MARINHA DO BRASIL

SERVIÇO DE SELEÇÃO DO PESSOAL DA MARINHA

GABARITO DESENVOLVIDO

CP-CEM / 2025 ENGENHARIA DE SISTEMAS DE COMPUTAÇÃO

1ª QUESTÃO (8 pontos)

a) (1 ponto)

O diagrama apresentado é um diagrama de sequência. Esses diagramas modelam as interações entre os objetos em um sistema, enfatizando quando as interações ocorrem, ajudando a mostrar a ordem em que estas ocorrem durante um caso de uso em particular ou em uma instância de caso de uso.

Pode-se dizer também que modela o fluxo de mensagens entre diferentes objetos ou classes no sistema ao longo do tempo, com o objetivo de representar o comportamento dinâmico de um caso de uso específico. Destaca a ordem temporal das interações.

b) (3 pontos)

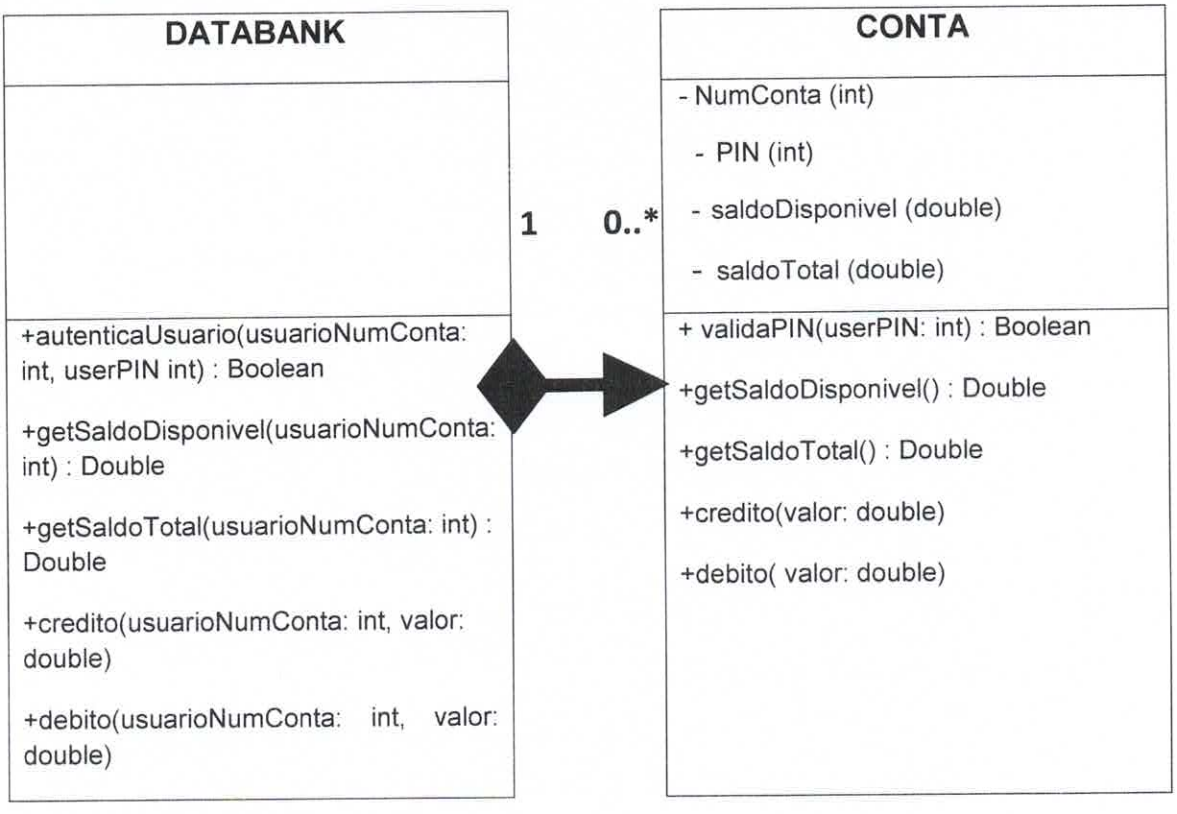
O diagrama representa as interações entre os objetos quando ocorre uma operação de depósito que executa com sucesso. O DEPOSITO primeiro envia uma mensagem displayMSG a TECLADO pedindo para o usuário inserir uma quantia de depósito. Em seguida, DEPOSITO envia uma mensagem lerEntrada a TELA para receber a entrada do usuário. DEPOSITO então instrui o usuário a inserir um envelope de depósito por meio de uma mensagem displayMSG enviada a TELA. DEPOSITO em seguida envia uma mensagem depositoRecebido a DEPOSLOT para confirmar que o envelope de depósito foi recebido e aguarda uma resposta. Por fim, DEPOSITO aumenta o atributo de saldo da conta do usuário, enviando uma mensagem ao DATABANK. DATABANK, então envia a mesma mensagem para CONTA, referente ao usuário.

