**Universidade do Estado de Santa Catarina – Bacharelado em Ciência da Computação**
**Sistemas Operacionais – Prof. Rafael R. Obelheiro**

# Laboratório – Sistema de arquivos ext2

## 1 Objetivo

O objetivo desta atividade de laboratório é ilustrar o funcionamento das estruturas internas do sistema de arquivos ext2, usado no Linux. O ext2 utiliza alocação indexada (inodos), seguindo a organização clássica de sistemas de arquivos UNIX, e é a base do atual ext4.

## 2 Roteiro

A atividade deve ser desenvolvida em um sistema Linux, em uma janela de terminal. Siga as etapas descritas abaixo, executando os comandos para observar seu funcionamento (o que precisa ser digitado está destacado em negrito). Em caso de dificuldades, não hesite em pedir ajuda ao professor.

1. Baixar o arquivo `disk.zip` do Moodle e descompactá-lo para obter a imagem `disk.img`:

```
$ unzip disk.zip
```

2. A imagem `disk.img` contém um sistema de arquivos ext2. Caso haja permissão (geralmente é necessário acesso de `root`, possivelmente via `sudo`), essa imagem pode ser montada:

```
$ mkdir mnt
$ sudo mount -r -o loop disk.img mnt
```

3. O conteúdo desse sistema de arquivos pode ser visto abaixo:

```
$ cd mnt
$ ls -li
total 13
  11 drwx------ 2 root  root  12288 Nov 21 11:19 lost+found
1281 drwxr-xr-x 3 udesc udesc  1024 Nov 21 11:19 udesc

$ cd udesc
$ ls -lRai
.:
total 716
1281 drwxr-xr-x 3 udesc udesc   1024 Nov 21 11:19 .
   2 drwxr-xr-x 4 udesc udesc   1024 Nov 21 11:19 ..
1282 -rw-rw-r-- 1 udesc udesc      9 Nov 21 11:19 arq1
1283 -rw-rw-r-- 1 udesc udesc      7 Nov 21 11:19 arq2
1284 -rw-rw-r-- 2 udesc udesc      7 Nov 21 11:19 arq3
1285 -rw-rw-r-- 1 udesc udesc  26624 Nov 21 11:19 arq4
1287 -rw-rw-r-- 1 udesc udesc     10 Nov 21 11:19 arq6
1289 -rw-rw-r-- 1 udesc udesc      7 Nov 21 11:19 arq8
1290 -rw-rw-r-- 1 udesc udesc 692224 Nov 21 11:19 arq9
  12 drwxrwxr-x 2 udesc udesc   1024 Nov 21 11:19 dir1
1284 -rw-rw-r-- 2 udesc udesc      7 Nov 21 11:19 hardlink
1291 lrwxrwxrwx 1 udesc udesc      4 Nov 21 11:19 symlink -> arq1

./dir1:
total 5
  12 drwxrwxr-x 2 udesc udesc 1024 Nov 21 11:19 .
1281 drwxr-xr-x 3 udesc udesc 1024 Nov 21 11:19 ..
```

```
28   13 -rw-rw-r-- 1 udesc udesc    5 Nov 21 11:19 arq1
29   14 -rw-rw-r-- 1 udesc udesc    8 Nov 21 11:19 arq2
30   15 -rw-rw-r-- 1 udesc udesc    8 Nov 21 11:19 arq3
```

O diretório raiz tem dois subdiretórios, `lost+found` e `udesc`. O conteúdo do diretório `udesc` é mostrado nas linhas 9 a 22, e o do subdiretório `dir1` nas linhas 24–30. A listagem de diretório tem o seguinte formato:

| inodo | tipo/ permissões | n° de ligações | dono | grupo | tamanho (bytes) | última modificação | nome do arquivo |
|---|---|---|---|---|---|---|---|
| 1283 | -rw-rw-r-- | 1 | udesc | udesc | 7 | 21-Nov-2016 11:19 | arq2 |

4. A imagem pode ser desmontada com `umount` (normalmente requer `root`):

```
1  $ cd
2  $ sudo umount mnt
```

5. A imagem pode ser aberta no `debugfs`, que é uma ferramenta de manipulação de baixo nível para sistemas de arquivos (neste caso, não é necessário acesso de `root`):

```
1  $ debugfs disk.img
2  debugfs 1.42.13 (17-May-2015)
```

