Número: Nome:

# LEIC/LERC – 2009/10
## 2º Exame de Sistemas Operativos

2 de Fevereiro de 2010
**Responda no enunciado, apenas no espaço fornecido. Identifique todas as folhas.**
Duração: 2h30m

### Grupo I [3 v.]

Considere um Sistema Operativo com um escalonamento com as seguintes características:

- Máquina mono-processador
- Multilista com 4 níveis de prioridade: 1 (menos prioritário) a 4 (mais prioritário)
- Preemptivo
- Cada nível é gerido em FIFO
- Sempre que um processo termina o seu time-slice a sua prioridade é decrementada de uma unidade.
- Sempre que um processo se bloqueia num semáforo a sua prioridade é incrementada de 1 unidade (limitada a 4).
- Com semáforos com fila de espera gerida em FIFO.
- Sincronização directa com as funções Supender e Acordar.

| Lista escalonador | | TS1 | TS2 | TS3 | TS4 | TS5 | TS6 |
|---|---|---|---|---|---|---|---|
| | Prio 4 | P1 | | | | | |
| | Prio 3 | P2, | P2, | | | | |
| | Prio 2 | | | | | | |
| | Prio 1 | P3 | P3 | | | | |
| | | | | | | | |
| Lista suspensos | | nil | nil | | | | |
| Semáforo S1 | | | | | | | |
| | Contador | 0 | 0 | | | | |
| | Lista | nil | P1 | | | | |
| Semáforo S2 | | | | | | | |
| | Contador | 1 | 1 | | | | |
| | Lista | nil | nil | | | | |

Considere que no sistema existem três processos e dois semáforos

- P1 com prioridade inicial de 4
- P2 prioridade inicial 3
- P3 prioridade inicial 1
- S1 com o valor 0 na sua variável de controlo
- S2 com o valor 1 na sua variável de controlo

**Legenda**
Na tabela a ordem dos processos representa a sua posição na estrutura de lista FIFO.
PE é a abreviatura para Processo em Execução.
TSx – indica um timeslice sendo a representação na tabela a situação no início do timeslice. Num timeslice os processos podem executar o seu algoritmo mas apenas considerámos as operações de sincronização relevantes. O processo que fica em execução num dado timeslice executa-o até ao fim

(caso não se bloqueie entretanto); caso se bloqueie, o timeslice é considerado concluído nesse momento e começa o próximo timeslice (com novo processo em execução).
Se um processo estiver em mais que uma lista represente-o em ambas.

Considere a evolução SEQUENCIAL do sistema descrita nas alíneas seguintes.

1. [0,75 v.]O preenchimento inicial da tabela inicial corresponde à seguinte sequência:

   - TS1 – PE executa Esperar (S1)
   - TS2 – PE executa Suspender P1 e Esperar (S2)

Preencha o estado no início de TS3.

2. [0,75 v.] Continue a evolução do sistema considerando que:

   - TS3 - PE executa Esperar (S2)

3. [0,75 v.] Continue a evolução do sistema considerando que:

   - TS4 – PE executa Assinalar (S1)

4. [0,75 v.] Continue a evolução do sistema considerando que:

   - TS5 - PE executa Acordar (P1)

## Grupo II [4 v.]

Considere um sistema que gere contas bancárias e que serve clientes que efectuam pedidos de transferência de dinheiro entre contas. Tais pedidos podem ocorrer concorrentemente, sendo cada pedido servido por uma thread diferente. Assuma os seguintes requisitos fundamentais:
**Requisito A**: Transferir uma dada quantia de uma conta origem, A, para uma conta destino, B, consiste em debitar essa quantia de A, e creditar B com a mesma quantia.
**Requisito B:** Uma dada transferência entre uma conta A e uma conta B deve parecer atómica do ponto de vista de outras transferências que observem o saldo de A ou de B.
**Requisito C:** Se, concorrentemente, um cliente pedir para transferir de A para B, e outro cliente pedir para transferir de C para D (em que A, B, C e D são contas diferentes), as duas transferências devem correr completamente em paralelo (i.e. sem que nenhuma seja obrigada a esperar pela outra).
**Requisito D:** O sistema não deve chegar a situações de interblocagem (deadlock).

