

Fundamentos de Sistemas de Operação
2º Teste, 4 de Janeiro de 2022

NOME DO ESTUDANTE: _____ **Nº:** _____

A duração do teste é 1h45 (incluindo a tolerância). Nas questões de escolha múltipla, as respostas erradas têm uma cotação negativa correspondente a 20% da classificação da questão. Por exemplo, se errar a resposta a uma questão cotada para 1.0 valor, terá uma cotação de -0,2 valores nessa questão. A classificação total das questões de escolha múltipla pode, portanto, ser negativa.

As questões de escolha múltipla devem ser respondidas na folha própria para o efeito.

Para anular uma resposta coloque uma cruz por cima e pinte a nova resposta (X ○ ○ ● ○).

Para reativar uma resposta previamente anulada faça um círculo à volta da resposta a reativar (○ X ○ ○ X ○).

As questões de desenvolvimento devem ser respondidas no próprio enunciado.

As páginas 2, 13 e 14 podem ser usadas para **RASCUNHO**

Fundamentos de Sistemas de Operação
2º Teste, 4 de Janeiro de 2022

QUESTÕES DE ESCOLHA MÚLTIPLA — VERSÃO A— 10 valores

1) Qual das alíneas considera apresentar as características mais importantes da técnica de gestão de memória designada como “memória virtual por paginação-a-pedido” (*demand paging*)?

boa utilização da RAM instalada, pode correr processos cuja dimensão do espaço de endereçamento (EE) excede a RAM instalada, protecção das diferentes regiões do EE deficiente (permite intrusão de hackers e/ou vírus), o desempenho só é bom se se usarem discos HDD em RAID-0 para a área de paginação

boa utilização da RAM instalada, pode correr processos cuja dimensão do espaço de endereçamento (EE) excede a RAM instalada, protecção das diferentes regiões do EE de acessos indevidos deficiente (só protege as páginas de código e de constantes), bom desempenho se os acessos exibirem boa localidade de referência espacial e temporal

boa utilização da RAM instalada, pode correr processos cuja dimensão do espaço de endereçamento (EE) excede a RAM instalada, boa protecção das diferentes regiões do EE de acessos indevidos, mas o desempenho só é bom se se usarem discos SSD para a paginação

boa utilização da RAM instalada, pode correr processos cuja dimensão do espaço de endereçamento (EE) excede a RAM instalada, boa protecção das diferentes regiões do EE de acessos indevidos, bom desempenho se os acessos exibirem boa localidade de referência espacial e temporal

má utilização da RAM instalada (por fragmentação interna excessiva), pode correr processos cuja dimensão do espaço de endereçamento (EE) excede a RAM instalada, protecção das diferentes regiões do EE de acessos indevidos, fraco desempenho se os acessos exibirem boa localidade de referência espacial e temporal

2) Num computador com um processador do tipo x86 que corre um SO que suporta “memória virtual por paginação-a-pedido” (*demand paging*), qual das seguintes afirmações sobre a dimensão do espaço de endereçamento (EE) de um processo é a mais correcta?

Pode exceder a dimensão da RAM instalada no computador

Pode exceder a dimensão da RAM instalada no computador, sendo limitada pela dimensão dos registos `eax`, `ebx`, `ecx`, `edx`

Não pode exceder a dimensão da RAM instalada no computador

Não pode exceder a dimensão da RAM instalada no computador e é limitada pela dimensão dos registos `eip` e `esp`

Pode exceder a dimensão da RAM instalada no computador, sendo limitada pela dimensão dos registos `eip` e `esp`

3) Num computador com um processador do tipo x86 que corre um SO que suporta “memória virtual por paginação-a-pedido” (*demand paging*), qual das seguintes afirmações sobre a ocorrência de uma falta de página (*page fault*) durante a execução de um processo é a mais correcta?

A execução do processo é terminada pelo SO

Se não houver RAM livre, a execução do processo é terminada pelo SO

A tabela de *frames* (molduras) é consultada para verificar se a *frame* referenciada na falta é válida

A tabela de páginas é consultada para verificar se a página referenciada na falta é válida

A tabela de páginas é consultada para verificar se a página referenciada na falta é válida e, caso seja, o TLB (*Translate Look-aside Buffer*) gera uma interrupção

4) Num computador com um processador do tipo x86 que corre um SO que suporta “memória virtual por paginação-a-pedido” (*demand paging*), qual das seguintes afirmações sobre as dimensões das páginas, *frames* (molduras) e blocos de disco é a mais correcta?