6. O comando `stats` mostra estatísticas do superbloco e dos descritores de grupo:

```
1  debugfs:  stats
2  Filesystem volume name:   <none>
3  Last mounted on:          <not available>
4  Filesystem UUID:          8250faaf-d156-4c7c-b6cd-50596a89f3d0
5  Filesystem magic number:  0xEF53
6  Filesystem revision #:    1 (dynamic)
7  Filesystem features:      ext_attr resize_inode dir_index filetype sparse_super
8  Filesystem flags:         signed_directory_hash
9  Default mount options:    user_xattr acl
10 Filesystem state:         clean
11 Errors behavior:          Continue
12 Filesystem OS type:       Linux
13 Inode count:              2560
14 Block count:              10240
15 Reserved block count:     512
16 Free blocks:              9102
17 Free inodes:              2536
18 First block:              1
19 Block size:               1024
20 Fragment size:            1024
21 Reserved GDT blocks:      39
22 Blocks per group:         8192
23 Fragments per group:      8192
24 Inodes per group:         1280
25 Inode blocks per group:   160
26 Filesystem created:       Mon Nov 21 11:19:17 2016
27 Last mount time:          n/a
28 Last write time:          Mon Nov 21 11:19:18 2016
29 Mount count:              1
30 Maximum mount count:      -1
31 Last checked:             Mon Nov 21 11:19:17 2016
32 Check interval:           0 (<none>)
33 Reserved blocks uid:      0 (user root)
```

```
34 Reserved blocks gid:      0 (group root)
35 First inode:              11
36 Inode size:               128
37 Default directory hash:   half_md4
38 Directory Hash Seed:      eb503ec1-0bc7-4ca1-baad-fb22a1baf40f
39 Directories:              4
40  Group  0: block bitmap at 42, inode bitmap at 43, inode table at 44
41            7801 free blocks, 1265 free inodes, 3 used directories
42  Group  1: block bitmap at 8234, inode bitmap at 8235, inode table at 8236
43            1301 free blocks, 1271 free inodes, 1 used directory
```

As informações mais relevantes são as seguintes:

- `Inode count`, `Block count` (linhas 13–14): número de inodos e blocos no SA
- `Reserved block count` (l. 15): número de blocos reservados para o usuário `root`
  - margem de segurança para evitar travamento do sistema em caso de disco cheio
  - em geral 5% do número total de blocos ($0{,}05 \times 10240 = 512$)
- `Free blocks`, `Free inodes` (l. 16–17): número de blocos e inodos livres
- `Block size` (l. 19): tamanho de bloco (1024 bytes = 1 KB)
- `Inode size` (l. 37): tamanho de um inodo, em bytes
- Informações sobre grupos de blocos
  - `Blocks per group`, `Inodes per group` (l. 22, 24): número de blocos e inodos em cada grupo de blocos
  - `Inode blocks per group` (l. 25): blocos ocupados por inodos em cada grupo
    - um bloco de 1 KB pode abrigar 8 inodos de 128 bytes
    - os 1280 inodos de um grupo ocupam $1280 \div 8 = 160$ blocos
  - linhas 41–44: informações sobre cada grupo de blocos
    - `block bitmap at` *nnn*: bloco inicial do mapa de bits dos blocos de dados
    - `inode bitmap at` *nnn*: bloco inicial do mapa de bits dos inodos
    - `inode table at` *nnn*: bloco inicial da tabela de inodos
    - resumos de blocos/inodos livres e diretórios criados no grupo

7. Para listar arquivos e navegar na árvore de diretórios são usados os comandos `ls` e `cd`:

```
1 debugfs:  ls
2  2  (12) .    2  (12) ..    11  (20) lost+found    1281  (980) udesc
```

O comando `ls` gera uma listagem compacta, onde cada arquivo é representado por uma tripla

