CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS VII - UNIDADE TIMÓTEO - Engenharia da Computação

Laboratório de Programação Concorrente
Prof. Lucas Pantuza Amorim

Exclusão mútua com bloqueio usando semáforos

O objetivo deste laboratório é introduzir o uso de semáforos. Usaremos a linguagem `C` (com a biblioteca `Pthread`). Para cada atividade, siga o roteiro proposto antes de passar para a próxima atividade.

Atividade 1

Objetivo: Introduzir o uso de semáforos na linguagem `C`.

Roteiro:

1. Abra o arquivo `semaf-1.c`.
2. Leia o programa e compreenda como o mecanismo de semáforo é usado em um programa `C`.
3. Execute o programa várias vezes. Os valores impressos foram sempre o valor esperado?

Relatório da atividade: Apenas observar a execução.

Atividade 2

Objetivo: Mostrar um exemplo de uso de semáforos para coordenar a ordem de execução das *threads*.

Roteiro:

1. Abra o arquivo `semaf-2.c`.
2. Leia o programa para entender como ele funciona. Quais são os possíveis valores finais da variável `y`?
3. Execute o programa várias vezes e observe os resultados impressos na tela.
4. A ordem de execução das *threads* e o valor final da variável `y` variou? Por que?
5. Altere o valor de inicialização dos semáforos de 0 para 1, ou seja:
   - `sem_init(&condt2,0,1)`
   - `sem_init(&condt3,0,1)`
6. Execute o programa várias vezes e observe os resultados impressos na tela. O que aconteceu e por que?

Relatório da atividade: Em um arquivo `.txt` descreva as respostas das questões colocadas e inclua exemplos dos resultados obtidos nas execuções realizadas.

Atividade 3

Objetivo: Projetar e implementar um programa *multithreading* 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).

Relatório da atividade: Em um arquivo `.txt` descreva as respostas das questões colocadas e inclua exemplos dos resultados obtidos nas execuções realizadas.

Atividade 4

Objetivo: Projetar e implementar uma aplicação produtor/consumidor.

Roteiro: Implemente uma aplicação em `C` com duas *threads*: uma que gera e deposita números inteiros em um *buffer* ("produtor") e outra que consome esses elementos ("consumidor"). Dica: use semáforos contadores.

1. Defina um *buffer* de 5 elementos.
2. A *thread* produtora gera 100 elementos usando a função $f(t) = 3t^2 + 7t$, onde t varia de 0 a 100.
3. A *thread* consumidora retira um número e verifica a sua primalidade (veja dica de função abaixo).

```c
//função para determinar se um numero é primo
int ehPrimo(long unsigned 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;
}
```

4. Os elementos devem ser consumidos na mesma ordem em que são inseridos no *buffer* e nenhum elemento deve ser perdido (sobreescrito) no *buffer*.
5. A *thread* produtora deve ser bloqueada sempre que tenta inserir um elemento e encontra o *buffer* cheio.
6. A *thread* consumidora deve ser bloqueada sempre que tenta retirar um elemento e encontra o *buffer* vazio.
7. Execute o programa várias vezes e verifique se a solução está correta.

Relatório da atividade: Em um arquivo `.txt` descreva as respostas das questões colocadas e inclua exemplos dos resultados obtidos nas execuções realizadas.

Atividade 5

Objetivo: Projetar e implementar uma aplicação com o padrão leitores/escritores.

Roteiro: Implemente uma aplicação em `C` com uma variável global compartilhada com dois campos (`struct`): um contador (inteiro) e um identificador de *thread* (inteiro). O conteúdo dessa variável será lido por *threads* leitoras e escrito/alterado por *threads* escritoras. As *threads* escritoras alteram a variável incrementando o valor do contador e armazenando o seu ID (identificador da *thread*) no outro campo da estrutura. As *threads* leitoras lêem o valor da variável (os dois campos) e o armazenam em uma variável local. Depois executam um processamento "bobo" sobre esse valor e voltam a solicitar nova leitura.

Relembrando as condições lógicas do problema leitores/escritores: mais de um leitor pode ler ao mesmo tempo, apenas um escritor pode escrever de cada vez e se um escritor está escrevendo nenhum leitor pode estar lendo.

1. Crie um número N de *threads* leitoras ($N >= 2$) e um número M de *threads* escritoras ($M >= 2$).
2. Insira no seu código a impressão de informações que permitam acompanhar a execução da aplicação para verificar se as condições lógicas do problema são satisfeitas.
3. Execute a aplicação várias vezes e avalie os resultados obtidos.
4. Altere o número de *threads* leitoras e escritoras e reexecute a aplicação.

Relatório da atividade: Em um arquivo `.txt` descreva as respostas das questões colocadas e inclua exemplos dos resultados obtidos nas execuções realizadas.

Atividade 6

Objetivo: Projete e implemente outra solução para o problema dos leitores/escritores (mesma aplicação da atividade anterior), agora com prioridade para escrita: quando um escritor deseja escrever, a entrada de novos leitores na seção crítica é impedida permanecendo assim até que a fila de escritores aguardando para executar escrita se esgote.

Roteiro:

1. Crie um número N de *threads* leitoras ($N >= 2$) e um número M de *threads* escritoras ($M >= 2$).
2. Insira no seu código a impressão de informações que permitam acompanhar a execução da aplicação para verificar se as condições lógicas do problema são satisfeitas.
3. Execute a aplicação várias vezes e avalie os resultados obtidos.
4. Altere o número de *threads* leitoras e escritoras e reexecute a aplicação.

Relatório da atividade: Em um arquivo `.txt` descreva as respostas das questões colocadas e inclua exemplos dos resultados obtidos nas execuções realizadas.