Exercícios com Mutex, Semáforos e Monitores

INF01151 - Sistemas Operacionais II - Prof. Eder John Scheid

9 de setembro de 2025

Este é o **segundo** de quatro exercícios práticos que compõem parte da avaliação prática (MPP). Os quatro exercícios, no total, valem 1,0 ponto da MPP; portanto, cada exercício corresponde a 0,25 ponto (25% do total).

Objetivo

Praticar a utilização de métodos de exclusão mútua e sincronização de threads por meio de **sem semáforos**, **mutex** e **monitores**.

Exercícios

1. **Soma em variável global (com seção crítica usando *mutex*)** *Linguagem: C (pthreads)*
   Objetivo didático: exercitar *exclusão mútua* com `pthread_mutex_t` para eliminar a condição de corrida na atualização de uma variável global.

   Crie um vetor **global** `int A[N]` com N **constante no código** (ex.: `#define N 1000000`) e uma variável **global** `long long SUM = 0`. Divida `A` em T fatias contíguas. Cada thread percorre **apenas sua fatia**, calcula um **parcial local** e, ao final, entra em uma **seção crítica** protegida por um **mutex** para acumular o parcial em `SUM`. A `main` aguarda todas com `pthread_join`, calcula a soma sequencial e compara os resultados.

   - **Requisitos:** (i) `A` e `SUM` **globais**; (ii) N constante no código; (iii) T pela linha de comando; (iv) **usar exclusivamente** `pthread_mutex_t` para proteger a seção crítica de atualização de `SUM`; (v) **não** usar semáforos nem operações atômicas; (vi) **sem** *busy-wait*; (vii) imprimir `SUM_paralelo`, `SUM_sequencial` e `OK`/`ERRO`; (viii) inicializar e destruir corretamente o mutex.
   - **Dica de compilação/execução:**
     ```
     gcc -O2 -pthread soma_global_mutex.c -o soma
     ./soma <T>
     ```

2. Estacionamento com S vagas (semáforos) *Linguagem: C (pthreads)*
Objetivo didático: exercitar controle de recursos com semáforos de contagem, sem *busy-wait*.

Crie N threads (carros) que competem por S vagas de um estacionamento. Cada carro: (i) "chega" após um atraso aleatório; (ii) tenta entrar ocupando uma vaga (primitiva **P**); (iii) permanece um tempo aleatório usando a vaga; (iv) sai, liberando a vaga (primitiva **V**). Imprima os eventos (chegada, entrada, saída) no terminal.

- **Requisitos:** (i) usar um **semáforo de contagem** inicializado com S para representar vagas; (ii) finalizar apenas após todas as threads terminarem; (iii) destruir o semáforo ao final.
- **Parâmetros:** N (nº de carros), S (vagas) e intervalos de chegada/uso (ms) por linha de comando (ou constantes no código).
- **Invariante a verificar:** ocupação $\leq S$ em todos os instantes.
- **Dica de compilação/execução:**
```
gcc -O2 -pthread estacionamento.c -o estac
./estac <N> <S> <chegada_max> <uso_max>
```
- **Dicas:** Você pode usar `usleep(microseconds)` da biblioteca `unistd.h` para simular o atraso na chegada e o tempo estacionado na vaga.

3. Ping–Pong (monitores) *Linguagem: Java (monitores nativos)*
Objetivo didático: praticar `synchronized` + `wait/notifyAll` com alternância de turnos entre duas threads.

Implemente uma classe `PingPong` que coordena duas threads: uma imprime `"ping"` e a outra imprime `"pong"`, alternadamente, por N iterações (`ping, pong, ping, pong, ...`). Use apenas monitores nativos de Java.

- **Requisitos:** (i) implementar métodos `doPing()` e `doPong()` *synchronized* que respeitem um *turno* compartilhado; (ii) esperar com `wait` dentro de `while` quando não for o turno; (iii) sinalizar a troca de turno com `notifyAll`; (iv) **não** usar `Semaphore`, `Lock`/`Condition`, `BlockingQueue` ou outras estruturas prontas; (v) a `main` cria as duas threads, inicia, e faz `join()` em ambas.
- **Parâmetro:** N (número de pares `ping/pong`), por linha de comando ou constante.
- **Saída esperada (exemplo):**
```
ping
pong
ping
pong
```
... (até completar N pares)
- **Dica de compilação/execução:**
```
javac PingPongDemo.java
java PingPongDemo <N>
```

Entrega

- Envie no Moodle um arquivo comprimido (*e.g.,* `.zip` ou `.tar.gz`) contendo os 3 arquivos fonte (*i.e.,* `.c` e `.java`).