Problema do Metrô

Guilherme David Branco[1] - 110012470, Gabriella de Oliveira Esteves[1] - 110118995

[1]Departamento de Ciência da Computação
Universidade de Brasília (UNB)
Brasília – DF – Brazil

1. Introdução

O projeto do Problema do Metro apresenta a situação em que existem no mínimo uma estação de metro, no mínimo um passageiro e no mínimo um metro, onde o fluxo do percurso depende de certos fatores, como tempo espera do metro na estação, número de passageiros que deixam/saem da estação, etc. A Figura 1 faz uma simples representação do cenário.

O trajeto do metrô é circular e o numero de passageiros total em todas as estações é constante, assim como o número de estações. As pessoas tem ”consciência”de decidirem entrar ou sair do metro, porém isto só pode ser feito quando existe um metro na estação atual. Os metros esperam os passageiros por um tempo fixo e entao travam suas portas e vão à proxima estacao.

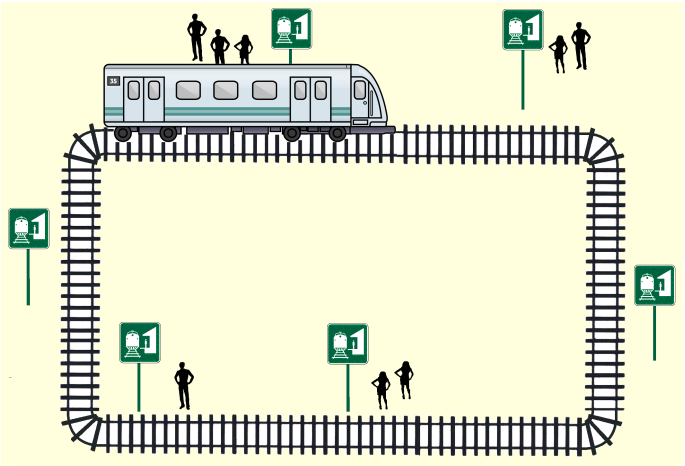

Figura 1. Cenário do Problema do Metrô.

2. Objetivo

Encontrar uma solução para o Problema do Metro, enunciado acima, utilizando recursos de programação concorrente.

3. Métodos

3.1. Descrição das Variáveis

Foram criadas três *structs* no programas, simbolizando pessoas, metros e estações.

A *struct estacao_t* possui a variável de condição *avisa*, que é utilizada para avisar aos passageiros que o metrô chegou na estação em questão, assim como um lock hold para não deixar que mais de um metro utilize a mesma estação. Por fim variáveis inteiras id e metro_estacao, identificando a estação e o metro que esta nela. Existem também dois métodos que auxiliam criação e destruição da struct.

```
1  typedef struct _estacao
2  {
3    pthread_cond_t avisa;
4    unsigned int id;
5    int metro_estacao;
6    pthread_mutex_t hold;
7  }estacao_t;
8
9  void EstacaoInit(estacao_t *estacao);
10 void EstacaoDestroy(estacao_t *estacao);
```

A *struct pessoa_t* possui assim como a struct anterior variáveis que apenas simbolizam identificações relacionadas a pessoa. Porém quatro métodos para facilitar criação, destruição e impressão de informação.

```
1  typedef struct _pessoa
2  {
3    unsigned int id;
4    unsigned int estacao_destino;
5    unsigned int estacao_atual;
6    unsigned int estado;
7    int meu_metro;
8  }pessoa_t;
9
10 void PessoaInit(pessoa_t *pessoa);
11 void ShowPessoa(pessoa_t pessoa);
12 void PessoaNovoDestino(pessoa_t *pessoa, int j);
13 void PessoaDestroy(pessoa_t *pessoa);
```

A *struct metro_t* possui identificadores como os anteriores, também dois mutexes porta e atualiza, a primeira garante a entrada e saída de pessoas, assim como a não entrada durante uma viagem, de maneira organizada, já a segunda serve apenas para atualização de variáveis internas a struct como qtd_pessoas. O semáforo lotacacao serve para se referir a quantidade de pessoas máxima dentro do vagão.

```
1  typedef struct _metro
2  {
3    unsigned int id;
4    unsigned int estacao_atual;
5    pthread_mutex_t porta;
6    pthread_mutex_t atualiza;
7    sem_t lotacao;
8    pthread_cond_t dentro;
9    unsigned int qtd_pessoas;
10   unsigned int estado;
11 }metro_t;
12
13 void MetroInit(metro_t *metro);
14 void MetroDestroy(metro_t *metro);
```

Foram criadas algumas constantes que definirão a simulação do programa.