\ dimensão da página é igual à dimensão da *frame* e independente da dimensão do bloco de disco

\ dimensão da página é igual à dimensão da *frame* e igual à dimensão do bloco de disco

\ dimensão da página é um múltiplo da dimensão da *frame* e igual à dimensão do bloco de disco

A dimensão da página é um sub-múltiplo da dimensão da *frame* e igual à dimensão do bloco de disco

A dimensão da página é um sub-múltiplo da dimensão da *frame* e independente da dimensão do bloco de disco

5) Quando um processo tenta o `open()` de um ficheiro, o controle de acessos do SO Unix/Linux, usando os 9 “símbolos” da “máscara” de permissões `r`, `w`, `x` ou `-`

deixa que qualquer processo abra o ficheiro (a validação só se faz quando o processo tenta ler ou escrever)

deixa abrir o ficheiro se o UID do processo é zero; senão, usa o UID do processo e duas “máscaras” de permissões: a da directoria onde se encontra “alojado” o ficheiro, e a do próprio ficheiro, para determinar se o processo o pode abrir

se a “máscara” de permissões da directoria onde se encontra “alojado” o ficheiro incluir em algum “lugar” a permissão `x`, qualquer o processo o pode abrir (o resto da validação só se faz quando o processo tenta ler ou escrever)

usa o UID do processo e a “máscara” de permissões da directoria onde se encontra “alojado” o ficheiro para determinar se o processo o pode abrir

deixa abrir o ficheiro se o processo estiver a correr em modo supervisor (ou *kernel*); senão, usa o UID do processo e duas “máscaras” de permissões: a da directoria onde se encontra “alojado” o ficheiro e a do próprio ficheiro, para determinar se o processo o pode abrir

6) Considere um grupo de 4 discos HDD absolutamente idênticos sobre os quais foi criado um volume RAID-5. Sabendo que para um disco individual acedido sequencialmente em leitura a latência é L_i e a largura de banda é B_i , para o grupo ter-se-à, **aproximadamente** (sendo, respectivamente, L_G a latência e B_G a largura de banda do volume)

$$L_G = L_i \text{ e } B_G = 5 \times B_i$$

$$L_G = 5 \times L_i \text{ e } B_G = 5 \times B_i$$

$$L_G = L_i / 4 \text{ e } B_G = 4 \times B_i$$

$$L_G = L_i \text{ e } B_G = 4 \times B_i \text{ (Lapso na pergunta, devia ser } 3 \times B_i \text{. 100\% a quem marcou esta, outras sem desconto)}$$

Nenhuma das anteriores, um volume RAID tem de ter 5 discos

7) Considere um grupo de 2 discos SSD absolutamente idênticos sobre os quais foi criado um volume RAID-1. Sabendo que para um disco individual, cuja capacidade é V_i , quando acedido sequencialmente a latência é L_i e a largura de banda é B_i (tanto para leitura como para escrita), para o grupo ter-se-à, **aproximadamente** (sendo, respectivamente, L_G a latência, B_G a largura de banda e V_G a capacidade do volume,

$$L_G = L_i \text{ e } B_G = 2 \times B_i; V_G = 2 \times V_i$$

$$L_G = L_i / 2 \text{ e } B_G = 2 \times B_i; V_G = V_i$$

$$L_G = L_i \text{ e } B_G = 2 \times B_i; V_G = V_i$$

Nenhuma está correcta, porque a largura de banda do volume em escrita é diferente da em leitura

Nenhuma das anteriores, porque o valor correcto para a latência é $L_G = 2 \times L_i$

8) A técnica de *journaling* de **metadados**, usada em sistemas de ficheiros (SF) consiste sucintamente em, quando um programa em execução faz uma operação de I/O que altera uma ou mais estruturas de metadados,

Nas operações de **read()** e **write()** garantir que os blocos modificados são efectivamente escritos pelo SO (*driver*) no disco antes de retornar ao processo de utilizador

Nas operações de **write()** garantir que os blocos modificados são efectivamente escritos pelo SO (*driver*) no disco antes de retornar ao processo de utilizador

