

Fundamentos de Sistemas de Operação
2º Teste, 8 de Janeiro de 2021

QUESTÕES DE ESCOLHA MÚLTIPLA — VERSÃO A

1) Numa arquitetura Intel x86, em que o SO suporta memória virtual por paginação-a-pedido (*demand paging*), a dimensão máxima possível (teórica) para o espaço de endereçamento de um processo é determinada:

- a) pela dimensão que resulta da soma da memória física instalada com o espaço disponível para swapping (a.k.a. paging) em disco
- b) pela dimensão dos registos genéricos (eax, ebx, ecx, ...) usados para operações aritméticas e lógicas
- c) pela dimensão dos registos de endereçamento (esp, ebp, eip) do CPU, podendo contudo ser limitada também pelo próprio SO ● ✓
- d) pelo espaço disponível para armazenar o mapa de tradução de endereços dos processos em execução
- e) pela capacidade da cache do Translate Lookaside Buffer (TLB)

2) Qual das alíneas considera apresentar as características mais importantes da técnica de gestão de memória designada como “memória virtual por paginação-a-pedido” (*demand paging*)?

- a) boa utilização da RAM instalada, pode correr processos cuja dimensão do espaço de endereçamento (EE) excede a RAM instalada, protecção das diferentes regiões do EE deficiente (permite intrusão de hackers e/ou vírus), o desempenho só é bom se se usarem discos HDD em RAID-0 para a área de paginação
- b) boa utilização da RAM instalada, pode correr processos cuja dimensão do espaço de endereçamento (EE) excede a RAM instalada, protecção das diferentes regiões do EE de acessos indevidos deficiente (só protege as páginas de código e de constantes), bom desempenho se os acessos exibirem boa localidade de referência espacial e temporal ✗
- c) boa utilização da RAM instalada, pode correr processos cuja dimensão do espaço de endereçamento (EE) excede a RAM instalada, protecção das diferentes regiões do EE de acessos indevidos, mas o desempenho só é bom se se usarem discos SSD para a paginação
- d) boa utilização da RAM instalada, pode correr processos cuja dimensão do espaço de endereçamento (EE) excede a RAM instalada, protecção das diferentes regiões do EE de acessos indevidos, bom desempenho se os acessos exibirem boa localidade de referência espacial e temporal ● ✓
- e) má utilização da RAM instalada (por fragmentação interna excessiva), pode correr processos cuja dimensão do espaço de endereçamento (EE) excede a RAM instalada, protecção das diferentes regiões do EE de acessos indevidos, fraco desempenho se os acessos exibirem boa localidade de referência espacial e temporal

3) Um processo, para abrir um ficheiro com permissões 000 (exibidas no `ls -l` pela “máscara” -----) e efectuar um `read()` necessita de:

- a) qualquer processo pode abrir e ler o ficheiro
- b) tendo conseguido abrir, qualquer processo pode ler o ficheiro desde que a página onde reside a chamada de sistema `read()` tenha o atributo “read” activo
- c) o UID do processo é root (i.e., zero) ou corresponde ao UID do dono (owner) do ficheiro ou a um UID pertencente ao grupo com acesso ao ficheiro
- d) o processo estiver a correr em modo supervisor (ou kernel)
- e) o UID do processo é root (i.e., zero) ● ✓

4) Numa directoria com permissões 123 (exibidas no `ls -l` pela "máscara" `--x-w--wx`), um processo não-root lançado pelo utilizador que é dono (*owner*) da directoria,

- a) consegue atravessar a directoria ● ✓
- b) consegue atravessar a directoria, e listar e abrir ficheiros nela existentes
- c) consegue atravessar a directoria e listar, criar, apagar e abrir ficheiros nela existentes
- d) consegue atravessar a directoria e listar, criar e apagar ficheiros nela existentes
- e) não tem nenhum acesso à directoria

5) A programação (de periféricos) por **espera activa** é **essencialmente** caracterizada por o *driver* de um periférico,

- a) esperar que termine a operação de I/O, e transferir os dados por DMA
- b) ler continuamente o registo de estado do periférico para determinar quando termina a operação de I/O ● ✓
- c) esperar que termine a operação de I/O, e activar um sinal de interrupt
- d) esperar que termine a operação de I/O, e depois executar a instrução INT
- e) esperar que termine a operação de I/O, e transferir os dados por I/O programado (IN e OUT) ✗

✗ 6) Considere um grupo de 3 discos SSD absolutamente idênticos sobre os quais foi criado um volume RAID-0. Sabendo que para um disco individual acedido sequencialmente em leitura a latência é L_i e a largura de banda é B_i , para o grupo ter-se-à, **aproximadamente** (sendo, respectivamente, L_G a latência e B_G a largura de banda do volume)

