# Utilizando um semáforo para sincronizar threads

Fausto Falcone



**Módulos**

- Hello World
- Porta Paralela s/ Interrupção
- Porta Paralela c/ Interrupção
- Character Device
- Block Device
- Referência

**Diversos**

- Semáforo x Threads
- Dump hexa de arquivos

## Descrição

Este exemplo demonstra a utilização de um semáforo para sincronizar a inicialização de threads que devem receber parâmetros.

O problema consiste em iniciar uma thread passando parâmetros para a mesma e continuar o processamento, porém se o dado passado for local para a função que iniciou a nova thread, pode ser que quando a mesma for ler os dados, os mesmos já tenham se alterado. Para evitar isso, um dos métodos usados é fazer a thread chamadora aguardar que a nova thread leia e copie os dados em variáveis locais, e uma das forma de fazer este sincronismo é utilizando semáforos.

## Construção:

Abaixo segue todo o código para o aplicativo.

thread.cpp

```
1 #include <iostream>
2 #include <pthread.h>
3 #include <semaphore.h>
4 #include <time.h>
5
6 #define  NUMTHREADS 20
7
8 using namespace std;
9
10 struct data
11 {
12     int      n;
13     sem_t*   s;
14 };
15
16 void* th(void* pVoid)
17 {
18     data*    dt  = (data*)pVoid;
19     int      n   = dt->n;
20     sem_post(dt->s);
21
22     sleep( (rand() % 10) + 1);
23     cout<<"thread - "
24         <<n
25         <<" terminada..."
26         <<endl;
27 }
28
29 int main(int agrc, char* argv[])
30 {
31     pthread_t   ht[NUMTHREADS];
32     data        dt;
33     sem_t       s;
34
35     srand(-time(NULL));
36     sem_init(&s, 0, 0);
37
38     for(int n=0;n<NUMTHREADS;++n)
39     {
40         dt.s = &s;
```

```
41          dt.n = n;
42
44          pthread_create(&ht[n], NULL, th, &dt);
45          sem_wait(&s);
47      }
48
49      cout<<"Aguardando threads..."
50          <<endl;
51
52      for(int n=0;n<NUMTHREADS;++n)
53          pthread_join(ht[n], NULL);
54
55      sem_close(&s);
56      return 0;
57  }
```

Após alguns includes, é definido o número de threads que o aplicativo irá criar durante o teste. Na sequência, uma estrutura é criada, a mesma será utilizada para passar dados da thread principal para todas as outras que serão criadas, no momento basta saber que a mesma contém o número da thread (n) e o semáforo (que na verdade poderia ser uma variavel global).

A função th será chamada tantas vezes quantas forem MAXTHREADS, o que será diferente em cada chamada será o parâmetro passado.

Recebido o parâmetro, a função deve salvar o conteúdo n e o ponteiro para o semafóro que serão usados no processamento, para isso converte o ponteiro void para o tipo data e então guarda n. Imediatamente libera a "main thread" incrementando o semáforo com a chamada a sem_post.

Uma vez liberada a thread principal, a thread entra em seu processamento, aqui representado por um sleep randômico e no final imprime no console o conteúdo de n demonstrando que seu valor foi salvo corretamente no início do processamento da thread.

A thread principal inicia o gerador de números aleatórios e logo em seguida inicia o semáforo, passando como valor inicial 0, ou seja, nenhuma aquisição pode ser feita ao mesmo, entrando em wait mesmo na primeira tentativa.

Dentro do for, o ponteiro para o semáforo é preenchido na estrutura data, bem como o número da thread que neste ponto esta em n. Usando a função pthread_create a nova thread é criada. A próxima instrução é a tentativa de adquirir o semáforo, porém seu valor esta em 0 e a thread entra em wait até que sem_post (que leva o valor para 1) seja executado dentro da nova thread, voltando a zero imediatamente após a saída do wait.

Este processo garante que n seja mantido para que a nova thread tenha tempo de copiar seu conteúdo.

Depois que todas as threads foram disparadas, é necessário esperar que cada uma delas encerre seu processamento antes de finalizar a thread principal, para isso este exemplo utiliza pthread_join para cada uma das threads criadas. Por último o semáforo é fechado.

**Compilando:**

Para compilar pode ser criado um makefile simples conforme mostrado abaixo, ou simplesmente usando:

```
g++ threads.cpp -o threads -pthread
```

Makefile

```
1 All:
2       g++ threads.cpp -o threads -pthread
3
```

**Testando o Sincronismo:**

Um bom teste para verificar o funcionamento do sincronismo é retirar do código a linha 45 que tenta adquirir o semáforo, abaixo é listado o resultado da execução com e sem a linha 45. Observar que sem a linha, todas as threads receberam o valor de n como 19 (este resultado pode depender de diversos fatores), ou seja, nesta execução a primeira thread criada somente teve chance de salvar o valor depois que todas as outras foram criadas.

Este comportamento pode variar bastante caso a máquina possua mais de um processador.

```
Exemplo com a linha 45

user@computer:/Linux/samples/threads$ ./threads
Aguardando threads...
thread - 2 terminada...
thread - 0 terminada...
thread - 1 terminada...
thread - 6 terminada...
thread - 19 terminada...
thread - 11 terminada...
thread - 13 terminada...
thread - 16 terminada...
thread - 7 terminada...
thread - 8 terminada...
thread - 9 terminada...
thread - 14 terminada...
thread - 3 terminada...
thread - 4 terminada...
thread - 5 terminada...
thread - 10 terminada...
thread - 17 terminada...
thread - 18 terminada...
thread - 12 terminada...
thread - 15 terminada...
```

```
Exemplo sem a linha 45

user@computer:/Linux/samples/threads$ ./threads
Aguardando threads...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
thread - 19 terminada...
```