Escrever, numa zona reservada do disco, em todas as operações que alteram alguma área de **dados ou metadados**, uma cópia de todas as alterações provocadas pela operação e, quando todas elas estiverem efectivamente escritas em disco, adicionar uma marca de fim-de-operação e, em seguida, começar a escrever a mesma informação, quando for oportuno, nas zonas do SF onde estão as áreas de **dados e metadados** afectadas pela operação

Escrever, numa zona reservada do disco, em todas as operações que alteram alguma área de **metadados**, uma cópia de todas as alterações provocadas pela operação e, quando todas elas estiverem efectivamente escritas em disco, adicionar uma marca de fim-de-operação e, em seguida, começar a escrever a mesma informação, quando for oportuno, nas zonas do SF onde estão as áreas de **metadados** afectadas pela operação

Escrever, numa zona reservada do disco, em todas as operações que alteram alguma área de **metadados**, uma cópia de todas as alterações provocadas pela operação e, quando todas estiverem efectivamente escritas em disco, escrever uma marca de fim-de-operação

9) A memória virtual por paginação-a-pedido tem, entre outras, as seguintes características:

cada processo tem o seu espaço de endereçamento dividido num conjunto de páginas de dimensão fixa

regra geral, cada página só é carregada para memória quando é referenciada

cada processo tem a uma tabela de páginas (PT – *Page Table*)

cada entrada na PT tem um bit de presença que indica se a página se encontra em memória

todas as anteriores (Se respondeu outras, tem ¼ da cotação por cada uma)

10) Nos sistemas de ficheiros (SF) nativos (ex.: **ext2**) em SOs *Unix-like*, o superbloco contém, entre outras, informações sobre

O número de ficheiros, bitmaps e i-nodes existentes no disco

O número de tabelas de ficheiros, tabelas de bitmaps e tabelas de i-nodes existentes no disco

As listas que gerem a ocupação de estruturas como i-nodes, blocos de dados, bitmaps e ficheiros

A *File Allocation Table* (FAT) do SF

Nenhuma das anteriores

11) O que caracteriza um escalonador MLFQ (*Multi-Level Feedback Queue*) é

- não usar fatias de tempo
- favorecer a execução dos processos CPU-bound
- distribuir equitativamente o tempo de CPU pelos processos
- escalonar os processos preservando a sua antiguidade (exibindo um carácter FIFO)
- nenhuma das anteriores

12) Num programa o programador declarou/inicializou a variável `i` da seguinte forma: `int *i = 0`. Depois, logo a seguir, escreveu a instrução `*i = 99`. O que acontece quando a execução do programa chega à referida instrução, numa máquina com um processador x86 e sob um SO Windows ou Linux?

Um *segmentation fault*, consequência de uma tentativa de acesso ao endereço 0, que está contido numa página inválida

Um *segmentation fault*, consequência de uma tentativa de acesso a um endereço contido na zona de endereços de memória virtual atribuídos ao *heap*, que não foi previamente “alocado”

Um *page fault*, consequência de uma tentativa de acesso a um endereço contido numa página do *heap*, que não foi previamente carregada em memória

Um *page fault*, consequência de uma tentativa de acesso a um endereço contido numa página do *stack*, que aponta para um endereço contido numa página do *heap* que não foi previamente carregada em memória

Nenhuma das anteriores, o programa executável não é criado porque falha na fase de *link* (ligação).

13) As estruturas de dados designadas por *bitmaps* são usadas em sistemas de ficheiros para

- indicar em que blocos do disco estão os ficheiros
- indicar em que blocos do disco estão os i-nodes
- indicar se outras estruturas de dados do SF estão livres (ou “vazias”) ou ocupadas (ou “cheias”)
- indicar, para cada ficheiro, quais os blocos que lhe estão atribuídos e se estes estão livres ou ocupados
- Nenhuma das anteriores

14) Depois de uma falha de energia ou de um *crash* do sistema de operação, num computador com um volume formatado com um determinado sistema de ficheiros (SF) e em uso (montado) exclusivamente para leitura, a consistência do SF

- tem de ser verificada
- não** tem de ser verificada
- só tem de ser verificada se se tratar de um SF com *journaling*
- só tem de ser verificada se se tratar de um SF sem *journaling*
- só tem de ser verificada se se tratar de um SF nativo do SO em causa (por exemplo, `ext2` num SO Linux ou NTFS num SO Windows)

15) Nos sistemas de ficheiros (dos quais o BFS do mini-projecto, ou os **ext** do Linux são exemplos), a operação usualmente designada como “formatar”

