

Project report on Implementation of File System on Raspberry Pi.

Members: Charmi Patel, Jayashree Pridhivi and Shashank Reddy

Instructor: Jing Li

Course: CS 630 - Operating Systems Design

**New Jersey Institute of Technology
323 Dr. Martin Luther King Blvd, Newark, NJ 08816**

1. Introduction

Nowadays Linux is an operating system of choice for many computing systems ranging from personal computers (PC), servers, mainframe computers and supercomputers to electronic devices known as embedded systems. Linux is a highly portable operating system; it can run on a variety of hardware architectures such as x86, PowerPC, MIPS, H8, SPARC, or ARM, to name just a few.

This portability also comes with limitations, however. It is no doubt that while the modular architecture of Linux makes it easy when porting to different types of architectures, a great deal of effort is required to build new kernel components in order to support a target platform. The Raspberry Pi platform is an example of a target device that Linux can be ported to run on it. General file system is one of the kernel modules that can be developed and mounted on Linux platform. Thus the goal of this project was to implement a File System on Raspberry Pi.

The intended result of this work was to get a deeper understanding on the Raspberry Pi platform, to learn what Linux file systems does and how it works, and finally to mount a file system for the Raspberry Pi platform which was coded from its scratch.

2. Raspberry Pi Model B Rev 2.0 Platform

The Raspberry Pi model B rev 2.0 platform was chosen for this project to implement the File System. This is due to the fact that Raspberry Pi is a low-cost Linux computer which costs approximately \$35 from the Farnell distributor. The platform is widely used by hobbyists and professional embedded developers around the world. Therefore, there is a large support from the community when it comes to getting help or information. For convenience, Raspberry Pi will be used when referring to the Raspberry Pi model B rev 2.0 platform throughout this report.

3. Milestones

In order to complete our project on time, we used Agile framework and broke the tasks down into 5 milestones. Our process is shown below.

Step	Milestones	Duration
1	Learning and preparing Raspberry Pi environment	3 weeks
2	Understanding the Linux Kernel	2 weeks
3	Implementing simple kernel module	1 weeks
4	File System coding and mounting	5 weeks
5	Testing and Reporting	2 week

4. VFS:

The Linux VFS supports multiple file systems. The kernel does most of the work while the file system specific tasks are delegated to the individual file systems through the handlers. Instead of calling the functions directly the kernel uses various Operation Tables, which are a collection of handlers for each operation (these are actually structures of function pointers for each handlers/callbacks). The kernel calls the handler present in the table for the operation. This enables different file systems to register different handlers. This also enables the common tasks to be done before calling the handlers. This reduces the burden on the handlers which can then focus on the operations that are only specific to that file system.

File systems are identified by their names. The supported file systems can be seen using 'cat /proc/filesystems'. The first step is to register the file system with the kernel. Since we are using a kernel module, the file system registration is done during the module initialization. This registers a handler which will be called to fill the super block structure while mounting, a handler to do the cleanup during unmounting the file system. These are other handlers but these two are essential.

The super block operations are set at the time of mounting. The operation tables for inodes and files are set when the inode is opened. The first step before opening an inode is lookup. The inode of a file is looked up by calling the lookup handler. The root-most inode of the new file system has to be allocated at the time of mounting i.e., during the super block initialization.

Once the operation tables are set on the data structures, the kernel calls the handlers depending on the operation.

5. Data structures used in our project:

This is a brief description about the data structures used in implementing our file system.

a. File System Type (struct file_system_type)

This structure is used to register the filesystem with the kernel. This data structure is used by the kernel at the time of mounting a file system. We have to fill the 'name' field with the name of our file system and the handlers to allocate and release the super block objects.

b. Super Block (struct super_block)

This stores the information about the mounted file system. The important fields to be filled are the operation table (s_ops field) and the root dentry (s_root). At the time of mounting a file system, the kernel calls the mount field of the file_system_type object to get a super block object.

c. Super Block Operations (struct super_operations)

Super block operations table.

d. Inode (struct inode)

Inode object is the kernel representation of the low level file. We return the dentry of the root of our file system. We have to attach a proper inode also to the dentry. This structure has two operation tables inode operations and file operations respectively.

