1)

a)

Hipótese 1:

12 bits de offset -> 28 bits para 3 tabelas -> 8/10/10 (+ 12 offset)

12 bits offset -> Páginas de 2^{12} bytes = 4096 Bytes = 4KB

10 bits de endereçamento numa Tabela de Entradas = 1024 entradas (de 8 Bytes cada uma) -> 8192 Bytes (2 páginas, ou seja não cumpre o requisito "dimensão máxima de uma página)

É nos impossível ter páginas com apenas 9 bits de endereçamento (9 + 9 + 9 = 27 < 28) a não ser que não se utilize um dos bits por isso Páginas de 4096 Bytes não são suficientes.

Hipótese 2:

13 bits de offset -> 27 bits para 3 tabelas -> 7/10/10 (+13 offset)

13 bits offset -> Páginas de 2^{13} bytes = 8192 bytes = 8KB

10 bits de endereçamento numa Tabela de Entradas tal como na hipótese anterior = 8KB (o mesmo tamanho que uma página nesta hipótese).

Esta solução é viável, no entanto apenas usamos 7 bits para endereçamento na Tabela de Entradas de 1º nível o que significa que só podemos endereçar para 2^7 entradas -> 128 entradas -> 1KB (1/8 de uma tabela) o que pode causar alguma fragmentação da memória.

Considerando um espaço de endereçamento físico igual ao virtual dispomos de 40 bits para endereçar na memória principal. Sendo assim, e sabendo que temos usamos 13 bits para o offset, sabemos então que são usados 40 - 13 = 27 bits usados para especificar o endereço físico em cada PTE.

b)

Utilizar 20 bits para flags em cada PTE tendo em conta que cada PTE contêm 8 Bytes (64 bits) significa que possivelmente temos 44 bits disponíveis para endereçamento. Juntado isto aos 13 bits de offset resulta num espaço de endereçamento indexável com 57 bits = 2^{57} bytes = 144,115188 PetaBytes.

c)

Não necessariamente, uma vez que existe uma cache associada á arquitetura se o dado estiver presente na cache não é necessário os quatro acessos a memória, se não estiver presente então trazido para cache em conjunto com os dados vizinhos.

2)

a)

1)A trap to the kernel occurs.

2)The kernel builds a machine-independent descriptor telling what happened.

3)The kernel passes the descriptor to the memory-manager part of the executive.

4)The memory-manager checks the access for validity.

5)case the faulted page falls within a commited region, it looks for the address in the list of VADs and finds (or creates) the process page-table entry.

case the faulted page be a shared page, the memory manager creates a copy of it, being now allowed to be edited.

b)

A alternativa usada ao "demand paging" tem o nome de "SuperFetch", que consiste em preparar páginas para o programa em execução sem este as ter pedido, colocando as páginas prontas para utilização na stand-by list, que tem tempos de acesso mais rápidos comparando com acessos ao disco, fazendo com que a rapidez de execução seja elevada, visto que a probabilidade de "page-miss" é baixa.

c)

A vantagem das large-pages reside no facto de quando um programa pede memória de valores elevados, suponhamos 1 Mib, a quantidade de páginas que necessitam de ser atribuidas, mais a memória usada exclusivamente para gerir os directorios das páginas é bastante elevado, o que torna a existência de páginas de maior dimensão uma solução eficaz.

d)

"Hard page faults" ocorrem quando uma page necessária ao programa se encontra em memória física, sendo o tempo necessário para a sua obtenção bastante superior comparando a uma página em "Soft page fault", sendo no último caso uma página que se encontra já em memória não-física (RAM), mas encontra-se em falta no working-set , devido ao facto de não estar marcada na MMU como carregada.

e)

A diferença geral entre "swap file" e "paging file" consiste no facto de page-file mover pages de um programa da memória de sistema para o "paging file" ficheiro de sitema onde se encontram as páginas que estão em stand-by, libertando memória para outros programas usarem. Swap-files por seu lado consiste em mover TODAS as pages de um ficheiro (não apenas parte delas como no page-file) para o ficheiro de sistema. Este último é usado em sistemas com menos capacidade de memória que um sistema normal, geralmente dispositivos móveis.

4)

a)

A vantagem das bibliotecas dinâmicas de sistema terem endereços bem definidos é que esta estratégia permite uma maior eficiência no tempo de carregamento e na gestão de memória.

b)

O loader procede à realocação das mesmas para outros endereços (isto ocorre em tempo de carregamento). Para evitar esta realocação em tempo de carregamento é possível utilizar o utilitário rebase para mudar os endereços antecipadamente e assim evitar intersecções.

5)

Ao iniciar o programa é reservado espaço (sem commit). Esta alteração no estado da memória do processe é identificada no gráfico que demonstra um aumento do espaço virtual. Este permanece assim até ser libertado. O ficheiro de paginação só demonstra um aumento quando é feito o commit da região reservada anteriormente pelo processo e também só retorna ao estado anterior quando este mesmo commit é “desfeito”. O gráfico demonstra depois uma grande subida nos Page Fault, estes são gerados ao escrever no espaço anteriormente reservado e commited que obriga a guardar no Private Working Set.