Fundamentos de Sistemas de Operação MIEI 2016/2017

2º Teste, 9 Dezembro 2016, 2 horas

Número_____ Nome _____

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

Questão 1 (2.0 valores) Um dos problemas que pode ocorrer na gestão da RAM é a fragmentação externa.

a) Explique porque é que, ao gerir a memória física com uma MMU constituído por um par registo base / registo limite, este problema pode ocorrer.

O uso de um par registo base/registo limite obriga à atribuição contígua de memória aos processos. Assim, a criação e destruição de processos vai fazer com que a memória física livre fique fragmentada por fragmentos não contíguos na memória. Como a maior imagem de processo que se pode carregar em memória é limitada pelo tamanho do maior fragmento de memória física livre, pode não ser possível carregar um processo, mesmo existindo memória livre suficiente para tal.

b) Se a MMU suportar páginas, este problema ocorre? Justifique.

O problema ocorre mas não é relevante, porque as imagens dos processos são carregadas em RAM página a página, não sendo necessário que as páginas físicas fiquem contíguas

Questão 2 (2.5 valores) Considere que o ficheiro *xxx.db* contém uma série de registos correspondentes à seguinte declaração

```
#typedef  struct {   float latitude; float longitude; }  ponto;
```

Escreva o código de uma função que recebe como parâmetro de entrada o nome de um ficheiro com a organização descrita e que, utilizando a chamada ao sistema *mmap* determina a maior latitude presente no conjunto de pontos guardado no ficheiro. Considere os seguintes protótipos de chamadas ao sistema relevantes.

*void * mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset);*
*int fstat(int fildes, struct stat *buf);*

Pode deixar em branco os campos *addr*, *prot*, *flags* e *offset* da chamada *mmap*. O tamanho do ficheiro está no campo *st_size* da estrutura *stat*

```
#include <sys/stat.h>
#include <sys/mman.h>
float procuraLatitude( char *fich ){ // sem teste de erros
    struct stat sf;
    ponto *p;
    int f, i;
    f = open( fich, O_RDONLY);
    fstat( f, &sf);
    p=(ponto *) mmap ( , sf.st_size , , , f, );
    float ml = -90.0 // menor latitude possível
    for(i=0; i<(sf.st_size / sizeof(ponto)); i++)
        if( ponto[i].latitude > ml) ml = ponto[i].latitude;
    return ml;
}
```

Questão 3 (2.5 valores) Considere a chamada ao sistema *fork( )* em que o SO executa numa configuração hardware em que a gestão da memória é feita com uma MMU que suporta páginas.

a) Diga como é preenchida a tabela de páginas do processo filho.

A tabela de páginas do filho é preenchida da seguinte forma
- zona de código e dados inicializados: cópia da tabela de páginas do pai
- zona de dados não inicializados, pilha e heap: páginas novas pedidas ao SO e que são preenchidas com conteúdo igual às páginas

b) Se o processo filho executa a chamada *execvp ( )* como é que e preenchida a tabela de páginas?

A tabela de páginas é deitada fora, sendo criada uma nova tabela. Essa tabela é preenchida a partir da informação existente no cabeçalho do fich. executável cujo nome é o 1º argumento da chamada ao sistema execvp

c) Explique como é que a técnica *copy on write* permite optimizar a ocupação de memória.

Na técnica "copy on write" toda a tabela de páginas do filho é uma cópia da tabela do pai; as páginas alteráveis são marcadas em ambas as tabelas como "read-only". Quando um dos processos tenta escrever na página, a MMU deteta um acesso inválido e o SO intervém; a intervenção do S.O. consiste em copiar a página para uma página nova e marcar ambas as páginas (pai e filho) como R/W. Desta forma, quando há um exec a seguir ao fork, poupa-se a tabela descrita em a)

Questão 4 (2.5 valores) Explique como é que usando uma MMU baseada em páginas e com um bit de validade V é possível executar programas em que os dados são maiores do que a RAM disponível para os utilizadores.

Nas páginas virtuais que pertencem à imagem do processo o bit V[P] indica o seguinte para a página virtual P

V[P] == 1 a página é válida e está carregada em RAM

V[P] == 0 a página é inválida, ou seja válida está apenas no disco.

Como os processos exibem características de localidade nas referências que fazem à memória, pode ter em memória apenas parte das suas páginas e o SO fazer paginação a pedido à medida que as páginas são referenciadas.

Quando o processo referencia uma página P válida mas não carregada em memória, a MMU deteta que V[P] é 0 e lança uma interrupção por falta de página; no atendimento desta interrupção o SO promove o carregamento da página P para RAM. Para conseguir carregar a página, o SO usa uma página física livre ou uma página retirada ao processo (ou a outro) usando uma estratégia de substituição de páginas, que vai escolher uma página não usada há algum tempo.