*inodo (tam_entrada) nome*

*inodo* é o número do inodo, *tam_entrada* é o tamanho da entrada de diretório e *nome* é o nome do arquivo. No exemplo, o tamanho 980 da entrada de diretório `udesc` indica que ela é a última entrada do diretório em questão, pois $12 + 12 + 20 + 980 = 1024$, que é o tamanho do bloco em bytes (ou seja, a entrada `udesc` vai do byte 44 até o byte 1023, que é o final do bloco).

```
1 debugfs:  cd udesc
2 debugfs:  ls -l
3    1281   40755 (2)   1001   1001    1024 21-Nov-2016 11:19 .
4       2   40755 (2)   1001   1001    1024 21-Nov-2016 11:19 ..
5    1282  100664 (1)   1001   1001       9 21-Nov-2016 11:19 arq1
6    1283  100664 (1)   1001   1001       7 21-Nov-2016 11:19 arq2
7    1284  100664 (1)   1001   1001       7 21-Nov-2016 11:19 arq3
8    1285  100664 (1)   1001   1001   26624 21-Nov-2016 11:19 arq4
```

```
   1287  100664 (1)   1001   1001      10 21-Nov-2016 11:19 arq6
   1289  100664 (1)   1001   1001       7 21-Nov-2016 11:19 arq8
   1290  100664 (1)   1001   1001  692224 21-Nov-2016 11:19 arq9
   1291  120777 (7)   1001   1001       4 21-Nov-2016 11:19 symlink
   1284  100664 (1)   1001   1001       7 21-Nov-2016 11:19 hardlink
     12   40775 (2)   1001   1001    1024 21-Nov-2016 11:19 dir1
```

O comando `ls -l` gera uma listagem longa, onde as entradas têm o seguinte formato:

```
1283  100664  (1)  1001  1001      7  21-Nov-2016 11:19  arq2
 (a)    (b)    (c)   (d)   (e)    (f)          (g)          (h)
```

(a) `1283`: número do inodo

(b) `100664`: modo do inodo → tipo do arquivo (1-2 dígitos) + permissões (4 dígitos)

- `10`: arquivo regular
  Os tipos de arquivo presentes no modo podem ser
  1 FIFO (usado para comunicação interprocessos)
  2 arquivo especial de caracter
  4 diretório
  6 arquivo especial de bloco
  10 arquivo regular
  12 ligação simbólica
  14 *socket*
- `664`: permissões para dono/grupo/outros (`rw-rw-r--`)

(c) `(1)`: tipo de arquivo na entrada de diretório – arquivo regular
Este atributo é redundante com o presente no modo do inodo, mas usa valores diferentes:
0 desconhecido
1 arquivo regular
2 diretório
3 arquivo especial de caracter
4 arquivo especial de bloco
5 FIFO (usado para comunicação interprocessos)
6 *socket*
7 ligação simbólica

(d) `1001`: ID do dono do arquivo (UID)

(e) `1001`: ID do grupo do arquivo (GID)

(f) `7`: tamanho do arquivo, em bytes

(g) `21-Nov-2016 11:19`: *timestamp* da última modificação

(h) `arq2`: nome do arquivo

8. O comando `cat` mostra o conteúdo de um arquivo usando o nome ou o número do inodo:

```
debugfs:  cat arq1
bacalhau

debugfs:  cat <1282>
bacalhau
```

9. O comando `stat` mostra o conteúdo de um inodo (o nome do arquivo também pode ser usado como parâmetro):

```
debugfs:  stat <1282>
Inode: 1282   Type: regular    Mode:  0664   Flags: 0x0
Generation: 2236643185    Version: 0x00000000
```

```
User:  1001   Group:  1001   Size: 9
File ACL: 0    Directory ACL: 0
Links: 1   Blockcount: 2
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x5832f456 -- Mon Nov 21 11:19:18 2016
atime: 0x5832f456 -- Mon Nov 21 11:19:18 2016
mtime: 0x5832f456 -- Mon Nov 21 11:19:18 2016
BLOCKS:
(0):8705
TOTAL: 1
```

Os blocos de dados usados pelo arquivo são listados em BLOCKS (linhas 11–13).