Passos:

- A classe DEPOSITO envia uma mensagem para TELA, no caso displayMSG (ex: digite valor do depósito);
- A classe DEPOSITO solicita uma entrada do TECLADO, no caso lerEntrada();

- A classe DEPOSITO envia novamente uma mensagem para TELA, no caso displayMSG (ex: insira o envelope);
- DEPÓSITO verifica o *slot* de depósito por meio de DEPOSLOT, no qual verifica se o depósito foi recebido com depositoRecebido().
- DEPOSITO solicita ao DATABANK o crédito do valor na conta com o método credito(numConta, Valor).
- DATABANK realiza o crédito no saldo da conta correspondente por meio do método credito(valor).

c) (4 pontos)



O losango preenchido indica a composição, mostrando que DATABANK é composta por objetos CONTA. A multiplicidade seria 0..*, ou no mínimo 2, já que o *array* exemplo contém dois objetos. Os métodos públicos são representados com o sinal de + e os atributos privados com o sinal de -.

Características dessa composição:

- Todo-parte: DATABANK é o todo e CONTA é a parte.
- Criação: O todo DATABANK cria as partes CONTA.
- Destruição: se o todo for destruído, as partes também serão.
- Exclusividade: As partes pertencem a apenas um todo por vez.

Esse relacionamento modela que as contas bancárias não existem independentemente de um banco/databank que as gerencie.

2ª QUESTÃO (8 pontos)

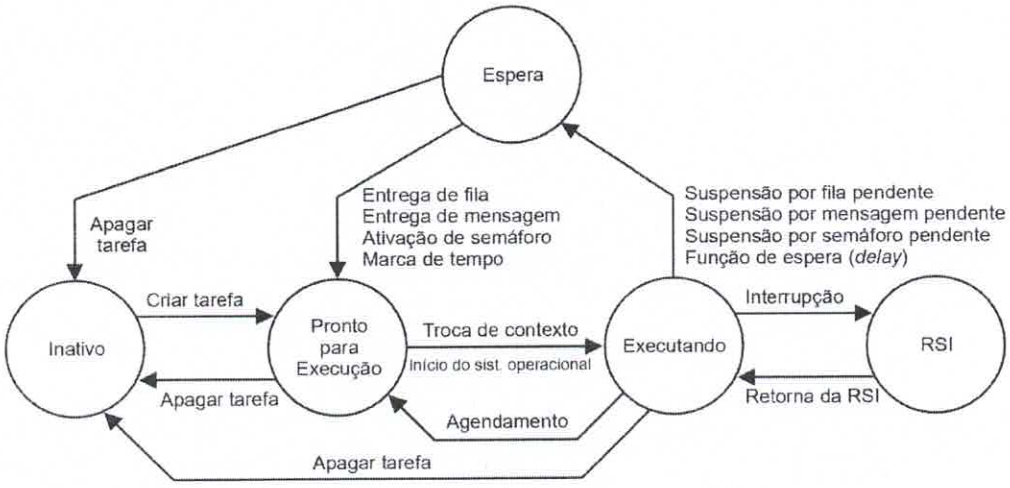
a) (1 ponto)

Uma tarefa tem como atribuição executar uma ação por meio de uma sequência de instruções. Suas principais características são: ser uma entidade que o sistema operacional pode executar concorrentemente e possuir sua própria área de pilha.

As corrotinas, embora similares às tarefas, possuindo prioridades e estados, não possuem suas próprias áreas de pilha, visto que não possuem um contexto próprio.

b) (3 pontos)

Os estados que uma tarefa pode assumir são: Espera, Pronto para Execução, Executando, Interrompido e Inativo. Segue abaixo o diagrama que representa as transições entre os estados.



c) (1 ponto)

Os algoritmos de escalonamento dirigidos por tempo podem utilizar tanto a ordem de chegada das tarefas na fila para suas execuções quanto as estimativas de tempo para execução de cada tarefa a ser executada. Sua principal vantagem é a facilidade de implementação.

Já nos algoritmos dirigidos à prioridade, por outro lado, as prioridades das tarefas são utilizadas para determinar seu grau de importância de execução. A alocação de prioridades pode ser estática ou dinâmica. Uma de suas vantagens é a garantia de execução de tarefas de maior prioridade, mesmo que elas sejam mais longas ou mais recentes.

d) (3 pontos)

O escalonamento por taxa monotônica foi proposto por Liu e Layland, em 1973, para obter a escala de execução de um conjunto de tarefas em um sistema preemptivo dirigido a prioridades fixas. Atualmente, é uma das técnicas de escalonamento mais utilizadas em RTOS comerciais. Sua implementação é simples e seu escalonamento é ótimo no sentido de que, caso seja possível encontrar uma escala de execução para um conjunto de tarefas usando prioridades fixas, nenhuma outra regra ou algoritmo de escalonamento pode definir uma escala que seja admissível para o conjunto tal que a escala encontrada pelo RMS não o seja.

