MAC 438 – Programação Concorrente
Prof. Marcel Parolin Jackowski
Departamento de Ciência da Computação
IME/USP – Primeiro Semestre de 2016
Segundo Exercício-Programa
Data de entrega: até 16/05/2016 às 23h55

Sincronização com POSIX threads

Este exercício-programa tem o objetivo de exercitar o uso de *threads* e primitivas de sincronização para implementar um programa concorrente, conforme requisitos descritos no enunciado abaixo.

A Festa Secreta

Anualmente acontece na USP uma festa cujos convites são limitados a n alunos. Esse evento acontece em uma sala inconspícua, previamente definida, em um dos institutos. Quando os convidados chegam à festa, eles mostram o convite, entram e se divertem pela madrugada afora. A sala pode comportar qualquer número de alunos, porém restrito a n convidados. Ao longo da noite, alunos deixam a sala, cansados da festa e vão embora. Existe um único segurança no instituto que faz rondas durante a noite e que pode interromper a festa. O segurança entra na sala quando não houver nenhum aluno na festa, ou se o número de alunos dentro da sala for igual ou maior que p, $p \leq n$ alunos. Se não houver alunos, ele inspeciona a sala e volta para a sua ronda. Mas se houver ao menos p alunos presentes, ele fica bravo e espera todos os alunos da sala irem embora, para depois voltar à ronda. Enquanto o segurança estiver na sala, nenhum aluno novo poderá entrar, mas alunos dentro da sala poderão sair. A festa só termina quando todos os n convidados participarem.

Implementação

Você deverá escrever um programa em C que receberá como argumentos os valores n e p, criará n *threads* representando n alunos, um *thread* segurança, e desenvolverá a solução para a sincronização entre eles, conforme o enunciado acima. Você também deverá receber como argumentos os seguintes tempos: t, que indica o intervalo máximo de chegada entre convidados; r, que indica o tempo máximo que cada aluno passa na festa; s, o tempo máximo que o segurança gasta em sua ronda, todos em milissegundos. Você só poderá utilizar a biblioteca POSIX threads (`pthread`) para a criação e sincronização dos *threads*. É importante que a sua solução imprima na tela os eventos que acontecem ao longo do tempo para que se possa avaliar se a sua solução está correta. Isso inclui a chegada dos alunos à festa, indicação se os alunos estão na festa ou foram embora, e o estado do segurança (se ele está na ronda, está inspecionando a sala etc.) e, finalmente, quando o programa terminar. Identifique cada aluno com um número único, começando em 1. Deixe para o usuário escolher quaisquer valores positivos para p e n, $p \leq n$. Os tempos t, r, s podem assumir qualquer valor maior ou igual a zero. Cada novo aluno (ou segurança) que vier à festa gasta um tempo aleatório entre 0 e t para chegar. Cada aluno gasta um tempo aleatório entre 0 e r na festa. A cada ronda do segurança ele deve gastar um tempo aleatório entre 0 e s. A sua solução deve ser livre de *deadlocks* e obedecer aos requisitos do problema.

