**I**nstituto **S**uperior de **E**ngenharia de **L**isboa

Licenciatura em Engenharia Informática e de Computadores

Semestre de Inverno - 2015/2016



Série de Exercícios 1  
Sistemas Operativos

**Trabalho elaborado por:**

**Grupo 12**

**José Pica Nº 35455**

**Pedro Duarte Nº 36832**

**João Borges Nº 36848**



|  |  |  |  |
| --- | --- | --- | --- |
| 1º Nível | 2º Nível | 3º Nível | *Offset* 13 *bits* |

Tamanho de tabela = tamanho de página / entrada da tabela de páginas

= 213 / 23 = 210

Número máximo de *bits* do endereço virtual = 3 \* número de *bits* para indexar página + *offset*

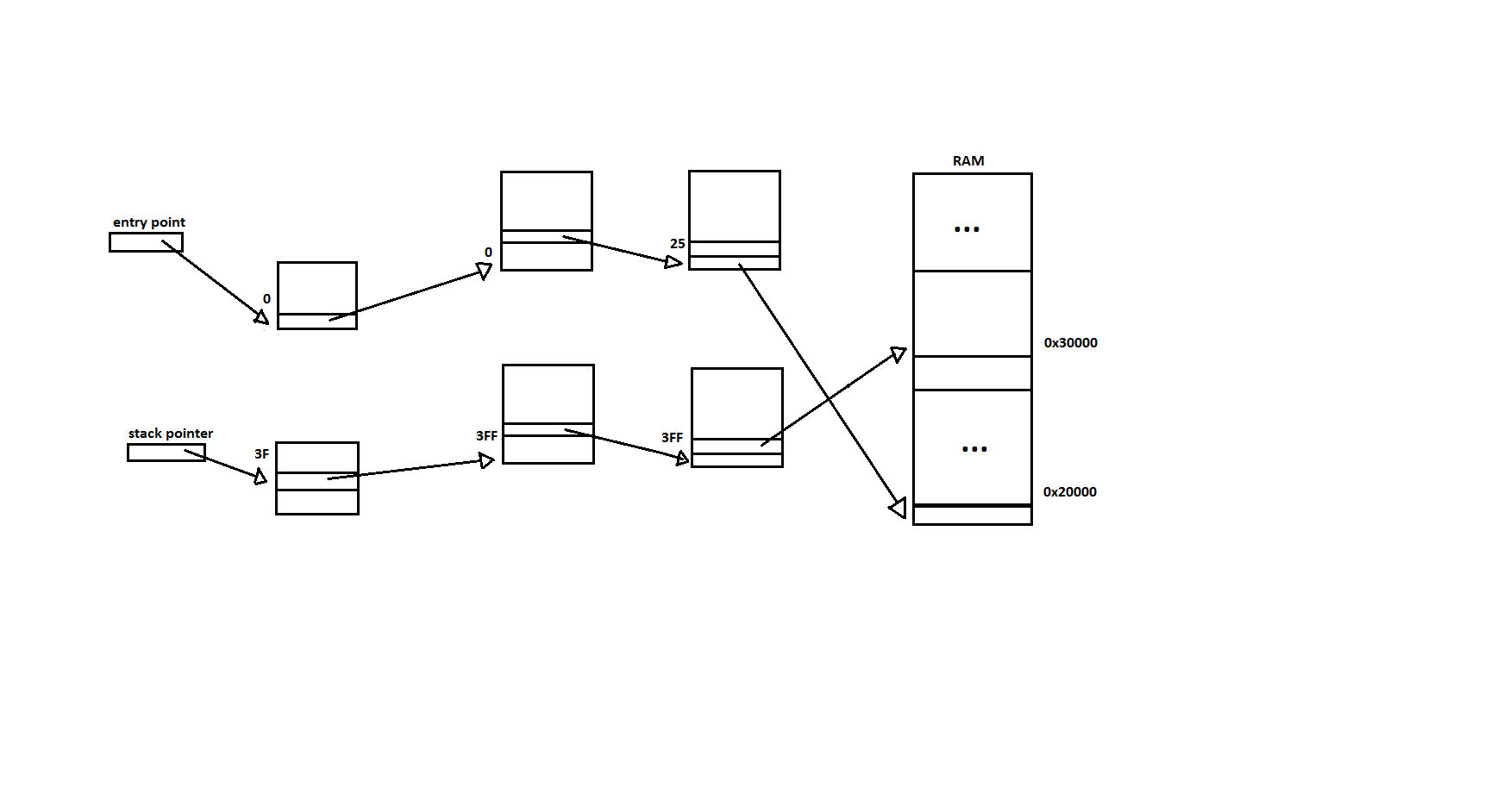
= 3 \* 10 + 13 = 43 *bits*.