e. Inode Operations (struct inode_operations)

This is the inode operations table with each field corresponding to a function pointer to handle the task. It has fields like mkdir, lookup etc.

f. Address Space Operations (struct address_space_operations)

Address space operations table.

g. DEntry (struct dentry)

The kernel uses dentries to represent the file system structure. dentries point to inode objects. This has pointers to store the parent-child relationship of the files. Inodes and files do not store any information about the hierarchy.

h. File (struct file)

File object is used to store the information about the file. The kernel takes care of filling the proper fields but we have to implement the file operation callbacks.

i. File Operations (struct file_operations)

This is the file operations table.

6. Code Snippets

The following table shows the fields we need to fill in the above data structures.

File System Type	DEntry	Super Block	Inode
<ul style="list-style-type: none">• name• mount• kill_sb• fs_flags	<ul style="list-style-type: none">• d_inode	<ul style="list-style-type: none">• s_root	<ul style="list-style-type: none">• i_ino• i_mode• i_op• i_fop

The following table shows the operation tables and the handlers used.

Super Operations	Inode Directory Operations	Inode File Operations	Address Space Operations	File Operations
<ul style="list-style-type: none">• statfs• drop_inode	<ul style="list-style-type: none">• lookup• link• mkdir• rmdir	<ul style="list-style-type: none">• setattr• getattr	<ul style="list-style-type: none">• readpage• write_begin• write_end	<ul style="list-style-type: none">• read_iter• write_iter• splice_read• splice_write• open• llseek

```
static struct file_system_type osfs_type = {
    .owner = THIS_MODULE,
    .name = "fscjs",
    .mount = &osfs_mount,
    .kill_sb = kill_block_super,
    .fs_flags = FS_USERNS_MOUNT,
};
```

Fig 1: File System Type

```
static struct dentry *osfs_mount(struct file_system_type *type, int flags,
                                char const *dev, void *data)
{
    return mount_nodev(type, flags, data, osfs_fill_sb);
}
```

Fig 2: The mount function returning a dentry

```

static int osfs_fill_sb(struct super_block *sb, void *data, int silent)
{
    struct inode * inode;
    struct osfs_fs_info *fsi = kzalloc(sizeof(*fsi), GFP_KERNEL);

    sb->s_fs_info = fsi;
    if (!fsi)
        return -ENOMEM;
    sb->s_maxbytes      = MAX_LFS_FILESIZE;
    sb->s_blocksize      = PAGE_SIZE;
    sb->s_blocksize_bits = PAGE_SHIFT;
    sb->s_magic           = OSFS_MAGIC;
    sb->s_op              = &osfs_super_ops;
    sb->s_time_gran       = 1;

    inode = osfs_get_inode(sb, NULL, S_IFDIR | fsi->mount_opts.mode, 0);
    sb->s_root = d_make_root(inode);
    if (!sb->s_root)
        return -ENOMEM;

    return 0;
}

```

Fig 3: Filling super block fields like s_root etc.

```

struct inode *osfs_get_inode(struct super_block *sb,
                           const struct inode *dir, umode_t mode, dev_t dev)
{
    struct inode * inode = new_inode(sb);
    if (inode) {
        inode->i_ino = get_next_ino();
        inode_init_owner(inode, dir, mode);
        inode->i_atime = inode->i_mtime = inode->i_ctime = current_time(inode);
        switch (mode & S_IFMT) {
            default:
                init_special_inode(inode, mode, dev);
                break;
            case S_IFREG:
                inode->i_op = &osfs_file_inode_operations;
                inode->i_fop = &osfs_file_operations;
                break;
            case S_IFDIR:
                inode->i_op = &osfs_dir_inode_operations;
                inode->i_fop = &simple_dir_operations;
                inc_nlink(inode);
                break;
            case S_IFLNK:
                inode->i_op = &page_symlink_inode_operations;
                inode_nohighmem(inode);
                break;
        }
    }
    return inode;
}

```

Fig 4: Filling inode fields like i_op, i_fop etc.

```
static const struct super_operations osfs_super_ops= {
    .statfs      = simple_statfs,
    .drop_inode  = generic_delete_inode,
};
```