Exemplos de execução

Observe abaixo alguns exemplos de execução e mensagens impressas indicando os eventos que devem acontecer. **Atenção:** Você precisa utilizar as mesmas frases para facilitar o trabalho de correção do seu EP!

```
$ ./ep2
Uso:
 ./ep2 <convidados> <minimo-alunos> <intervalo> <tempo_maximo>
Onde:
       <convidados>: numero total de convidados para a festa
    <minimo-alunos>: numero minimo de alunos na festa para o seguranca esvaziar a sala
        <intervalo>: intervalo maximo de tempo dentre chegadas de convidados em ms
      <tempo-festa>: tempo maximo de participacao na festa para cada aluno em ms
      <tempo-ronda>: tempo maximo de ronda do seguranca em ms

$ ./ep2 7 3 10 10 10
 Aluno 3 na porta.
 Aluno 3 na festa.
 Aluno 5 na porta.
 Aluno 5 na festa.
 Aluno 7 na porta.
 Aluno 4 na porta.
 Aluno 1 na porta.
Seguranca na porta.
 Aluno 7 na festa.
 Aluno 4 na festa.
 Aluno 1 na festa.
Segurança expulsa alunos.
 Aluno 1 vai embora.
 Aluno 6 na porta.
 Aluno 2 na porta.
 Aluno 7 vai embora.
 Aluno 3 vai embora.
 Aluno 4 vai embora.
 Aluno 5 vai embora.
Segurança em ronda.
 Aluno 6 na festa.
 Aluno 2 na festa.
Seguranca na porta.
Segurança em ronda.
 Aluno 6 vai embora.
 Aluno 2 vai embora.
Seguranca na porta.
Segurança inspeciona a sala.
Segurança em ronda.
Término de festa!

$ ./ep2 7 3 10 10 10
 Aluno 2 na porta.
 Aluno 2 na festa.
 Aluno 1 na porta.
 Aluno 1 na festa.
 Aluno 5 na porta.
 Aluno 5 na festa.
 Aluno 1 vai embora.
```

```
 Aluno 6 na porta.
 Aluno 6 na festa.
 Aluno 5 vai embora.
 Aluno 4 na porta.
 Aluno 4 na festa.
 Aluno 4 vai embora.
 Aluno 3 na porta.
 Aluno 3 na festa.
 Aluno 7 na porta.
 Aluno 2 vai embora.
Seguranca na porta.
 Aluno 7 na festa.
Segurança expulsa alunos.
 Aluno 6 vai embora.
 Aluno 7 vai embora.
 Aluno 3 vai embora.
Segurança em ronda.
Término de festa!

$ ./ep2 7 3 1 1 1
Seguranca na porta.
 Aluno 1 na porta.
 Aluno 2 na porta.
 Aluno 3 na porta.
 Aluno 4 na porta.
Segurança inspeciona a sala.
 Aluno 5 na porta.
 Aluno 6 na porta.
 Aluno 7 na porta.
Segurança em ronda.
 Aluno 1 na festa.
 Aluno 2 na festa.
 Aluno 3 na festa.
 Aluno 4 na festa.
 Aluno 5 na festa.
Seguranca na porta.
 Aluno 6 na festa.
 Aluno 7 na festa.
 Aluno 1 vai embora.
 Aluno 2 vai embora.
 Aluno 3 vai embora.
 Aluno 4 vai embora.
 Aluno 5 vai embora.
Segurança em ronda.
 Aluno 6 vai embora.
 Aluno 7 vai embora.
Seguranca na porta.
Segurança inspeciona a sala.
Segurança em ronda.
Término de festa!
```

```
$ ./ep2 7 3 1 10 1
Seguranca na porta.
 Aluno 1 na porta.
 Aluno 2 na porta.
 Aluno 3 na porta.
 Aluno 4 na porta.
Segurança inspeciona a sala.
 Aluno 5 na porta.
 Aluno 6 na porta.
 Aluno 7 na porta.
Segurança em ronda.
 Aluno 1 na festa.
Seguranca na porta.
 Aluno 2 na festa.
Segurança em ronda.
 Aluno 3 na festa.
 Aluno 4 na festa.
 Aluno 5 na festa.
 Aluno 2 vai embora.
 Aluno 6 na festa.
 Aluno 7 na festa.
Seguranca na porta.
Segurança expulsa alunos.
 Aluno 4 vai embora.
 Aluno 5 vai embora.
 Aluno 7 vai embora.
 Aluno 1 vai embora.
 Aluno 6 vai embora.
 Aluno 3 vai embora.
Segurança em ronda.
Seguranca na porta.
Segurança inspeciona a sala.
Segurança em ronda.
Término de festa!

$ ./ep2 7 3 0 0 0
Seguranca na porta.
 Aluno 1 na porta.
 Aluno 2 na porta.
 Aluno 3 na porta.
 Aluno 4 na porta.
Segurança inspeciona a sala.
 Aluno 5 na porta.
 Aluno 6 na porta.
 Aluno 7 na porta.
Segurança em ronda.
 Aluno 1 na festa.
 Aluno 2 na festa.
 Aluno 3 na festa.
```

```
 Aluno 4 na festa.
 Aluno 5 na festa.
 Aluno 6 na festa.
 Aluno 7 na festa.
Seguranca na porta.
 Aluno 1 vai embora.
 Aluno 2 vai embora.
 Aluno 3 vai embora.
 Aluno 4 vai embora.
 Aluno 5 vai embora.
 Aluno 6 vai embora.
 Aluno 7 vai embora.
Segurança inspeciona a sala.
Segurança em ronda.
```

Para controlar o intervalo de chegadas, o tempo que cada aluno fica na festa e o tempo gasto pelo segurança em suas rondas utilize a *syscall* `nanosleep()`. Ela dá acesso à um timer de alta resolução temporal. No entanto, para uso apropriado desta função, você precisará converter milisegundos em segundos e nanosegundos. Para gerar tempos (pseudo) aleatórios, utilize a função `rand()`.

