Fundamentos de Sistemas de Operação MIEI 2013/2014

2º Teste, 7 Dezembro 2013, 2 horas – versão A

Número_____ Nome _____

Sem consulta e sem esclarecimento de dúvidas; indique eventuais hipóteses assumidas nas sua respostas.

Questão 1 (2.0 valores) Um dos problemas da gestão da memória central em ambientes que suportam um número variável de processos carregados em memória é a *fragmentação externa*.

a) Explique em que consiste esse problema

b) Se um sistema tiver uma MMU com registo base e um registo limite está sujeito a este problema? Justifique.

c) E se a MMU for baseada em páginas? Justifique.

a) O problema ocorre porque a imagem do processo tem de ser carregada em uma (ou mais) zonas contíguas de memória. À medida que os processos são criados e terminam e a memória livre deixa de estar contígua e fica subdividida em múltiplos pequenos fragmentos. Isto pode fazer com que não se consiga carregar em RAM um processo cuja imagem tem I bytes, sendo a quantidade de RAM livre superior a I bytes.

b) A MMU com registo base e registo limite suporta precisamente o carregamento numa partição contígua que é o caso de a)

c) Ao introduzir um sistema baseado em páginas, desaparece a necessidade de carregar a imagem contiguamente em memória. Assim sendo, para carregar um processo cuja imagem tem I bytes apenas é preciso encontrar I/ tamanho_página páginas livres na RAM independentemente de serem ou não contíguas.

Questão 2 (2.0 valores) Um sistema em que os endereços virtuais têm 32 bits, usa uma MMU que suporta páginas com dimensão 64 KBytes (2^16). 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	0xCAFE
1	0xDEAD
2	Inválida
3	0xBEEF

As outras entradas são todas inválidas. Indique, justificando, os endereços físicos que correspondem aos seguintes endereços virtuais. Responda "Endereço Inválido" se o endereço virtual for inválido.

a) 0x00000000

b) 0x00022001

c) 0x10001001

d) 0x0003BA11

a) PV=0x0000 , D = 0x0000 → PF = Tab_Apg[0x0000] = 0xCAFE, D=0x0000 → Endereço físico 0xCAFE0000

b) PV=0x0002 → nº página virtual inválido *c) PV=0x1000 → nº página virtual inválido*

d) PV=0x0003 D = 0xBA11 → PF = Tab_Apg[0x0003] = 0xBEEF, D=0xBA11 → Endereço físico 0xBEEFBA11

Questão 3 (2.5 valores) Considere um sistema que tem uma MMU baseada em páginas e que usa paginação a pedido. O endereço virtual gerado tem os bits mais significativos a página virtual PV e nos bits menos significativos o deslocamento D. Cada entrada da tabela de páginas tem um bit V que vale 1 se a página PV está carregada na RAM e 0 se a página não está carregada em RAM.

Quando um processo gera um endereço virtual em que na entrada correspondente a PV o bit V está a 0, é gerada uma interrupção por falta de página e a MMU regista o valor PV. Supondo que a página PV pertence ao espaço de endereçamento do processo, descreva as acções do sistema operativo após a ocorrência da interrupção.

- *Verifica se o endereço emitido é válido; se tal não acontecer é gerada uma excepção*
- *Obtém uma página física livre F*
- *Se não há páginas físicas livres invoca um algoritmo de substituição de páginas*
- *Lança o carregamento da página virtual PV do processo corrente na página física F*
- *Passa o processo corrente para BLOCKED*
- *Quando acaba a transferência, actualiza a entrada PV da tabela de páginas (bit de validade e página física)*
- *Passa o processo que incorreu na falta de página para o estado READY. Quando o processo voltar a ser escolhido para RUNNING a instrução que provocou a falta de página é reiniciada.*

