Nome:		Número:	

LEIC/LERC– 2008/09

Primeiro Exame de Sistemas Operativos

20 de Janeiro de 2009

Duração: 2h30m

Identifique o cabeçalho de todas as folhas da prova.

O exame é resolvido no espaço dedicado ao efeito após cada pergunta.

O número de linhas reservado para o efeito não pode ser excedido, havendo lugar a uma penalização para quem responder num número de linhas superior.

Em caso de engano, poderá usar em alternativa o espaço da última página para responder à questão, devendo indicá-lo claramente, e respeitar o limite de linhas da questão.

Nas perguntas de escolha múltipla, cada resposta errada desconta ¼ da cotação.

Grupo I [3 valores]

```
int contador =  1;

fx () {
      monitor_enter();
      contador = contador --;
      while (contador < 0) monitor_wait();
      monitor_exit ();
      }
fy ()) {
      monitor_enter();
      contador = contador ++;
      if (contador <= 0) monitor_signal();
      monitor_exit ();
      }
`
```

1. Considere que as funções `monitor_enter`, `monitor_exit`, `monitor_signal` e `monitor_wait` têm o mesmo significado que as funções definidas no enunciado do trabalho e são semanticamente semelhantes às existentes no CLI da Microsoft.

   1.1. [0,7 val.] Explique o funcionamento da função `fx` tendo particular atenção à sincronização.

   1.2. [0,7 val.] Explique o funcionamento da função `fy` tendo particular atenção à sincronização.

2. [0,8 val.] A variável `contador` é partilhada e como tal tem de ser modificada dentro de uma secção crítica. Explique porquê. Dê um exemplo com base neste programa onde a não existência de uma secção crítica poderia originar um erro.

3. [0,8 val.] Quando a tarefa é desbloqueada, está outra tarefa a executar o monitor. Como é resolvida a exclusão mútua nessa situação?

Grupo II [4 valores]

1. [0,7 val.] A inibição das interrupções é uma forma de implementar uma secção crítica. Esta solução funciona num multi-processador? Justifique a resposta.

2. [0,6 val.] Qual das seguintes afirmações **é verdadeira** em relação à utilização de interrupções de software para implementar as chamadas sistema.

   ☐ As interrupções de *software* tem um vector que permite agulhar para funções do núcleo mas obrigam a outra instrução para mudar o modo de protecção do processador.

   ☐ As interrupções de *software* mudam automaticamente o modo de protecção mas não conseguem indicar qual a função sistema a executar o que obriga a utilizar outras instruções.

   ☐ Faz com que o código da chamada ao sistema operativo corra atomicamente, sem poder ser interrompido, por exemplo, por uma interrupção de *hardware*.

   ☐ Permite que o sistema operativo possa ser modificado (desde que não se altere a interface) sem necessidade de recompilar as aplicações.

Número:

3. Suponha que num sistema operativo estão a correr 3 processos. O **processo P1** é CPU-intensivo (não executa operações de E/S).

O **processo P2** executa o seguinte código:

```
main( ) {
 while (!done) {
  doCPUwork(); // Requer uma unidade de tempo de CPU
  doCPUwork(); // Requer uma unidade de tempo de CPU
  doIO();     // Efectua uma operacao de E/S que bloqueia o
              // processo durante duas unidades de tempo
  doCPUwork(); // Requer uma unidade de tempo de CPU
 }
}
```

E o **processo P3** o seguinte código:

```
main( ) {
 while (!done) {
  doCPUwork(); // Requer uma unidade de tempo de CPU
  doIO();     // Efectua uma operacao de E/S que bloqueia o
              // processo durante duas unidades de tempo
  }
}
```

Os processos têm as seguintes características:

Processo	Prioridade Base	Instante do início da execução	Tempo total de CPU que o processo irá consumir na sua execução
P1	**10**	**0**	**6**
P2	**11**	**3**	**4**
P3	**9**	**4**	**4**

3.1. [2 val.] Suponha que o algoritmo de escalonamento utilizado é **preemptivo**, **com prioridades dinâmicas** (em que um valor numérico de prioridade mais elevado corresponde a um processo mais prioritário). A prioridade de um processo começa por ser a sua prioridade base, e periodicamente ao fim de 4 *quantums* (nos *quantum* 4, 8, 12, etc.), o escalonador calcula a prioridade dos processos através da seguinte fórmula:
prioridade = prioridade – (int) (Número de time slices usados no intervalo/2 + 0,5)

