

# Fundamentos de Sistemas de Operação MIEI 2014/2015

2º Teste, 9 de Dezembro 2014, 2 horas – versão A

Número \_\_\_\_\_ Nome \_\_\_\_\_

*Sem consulta e sem esclarecimento de dúvidas; indique explicitamente nas suas respostas eventuais hipóteses colocadas*

*A detecção de fraude durante a realização ou correcção do teste implica, no mínimo, a reprovação à cadeira.*

**Questão 1 (2.0 valores)** Considere um sistema operativo em que o algoritmo de escalonamento é Round Robin com uma única fila READY e uma fatia fixa de tempo  $T_f$ . Suponha que existem no sistema 20 processos *cpu-bound* e que não é aceitável que um processo esteja mais de 500 milisegundos sem que lhe seja atribuído CPU. Faça, justificando, uma estimativa para um valor de  $T_f$  que permitiria assegurar o requisito anterior. O valor que calculou é aceitável se considerarmos um tempo de comutação entre processos ( $T_c$ ) de 10 microsegundos?

**Questão 2 (2.0 valores)** Considere um sistema operativo que utiliza um algoritmo de escalonamento com uma única fila de processos prontos e uma fatia fixa de tempo  $T_f$ . Admita que dois processos A e B são os únicos no sistema, tendo entrado na fila de processos prontos no instante de tempo 0, pela ordem A, B. Quando um processo passa de WAITING para READY é colocado à cabeça da fila READY. Os processos realizam as seguintes ações:

- Processo A: Usa o CPU durante 25 milisegundos e termina
- Processo B: Faz 3 vezes um conjunto de acções em que usa o CPU durante 2 milisegundos e de seguida faz uma operação de entrada/saída que demora 8 milisegundos; após o ciclo, termina

Supondo que a troca de contexto tem uma duração desprezável, diga, justificando, em que instante de tempo terminam os processos A e B para um algoritmo de escalonamento *Round Robin* com uma fatia de tempo de 5 ms. Justifique a sua resposta fazendo um diagrama em que no eixo dos XX está o tempo e sobre ele estão duas linhas, uma para o processo A e outra para o processo B em que seja possível saber, em cada momento, em que estado (RUNNING, READY, WAITING) estão os processos A e B.

**Questão 3 (2.0 valores)** Considere um escalonador de processos que usa múltiplas filas READY ordenadas por prioridade (*MLFQ Multi-level Feedback Queue*). A prioridade máxima P corresponde ao valor numérico 0 e a prioridade mínima ao valor numérico 199. O escalonador recalcula a prioridade P de um processo de 1 em 1 segundos de acordo com a fórmula

$$P = (V / 2) + 60$$

em que V é directamente proporcional ao tempo de CPU atribuído recentemente ao processo. Diga o que acontece à prioridade de um processo *CPU-bound* e explique porque é que é necessário dividir V por 2 a cada segundo que passa.

**Questão 4 (3 valores)** Um sistema em que os endereços virtuais têm 32 bits, usa uma MMU que suporta páginas com dimensão de 1 Mbyte ( $2^{20}$ ).

- a) Para as condições acima referidas, pretende-se fazer uma simulação do número de faltas de página correspondentes a uma dada sequência de endereços virtuais. O primeiro passo é obter o número da página virtual *npv* e o deslocamento *desc* correspondente a um dado endereço virtual *ev*. Para esse efeito escreveram-se as seguintes linhas de código C que se pretende que complete:

#define PAGE\_SIZE \_\_\_\_\_

npv = \_\_\_\_\_;

desc = \_\_\_\_\_ ;

- b) Para um dado processo em execução, as primeiras entradas da tabela de páginas são as seguintes:

Nº página virtual	Nº página física (em base 16)
0	0xBAD
1	0xABC
2	Inválida
3	0xBEE

As outras entradas são todas inválidas. Indique, justificando, os endereços físicos que correspondem aos endereços virtuais (em base 16) abaixo indicados. Responda “Endereço Inválido” se o endereço virtual for inválido.