3ª QUESTÃO (8 pontos)

a) (2 pontos)

A Análise subdivide o programa fonte em partes constituintes e impõe uma estrutura gramatical sobre elas. Depois, usa essa estrutura para criar uma representação intermediária do programa fonte. A análise também coleta informações sobre o programa fonte e as armazena em uma estrutura de dados chamada tabela de símbolos, que é passada adiante junto com a representação intermediária para a parte de síntese.

Já a Síntese constrói o programa objeto desejado a partir da representação intermediária e das informações na tabela de símbolos.

b) (2 pontos)

As três etapas presentes na análise são: Léxica, Sintática e Semântica.

A análise léxica é a primeira fase do compilador e é responsável por ler o fluxo de caracteres que compõem o programa fonte e os agrupar em sequências significativas, chamadas lexemas, composto por *tokens*.

A análise sintática utiliza os *tokens* produzidos pelo analisador léxico para criar uma representação intermediária do tipo árvore, que mostra a estrutura gramatical da sequência de *tokens*.

Por fim, a análise semântica utiliza a árvore de sintaxe e as informações na tabela de símbolos para verificar a semântica do programa fonte com a definição da linguagem. Ela também reúne informações sobre tipos e as salva na árvore de sintaxe ou na tabela de símbolos, para o uso subsequente durante a geração de código intermediário.

c) (2 pontos)

As três etapas principais da síntese são: Geração de Código Intermediário, Otimização de Código e Geração de Código.

A geração de código intermediário é utilizada no processo de tradução do programa fonte para um código objeto. Após a passagem por essa etapa, representações intermediárias explícitas de baixo nível são geradas.

A otimização de código tem como objetivo transformar o código intermediário visando produzir um código objeto melhor. A otimização pode ser em relação a tempo de execução, menor código objeto, menos consumo de energia, entre outros.

Por fim, a geração de código recebe como entrada uma representação intermediária do programa fonte e o mapeia em uma linguagem objeto.

d) (2 pontos)

O compilador gera um programa objeto que pode ser um programa em linguagem de máquina executável, podendo ser executado pelo usuário para processar entradas e produzir saídas.

Já o interpretador, ao invés de produzir um programa objeto como resultado da tradução, executa diretamente as operações especificadas no programa fonte sobre as entradas fornecidas pelo usuário.

O programa objeto em linguagem de máquina produzido por um compilador normalmente é muito mais rápido no mapeamento de entradas para saída do que um interpretador. Exemplos de linguagem compiladas são C, C++, GoLang, entre outras. Já para interpretadas, podem ser citadas Java, Phyton, Shell Script, entre outras.

4ª QUESTÃO (8 pontos)

a) (2 pontos)

```
struct no {
	int chave;
	no* anterior;
	no* próximo;
}

classe Fila {
	private:
		no* primeiro;
		no* ultimo;

	public:
		Fila () {
			ultimo = primeiro = nullptr;
		}

		void inserir (int chave) {
			no* novo = new no();
			novo->chave = chave;
			novo->anterior = ultimo;
			novo->proximo = nullptr;

			if (primeiro == nullptr) {
				primeiro = novo;
			} else {
				ultimo->proximo = novo;
				ultimo = novo;
			}
		}

		bool buscar(int chave) {
			no* atual = primeiro;

			while (atual != nullptr) {
				if (atual->chave == chave) {
					return true;
				}
				atual = atual->proximo;
			}

			return false;
		}
}
```

b) (1 ponto)

```
bool remove (int chave) {
	if (primeiro == nullptr) {
		return false;
	}
```

```
        no* atual = primeiro;

        do {
                if (atual->chave == chave) {
                        if (atual->anterior == nullptr) {
                                primeiro = atual->proximo;
                                delete atual;
                        } else {
                                atual->anterior->proximo = atual->proximo;
                                if (atual->proximo != nullptr) {
                                        atual->próximo->anterior = atual->anterior;
                                }
                                delete atual;
                        }

                        return true;
                }
                atual = atual->proximo;
        } while (atual != nullptr);

        return false;
}
```

c) (2 pontos)

```
class No {
        int chave;
        No esquerda, direita;

        public No (int chave) {
                this.chave = chave;
                esquerda = direita = null;
        }
}

public class Arvore {
        No raiz;

        Public Arvore () {
                raiz = null;
        }

        public void inserir (int chave) {
                raiz = inserirRec(raiz, chave);
        }

        private No inserirRec (No raiz, int chave) {
                if (raiz == null) {
                        return new No(chave);
                }
```

```
            if (chave < raiz.chave) {
                raiz.esquerda = inserirRec(raiz.esquerda, chave);
            } else if (chave > raiz.chave) {
                raiz.direita = inserirRec(raiz.direita, chave);
            }

            return raiz;
        }

        private bool buscar (int chave) {
            No temp = raiz;
            while (temp != null) {
                if (temp.chave == chave) {
                    return true;
                } else if (temp.chave > chave) {
                    temp = temp.esquerda;
                } else {
                    temp = temp.direita;
                }
            }

            return false;
        }
}
```

d) (1 ponto)