- a) $L_G = L_i$ e $B_G = 3 \times B_i$
- b) $L_G = 3 \times L_i$ e $B_G = 3 \times B_i$
- c) $L_G = L_i / 3$ e $B_G = 3 \times B_i$
- d) $L_G = L_i / 3$ e $B_G = B_i$
- e) Nenhuma das anteriores, a tecnologia RAID só é aplicável a discos HDD (magnéticos)

7) Depois de uma falha de energia ou de um *crash* do sistema de operação, num computador com um volume RAID-5 formatado com um determinado sistema de ficheiros (SF) que utiliza técnicas de *journaling*, e com o volume em uso (montado) para leitura e escrita na altura da falha,

- a) Não é necessário recuperar da falta/falha usando o *journal* porque um volume RAID-5 tem redundância e recupera faltas/falhas automaticamente
- b) É necessário aplicar o *journal* para recuperar da falta/falha, o que envolve todos os discos do volume
- c) É necessário aplicar o *journal* para recuperar da falta/falha, mas tal só envolve o disco que contém o superbloco
- d) É necessário aplicar o *journal* para recuperar da falta/falha, mas tal só envolve discos que contenham directorias e/ou i-nodes
- e) Nenhuma das anteriores, a técnica de *journaling* não é aplicável a volumes RAID

8) Uma diferença fundamental entre máquinas virtuais (VMs) e *containers* é,

- a) um container pode executar um sistema de operação (SO) diferente do SO do host, uma VM não pode ✓
- b) uma VM pode executar um sistema de operação (SO) diferente do SO do host, um container não pode ●
- c) uma VM pode executar aplicações escritas em qualquer linguagem (e.g., C, Java, Python), um container só pode executar código originalmente escrito em C
- d) uma VM pode executar múltiplos processos, um container só executa um único processo
- e) A tecnologia de containers é unicamente suportada sobre SOs Unix-like, a de virtualização sobre estes e outros, como Windows e MacOS

9) Em Linux (e outros SOs Unix-like), quando dois processos, depois de abrirem um mesmo ficheiro, executam operações de leitura e/ou escrita que acedem concorrentemente a regiões sobrepostas (por ex., um `read()` lê do `offset` 100 a 200 enquanto um `write()` escreve do `offset` 150 ao 220) desse ficheiro,

- a) as operações retornam sempre erro, qualquer que seja a forma como os ficheiros são acedidos
- b) as operações têm sempre sucesso, qualquer que seja a forma como os ficheiros são acedidos
- c) se ambos os processos usarem trincos opcionais (advisory locking) e os trincos colocados pelos processos forem incompatíveis, as operações retornam sempre erro
- d) se ambos os processos usarem trincos (de qualquer tipo, advisory ou mandatory) e os trincos colocados pelos processos forem incompatíveis, as operações ou retornam ambas erro ou um dos processos é bloqueado ● ✓
- e) se ambos os processos usarem trincos (de qualquer tipo, advisory ou mandatory) e os trincos colocados pelos processos forem incompatíveis, as operações ou retornam ambas erro ou ambos os processos são bloqueados

10) Em Linux (e outros SOs), um *driver* de disco é um módulo de software

- a) que é parte do SO, corre no CPU interno do controlador do disco em modo supervisor, e é chamado pelo sistema de ficheiros para executar as operações de `open()`, `close()`, `read()`, `write()` e `stat()`
- b) que é parte do SO, corre no CPU em modo supervisor, e é chamado pelo sistema de ficheiros quando precisa de ler ou escrever blocos, ou interrogar/alterar o estado do disco ✓
- c) que é parte do SO, corre no CPU em modo utilizador, na maior parte do tempo, e é chamado pelo sistema de ficheiros quando precisa de ler ou escrever blocos, ou interrogar/alterar o estado do disco; o driver necessita apenas do modo supervisor quando trata das interrupções ✗
- d) que é parte da biblioteca de funções do sistema, corre no CPU em modo utilizador, na maior parte do tempo, e é chamado pelo sistema de ficheiros quando precisa de ler ou escrever blocos, ou interrogar/alterar o estado do disco; o driver necessita apenas do modo supervisor quando trata das interrupções
- e) que é parte da biblioteca de funções do sistema `open()`, `close()`, `read()`, `write()` e `stat()`, corre no CPU em modo utilizador, na maior parte do tempo, apenas quando precisa de ler ou escrever blocos, ou interrogar/alterar o estado do disco muda para modo supervisor para realizar essas operações

