MINISTÉRIO DA EDUCAÇÃO

INSTITUTO FEDERAL DO ESPÍRITO SANTO

REITORIA

Avenida Rio Branco, 50 – Santa Lúcia – 29056-255 – Vitória – ES

27 3357-7500

CONCURSO PÚBLICO
EDITAL Nº 03 / 2015

Professor do Magistério do Ensino Básico, Técnico e Tecnológico

ÍNDICE DE INSCRIÇÃO	332
CAMPUS	Serra
ÁREA/SUBÁREA	Sistemas de Computação II

PROVA DE CONHECIMENTOS ESPECÍFICOS | DISCURSIVA
MATRIZ DE CORREÇÃO

QUESTÃO 01
Deve descrever no mínimo: - arquitetura Harvard e tecnologia RISC, pois utiliza barramento de dados diferente do barramento de programa, com isto é possível ter um número reduzido de instruções simples - barramento de programa é de 14bits - barramento de dados é de 1byte (8bits), ou seja, os dados utilizados são de 8bits - possuí três timers, conversor A/D de 10bits, comparador e porta serial - três conjuntos de entrada/saídas: PORTA, PORTB e PORTC - clock externo

QUESTÃO 02

```c
int *cria_campo_minado(int tamLinhas, int tamColunas, int qtdBombas)
{
	int i, j, iniL, fimL, iniC, fimC, x, y, soma;

	// verificando os valores passados: pode ser com assert ou com if
	assert(tamLinhas>0);
	assert(tamColunas>0);
	assert(qtdBombas>0);
	// if ((tamLinhas<=0)||(tamColunas<=0)||(qtdBombas<=0)) return NULL;

	// aloca a matriz
	int *campo = (int *)malloc(tamLinhas*tamColunas*sizeof(int));
	assert (campo!=NULL);
	//if (campo == NULL) return NULL;

	// inicializa o campo: memset ou percorrer um a um
	memset (campo, 0, (tamLinhas*tamColunas*sizeof(int)));
	//for (i = 0; i < tamLinhas; i++)
	//    for (j = 0; j < tamColunas; j++)
	//         campo[i*tamColunas + j] = 0;

	// bombas em posições aleatórias
	srand(time(NULL));
	for (i = 0; i < qtdBombas; i++)
		campo[(rand()%tamLinhas)*tamColunas + (rand()%tamColunas)] = -1;

	// calculando a quantidade de bombas do vizinho
	for (i = 0; i < tamLinhas; i++)
	{
		for (j = 0; j < tamColunas; j++)
		{
			if (campo[i*tamColunas + j] == 0)
			{
				iniL = (i==0)? i: (i-1);
				fimL = (i==(tamLinhas - 1))? i: (i+1);
				iniC = (j==0)? j: (j-1);
				fimC = (j==(tamColunas - 1))? j: (j+1);
				soma = 0;

				for (x = iniL; x <= fimL; x++)
					for (y = iniC; y <= fimC; y++)
						if (campo[x*tamColunas + y] == -1) soma++;


				campo[i*tamColunas + j] = soma;
			}
		}
	}
	return campo;
}
```

* Não está sendo verificado o limite superior da quantidade de bombas, pois não foi limitada no enunciado. Embora fosse possível se avaliar se cabem no jogo, como algo do tipo "assert(qtdBombas <(tamColunas*tamLinhas));", não consta da matriz de correção e não será avaliado.
* É possível que bombas sejam colocadas na mesma posição e o jogo tenha uma quantidade menor de bombas, mas o tratamento deste caso específico não será avaliado.

QUESTÃO 03

Questão datagrama IP
VERS: versão do protocolo IP que foi usada para criar o datagrama
HLEN: comprimento do cabeçalho
SERVICE-TYPE: especifica como o datagrama poderia ser manuseado assim que circularem pela rede
TOTAL-LENGTH: define o tamanho do datagrama
IDENTIFICATION: usado principalmente para identificar fragmentos identificativos do datagrama IP original
FLAGS: controla a fragmentação
FRAGMENT OFFSET: especifica o início do datagrama original dos dados que estão sendo transportados no fragmento
TIME TO LIVE: é o tempo de vida do pacote. Evita o congestionamento da rede pelos datagramas perdidos
PROTOCOL: especifica qual protocolo foi usado para criar a mensagem que está sendo transportada na área de dados do datagrama
HEADER-CHECKSUM: é responsável por detectar inconsistência no datagrama IP
SOURCE IP ADDRESS: especifica o endereço IP de origem
DESTINATION IP ADDRESS: especifica o endereço IP de destino
IP OPTIONS: é um campo com informações adicionais para o protocolo IP
PADDING: garante que o cabeçalho tem um tamanho múltiplo de 4 bytes
DATA: contém o seguimento da camada de transporte

QUESTÃO 04

A.

