Número: Nome:

LEIC/LERC – 2009/10

Repescagem do 2º Teste de Sistemas Operativos

2 de Fevereiro de 2010

Responda no enunciado, apenas no espaço fornecido. Identifique todas as folhas.

Duração: 1h30m

Grupo I [8,4 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.   }
```

(continua na página seguinte...)

```
48.
49.    inode->size = MAX(offset + count, inode->size);
50.    // update the inode in disk
51.    escreveNaTabelaInodes(inode, fileInodeNumber);
52.    return 0;
53.    }
```

1. [3v] 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. [1,2v] Por que razão a escrita de novos dados é feita de forma diferente consoante seja num bloco já existente ou num dos novos blocos reservados? Suporte a sua justificação com referências a linhas do programa.

3. [0,9v] Qual a dimensão máxima de um ficheiro neste sistema de ficheiros? Responda com base no código.

4. [0,9v] 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.

5. [1,1v] Esta implementação da função fs_write suporta correctamente múltiplas chamadas em paralelo? Se sim, justifique. Se não, indique como poderia ser modificada para suportar chamadas concorrentes.

6. [1,3v] 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 II [7,2v]

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. [0,7v] Que secções teria de modificar? Justifique.

b. [0,9v] 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. [0,7v] Que secções teria de modificar? Justifique.

   b. [0,9v] 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. [0,4v] Qual é?

   b. [0,7v] 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. [0,9v] Com um socket tipo stream passa-se o mesmo? Justifique indicando quais as diferenças.

b. [1,1v] 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. [0,9v] Relacione a resposta anterior com o modelo de comunicação “muitos para um”.

Grupo III [4,4v]

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

2. [1,5v] 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. [0,9v] Explique como é que o núcleo Unix sabe fazer corresponder esta chamada a um periférico físico cujo controlador está mapeado num dado endereço de memória.

4. [0,9v] 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.