Questão 5 (3.0 valores) Considere que o periférico */dev/my_device* foi criado com os seguintes comandos executados pelo administrador de sistema.

```
mknod /dev/my_device c 298 0
chmod ugo+rw /dev/my_device
```

Supondo que existe um device driver no sistema que trata este dispositivo, descreva o que se passa quando um programa executa cada uma das chamadas aos sistema abaixo indicadas. Inclua na sua resposta o papel da estrutura *file_operations*.

```
int f = open( "/dev/my_device", O_WRONLY);
write( f, '123456', 6);
```

Quando é feito o open, obtém-se o inode de "/dev/my_device" onde está o Major Device Number (MDN) e o Minor Device Number (mDN) do periférico; uma referência para esse i-node é colocada na entrada f da tabela de canais abertos do processo; também é chamada a função xxx_open que está na linha MDN de matriz que contém todas as estruturas file_operations relativas aos periféricos em uso.

No caso de write, é chamada a função xxx_write da linha de matriz cujo índice é o MDN.

Questão 6 (1.5 valores) Considere um sistema operativo que usa uma cache de blocos para acesso aos discos que contêm ficheiros; suponha que cada bloco do disco tem 2048 bytes. Explique as vantagens que tem a utilização da cache na seguinte sequência de código

```
(1) int f = open( "./xxx", O_WRONLY | O_CREAT);
(2) for( int i = 0; i < 256; i++)  write( f, '12345678', 128);
```

Na linha (2) as 256 chamadas ao sistema write() vão escrever sempre no mesmo bloco em RAM. O ficheiro "xxx" só é actualizado no disco, quando for fechado (ou ocorrer um flush)

Questão 7 (3 valores) Considere a seguinte sequência de chamadas ao sistema

```
(1) int f = open( "/x.log", O_WRONLY | O_CREAT);
(2) write( f, '123456', 6);
(3) close(f);
```

Para cada uma das chamadas ao sistema explique que consultas e alterações são feitas na área de meta-dados do disco que suporta o sistema de ficheiros em que está o ficheiro *x.log*. Suponha que o ficheiro não existia antes da chamada ao sistema (1) e que o processo que faz as chamadas tem permissões para escrever na diretoria raiz do sistema.

(1) - é lido para RAM o i-node R de directoria raiz
- a partir do i-node R é obtido o conteúdo do dir. raiz
- a partir do dir. raiz, procura-se "x.log", que não é encontrado
- obtém-se uma entrada I livre na tabela de i-nodes, e inicializa-se essa entrada, que é copiada para RAM
- actualiza-se a directoria raiz introduzindo-se uma entrada "x.log" / I
- preenche-se a entrada f da tabela de canais abertos, apontando para a cópia de I em RAM

(2) - é obtido o bloco livre B no bit map de blocos; actualiza-se o bit map de blocos ocupados
- é escrito no bloco B "123456"
- actualiza-se o i-node I em RAM - data de acesso, comprimento, blocos

(3) - actualiza o i-node I no disco
- flush dos buffers
- liberta a entrada f na tabela de canais abertos
- o i-node I é marcado como tendo menos uma referência em RAM

Questão 8 (1.5 valores) Considere um disco que depois de formatado tem 2 24 blocos e em que cada bloco tem 1024 (2^{10}) bytes; os endereços dos blocos têm 32 bits (4 bytes).

Suponha que neste disco foi colocado um sistema de ficheiros que usa uma estratégia de atribuição de espaço em disco indexada, em que os blocos atribuídos a um ficheiro são definidos por 10 endereços de blocos. Os endereços 0 a 7 são endereços diretos e os endereço 8 e 9 são os endereços de dois blocos que contêm endereços de blocos. Diga, justificando, qual é o tamanho máximo, usando esta estratégia de atribuição, que um ficheiro pode ter?

tamanho máximo = nº máximo de blocos * tamanho de 1 bloco

$$= 1K * \left(8 + 2 * \frac{1024}{4}\right) = 1K * 520$$

($\frac{1024}{4}$ = nº de endereços num bloco)

Questão 9 (1.5 valores) Diga porque é que a utilização da cache de blocos pode contribuir para a criação de inconsistências no sistema de ficheiros.

O uso de cache de blocos através de "delayed writing" faz com que, durante algum tempo, as alterações feitas a blocos que estão em RAM ainda não estão no disco; por outro lado, a ordem das escritas em RAM e no disco podem ser diferentes. Uma alteração no sistem