UNIVERSIDADE FEDERAL DE PELOTAS
Centro de Desenvolvimento Tecnológico
Curso de Bacharelado em Ciência da Computação

Simulador Multithread de um Cyber Café Futurista (CyberFlux)

Fulano da Silva

Pelotas, 2025

Fulano da Silva

Simulador Multithread de um Cyber Café Futurista (CyberFlux)

Pelotas, 2025

1 INTRODUÇÃO

Este trabalho é sobre um simulador chamado CyberFlux, que representa um cybercafé futurista. A ideia é simular clientes (threads) competindo por recursos limitados: PCs, headsets de realidade virtual (VR) e cadeiras ergonômicas (GC, ou Gaming Chairs). O objetivo é aprender na prática sobre concorrência, threads e sincronização usando semáforos, evitando problemas como *deadlock* e *starvation*.

Este relatório apresenta a solução criada, os desafios encontrados e resultados obtidos.

2 DESENVOLVIMENTO DA SOLUÇÃO

2.1 Visão Geral do Algoritmo

O código (`cyberflux.c`) cria threads que representam clientes com diferentes necessidades:

- **STUDENT**: precisa apenas de um PC.
- **GAMER**: precisa de PC, headset VR e cadeira, com prioridade maior para PC e VR.
- **FREELANCER**: precisa de PC, cadeira e headset VR, com prioridade maior para PC e cadeira.

Para garantir o controle dos recursos, usamos três semáforos: `semPC` (10 PCs), `semVR` (6 headsets VR) e `semGC` (8 cadeiras).

2.1.1 Modos de Alocação

O programa tem dois modos principais de alocação, selecionados pelo parâmetro `--force-deadlock` (ou por prompt no início):

1. **Evita deadlock (forceDeadlock=0)**:
   Usa a estratégia *All or Nothing*, onde o cliente só prossegue se conseguir todos os recursos necessários simultaneamente. Caso contrário, libera tudo e tenta de novo.
2. **Força deadlock (forceDeadlock=1)**:
   Clientes diferentes tentam pegar recursos em ordens diferentes propositalmente, aumentando as chances de deadlock.

2.1.2 Funcionamento da Simulação

A simulação acontece por um período fixo (horas simuladas em segundos reais). Cada cliente/thread:

1. Tenta adquirir os recursos.
2. Desiste (starvation) se não conseguir um PC em até `MAX_WAIT_BEFORE_GIVEUP`.
3. Usa os recursos por tempo aleatório, depois os libera.

No final, são exibidas estatísticas sobre uso dos recursos e clientes atendidos.

3 COMPILAÇÃO E EXECUÇÃO

3.1 Compilação

Para compilar no Linux usando `gcc` e *pthread*, rode:

```
gcc cyberflux.c -o cyberflux -lpthread
```

3.2 Execução

Execute o programa com:

```
./cyberflux [opções]
```

Parâmetros disponíveis:

- `--clients-min <N>`: número mínimo de clientes.
- `--clients-max <N>`: número máximo de clientes.
- `--open-hours <N>`: horas simuladas (cada "hora"é alguns segundos reais).
- `--force-deadlock 0` ou `1`: escolhe o modo de operação (evitar ou forçar deadlock).
- `--verbose 0` ou `1`: ativa mensagens detalhadas durante execução.
- `-h` ou `--help`: mostra ajuda.

Exemplo:

```
./cyberflux --clients-min 20 --clients-max 50 --open-hours 8 --verbose 1
```

4 DIFICULDADES ENCONTRADAS

A maior dificuldade foi entender exatamente a especificação fornecida, pois havia espaço para diferentes interpretações sobre as prioridades dos clientes. Por exemplo:

- GAMER poderia tentar pegar PC e VR simultaneamente, e depois a cadeira, ou em sequência.
- FREELANCER poderia priorizar cadeira e PC ao mesmo tempo ou em sequência.
- STUDENT poderia querer recursos opcionais (VR e cadeira) ou não.

Após analisar, escolhemos a abordagem *All or Nothing*, em que clientes só começam se conseguirem todos os recursos simultaneamente. Essa escolha simplifica a implementação e elimina possibilidades de deadlocks.

Outras interpretações poderiam complicar a implementação e aumentar o risco de deadlocks, exigindo técnicas mais complexas como rollback ou detecção ativa.

5 CONCLUSÃO E TRABALHOS FUTUROS

O simulador CyberFlux permitiu uma compreensão prática dos conceitos de concorrência e sincronização. A estratégia *All or Nothing* mostrou-se eficaz para evitar deadlocks, embora possa causar maior tempo de espera em casos específicos.

Sugestões para melhorias futuras:

- Políticas de prioridade mais detalhadas entre clientes.
- Implementação de algoritmos avançados de escalonamento e *fairness*.
- Adição de recursos extras na simulação, como impressoras 3D ou salas de reunião.