Sistemas Operacionais

Trabalho prático II

Sistema de Arquivos T2FS

Professor Alexandre Carissimi

Caroline Knorr Carvalho – 00229753

Guilherme de Oliveira Tassinari – 00231060

Universidade Federal do Rio Grande do Sul

- Identify2

```
int identify2 (char *name, int size){
 char * identification = "Guilherme 00231060 | Caroline 00229753";
 int identificationSize = strlen(identification)*sizeof(char);
 if(size < identificationSize) return ERROR;
 strcpy(name, identification);
 return SUCCESS;
}
```

  Em funcionamento: sim.

- Create2
  Em funcionamento: Não.

- Delete2
  Em funcionamento: Não.

- Open2
  Em funcionamento: Não.

- Close2
  Em funcionamento: Não.

- Read2
  Em funcionamento: Não.

- Write2
  Em funcionamento: Não.

- Truncate2
  Em funcionamento: Não.

- Seek2
  Em funcionamento: Não.

- Mkdir2
  Em funcionamento: Não.

- Rmdir2

Em funcionamento: Não.

- Opendir2
  Em funcionamento: Não.

- Readdir2
  Em funcionamento: Não.

- Closedir2

  Em funcionamento: Não.

Apesar de não terem sido implementadas as funcionalidades, foi possível implementar interfaces para a leitura e escrita de Inodes, blocos, pesquisa de records em blocos e leitura do superbloco.

- Testes
  - Block:

```
int main(){
  char * identification = "Guilherme 00231060 | Caroline 00229753";
  memcpy(blockBuffer, identification, strlen(identification)*sizeof(char));
  int blockPosition = writeBlock(blockBuffer);
  blockBuffer = malloc(blockSize());
  readBlock(blockPosition, blockBuffer);
  int i;
  for(i = 0; i <= strlen(identification); i++){
    printf("%c", blockBuffer[i]);
  }
}
```

O teste acima é responsável por garantir o funcionamento da leitura e escrita de blocos no disco. Primeiramente criamos um bloco contendo, nos seus primeiros endereços, a string de identificação da dupla do projeto. Após, o mesmo é escrito em disco com a função `writeBlock` , que retorna o índice onde o bloco foi escrito. Então, através de `readBlock` , lemos o bloco novamente do disco e printamos na tela as primeiras posições, onde deve estar a string de identificação. Espera-se ao final do teste, que o console mostre a string de identificação.

- Inode

```
int main(){
  struct t2fs_inode * inodeBuffer = malloc(sizeof(struct t2fs_inode));
  inodeBuffer->dataPtr[0] = 1;
  inodeBuffer->dataPtr[1] = 2;
  inodeBuffer->singleIndPtr = 3;
  inodeBuffer->doubleIndPtr = 4;
  int inodeIndex = writeInode(inodeBuffer);
  inodeBuffer = malloc(sizeof(struct t2fs_inode));
  readInode(inodeIndex, inodeBuffer);
  printf("%d - %d - %d - %d\n", inodeBuffer->dataPtr[0], inodeBuffer->dataPtr[1],
inodeBuffer->singleIndPtr, inodeBuffer->doubleIndPtr);
}
```

O teste acima é responsável por garantir as funcionalidades de leitura e escrita de Inodes em disco. Primeiramente é criado um inode, cujos ponteiros possuem, respectivamente, os valores 1, 2, 3 e 4. O Inode é escrito em disco e, logo após, lido novamente, para então ter seus ponteiros printados em tela. Espera-se que o console mostre, em ordem, os valores 1, 2, 3 e 4, que são os valores dos ponteiros iniciais.

- Superblock

```c
int main(){
  printf("\n");
  testId();
  testVersion();
  testSuperBlockSize();
  testFreeBlocksBitmapSize();
  testFreeInodeBitmapSize();
  testInodeAreaSize();
  testBlockSize();
  testDiskSize();
  printf("\n");
}


int testId(){
  char * buffer = malloc(sizeof(char)*4);
  getId(buffer);
  printf("ID : %c%c%c%c\n", buffer[0], buffer[1], buffer[2], buffer[3]);
}


int testVersion(){
  WORD * buffer = malloc(sizeof(WORD));
  getVersion(buffer);
  printf("Version : %d\n", (short)(*buffer));
}


int testSuperBlockSize(){
  WORD * buffer = malloc(sizeof(WORD));
  getSuperBlockSize(buffer);
  printf("Super Block Size : %d\n", (short)(*buffer));
}


int testFreeBlocksBitmapSize(){
  WORD * buffer = malloc(sizeof(WORD));
  getFreeBlocksBitmapSize(buffer);
  printf("Free Blocks Bitmap Size : %d\n", (short)(*buffer));
}


int testFreeInodeBitmapSize(){
  WORD * buffer = malloc(sizeof(WORD));
  getFreeInodeBitmapSize(buffer);
  printf("Free Inodes Bitmap Size : %d\n", (short)(*buffer));
```

O código acima é responsável por garantir a funcionalidade de leitura e fornecimento de todos os dados contidos no superbloco do disco. Um a um, todos os campos são lidos e printados em tela. Espera-se que todos sejam mostrados corretamente no console.

- Teste1

```
int main(){
  char nomes[100];
  if(identify2(nomes, (sizeof(char)*100)) == 0){
    printf("\n%s\n", nomes);
  } else {
    printf("ERRO");
 }
}
```

Teste responsável por garantir o funcionamento da funcionalidade Identify2. Se funcionando corretamente, deve printar no console a string de identificação, senão, deve printar “ERRO”.

- Dificuldades:
  - Complexidade do sistema:

    O sistema de arquivos é complexo no seu entendimento e implementação. Mesmo para alunos mais experientes em programação, foi difícil compreender exatamente como implementar suas funcionalidades.
  - Tempo para implementação:

    Mesmo com duplas e utilizando serviços de versionamento e gerenciamento de código, falta tempo hábil para executar o projeto, visto que os componentes também trabalham e possuem outras cadeiras e projetos para executar.