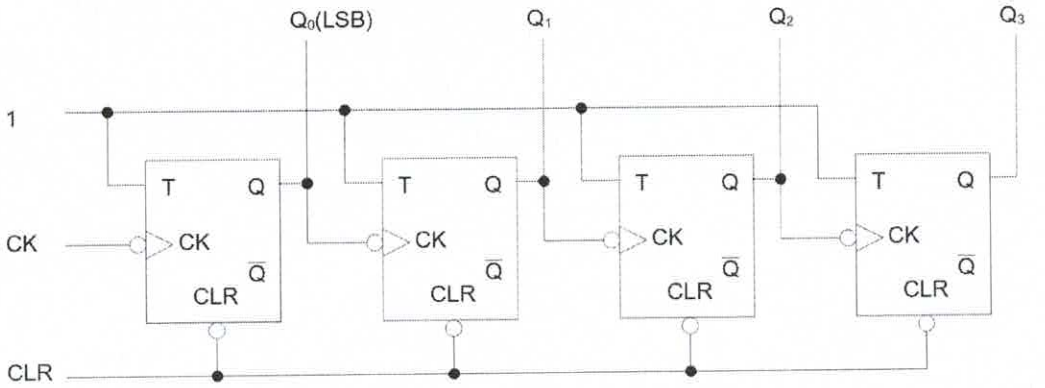
```
public void imprimirOrdenado () {
        imprimirOrdenadoRec (raiz);
}

private void imprimirOrdenadoRec (No raiz) {
        if (raiz != null) {
            imprimirOrdenadoRec (raiz.esquerda);
            System.out.println(raiz.chave + " ");
            imprimirOrdenadoRec (raiz.direita);
        }
}
```

e) (2 pontos)

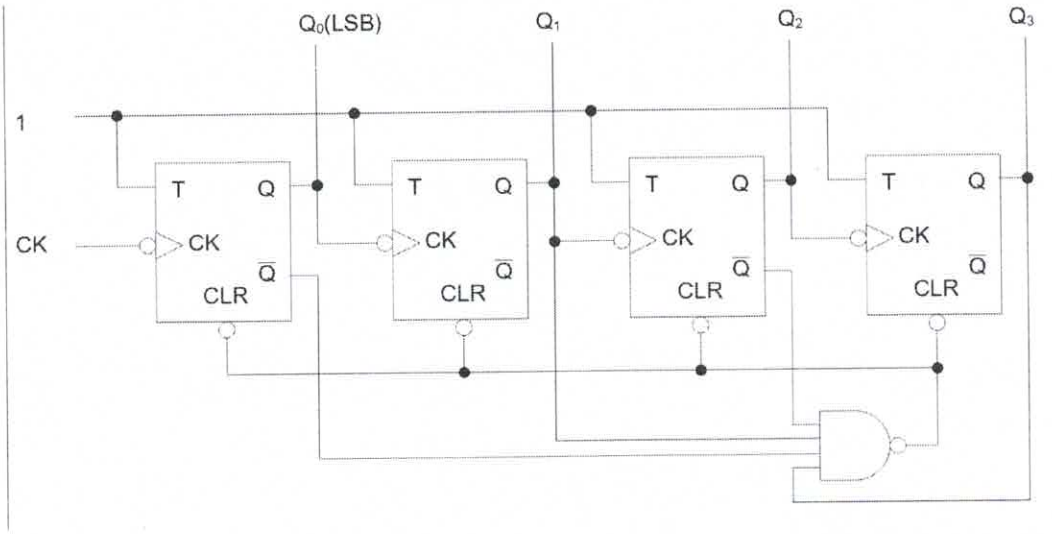
Para o caso da lista duplamente ligada, a complexidade de tempo da busca é de $\Theta(n)$, onde n é o número de nós da lista, visto que a busca pode ter que percorrer a lista inteira. Já no caso da árvore binária, a complexidade de tempo da busca é de $\Theta(h)$, onde h é a altura da árvore. Numa árvore binária totalmente desbalanceada, $h = n$, portanto, a complexidade de tempo é $\Theta(n)$. Já no caso de uma árvore binária completamente balanceada, $h = log_2 n$, portanto, a complexidade de tempo da busca é $\Theta(\log_2 n)$.