11) As instruções `IN`, `OUT`, `CLI` (desactivar atendimento de interrupções) e `STI` (reactivar atendimento de interrupções) são usadas na manipulação de periféricos. Um periférico, pode ser partilhado entre processos do mesmo ou de diferentes utilizadores e acedido concorrentemente ou em exclusividade. Colocam-se assim, entre outras, as seguintes questões: i) confidencialidade (a um utilizador pode não ser permitido aceder a parte/totalidade dos dados “contidos” no periférico); ii) integridade (a má programação não deve corromper estruturas de dados e comprometer o acesso ao periférico). Assim, as instruções `IN`, `OUT`, `CLI` e `STI` são

- a) `IN` e `OUT`: privilegiadas porque o acesso a periféricos só pode ser permitido ao nível do núcleo do SO, senão poderiam comprometer-se as condições (i) e (ii). `CLI` e `STI`: privilegiadas porque a incorrecta desactivação ou reactivação do atendimento de interrupções põe em risco o funcionamento do próprio SO
- b) `IN` e `OUT`: privilegiadas porque o acesso a periféricos só pode ser permitido ao nível do núcleo do SO, senão poderiam comprometer-se as condições (i) e (ii). `CLI` e `STI`: não-privilegiadas porque a desactivação/reactivação do atendimento de interrupções não põe em risco o funcionamento do próprio SO, mas apenas dos periféricos que usem interrupções, e estes podem sempre ser configurados para não as usar.
- c) `IN` e `OUT`: não-privilegiadas porque o acesso a periféricos pode ser permitido em modo utilizador, com funções de biblioteca, cuja correcção garante o não-comprometimento de (i) e (ii). `STI`: não-privilegiada porque permitir o atendimento de interrupções é uma operação “inofensiva”. `CLI`: privilegiada porque uma desactivação incorrecta do atendimento de interrupções põe em risco o funcionamento do próprio SO
- d) `IN` e `OUT`: não-privilegiadas porque o acesso a periféricos pode ser permitido em modo utilizador, com funções de biblioteca, cuja correcção garante o não-comprometimento de (i) e (ii). `CLI` e `STI`: privilegiadas porque uma incorrecta desactivação ou reactivação do atendimento de interrupções põe em risco o funcionamento do próprio SO
- e) `OUT` não-privilegiada porque a escrita em periféricos pode ser permitida em modo utilizador, com funções de biblioteca, cuja correcção garante o não-comprometimento de (i) e (ii). `STI`: não-privilegiada porque permitir o atendimento de interrupções é uma operação “inofensiva”. `IN`: privilegiada porque em leitura não há forma de, com funções de biblioteca, garantir (i). `CLI`: privilegiada porque a desactivação do atendimento de interrupções põe em risco o funcionamento do próprio SO



12) A estrutura de dados que, nos sistemas de ficheiros nativos (i.e., desenhados especificamente para serem usados num determinado SO, como o `ext2`, no Linux) de SOs *Unix-like*, contém informação sobre as diferentes regiões que constituem um volume/disco formatado designa-se por

- a) Zona de bitmaps
- b) Tabela de i-nodes
- c) Zona de metadados
- d) superbloco
- e) root



13) As estruturas de dados designadas por *bitmaps* são usadas em sistemas de ficheiros para

- a) indicar em que blocos do disco estão os superblocos de backup
- b) indicar em que blocos do disco estão os i-nodes
- c) gerir outras estruturas de dados indicando se estas estão livres (ou “vazias”) ou ocupadas (ou “cheias”), ou são válidas, ou inválidas, etc.
- d) indicar, para cada ficheiro, quais os blocos que lhe estão atribuídos e se estes estão livres ou ocupados
- e) Nenhuma das anteriores



14) Nos sistemas de ficheiros (SF) tipicamente usados em SOs Unix-like (como o `ext2`, no Linux), os *i-nodes* são usados para

- a) indicar em que blocos do disco começa e acaba um ficheiro
- b) indicar, para cada ficheiro, quais os blocos que lhe estão atribuídos e se estes estão livres ou ocupados 
- c) armazenar toda a informação que o SF guarda sobre um ficheiro
- d) armazenar toda a informação que o SF guarda sobre um ficheiro, excepto o seu nome e dimensão
- e) armazenar toda a informação que o SF guarda sobre um ficheiro, excepto o seu nome 


15) Depois de uma falha de energia ou de um *crash* do sistema de operação, num computador com um volume formatado com um determinado sistema de ficheiros (SF) e em uso (montado) para R/W na altura da falha, a consistência do SF

- a) tem de ser verificada  
- b) não tem de ser verificada
- c) só tem de ser verificada se se tratar de um SF com journaling
- d) só tem de ser verificada se se tratar de um SF sem journaling
- e) só tem de ser verificada se se tratar de um SF nativo do SO em causa (por exemplo, `ext2` num SO Linux ou NTFS num SO Windows)

16) Num sistema de ficheiros em que a atribuição de espaço (*data blocks*) aos ficheiros é de tipo contíguo,