* 1. Dimensão máxima da memória física =

= *Bits* endereço físico \* tamanho da página = 227 \* 213 = 240

* 1. Passagem a indexar os níveis com 21 *bits*:

|  |  |  |  |
| --- | --- | --- | --- |
| 1º Nível – 1 *bit* | 2º Nível – 10 *bits* | 3º Nível – 10 *bits* | *Offset* 13 *bits* |



* 1. Espaço endereçamento virtual = 2^48 = 256 *TB*.
  2. Espaço endereçamento físico = 2^48 = 256 *TB*.
  3. 4 *KB* com 512 entradas, 16 *KB* com 2048 entradas (2 entradas no primeiro nível) ou 64 *KB* com 8192 entradas (64 entradas no primeiro nível).
  4. Dimensão das entradas das tabelas = 64 *bits* = 8 *bytes*.
  5. Para páginas de 4 *KB* e 16 *KB* - 4 níveis de tradução, para as de 64 *KB* – 3 níveis de tradução.

1. Esta afirmação é verdadeira, pois está demonstrado na resposta ao número de níveis de tabelas por dimensão de página da alínea anterior. Os potenciais problemas que isto pode trazer são aumento número de entradas da tabela de página leva a existir mais bits de endereço menos significativos necessários para tratar uma página, estes bits não requerem tradução, e também maior fragmentação de dados, possibilidade de ter em memória código pouco usado e aumento dos tempos de leitura ou escrita.
   1. A memória adicional provém da lista standby, que não é contabilizada na contagem de memória livre, mas é contabilizada na memória disponível. A vantagem de ter pouca memória livre é a facilidade de aceder a memória já alocada. Um programa pode manter informação alocada para a usar mais tarde em vez de a libertar e voltar a alocar.
   2. A função *UnMapViewOfFile*, pois remove a entrada do *working* *set* para cada página alocada reduzindo o tamanho do *working* *set* e liberta da memória física esse contagem de memória partilhada. E também a função *VirtualFree*, pois liberta uma região alocada anteriormente.
   3. O *Windows* para resolver o problema utiliza a política de carregamento de páginas por *Demand*-*paging*. Nesta política, as páginas são carregadas em memória física apenas quando são acedidas. As páginas partilhadas são marcadas como *read*- *only* (e *copy*-*on*-*write* usando um dos bits disponíveis na *PTE*). Uma tentativa de escrita gera exceção com as seguintes consequências: criação de uma cópia para o *working* *set* desse mesmo processo, se a página estiver a ser partilhada, e colocação da página *read*/*write*. Consegue-se uma redução da memória física utilizada, pois é possível partilhar código e dados iniciados enquanto não ocorrerem alterações.
2. Neste programa que, dado o identificador de um processo, apresenta as *DLLs* correntemente mapeadas pelo processo, em primeiro é necessário utilizar a função O*penProcess* com as *flags* de acesso *PROCESS*\_*QUERY*\_*INFORMATION* e *PROCESS*\_*VM*\_*READ*, para ter um *handle* que representa o processo. Para iterar sobre as regiões de memória do *working* *set*, é utilizada a função *VirtualQueryEx*, passando como argumentos: o *handle*, o endereço base zero e a estrutura *MEMORY*\_*BASIC*\_*INFORMATION*, sendo preenchida por esta função. Pelo estado presente na estrutura, só interessa analisar regiões com o estado *COMMITED*. Através da função *GetModuleFileNameEx*, permite obter o *pathname* dos ficheiros mapeados. Esta função recebe um *handle*, um endereço de memória onde a *dll* foi mapeada e um *array* de *charg* que irá conter o *pathname*.