10. O arquivo hardlink é uma ligação estrita para o inodo 1284, também referenciado pelo arquivo arq3. Na linha 12, a contagem de ligações para esse inodo aparece como 2:

```
debugfs:  cat arq3
baiacu

debugfs:  cat hardlink
baiacu

debugfs:  stat arq3
Inode: 1284   Type: regular    Mode:  0664   Flags: 0x0
Generation: 2236643187    Version: 0x00000000
User:  1001   Group:  1001   Size: 7
File ACL: 0    Directory ACL: 0
Links: 2   Blockcount: 2
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 0x5832f456 -- Mon Nov 21 11:19:18 2016
atime: 0x5832f456 -- Mon Nov 21 11:19:18 2016
mtime: 0x5832f456 -- Mon Nov 21 11:19:18 2016
BLOCKS:
(0):8707
TOTAL: 1
```

11. Fora do debugfs, pode-se extrair o bloco de dados usado pelo arquivo arq3, salvá-lo em um arquivo e exibir seu conteúdo. O comando

    $ dd if=*arqent* of=*arqsai* bs=*tambl* count=*numbl* skip=*binic*

    lê *numbl* blocos (de tamanho *tambl*) do arquivo *arqent*, começando pelo bloco *binic*, e escreve-os no arquivo *arqsai*. O comando hd faz um *dump* do conteúdo do arquivo em hexadecimal e ASCII.

```
$ dd if=disk.img of=bloco-8707 bs=1K count=1 skip=8707
1+0 records in
1+0 records out
1024 bytes (1,0 kB, 1,0 KiB) copied, 0,00015151 s, 6,8 MB/s

$ hd bloco-8707
00000000  62 61 69 61 63 75 0a 00  00 00 00 00 00 00 00 00  |baiacu..........|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000400

$ cat bloco-8707
baiacu
```

12. Arquivos que ocupam até $12 \times 1024 = 12.288$ bytes têm seus endereços de bloco nos ponteiros diretos do inodo. O arquivo `arq4` tem 26 KB, o que corresponde a 26 blocos de dados. Isso significa que os endereços dos 12 primeiros blocos estão contidos diretamente no inodo, e os endereços dos 14 blocos seguintes estão em um bloco de indireção simples. A saída do comando `stat` contém a lista de blocos usados:

```
1  debugfs:  stat arq4
2  Inode: 1285   Type: regular    Mode:  0664   Flags: 0x0
3  Generation: 2236643188    Version: 0x00000000
4  User:  1001   Group:  1001   Size: 26624
5  File ACL: 0    Directory ACL: 0
6  Links: 1   Blockcount: 54
7  Fragment:  Address: 0    Number: 0    Size: 0
8  ctime: 0x5832f456 -- Mon Nov 21 11:19:18 2016
9  atime: 0x5832f456 -- Mon Nov 21 11:19:18 2016
10 mtime: 0x5832f456 -- Mon Nov 21 11:19:18 2016
11 BLOCKS:
12 (0-11):9217-9228, (IND):9229, (12-25):9230-9243
13 TOTAL: 27
```

Os 12 primeiros blocos de dados (blocos 0–11) vão de 9217 a 9228 (linha 12). Os últimos 14 blocos (12–25) vão de 9230 a 9243. O bloco 9229 é o bloco de indireção simples, denotado por `(IND)`. No total, o arquivo ocupa 27 blocos (linha 13), os 26 de dados mais um bloco de índices.

É possível extrair (fora do `debugfs`) o bloco de indireção e exibir seu conteúdo:

```
1  $ dd if=disk.img of=bloco-9229 bs=1K count=1 skip=9229
2  1+0 records in
3  1+0 records out
4  1024 bytes (1,0 kB, 1,0 KiB) copied, 0,000155982 s, 6,6 MB/s
5
6  $ od -t u4 bloco-9229
7  0000000       9230       9231       9232       9233
8  0000020       9234       9235       9236       9237
9  0000040       9238       9239       9240       9241
10 0000060       9242       9243          0          0
11 0000100          0          0          0          0
12 *
13 0002000
```

O comando `od -t u4` exibe o conteúdo do arquivo como uma sequência de números inteiros sem sinal (`u`) de 4 bytes (`4`), que é o formato dos números de bloco no ext2. Percebe-se, portanto, que o bloco indireto simples contém a sequência dos números de blocos correspondentes aos blocos 12–25 do arquivo (9230–9243). Constata-se ainda que não há nenhum marcador indicando que as entradas do bloco de índices após 9243 são inválidas – o controle de fim de arquivo é feito pelo tamanho armazenado no inodo.