5ª QUESTÃO (8 pontos)

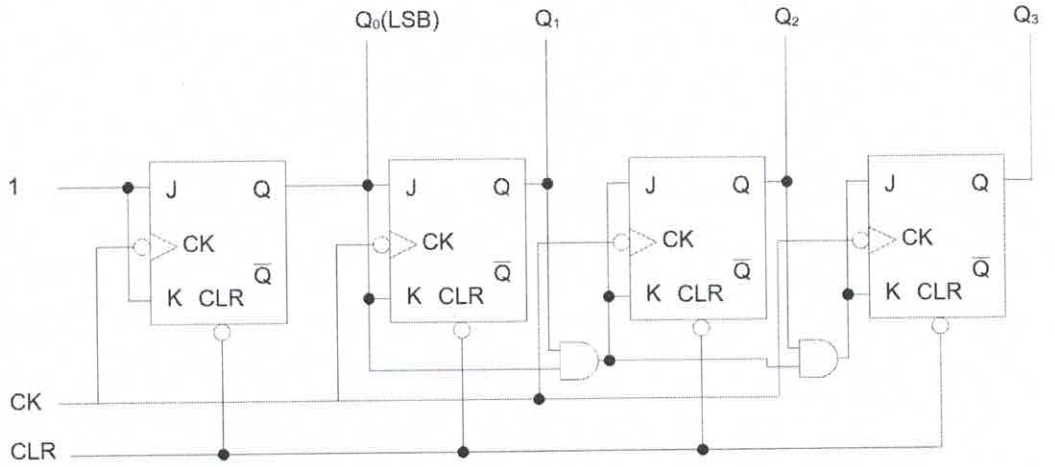
a) (3 pontos)



b) (2 pontos)



c) (3 pontos)



6ª QUESTÃO (8 pontos)

a) (2 pontos)

Geralmente, as modernas redes de computadores utilizam algoritmos de roteamento dinâmicos mais complexos que o roteamento estático, porém mais eficientes, pois são capazes de encontrar os caminhos mais curtos de acordo com a topologia atual da rede. Dois algoritmos de roteamento dinâmico, o roteamento por vetor de distância e o roteamento de estado de enlace, são os mais conhecidos.

Os algoritmos de roteamento por vetor de distância operam fazendo cada roteador manter uma tabela que fornece a melhor distância conhecida até cada destino e determina qual enlace deve ser utilizado para chegar lá. Neste tipo de implementação, cada roteador mantém uma tabela de roteamento para cada roteador da rede, contendo uma interface de saída preferencial a ser utilizada para o destino e uma estimativa da distância até esse destino.

Os algoritmos de roteamento de estado de enlace operam basicamente em cinco passos: descobrir seus vizinhos e aprender seus endereços de rede, medir a distância ou curso até cada um de seus vizinhos, criar um pacote que informe o que foi aprendido e distribuí-lo para todos os roteadores, receber esses mesmos pacotes dos outros roteadores e calcular o caminho mais curto até cada um dos outros roteadores.

O roteamento por vetor de distância foi utilizado na ARPANET até 1979, quando foi substituído pelo roteamento de estado de enlace. O principal problema que causou sua retirada foi que o algoritmo geralmente levava muito tempo para convergir. A convergência numa rede significa o tempo para o estabelecimento de rotas para os melhores caminhos pela rede.

Diante do exposto, a opção que se mostra mais adequada para a situação apresentada é o roteamento de estado de enlace.

b) (2 pontos)

Para gerarmos a tabela de roteamento, precisamos calcular o caminho mais curto entre o nó indicado, no caso D, utilizando-o como origem e calculando o caminho mais curto deste para todos os outros nós. A tabela de roteamento indicará o próximo nó no caminho mais curto e o custo total para se alcançar o destino.

Calculando o caminho mais curto a partir do nó D:

D -> C (11)

D -> G (3)

D-> C -> B (11+3=14)

D -> C -> F (11+2=13)

D -> G -> F (3+5 = 8)

D -> C -> B -> E (14 + 6 = 20)

D -> G -> F -> B (8 +8 = 16). Aqui observe que o caminho mais curto para B é na verdade D -> C -> B (14).

D -> C -> B -> E -> A (20 + 9 = 29)

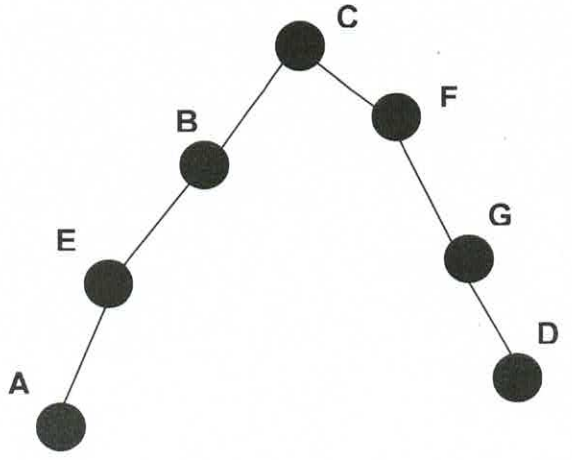
Montando a Tabela de roteamento do roteador D:

Destino (A partir do nó D)	Próximo Salto (A partir do nó)	Custo
A	C	29
B	C	14
C	C	11
E	C	20
F	G	8
G	G	3

A tabela apresentada fornece, a partir do nó de origem, o próximo nó no caminho mais curto para cada destino e o custo total desse caminho.

c) (2 pontos)