Percorre todos os ficheiros existentes, apagando primeiro os seus blocos e depois apagando as correspondentes entradas nas directorias, até que só reste a directoria raiz, que fica também vazia

Define as partições e o número de i-nodes e de blocos de dados

Inicializa todas as estruturas de dados do SF que são usadas para gerir a informação que irá futuramente ser guardada no disco

Inicializa as estruturas de dados do SF que são usadas para gerir a informação que irá ser guardada no disco, e depois monta o disco

Inicializa unicamente o superbloco

16) Num HDD contendo um sistema de ficheiros em que a atribuição de espaço (*data block allocation*) é de tipo contíguo, a velocidade de acesso a posições logicamente contíguas do ficheiro é

- a) maior em escrita do que em leitura
- b) maior do que no caso de uma atribuição em lista ligada, mas menor do que no caso de atribuição indexada
- c) maior do que em qualquer outro tipo de atribuição
- d) maior do que em qualquer outro tipo de atribuição, para acessos em leitura
- e) menor do que em qualquer outro tipo de atribuição

17) Num disco com B blocos, formatado para conter um sistema de ficheiros que suporta atribuição (de blocos de dados) baseada unicamente num vector com N apontadores directos de dimensão A_p **bytes**, e em que cada apontador aponta para um *cluster* de 8 blocos, a dimensão mínima para um apontador, A_p , é dada por [Nota: B e N são potências de 2]

$$A_p = \text{ceil}(\log_2(N)/B)/8$$

$$A_p = \text{ceil}(\log_2(N/B)/8)$$

$$A_p = \text{ceil}(\log_2(B)/8)$$

$$A_p = \text{ceil}(\log_2(B/8)/8)$$

$$A_p = (2^8) * \text{ceil}(\log_2(N/8))$$

18) Num disco com B blocos de 512 bytes, formatado para conter um sistema de ficheiros que suporta atribuição (de blocos de dados) baseada unicamente num vector com N apontadores directos de dimensão A_p **bytes**, e em que cada apontador aponta para um *cluster* de 8 blocos, a dimensão máxima, D_f , de um ficheiro, é dada (em bytes) por [Nota: B e N são potências de 2]

a) $D_f = N*B*512*8$

b) $D_f = N*B*(512/A_p)$

c) $D_f = N*B*(512/A_p)*8$ Pergunta ANULADA, cotada 100% em todos os testes

d) $D_f = N*A_p*B*512*8$

e) $D_f = N*A_p*B*512$

19) Na indicação de quais os blocos pertencentes a um ficheiro

a atribuição contígua gera fragmentação externa

a atribuição contígua apresenta dificuldades no crescimento dos ficheiros

a atribuição ligada apresenta um acesso directo muito lento

a atribuição indexada tem como desvantagens o espaço ocupado pelos índices e a complexidade na sua gestão

todas as afirmações anteriores são verdadeiras

20) Num sistema de operação, muitas das técnicas e opções utilizadas no seu “desenho” (*design*) ou implementação são motivadas pela necessidade de acelerar a execução de aplicações e diminuir a latência de acesso aos dados. Assim, é importante conhecer as ordens de grandeza (valores típicos) de tempos de: execução de instruções sobre registos do CPU (*tCPUins*), acesso a RAM (*tRAM*), acesso a um bloco (não *cached*) de disco num HDD (*tHDD*) e num SSD (*tSSD*), tradução de endereço virtual/físico num TLB aquando de um hit (*tHIT*) ou miss (*tMISS*), etc., pois só assim se compreende o SO. Considerando as seguintes unidades **ns** (nano-segundos) **μs** (micro-segundos) e **ms** (mili-segundos), qual das opções abaixo representa valores credíveis?

tCPUins: dezenas de ns; *tRAM*: dezenas de ns; *tHDD*: dezenas de ms; *tSSD*: μs; *tHIT*: dezenas de ns; *tMISS*: dezenas de μs

tCPUins: ns; *tRAM*: dezenas de ns; *tHDD*: ms; *tSSD*: dezenas/centenas de μs; *tHIT*: ns; *tMISS*: ns

tCPUins: ns; *tRAM*: ms; *tHDD*: ms; *tSSD*: dezenas/centenas de μs; *tHIT*: ns; *tMISS*: dezenas de μs

tCPUins: ns; *tRAM*: dezenas de ns; *tHDD*: ms; *tSSD*: ms; *tHIT*: ns; *tMISS*: dezenas de μs

tCPUins: ns; *tRAM*: dezenas de ns; *tHDD*: ms; *tSSD*: dezenas/centenas de μs; *tHIT*: ns; *tMISS*: dezenas de μs