Fig 5: Superblock Operations.

```
static const struct inode_operations osfs_dir_inode_operations = {
    .lookup      = simple_lookup,
    .link        = simple_link,
    .mkdir       = osfs_mkdir,
    .rmdir       = simple_rmdir,
};
```

Fig 6: Inode Directory Operations.

```
static const struct inode_operations osfs_file_inode_operations = {
    .setattr     = simple_setattr,
    .getattr     = simple_getattr,
};
```

Fig 7: Inode File Operations.

```
static const struct address_space_operations osfs_aops = {
    .readpage    = simple_readpage,
    .write_begin = simple_write_begin,
    .write_end   = simple_write_end,
};
```

Fig 8: Address Space Operations.

```
static const struct file_operations osfs_file_operations = {
    .open         = generic_file_open,
    .read_iter    = generic_file_read_iter,
    .write_iter   = generic_file_write_iter,
    .splice_read  = generic_file_splice_read,
    .splice_write = iter_file_splice_write,
    .llseek       = generic_file_llseek,
};
```

Fig 9: File Operations.

7. Mounting/Unmounting:

Step 1: Creating a Makefile

```
obj-m += fscjs.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Step 2: This generates a file fscjs.o. Loading the file system is same as loading a module.
insmod fscjs.o

Step 3: Mount the file system.

```
pi@raspberrypi:~/Desktop/filesystem $ touch image
pi@raspberrypi:~/Desktop/filesystem $ mkdir FSCJS
pi@raspberrypi:~/Desktop/filesystem $ ls
FSCJS  fscjs.ko  fscjs.mod.o  image  modules.order
fscjs.c  fscjs.mod.c  fscjs.o  Makefile  Module.symvers
pi@raspberrypi:~/Desktop/filesystem $ sudo mount -o loop -t fscjs ./image ./FSCJS
```

Step 4: Check if the file system is mounted