- ESTADO_ENTRAR : Define que a pessoa deve entrar no metro
- ESTADO_SAIR : Define que a pessoa deve sair do metro
- ESTADO_FUNCIONANDO : Define que o metro esta funcionando
- ESTADO_QUEBRADO : Define qu e o metro não esta funcionando
- MAX_LOTACAO : Define a quantidade máxima de pessoas dentro do vagão
- QTD_PESSOAS : Define quantas pessoas existirão na simulação
- QTD_ESTACOES : Define quantas estações existirão na simulação
- QTD_METROS : Define quantos metros existirão na simulação
- METRO : Define qual número representa uma pessoa dentro do metro
- TEMPO_ESPERA_METRO : Define o tempo de espera do metro numa estação ao parar
- TEMPO_VIAGEM : Define o tempo de viagem do metro
- TEMPO_ESPERA_PESSOA : Define o tempo de espera da pessoa para gerar um novo destino e voltar a entrar no metro
- TEMPO_CONSERTAR : Define o tempo de conserto do metro

Para este trabalho, as constantes obtiveram os seguintes valores:

```
#define ESTADO_ENTRAR 0
#define ESTADO_SAIR 1
#define ESTADO_FUNCIONANDO 0
#define ESTADO_QUEBRADO 1
#define MAX_LOTACAO 4
#define QTD_PESSOAS 20
#define QTD_ESTACOES 5
#define QTD_METROS 2
#define METRO QTD_ESTACOES
#define TEMPO_ESPERA_METRO 5
#define TEMPO_VIAGEM 7
#define TEMPO_ESPERA_PESSOA 3
#define TEMPO_CONSERTAR 10
```

3.2. Método de Concorrência

Primeiramente foram inicializadas todas as variáveis das três estruturas principais do programa e em seguida foram criadas todas as threads das pessoas, direcionadas para a função *Parada* e do metrô, direcionadas para a função *Viagem*.

A função *controle_pessoa* funciona da seguinte maneira : Possui uma máquina de estados, de modo que cada pessoa esteja tentando sair ou entrar no metro. O estado de saída faz a pessoa sair ao chegar ao seu destino, caso ainda não tenha chegado ao seu destino ela deve esperar dentro do metro, e então pensa num novo destino que deseja ir. Já o estado de entrar faz a pessoa esperar numa dada estação atual até que o metro chegue na mesma.

```
void *controle_pessoa(void *id)
{
  int meu_id = (intptr_t)id;
  int estacao_id, metro_id;
  while(1)
  {
    estacao_id = pessoas[meu_id].estacao_atual;
    if(pessoas[meu_id].estacao_atual == METRO)
    {
      metro_id = pessoas[meu_id].meu_metro;
    }
    else
    {
      metro_id = estacoes[estacao_id].metro_estacao;
    }
    if(metro_id!=-1)
    {
      switch(pessoas[meu_id].estado)
      {
        case ESTADO_SAIR:
          /*Decide se a pessoa deve esperar dentro do metro ate chegar
              em sua estacao de destino*/
          pthread_mutex_lock(&metro[metro_id].porta);
          while(pessoas[meu_id].estacao_destino!=metro[metro_id].
              estacao_atual && pessoas[meu_id].estacao_atual == METRO)
          {
```

```
25              printf(" Pessoa %d esperando o metro %d chegar no destino %
                    d [Dentro do Metro]\n",pessoas[meu_id].id,metro_id,
                    pessoas[meu_id].estacao_destino);
26              pthread_cond_wait(&metro[metro_id].dentro,&metro[metro_id].
                    porta);
27            }
28            pthread_mutex_unlock(&metro[metro_id].porta);
29            sairmetro(meu_id,metro_id);
30            printf(" <<<< Pessoa %d saiu do metro %d na estacao %d\n",
                  pessoas[meu_id].id,metro_id,metro[metro_id].estacao_atual
                  );
31            sleep(TEMPO_ESPERA_PESSOA);
32            PessoaNovoDestino(pessoas, meu_id);
33            break;
34          case ESTADO_ENTRAR:
35            /*Decide se a pessoa deve esperar na estacao ate um metro
                  chegar nela*/
36            pthread_mutex_lock(&metro[metro_id].porta);
37            while(pessoas[meu_id].estacao_atual!=metro[metro_id].
                    estacao_atual && pessoas[meu_id].estacao_atual != METRO)
38            {
39              //printf("Pessoa %d esperando um metro chegar na estacao %d
                    [Fora do Metro]\n",pessoas[meu_id].id,pessoas[meu_id].
                    estacao_atual);
40              pthread_cond_wait(&estacoes[pessoas[meu_id].estacao_atual].
                    avisa,&metro[metro_id].porta);
41            }
42            pthread_mutex_unlock(&metro[metro_id].porta);
43            if(entrarmetro(meu_id,metro_id)==0)
44            {
45              printf(" >>>> Pessoa %d entrou no metro %d na estacao %d
                    com destino %d\n", pessoas[meu_id].id,pessoas[meu_id].
                    meu_metro,metro[metro_id].estacao_atual,pessoas[meu_id
                    ].estacao_destino);
46            }
47            break;
48        }
49      }
50    }
51    return 0;
52  }
```

