Laboratório 7
Sincronização com semáforos

Programação Concorrente (ICP-361)
Profa. Silvana Rossetto

[1]Instituto de Computação/UFRJ

Introdução

O objetivo deste Laboratório é introduzir e praticar o uso de **semáforos** para implementar exclusão mútua e sincronização condicional em programas concorrentes.

Atividade 1

Objetivo: Introduzir o uso de semáforos na linguagem C e sistema operacional Linux.

Roteiro: Abra o arquivo **semaf-1.c** e siga os passos abaixo:

1. Leia o programa e compreenda como o mecanismo de semáforo é usado em um programa C para implementar exclusão mútua.
2. Com qual valor o semáforo foi inicializado?
3. Execute o programa **várias vezes**. **Os valores impressos foram sempre o valor esperado?**
4. O que vai acontecer se o semáforo for inicializado com 0 sinais? Por que? Depois de responder, faça essa alteração no programa e verifique os resultados.
5. O que vai acontecer se o semáforo for inicializado com 2 sinais? Por que? Depois de responder, faça essa alteração no programa e verifique os resultados.

Atividade 2

Objetivo: Mostrar um exemplo de uso de semáforos para atender requisitos de ordem de execução das threads.

Roteiro: Abra o arquivo **semaf-2.c** e siga os passos abaixo:

1. Leia o programa para entender como ele funciona. *Acompanhe a explanação da professora.*
2. Com quais valores os semáforos devem ser inicializados para atender aos requisitos de ordem colocados? Por que?
3. Execute o programa **várias vezes** e observe os resultados impressos na tela.
4. Agora inicilize os semáforos com 1 sinal. O vai acontecer e por que? Execute o programa várias vezes e avalie os resultados.

Atividade 3

Objetivo: Mostrar uma implementação do padrão **barreira** usando semáforos.

Roteiro: Abra o arquivo **barreira.c** e siga os passos abaixo:

1. Leia o programa para entender como ele funciona.
2. Por que as versões `barreira0` e `barreira1` estão incorretas? *Acompanhe a explanação da professora.*
3. Execute o programa **várias vezes** e observe os resultados impressos na tela. (O programa executa em loop infinito, então sua execução precisará ser interrompida.)

Atividade 4

Objetivo: Mostrar uma implementação do padrão **produtor/consumidor** usando semáforos.

Roteiro: Abra o arquivo **pc.c** e siga os passos abaixo:

1. Leia o programa para entender como ele funciona.
2. Qual é a finalidade das variáveis **in** e **out** e por que foram definidas como variáveis **static**?
3. Execute o programa **várias vezes** e observe os resultados impressos na tela. (O programa executa em loop infinito, então sua execução precisará ser interrompida.)
4. O programa executou corretamente? Como verificar isso?

Atividade 5

Objetivo: Projetar e implementar um programa concorrente em C que usa o padrão produtor/consumidor.

Descrição: Vamos retomar com o problema da primalidade :-)

Implemente um programa concorrente onde UMA thread PRODUTORA gera uma sequência de N **números inteiros** e os deposita no **canal de inteiros** de tamanho M (M pequeno) — um de cada vez — que será compartilhado com e UMA thread CONSUMIDORA. Os valores de N, M devem ser lidos da entrada do programa.

A thread CONSUMIDORAS retiram os números — um de cada vez — e avaliam sua primalidade, usando a seguinte função:

```
int ehPrimo(long long int n) {
   int i;
      if (n<=1) return 0;
      if (n==2) return 1;
      if (n%2==0) return 0;
      for (i=3; i<sqrt(n)+1; i+=2)
         if(n%i==0) return 0;
      return 1;
```

Ao final do programa (depois que os N números foram processados), deverá ser retornado: **a quantidade de números primos encontrados (para avaliar a corretude do programa).**