```
pi@raspberrypi:~/Desktop/filesystem $ cat /proc/filesystems
nodev  sysfs
nodev  rootfs
nodev  ramfs
nodev  bdev
nodev  proc
nodev  cpuset
nodev  cgroup
nodev  cgroup2
nodev  tmpfs
nodev  devtmpfs
nodev  configfs
nodev  debugfs
nodev  tracefs
nodev  sockfs
nodev  pipefs
nodev  rpc_pipefs
nodev  devpts
    ext3
    ext2
    ext4
    vfat
    msdos
nodev  nfs
nodev  nfs4
nodev  autofs
    f2fs
nodev  mqueue
    fuseblk
nodev  fuse
nodev  fusectl
    fscjs
pi@raspberrypi:~/Desktop/filesystem $ cat /proc/filesystems|grep fscjs
fscjs
pi@raspberrypi:~/Desktop/filesystem $
```

Step 5: Unmount

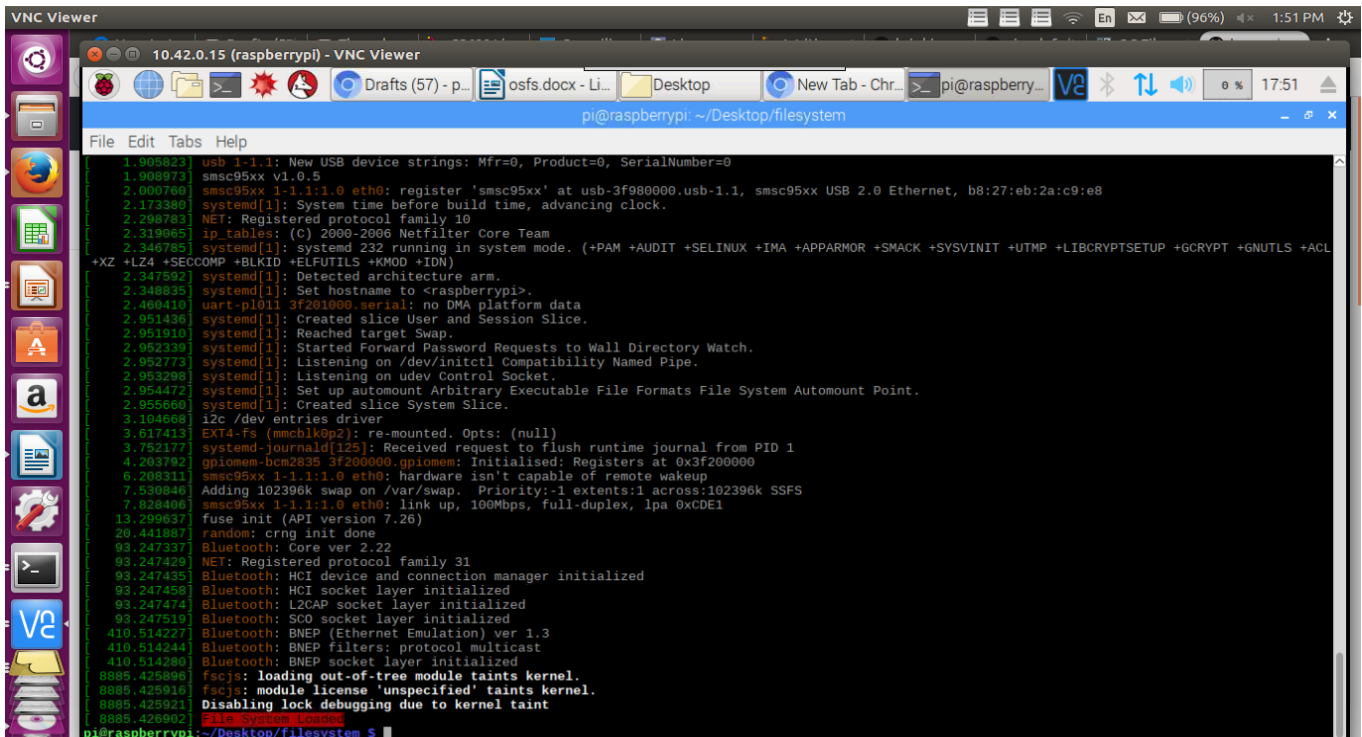
umount /mnt/rkfs

Step 6: Unload the module

rmmod rkfs

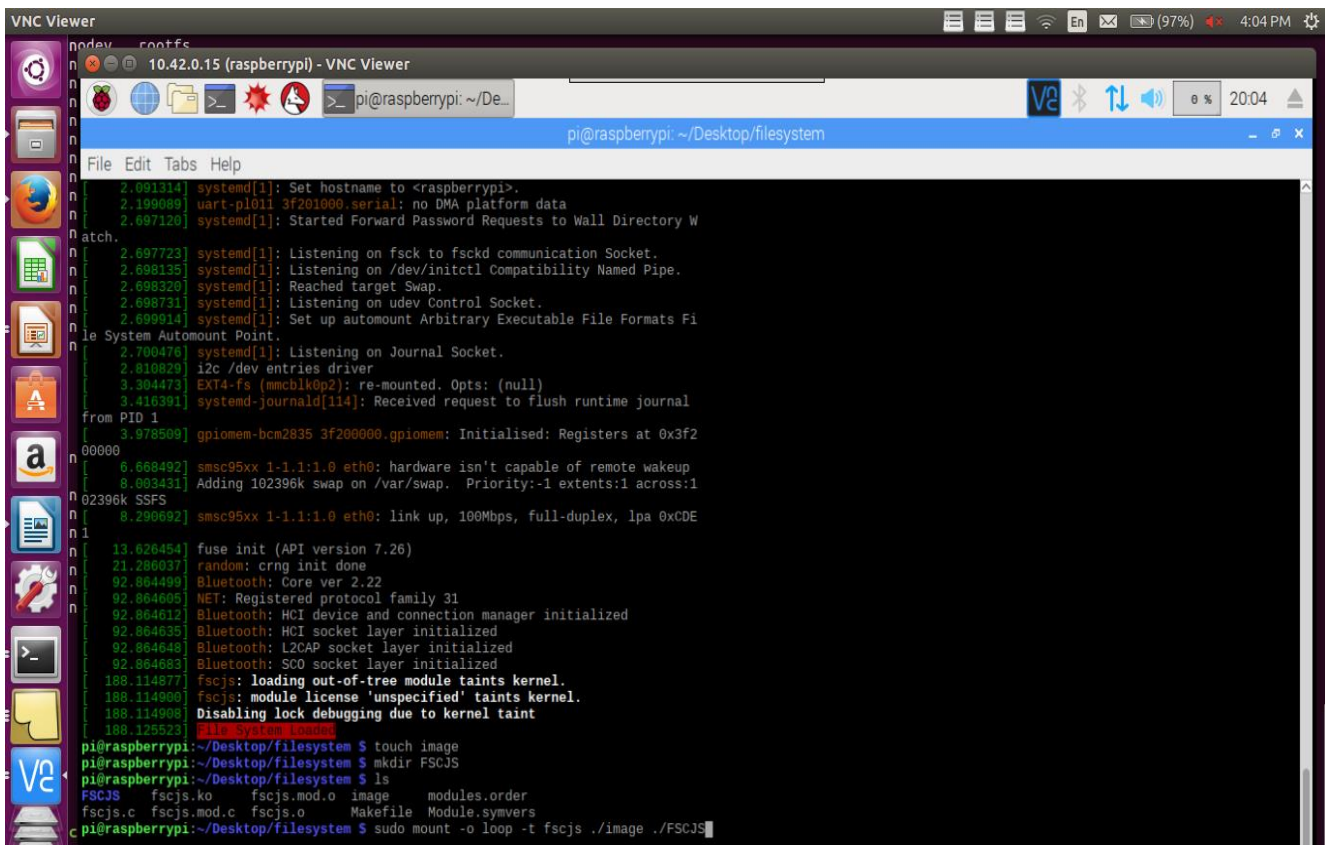
8. Working Screenshots

a. Loading filesystem



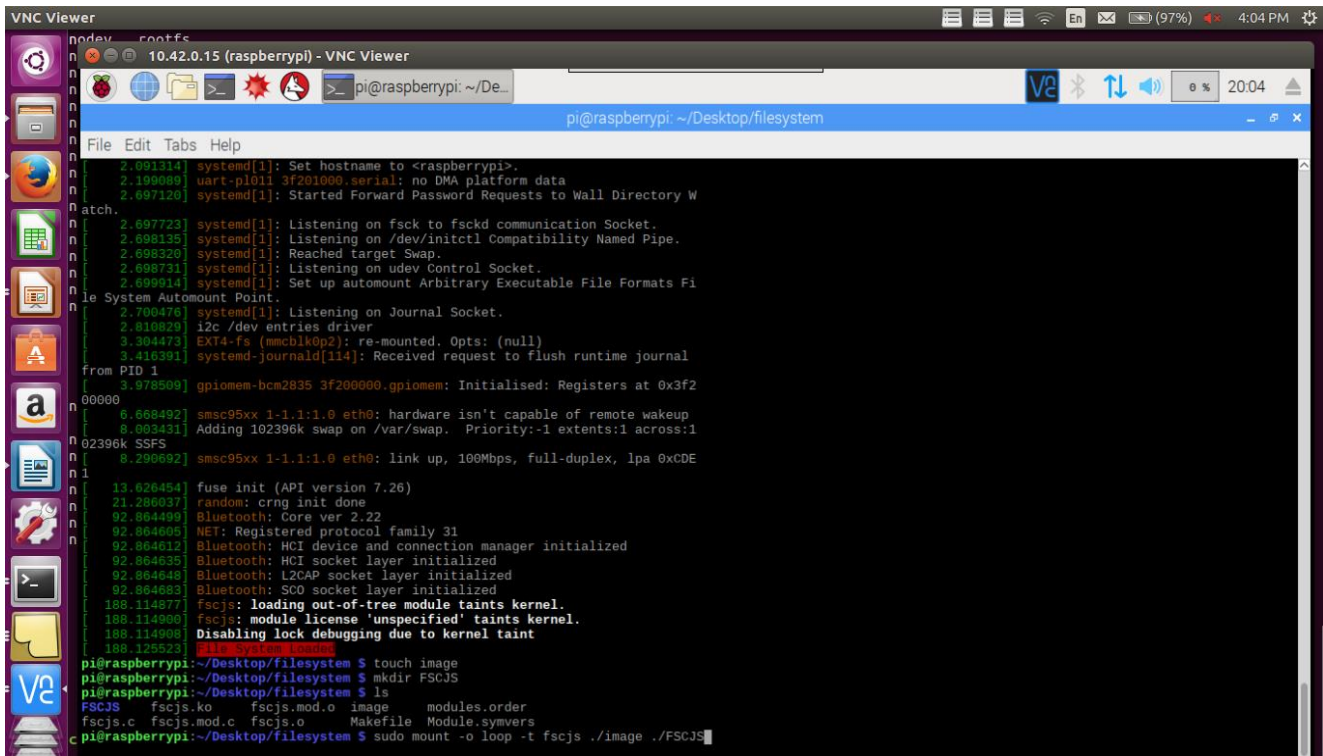
```
1.905823 usb 1-1.1: New USB device strings: Mfr=0, Product=0, SerialNumber=0
1.908973 smsc95xx v1.0.5
2.000760 smsc95xx 1-1.1:1.0 eth0: register 'smc95xx' at usb-3f980000.usb-1.1, smc95xx USB 2.0 Ethernet, b8:27:eb:2a:c9:e8
2.173380 systemd[1]: System time before build time, advancing clock.
