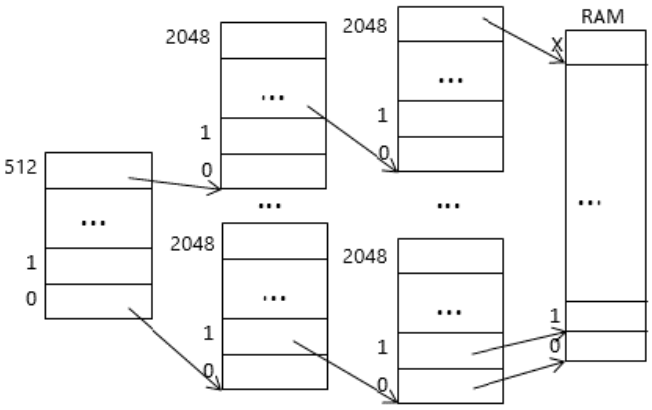
Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Sistemas Operativos, Inverno de 2018/19
Primeira Série de Exercícios

1) Considere a organização da MMU apresentada à frente. Sabendo que cada PTE ocupa 4 bytes e que cada tabela de 3º nível ocupa exatamente uma página, indique, apresentando cálculos justificativos:

a) Qual a dimensão de cada página, e qual o número de *bits* de um endereço virtual?

b) Sabendo que cada PTE está organizada em 27 *bits* para especificar uma PFN e os restantes *bits* para controlo, determine o número total de bits do espaço de endereçamento físico do sistema, o valor da última PFN (X, na imagem) e o número de *bits* de controlo de uma PTE.

c) Até quanto pode crescer o espaço de endereçamento virtual, mantendo o número de níveis apresentado para a MMU?



d) Apresente um diagrama com as tabelas e as respetivas PTEs envolvidas na tradução do endereço virtual 0x701053CC na página física que começa no endereço 0x100400000. Considere que as tabelas de 1º, 2º e 3º nível usadas na tradução estão alojadas nas páginas físicas com os endereços 0x200000, 0x0A0000 e 0x01000000, respectivamente. Apresente no seu diagrama os valores dos endereços físicos envolvidos na tradução (PTEs e endereço físico final).

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

a) 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*.

b) Apresente, resumidamente, como o Windows, baseado exclusivamente no endereço virtual cuja tradução provocou excepção, produz os endereços virtuais das PTE's envolvidas na tradução.

c) No texto é apresentado um esquema de carregamento de páginas usada no Windows em alternativa ao *demand paging*. Como se designa a tecnologia que o suporta, em que consiste, e quais as potenciais vantagens?

d) 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.

e) Nas versões mais recentes do Windows, nomeadamente versões 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?

3) Considere a solução em anexo `MemManagment` constituída pelo programa `MemMngApp` e pelas DLL's `MemMngDll` e `MemMngExp1`. Execute a três experiências presentes na solução e preencha, para cada experiência, a próxima tabela indicando em cada ponto os valores solicitados em cada coluna. Indique a ausência de variações ou variações que considere insignificantes com o carácter '-'. Para observar os valores solicitados na tabela, utilize o utilitário *Task Manager* e seleccione na página *Details* as colunas *Commit*, *WorkingSet* privado e partilhado. Justifique para cada ponto (2 e seguintes) as variações observadas.

	Commit #processId	WS privado #processId	WS partilhável #processId	Justificação
Ponto 1				
Ponto 2 e seguintes				

4) Implemente a função `GetCommitCountersFromProcess` que determina a dimensão da memória *Committed* alocada ao processo com o identificador `pid` descriminada pelos tipos de alocação (imagem, mapeada ou privada). A função retorna `TRUE` e em `counters` os valores dos contadores de *Commit* ou `FALSE` e zero nos contadores, em caso de erro. Teste a função para os vários pontos de execução das experiências anteriores. Para o teste, implemente um programa que recebe como argumento da aplicação o identificador do processo a analisar.

```
typedef struct {
      DWORD img;
      DWORD map;
      DWORD prv;
} CommitCounters, * PCommitCounters;

BOOL GetCommitCountersFromProcess(int pid, _Out_ PCommitCounters counters);
```

5) As câmaras digitais actuais utilizam o *standard Exif* (*Exchangeable Image File Format*) como formato de armazenamento das fotografias, quer dos pixeis constituintes, quer dos metadados associados. 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 a função `VOID PrintExifTags(TCHAR filename)` 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 of picture taken>
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>
```

Algumas *Tags* apresentadas acima estão relacionadas com as condições de aquisição da fotografia (ISO, velocidade e abertura) e com a localização onde foi tirada (latitude, longitude e altitude). Estas *Tags* estão definidas em descriptores denominados por *subIFD* (*sub Image File Directory*) referidas no descriptor 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 exporte a função implementada na 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.

Data limite de entrega: 4 de Novembro de 2018
João Pedro Patriarca
ISEL, 6 de Outubro de 2018