Bônus

Inclua a possibilidade de se especificar mais que 1 segurança na sua solução. Em particular, adicione os tempos máximos de cada ronda de cada um dos seguranças como argumentos na linha de comando. Não mais que um segurança poderá inspecionar a sala ou expulsar alunos. Não é necessário garantir que todos os seguranças façam a inspeção ou expulsem alunos da festa.

Observações finais

Todo o EP deve ser escrito em C, sem a utilização de bibliotecas externas, exceto `-lpthread`. É fortemente recomendada a utilização de interfaces (*header files*, arquivos .h) e a organização do EP em módulos, que separem a definição da estrutura de dados e métodos utilizados. Isso ajudará principalmente *você*.

Sobre a elaboração:

Este EP pode ser elaborado por equipes de um ou dois alunos, desde que sejam respeitadas as seguintes regras:

- Os alunos devem trabalhar sempre juntos cooperativamente.
- Caso em um grupo exista um aluno com maior facilidade, este deve explicar as decisões tomadas. E o seu par deve participar e se esforçar para entender o desenvolvimento do programa (chamamos isso de *programação em pares*, que é uma excelente prática que vocês devem se esforçar para adotar).
- Mesmo a digitação do EP deve ser feita em grupo, enquanto um digita, o outro fica acompanhando o trabalho.

Sobre a avaliação:

- É sua responsabilidade manter o código do seu EP em sigilo, ou seja, apenas você e seu par devem ter acesso ao código.
- **Não serão toleradas cópias!** Exercícios copiados (com ou sem eventuais disfarces) levarão à reprovação da disciplina e o encaminhamento do caso para a Comissão de Graduação.
- Exercícios com erros de sintaxe (ou seja, erros de compilação) receberão nota zero. Em particular, lembre-se de validar os argumentos que o programa irá receber na linha de comando, garantindo que as condições do enunciado sejam observadas.
- É muito importante que seu programa seja elegante, claro e bem indentado, ou seja, digitado de maneira a ressaltar a estrutura de subordinação dos comandos do programa. A qualidade do seu trabalho sob esse ponto de vista influenciará sua nota!
- As informações impressas pelo seu programa na tela devem aparecer da forma mais clara possível, seguindo a especificação do enunciado. Esse aspecto também será levado em consideração no cálculo da sua nota.
- Uma regra básica é a seguinte: do ponto de vista do monitor responsável pela correção dos trabalhos, quanto mais convenientemente apresentado estiver o seu programa, melhor avaliado será seu trabalho.

Sobre a entrega:

- Entregar apenas um arquivo de nome `EP2.zip`, contendo todos os arquivos de seu exercício. Se quiser entregar também um arquivo texto contendo uma explicação sobre o programa e referências utilizadas (livros, internet etc.), dê a ele o nome `LEIAME`. Caso o seu programa tenha vários módulos, não se esqueça de incluir um Makefile.
- No início de cada arquivo, acrescente um cabeçalho bem informativo, como o seguinte:

```
/*****************************************************************/
/**   MAC 438  - Programação Concorrente                        **/
/**   IME-USP  - Primeiro Semestre de 2016                      **/
/**   Prof. Marcel Parolin Jackowski                            **/
/**                                                             **/
/**   Segundo Exercício-Programa                                **/
/**   Arquivo: EP2.c                                            **/
/**                                                             **/
/**   <nome do(a) aluno(a)>               <número USP>          **/
/**                                                             **/
/**   <data de entrega>                                         **/
/*****************************************************************/
```

  Não é obrigatório que o cabeçalho seja idêntico a esse, apenas que contenha pelo menos as mesmas informações.

- Para a entrega, utilize exclusivamente o Paca. Você pode entregar várias versões de um mesmo EP até o prazo, mas somente a última será armazenada pelo sistema.
- Não serão aceitas submissões por email ou atrasadas. Não deixe para a última hora, pois o sistema pode ficar congestionado e você poderá não conseguir enviar.

- Guarde uma cópia do seu EP pelo menos até o fim do semestre e, novamente, você é responsável por manter o sigilo de seu código-fonte.