2.298783 NET: Registered protocol family 10
2.319065 ip_tables: (c) 2000-2006 Netfilter Core Team
2.346785 systemd[1]: systemd 232 running in system mode. (+PAM +AUDIT +SELINUX +IMA +APPARMOR +SMACK +SYSVINIT +UTMP +LIBCRYPTSETUP +GCRYPT +GNUTLS +ACL
+XZ +LZ4 +SECCOMP +BLKID +ELFUTILS +KMOD +IDN)
2.347592 systemd[1]: Detected architecture arm.
2.348835 systemd[1]: Set hostname to <raspberrypi>.
2.460410 uart-pl011 3f201000.serial: no DMA platform data
2.951436 systemd[1]: Created slice User and Session Slice.
2.951910 systemd[1]: Reached target Swap.
2.952339 systemd[1]: Started Forward Password Requests to Wall Directory Watch.
2.952773 systemd[1]: Listening on /dev/initctl Compatibility Named Pipe.
2.953298 systemd[1]: Listening on udev Control Socket.
2.954472 systemd[1]: Set up automount Arbitrary Executable File Formats File System Automount Point.
2.955600 systemd[1]: Created slice System Slice.
3.104668 i2c /dev entries driver
3.617413 EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
3.752177 systemd-journald[125]: Received request to flush runtime journal from PID 1