Para criar uma árvore de escoamento a partir do nó C, pode-se utilizar o algoritmo de Dijkstra para encontrar o caminho mais curto de C para todos os outros nós do grafo. A árvore de escoamento mostrará o caminho mais curto do nó raiz C para cada um dos outros nós. Essa árvore é construída rastreando os pais de cada nó no caminho mais curto a partir de C.



d) (2 pontos)

A sequência de nós percorridos utilizando o algoritmo de caminho mais curto é G -> F -> B -> E -> A . Custo 28 (5 + 8+ 6 + 9 = 28)

7ª QUESTÃO (8 pontos)

a) (4 pontos)

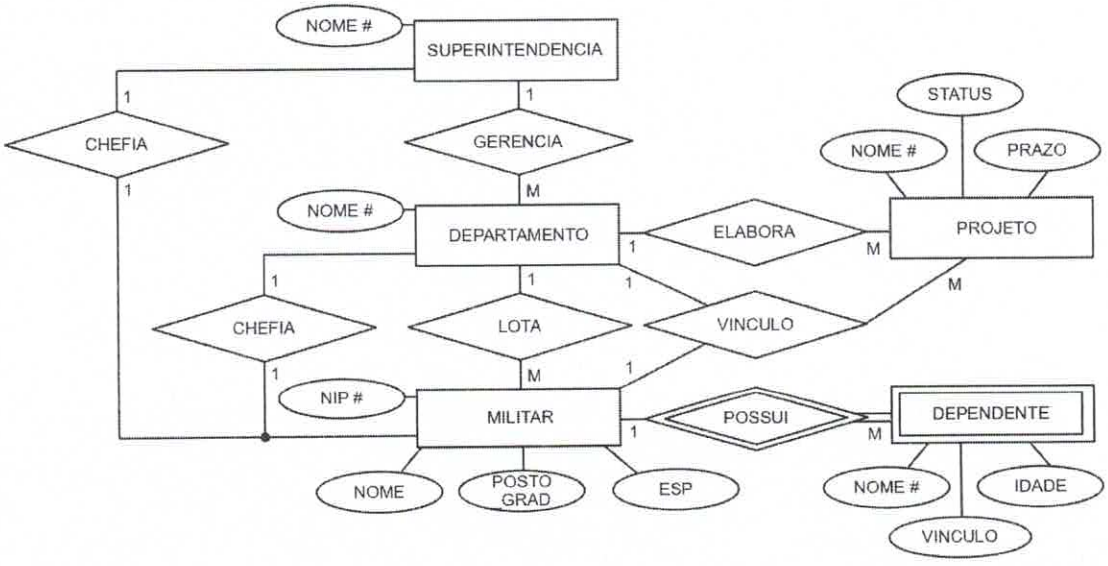
```
CREATE TABLE MILITAR
        (NIP                INTEGER,
         NOME               VARCHAR,
         POSTO_GRAD         VARCHAR,
         ESP                VARCHAR,
        PRYMARY KEY (NIP));

CREATE TABLE SUPERINTENDENCIA
        (NOME               VARCHAR,
         NIP_CHEFE          INTEGER,
        PRYMARY KEY (NOME)),
        FOREIGN KEY (NIP_CHEFE) REFERENCES MILITAR;

CREATE TABLE DEPARTAMENTO
        (NOME               VARCHAR,
         NOME_SUP           VARCHAR,
         NIP_CHEFE          INTEGER,
        PRYMARY KEY (NOME)),
        FOREIGN KEY (NOME_SUP) REFERENCES SUPERINTENDENCIA,
        FOREIGN KEY (NIP_CHEFE) REFERENCES MILITAR;
```

```
CREATE TABLE PROJETO
        (NOME              VARCHAR,
         NOME_DEP          VARCHAR,
         STATUS            VARCHAR,
         PRAZO             DATE,
        PRYMARY KEY (NOME)),
        FOREIGN KEY (NOME_ DEP) REFERENCES DEPARTAMENTO;


CREATE TABLE DEPENDENTE
        (NOME              VARCHAR,
         NIP_MIL           INTEGER,
         IDADE             INTEGER,
         VINCULO           VARCHAR,
        PRYMARY KEY (NOME, NIP_MIL),
        FOREIGN KEY (NIP_MIL) REFERENCES MILITAR;
```

b) (2 pontos)

```
DELETE FROM DEPENDENTE

WHERE IDADE = (

                SELECT MAX (IDADE)

                FROM DEPENDENTE

                WHERE NIP_MIL = 11111111

        ) AND NIP_MIL = 11111111;
```

c) (2 pontos)



8ª QUESTÃO (8 pontos)

a) (2 pontos)

Ao se verificar o extrato do arquivo /etc/passwd, é possível verificar que a linha referente ao usuário "ofseq" encontra-se comentada. O comentário é representado pelo símbolo #. A solução indicada para a resolução do problema é editar o arquivo utilizando um editor de texto, descomentando a linha referente ao usuário "ofseq" e salvá-lo. Após isso, reiniciar o sistema.

b) (1 ponto)

A partir do diretório raiz do sistema, executar nano ou Vi /etc/passwd para abrir o arquivo.

