1. Processos e Threads

Exercício 1.1: Implemente um programa em C que cria 3 processos filhos usando fork(). Cada filho deve imprimir:

"Processo filho (PID: X), PID do pai: Y".

O processo pai deve aguardar todos os filhos terminarem e imprimir "Processo pai finalizado".

Exercício 1.2: Crie um programa em C que inicia 5 threads. Cada thread deve imprimir "Thread ID: TID" (use pthread_self()). Use pthread_join para sincronizar.

2. Exclusão Mútua

Exercício 2.1: Crie 10 threads que incrementam um contador global 1000 vezes cada. Ao final, o programa imprime o valor do contador.

Execute sem sincronização e observe o resultado incorreto.

Corrija usando mutexes (ex: pthread_mutex_t).

3. Semáforos

Exercício 3.1: Implemente um buffer circular de tamanho 5 usando semáforos:

1 thread produtor insere números de 1 a 10.

1 thread consumidor remove e imprime os números.

Use semáforos para vazio, cheio e um mutex para acesso ao buffer.

Dica (semáforos em C):

```
sem_t empty, full;
sem_init(&empty, 0, 5); // Inicializa semáforo de vazios
sem_init(&full, 0, 0);  // Inicializa semáforo de cheios
```

4. Monitores

Exercício 4.1: Em Java, implemente uma fila com tamanho máximo 3 usando synchronized,

wait() e notifyAll().

Método put() para adicionar itens (bloqueia se cheia).

Método take() para remover itens (bloqueia se vazia).

Esqueleto:

```
public class BlockingQueue<T> {
    private final Queue<T> queue = new LinkedList<>();
    private final int maxSize;

    public synchronized void put(T item) throws
InterruptedException {
        // Complete aqui
    }

    public synchronized T take() throws InterruptedException {
        // Complete aqui
    }
}
```

5. Correção de Bugs

Exercício 5.1. Por que este código cria processos zumbis? Corrija-o usando wait().

```
#include <stdio.h>
#include <unistd.h>

int main() {
    for (int i = 0; i < 3; i++) {
        if (fork() == 0) {
            printf("Filho %d criado\n", getpid());
```

```
            return 0;
        }
    }
    sleep(5);
    printf("Pai finalizado\n");
    return 0;
}
```

Exercício 5.2. Identifique os 2 erros de sincronização e corrija-os com pthread_join e mutex.

```
#include <pthread.h>
#include <stdio.h>
int counter = 0;

void *increment(void *arg) {
    for (int i = 0; i < 1000; i++) {
        counter++;
    }
    return NULL;
}

int main() {
    pthread_t t1, t2;
    pthread_create(&t1, NULL, increment, NULL);
    pthread_create(&t2, NULL, increment, NULL);
    printf("Counter: %d\n", counter);
    return 0;
}
```

Exercício 5.3. Por que ocorre deadlock? Corrija a inicialização e uso dos semáforos.

```
sem_t empty, full;
int buffer[5], index = 0;

void produtor() {
    for (int i = 1; i <= 10; i++) {
        sem_wait(&full);
        buffer[index++] = i;
        sem_post(&empty);
    }
}
```

Exercício 5.4. Em um cenário com múltiplos consumidores, por que alguns threads podem ficar em starvation? Corrija o código.

```
public class Fila {
    private Queue<Integer> fila = new LinkedList<>();

    public synchronized void adicionar(int item) {
        fila.add(item);
        notify();
    }

    public synchronized int remover() throws InterruptedException {
        while (fila.isEmpty()) {
            wait();
        }
```

```
        return fila.poll();
    }
}
```

Exercício 5.5. Por que ocorre deadlock? Implemente uma solução (ex: ordenar garfos).

```
sem_t garfo[5];
void filosofo(int id) {
    while (1) {
        sem_wait(&garfo[id]);
        sem_wait(&garfo[(id+1)%5]);
        // Come...
        sem_post(&garfo[id]);
        sem_post(&garfo[(id+1)%5]);
    }
}
```

Exercício 5.6. Há uma condição de corrida na manipulação de cadeiras_livres. Corrija o uso dos semáforos.

```
sem_t clientes = 0, barbeiro = 0, mutex = 1;
int cadeiras_livres = 5;

void cliente() {
    sem_wait(&mutex);
    if (cadeiras_livres > 0) {
        cadeiras_livres--;
        sem_post(&clientes);
        sem_post(&mutex);
```

```
        sem_wait(&barbeiro);
    } else {
        sem_post(&mutex);
    }
}

void barbeiro() {
    while (1) {
        sem_wait(&clientes);
        sem_wait(&mutex);
        cadeiras_livres++;
        sem_post(&barbeiro);
        sem_post(&mutex);
        // Corta cabelo...
    }
}
```

Exercício 5.7. Escritores podem sofrer starvation. Modifique o código para priorizá-los.

```
public class BancoDeDados {
    private int leitores = 0;

    public synchronized void iniciarLeitura() {
        leitores++;
    }

    public synchronized void terminarLeitura() {
```

```
        leitores--;
        if (leitores == 0) notifyAll();
    }

    public synchronized void iniciarEscrita() {
        while (leitores > 0) wait();
    }

    public synchronized void terminarEscrita() {
        notifyAll();
    }
}
```

6. Responda

Exercício 6.1: No código do Barbeiro Sonolento, o que acontece se o semáforo mutex for removido?

Exercício 6.2: No Jantar dos Filósofos, qual é o impacto de permitir que um filósofo pegue um garfo e depois verifique se o outro está livre?

Exercício 6.3: Por que usar notify() em vez de notifyAll() no código da fila Java pode causar starvation?

Instruções Gerais:

Para C: Compile com gcc -pthread.

Teste os programas múltiplas vezes para observar inconsistências sem sincronização.

Analise saídas usando printf para depurar threads/processos.