LINGUAGEM DE PROGRAMAÇÃO II

Centro de Informática – Universidade Federal da Paraíba – Campus I

Professor: Carlos Eduardo Batista - Semestre: 2020.1

Entrega por e-mail: 23h59 de 30/11/2020.

O título do e-mail deve conter: "[LPII-20201-E002] NOME DO ALUNO – MATRICULA". Arquivo de entrega deve anexar todos os códigos fonte em C/C++ dentro de um diretório nomeado "MATRICULA_ALUNO-LPII-20201-E002" o qual deve ser comprimido em um arquivo ZIP ("MATRICULA_ALUNO-LPII-20201-E002"). O arquivo ZIP deve conter obrigatoriamente um arquivo de texto chamado README.txt contemplando todas as instruções de compilação e execução, além de qualquer observação que se fizer necessária para correção.

TRABALHO PRÁTICO: *LOCKS* E SOLUÇÕES JUSTAS

Montanha Russa concorrente (3,0 pts)

Suponha que existam *20* pessoas e um carro de uma montanha russa em um parque. As pessoas, repetidamente, esperam para dar uma volta no carro. O carro tem capacidade para *5* pessoas. Antes de começar uma volta, o carro deve aguardar um tempo fixo e partir, mesmo que não esteja cheio. Após dar uma volta na montanha russa (volta com tempo aleatório), cada pessoa passeia pelo parque de diversões e depois retorna à montanha russa para a próxima volta. O programa a ser desenvolvido deverá utilizar o algoritmo do *ticket* apresentado em sala de aula.

Tanto o carro quanto as pessoas devem ser representados por threads. O programa deverá ser finalizado (parque fechar) depois que o carro der 10 voltas. A implementação das *threads* das pessoas devem basear-se no seguinte pseudocódigo:

```
thread pessoa[i = 1 to 20]  {
    while (!fechouParque) {
        entraNoCarro();

      /*   Incrementa   contador   que   registra   o   número   de
           passageiros transportados pelo carro.  */
        esperaVoltaAcabar();
        saiDoCarro(); // decrementa o contador
        passeiaPeloParque(); // tempo aleatório
    }
}
```

Da mesma forma, a implementação da thread do carro deverá a seguinte estrutura:

```
thread carro {
    while (existemPassageirosNoParque) {
        esperaTempoArbitrario();
        daUmaVolta(); //Tempo aleatório
        esperaEsvaziar();
        volta++; // Indicador para o fechamento do parque.
    }
}
```

A implementação deverá ser feita em C/C++ utilizando a biblioteca Pthreads ou as threads STD (C++11), e o algoritmo do ticket deverá ser implementado usando a função atômica *fetch-and-add*[1]. A implementação deverá atender às quatro propriedades de uma solução para o problema da seção crítica: exclusão mútua, ausência de *deadlock*, ausência de atraso desnecessário e entrada eventual. A saída do seu programa deve ser bem planejada, de forma a mostrar o que está acontecendo a cada momento.

1 https://gcc.gnu.org/onlinedocs/gcc/_005f_005fsync-Builtins.html