Trabalho 1 – Serviço de lavação de carros

INE5611 – Sistemas Operacionais – UFSC
Profs. Cristian Koliver, Odorico M. Mendizabal

1 Definição do trabalho

O problema em questão consiste em modelar um serviço de lavação de carros usando *threads* e estruturas de sincronização (mutex e semáforos). A loja contém um estacionamento com até n vagas de espera, além do espaço para a lavagem, que comporta um único veículo por vez. Se não houver nenhum cliente para ser servido, o funcionário da loja de lavação pode descansar. Caso um cliente chegue com um veículo e todas as vagas estiverem ocupadas, então o cliente vai embora. Se o funcionário está ocupado, mas há vagas no estacionamento, então o cliente estaciona o veículo e aguarda. Caso o funcionário esteja descansando, ao chegar um cliente, o funcionário inicia o serviço imediatamente.

Você deve escrever um programa que coordene as ações entre os clientes e o funcionário da loja. Para auxiliar na estruturação do programa, algumas informações adicionais são passadas e você deve garantir que essas restrições sejam respeitadas pelo seu programa:

- *Threads* clientes devem invocar a função `carro_em_lavagem()` quando o seu carro estiver em lavagem;
- Se uma *thread* cliente chega na loja quando o estacionamento está lotado, ela desiste de lavar o carro, invocando uma função `desiste()`, que retorna com `pthread_exit()`;
- O funcionário da loja deve invocar a função `lavando_carro()` como forma de atender o cliente;
- Quando o funcionário invoca `lavando_carro()`, deve haver exatamente uma *thread* cliente invocando a função `carro_em_lavagem()` concorrentemente.

A Figura 1 apresenta as sugestões de variáveis e estruturas de sincronização que você pode usar na implementação do seu programa.

- *vagas* é o número total de carros que podem estar na loja. No exemplo da Figura 1: $vagas = 3$ significa que até 2 veículos pode estar aguardando e 1 sendo lavado;
- *clientes* contabiliza o número de clientes na loja. Essa variável deve ser protegida por *mutex*;
- o *funcionário* espera em *cliente* até que um *cliente* entre na loja, então o *cliente* espera em $funcionario$ até o *funcionário* sinalizar ao cliente que o seu carro será lavado;
- Após a lavação do carro, o *cliente* sinaliza $cliente\_pronto$ e espera em $funcionario\_pronto$.

```
vagas = 3
clientes = 0
mutex = mutex()
cliente = semaforo(0)
funcionario = semaforo(0)
cliente_pronto = semaforo(0)
funcionario_pronto = semaforo(0)
```

Figura 1: Dicas para resolver o problema de sincronização.

A Figura 2 ilustra uma execução do programa com 7 clientes e $vagas = 3$.

Dicas para a execução: Para tornar a execução mais realista e permitir mais aleatoriedade nas execuções, você pode adicionar tempos de espera aleatórios tanto na chegadas dos clientes na loja, quanto nos tempos para a lavagem do carro. A Figura 3 ilustra a função `lavando_carro()` com uma probabilidade dela levar entre 0 e 8 segundos.

```
$ ./lavacarro 7
funcionário chegou!
cliente 2 entrou na loja
cliente 2 posicionou carro para a lavagem
funcionário lavou um carro
cliente 6 entrou na loja
cliente 4 entrou na loja
cliente 5 entrou na loja
cliente 5 desistiu e foi embora!
cliente 3 entrou na loja
cliente 3 desistiu e foi embora!
cliente 2 saiu da loja
cliente 6 posicionou carro para a lavagem
cliente 0 entrou na loja
cliente 1 entrou na loja
cliente 1 desistiu e foi embora!
funcionário lavou um carro
cliente 6 saiu da loja
cliente 4 posicionou carro para a lavagem
funcionário lavou um carro
cliente 4 saiu da loja
cliente 0 posicionou carro para a lavagem
funcionário lavou um carro
cliente 0 saiu da loja
Nao há mais clientes
```

Figura 2: Um exemplo de saída com $vagas = 3$ e 7 *threads* cliente.

```
void lavando_carro(){
  sleep(rand() % 8);
  printf("funcionário lavou um carro\n");
  fflush(stdout);
}
```

Figura 3: Um exemplo de execução de função com uso de `sleep()` para simular o tempo de execução da função.

2 Grupos e Entrega

O trabalho poderá ser realizado **em trios**. Cada grupo deve entregar o código fonte e um breve relatório com (i) instruções para executar o programa, (ii) explicação sobre as estratégias de sincronização adotadas pelo grupo e (iii) uma análise da execução do programa considerando diferentes parâmetros. Procure explorar diferentes cenários com maior ou menor procura pelo serviço (você pode alterar o número de *threads*, vagas e tempo de execução das funções para ilustrar situações diversas).