Universidade Católica de Pernambuco - UNICAP
Recife, 2 de junho de 2024
Professor: Marcos José Canejo Estevão de Azevedo

Equipe 9: Eduardo Costa Braga - 848640
Henrique Franca Alves de Lima - 848801
Rafael Viana Angelim - 848869

Infraestrutura de Software - Implementando Mutex

Logo na parte inicial do código, alguns itens muito necessários para o funcionamento do processo como um todo são declarados globalmente.

A matriz global 5x5 representando a rua, vai ser nossa área compartilhada para todas as threads. Inicializada com '32', na tabela *ASCII* indica um espaço em branco.

```
int rua[5][5] = {
    {32,32,32,32,32},
    {32,32,32,32,32},
    {32,32,32,32,32},
    {32,32,32,32,32},
    {32,32,32,32,32}
};
```

Também vamos ter uma *struct* que será do tipo *car*, ela vai fazer com que cada carro criado tenha um ID, sua posição x e y (na Matriz[x][y]) e terá uma direção, que será identificado como 1 quando o sentido for de cima para baixo e 2 quando for da esquerda para a direita.

```
typedef struct car{
    int id;
    int x;
    int y;
    int direction;
}car;
```

Além disso, teremos uma array do tipo *car*, onde vamos ter armazenados todos os carros criados, uma variável para representar o sinal ou semáforo da rua que posteriormente será associada aos valores 1 para indicar a liberação da via vertical, e 2 para a horizontal.

```
#define VERMELHO    "\x1b[31m"
#define VERDE   "\x1b[32m"
#define RESET   "\x1b[0m"
```

Por fim, temos alguns outros detalhes que utilizaremos para saída do código, como definir o método de colorir na saída do terminal com as cores VERDE e VERMELHO do semáforo, e o RESET é apenas complemento desse método onde o que deve ser colorido, deve ficar entre o RESET e a cor desejada, posteriormente associamos essas cores às *strings* 'vertical' e 'horizontal' para facilmente mudar qual semáforo será que cor.

Temos algumas funções para modularizar e organizar melhor as funcionalidades do nosso código:

A função '*add()*', serve para atualizar de forma mais simples a posição de cada carro na matriz Rua.

A função '*mov()*', vai trabalhar com a movimentação do carro, em questão de percorrer a matriz Rua, nela é tratada as duas direções, caso o carro venha pela rua vertical ou pela rua horizontal. Também é tratado caso o carro chegue ao final da rua.

A função '*semaforo()*', criada para fazer a troca de sinal, onde caso seja vermelha vira verde e o contrário também.

A função '*criando_carros()*', serve para inicializar os carros com suas devidas posições na rua e suas direções, sendo (0,2) a posição inicial vertical e (2,0) a posição inicial horizontal.

A função '*func()*', é a principal parte do código, nela é realizada toda a movimentação da rua, através do acesso das *threads*, além de controlar a área crítica da rua (o cruzamento) para que apenas um sentido da via seja liberado por vez utilizando do

Mutex. A cada vez que um carro (*thread*) acessar a área restrita, o *Mutex* irá ser travado, bloqueando o acesso para o outro carro (*thread*), que terá que aguardar até que o outro carro saia da área restrita para o sinal mudar e poder prosseguir. Logo, o *Mutex* fará a função de um sinal de rua como na vida real.

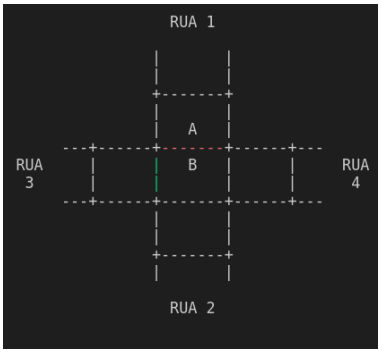

imagem 1

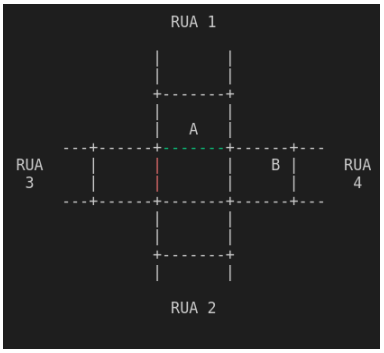

imagem 2

Na imagem 1, o Carro B entra na área crítica e o sinal é vermelho para o Carro A na rua 1. Logo depois, na imagem 2, vemos que o Carro B já saiu da área crítica e o Mutex é desbloqueado e logo em seguida o sinal da rua 1 fica verde para o Carro A.

Por fim, no *'main()'*, a função '*criando_carros()'* é chamada inicializando os dois carros com posições e direções iniciais. Tendo, em seguida, a função *´srand()´* inicializa o gerador de números aleatórios, juntamente com a definição aleatória da variável sinal como 1 (libera verticalmente) ou 2 (libera horizontalmente).

O *mutex*, inicializado com *'pthread_mutex_init'* que garante acesso à área crítica das threads. Com isso, a thread *'print'* é criada com funcionalidade de executar a função *'transito()'*, juntamente com as threads *'t1'* e *'t2'* que são criadas para cada carro executando a *'func()'*.

Assim, o *'pthread_join()'* é usado para aguardar que as threads *'print', 't1' e 't2'* terminem. Finalizando, o *'pthread_mutex_destroy'* é chamado para liberar os recursos alocados para o *Mutex*.