);  
  
unlock(trinco);
```

Código
que muda
ativa
condição



Exemplo: acesso a parque de estacionamento

```
int vagas = N; mutex m; cond c;
```

```
void entrar() {  
    lock (m) ;  
    while (vagas == 0)  
        wait(c, m) ;  
    vagas --;  
    unlock (m) ;  
}
```

```
void sair() {  
    lock (m) ;  
    vagas ++;  
    signal (c) ;  
    unlock (m) ;  
}
```



Exercício: canal de comunicação (a.k.a. problema do produtor-consumidor)

```
int buffer[N];
int prodptr=0, consptr=0, count=0;

enviar(int item) {
    if (count == N)
        return -1;
    buffer[prodptr] = item;
    prodptr++;
    if (prodptr == N) prodptr=0;
    count++;
    return 1;
}

int receber(){
    int item;
    if (count == 0)
        return -1;

    item = buffer[consptr];
    consptr++;
    if (consptr ==N) consptr = 0;
    count--;
    return item;
}
```

**Problemas caso
enviar/receber sejam
chamadas
concorrentemente?**

**Como estender para
suportar envio/recepção
síncrona?**



Exercício: canal de comunicação (a.k.a. problema do produtor-consumidor)

```
int buf[N], prodptr=0, consptr=0, count=0;  
pthread_mutex_t mutex;
```

```
1. enviar(int item) {  
2.     pthread_mutex_lock(&mutex);  
3.     //AQUI QUERO:  
4.     //Esperar enquanto buffer cheio  
5.     buffer[prodptr] = item;  
6.     prodptr++;  
7.     if (prodptr == N) prodptr=0;  
8.     count++;  
9.     pthread_mutex_unlock(&mutex);  
10.    return 1;  
11.}  
  
12.int receber(){  
13.    int item;  
14.    pthread_mutex_lock(&mutex);  
15.    //AQUI QUERO:  
16.    //Esperar enquanto buffer vazio  
17.    item = buffer[consptr];  
18.    consptr++;  
19.    if (consptr == N) consptr = 0;  
20.    count--;  
21.    pthread_mutex_unlock(&mutex);  
22.    return item;  
23.}
```



Produtor – Consumidor com Variáveis Condição

```
int buf[N], prodptr=0, consptr=0, count=0;
```

```
pthread_mutex_t mutex;  
pthread_cond_t podedProd, podedCons;
```

```
produtor() {  
    while(TRUE) {  
        int item = produz();  
        pthread_mutex_lock(&mutex);  
        while (count == N) pthread_cond_wait(&podedProd, &mutex);  
        buf[prodptr] = item;  
        prodptr++; if(prodptr==N) prodptr = 0;  
        count++;  
        pthread_cond_signal(&podedCons);  
        pthread_mutex_unlock(&mutex);  
    }  
}
```

```
consumidor() {  
    while(TRUE) {  
        int item;  
        pthread_mutex_lock(&mutex);  
        while (count == 0) pthread_cond_wait(&podedCons, &mutex);  
        item = buf[conspr];  
        conspr++; if (conspr == N) conspr = 0;  
        count--;  
        pthread_cond_signal(&podedProd);  
        pthread_mutex_unlock(&mutex);  
    }  
}
```



Variável de Condição: discussão (I)

- Variável de condição *não tem memória*
 - *Signal/broadcast* sobre variável de condição com fila vazia não fica registrado para *waits* posteriores
 - Consequência: *wait* sempre precedido por verificação da condição (booleano)

Diferença em relação a semáforos?



Variável de Condição: primitivas (semântica Mesa)

- `wait(conditionVar, mutex)`
 - **Atomicamente**, liberta o trinco associado e bloqueia a tarefa
 - Tarefa é colocada na fila de espera associada à variável de condição
 - Quando for desbloqueada, a tarefa **re-adquire o trinco** e só depois é que a função espera

Uma tarefa só pode chamar
trinco associado à variável



Padrões habituais de programação com variável de condição

```
lock(trinco);  
/* ..acesso a variáveis partilhadas.. */  
while (! condiçãoSobreEstadoPartilhado)  
    wait(varCondicao, trinco);  
/* ..acesso a variáveis partilhadas.. */  
unlock(trinco);
```

Código
que espera
por condição

```
lock(trinco);  
/* ..acesso a variáveis partilhadas.. */  
  
/* se o estado foi modificado de uma forma  
que pode permitir progresso a outras tarefas,  
chama signal (ou broadcast) */  
signal/broadcast(varCondicao);  
  
unlock(trinco);
```

Código
que muda
ativa
condição



Exemplo: acesso a parque de estacionamento

```
int vagas = N; mutex m; cond c;
```

```
void entrar() {  
    lock (m) ;  
    while (vagas == 0)  
        wait(c, m) ;  
    vagas --;  
    unlock (m) ;  
}
```