4.203792 gpionem-bcm2835 3f200000.gpionem: Initialised: Registers at 0x3f200000
6.208311 smsc95xx 1-1.1:1.0 eth0: hardware isn't capable of remote wakeup
7.530846 Adding 102396k swap on /var/swap. Priority:-1 extents:1 across:102396k SSFS
7.828406 smsc95xx 1-1.1:1.0 eth0: link up, 100Mbps, full-duplex, lpa 0xCDE1
13.290637 fuse init (API version 7.26)
20.441887 random: crng init done
93.247387 Bluetooth: Core ver 2.22
93.247429 NET: Registered protocol family 31
93.247435 Bluetooth: HCI device and connection manager initialized
93.247458 Bluetooth: HCI socket layer initialized
93.247474 Bluetooth: L2CAP socket layer initialized
93.247519 Bluetooth: SCO socket layer initialized
410.514227 Bluetooth: BNEP (Ethernet Emulation) ver 1.3
410.514244 Bluetooth: BNEP filters: protocol multicast
410.514280 Bluetooth: BNEP socket layer initialized
8885.425896 fscjs: loading out-of-tree module taints kernel.
8885.425916 fscjs: module license 'unspecified' taints kernel.
8885.425921 Disabling lock debugging due to kernel taint
8885.426902 [taint system init]
pi@raspberrypi: ~/Desktop/filesystem $
```

b. Inserting module.



