

Computação Concorrente (DCC/UFRJ)

Aula 6: Sincronização condicional com variáveis de condição

Prof. Silvana Rossetto

24 de setembro de 2019

Sincronização por condição



Visa garantir que **uma thread fique bloqueada enquanto uma determinada condição lógica da aplicação não for satisfeita**

A solução para a sincronização por condição é definida **impedindo a continuação da execução de uma thread até que o estado da aplicação seja correto para a sua execução**

Exemplo: problema produtor/consumidor



- Os **produtores** **produzem/geram** novos elementos
- Os **consumidores** **consomem/processam** esses elementos

Uma **área de dados** é compartilhada entre **produtores e consumidores**

- Produtores **depositam** os elementos gerados na área de dados
- Consumidores **retiram** os elementos que devem ser processados da área de dados

O problema dos produtores e consumidores

Condições do problema

- 1 Os produtores não podem inserir novos elementos quando a área de dados já está cheia
- 2 Os consumidores não podem retirar elementos quando a área de dados já está vazia
- 3 Os elementos devem ser retirados na mesma ordem em que foram inseridos
- 4 Os elementos inseridos não podem ser perdidos (sobreescritos por novos elementos)
- 5 Um elemento só pode ser retirado uma única vez

Variáveis de condição

Definição

Variáveis especiais que permitem que as threads esperem (bloqueando-se) até que sejam sinalizadas (avisadas) por outra thread que a condição lógica foi atendida

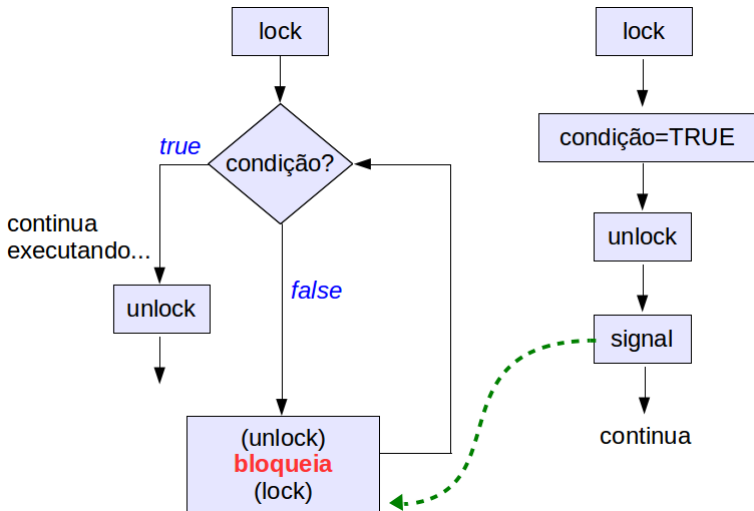
Operações básicas

- **WAIT(condvar)**: bloqueia a thread na fila da variável de condição
- **SIGNAL(condvar)**: desbloqueia uma thread na fila da variável de condição
- **BROADCAST(condvar)**: desbloqueia todas as threads na fila da variável de condição

Como funcionam as variáveis de condição

- Uma **variável de condição** é sempre usada em conjunto com uma **variável de lock**
- A thread usa o **bloco de lock para checar a condição lógica da aplicação e decidir** por WAIT ou SIGNAL
- O lock é **implicitamente liberado** quando a thread é bloqueada e é **implicitamente devolvido** quando a thread retoma a execução do ponto de bloqueio

Como funcionam as variáveis de condição



Locks versus Variáveis de condição

- **Locks** são usados para implementar **sincronização por exclusão mútua** (**controle do acesso ao dado**)
- **Variáveis de condição** são usadas para implementar a **sincronização por condição** (**lógica de execução baseado no valor do dado**)

Variáveis de condição em PThreads

A biblioteca PThreads define um tipo especial chamado **pthread_cond_t** com as seguintes rotinas:

- **pthread_cond_wait (condvar, mutex)**: bloqueia a thread na condição (*condvar*) (deve ser chamada com *mutex* locado para a thread e depois de finalizado deve desalocar *mutex*)
- **pthread_cond_signal (condvar)**: desbloqueia uma thread esperando pela condição (*condvar*)
- **pthread_cond_broadcast (condvar)**: usado no lugar de SIGNAL quando todas as threads na fila da condição podem ser desbloqueadas

Variáveis de condição em PThreads

- **pthread_cond_init (condvar, attr):** inicializa a variável
- **pthread_cond_destroy (condvar):** libera a variável

Implementar uma solução para o problema dos **produtores/consumidores**

- 1 apenas um produtor e um consumidor
- 2 mais de um produtor e mais de um consumidor

Produtor

```
void * produtor(void * arg) {  
    tElemento elemento;  
    while(REPETE) {  
        elemento = produzElemento();  
        Insere(elemento); //pode bloquear!  
    }  
    pthread_exit(NULL);  
}
```

```
void * consumidor(void * arg) {  
    tElemento elemento;  
    while(REPETE) {  
        elemento = Retira(); //pode bloquear!  
        processaElemento(elemento);  
    }  
    pthread_exit(NULL);  
}
```

Funções para inserir e retirar elemento

```
void Insere (tElemento elem) {  
    ...  
}
```

```
tElemento Retira (void) {  
    ...  
}
```

Variáveis globais

```
#define N 5

//variaveis do problema
int Buffer[N];
int count=0, in=0, out=0;

//variaveis para sincronizacao
pthread_mutex_t mutex;
pthread_cond_t cond_cons, cond_prod;
```

Função para inserir elemento

```
void Insere (tElemento elem) {  
    pthread_mutex_lock(&mutex);  
    while(count == N) {  
        pthread_cond_wait(&cond_prod, &mutex);  
    }  
    count++;  
    Buffer[in] = elem;  
    in = (in + 1) % N;  
    pthread_mutex_unlock(&mutex);  
    pthread_cond_signal(&cond_cons);  
}
```


Função para retirar elemento

```
tElemento Retira (void) {  
    tElemento elem;  
    pthread_mutex_lock(&mutex);  
    while(count == 0) {  
        pthread_cond_wait(&cond_cons, &mutex);  
    }  
    count--;  
    elem = Buffer[out];  
    out = (out + 1) % N;  
    pthread_mutex_unlock(&mutex);  
    pthread_cond_signal(&cond_prod);  
    return elem;  
}
```

...mostrar execução do problema

Projetar e implementar um programa concorrente com 4 threads onde a ordem de execução das threads é controlada no programa:

- a thread 1 imprime a frase “tudo bem?”
- a thread 2 imprime a frase “hola!”
- a thread 3 imprime a frase “até mais tarde.”
- a thread 4 imprime a frase “tchau!”

As threads 1 e 2 devem executar antes das threads 3 e 4 sempre (a ordem de execução entre as threads 1 e 2 não importa, assim como a ordem de execução entre as threads 3 e 4).

- *Computer Systems - A Programmer's Perspective* (Cap. 12)
- *Programming Language Pragmatics*, M.L.Scott, Morgan-Kaufmann, ed. 2, 2006
- *Modern Multithreading*, Carver e Tai, Wiley, 2006
- *An Introduction to Parallel Programming*, Peter Pacheco, Morgan Kaufmann, 2011