Realizar a remoção do comentário, navegando até a linha comentada.

Salvar o arquivo utilizando "CTRL+O", no caso do nano, e wq, no caso do Vi.

c) (2 pontos)

Ao se analisar a saída do comando ls –la apresentada, verifica-se que o arquivo REL-SVC.txt encontra-se com as permissões de leitura, escrita e execução negadas para o dono do arquivo bem como para o grupo ao qual o arquivo pertence. Nesse caso, devem-se alterar as permissões do arquivo de forma que o usuário dono, no caso ofseq, e o grupo dono OSE possam realizar a manipulação do arquivo.

d) (2 pontos)

A partir do diretório do arquivo, realizar a alteração das permissões do arquivo utilizando o comando chmod.

Numérico: chmod 777 REL-SVC.txt

Simbólico: chmod u+rwx,g+rwx,o+rwx REL-SVC.txt

e) (1 ponto)

De forma a evitar a violação e a corrupção de dados do arquivo por usuários externos, as permissões do grupo "outros" devem ser retiradas totalmente. Mesmo a permissão de leitura permite que outros usuários consigam ter acesso ao arquivo, o que não é o desejado.

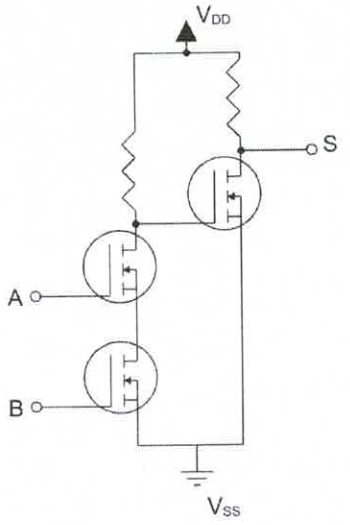
Remoção total de acesso:

Simbólico: chmod u+rwx,g+rwx,o-rwx REL-SVC.txt

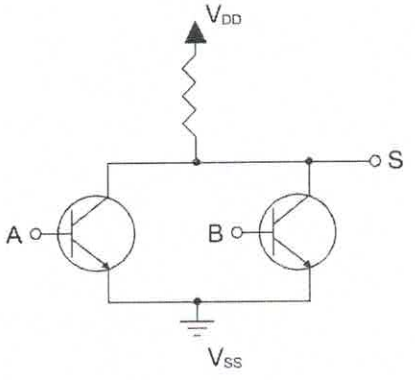
Numérico: chmod 770 REL-SVC.txt

9ª QUESTÃO (8 pontos)

a) (2 pontos)



b) (2 pontos)



c) (1 ponto)

Num sistema de malha aberta, o sinal de saída não afeta a ação do controle, de forma que o sinal de saída não é medido nem enviado de volta para a comparação com o sinal de entrada.

Já num sistema de malha fechada, o sinal atuante de erro, que é a diferença entre o sinal de entrada e o sinal de saída, excita o controlador de modo a reduzir o erro e trazer o valor do sinal de saída para o valor desejado.

D) (3 pontos)

Nomeando o bloco superior como G(s) e o inferior como H(s), temos:

$$G(s)=\frac{s+3}{s^2+4s+4}=\frac{A(s)}{B(s)} \quad e \quad H(s)=\frac{s}{s^2-9}=\frac{C(s)}{D(s)}$$

A função de transferência do sistema realimentado negativamente é dada por:

$$\frac{C(s)}{R(s)}=\frac{G(s)}{1+G(s)H(s)}=\frac{\frac{A(s)}{B(s)}}{1+\frac{A(s)}{B(s)}\frac{C(s)}{D(s)}}=\frac{\frac{A(s)}{B(s)}}{\frac{B(s)D(s)+A(s)C(s)}{B(s)D(s)}}=\frac{A(s)}{B(s)}\frac{B(s)D(s)}{B(s)D(s)+A(s)C(s)}=\frac{A(s)D(s)}{B(s)D(s)+A(s)C(s)}$$

Substituindo:

$$\frac{C(s)}{R(s)}=\frac{(s+3)(s^2-9)}{(s^2+4s+4)(s^2-9)+(s+3)s}=\frac{(s+3)(s^2-9)}{(s^2+4s+4)(s+3)(s-3)+(s+3)s}=\frac{s^2-9}{(s^2+4s+4)(s-3)+s}=$$

$$\frac{C(s)}{R(s)}=\frac{s^2-9}{s^3+4s^2+4s-3s^2-12s-12+s}=\frac{s^2-9}{s^3+s^2-7s-12}$$

Portanto, a Função de Transferência do sistema é dada por:

$$\frac{C(s)}{R(s)}=\frac{s^2-9}{s^3+s^2-7s-12}$$

10ª QUESTÃO (8 pontos)

a) (2 pontos)

Considerando que o servidor possui duas interfaces de rede, uma delas deverá estar conectada à rede interna e a outra à rede externa. Neste caso, eth0 será conectada à rede interna e eth1, à rede externa. A eth0 atuará como o *gateway* para a rede interna. Na configuração, deverão ser atribuídos IPs estáticos a ambas as interfaces. O *gateway* da interface eth1 será o endereço IP do roteador A. O arquivo de configuração do servidor DHCP deverá ser configurado e o encaminhamento de pacotes automático ativado no kernel do linux. Caso haja firewall, este deverá ser configurado de forma a permitir o encaminhamento de pacotes entre as interfaces.