```
2.091314 systemd[1]: Set hostname to <raspberrypi>.
2.199089 uart-pl011 3f201000.serial: no DMA platform data
2.697120 systemd[1]: Started Forward Password Requests to Wall Directory W
natch.
2.697723 systemd[1]: Listening on fscjs to fscjd communication Socket.
2.698135 systemd[1]: Listening on /dev/initctl Compatibility Named Pipe.
2.698320 systemd[1]: Reached target Swap.
2.698731 systemd[1]: Listening on udev Control Socket.
2.69914 systemd[1]: Set up automount Arbitrary Executable File Formats Fi
nle System Automount Point.
2.700476 systemd[1]: Listening on Journal Socket.
2.810820 i2c /dev entries driver
3.304473 EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
3.416391 systemd-journald[114]: Received request to flush runtime journal
from PID 1
3.978509 gpionem-bcm2835 3f200000.gpionem: Initialised: Registers at 0x3f2
00000
6.668492 smsc95xx 1-1.1:1.0 eth0: hardware isn't capable of remote wakeup
6.803431 Adding 102396k swap on /var/swap. Priority:-1 extents:1 across:1
02396k SSFS
8.290692 smsc95xx 1-1.1:1.0 eth0: link up, 100Mbps, full-duplex, lpa 0xCDE
n1
13.626454 fuse init (API version 7.26)
21.206037 random: crng init done
92.804498 Bluetooth: Core ver 2.22
92.804605 NET: Registered protocol family 31
92.804612 Bluetooth: HCI device and connection manager initialized
92.804635 Bluetooth: HCI socket layer initialized
92.804648 Bluetooth: L2CAP socket layer initialized
92.804683 Bluetooth: SCO socket layer initialized
188.114877 fscjs: loading out-of-tree module taints kernel.
188.114900 fscjs: module license 'unspecified' taints kernel.
188.114908 Disabling lock debugging due to kernel taint
188.125523 [taint system init]
pi@raspberrypi: ~/Desktop/filesystem $ touch image
pi@raspberrypi: ~/Desktop/filesystem $ mkdir FSCJS
pi@raspberrypi: ~/Desktop/filesystem $ ls
FSCJS  fscjs.ko  fscjs.mod.o  image  modules.order
fscjs.c  fscjs.mod.c  fscjs.o  Makefile  Module.symvers
pi@raspberrypi: ~/Desktop/filesystem $ sudo mount -o loop -t fscjs ./image ./FSCJS
```

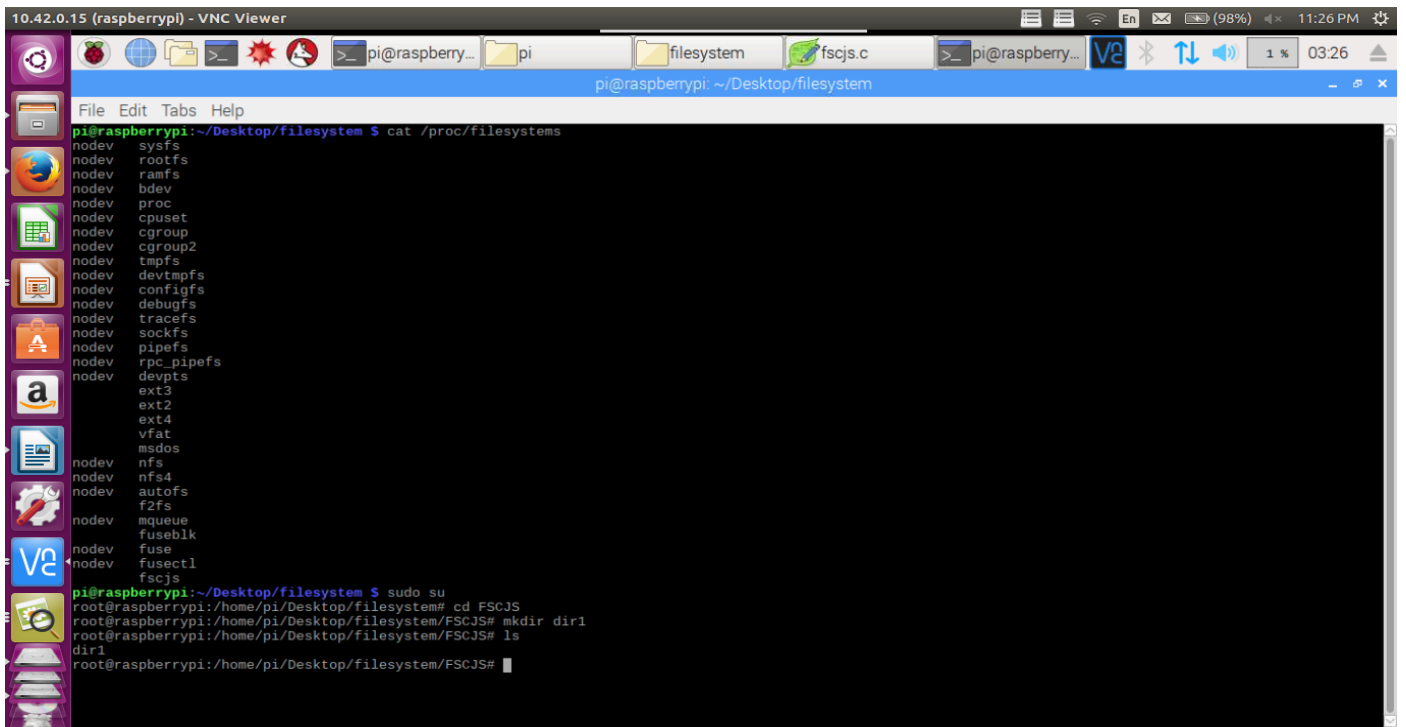

c. Mounting on kernel