13. Cada bloco de índices pode armazenar $1024 \div 4 = 256$ números de bloco. Portanto, arquivos que ocupam mais de $12 + 256 = 268$ blocos precisam recorrer à indireção dupla. O arquivo `arq9` tem 692.224 bytes, o que corresponde a 676 blocos. Isso exige $\left\lceil \frac{676-268}{256} \right\rceil = 2$ blocos de índices de 2° nível. A listagem de blocos do inodo permite visualizar isso:

```
1  debugfs:  stat arq9
2  Inode: 1290   Type: regular    Mode:  0664   Flags: 0x0
3  Generation: 2236643193    Version: 0x00000000
4  User:  1001   Group:  1001   Size: 692224
5  File ACL: 0    Directory ACL: 0
6  Links: 1   Blockcount: 1360
7  Fragment:  Address: 0    Number: 0    Size: 0
```

```
8  ctime: 0x5832f456 -- Mon Nov 21 11:19:18 2016
9  atime: 0x5832f456 -- Mon Nov 21 11:19:18 2016
10 mtime: 0x5832f456 -- Mon Nov 21 11:19:18 2016
11 BLOCKS:
12 (0-11):9729-9740, (IND):9741, (12-267):9742-9997, (DIND):9998, (IND):9999,
       (268-507):10000-10239, (508-523):219-234, (IND):235, (524-675):236-387
13 TOTAL: 680
```

Os 676 blocos de dados são subdivididos em 12 diretos (0–11), 256 indiretos simples (12–267) e 408 indiretos duplos (268–675). No total, o arquivo ocupa 680 blocos (linha 13), sendo 676 de dados e 4 de índices. A linha 12 mostra que o bloco 9998 é o bloco de indireção dupla (`DIND`), e os blocos 9999 e 235 são os blocos de índices de 2° nível (que contêm os endereços dos blocos 268–675). Repetindo o processo usado anteriormente (`dd`/`od`) é possível inspecionar esses blocos:

```
1  $ dd if=disk.img of=bloco-9998 bs=1K count=1 skip=9998
2  1+0 records in
3  1+0 records out
4  1024 bytes (1,0 kB, 1,0 KiB) copied, 0,000170079 s, 6,0 MB/s
5
6  $ dd if=disk.img of=bloco-9999 bs=1K count=1 skip=9999
7  1+0 records in
8  1+0 records out
9  1024 bytes (1,0 kB, 1,0 KiB) copied, 0,000151548 s, 6,8 MB/s
10
11 $ dd if=disk.img of=bloco-235 bs=1K count=1 skip=235
12 1+0 records in
13 1+0 records out
14 1024 bytes (1,0 kB, 1,0 KiB) copied, 0,00020152 s, 5,1 MB/s
15
16 $ od -t u4 bloco-9998
17 0000000       9999        235          0          0
18 0000020          0          0          0          0
19 *
20 0002000
21
22 $ od -t u4 bloco-9999
23 0000000      10000      10001      10002      10003
24 0000020      10004      10005      10006      10007
25 ...
26 0001640      10232      10233      10234      10235
27 0001660      10236      10237      10238      10239
28 0001700        219        220        221        222
29 0001720        223        224        225        226
30 0001740        227        228        229        230
31 0001760        231        232        233        234
32 0002000
33
34 $ od -t u4 bloco-235
35 0000000        236        237        238        239
36 0000020        240        241        242        243
37 ...
38 0001100        380        381        382        383
39 0001120        384        385        386        387
40 0001140          0          0          0          0
41 *
42 0002000
```

O bloco 9998, que é o bloco de indireção dupla, contém apenas dois endereços para os blocos de índices de 2° nível, que são os blocos 9999 e 235 (linhas 16–20). O bloco 9999 contém os números dos blocos de dados 268–523, que são os blocos de 10000 a 10239 e de 219 a 234 (linhas 22–32). O

bloco 251 contém os números dos blocos de dados 524–675, que são 236–387 (linhas 34–42). As linhas com `...` contém números de blocos em sequência, e foram suprimidas do documento por questão de clareza.