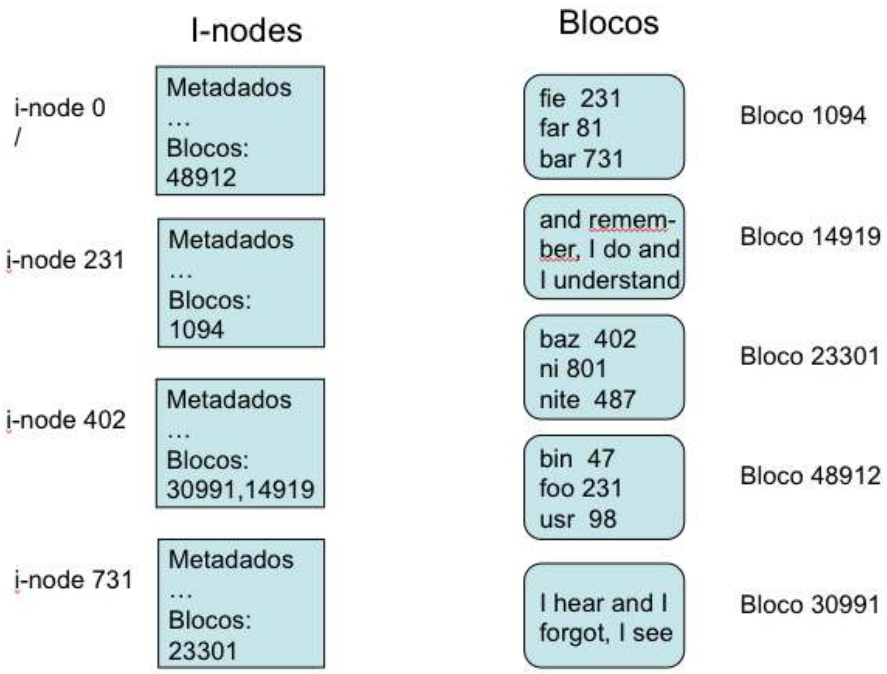
Questão 4 (2.5 valores) Considere o código C abaixo que faz parte de um simulador de memória virtual semelhante ao que foi desenvolvido nas aulas práticas. O sistema simulado tem MAX_FRAMES páginas físicas e o estado de cada página física está representado da seguinte forma:

```
#define MAX_FRAMES 128
struct frame_entry{
  unsigned int referenced;   //  1 se a página foi referenciada desde que o campo
                             // foi colocado a 0; 0 se tal não aconteceu
  unsigned int virtualPage; // número da página virtual que está carregada nesta página física
}
struct frame_entry frame_table[MAX_FRAMES];
int currentPos = 0; //variável global cujo valor sobrevive entre
                    // invocações de findVictim2ndChance()
```

A função seguinte – que está incompleta - simula o algoritmo que é chamado quando não há páginas físicas livres e é preciso escolher uma página física para carregar uma página virtual. O algoritmo usado para escolher a página física que vai acolher a página é o "second chance algorithm" ou "clock". Complete o código apresentado.

```
int findVictim2ndChance(){
    victim = -1;
    while( victim < 0){
        if( frame_table[ currentPos].referenced == 0)
            victim = frame_table[ currentPos ].virtualPage;
                                // ou victim = currentPos;
        currentPos ++;
    }
    return victim;
}
```

Questão 5 (2.0 valores) A figura seguinte representa uma parte de um sistema de ficheiros UNIX.



No kernel do sistema existe a indicação de que a directoria raiz corresponde ao i-node 0; a tabela de i-nodes já foi lida para RAM quando o disco foi montado. Diga que blocos de disco são lidos e por que ordem, quando um processo faz a chamada ao sistema `f = open("/foo/bar/baz", O_RDONLY)`

- *leitura do bloco **48912** que contém a directoria raiz*
- *é encontrada a entrada **foo** que remete para o inode 231; este diz que a directoria foo está no bloco **1094** que é lido*
- *é encontrada a entrada **bar** que remete para o inode 731; este assinala que a directoria bar está no bloco **23301**; esse bloco é lido e permite saber que o ficheiro **baz** está descrito no inode 402*

Questão 6 (2.5 valores) Para um sistema de ficheiros UNIX, indique, justificando, a sequência de leituras e escritas feitas no disco quando se executam os seguintes comandos do *shell*

a) Se muda o nome ao ficheiro /xxx. `mv /xxx /yyy`

- *inode da directoria / é consultado e lê-se o conteúdo dessa directoria*
- *a entrada **xxx** é procurada e muda-se para **yyy***
- *o inode é actualizado (datas de acesso) → opcional*

b) Se apaga o ficheiro `/zzz`. `rm /zzz`

Neste ultimo caso, supõe-se que o *link count* do *inode* de /zzz está a 1.