Considere as seguintes implementações da função transfere (que cada thread executa para servir cada pedido de transferência):

```
I1:
transfere(int A, int B, int quantia) {
   saldo[A] -= quantia;
   saldo[B] += quantia;
}
```

```
I2:
monitor mon;

transfere(int A, int B, int quantia) {
   mon.enter();
   saldo[A] -= quantia;
   saldo[B] += quantia;
   mon.exit();
}
```

```
I3:
monitor mon[NUM_CONTAS];

transfere(int A, int B, int quantia) {
   mon[A].enter();
   mon[B].enter();
   saldo[A] -= quantia;
   saldo[B] += quantia;
   mon[A].exit();
   mon[B].exit();
}
```

Assuma que a instrução alto-nível "`A.saldo -= quantia`" corresponde à seguinte sequência de instruções máquina:

```
mov AX, A.saldo
mov BX, quantia
sub AX, BX
mov A.saldo, AX
```

E que a instrução alto-nível "`B.saldo += quantia`" corresponde à seguinte sequência de instruções máquina:

```
mov AX, B.saldo
mov BX, quantia
add AX, BX
mov B.saldo, AX
```

1. [3 v] Para cada implementação, qual ou quais dos requisitos não são satisfeitos. Ilustre com um exemplo cada requisito que afirmar nao ser satisfeito.

2. [1 v] Estenda a **solução I3** para que, no caso da conta de origem não ter saldo suficiente, a transferência se **bloqueie** até que tal aconteça. Mais uma vez, não utilize nem semáforos nem mutexes.

## Grupo III [3,1 v]

Considere uma arquitectura paginada de 32 bits, com páginas de 4KB.
**Não existe qualquer tipo de cache, nem TLB.**
Existem **apenas 4 páginas** de memória física disponíveis para páginas de processos. A política de substituição é FIFO.
O tempo assume-se contínuo e crescente.
Num dado momento, um processo, P1, está em execução, tendo a sua tabela de páginas o seguinte estado:

| | Presente | Permissões | Base | Instante de carregamento | Modificada (dirty) |
|---|---|---|---|---|---|
| 0 | 1 | X | 0x00000 | 1002 | 0 |
| 1 | 0 | R | 0x00000 | 700 | 0 |
| 2 | 1 | RW | 0x00001 | 2030 | 1 |
| 3 | 1 | RW | 0x00002 | 2300 | 0 |
| 4 | 1 | R | 0x00003 | 4023 | 0 |
| 5 | 0 | R | 0x00001 | 650 | 0 |
| 6 | 0 | R | 0x00003 | 800 | 0 |

1. [1,5 v] Partindo do estado inicial da tabela de páginas que é apresentado acima, P1 efectuou a seguinte sequência de acessos. Preencha a tabela seguinte:

| Acesso pedido à mem. virtual | Tab. Páginas foi consultada? | Houve mudança para modo núcleo? | Houve substituição de pág.? Se sim: (i) qual a página e (ii) teve de ser escrita em disco? | End. real calculado |
|---|---|---|---|---|
| 1. Execução de 0x00000003 | | | | |
| 2. Leitura de 0x00005000 | | | | |
| 3. Execução de 0x00000004 | | | | |
| 4. Leitura de 0x00005500 | | | | |
| 5. Execução de 0x00000005 | | | | |

2. [0,6 v] Indique o estado da tabela de páginas de P1 após os acessos na alínea anterior.

| | Presente | Permissões | Base | Instante de carregamento | Modificada (dirty) |
|---|---|---|---|---|---|
| 0 | | | | | |
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |
| 6 | | | | | |

3. [1 v] Proponha um melhoramento ao sistema que permita um desempenho melhor que o observado na alínea 1. Preencha a tabela de novo, assumindo que a mesma sequência de acessos se repete no sistema melhorado.

