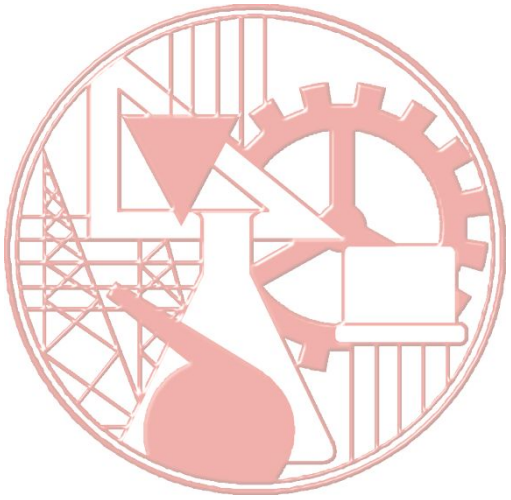
Instituto **S**uperior de **E**ngenharia de **L**isboa
Licenciatura em Engenharia Informática e de Computadores

Semestre de Inverno 2018/2019



Série 1

Sistemas Operativos

Trabalho elaborado por:

35383 – Flávio Cadete

(1)

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

sizeof(PTE) = 4 bytes = 2^2 bytes = 32 bits
sizeof(PT) = 4 KByte Page
1ºnível: 9 bits (0 > 512 = 2^9 - 1)
2ºnível: 11 bits (0 > 2047 = 2^11)
3ºnível: 11 bits (0 >2048 = 2^11)
Nº de bits de endereçamento virtual = 31 + 12 (offset) = 43 bits

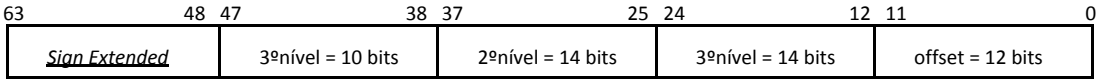
b) Sabendo que cada PTE está organizada em 27 bits para especificar uma PFN e os restantes bits para controlo, determine o número total de bits do espaço de endereçamento físico do sistema, o valor da última PFN (X, na imagem) e o número de bits de controlo de uma PTE.

size(PTE) - PFN = **32 - 27 = 5 bits de controlo**
A dimensão máxima da memória física é igual ao offset + o nº de bits do endereço físico armazenados por PTE. Com um offset de 12 bits obtemos um endereço de 39 bits.
o que nos dá um valor para a **memória física de 2^27+2^12 = 2^39 (512 Gigabytes).**

X = 0x7FFFFFFFFF

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

Pode crescer até 48 bits. Tendo 43 é possível acrescentar mais 1 bit ao 1º nivel e dois ao 2º e 3º níveis. Ficando assim com a estrutura:



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)

a)

1. The page referenced is not committed.

2. Access to a page has been attempted in violation of the permissions.

- Estes page faults devem-se a erros de programação, como por exemplo, quando um programa tenta aceder a um endereço ao qual não é suposto ter acesso, ou tenta escrever numa página *read-only*. Estes erros, denominados de ***access violation*** e que usualmente resultam no término do programa, devem-se maioritariamente ao uso de ponteiros para memória que já foi libertada ou que não foi mapeada.

3. A shared copy-on-write page was about to be modified.

- o 3º caso tem a mesma causa que o 2º (escrita em página *read-only*), mas o tratamento é feito de maneira diferente. As páginas estão marcadas como *copy-on-write*, o MMU detecta esta flag e faz uma cópia privada da página para o processo corrente (ao invés de produzir uma *access violation*). É devolvido o controlo á thread, que voltará a tentar fazer a mesma escrita, agora com sucesso e sem causar *page fault*.

4. The stack needs to grow.

- Este 4º caso ocorre quando a thread faz *push* de um valor no sua stack e essa página ainda não foi alocada. O MMU detecta que é um caso especial (enquanto existir páginas virtuais livres e reservadas do stack), devolvendo uma nova página física, limpa-a e mapeia ao processo. Quando a thread continuar a correr, irá fazer novo acesso, mas desta vez com sucesso.

