Laboratório 9
Introdução ao mecanismo de semáforo

Computação Concorrente (MAB-117)
Prof. Silvana Rossetto

[1]DCC/IM/UFRJ — 3 de novembro de 2016

Introdução

O objetivo deste Laboratório é introduzir o uso de semáforos. Para cada atividade, siga o roteiro proposto e responda às questões colocadas.

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.
2. Execute o programa **várias vezes**. Os valores impressos foram sempre o valor esperado?

Atividade 2

Objetivo: Mostrar um exemplo de uso de semáforos para coordenar a 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. Quais são os possíveis valores finais da variável **y**?
2. Execute o programa **várias vezes** e observe os resultados impressos na tela.
3. O valor final da variável **y** variou? Por que?
4. Altere o valor de inicialização dos semáforos de 1 para 0: faça `sem_init(&condt2,0,0)` e `sem_init(&condt3,0,0)`
   Execute o programa **várias vezes** e observe os resultados impressos na tela. O que aconteceu e por que?

Atividade 3

Objetivo: Projetar e implementar um programa concorrente em C 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 “olá, tudo bem?” A thread 2 imprime a frase “hello!” 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 4

Projetar e implementar uma aplicação com o padrão produtor/consumidor usando semáforos:

- As threads produtoras inserem no buffer o seu próprio ID ($ID > 0$).
- As threads consumidoras retiram um elemento no buffer, colocam o valor -1 na posição retirada, e imprimem o número retirado na tela.
- Os elementos devem ser consumidos na mesma ordem em que são inseridos no buffer e nenhum elemento deve ser perdido (sobreescrito) no buffer.
- As threads produtoras devem ser bloqueadas sempre que tentarem inserir um elemento e encontrarem o buffer cheio.
- As threads consumidoras devem ser bloqueadas sempre que tentarem retirar um elemento e encontrarem o buffer vazio.
- As threads executam indefinidamente.

Roteiro:

1. Defina um buffer de **5** elementos.
2. **Imprima o estado do buffer após cada operação de inserção e remoção** (dentro da seção crítica para garantir que o estado do buffer será escrito antes da próxima operação).
3. Execute o programa **várias vezes** e verifique se a solução está correta.
4. Mostre a execução para a professora.

Empacote e envie os programas para correção Crie um diretório e o nomeie juntando seu "primeiro" e "último" nome. Copie pra dentro desse diretório os códigos fonte das atividades **3 e 4**. 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 Lab9**".

O email deve ser enviado até amanhã, dia 4/11.