Passos:

1. Configurar os IPs estáticos nas interfaces eth0 e eth1 e ativar as interfaces.
2. Configurar o arquivo de configuração do serviço DHCP, incluindo as rotas a serem utilizadas.
3. Habilitar o encaminhamento de pacotes automático no kernel do linux.
4. Configurar o *firewall* para permitir o tráfego entre as interfaces.

b) (2 pontos)

Considerando que eth0 se comunica com a rede interna e a eth1 com a rede externa, os IPs escolhidos para a eth0 e eth1 foram, respectivamente, 192.168.1.200 e 10.0.0.10, haja vista a necessidade de que cada interface deve possuir um IP pertencente à rede com a qual deseja se comunicar.

Para atribuição dos endereços de rede, a interface de rede, usam-se os comandos:

```
sudo ifconfig eth0 192.168.1.200 netmask 255.255.255.0 up
sudo ifconfig eth1 10.0.0.10 netmask 255.255.255.0 up
```

Para o acesso à rede externa, um *gateway* deverá ser configurado utilizando o comando route na interface de acesso à rede externa:

```
sudo route add default gw 10.0.0.1 eth1
```

Para garantir que as configurações se tornarão persistentes, deve-se realizar a alteração do arquivo /etc/network/interfaces adicionando um bloco de configuração para cada interface, como mostrado a seguir:

```
auto eth0
iface eth0 inet static
address 192.168.1.200
netmask 255.255.255.0
auto eth1
iface eth1 inet static
address 10.0.0.10
netmask 255.255.255.0
gateway 10.0.0.1
```

c) (2 pontos)

Para a configuração do serviço DHCP no SERVDOR DHCP, deveremos alterar o arquivo de configuração /etc/dhcp/dhcp.conf e inserir a configuração necessária para a atribuição de endereços dinâmicos aos *hosts* da rede.

```
subnet 192.168.1.0 netmask 255.255.255.0 {
    range 192.168.1.30 192.168.180;
    option routers 192.168.1.200;
    option subnet-mask 255.255.255.0;
    option broadcast-address 192.168.1.255;
}
```

No arquivo de configuração do SERVIDOR DHCP, o serviço deverá ser configurado para escutar na interface eth0.

Dependendo da distribuição, o arquivo de configuração pode mudar. Exemplos:

Caso o arquivo seja /etc/default/isc-dhcp-server, a linha a ser alterada é a INTERFACESv4="" para INTERFACESv4="eth0".

Caso o arquivo seja o /etc/default/dhcp3-server, a linha a ser alterada é a INTERFACES="" para INTERFACES="eth0".

Para ativar o encaminhamento automático de pacotes IP entre as interfaces de rede, poderá ser dado o comando:

sudo sysctl –w net.ipv4.ip_forward=1

que ativará o encaminhamento imediatamente. Entretanto, para fazê-lo de forma persistente, no arquivo de configuração do sysctl, geralmente /etc/sysctl.conf, ou algum arquivo dentro de /etc/sysctl.d/, deverá ser adicionada ou descomentada a linha:

net.ipv4.ip_forward=1

O arquivo deverá ser salvo e as alterações aplicadas com:

sudo sysctl –p

Para permitir o tráfego entre as interfaces, o *firewall* deverá ser configurado de forma a permitir o encaminhamento

sudo iptables –P FORWARD ACCEPT

sudo iptables –A FORWARD –i eth0 –o eth1 –j ACCEPT

sudo iptables –A FORWARD –i eth1 –o eth0 –m state –state RELATED,STABLISHED –j ACCEPT

Para permitir que os clientes da rede interna acessem a rede externa através da interface eth1, deverá ser usado um mascaramento de IP utilizando o protocolo NAT com o comando:

sudo iptables -t nat –A POSTROUTING –o eth1 –j MASQUERADE

As regras deverão ser salvas. Um exemplo pode ser os comandos abaixo, mas que varia conforme a distribuição :

sudo apt install iptables-persistent

sudo netfilter-persistent save

d) (2 pontos)

No arquivo /etc/dhcp/dhcp.conf, a configuração abaixo deverá ser inserida e salva.

```
host SERVIDOR1 {
    hardware ethernet 08:00:07:26:c0:a5;
    fixed-address 192.168.1.1;
}

host IMPRESSORA1 {
    hardware ethernet AA:BB:CC:DD:EE:FF;
    fixed-address 192.168.1.21;
}

host TELEFONE1 {
    hardware ethernet 00:11:22:33:44:55;
    fixed-address 192.168.1.22;
}
```