5. The page referenced is committed but not currently mapped in.

- *Page fault* normal. No entanto, existem deste *page fault*. Se a página mapeada por um ficheiro, o MMU precisa de procurar na estutura em que a página foi associada à secção obj:
    - Se a encontrar noutro processo ou nas listas de *standby* ou *modified*, a página será partilhada (provavelmente com a marcação *copy-on-write*).
    - Caso contrário, o MMU vai alocar espaço para uma página física e copiar a página para o ficheiro em disco.

b)

The page-table entries in Fig. 11-31 refer to physical page numbers, not virtual page numbers. To update page-table (and page-directory) entries, the kernel needs to use virtual addresses. Windows maps the page tables and page directories for the current process into kernel virtual address space using self-map entries in the page directory, as shown in Fig. 11-32.

By making page-directory entries point at the page directory (the self-map), there are virtual addresses that can be used to refer to page-directory entries (a) as well as page table entries (b). The self-map occupies the same 8 MB of kernel virtual addresses for every process (on the x86).

c)

Superfetch - esta alternativa ao *demanding paging* consiste em preparar várias páginas para serem carregadas para memórias mesmo que ainda não tenha havido uma tentativa de acesso a elas.

Isto reduz o tempo de espera ao iniciar um aplicação ao sobrepor a leitura destas páginas com a execução do código de inicialização permitindo aos drivers do disco organizar as operações de leitura de forma a minimizar o tempo de pesquisa.

d)

- **Hard Page Faults**: o MMU não conseguiu resolver o *page fault* e por isso a página teve de ser carregada do disco para memória principal.

- **Soft Page Faults**: o MMU consegue resolver o *page fault* sem ter de carregar a página do disco (sendo por isso muito mais rápido de resolver do que Hard Page Fault):
    - quando há um acesso a uma página partilhada que já está mapeada para outro processo;
    - quando é apenas necessária uma clean page (zeroed);
    - quando uma página foi removida do working set mas voltou a ser pedida antes de ser reutilizada.

e)

Swap File - este conceito consiste em identificar um processo que não esteja a ser utilizado há algum tempo, quando é necessária memória, e escrever as suas páginas em blocos num ficheiro (swap file) por forma a minimizar as operações de I/O.

Esta técnica surgiu do facto de nas aplicações modernas não serem alocados recursos ao processo que trata do foreground de aplicações que estejam em background, não sendo necessário ter as suas páginas presentes em memória.

(3)

Experience 1	Commit #24380	WS privado #24380	WS partilhável #24380	Justificação
Ponto 1	644K	316K	2.148K	
Ponto 2	**273.892K**	328K	2.216K	Static initialized data (256Mb)
Ponto 3	-	-	**133.288K**	Reads page frames of static data
Ponto 4	-	**131.400K**	2.216K	Writes, so those pages aren't shareable no more
Ponto 5	**652K**	**324K**	2.171K	Writes more data

Experience 2	Commit #10744	WS privado #10744	WS partilhável #10744	Justificação
Ponto 1	732K	360K	2.156K	
Ponto 2	**263.300K**	312K	2.160K	Allocate 256Mb of pages (reserve and commit), RW
Ponto 3	-	**131.384K**	-	Reads page frames
Ponto 4	**132.228K**	312K	-	Deallocate 128Mb (free page frames)
Ponto 5	**640K**	-	-	Deallocate the rest of the page frames

Experience 3	Commit #14848	WS privado #14848	WS partilhável #14848	Justificação
Ponto 1	728K	360K	2.164K	*Created a file and close handle*
Ponto 2	**263.384K**	-	2.172K	Open existing file and create file mapping, RO
Ponto 3	263.292K	312K	**133.244K**	Closes handle and reads data
Ponto 4	**636K**	-	**2.176K**	Unmap file -> released memory