A função *controle_metro* funciona da seguinte maneira : Possui uma máquina de estados, de modo que cada metro esteja funcionando ou quebrado. O metro funcionando chega a uma estação e avisa aos seus passageiros em qual estação ele esta para que eles possam sair caso necessário, também avisa às pessoas numa dada estação que ele chegou para que elas possam entrar. Nesta função também é dado um tempo limite para que ele possa avançar a próxima estação. Já o metro quebrado libera as pessoas na última estação, tranca as suas portas e espera um mecânico chegar para conserta-lo.

```
1  void *controle_metro(void *id)
2  {
3    int meu_id = (intptr_t)id;
4    int estacao_id;
5    unsigned int seed = time(NULL);
```

```
6    unsigned int seed2 = time(NULL)*meu_id;
7    while(1)
8    {
9      switch(metro[meu_id].estado)
10     {
11       case ESTADO_FUNCIONANDO:
12         //Segura as portas do metro para que as pessoas nao possam
               entrar quando for mudar de estacao
13         metro[meu_id].estacao_atual = (metro[meu_id].estacao_atual+1)%
               QTD_ESTACOES;
14         estacao_id = metro[meu_id].estacao_atual;
15         //Nao deixa outro metro chegar na estacao se ja existe algum la
16         pthread_mutex_lock(&estacoes[estacao_id].hold);
17         pthread_mutex_lock(&metro[meu_id].porta);
18         estacoes[estacao_id].metro_estacao = metro[meu_id].id;
19         printf("\nMetro %d chegou na estacao %d\n\n",metro[meu_id].id,
               metro[meu_id].estacao_atual);
20         //Solta as portas para que pessoas possam utilizar o metro
               enquanto estiver parado
21         pthread_mutex_unlock(&metro[meu_id].porta);
22         //Dar os avisos as pessoas, tanto de dentro do metro quanto na
               estacao que parou
23         pthread_cond_broadcast(&metro[meu_id].dentro);
24         pthread_cond_broadcast(&estacoes[metro[meu_id].estacao_atual].
               avisa);
25         //Espera um pouco para as pessoas poderem entrar no metro
26         sleep(TEMPO_ESPERA_METRO);
27         //Segura as portas para comecar a viagem
28         pthread_mutex_lock(&metro[meu_id].porta);
29         printf("----Metro %d esta em viagem----\n",metro[meu_id].id);
30         estacoes[estacao_id].metro_estacao = -1;
31         pthread_mutex_unlock(&estacoes[estacao_id].hold);
32         sleep(TEMPO_VIAGEM);//viagem
33         seed = rand_r(&seed)*rand_r(&seed2);
34         if(rand_r(&seed)%100 < 21)
35         {
36           metro[meu_id].estado = ESTADO_QUEBRADO;
37           printf("---|METRO %d QUEBROU|---\n", meu_id);
38         }
39         pthread_mutex_unlock(&metro[meu_id].porta);
40         break;
41       case ESTADO_QUEBRADO: //Nao implementado
42         pthread_mutex_lock(&metro[meu_id].porta);
43         sem_post(&mecanico);
44         sleep(TEMPO_CONSERTAR+1);
45         metro[meu_id].estado = ESTADO_FUNCIONANDO;
46         printf("---|METRO %d FUNCIONANDO|---\n", meu_id);
47         pthread_mutex_unlock(&metro[meu_id].porta);
48         break;
49     }
50   }
51   return 0;
52 }
```

