Fundamentos de Sistemas de Operação
2º Teste, 19 de Dezembro de 2019

QUESTÕES DE DESENVOLVIMENTO

Para todas as questões relacionadas com Sistemas de Ficheiros, considere SF, hipotético (mas em termos de funções **`fs_*`** e **`disk_*`** muito similar ao trabalho prático realizado nas aulas), e cuja representação em disco é a seguinte:

Em que: SBp e SBr são, respectivamente, os superblocos primário e secundário; TabInodes é a tabela onde estão armazenados os inodes; e, finalmente, Db0,…,Dbn são os blocos disponíveis para dados ou outros fins, se aplicável.

Premissas: i) os blocos são de 512 bytes; ii) os superblocos têm exactamente a mesma estrutura e dimensão, sendo esta de 1 bloco. iii) cada inode.

As estruturas relevantes são:

```
typedef struct {
   unsigned int magic;
   unsigned int diskSize;   // Dimensão (em blocos) do disco
   unsigned int InTabSize;  // Dimensão (em blocos) da Tabela de inodes
   unsigned int timestamp;  // Timestamp da última escrita neste SB
} OnDiskSuperBlock;

typedef struct {
   unsigned int valid;      // Inode valido se 1, inválido se 0
   unsigned int fsize;      // Dimensão do ficheiro, em bytes
   unsigned int dptr[4];    // Apontadores directos
   unsigned int iptr[2];    // Apontadores indirectos (um nível)
} OnDiskInode;

typedef struct {
  OnDiskInode inodes[INODES_PER_BLOCK];
} OnDiskInodeBlock;
```

Considere ao seguintes protótipos de funções, já implementadas:

```
int disk_read(unsigned int nBloco, unsigned char *data);
int disk_write(unsigned int nBloco, unsigned char *data);
unsigned int getDiskSize();
```

As funções **`disk_read`** e **`disk_write`** permitem ler ou escrever um bloco, dado o seu número; a função **`getDiskSize`** permite obter a dimensão do disco (em blocos). As funções **`disk_read`** e **`disk_write`** retornam -1 em caso de erro.

Finalmente, como um bloco pode "conter" um superbloco ou inodes ou simplesmente "dados", definimos

```
union fs_block {
  OnDiskSuperBlock sb;
  OnDiskInodeBlock inob;
  unsigned char [512] data;
}
```

D1) Complete, preenchendo o espaço vazio, o **#define** que calcula quantos inodes cabem num bloco de disco:

```
#define INODES_PER_BLOCK
```

D2) Complete a função para formatar um disco em "MeuFS", preenchendo os espaços vazios. O argumento diz quantos inodes se quer ter na tabela.

```
int fs_format(unsigned int maxNumberOfInodes) {
  union fs_block block;
  unsigned char zeroedBlock[512];  //Um bloco só com 0s, use-o se quiser
  unsigned int InTabSize;    //Para calcular a dimensão em blocos da Tabela de Inodes
  unsigned int InTabStart;   //Para especificar o bloco onde começa a Tabela de Inodes

  InTabSize=ceil(                                                    );

  memset(block.data, 0, 512);      // "zerar" o bloco fs inteiro
  memset(zeroedBlock, 0, 512);      // preparar um bloco só com zeros

  block.sb.magic= 0x01234567;
  block.sb.InTabSize= InTabSize;

  block.sb.diskSize=                                  ;

  // Escrever o 1° superbloco
  block.sb.timestamp= getTime();
  disk_write                                  

  // Escrever a Tabela de inodes. Nota: lembre-se que bits a zero
  // representam inodes livres

  inTabStart=                                  // Bloco onde começa a TabInodes

  for (int i= 0; i <                 ; i++) {

    disk_write                                  ) ;

  }

  // Escrever o 2° superbloco

                                                                    

}
```

D3) Considere que o disco está montado (a informação do superbloco já está em memória) mas a Tabela de inodes ainda não foi acedido; depois, um ficheiro, p.ex. descrito pelo inode x foi aberto, mas não se acedeu ao seu conteúdo.

Descreva que estruturas de dados do SF (SBp, TabInodes, SBr) são acedidas para satisfazer a seguinte instrução, **`fs_read(inode, data, offset, length)`**, e as operações que são efectuadas - sobre cada estrutura, e nos restantes blocos do disco

Estruturas acedidas

Operações realizadas para satisfazer o fs_read (incluir validações e erros)

D4) Descreva os aspectos essenciais, do ponto de vista *hardware* e *software*, do funcionamento de um periférico e seu correspondente *driver* quando este (o software do *driver*) efectua o controle do periférico por espera activa. **Sugestão**: use um modelo simples de um periférico simples – o chamado modelo **canónico** do periférico.

Hardware: registos fundamentais para funcionamento do periférico

Software: acções fundamentais para controlar o funcionamento do periférico