Laboratório 7
Sincronização condicional com variáveis de condição

Computação Concorrente (MAB-117)
Prof. Silvana Rossetto

[1]DCC/IM/UFRJ — 26 de novembro de 2015

Introdução

O objetivo deste Laboratório é introduzir o uso de soluções de sincronização por condição, usando variáveis de condição oferecidas pela biblioteca Pthreads. Para cada atividade, siga o roteiro proposto e responda às questões colocadas.

Atividade 1

Objetivo: Exemplificar o uso de *variáveis de condição* provido pela biblioteca Pthreads.

Roteiro:

1. Abra o arquivo **hellobye.c** e compreenda como a aplicação funciona (*acompanhe a explanação da professora*).
2. Execute o programa **várias vezes**. O log de execução impresso na tela foi sempre o esperado? A condição lógica da aplicação foi atendida em todas as execuções?
3. Agora altere o número de threads A para 1. O que vai ocorrer na execução? O programa vai terminar? Por que?
4. Altere o número de threads A de volta para 2. O programa voltou a funcionar?
5. **Agora altere o número de threads B para 2 e faça as correções necessárias no código para que a aplicação continue funcionando.**

Atividade 2

Objetivo: Exemplificar o uso de *variáveis de condição* provido pela biblioteca Pthreads.

Roteiro:

1. Abra o arquivo **byehello.c** e compreenda como a aplicação funciona (*acompanhe a explanação da professora*).
2. Execute o programa **várias vezes**. O log de execução impresso na tela foi sempre o esperado? A condição lógica da aplicação foi atendida em todas as execuções?

Atividade 3

Objetivo: Explorar características do funcionamento das operações sobre variáveis de condição em Pthreads.

Roteiro:

1. Abra o arquivo **printX.c** e compreenda como a aplicação funciona (*acompanhe a explanação da professora*).
2. Execute o programa **várias vezes**. O log de execução impresso na tela foi sempre correto? (atendeu à condição lógica da aplicação?)
3. Podemos substituir a linha 51 pela linha 52? Justifique.
4. Comente a linha 51 e descomente a linha 52, e execute novamente a aplicação **várias vezes**. O log de execução impresso na tela foi sempre correto? (atendeu à condição lógica da aplicação?)

Atividade 4

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

Roteiro:

1. Implemente um programa com 4 threads. 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!".
2. 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).

Atividade 5

Objetivo: Implementar uma "barreira" para sincronização das threads de uma aplicação.

Descrição: Implemente um programa concorrente em C com 5 threads. Todas as threads devem executar o mesmo trecho de código mostrado abaixo. A cada passo (iteração), cada thread deve incrementar sua variável contadora, imprimir o seu ID, o valor da sua variável contadora e o passo atual, e só podem continuar depois que todas as threads completarem esse passo.

Roteiro:

1. Implemente a função `barreira()` para forçar todas as threads a esperar ao final de cada iteração.
2. Implemente o programa completo e avalie a sua corretude.

```
#define NTHREADS 5
#define PASSOS  5
//variaveis globais
...

void barreira(int nthreads)
{ ... }
```

```
void *A (void *arg) {
  int tid = *(int*)arg, i;
  int cont = 0, boba1, boba2;

  for (i=0; i < PASSOS; i++) {
    cont++;
    printf("Thread %d: cont=%d, passo=%d\n", tid, cont, i);

    //sincronizacao condicional
    barreira(NTHREADS);

    /* faz alguma coisa inutil pra gastar tempo... */
    boba1=100; boba2=-100; while (boba2 < boba1) boba2++;
  }
  pthread_exit(NULL);
}

int main(int argc, char *argv[])  { ... }
```

A saída do programa deverá ser como mostrado abaixo. Apenas a ordem das threads na coluna 1 pode variar de uma execução pra outra.

```
Thread 0: cont=1, passo=0
Thread 1: cont=1, passo=0
Thread 2: cont=1, passo=0
Thread 3: cont=1, passo=0
Thread 4: cont=1, passo=0
Thread 4: cont=2, passo=1
Thread 0: cont=2, passo=1
Thread 2: cont=2, passo=1
Thread 1: cont=2, passo=1
Thread 3: cont=2, passo=1
Thread 3: cont=3, passo=2
Thread 4: cont=3, passo=2
Thread 0: cont=3, passo=2
Thread 2: cont=3, passo=2
Thread 1: cont=3, passo=2
Thread 1: cont=4, passo=3
Thread 3: cont=4, passo=3
Thread 4: cont=4, passo=3
Thread 0: cont=4, passo=3
Thread 2: cont=4, passo=3
Thread 2: cont=5, passo=4
Thread 1: cont=5, passo=4
Thread 4: cont=5, passo=4
Thread 0: cont=5, passo=4
Thread 3: cont=5, passo=4
```

Empacote e envie seu programa e o arquivo lab7.txt para correção Crie um diretório e o nomeie juntando seu "primeiro" e "último" nome. Copie pra dentro desse diretório o arquivo `lab7.txt` e o código fonte das atividades **4 e 5**. Comprima o diretório (ex., `zip -r JoseSilva.zip JoseSilva/`) e envie o arquivo comprimido para o endereço de email **computacao.concorrente.ufrj@gmail.com** com subject "**CompConc Lab7**".

O email deve ser enviado até amanhã, dia 27/11!