Arquitectura de Computadores

Licenciatura em Engenharia Informática

Teste 3 (A) – 2010/05/31 – Duração: 1h45m

Nome:_____ Número:_____

Teste sem consulta.
A interpretação do enunciado faz parte da avaliação. Explicite nas suas respostas todas as hipóteses assumidas.
Por favor, tente ser conciso nas sua respostas para que estas caibam na zona delimitada.

Escolha Múltipla

Indique o número da resposta que achar correcta no quadrado definido para o efeito. Cada resposta errada desconta 20% da cotação da pergunta. A cotação mínima nesta parte é 0.0 valores.

Q-1 [0.5 val.] Suponha que um dado CPU emite endereços virtuais com 36 bits e que a unidade de transformação de endereços gere a memória física dividindo-a em páginas de 8 KBytes.

1. O número de bits usado para identificar o número da página virtual é 23 e o número de bits usado para o deslocamento dentro da página é 13.
2. O número de bits usado para identificar o número da página virtual é 22 e o número de bits usado para o deslocamento dentro da página é 12.
3. O número de bits usado para identificar o número da página virtual é 12 e o número de bits usado para o deslocamento dentro da página é 24.
4. O número de bits usado para identificar o número da página virtual é 20 e o número de bits usado para o deslocamento dentro da página é 16.

Opção correcta ☐

Q-2 [0.5 val.] Suponha que um dado CPU emite endereços virtuais com 36 bits e que a unidade de transformação de endereços gere a memória física dividindo-a em páginas de 8 KBytes. Admitindo que cada entrada da tabela de páginas ocupa 4 bytes, a tabela de páginas de um processo ocupará:

1. 2^{23} bytes
2. 2^{24} bytes
3. 2^{25} bytes
4. 2^{26} bytes

Opção correcta ☐

Q-3 [0.5 val.] Num dado CPU, a unidade de transformação de endereços (MMU) tem um registo base e um registo limite. Neste hardware, é executado um sistema operativo que suporta multiprogramação. Usando estes dois registos, o sistema operativo:

1. assegura a recolocação dos programas, mas não impede que um processo faça acesso ao espaço de endereçamento de outro processo.
2. não assegura a recolocação dos programas, mas impede que um processo faça acesso ao espaço de endereçamento de outro processo.
3. assegura a recolocação dos programas, e impede que um processo faça acesso ao espaço de endereçamento de outro processo.
4. não assegura a recolocação dos programas, nem impede que um processo faça acesso ao espaço de endereçamento de outro processo.

Opção correcta ☐

Q-4 [0.5 val.] Quando a memória física é gerida por partições dinâmicas, as partições são criadas com tamanho igual ao da imagem do processo que se pretende carregar e são destruídas quando o programa termina. Nestas condições,

1. existe fragmentação interna, mas não existe fragmentação externa.
2. não existe fragmentação interna, mas existe fragmentação externa.
3. não existe fragmentação interna nem fragmentação externa.
4. existe fragmentação interna e fragmentação externa.

Opção correcta ☐

Q-5 [0.5 val.] Quando a memória física é gerida por páginas

1. a fragmentação interna é pouco grave; a memória física livre não está contígua e esta situação causa problemas.
2. a fragmentação interna é pouco grave; a memória física livre não está contígua, mas isto não é importante porque as páginas virtuais de um processo não precisam de ficar contíguas na memória física.
3. a fragmentação interna é um problema grave.
4. a fragmentação externa é um problema grave, porque a memória física livre não está contígua.

Opção correcta ☐

Q-6 [0.5 val.] Quando a linha de interrupção do CPU é accionada por um controlador, o CPU

1. salta imediatamente para a rotina de tratamento da interrupção.
2. acaba a instrução corrente e salta para a rotina de tratamento da interrupção.
3. salta para a rotina de tratamento da interrupção, mas só se as interrupções estiverem desinibidas.
4. acaba a instrução corrente e salta para a rotina de tratamento da interrupção, mas só se as interrupções estiverem desinibidas.

Opção correcta ☐

Q-7 [0.5 val.] Quando uma rotina de tratamento de interrupção está em execução,

1. ela pode ser sempre interrompida por outra rotina de tratamento de interrupção
2. ela não pode ser interrompida por outra rotina de tratamento de interrupção
3. ela pode ser interrompida por outra rotina de tratamento de interrupção, se o sistema de interrupções não estiver inibido
4. ela pode ser interrompida por outra rotina de tratamento de interrupção, se o periférico que provocou a nova interrupção for muito rápido.

