Manual - ThreadBet

Trabalho da disciplina Sistemas Operacionais

Daniel Jorge Manzano - 15446861
Nicolas Amaral dos Santos - 16304033

November 2025

1 Introdução

Este manual tem como objetivo servir como um guia ao usuário sobre como baixar, rodar e jogar o "ThreadBet", além de apontar resumidamente as aplicações e implementações de threads e semáforos presentes nele.

2 Instalação e Requisitos

Para a instalação, certifique-se de ter baixado todos os códigos necessários: *main.cpp*, *corredor.hpp* e *Makefile*, ou apenas baixe e extraia *trabalho_SO.zip*, que contém todos os arquivos necessários. Para encontrá-los, é possível acessar o repositório localizado em ThreadBet; a partir dele, clone o repositório localmente ou baixe o(s) arquivo(s) manualmente.

2.1 Compilação e Execução

Para compilar o jogo, é necessário ter um compilador C++ instalado (como g++). No terminal, navegue até a pasta do projeto e digite o comando

```
make
```

Com o código compilado, execute o jogo com o comando

```
make run
```

Nota: a qualquer momento, a execução pode ser interrompida forçadamente através do comando `Ctrl + C` (válido para Windows, Linux e macOS).
Para "limpar" a pasta (deletar o executável gerado), use o comando

```
make clean
```

3 Manual do Jogador

3.1 Fluxo do Jogo

Ao iniciar o **ThreadBet**, o usuário deve seguir os seguintes passos:

1. **Configuração do Sistema:** Selecionar o Sistema Operacional (Windows ou Linux/Mac) para garantir que a limpeza de tela funcione corretamente.
2. **Definição da Pista:** Escolher o tamanho da pista em metros (recomendado entre 500 e 1000 para uma boa visualização).
3. **Número de Corredores:** Definir quantos corredores participarão da prova (máximo de 20).
4. **Análise e Aposta:** O sistema gerará corredores com atributos aleatórios (Velocidade Mínima, Máxima e Resistência). O jogador deve analisar esses dados e escolher um ID para apostar.
5. **Corrida:** A corrida inicia. O jogador pode acompanhar o progresso em tempo real e verificar o resultado do pódio da corrida, assim como o sucesso de sua aposta, ao fim da simulação.

3.2 Legenda Visual

O progresso dos corredores é simbolizado por uma sequência de hífens (-). Ao fim dessa sequência, um caracter simboliza o movimento do corredor; os caracteres, com seus respectivos significados, são:

- $>$: O corredor está se movendo normalmente.
- **X** : O corredor tropeçou (azar) e perdeu o turno. O visual aparece apenas durante o tempo que ele está parado.
- # : O corredor sofreu uma lesão grave e foi eliminado da prova.
- **F** : O corredor finalizou a prova.

4 Implementação Técnica

A implementação do *ThreadBet* baseia-se nos conceitos de programação concorrente estudados na disciplina de Sistemas Operacionais. Nesse sentido, exploramos, abaixo, particularidades acerca disso.

4.1 Threads (std::thread)

Para simular uma corrida real, onde cada competidor age de forma independente e simultânea, utilizamos a biblioteca `<thread>` do C++.

- **Por que foi utilizada?** Se utilizássemos uma abordagem sequencial (um loop simples), um corredor teria que terminar seu movimento para que o outro começasse, ou teríamos que simular "tiques" de relógio manualmente. Com threads, o Sistema Operacional se encarrega do escalonamento, permitindo que cada instância da função `correr()` seja executada em paralelo (ou pseudo-paralelo, dependendo do hardware).
- **Como foi utilizada?** Cada corredor é instanciado como uma thread separada dentro de um `std::vector<thread>`. A função `correr` recebe por referência o objeto do corredor e os vetores de estado compartilhados. Ao final, utilizamos `join()` na thread principal (*main*) para garantir que o programa aguarde o término de todas as execuções antes de encerrar.

4.2 Sincronização e Exclusão Mútua (std::mutex)

Com múltiplas threads acessando e modificando recursos compartilhados simultaneamente, surgem problemas de *Race Condition* (Condição de Corrida). Para resolver isso, utilizamos Semáforos Binários (implementados via `std::mutex`).

Foram identificadas três Regiões Críticas principais no código:

1. **O Pódio (mutexVencedor):** O vetor `podio` é um recurso compartilhado. Se dois corredores chegassem na linha de chegada no exato mesmo ciclo de processamento, ambos tentariam fazer um `push_back` ao mesmo tempo, podendo corromper a memória ou perder dados. O mutex garante que apenas um corredor por vez registre sua chegada.
2. **Gerador de Números Aleatórios (mutexRNG):** A função `rand()` ou geradores do `std::random` mantêm um estado interno. O acesso concorrente a esse gerador pode causar falhas de segmentação ou gerar números repetidos/previsíveis. O mutex garante que apenas um corredor gere os números por vez.
3. **Contador de Finalizados (mutexThreads):** Para evitar que o jogo entre em loop infinito caso um corredor seja eliminado (não chegando ao pódio), utilizamos um contador global de threads encerradas. Como todas as threads incrementam essa variável ao sair, o acesso precisa ser atômico/protegido.

5 Conclusão

O desenvolvimento do *ThreadBet* cumpriu o objetivo de ilustrar o gerenciamento de processos em Sistemas Operacionais. A implementação da execução paralela

com threads e do controle de concorrência com semáforos resultou em uma aplicação lúdica que, além de funcional, serviu para consolidar o entendimento a respeito da aplicação dessas ferramentas.