0x00000000

0x00222001

0x10001001

0x0033BA11

**Questão 5 (3 valores)** Considere um sistema de operação e um CPU+MMU que suportam paginação a pedido.

- a) Explique como é que é possível executar programas cuja imagem é superior ao tamanho da memória física disponível.
- b) Para este tipo de sistema, diga como funciona a ligação dinâmica de bibliotecas. Relacione esse funcionamento com a possibilidade que existe de mapear ficheiros no espaço de endereçamento de um processo (como faz a chamada ao sistema *mmap()* do UNIX).
- c) Neste contexto, diga o que é uma situação de *thrashing*. Supondo que dispõe permanentemente dos seguintes indicadores sobre o desempenho do sistema
- Taxa de ocupação do CPU (PCPU)
  - Número de operações sobre o disco de paginação (NIOP)
  - Número de páginas físicas livres (NFL)

que combinações de valores de (PCPU, NIOP e NFL) permitiriam concluir que o sistema estava numa situação de *thrashing*?

**Questão 6 (2 valores)** Suponha uma variante do sistema de ficheiros UNIX em que os blocos têm 8 Kbytes e os endereços de blocos têm 4 bytes. Cada i-node tem:

- 12 endereços directos
- 1 endereço indirecto - isto é que contém o endereço de um bloco com endereços
- 1 endereço duplamente indirecto - contém endereços de blocos que contêm endereços de blocos com endereços

Diga, justificando, qual é máxima dimensão em blocos de um ficheiro neste sistema.

**Questão 7 (3 valores)** O sistema de ficheiros XPTO usa uma forma de designação dos ficheiros igual à do UNIX - */dir1/dir2/dir3/.../ficheiro*. Estão disponíveis as chamadas ao sistema tradicionais sobre ficheiros (*open / read / write / close*). Na zona dos meta-dados de um disco formatado com o sistema de ficheiros XPTO há

- Um *super bloco* que contém a dimensão do disco e o número do bloco onde se encontra a directoria raiz.
- Um *bitmap* de ocupação de blocos
- Uma *tabela geral de ficheiros* (TGF) que tem uma entrada para cada ficheiro existente no disco e em que cada entrada tem os campos:
  - Em uso/livre
  - Comprimento do ficheiro em bytes
  - Lista de blocos onde estão os dados do ficheiro
- As directorias são ficheiros que podem ser vistos como um vector de registos em que cada registo tem os seguintes campos:
  - Tipo da entrada: ficheiro, directoria, livre
  - Nome (cadeia de caracteres)
  - Índice na TGF onde está a informação sobre o ficheiro.

Para o sistema de ficheiros XPTO pretende-se que descreva em detalhe como seriam implementadas as chamadas ao sistema referidas nas alíneas abaixo. Repare que o que se pretende é que indique o que é feito em termos de preenchimento das tabelas do sistema operativo (quer as globais quer as associadas a cada processo) e as leituras e escritas que são feitos na zona de meta-dados do disco e não o que fazem as chamadas ao sistema do ponto de vista do programador.

a) Abertura para escrita de um ficheiro que se supõe que já existia antes mas que está vazio

*f* = *open*( *"/dir1/f"*, *O\_WRONLY* )

b) Seja *BLOCK\_SIZE* o tamanho de um bloco do disco. Escrita de *BLOCK\_SIZE* bytes no final do ficheiro e em que o *offset* corrente é um múltiplo de *BLOCKSIZE*.

*nw* = *write*( *f*, *buf*, *BLOCKSIZE* )

c) Fecho do ficheiro *close*(*f*).

**Questão 8 (3 valores)** Considere o sistema de ficheiros descrito na pergunta 7. Suponha que o mapa de blocos ocupa apenas um bloco no disco e que cada directoria também ocupa no máximo um bloco. Imagine que vai remover o ficheiro `/dir1/f7` que se supõe que existe.

- a) A operação referida implica a escrita em mais do que um bloco do disco. Que blocos são esses?
- b) O que aconteceria se houvesse uma falha de energia ou um erro no sistema operativo que se manifestasse pela terminação abrupta do sistema operativo enquanto a operação de remoção do ficheiro estava a decorrer. Quais seriam as consequências para o sistema de ficheiros?
- c) Suponha que se pretendia corrigir o erro anterior da próxima vez que se fizesse *mount( )* do disco. O que haveria a fazer nessa ocasião? Refira apenas o que fazer em relação a remoções de ficheiros que não correram bem.