- *inode da directoria / é consultado e lê-se o conteúdo dessa directoria*
- *a entrada **zzz** é procurada e obtém-se o inode I correspondente*
- *o Inode I é consultado; como o "link count" é 1 pode-se eliminar i-node I*
- *Os blocos referenciados no inode I são declarados como livres (actualiza bit map no disco)*
- *A entrada I da tabela de I-nodes é declarada como livre (actualiza bit map no disco)*
- *A entrada **zzz** na directoria é eliminada (actualiza conteúdo da directoria no disco)*

Questão 7 (1.5 valores) Suponha uma variante do sistema de ficheiros UNIX em que os blocos têm 6Kbytes e os endereços de blocos têm 6 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 de um ficheiro neste sistema.

Os blocos podem ser visto como um vector address com 14 posições. Um bloco com 6K contém 1024 (1K) endereços. O nº máximo de blocos é a soma de três componentes

- 12 blocos apontados directamente por address[0] a address[11]

- um bloco com 1024 endereços apontado por address[12]

- um bloco com 1024 endereços apontado por address[13] em que cada bloco tem 1024 endereços

*Tamanho máximo do ficheiro = 6 KB * nº máximo de blocos = 6KB* (12 + 1024 + 1024 * 1024) aprox 6 Gigabytes*

Questão 8 (1.5 valores) Explique a diferença entre um *hard link* e um *link* simbólico. Indique o conteúdo dos respectivos i-nodes (se existirem).

Um hard link é uma entrada numa directoria que aponta para um inode específico que já existe quando o hard link é criado . Quando é criado um hard link não é criado nenhum inode, é somado um ao link count do inode referenciado.

Um link simbólico é também uma entrada numa directoria. Essa entrada tem um inode associad; o conteúdo desse inode é o nome do ficheiro para o qual se pretende criar a ligação

Questão 9 (1.5) Suponha um disco com as seguintes características:

- seek time médio: 10 ms
- tempo de uma rotação: 4 ms
- 500 sectores por pista; o controlador do disco tem memória suficiente para ler uma pista inteira
- Um sector demora 10 micro-segundos a ser lido

Apresente, justificando, uma estimativa para o tempo que demora a leitura de 100 blocos nas seguintes duas situações:

a) os 100 blocos estão espalhados aleatoriamente pelo disco

b) os 100 blocos estão todos contíguos na mesma pista.

*a) 100 * (10ms + 4 ms / 2 + 0.01 ms) = 1201 ms – como cada seek é feito para uma pista aleatória o seek time médio é uma boa aproximação; o tempo médio de espera pelo sector é metade do tempo de uma rotação*

*b) 10 ms + 4 ms / 2 + 100 * 0.01 ms = 13 ms – há apenas um seek, uma espera pelo sector pretendido (1/2 de uma revolução) e transferência de 100 blocos contíguos*

Questão 10 (2.0 valores) Considere o seguinte programa que usa semáforos da interface *POSIX threads*.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

int x = 0;
sem_t s1, s2;

void * f1(void *arg) {
   sem_wait(&s2); sem_wait(&s1); x = x*2; sem_post(&s1);
}
void * f2(void *arg) {
   sem_wait(&s1); x = x*x; sem_post(&s1);
}
void * f3(void *arg) {
   sem_wait(&s1); x = x+3; sem_post(&s2); sem_post(&s1);
}

int main(int argc, char *argv[]) {
  pthread_t p1, p2, p3;
  sem_init( &s1, 0, 1); // semaphore s1 initial value is 1
  sem_init( &s2, 0, 0); // semaphore s2 initial value is 0
  pthread_create(&p1, NULL, f1, NULL);pthread_create(&p2, NULL, f2, NULL);
  pthread_create(&p3, NULL, f3, NULL);
  pthread_join(p1, NULL); pthread_join(p2, NULL); pthread_join(p3, NULL);

  printf("%d\n", x);
  return 0;
}
```

Suponha que se executa o programa várias vezes. Diga, justificando, qual (ou quais) o(s) valor(es) finais de x.

s1 é usado para conseguir exclusão mútua no acesso à variável x

*s2 é usado para garantir que a acção de f1 x = x * 2 só é efectuada depois de f3 fazer x = x+ 3*

São possíveis as seguintes sequências de alterações a x

*(f2) x = x * x (f3) x = x+ 3 (f1) x = x * 2 Resultado x ← 6*

*(f3) x = x + 3 (f2) x = x * x (f1) x = x * 2 Resultado x ← 18*

*(f3) x = x + 3 (f1) x = x * 2 (f2) x = x * x Resultado x ← 36*