2022-03-21 filesystem

類型
個人作業

開放繳交
2022-03-21 12:08

繳交期限
2022-04-04

已繳交
29人

允許遲交
是

成績比重
0%

評分方式
直接打分數

說明
請根據以下程式修改，達成以下功能
1.將檔案系統的跟目錄改為下列結構

```
/--+ input (dir)
   |   |
   |   +-- a (file)
   |   +-- b (file)
   |
   + output (dir)
       |
       +-- add (file)
       +-- sub (file)
```

2.可以透過 echo **數字** > /input/a 和 echo **數字** >/input/b 來設定a和b的值，數值大小0~255之間
3.可以透過 cat /output/add取得a+b的值，透過cat /output/sub取得a-b的值

myfs.c

```c
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/module.h>
#include <linux/pagemap.h>      /* PAGE_CACHE_SIZE */
#include <linux/fs.h>           /* This is where libfs stuff is declared */
#include <asm/atomic.h>
#include <asm/uaccess.h>        /* copy_to_user */


```

```
10  MODULE_LICENSE("GPL");
11
12  #define MYFS_MAGIC 0x20210607
13
14
15  static struct inode *myfs_make_inode(struct super_block *sb, int mode)
16  {
17          struct inode *ret = new_inode(sb);
18
19          if (ret) {
20                  ret->i_mode = mode;
21                  ret->i_uid = ret->i_gid = 0;
22                  ret->i_blocks = 0;
23                  ret->i_atime = ret->i_mtime = ret->i_ctime = CURRENT_TIME;
24          }
25          return ret;
26  }
27
28  static int myfs_open(struct inode *inode, struct file *filp)
29  {
30          filp->private_data = inode->i_private;
31          return 0;
32  }
33
34  #define TMPSIZE 20
35
36  static ssize_t myfs_read_file(struct file *filp, char *buf,
37                  size_t count, loff_t *offset)
38  {
39          atomic_t *counter = (atomic_t *) filp->private_data;
40          int v, len;
41          char tmp[TMPSIZE];
42
43          v = atomic_read(counter);
44          if (*offset > 0)
45                  v -= 1;  /* the value returned when offset was zero */
46          else
47                  atomic_inc(counter);
48          len = snprintf(tmp, TMPSIZE, "%d\n", v);
49          if (*offset > len)
50                  return 0;
51          if (count > len - *offset)
52                  count = len - *offset;
53
54          if (copy_to_user(buf, tmp + *offset, count))
55                  return -EFAULT;
56          *offset += count;
57          return count;
58  }
59
60  static ssize_t myfs_write_file(struct file *filp, const char *buf,
61                  size_t count, loff_t *offset)
62  {
63          atomic_t *counter = (atomic_t *) filp->private_data;
64          char tmp[TMPSIZE];
65
66          if (*offset != 0)
67                  return -EINVAL;
68
69          if (count >= TMPSIZE)
70                  return -EINVAL;
71          memset(tmp, 0, TMPSIZE);
72          if (copy_from_user(tmp, buf, count))
```

```
                return -EFAULT;

        atomic_set(counter, simple_strtol(tmp, NULL, 10));
        return count;
}


static struct file_operations myfs_file_ops = {
        .open   = myfs_open,
        .read   = myfs_read_file,
        .write  = myfs_write_file,
};


static struct dentry *myfs_create_file (struct super_block *sb,
                struct dentry *dir, const char *name,
                atomic_t *counter)
{
        struct dentry *dentry;
        struct inode *inode;
        struct qstr qname;

        qname.name = name;
        qname.len = strlen (name);
        qname.hash = full_name_hash(name, qname.len);

        dentry = d_alloc(dir, &qname);
        if (! dentry)
                goto out;
        inode = myfs_make_inode(sb, S_IFREG | 0644);
        if (! inode)
                goto out_dput;
        inode->i_fop = &myfs_file_ops;
        inode->i_private = counter;

        d_add(dentry, inode);
        return dentry;

  out_dput:
        dput(dentry);
  out:
        return 0;
}


static struct dentry *myfs_create_dir (struct super_block *sb,
                struct dentry *parent, const char *name)
{
        struct dentry *dentry;
        struct inode *inode;
        struct qstr qname;

        qname.name = name;
        qname.len = strlen (name);
        qname.hash = full_name_hash(name, qname.len);
        dentry = d_alloc(parent, &qname);
        if (! dentry)
                goto out;

        inode = myfs_make_inode(sb, S_IFDIR | 0644);
        if (! inode)
                goto out_dput;
        inode->i_op = &simple_dir_inode_operations;
```

```
136         inode->i_fop = &simple_dir_operations;
137
138         d_add(dentry, inode);
139         return dentry;
140
141   out_dput:
142         dput(dentry);
143   out:
144         return 0;
145   }
146
147
148   static atomic_t counter, subcounter;
149
150   static void myfs_create_files (struct super_block *sb, struct dentry *root)
151   {
152         struct dentry *subdir;
153
154         atomic_set(&counter, 0);
155         myfs_create_file(sb, root, "counter", &counter);
156
157         atomic_set(&subcounter, 0);
158         subdir = myfs_create_dir(sb, root, "subdir");
159         if (subdir)
160                 myfs_create_file(sb, subdir, "subcounter", &subcounter);
161   }
162
163
164
165   static struct super_operations myfs_s_ops = {
166         .statfs         = simple_statfs,
167         .drop_inode     = generic_delete_inode,
168   };
169
170   static int myfs_fill_super (struct super_block *sb, void *data, int silent)
171   {
172         struct inode *root;
173         struct dentry *root_dentry;
174
175         sb->s_blocksize = PAGE_CACHE_SIZE;
176         sb->s_blocksize_bits = PAGE_CACHE_SHIFT;
177         sb->s_magic = MYFS_MAGIC;
178         sb->s_op = &myfs_s_ops;
179
180         root = myfs_make_inode (sb, S_IFDIR | 0755);
181         if (! root)
182                 goto out;
183         root->i_op = &simple_dir_inode_operations;
184         root->i_fop = &simple_dir_operations;
185
186         root_dentry = d_alloc_root(root);
187         if (! root_dentry)
188                 goto out_iput;
189         sb->s_root = root_dentry;
190
191         myfs_create_files (sb, root_dentry);
192         return 0;
193
194   out_iput:
195         iput(root);
196   out:
197         return -ENOMEM;
198   }
```

```
199
200 static struct dentry *myfs_get_super(struct file_system_type *fst,
201                 int flags, const char *devname, void *data)
202 {
203         return mount_bdev(fst, flags, devname, data, myfs_fill_super);
204 }
205
206 static struct file_system_type myfs_type = {
207         .owner          = THIS_MODULE,
208         .name           = "myfs",
209         .mount          = myfs_get_super,
210         .kill_sb        = kill_litter_super,
211 };
212
213 static int __init myfs_init(void)
214 {
215         return register_filesystem(&myfs_type);
216 }
217
218 static void __exit myfs_exit(void)
219 {
220         unregister_filesystem(&myfs_type);
221 }
222
223 module_init(myfs_init);
224 module_exit(myfs_exit);
```

檢視 / 修改我的作業

線上： 2116 人