14. O arquivo com nome `symlink` é uma ligação simbólica para `arq1`. Uma ligação simbólica tem inodo próprio (diferente do alvo da ligação) e é de um tipo diferente de arquivo. Além disso, caso o alvo da ligação seja um nome de caminho com até 60 bytes, o arquivo de ligação não ocupa blocos de dados, pois o nome alvo é armazenado nos 15 números de bloco disponíveis no próprio inodo. A saída de `stat` traz o nome alvo no lugar da lista de blocos (linha 11); um *dump* do inodo (linhas 13–20), obtido com o comando `idump`, confirma que o nome é armazenado no próprio inodo (linha 16):

```
1  debugfs:  stat symlink
2  Inode: 1291   Type: symlink    Mode:  0777   Flags: 0x0
3  Generation: 2236643194    Version: 0x00000000
4  User:  1001   Group:  1001   Size: 4
5  File ACL: 0    Directory ACL: 0
6  Links: 1   Blockcount: 0
7  Fragment:  Address: 0    Number: 0    Size: 0
8  ctime: 0x5832f456 -- Mon Nov 21 11:19:18 2016
9  atime: 0x5832f456 -- Mon Nov 21 11:19:18 2016
10 mtime: 0x5832f456 -- Mon Nov 21 11:19:18 2016
11 Fast_link_dest: arq1
12
13 debugfs:  idump <1291>
14 0000  ffa1 e903 0400 0000 56f4 3258 56f4 3258  ........V.2XV.2X
15 0020  56f4 3258 0000 0000 e903 0100 0000 0000  V.2X............
16 0040  0000 0000 0000 0000 6172 7131 0000 0000  ........arq1....
17 0060  0000 0000 0000 0000 0000 0000 0000 0000  ................
18 *
19 0140  0000 0000 7a77 5085 0000 0000 0000 0000  ....zwP.........
20 0160  0000 0000 0000 0000 0000 0000 0000 0000  ................
```

15. Os diretórios são um tipo particular de arquivo, formados por listas encadeadas de registros de tamanho variável. O comando abaixo salva o conteúdo do diretório `udesc` (inodo 1281) no arquivo `dump-udesc`:

```
1  debugfs:  dump <1281> dump-udesc
```

O programa `dumpdir.c`, disponível no Moodle, é capaz de exibir o conteúdo de um diretório salvo usando o comando `dump`:

```
1  $ cc -Wall -o dumpdir dumpdir.c
2  $ ./dumpdir dump-udesc
3  dump-udesc:
4      #  inode reclen name_len file_type  name
5    000:  1281     12        1    2/dir   .
6    001:     2     12        2    2/dir   ..
7    002:  1282     12        4    1/reg   arq1
8    003:  1283     12        4    1/reg   arq2
9    004:  1284     12        4    1/reg   arq3
10   005:  1285     24        4    1/reg   arq4
11   006:  1287     24        4    1/reg   arq6
12   007:  1289     12        4    1/reg   arq8
13   008:  1290     12        4    1/reg   arq9
14   009:  1291     16        7    7/syml  symlink
15   010:  1284     16        8    1/reg   hardlink
16   011:    12    860        4    2/dir   dir1
```

16. Embora geralmente não seja possível recuperar arquivos apagados em um sistema de arquivos Ext2, o conhecimento da estrutura interna do SA permite, em alguns casos, essa recuperação. Suponha que o usuário apagou erroneamente um arquivo cujo conteúdo era `borriquete`; ele não sabe ao certo o nome do arquivo, apenas que ele estava no subdiretório `udesc`.[1] No `debugfs`, o comando `lsdel` lista os inodos apagados (e recuperáveis) de um sistema de arquivos:

```
1  debugfs:  lsdel
2   Inode  Owner  Mode    Size      Blocks   Time deleted
3    1286   1001 100664      6      1/     1 Mon Nov 21 11:19:18 2016
4    1288   1001 100664     11      1/     1 Mon Nov 21 11:19:18 2016
5  2 deleted inodes found.
6
7  debugfs:  cat <1286>
8  bagre
9
10 debugfs:  cat <1288>
11 borriquete
```

Os inodos 1286 e 1288 aparecem como apagados. Uma inspeção do conteúdo revela que o arquivo em questão tinha o inodo 1288.