Opção correcta ☐

Q-8 [0.5 val.] No Pentium a instrução máquina *iret* (interrupt return)

1. restaura o estado da computação que decorria quando ocorreu a interrupção.
2. restaura o *program counter* e o registo de *flags* para os valores que estes tinham quando ocorreu a interrupção.
3. restaura o *program counter* e todos os registos de uso geral para o valor que estes tinham quando ocorreu a interrupção.
4. coloca o CPU em modo utilizador e desinibe o sistema de interrupções

Opção correcta ☐

Q-9 [0.5 val.] Na arquitectura hardware de um PC existe um controlador de interrupções (PIC). Quando uma rotina de tratamento de interrupções termina, antes de fazer *iret*, é preciso executar a instrução máquina

```
out 0x20, 0x20  ; Enviar EndOfInterrupt para o PIC
```

Porque é que é preciso fazer isto?

1. para reinicializar o PIC
2. para permitir que o CPU seja interrompido
3. para permitir interrupções com prioridade superior à corrente
4. para permitir interrupções com prioridade igual ou inferior à corrente

Opção correcta ☐

Teórica

Q-10 Suponha que um dado CPU gera endereços virtuais e que existe uma MMU que gera a memória física dividindo-a em páginas.

a) [0.5 val.] Admita que a tabela de páginas de um processo cabe completamente dentro da MMU. Para esta situação, faça um esquema em que mostre a transformação dos endereços virtuais em reais.

b) [0.5 val.] Admita que a tabela de páginas de um processo está armazenada na RAM, explique a necessidade de ser usada um *TLB* (Translation Lookaside Buffer).

c) [1.0 val.] Apresente uma nova versão do esquema da alínea a) para a situação em que a tabela de páginas está em RAM e existe TLB.

Q-11 [1.5 val.] Considere uma unidade de transformação de endereços (MMU) baseada em páginas, em que cada entrada da tabela de páginas tem um bit de validade V. Este bit é usado para suportar um sistema de memória virtual que funciona pelo técnica da *paginação a pedido*.

Quando um endereço E gerado pelo CPU referencia uma página P em que o bit de validade está a 0, a MMU guarda o endereço E e envia uma interrupção ao CPU. As linhas seguintes contêm uma descrição das acções efectuadas pelo sistema operativo (SO) quando ocorre uma *interrupção por falta de página*. Essa descrição não está completa; preencha o que falta:

- Se P não pertence ao espaço de endereçamento do processo, o processo é terminado.
- Se P pertence ao ao espaço de endereçamento do processo, é preciso carregar a página P na memória física. O processo corrente é suspenso enquanto isso acontece.
- O SO descobre que a página virtual P está no bloco B do disco
- O SO ...

- O SO programa o controlador de disco para transferir o bloco B para a página física F.
- O SO recebe a indicação de que a página virtual P já está na página física F.
- O SO actualiza a tabela de páginas do processo P da seguinte forma:

- A instrução que provocou a falta de página é retomada.

Q-12 [1.0 val.] Nos trabalhos práticos sobre entradas e saídas foi usado o sistema QEMU. O QEMU simula um PC completo com CPU, RAM, controlador interrupções, controlador de periféricos e alguns periféricos. Quando sobre esta máquina virtual se executa o sistema operativo FreeDOS, o CPU emulado funciona em modo de 16 bits e não tem instruções privilegiadas.
Explique porque é que estes exercícios foram efectuados neste ambiente. Seria possível tê-los realizado o sistema LINUX directamente? Justifique a resposta.

Q-13 [1.0 val.] Quando os controladores dos periféricos são programados usando espera activa, existe um grande desperdício das capacidades do CPU. Porquê?

Prática

Q-14 Considere que tem um programa, guardado no ficheiro executável *prog*, cujo código fonte em C está distribuído por três ficheiros: *f1.c*, *f2.c* e *f3.c*.

a) [0.5 val.] Que comando usaria para gerar os ficheiros objecto correspondentes, nomeadamente *f1.o*, *f2.o* e *f3.o*?

1. cc -c f1.c f2.c f3.c
2. cc -o f1.c f2.c f3.c
3. cc -o f1.o f1.c -o f2.o f2.c -o f3.o f3.c
4. cc -c prog f1.c f2.c f3.c
5. cc -o prog f1.o f2.o f3.o

Opção correcta ☐

b) [0.5 val.] Uma vez obtidos os ficheiros *f1.o*, *f2.o* e *f3.o*, que comando usaria para gerar o executável de nome *prog* a partir destes?

