Universidade de São Paulo
Instituto de Matemática e Estatística
Departamento de Ciência da Computação

MAC0438 - Programação Concorrente
EP1 - Modalidade Omnium de Ciclismo

Professor: Daniel Macedo Batista

Autoras:
Bárbara de Castro Fernandes - 7577351
Taís Aparecida Pereira Pinheiro - 7580421

São Paulo - SP, 22 de abril de 2015

Sumário

1 Introdução

Uma das várias modalidades de ciclismo realizada em velódromos é a Omnium, uma competição na qual os ciclistas vão conquistando pontos em diversas submodalidades de provas. Uma dessas submodalidades é a corrida por eliminação. Vamos considerar uma competição em que esta submodalidade será única para definir os medalhistas de ouro, prata e bronze.

Na modalidade por eliminação – a que vamos abordar – os ciclistas iniciam a prova enfileirados e, a cada duas voltas concluídas no velódromo, o último ciclista que passar pela linha de chegada é desclassificado.

A partir do momento em que houver 3 ciclistas, o primeiro destes a ser eliminado ganha a medalha de bronze, o segundo deles ganha a medalha de prata e o último ciclista que sobrar na pista ganha a medalha de ouro.

O comprimento do velódromo é dado pelo valor **d** e o número de ciclistas é dado por **n**. A posição inicial de cada ciclista é dada aleatoriamente e no máximo 4 ciclistas podem compartilhar um mesmo ponto da pista durante o decorrer da prova.

O objetivo deste exercício-programa é simular a submodalidade de corrida por eliminação e exibir as classificações de cada ciclista.

2 Desenvolvimento

2.1 Bibliotecas

Das bibliotecas-padrão da linguagem C utilizadas neste EP, destacam-se graças às suas funcionalidades as seguintes:

- **pthread.h**: Responsável pela criação e gerenciamento das threads correspondentes a cada ciclista.
- **semaphore.h**: Define um tipo de variável "semáforo", e é utilizada para realizar as operações com semáforos necessárias para o funcionamento correto das threads.

2.2 Estruturas principais

Este EP simulador trabalha com duas estruturas principais:

Ciclista

```
typedef struct ciclista
{
    pthread_t thread;
    int velocidade;
    int id;
    int volta;
    int posicao;
} Ciclista;
```

Lugar

```
typedef struct lugar
{
    int cic1;
    int cic2;
    int cic3;
    int cic4;
} Lugar;
```

A estrutura *Ciclista* é responsável por armazenar os dados de cada um dos n ciclistas e a pista de corrida é dada por um vetor de d "lugares"onde cada *Lugar* é apto para registrar os 4 possíveis ciclistas.

2.3 Inicialização

A princípio temos a instanciação dos n ciclistas e da pista de corrida, assim como a distribuição aleatória das posições de cada ciclista e sua devida alocação na pista de corrida. Essas determinações são feitas pelas seguintes funções:

- **void inicializaCiclistas(Ciclista*, int)**: Atribui valores de identificação de cada ciclista;
- **void inicializaPista()**: Instancia a pista de corrida;

- **inicializaSemaforos()**: Inicializa todos os semáforos presentes no programa;
- **void definePosicoesIniciais(Ciclista*)**: Determina posições dos ciclistas e os organiza na pista;

  Existem ainda outras funções menores que oferecem suporte a estas citadas.

2.4 Simulação da corrida

2.4.1 Corrida com velocidade uniforme

Na corrida com velocidade uniforme os ciclistas mantém a velocidade constante de 50Km/h durante toda prova. Desta forma, não há ultrapassagens.

2.4.2 Corrida com velocidade não uniforme

Na corrida com velocidade não uniforme, todos os ciclistas iniciam a corrida com velocidade de 25Km/h. A partir da segunda volta, no entanto, a velocidade deles passa a ser definida aleatoriamente (podendo ser 25Km/h ou 50Km/h), e continua a ser determinada desta forma a cada volta que eles derem.

2.4.3 Implementação da corrida

Um vetor de n threads é iniciado para gerenciar cada um dos n ciclistas.

Cada thread/Ciclista inicia a corrida pela função **void passouFinal(Ciclista*)**. Essa função gerencia cada volta do ciclista até que este seja eliminado ou seja vencedor da corrida. A função **void passouParcial(Ciclista*)** é utilizada como apoio, pois realiza cada avanço na pista e controla o número de voltas. Dependendo do número de voltas determina-se a probabilidade de "quebra"e o eliminado da rodada.

2.5 Exibição dos resultados

A relação dos resultados é exibida através de um relatório. Para obtenção dos dados utilizamos as seguintes funções:

- **void calculaUltimasColocacoes(Ciclista*,int*,int*,int*)**: Encontra os três ciclistas com últimas posições.
- **void calculaPrimeirasColocacoes(Ciclista*,int*,int*,int*)**: Encontra os três ciclistas com melhores posições.
- **void imprimeRelatorioParcial(Ciclista*)**: A cada duas voltas completadas pelo participante que se encontra em primeiro lugar, um relatório é impresso informando quem foi eliminado e os últimos colocados;
- **void imprimeRelatorioFinal(Ciclista*)**: Imprime o relatório final, que informa a colocação final de todos os ciclistas e quais destes ganharam medalhas;

3 Dificuldades

A principal dificuldade foi testar o programa a cada passo. Como este exercício foi o primeiro contato com um programa concorrente, na maioria dos momentos não conseguimos entender os resultados que obtínhamos e portanto houve muita dificuldade em traçar caminhos para construir a simulação da corrida.

4 Modo de execução

Este projeto foi entregue em um arquivo comprimido de extensão **.tar.gz** contendo 5 arquivos:

- **ep1.c**: Arquivo com o código fonte deste exercício-programa.
- **ep1.h**: Arquivo header do código fonte ep1.c.
- **Makefile**: Makefile para compilação do programa.
- **LEIAME**: Arquivo com a descrição técnica no programa.
- **Relatório.pdf**: Este relatório.

4.1 Compilação

Para compilar, digite:

```
$ make
```

4.2 Entrada

Existem três maneiras de se executar o programa principal após a devida compilação:

- **./ep1 d n u**: Realiza a corrida de n ciclistas em uma pista de tamanho d e todos os ciclistas tem velocidade constante = 50 Km/h.
- **./ep1 d n v**: Realiza a corrida de n ciclistas em uma pista de tamanho d e todos os ciclistas tem velocidade = 25 Km/h na primeira volta. A partir da segunda volta a velocidade é alterada de forma aleatória.
- **./ep1 d n [u|v] debug**: Modo debug. Neste modo, que pode ser executado tanto com a opção u quanto com a v, exibe-se na tela a cada 14,4s a volta em que cada ciclista está e a posição dele naquela volta.

  Exemplificando a execução para d= e n= no modo V:

```
taix@taix-K53SD:~/Área de Trabalho/1$ ./ep1 400 20 v
Resultado final 0:
        Medalha de ouro: id = 12
        Medalha de prata: id = 10
        Medalha de bronze: id = 19
```

  Exemplificando a execução para d= e n= no modo U:

```
taix@taix-K53SD:~/Área de Trabalho/1$ ./ep1 300 10 u
Resultado final 0:
        Medalha de ouro: id = 2
        Medalha de prata: id = 9
        Medalha de bronze: id = 7
```

5 Referências

Referências

[1] Enunciado Requerido

[2] POSIX Threads - Wikipedia

[3] POSIX threads explained - IBM