É possível tentar descobrir o nome do arquivo:

```
1  debugfs:  cd /udesc
2  debugfs:  ls -ld
3     1281   40755 (2)   1001   1001    1024 21-Nov-2016 11:19 .
4        2   40755 (2)   1001   1001    1024 21-Nov-2016 11:19 ..
5     1282  100664 (1)   1001   1001       9 21-Nov-2016 11:19 arq1
6     1283  100664 (1)   1001   1001       7 21-Nov-2016 11:19 arq2
7     1284  100664 (1)   1001   1001       7 21-Nov-2016 11:19 arq3
8     1285  100664 (1)   1001   1001   26624 21-Nov-2016 11:19 arq4
9  <     0>      0 (1)      0      0       0                   arq5
10    1287  100664 (1)   1001   1001      10 21-Nov-2016 11:19 arq6
11 <     0>      0 (1)      0      0       0                   arq7
12    1289  100664 (1)   1001   1001       7 21-Nov-2016 11:19 arq8
13    1290  100664 (1)   1001   1001  692224 21-Nov-2016 11:19 arq9
14    1291  120777 (7)   1001   1001       4 21-Nov-2016 11:19 symlink
15    1284  100664 (1)   1001   1001       7 21-Nov-2016 11:19 hardlink
16      12   40775 (2)   1001   1001    1024 21-Nov-2016 11:19 dir1
```

Observa-se que os arquivos `arq5` e `arq7` foram apagados (linhas 9 e 11). Como o número do inodo é zerado quando um arquivo é removido, não é possível saber com certeza qual o inodo correspondente a cada um deles, embora provavelmente `arq7` correspondesse ao inodo 1288 (os inodos parecem ter sido atribuídos na mesma sequência dos nomes de arquivo).

Portanto, deseja-se recuperar o arquivo `udesc/arq7`, cujo inodo é 1288. Para isso é necessário modificar a imagem, o que exige que o `debugfs` seja invocado com a opção `-w`. O comando `mi` permite modificar um inodo de forma interativa; é preciso mudar os seguintes campos:

- `Deletion time: 0`
- `Link count: 1`

Para os demais campos, basta apertar Enter.

[1] Obviamente neste caso seria muito mais simples apenas recriar o arquivo, uma vez que seu conteúdo é conhecido. O procedimento aqui apresentado, porém, pode ser usado com arquivos mais complexos.

```
debugfs:  mi <1288>
mi: Filesystem opened read/only
debugfs:  quit

$ debugfs -w disk.img
debugfs 1.42.13 (17-May-2015)
debugfs:  mi <1288>
                          Mode    [0100664]
                       User ID    [1001]
                      Group ID    [1001]
                          Size    [11]
                 Creation time    [1479734358]
             Modification time    [1479734358]
                   Access time    [1479734358]
                 Deletion time    [1479734358] 0
                    Link count    [0] 1
               Block count high    [0]
                    Block count    [2]
                     File flags    [0x0]
                     Generation    [0x85507777]
                       File acl    [0]
           High 32bits of size    [0]
              Fragment address    [0]
               Direct Block #0    [8710]
               Direct Block #1    [0]
               Direct Block #2    [0]
               Direct Block #3    [0]
               Direct Block #4    [0]
               Direct Block #5    [0]
               Direct Block #6    [0]
               Direct Block #7    [0]
               Direct Block #8    [0]
               Direct Block #9    [0]
              Direct Block #10    [0]
              Direct Block #11    [0]
                 Indirect Block    [0]
         Double Indirect Block    [0]
         Triple Indirect Block    [0]
```