Se vários processos tiverem igual prioridade, o escalonador escolhe o que não é executado há mais tempo ou o que ainda não se executou.
Preencha a seguinte tabela, indicando, para cada instante de tempo, qual o estado de cada processo do sistema, e a respectiva prioridade no início do quantum. (considere que o escalonador se executa no final do quantum 3 e antes do quantum 4)
Use a seguinte notação: Letra **E** – Em execução. Letra **B** – Bloqueado. Letra **V** – Executável. Entrada em branco – processo não está activo no sistema (não iniciou ou já terminou).
Note que, na solução correcta, não é necessário utilizar mais espaço do que o fornecido.

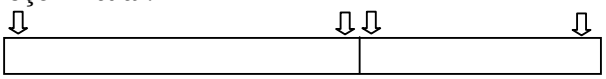
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
P1	E / 10														
P2															
P3															

3.2. [0,7 val] O algoritmo de cálculo das prioridades descrito na pergunta 3.1 assemelha-se ao utilizado por vários sistemas operativos, em particular o Unix. Contudo no Unix a formula não é exactamente esta. Indique quais são as diferenças e que vantagens vê do ponto de vista do escalonamento nessa alteração, ilustrando com o exemplo da pergunta anterior.

Grupo III [4 valores]

1. Considere um sistema com um endereçamento virtual de 31 bits e páginas de 4k bytes.

1.1. [0,5 val.] Indique na figura a ordem do bit correspondente a cada seta na estrutura do endereço virtual.



1.2. [0,5 val.] Admita que cada entrada na tabela de páginas ocupa 64 bits. Qual a dimensão máxima da tabela de páginas de um processo?

☐ 512 Kbytes.

☐ 4 Mbytes.

☐ 32 Mbytes.

☐ 128Mbytes.

1.3. [0,5 val.] Indique o cálculo efectuado.

1.4. [0,5 val.] Que problemas levanta esta dimensão da tabela e que proposta conhece para as resolver?

Número:

2. Considere o mesmo sistema com um endereçamento virtual de 31 bits e páginas de 4k bytes e a seguinte tabela de páginas em que `Idade` tem o significado habitual de tempo desde o último acesso e `Carregada` indica o tempo de permanência em memória.

Página	Presente	Protecção	Idade	Carregada	Base
0	0	RW	0		0x00001
1	1	R	2	100	0x00001
2	0	R	0		0x00001
3	1	RW	1	200	0x00002
4	1	RW	0		0x00000
5	0	E	0	50	0x00000

Considere os seguintes acessos à memória tendo como base esta tabela de páginas. Indique qual o resultado do endereçamento, especificando:

- o endereço real, em caso de sucesso;
- se ocorreu violação da protecção;
- se ocorreu falta de página.

2.1. [0,2 val.] Qual é o resultado do endereçamento?
Acesso em leitura a `0x00001124`

2.2. [0,2 val.] Qual é o resultado do endereçamento?
Acesso em escrita a `0x00001124`

2.3. [0,2 val.] Qual é o resultado do endereçamento?
Acesso para leitura da instrução a `0x00005421`

2.4. [0,2 val.] Qual é o resultado do endereçamento?
Acesso em escrita a `0x00005124`

2.5. [0,2 val.] Qual é o resultado do endereçamento?
Acesso em escrita a `0x00004421`

3. Considere o algoritmo de substituição de páginas denominado NRU (Não Usada Recentemente) no qual as páginas são organizadas em 4 grupos, consoante os bits R (página referenciada) e M (página modificada).

   3.1. [0,5 val.] Na escolha das páginas a seleccionar, a algoritmo NRU dá prioridade a páginas com o bit M=0. Explique porquê essa é uma boa opção.

   3.2. [0,5 val.] Explique em que circunstâncias é que uma página pode estar no estado R=0, M=1 (Não referenciada, modificada).

Grupo IV [4 valores]

1. Considere uma partição organizada da seguinte forma, montada na raiz ("/") do sistema de ficheiros de uma máquina.

Super Bloco	Mapa de Blocos Livres	Mapa de Inodes Livres	Tabela de Inodes	Blocos de Dados

Assuma que **não há qualquer mecanismo de caching activo** e que o **conteúdo do directório "/" ocupa menos que a dimensão de um bloco**.