Setores	Rede	Endereço *broadcast*
Laboratórios	200.30.20.0/25	200.30.20.127
Professores	200.30.20.128/26	200.30.20.191
Administração	200.30.20.192/27	200.30.20.223
Direção ou Coordenadorias	200.30.20.224/28	200.30.20.239
Coordenadorias ou Direção	200.30.20.240/28	200.30.20.255

B.
I – Endereço IP é um número (composto por 32 bits se IPv4 ou 128 bits se IPv6), com o objetivo de interligar diversos dispositivos em rede. Endereço IP, de forma genérica, é uma identificação de um dispositivo (computador, impressora, etc) em uma rede local ou pública, que utiliza protocolo IP (Internet Protocol).

II – As classes são A, B, C, D e E e se diferenciam pelo número de bytes que representa a rede (NETID). Praticamente as redes A, B e C são as mais utilizadas.

III – Podem ser: estáticos, os números IP são atribuídos manualmente pelo usuário; dinâmicos que é atribuído por alguma interface (por exemplo um servidor DHPC).

QUESTÃO 05

Nos itens abaixo será utilizada uma tabela com uma linha para cada um dos três *frames* de memória e ainda uma linha adicional para indicar a evolução do número de *page faults*; nas colunas serão indicadas as requisições de páginas de acordo com o enunciado da questão.

Item A.I

Para o algoritmo LRU (*Least Recently Used*) temos a seguinte sequência de requisições e substituições de páginas:

	7	2	3	1	2	5	3	4	6	7	7	1	0	5	4	6	2	3	0	1
Frame 1	7	7	7	1	1	1	3	3	3	7	7	7	7	5	5	5	2	2	2	1
Frame 2	–	2	2	2	2	2	2	4	4	4	4	1	1	1	4	4	4	3	3	3
Frame 3	–	–	3	3	3	5	5	5	6	6	6	6	0	0	0	6	6	6	0	0
Page Faults	1	2	3	4	4	5	6	7	8	9	9	10	11	12	13	14	15	16	17	18

Assim, pelo algoritmo LRU ocorrerão 18 *page faults*.

Item A.II

Para o algoritmo FIFO (*First In, First Out*) temos a seguinte sequência de requisições e substituições de páginas:

	7	2	3	1	2	5	3	4	6	7	7	1	0	5	4	6	2	3	0	1
Frame 1	7	7	7	1	1	1	1	1	6	6	6	6	0	0	0	6	6	6	0	0
Frame 2	–	2	2	2	2	5	5	5	5	7	7	7	7	5	5	5	2	2	2	1
Frame 3	–	–	3	3	3	3	3	4	4	4	4	1	1	1	4	4	4	3	3	3
Page Faults	1	2	3	4	4	5	5	6	7	8	8	9	10	11	12	13	14	15	16	17

Assim, pelo algoritmo FIFO ocorrerão 17 *page faults*.

Item A.III

Para o *algoritmo ótimo* (um algoritmo preditivo hipotético que gera o menor número de *page faults* possível, ou em outras palavras, um algoritmo que sempre remove a página que não será utilizada pelo maior período de tempo) temos a seguinte sequência de requisições e substituições de páginas:

	7	2	3	1	2	5	3	4	6	7	7	1	0	5	4	6	2	3	0	1
Frame 1	7	7	7	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Frame 2	–	2	2	2	2	5	5	5	5	5	5	5	5	5	4	6	2	3	3	3
Frame 3	–	–	3	3	3	3	3	4	6	7	7	7	0	0	0	0	0	0	0	0
Page Faults	1	2	3	4	4	5	5	6	7	8	8	8	9	9	10	11	12	13	13	13

Assim, pelo algoritmo ótimo ocorrerão 13 *page faults*.

Item B.I

Instalar uma CPU mais rápida – não aumentaria o grau de utilização da CPU. Pelo contrário, como a CPU fica a maior parte do tempo esperando por dispositivos de I/O, provavelmente uma CPU mais rápida diminuiria o grau de utilização.

Item B.II

Aumentar o grau de multiprogramação – não aumentaria o grau de utilização da CPU. De fato, como um grau de multiprogramação maior, em geral, significa a ocorrência maior de paginação, provavelmente o grau de utilização da CPU diminuiria.

Item B.III

Aumentar a memória principal – provavelmente aumentaria o grau de utilização da CPU pois permitiria que mais páginas fossem mantidas em memória física diminuindo a quantidade de paginação necessária e consequentemente diminuindo o tempo de espera por I/O.

Item B.IV

Instalar um HD mais rápido ou múltiplos controladores com múltiplos HD's – provavelmente aumentaria o grau de utilização da CPU pois diminuiria o gargalo na utilização dos discos fazendo com que a CPU receba mais dados e mais rapidamente.

Item B.V

Aumentar o tamanho da página – Resultaria em menos *page faults* se os dados estiverem sendo acessados sequencialmente, entretanto, se o acesso aos dados for aproximadamente randômico, isso pode aumentar o número de *page faults* pois um número menor de páginas poderia ser mantido na memória simultaneamente e a cada *page fault* uma quantidade maior de dados teria que ser transferida de e para os discos rígidos. Então esta alteração tem tanta chance de aumentar o grau de utilização quanto de diminuí-lo.

_____ _____ _____ / _____ / 2015

Assinatura Presidente Assinatura Membro