SO - EXAME

Questão I.IV: Considere um sistema computacional em que o espaço de endereçamento lógico (virtual) total é de 64 páginas, em que cada página contém um tamanho de 1024 bits, mapeado num espaço de memória física de 32 "frames". Quantos bits são necessários para representar um endereço lógico e físico?

(a) Endereço lógico:6; Endereço físico:5;

(b) Endereço lógico:16; Endereço físico:15;

(c) Endereço lógico:32; Endereço físico:16;

(d) Endereço lógico:64; Endereço físico:32;

Resposta: (b) Endereço lógico:16; Endereço físico:15.

Explicação:

- **Endereço lógico:** 64 páginas requerem 6 bits para representar o número da página (2^6 = 64). Cada página tem 1024 bits, o que requer 10 bits para o deslocamento (2^10 = 1024). Portanto, o endereço lógico total é 6 + 10 = 16 bits.
- **Endereço físico:** 32 frames requerem 5 bits para representar o número do frame (2^5 = 32). O deslocamento dentro do frame é o mesmo que na página, 10 bits. Portanto, o endereço físico total é 5 + 10 = 15 bits.

Questão II [1,0 valor]

Considerando as várias definições de um sistema operativo, discuta se o mesmo poderá incluir como programas de sistema um "web browser" ou um programa de e-mail. Justifique se deverá incluir ou não.

Resposta: Um sistema operativo tradicionalmente inclui programas de sistema que gerenciam recursos de hardware e fornecem serviços básicos para aplicativos de usuário. Um navegador web (web browser) ou um programa de e-mail são considerados aplicativos de usuário, não programas de sistema. Portanto, um sistema operativo não deve incluir esses programas como parte de seu núcleo, mas pode fornecer interfaces e serviços que permitam que esses aplicativos funcionem corretamente.

Questão III [2,0 valores]

"As versões originais de alguns dos mais populares sistemas operativos para dispositivos móveis não disponibilizavam processamento concorrente".

Comente a afirmação, destacando a razão pela qual originalmente os dispositivos móveis não disponibilizavam processamento concorrente e indique pelo menos três consequências/desafios que a adição do processamento concorrente a estes sistemas operativos pode trazer.

Resposta: Originalmente, os dispositivos móveis não disponibilizavam processamento concorrente devido a limitações de hardware, como processadores menos potentes e menor capacidade de memória. A adição de processamento concorrente trouxe desafios como:

1. **Gerenciamento de recursos:** A necessidade de gerenciar eficientemente a CPU, memória e outros recursos entre múltiplas tarefas.
2. **Sincronização:** Garantir que as tarefas concorrentes não interfiram umas com as outras, evitando condições de corrida e deadlocks.
3. **Consumo de energia:** O processamento concorrente pode aumentar o consumo de energia, o que é crítico em dispositivos móveis com bateria limitada.

Questão VI [5.0 valores]

a) Identifique no excerto de código apresentado uma seção crítica, fundamentando a sua indicação. [1,5 valores]

No código fornecido, a seção crítica está no método run() da classe Exemplo. A seção crítica é a parte do código onde a variável i é modificada. Como múltiplas threads podem acessar e modificar essa variável simultaneamente, isso pode levar a condições de corrida (race conditions), onde o valor de i pode ser inconsistente devido à execução concorrente.

```
public void run() {
    for (int i = 0; i < 20; i++) {
        i = i + 1; // Seção crítica
    }
}
```

b) Mantendo a estrutura base, altere o código apresentado de forma a garantir, utilizando o mecanismo de sinalização entre thread, que as seções críticas do código apresentado sejam protegidas. [2,5 valores]

```
public class Exemplo implements Runnable {
    private int i = 0; // Variável compartilhada entre as threads

    public synchronized void run() {
        for (int j = 0; j < 20; j++) {
            i++;
        }
        notifyAll(); // Notifica que a atualização foi concluída
    }

    public synchronized void printI() {
        try {
            wait(); // Aguarda até que alguma thread termine a execução de run()
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
        System.out.println("-> " + i);
    }

    public static void main(String[] args) {
        Thread[] ths = new Thread[5];
        Exemplo exemplo = new Exemplo(); // Criamos apenas uma instância compartilhada

        for (int i = 0; i < ths.length; i++) {
            ths[i] = new Thread(exemplo); // Todas as threads compartilham o mesmo objeto
            ths[i].start();
        }

        exemplo.printI(); // Aguarda e imprime o valor final de 'i'
    }
}
```

c) Explique a diferença com exemplos entre os mecanismos de sincronização Semáforo e Monitor. [1,0 valores]

Semáforo: Um semáforo é uma variável inteira que é acessada apenas através de duas operações atômicas: wait() (ou P()) e signal() (ou V()). Ele é usado para controlar o acesso a um recurso compartilhado por múltiplas threads. Um exemplo comum é limitar o número de threads que podem acessar um recurso simultaneamente.