- a) a dimensão do ficheiro não tem de ser especificada na altura da criação. Este tipo é o que, de entre todos, oferece melhor velocidade de acesso
- b) a dimensão do ficheiro não tem de ser especificada na altura da criação. Este tipo é o que, de entre todos, oferece pior velocidade de acesso
- c) a dimensão do ficheiro tem de ser especificada na altura da criação, e há reserva prévia dos blocos; o ficheiro não pode (geralmente) crescer (para além da reserva inicial) depois de criado. Este tipo é o que, de entre todos, oferece melhor velocidade de acesso  
- d) a dimensão do ficheiro tem de ser especificada na altura da criação, e há reserva prévia dos blocos; o ficheiro não pode (geralmente) crescer (para além da reserva inicial) depois de criado. Este tipo é o que, de entre todos, oferece pior velocidade de acesso
- e) a dimensão do ficheiro tem de ser especificada na altura da criação, mas não há reserva prévia de blocos; o ficheiro pode crescer depois de criado. Este tipo é o que, de entre todos, oferece melhor velocidade de acesso

17) ANULADA

 18) Para gerir uma tabela com N i-nodes, a dimensão de um *bitmap* deve ser b **bits** (note bem: b minúsculo) e ocupar X blocos, em que cada bloco tem B **bytes** de dimensão. O cálculo de b e de X é dado por [Nota: a função $\text{ceil}(\text{float } \text{arg})$, *ceiling* ou *tecto*, calcula o menor inteiro não inferior ao argumento arg]

- a) $b = N * 8$ e $X = \text{ceil}(b / (\text{float})B)$
- b) $b = N / 8$ e $X = \text{ceil}(b / (\text{float})B)$
- c) $b = N * 8$ e $X = \text{ceil}(\text{ceil}(b/(\text{float})8) / (\text{float})B)$
- d) $b = N$ e $X = \text{ceil}(\text{ceil}(b/8) / (\text{float})B)$
- e) Nenhuma das anteriores

19) Na memória virtual por paginação-a-pedido (*demand paging*) a Tabela de Páginas de um processo tem numerosas entradas; cada entrada (PTE – *Page Table Entry*) contém, **essencialmente**, a seguinte informação:

- a) Número da página e da frame, e bits de: protecção, “presente em RAM”, “página modificada” e “página acedida” 
- b) Número da frame e bits de: protecção, “presente em RAM”, “página modificada”, “posição no TLB”
- c) Número da frame e bits de: protecção, “presente em RAM”, “página modificada” e “página acedida” 
- d) Número e offset da página e da frame, e bits de: protecção, “presente em RAM”, “página modificada” e “página acedida”
- e) Número e offset da página e da frame, e bits de: protecção, “presente em RAM”, “página modificada” e “página acedida” e “posição no TLB”

20) Num sistema de operação, muitas das técnicas e opções utilizadas no seu “desenho” (*design*) ou implementação são motivadas pela necessidade de acelerar a execução de aplicações e diminuir a latência de acesso aos dados. Assim, é importante conhecer as ordens de grandeza (valores típicos) de tempos de: execução de instruções sobre registos do CPU (t_{CPUins}), acesso a RAM (t_{RAM}), acesso a um bloco (não *cached*) de disco num HDD (t_{HDD}) e num SSD (t_{SSD}), tradução de endereço virtual/físico num TLB aquando de um hit (t_{HIT}) ou miss (t_{MISS}), etc., pois só assim se compreende o SO. Considerando as seguintes unidades **ns** (nano-segundos), **μs** (micro-segundos) e **ms** (mili-segundos), qual das opções abaixo representa valores credíveis?

- a) t_{CPUins} : dezenas de ns; t_{RAM} : dezenas de ns; t_{HDD} : dezenas de ms; t_{SSD} : s; t_{HIT} : dezenas de ns; t_{MISS} : dezenas de s
- b) t_{CPUins} : ns; t_{RAM} : dezenas de ns; t_{HDD} : ms; t_{SSD} : dezenas/centenas de s; t_{HIT} : ns; t_{MISS} : ns
- c) t_{CPUins} : ns; t_{RAM} : ms; t_{HDD} : ms; t_{SSD} : dezenas/centenas de s; t_{HIT} : ns; t_{MISS} : dezenas de s
- d) t_{CPUins} : ns; t_{RAM} : dezenas de ns; t_{HDD} : ms; t_{SSD} : ms; t_{HIT} : ns; t_{MISS} : dezenas de s
- e) t_{CPUins} : ns; t_{RAM} : dezenas de ns; t_{HDD} : ms; t_{SSD} : dezenas/centenas de s; t_{HIT} : ns; t_{MISS} : dezenas de s 