Um processo chamou a função sistema `abre("/so.txt", READ_ONLY)`, que retornou com sucesso. Desde o início da execução da chamada a `abre` até ser finalmente obtido o inode do ficheiro "/so.txt", ocorrem uma ou mais leituras às componentes "Tabela de Inodes" e "Blocos de Dados" da partição.

   1.1. [1,5 val.] Enumere a história dessas leituras (no máximo de 5), indicando para cada uma: (1) a componente acedida (tab. de inodes ou blocos de dados) e (2) como se determinou a posição (número de inode ou número de bloco) a ler dessa componente.

Número:

A primeira leitura é dada como exemplo.

	Componente acedida (Tab. Inodes ou Blocos de Dados)	Inode/Bloco lido	Como se determinou número de inode/número de bloco a ler
1.	Tabela de inodes	Inode do directório "/"	Número de inode de "/" é uma constante pré-conhecida
2.			
3.			
4.			
5.			

1.2. [1 val.] Cada leitura do exercício anterior é feita directamente sobre a partição em disco. Se se usassem *caches*, algumas das leituras descritas anteriormente poderiam ser imediatamente servidas a partir da memória primária (em caso de *hit*).
Para cada *cache* apresentada a seguir, indique quais as leituras a partir da partição que indicou na alínea anterior (escolhidas de entre 1 e 5) que essa *cache* poderia evitar (em caso de *hit*).
Considere cada *cache* separadamente. Isto é, se duas *caches* puderem evitar a mesma leitura, indique essa leitura na resposta de ambas.