4. Resultados e Conclusão

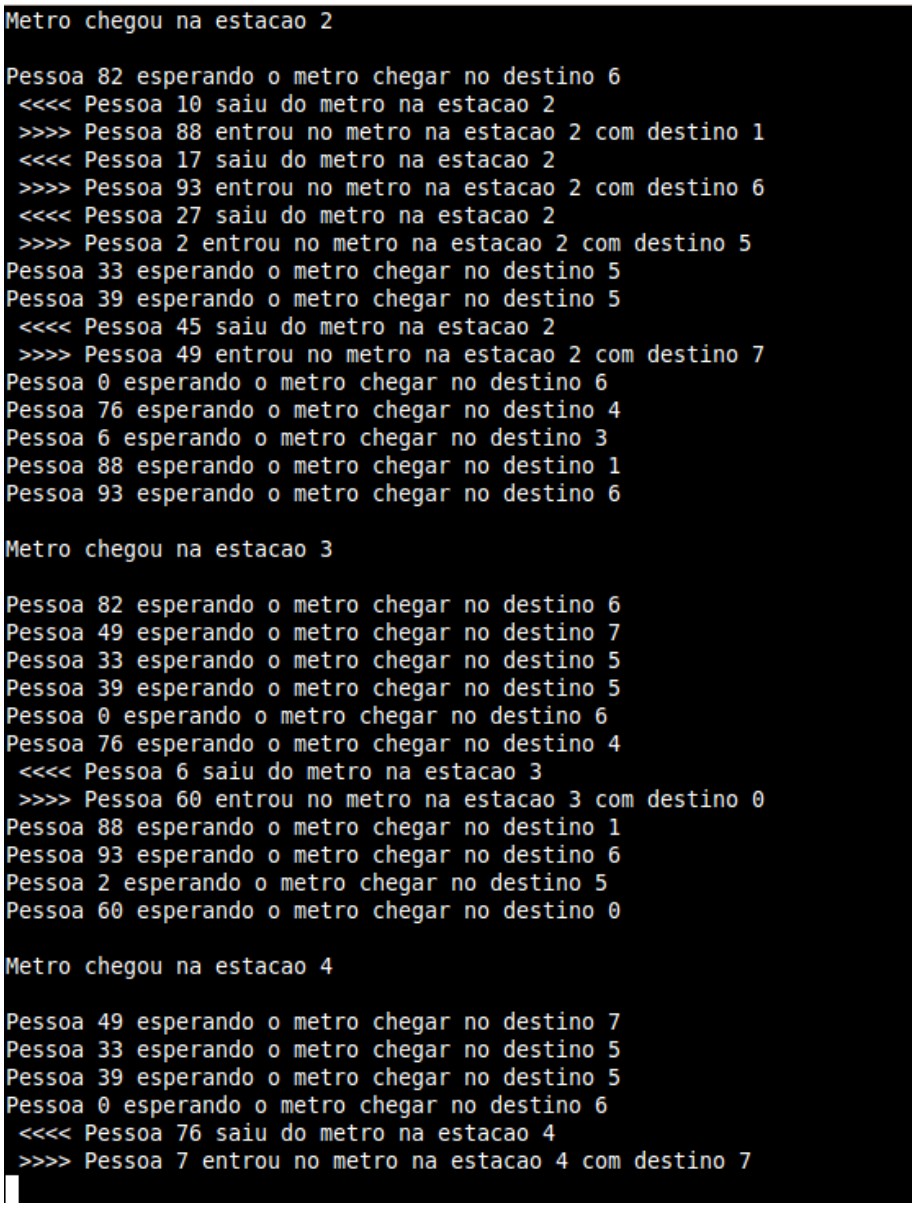

Figura 2. Cenário do Problema do Metrô.

A solução está de acordo com o especificado e possui abertura para melhorias, como escalonamento dos metros e mapeamento não circular de estações, entre outros.