Melhoramento:

| |
|---|
| |
| |
| |
| |

| Acesso pedido à mem. virtual | Tab. Páginas foi consultada? | Houve mudança para modo núcleo? | Houve substituição de pág.? Se sim: (i) qual a página e (ii) teve de ser escrita em disco? | End. real calculado |
|---|---|---|---|---|
| 1. Execução de 0x00000003 | | | | |
| 2. Leitura de 0x00005000 | | | | |
| 3. Execução de 0x00000004 | | | | |
| 4. Leitura de 0x00005500 | | | | |
| 5. Execução de 0x00000005 | | | | |

**Grupo IV [3,9 v]**

Analise cuidadosamente o sistema de ficheiros no qual o projecto se baseou, SNFS-server. O programa seguinte é um extrato simplificado da função fs_write do sistema de ficheiros.

```
1. int fs_write(fs_t* fs, int fileInodeNumber, unsigned offset, unsigned count,
   char* buffer)
2. {
3. fs_inode_t* inode = lêDaTabeladeInodes(fileInodeNumber);
4.
5. int numBlocosUsados = numBlocos(inode->size);
6. int numNovosBlocosNecessarios =
7.        MAX(numBlocos(offset+count),numBlocosUsados)- numBlocosUsados;
8.
9. if (numNovosBlocosNecessarios > 0) {
10.
11.         if(numNovosBlocosNecessarios > INODE_NUM_BLKS - numBlocosUsados) {
12.               dprintf("[fs_write] no free block entries in inode.\n");
13.               return -1;
14.         }
15.
16.         for (int i = numBlocosUsados;
17.                         i < numBlocosUsados + numNovosBlocosNecessarios; i++) {
18.               inode->blocks[i] = ProcuraBlocoLivreNoBlockBitMap();
19.               if (inode->blocks[i]==-1) {
20.                     dprintf("[fs_write] there are no free blocks.\n");
21.                     return -1;
22.               }
23.               ReservaBlockNoBlockBitMap(inode->blocks[i]);
24.         }
25.   }
26.
27.   char block[BLOCK_SIZE];
28.   int num = 0, pos;
29.   int iblock = offset/BLOCK_SIZE;
30.
31.   while (num < count && iblock < numBlocosUsados) {
32.         block = block_read(inode->blocks[iblock]);
33.         int start = ((num == 0)?(offset % BLOCK_SIZE):0);
34.         for (int i = start; i < BLOCK_SIZE && num < count; i++, num++) {
35.               block[i] = buffer[num];
36.         }
37.         block_write(inode->blocks[iblock], block);
38.         iblock++;
39.   }
40.
41.   while (num < count && iblock < numBlocosUsados + numNovosBlocosNecessarios) {
42.         for (int i = 0; i < BLOCK_SIZE && num < count; i++, num++)
43.               block[i] = buffer[num];
44.
45.         block_write(inode->blocks[iblock], block);
46.         iblock++;
47.   }
48.
49.   inode->size = MAX(offset + count, inode->size);
50.   // update the inode in disk
51.   escreveNaTabelaInodes(inode, fileInodeNumber);
52.   return 0;
53.   }
```

1. [1,5 v] Identifique as regiões de código onde os seguintes passos fundamentais do algoritmo de escrita são implementadas (indique números de linhas de código, ou gamas de linhas de código):
   a. Obtenção do inode do ficheiro.

   b. Reserva de blocos novos no caso da escrita ser para além da dimensão do ficheiro.

   c. Escrita de novos dados em blocos já existentes do ficheiro.

   d. Escrita de novos dados em novos blocos reservados para o ficheiro.

   e. Actualização do inode do ficheiro com o novo tamanho e blocos.

2. [0,8 v] Qual a dimensão máxima de um ficheiro neste sistema de ficheiros? Responda com base no código.

3. [0,8 v] É possível que uma escrita que obrigue a acrescentar novos blocos ao ficheiro falhe apesar do ficheiro não ter alcançado a dimensão máxima por ficheiro. O que pode levar a essa situação? Jutifique com referências a linha(s) do código.

4. [0,8 v] Suponha que dispõe de uma cache de blocos e de uma cache de inodes implementadas. Indique todas as linhas de código antes das quais deveria consultar cada uma dessas caches (e só no caso de uma falta de cache é que essas linhas seriam executadas).