Fundamentos de Sistemas de Operação
2º Teste, 4 de Janeiro de 2022

QUESTÕES DE DESENVOLVIMENTO — VERSÃO A – 10 valores

D1) Considere o fragmento de programa (fonte) abaixo (Fig 1.1) que é executado num computador “imaginário” com um SO que usa paginação-a-pedido. O CPU tem registos gerais de **32 bits** mas usa endereços (físicos e lógicos) com apenas **8 bits**, e páginas de **4 bytes**. O espaço de endereçamento (virtual, Fig. 1.2), o mapa (ou tabela) de páginas (Fig. 1.3), e a RAM (Fig. 1.4) estão também desenhados.

```
...
int v[MAX];
for (i= 0; i < MAX; i++) v[i]= i;
...
```

Fig 1.1

Página	Frame
0	0
1	NP
...	NP

Fig. 1.3

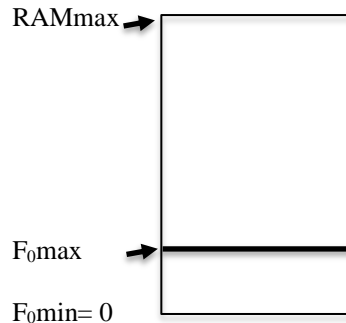


Fig. 1.4

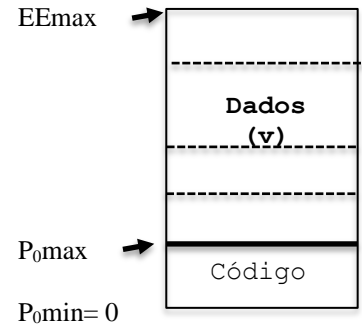


Fig. 1.2

Notas adicionais (**ler com atenção**):

- i) a variável *i* foi “alocada” num registo, não está em memória; *v* não está no *stack*, mas é global
- ii) “Faça de conta” que o SO não ocupa memória.
- iii) A indicação NP indica que a página em causa é válida mas não está carregada em memória.
- iv) EEmax e RAMmax representam os maiores endereços possíveis para, respectivamente, o Espaço de Endereçamento virtual (EEmax) e físico (RAMmax).
- v) Analogamente, P0max e F0max representam os maiores deslocamentos possíveis para, respectivamente, uma página (neste caso, a zero) e uma *frame* (também neste caso a zero).
- vi) Só há uma página de código, que é imediatamente seguida pelas páginas de dados. Não há *stack* nem *heap*.

a) Calcule os seguintes valores

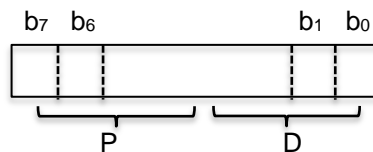
EEmax=

RAMmax= .

P0max=

F0max=

b) Decomponha um endereço virtual, de 8 bits, em “Nº de página” (P) e “deslocamento” (D) indicando na figura abaixo, quantos bits têm P e D



P=

D= .

- c) Descreva o que acontece quando é necessário executar pela primeira vez a instrução máquina correspondente à atribuição (em C) $v[i] = i$ que pode ver no ciclo `for`, Fig. 1.1. Tenha em atenção a tabela de páginas (*page table*) apresentada na Fig. 1.3. Considere que toda a RAM excepto, naturalmente, a *frame* 0, está vazia.

- d) Considere $MAX = 16$. Calcule P_v , o número de páginas ocupado por $v[MAX]$ e P_f , o número de faltas de página sofridas pelo programa até que se complete o ciclo `for`. [Recorde que se trata de uma arquitectura com registos de **32 bits** e v é um vector de **inteiros**]

$P_v =$

$P_f =$

Fundamentos de Sistemas de Operação
2º Teste, 4 de Janeiro de 2022

QUESTÕES DE DESENVOLVIMENTO — VERSÃO B – 10 valores

D1) Considere o fragmento de programa (fonte) abaixo (Fig 1.1) que é executado num computador “imaginário” com um SO que usa paginação-a-pedido. O CPU tem registos gerais de **32 bits** mas usa endereços (físicos e lógicos) com apenas **10 bits**, e páginas de **8 bytes**. O espaço de endereçamento (virtual, Fig. 1.2), o mapa (ou tabela) de páginas (Fig. 1.3), e a RAM (Fig. 1.4) estão também desenhados.

```
...
int v[MAX];
for (i= 0; i < MAX; i++) v[i]= i;
...
```

