

- 1) Considere variantes de uma arquitectura com páginas de 4KiB, três níveis de tabelas de páginas e com entradas de 8 bytes. Cada tabela de páginas ocupa, no limite, uma página. Indique justificando:
- Qual o número máximo de bits úteis que pode ter um endereço virtual?
  - Se cada entrada das tabelas de páginas armazenar 38 bits do endereço físico, qual é a dimensão máxima da memória principal?
  - Considere um endereço virtual constituído por 36 bits. Descreva a organização do endereço virtual que será usado para indexar os três níveis das tabelas de páginas de modo a minimizar o espaço desperdiçado interno das tabelas.
  - Para as condições das alíneas b e c, considere os valores de tabelas de páginas na memória principal que suportam a tradução de endereços para o processo em execução com excerto de código apresentado ao lado. Considere ainda que a tabela raiz está localizada no endereço 0x22000 (endereço alinhado à dimensão da página).

```

mov ecx, [ 4304F0 ] ; ecx = 1
mov edx, [ 4325E8 ] ; edx = 435FF8
xor eax, eax
jmp .L3
.L2: add [edx], eax      ; *edx += eax
    i.      add eax, 4
    sub ecx, 1
.L3: cmp ecx, 0
    mov [ 430700 ], eax
    jg .L2
.L4:

```

- Indique, para cada endereço de dados apresentado no código, o respectivo endereço físico. Justifique a sua resposta.
- Qual a consequência de se retirar a permissão de escrita das PTE de índice 0x30, 0x32 e 0x35 na tabela de 3º nível?

- 2) O documento *pdf* em anexo contém a secção 11.5.3 do livro *Modern Operating System (4ed)*, de Andrew Tanenbaum, que apresenta aspectos de implementação do gestor de memória no Windows 8 (e Windows 10). Usando o texto como fonte de informação principal, responda às seguintes questões:

- O texto organiza o tratamento da excepção *page fault* pelo respectivo *handler* em cinco categorias distintas. Enumere-as fazendo uma descrição breve do seu significado e descrevendo o tratamento correspondente do *page fault handler*.
- No texto é apresentado um esquema de carregamento de páginas usada no Windows no arranque dos processos em alternativa ao *demand paging*. Como é designada pela Microsoft a tecnologia que o suporta, em que consiste, e quais as potenciais vantagens?
- Apresentada a distinção entre *soft page fault* e *hard page fault*. Quais os cenários em que segundo o autor levam à ocorrência de *soft page faults*? Na resposta a esta questão considere a informação adicional presente em <http://www.tenforums.com/windows-10-news/17993-windows-10-memory-compression.html>.
- Nas versões do Windows usadas em *mobile*, introduziu-se o conceito de *swap file* em alternativa ao *paging file* tradicional. Em que consiste e qual a motivação para o seu aparecimento?

Mem. principal		
		0x020000
0x02	0x024000   rx	
	...	
0x00	0x020000   rx	0x022000
	...	
		0x024000
0x01	0x035000   rx	
0x02	0x0AB04000   rx	
0x30	0x6A00000   rw	
0x31	0x6B00000   rw	
0x32	0x6C00000   rw	
0x33	0x5A00000   rw	
0x34	0x5B00000   rw	
0x35	0x5C00000   rw	
Índices	...	Endereços de

- e Na figura 11.31 o autor sugere que as PTE das MMU x86 e x64 sejam de 8 bytes (64 bits), quando em geral as PTE são de 4 bytes na MMU de 32 bits. Investigue as razões porque o autor não está realmente a cometer uma incorreção ao fazer essa afirmação. Note que o autor se refere a versões do Windows a partir do Windows 8.

Sugestão: a partir do Windows 8, de modo a evitar que páginas de dados possam ser executadas (*Data Execution Prevention* – DEP), é usado o bit NX (No eXecutable) disponível nas arquitecturas x86 e x64.

- 3) Implemente um programa que determina a região reservada do espaço de endereçamento do utilizador com maior dimensão do processo passado por argumento à aplicação. O programa recebe por argumento o identificador do processo a analisar e apresenta na consola o nome do executável do processo analisado, o endereço base e dimensão da região determinada. A dimensão da região é apresentada na unidade Bytes.
- 4) Observe o espaço de endereçamento virtual de vários processos à sua escolha através da ferramenta VMMMap (ferramenta disponível para descarregar em <https://technet.microsoft.com/en-us/sysinternals/vmmmap.aspx>). Localize os endereços base das bibliotecas dinâmicas de sistema kernel32.dll e user32.dll presentes praticamente em todos os processos em execução no sistema. Verificará que estas bibliotecas (e outras de sistema) ocupam as mesmas regiões de memória independentemente do processo em observação.
- a) Explique as vantagens desta característica.
- b) No caso de ocorrerem intersecções entre duas *dll*'s, que não de sistema, necessárias a determinado processo, como resolve o *loader* essa questão?
- 5) O *standard Exif* (*Exchangeable Image File Format*) especifica o formato de armazenamento dos metadados associados a ficheiros de imagem, nomeadamente ficheiros JPEG. Os metadados especificados pelo *Exif* incluem informação da câmara, das características e condições de aquisição da foto e do contexto externo, nomeadamente a localização GPS da captura da foto. A especificação do *standard Exif* encontra-se em <http://www.exif.org/Exif2-2.PDF>. Uma versão menos formal pode ser encontrada em <http://www.media.mit.edu/pia/Research/deepview/exif.html>.

Implemente um programa que apresenta na consola o conjunto de *Tags* de metadados *Exif* de acordo com o formato seguinte:

```
Modelo: <camera model value>
Dimensão: <width value> px x <height value> px
Data: <date and time value>
ISO: <ISO value>
Velocidade: 1 / <exposure time value> s
Abertura: F 1 / <aperture value>
Latitude: <latitude value> <N | S>
Longitude: <longitude value> <E | W>
Altitude: <altitude value> m
```

As *Tags* apresentadas acima relacionadas com as condições de aquisição da fotografia (ISO, velocidade e abertura) e com o local da captura (latitude, longitude e altitude) estão definidas em descritores denominados por *subIFD* (*sub Image File Directory*) referidos no descritor IFD0.

Na sua implementação mapeie a imagem *filename* no espaço de endereçamento do processo corrente e defina estruturas que caracterizam os metadados *Exif* que permitam navegar directamente nos metadados *Exif* da imagem mapeada em memória.

- 6) Implemente uma DLL que exporta a função `VOID PrintExifTags(TCHAR filename)` sendo uma adaptação do programa da alínea anterior. A sua implementação deve suportar tanto clientes que usam caracteres codificados no código ASCII como clientes que usam caracteres codificados no código Unicode.