Cache de blocos (linhas em que seria feita a consulta à cache):

Cache de inodes (linhas em que seria feita a consulta à cache):

## Grupo V [3,9 v]

O código abaixo permite a criação de um socket datagram para um processo servidor e é semelhante ao protótipo disponibilizado no trabalho da cadeira.

```
void srv_init_socket(struct sockaddr_un* servaddr)
{
    if ((sockfd = socket(AF_UNIX, SOCK_DGRAM, 0)) < 0){
        printf("[snfs_srv] socket error: %s.\n", strerror(errno));
        exit(-1);
    }
```

1

```
    bzero(servaddr, sizeof(*servaddr));
    servaddr->sun_family = AF_UNIX;
    strcpy(servaddr->sun_path, SERVER_SOCK);
```

2

```
    if (unlink(servaddr->sun_path) < 0 && errno != ENOENT) {
        printf("[snfs_srv] unlink error: %s.\n", strerror(errno));
        exit(-1);
    }
```

3

```
    if (bind(sockfd, (struct sockaddr *) servaddr, sizeof(*servaddr)) < 0){
        printf("[snfs_srv] unbind error: %s.\n", strerror(errno));
        exit(-1);
    }
}
```

4

1. Suponha que pretende utilizar um socket no domínio TCP/IP.
   a. [0,4 v] Que secções teria de modificar? Justifique.

| |
|---|
| |
| |
| |

   b. [0,4 v] Programe as modificações numa das secções que indicou anteriormente.

| |
|---|
| |

2. Suponha agora que pretende utilizar um socket tipo stream mas mantendo o domínio Unix
   a. [0,4 v] Que secções teria de modificar? Justifique.

| |
|---|
| |
| |
| |

   b. [0,4 v] Programe as modificações numa das secções que indicou anteriormente.

| |
|---|
| |

3. Das secções acima uma não é específica dos sockets e tem um âmbito mais abrangente.
   a. [0,3 v] Qual é?

   b. [0,4 v] Para que serve?

4. No modelo datagram o socket depois de criado poderia ser usado para receber mensagens com esta função.

```
reqsz = recvfrom(sockfd, (void*)req, sizeof(*req), 0, (struct sockaddr *)cliaddr,
clilen);
```

   a. [0,5 v] Com um socket tipo stream passa-se o mesmo? Justifique indicando quais as diferenças.

   b. [0,6 v] Num socket tipo stream pode receber mensagens utilizando a função read habitual da interface de ficheiros. Justifique a complexidade acrescida em termos de parâmetros da função recvfrom utilizada no exemplo acima.

   c. [0,5 v] Relacione a resposta anterior com o modelo de comunicação “muitos para um”.

## Grupo VI [2,1 v]

O programa seguinte envolve operações de Entrada/saída.

```
main()
{
char c ;
int count=0;

while ( ( c = getchar() ) != EOF )
   count ++ ;

printf( "%d characters\n" , count ) ;
}
```

1. [0,6 v] Para que periféricos se efectuam as E/S e quando foram efectuadas as aberturas do respectivo canal?

| |
|---|
| |
| |
| |

2. [0,9 v] Ao chamar a função getchar, o processo chamador evolui para um/vários estado(s) diferente(s). Indique cada estado por onde o processo passa e indique sucintamente o que sucedeu para o processo passar a esse estado. Considere a situação em que nenhum input foi dado pelo utilizador.

   Na coluna "estado", assuma os seguintes estados possíveis: em execução em modo utilizador, em execução em modo núcleo, executável em modo utilizador, executável em modo núcleo, bloqueado e suspenso. (Nota: não tem de preencher todas as linhas disponíveis para resposta.)

Estado inicial:

| |
|---|
| |

| Estado para onde transitou: | O que sucedeu para transitar para o estado: |
|---|---|
| | |
| | |
| | |
| | |

3. [0,6 v] Suponha agora que depois de iniciar a sua execução o utilizador carrega nas seguintes teclas

   a "return"

O acto de carregar nas teclas que componente do sistema de Entradas/Saídas vai activar? Justifique.

| |
|---|
| |
| |
| |
| |