Fig 1.1

Página	Frame
0	0
1	NP
...	NP

Fig. 1.3

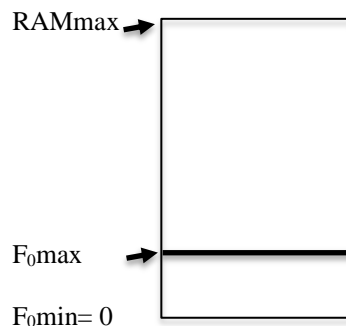


Fig. 1.4

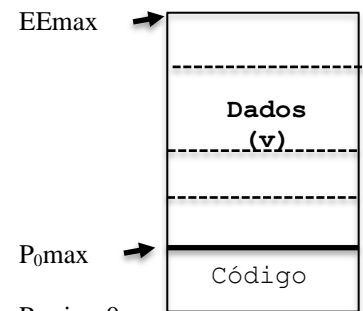


Fig. 1.2

Notas adicionais (**ler com atenção**):

- i) a variável *i* foi “alocada” num registo, não está em memória; *v* não está no *stack*, mas é global
- ii) “Faça de conta” que o SO não ocupa memória.
- iii) A indicação NP indica que a página em causa é válida mas não está carregada em memória.
- iv) *EEmax* e *RAMmax* representam os maiores endereços possíveis para, respectivamente, o Espaço de Endereçamento virtual (*EEmax*) e físico (*RAMmax*).
- v) Analogamente, *P0max* e *F0max* representam os maiores deslocamentos possíveis para, respectivamente, uma página (neste caso, a zero) e uma *frame* (também neste caso, a zero).
- vi) Só há uma página de código, que é imediatamente seguida pelas páginas de dados. Não há *stack* nem *heap*.

a) Calcule os seguintes valores

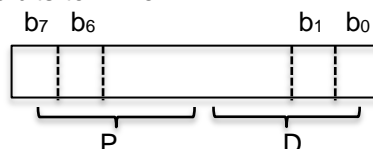
EEmax=

RAMmax=

P0max=

F0max=

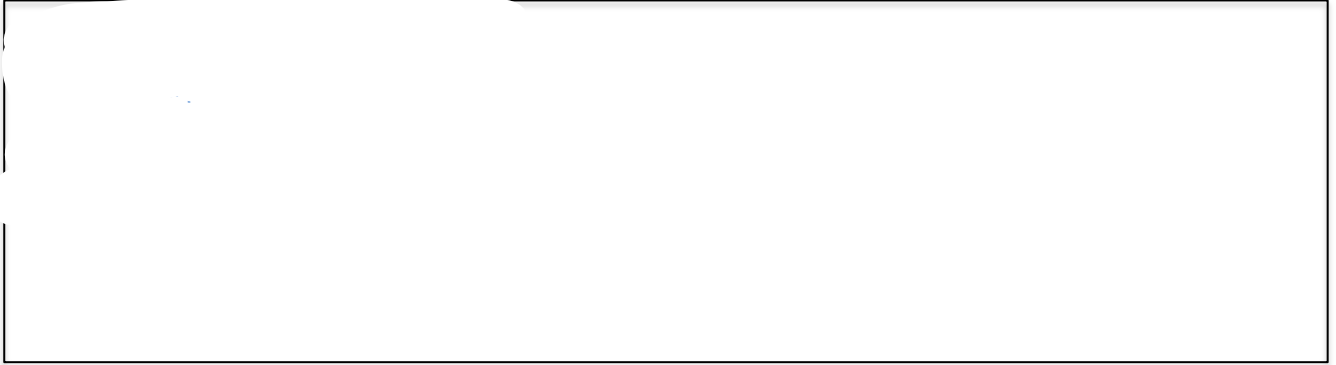
b) Decomponha um endereço virtual, **de 8 bits** (**lapso, são 10 bits!**), em “Nº de página” (*P*) e “deslocamento” (*D*) indicando na figura abaixo, quantos bits têm *P* e *D*



P=

D=

- c) Descreva o que acontece quando é necessário executar pela primeira vez a instrução máquina correspondente à atribuição (em C) $v[i] = i$ que pode ver no ciclo `for`, Fig. 1.1. Tenha em atenção a tabela de páginas (*page table*) apresentada na Fig. 1.3. Considere que toda a RAM excepto, naturalmente, a *frame* 0, está vazia.