```
void sair() {  
    lock (m) ;  
    vagas ++;  
    signal (c) ;  
    unlock (m) ;  
}
```



Variável de Condição: discussão (I)

- Tarefa que chama wait liberta o trinco e entra na fila de espera **atomicamente**
 - Consequência: caso a condição mude e haja *signal*, pelo menos uma tarefa na fila será desbloqueada

O que aconteceria se não houvesse a garantia?



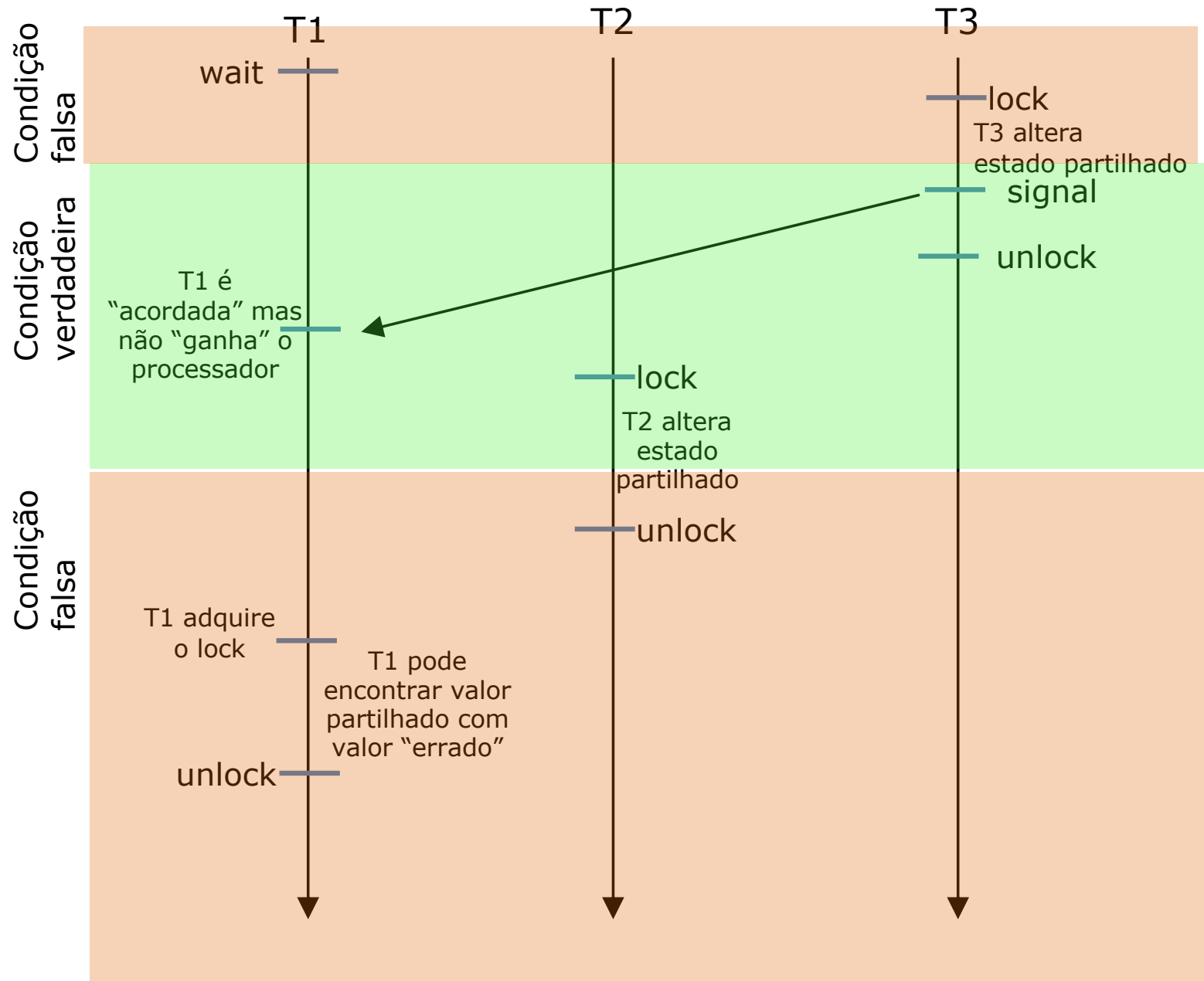
Variável de Condição: discussão(II)

- Tarefa em espera que seja desbloqueada por signal/broadcast não corre imediatamente
 - Simplesmente é tornada executável
 - Para que *wait* retorne, tem de re-adquirir o trinco

Consequência?

Variável de Condição: discussão(II)

- durante o tempo que medeia entre o signal (feito por T3) e uma tarefa ser “acordada” (T1) adquirindo o trinco
- a variável de condição pode ser alterada por outra tarefa (T2) !!!





Variável de Condição: discussão(II)

- Retorno do wait não garante que condição que lhe deu origem se verifique
 - Tarefa pode não ter sido a primeira tarefa a entrar na secção crítica depois da tarefa que assinalou a ter libertado
- Logo, após retorno do wait, re-verificar a condição:
 - Não fazer: **if** (testa variável partilhada) wait
 - Fazer: **while** (testa variável partilhada) wait



Variável de Condição: discussão(II)

- Algumas implementações de variáveis de condição permitem que tarefa retorne do *wait* sem ter ocorrido *signal/broadcast*
 - “Spurious wakeups”
- Mais uma razão para testar condição com *while* em vez de *if*



Comparação com os semáforos

Variáveis de Condição

- Não tem qualquer variável de controlo nem estado (aberto/fechado) como os *mutexes*
- Wait - Bloqueia sempre que é invocado
- Signal – desbloqueia se há tarefas bloqueadas senão não tem efeito
- Broadcast – desbloqueia todas as tarefas

Semáforos

- Tem um contador
- Esperar – bloqueia se contador == 0
- Assinalar – desbloqueia um tarefa ou incrementa o contador
- Não tem (seria útil nos leitores/escritores para acordar todos os leitores)