***Cache* de blocos**

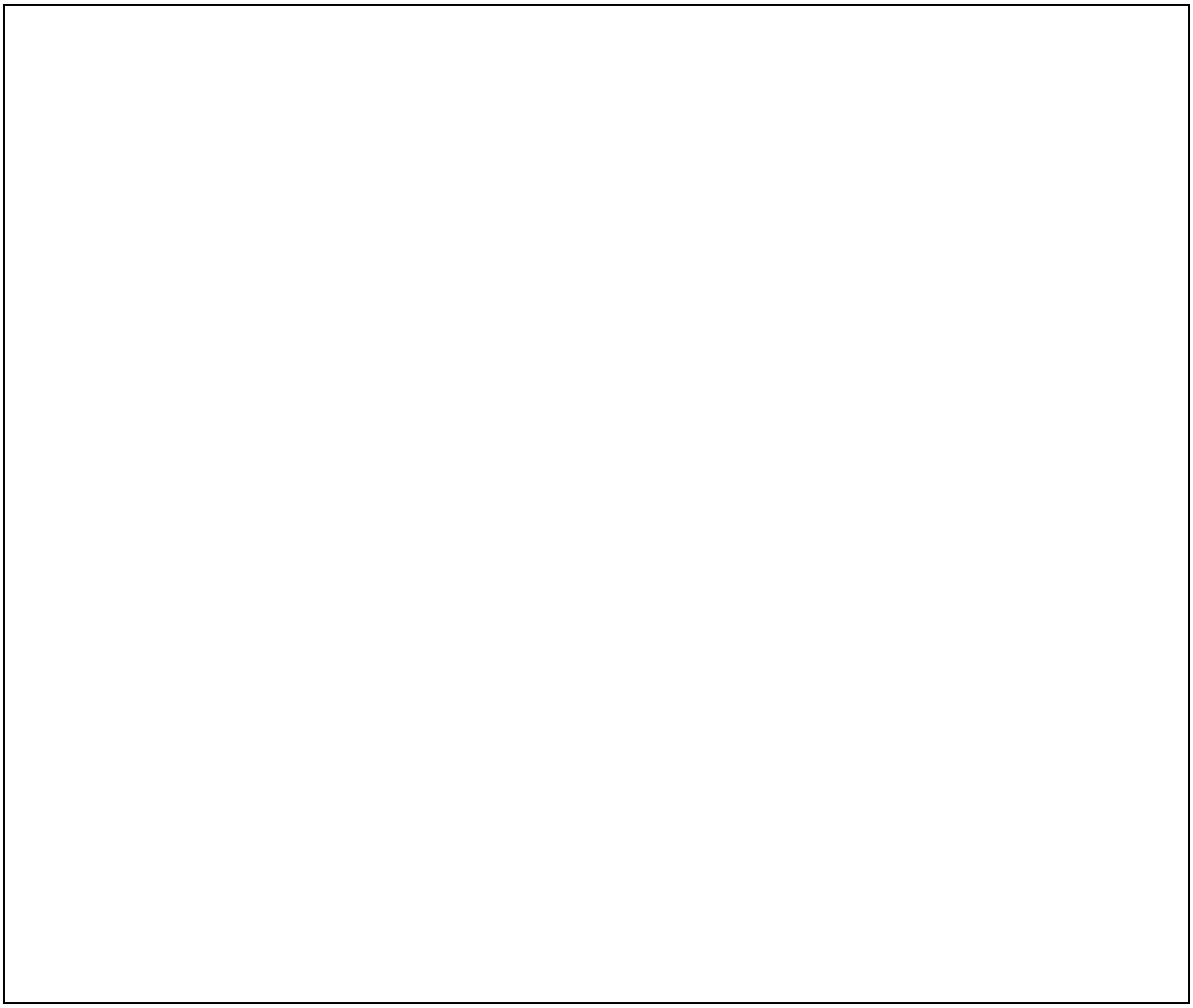
***Cache* de i-nodes**

2. [1,5 val.] Considere a seguinte sequência de acontecimentos, em que 2 processos P1 e P2 acedem aos ficheiros indicados na ordem indicada:

```
P1: f = open ("/users/cnr/dir/f1"); read (f, buf, 50);
P2: g = open ("/users/cnr/dir/f1"); read (g, buf, 100);
P2: h = open ("/tmp/tempfile");
P1: p = fork(); /* cria um processo filho P3 */
P1: read (f, buf, 150);
P2: read (f, buf, 200);
```

Represente graficamente as tabelas de *file descriptors* por processo, a tabela de ficheiros abertos global do sistema e a tabela de descritores de ficheiros (*inodes*) após a sequência de acontecimentos anterior.

Inclua o valor dos índices que indicam a posição (i.e. os cursores).



Número:

Grupo V [4 valores]

1. Escreva em pseudo-código as partes que faltam de um programa servidor que receba pedidos por *sockets stream* e responda a esses pedidos. Cada pedido consiste numa *string* (e.g. "terceira", max 256 caracteres) e a resposta é a concatenação da *string* enviada no pedido com todas as *strings* enviadas anteriormente (e.g. "primeirasegundaterceira") até um máximo de 10000 caracteres. Seja o mais preciso possível nas chamadas sistema.

Início do programa

```
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/un.h>

#define SERV_ADDR "/tmp/server.socket"
#define MAX_TEXT 10000
#define MAX_MSG 256

main() {
  struct sockaddr_un addr;
  int sock, clisock, addr_size, readbytes, cur = 0;
  char text[MAX_TEXT];
  char buf[MAX_MSG+1];
  text[0] = '\0';
  memset(text, 0, MAX_TEXT);
  sock = socket(AF_UNIX, SOCK_STREAM, 0);
```

1.1. [0,5 val.] O *socket* anteriormente criado não tem nome global, escreva em pseudo-código a atribuição de um nome ao *socket*.

1.2. [0,5 val.] Para que o *socket* aceite ligações, o que é necessário? Escreva em pseudo-código.

1.3. [2 val.] Programe o ciclo principal do servidor.

2. Considere dois processos numa máquina Linux que cooperam modificando concorrentemente uma matriz de 1000x1000 inteiros. Cada actualização dos valores da matriz é calculada com base em valores pré-existentes na matriz.
   2.1. [0,5 val.] Qual o mecanismo de comunicação entre processos que deverá ser utilizado para que ambos os processos tenham acesso à versão actualizada da matriz? Justifique a sua escolha.

Número:

2.2. [0,5 val.] Descreva sucintamente como se processa a comunicação entre os dois processos. Caso tenha recorrido a outros mecanismos do sistema operativo, refira-os.

Grupo VI [1,5 valores]

1. [0,5 val.] Quais são as vantagens da utilização de IORB no modelo de Entradas/Saídas?

2. [0,5 val.] Descreva sucintamente os passos que um utilizador deve efectuar para inserir dinamicamente um periférico do tipo carácter num sistema Linux.