- d) Considere $MAX = 16$. Calcule P_v , o número de páginas ocupado por $v[MAX]$ e P_f , o número de faltas de página sofridas pelo programa até que se complete o ciclo `for`. [Recorde que se trata de uma arquitectura com registos de **32 bits** e v é um vector de **inteiros**]

$P_v =$

$P_f =$

D2) Considere o tipo `inodeC`, cuja estrutura é apresentada abaixo e representa um ficheiro contíguo. Naturalmente, o comprimento do ficheiro é `size` bytes, sendo `start` o endereço lógico na zona de dados do primeiro bloco ocupado pelo ficheiro no disco e `end` o endereço do último.

Pretende-se que escreva o código da função `fullFileRead` que é chamada para ler integralmente os dados do ficheiro representado pelo argumento `in` (que já aponta um i-node válido). O ficheiro pode estar vazio, mas também pode ter sido previamente escrito e, portanto, ter um certo conteúdo. **A função deve retornar o número de bytes lido**, ou um código de erro que resulte da chamada a outra função (pode usar -1 para outros erros, se no seu código encontrar situações que acha serem erros). Os bytes serão lidos para `buf`, uma variável local da função.

```
struct inodeC {
    unsigned short size;
    unsigned short start;
    unsigned short end;
};

// converte o endereço lógico de um bloco de dados
// no endereço físico desse bloco em disco. Retorna < 0 se erro
extern unsigned int L2FdataBlock(unsigned int lBlock);

// Lê o conteúdo do bloco de disco cujo endereço é pBlock
// Retorna < 0 se erro
extern int BlockRead(unsigned int pBlock, unsigned char *blk)
```

```
int fullFileRead(struct inodeC *in) {
    unsigned char buf[DISK_BLOCK_SIZE];

}
}
```

D3) Considere o tipo `inodeI`, cuja estrutura é apresentada abaixo e representa um ficheiro indexado. Naturalmente, o comprimento do ficheiro é `size` bytes, sendo `idx` o endereço lógico, na zona de dados, do bloco de índice. Um bloco de índice tem `IDX_POINTERS`, que são apontadores para os blocos de dados do ficheiro. Cada apontador ocupa 2 bytes.

Pretende-se que escreva o código da função `fullFileRead` que é chamada para ler integralmente os dados do ficheiro representado pelo argumento `in` (que já aponta um i-node válido). O ficheiro pode estar vazio, mas também pode ter sido previamente escrito e, portanto, ter um certo conteúdo. **A função deve retornar o número de bytes lido**, ou um código de erro que resulte da chamada a outra função (pode usar -1 para outros erros, se no seu código encontrar situações que acha serem erros). Os bytes serão lidos para `buf`, uma variável local da função. Também existe uma variável local, `idx`, que servirá para conter o bloco de índice.

```
struct inodeI {
    unsigned short size;
    unsigned short idx;
};

// converte o endereço lógico de um bloco de dados
// no endereço físico desse bloco em disco. Retorna < 0 se erro
extern unsigned int L2FdataBlock(unsigned int lBlock);

// Lê o conteúdo do bloco de disco cujo endereço é pBlock
// Retorna < 0 se erro
extern int BlockRead(unsigned int pBlock, unsigned char *blk);

#define IDX_POINTERS (DISK_BLOCK_SIZE/sizeof(unsigned short));

int fullFileRead(struct inodeC *in) {
    unsigned int eBlk; // endereço (nº) lógico de bloco
    unsigned char buf[DISK_BLOCK_SIZE];
    union b_idx {
        unsigned char block[DISK_BLOCK_SIZE];
        unsigned short dptr[IDX_POINTERS];
    } idx;

    // Ler o bloco de índice
    if ( (eBlk=L2FdataBlock(in->idx)) < 0) return blk;
    if ( BlockRead(eBlk, idx.block) < 0) return blk

    // Enquanto forem válidos, percorrer os apontadores no bloco de índice
    // e ler cada bloco de dados apontado, e actualizar a quantidade de
    // bytes lidos

    // Marcado a amarelo = não desconta
    // Marcado a azul: lapso no enunciado

}
```