Neste momento, o inodo foi restabelecido, mas é preciso definir uma entrada de diretório para ele, o que pode ser feito com o comando `ln`:

```
debugfs:  ln <1288> arq7
debugfs:  ls -ld
   1281   40755 (2)   1001   1001    1024 21-Nov-2016 11:19 .
      2   40755 (2)   1001   1001    1024 21-Nov-2016 11:19 ..
   1282  100664 (1)   1001   1001       9 21-Nov-2016 11:19 arq1
   1283  100664 (1)   1001   1001       7 21-Nov-2016 11:19 arq2
   1284  100664 (1)   1001   1001       7 21-Nov-2016 11:19 arq3
   1285  100664 (1)   1001   1001   26624 21-Nov-2016 11:19 arq4
   1288  100664 (1)   1001   1001      11 21-Nov-2016 11:19 arq7
   1287  100664 (1)   1001   1001      10 21-Nov-2016 11:19 arq6
<     0>      0 (1)      0      0       0                    arq7
   1289  100664 (1)   1001   1001       7 21-Nov-2016 11:19 arq8
   1290  100664 (1)   1001   1001  692224 21-Nov-2016 11:19 arq9
   1291  120777 (7)   1001   1001       4 21-Nov-2016 11:19 symlink
   1284  100664 (1)   1001   1001       7 21-Nov-2016 11:19 hardlink
     12   40775 (2)   1001   1001    1024 21-Nov-2016 11:19 dir1
```

Foi atribuída uma entrada `arq7` no diretório `udesc`, apontando para o inodo 1288 (linha 9). Com as modificações efetuadas, porém, as informações de gerenciamento de alocação de blocos e inodos estão inconsistentes. É necessário, portanto, reparar o sistema de arquivos:

```
1  debugfs:  quit
2
3  $ fsck.ext2 -f disk.img
4  e2fsck 1.42.13 (17-May-2015)
5  Pass 1: Checking inodes, blocks, and sizes
6  Pass 2: Checking directory structure
7  Pass 3: Checking directory connectivity
8  Pass 4: Checking reference counts
9  Pass 5: Checking group summary information
10 Block bitmap differences:  +8710
11 Fix<y>? yes
12
13 Free blocks count wrong for group #1 (1301, counted=1300).
14 Fix<y>? yes
15
16 Free blocks count wrong (9102, counted=9101).
17 Fix<y>? yes
18
19 Inode bitmap differences:  +1288
20 Fix<y>? yes
21
22 Free inodes count wrong for group #1 (1271, counted=1270).
23 Fix<y>? yes
24
25 Free inodes count wrong (2536, counted=2535).
26 Fix<y>? yes
27
28
29 disk.img: ***** FILE SYSTEM WAS MODIFIED *****
30 disk.img: 25/2560 files (4.0% non-contiguous), 1139/10240 blocks
31
32 $ debugfs disk.img
33 debugfs 1.42.13 (17-May-2015)
34 debugfs:  cat udesc/arq7
35 borriquete
```

Uma forma menos traumática de atingir o mesmo resultado seria usar o comando `undel`. Por exemplo, para recuperar o inodo 1286, associando-o ao arquivo `arq5`:

```
1  debugfs:  cd /udesc
2  debugfs:  undel <1286> arq5
```

Continua sendo recomendável executar `fsck.ext2` após modificar o sistema de arquivos.

## 3 Exercícios

Os exercícios abaixo devem ser realizados com a imagem `disk2.img`, disponível no Moodle.

1. Quantos blocos e inodos tem o sistema de arquivos? Qual o tamanho de bloco?
2. Quantos grupos de blocos tem o sistema de arquivos? Qual o tamanho desses grupos? Qual dos grupos está mais ocupado?
3. Qual o espaço disponível no sistema de arquivos? Quantos arquivos ainda podem ser criados?
4. O arquivo `gama` ocupa quantos inodos e quantos blocos de dados no total? Quantos blocos do arquivo são endereçados pela indireção simples, e quais são esses blocos de dados?
5. Extraia todos os blocos de dados usados pelo arquivo `delta`, e faça um *dump* do seu conteúdo em hexadecimal e ASCII.
6. Qual o tamanho do arquivo `alfa`? Enumere os blocos de dados usados por esse arquivo.
7. Salve o conteúdo do diretório em um arquivo e mostre-o usando o programa `dumpdir.c`. A seguir, remova o arquivo `beta` (use o comando `rm` do `debugfs`) e repita a análise do conteúdo do diretório. O que muda no conteúdo do diretório quando o arquivo é removido?
8. Recupere o arquivo `beta` **sem** usar o comando `undel`. Não se esqueça de reparar o sistema de arquivos ao final.