Monitor: Um monitor é uma construção de alto nível que encapsula a sincronização de threads. Em Java, cada objeto tem um monitor associado, que pode ser acessado usando a palavra-chave synchronized. O monitor garante que apenas uma thread por vez pode executar um bloco de código sincronizado.

Questão VII [5.0 valores]

Escreva um programa (com as classes e estruturas dados que entender adequadas) em Java que simule uma situação de bloqueio (deadlock) e um método para o evitar utilizando para o efeito o algoritmo do banqueiro.

```
import java.util.Arrays;

class Resource {
    private final String name;

    public Resource(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }
}
```

```
class DeadlockSimulation {
    public static void main(String[] args) {
        Resource resource1 = new Resource("Recurso 1");
        Resource resource2 = new Resource("Recurso 2");

        // Criando a primeira thread que tenta bloquear resource1 e depois resource2
        Thread thread1 = new Thread(new Runnable() {
            @Override
            public void run() {
                synchronized (resource1) {
                    System.out.println(Thread.currentThread().getName() + " bloqueou " + resource1.getName());
                    try {
                        Thread.sleep(50); // Simula um pequeno atraso
                    } catch (InterruptedException ignored) {
                    }
                    synchronized (resource2) {
                        System.out.println(Thread.currentThread().getName() + " bloqueou " + resource2.getName());
                    }
                }
            }
        });
```

```
        // Criando a segunda thread que tenta bloquear resource2 e depois resource1
        Thread thread2 = new Thread(new Runnable() {
            @Override
            public void run() {
                synchronized (resource2) {
                    System.out.println(Thread.currentThread().getName() + " bloqueou " + resource2.getName());
                    try {
                        Thread.sleep(50); // Simula um pequeno atraso
                    } catch (InterruptedException ignored) {
                    }
                    synchronized (resource1) {
                        System.out.println(Thread.currentThread().getName() + " bloqueou " + resource1.getName());
                    }
                }
            }
        });

        // Iniciando as threads
        thread1.start();
        thread2.start();
    }
}
```

```java
class BankersAlgorithm {
    private final int n; // Número de processos
    private final int m; // Número de recursos
    private final int[][] max;   // Máxima necessidade de cada processo
    private final int[][] allocated; // Recursos atualmente alocados
    private final int[][] need;  // Necessidade restante
    private final int[] available; // Recursos disponíveis

    public BankersAlgorithm(int[][] max, int[][] allocated, int[] available) {
        this.n = max.length;
        this.m = available.length;
        this.max = max;
        this.allocated = allocated;
        this.available = available;
        this.need = new int[n][m];

        for (int i = 0; i < n; i++)
            for (int j = 0; j < m; j++)
                this.need[i][j] = max[i][j] - allocated[i][j];
    }

    public boolean isSafe() {
        int[] work = Arrays.copyOf(available, m);
        boolean[] finish = new boolean[n];

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (!finish[j]) {
                    boolean canFinish = true;
                    for (int k = 0; k < m; k++) {
                        if (need[j][k] > work[k]) {
                            canFinish = false;
                            break;
                        }
                    }
                    if (canFinish) {
                        finish[j] = true;
                        for (int x = 0; x < m; x++)
                            work[x] += allocated[j][x];
                    }
                }
            }
        }

        for (boolean f : finish) {
            if (!f) return false;
        }
        return true;
    }

    public boolean requestResources(int process, int[] request) {
        for (int i = 0; i < m; i++) {
            if (request[i] > need[process][i] || request[i] > available[i]) {
                return false;
            }
        }

        // Tenta alocar recursos temporariamente
        for (int i = 0; i < m; i++) {
            available[i] -= request[i];
            allocated[process][i] += request[i];
            need[process][i] -= request[i];
        }

        // Verifica se o estado ainda é seguro
        if (!isSafe()) {
            for (int i = 0; i < m; i++) {
                available[i] += request[i];
                allocated[process][i] -= request[i];
                need[process][i] += request[i];
            }
            return false;
        }

        return true;
    }

    public static void main(String[] args) {
        int[][] max = { {7, 5, 3}, {3, 2, 2}, {9, 0, 2}, {2, 2, 2} };
        int[][] allocated = { {0, 1, 0}, {2, 0, 0}, {3, 0, 2}, {2, 1, 1} };
        int[] available = {3, 3, 2};

        BankersAlgorithm ba = new BankersAlgorithm(max, allocated, available);

        System.out.println("Estado seguro inicial: " + ba.isSafe());

        int[] request = {1, 0, 2};
        boolean granted = ba.requestResources(1, request);
        System.out.println("Pedido de recursos " + Arrays.toString(request) + " para P1: " + (granted ? "Concedido" : "Negado"));

        System.out.println("Estado seguro após requisição: " + ba.isSafe());
    }
}
```