Universidade Federal de Campina Grande
Departamento de Sistemas e Computação
Disciplina: Programação concorrente
Professor: Thiago Emmanuel Pereira

(As respostas, tanto para as questões teóricas quanto para as práticas devem ser disponibilizadas em um repositório privado do github compartilhado com o professor. A avaliação considerará a defesa oral das soluções)

1. Em sala, mostramos o seguinte trecho de código (desprotegido). Nesse código, três threads incrementam um contador compartilhado. Cada thread realiza 10.000.000 operações de incremento. Modifique o código para que nenhum incremento seja perdido, e, ao fim da execução do programa, o valor do contador seja 30.000.000. Execute experimentos para coletar o tempo de execução de cada uma das duas versões do programa. Explique a razão da diferença de desempenho. Forneça evidências que corroborem sua explicação (no Linux, você pode usar as ferrametnas eBPF e perf, por exemplo).

```c
#include <stdio.h>
#include <assert.h>
#include <pthread.h>

long int counter = 0;

void* run (void* args) {
    int my_id;
    long int j;

    my_id = (int) args;

    for (j = 0; j < 1e7; j++) {
        counter = counter + 1;
    }

    printf("my_id=%d j=%ld counter=%ld\n", my_id, j, counter);
    pthread_exit(my_id);
}

int main (int argc, char *argv[]) {

    int i;
    pthread_t pthreads[3];

    for (i = 0; i < 3; i++) {
        pthread_create (&pthreads[i], NULL, &run, (void*) i);
    }

    for (i = 0; i < 3; i++) {
        pthread_join(pthreads[i], NULL);
    }

    printf("counter=%ld\n", counter);
}
```

2. Uma abstração bastante usada em programação concorrentes são os canais. Um canal recebe mensagens enviadas por processos (threads) remetentes. Processos recipientes lêem as mensagens enviadas no canal. Mensagens devem ser lidas na ordem que entraram no canal. Uma vez lida, a mensagem não pode ser lida novamente. O canal deve ter uma capacidade máxima, ou seja, ao atingir o limite, novas mensagens não podem ser enviadas para o canal imediatamente. Considere que o construtor do canal recebe um inteiro que indica sua capacidade máxima. Mensagens não podem ser descartadas. Implemente a interface abaixo para o canal, usando quaisquer mecanismos de coordenação e controle de concorrência da linguagem Java, exceto as estruturas de dados de *Concurrent Collections.* Considere tanto critérios de corretude quanto de eficiência (p.ex evite spin locks quando possível).

```
public interface Channel {
	public void putMessage(String message);
	public String takeMessage();
}
```

3. Considere um programa com duas funções principais indicadas abaixo:

int gateway(int num_replicas)

int request()

A função **request** deve sortear um número aleatório entre 1 e 30, dormir por uma quantidade de segundos dada pelo número aleatório sorteado, e retornar o valor do número. Por sua vez, a função **gateway** deve iniciar ***num_replicas*** threads, e cada thread executará a função **request**.

a) Escreva, em Java e em C, uma versão do programa em que a função **gateway** retorna o valor da primeira thread que termine a execução da função **request** (sem que seja necessário esperar as demais threads terminarem).
b) Escreva, em Java e em C, ma versão do programa em que a função **gateway** retorna a soma dos valores retornados na execução da função **request** por todas as réplicas criadas.

obs 1) Pequenas mudanças na API indicada podem ser realizadas, para que seja possível adaptar o código para as linguagens indicadas. Contacte o professor caso queira realizar alguma mudança.

obs 2) Em Java, não use nenhuma biblioteca além da SDK. Também, não considere nenhum mecanismo avançado de controle de concorrência (somente as classes/interfaces Runnable e Thread, bem como as primitivas synchronized, volatile e wait/notify). Em C, use pthreads.

4. Considere novas versões para os programas acima de modo que:
    a. A função **gateway** retorne -1 caso nenhuma thread tenha terminado antes de 8 segundos nos programas da questão **3a**.
    b. A função **gateway** retorne -1 caso alguma thread ainda esteja executando após 16 segundos nos programas da questão **3b.**

5. Realize uma comparação de desempenho entre os seguintes pares de estruturas de dados alternativas. Sua resposta deve conter uma descrição dos experimentos realizados, o código para executar os experimentos, bem como o código para analisar os dados experimentais. Alem disso, uma breve descrição dos resultados alcançados com plots que os ilustram.
    a. ConcurrentHashMap vs Collections.synchronizedMap
    b. CopyOnWriteArrayList vs Collections.synchronizedList

    obs) É importante avaliar as diferenças de desempenho conforme o número de threads aumenta (ou seja, conforme aumenta a conteção). Ainda, é importante avaliar a proporção da carga de trabalho (entre operações que alteram a estrutura de dados e operações que não alteram). Particularmente, o tamanho das estruturar pode ser também um fator importante (é o caso claro de CopyOnWriteArrayList).