1. cc -c prog f1.c f2.c f3.c
2. cc -g prog f1.c f2.c f3.c
3. cc -o prog f1.o f1.c f2.o f2.c f3.o f3.c
4. cc -o prog f1.c f2.c f3.c
5. cc -o prog f1.o f2.o f3.o

Opção correcta ☐

c) [0.5 val.] Assuma que que tinha passado pelo processo das duas alíneas anteriores, mas observou um erro de execução no seu programa. Esse erro estava no ficheiro *f3.c*, que editou e corrigiu. Quer agora voltar a produzir o executável *prog*. Que comando usaria?

1. cc -c prog f1.c f2.c f3.c
2. cc -o prog f1.o f2.o f3.c
3. cc -o prog f1.o f2.o f3.o
4. cc -c f3.c
5. cc -o prog f3.o

Opção correcta ☐

Q-15 [2.5 val.] Considere os seguintes registos e respectivos bits, relevantes na utilização da porta série para a recepção de dados usando espera activa

Endereço	Registo	Bits	Significado
3F8h	RBR	0..7	Registo de recepção de dados
3FDh	LSR	0	Quando existe um carácter disponível no registo RBR, este bit fica a 1

Complete o seguinte código de uma função em C que deverá retornar um byte lido da porta série, utilizando espera activa. Assuma o acesso às funções

unsigned char inByte(**unsigned short** ad)	retorna o conteúdo do registo de E/S com o endereço *ad*
void outByte(**unsigned short** ad, **unsigned char** v)	escreve o valor *v* no registo com o endereço *ad*

```
unsigned char receiveSerial() {

        _________________ byte;

        // Esperar que exista um carácter disponível na porta série
        do {
                byte = inByte(____________);

        } while ( ___________________________ == 0);

        byte = ___________(_________);    // Ler e retornar o carácter disponível na porta série
        return byte;
}
```

Q-16 [2.5 val.] Assuma que já dispõe da função **unsigned char** receiveSerial(**void**) que permite receber um byte por uma porta série de um computador. Complete o código de receberFicheiro que é uma função que permita guardar num ficheiro tudo o que receber pela porta série. O último caracter a receber e que indica que o ficheiro chegou ao fim é o carácter ASCII EOT que tem o valor 0x04.

(a pergunta continua na página seguinte)

```
void receberFicheiro( char *nomeFicheiro ) {
    unsigned char b;
    FILE *f;
    f = ____________________ ( ____________________________ , "w" );
    if ( f==NULL ) return; //houve erro
```

}

Q-17 Considere os seguintes registos e respectivos bits, relevantes na utilização da porta série para a recepção de dados usando interrupções.

Endereço	Registo	Bits	Siginificado
3F8h	RBR	0..7	Registo de recepção de dados
3F9h	IER	0	Deverá ser gerada uma interrupção quando chega um carácter
3FCh	MCR	3	A UART deverá gerar interrupções
3FDh	LSR	0	Existe um carácter disponível no registo RBR

a) [2.0 val.] Complete o seguinte código de uma função de tratamento de interrupções em C, que deverá ler um carácter da porta série e colocá-lo num *buffer* circular como o utilizado nas aulas práticas. Assuma, que além das funções inByte() e outByte() definidas na pergunta 15, tem também acesso às seguintes funções:

void putBuf(**unsigned char**)	Coloca o byte passado como argumento no buffer circular; assume que não está cheio
unsigned char getBuf(**void**)	Retorna o byte que está há mais tempo no buffer circular; assume que não está vazio
int bufEmpty(**void**)	Retorna verdade se o buffer circular está vazio
int bufFull(**void**)	Retorna verdade se o buffer circular está completamente cheio

```
static void interrupt handlingRoutine() {

    ____________________ byte;

    if( ____________________________ ) {  // ler o carácter da porta série e colocar no buffer circular

        byte = inByte( ____________________ );

        ____________________________________;
    }
    outByte(0x20, 0x20);      // Enviar EndOfInterrupt para o PIC
}
```

b) [1.5 val.] Complete o seguinte código de uma função que programa a UART para gerar interrupções sempre que há um carácter disponível no registo RBR:

```
static void enableIntUART() {
        unsigned char b;

        b = inByte(_________________);

        outByte(_________________, _____________________);

        b = inByte(_________________);

        outByte(_________________, _____________________);
}
```