```
nodev sysfs
nodev rootfs
nodev ramfs
nodev bdev
nodev proc
nodev cpuset
nodev cgroup
nodev cgroup2
nodev tmpfs
nodev devtmpfs
nodev configfs
nodev debugfs
nodev tracefs
nodev sockfs
nodev pipefs
nodev rpc_pipefs
nodev devpts
nodev ext3
nodev ext2
nodev ext4
nodev vfat
nodev msdos
nodev nfs
nodev nfs4
nodev autofs
nodev f2fs
nodev mqueue
nodev fuseblk
nodev fuse
nodev fusectl
nodev fscjs

pi@raspberrypi: ~/Desktop/filesystem
pi@raspberrypi: ~/Desktop/filesystem
2.091314 systemd[1]: Set hostname to <raspberrypi>.
2.199089 uart-pl011: 3f201000,serial: no DMA platform data
2.697120 systemd[1]: Started Forward Password Requests to Wall Directory W
2.697723 systemd[1]: Listening on fsck to fsckd communication Socket.
2.698135 systemd[1]: Listening on /dev/initctl Compatibility Named Pipe.
2.698320 systemd[1]: Reached target Swap.
2.698731 systemd[1]: Listening on udev Control Socket.
2.699914 systemd[1]: Set up automount Arbitrary Executable File Formats Fi
le System Automount Point.
2.700476 systemd[1]: Listening on Journal Socket.
2.810820 i2c /dev entries driver
3.304473 EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
3.416391 systemd-journald[114]: Received request to flush runtime journal
from PID 1
3.978509 gpionem-bcm2835 3f200000.gpionem: Initialised: Registers at 0x3f2
00000
6.668492 smsc95xx 1-1.1:1.0 eth0: hardware isn't capable of remote wakeup
8.003431 Adding 102396k swap on /var/swap. Priority:-1 extents:1 across:1
02396k SSFS
8.296692 smsc95xx 1-1.1:1.0 eth0: link up, 100Mbps, full-duplex, lpa 0xCDE
n1
13.026454 fuse init (API version 7.26)
21.286037 random: crng init done
92.864498 Bluetooth: Core ver 2.22
92.864605 NET: Registered protocol family 31
92.864612 Bluetooth: HCI device and connection manager initialized
92.864635 Bluetooth: HCI socket layer initialized
92.864648 Bluetooth: L2CAP socket layer initialized
92.864683 Bluetooth: SCO socket layer initialized
188.114877 fscjs: loading out-of-tree module taints kernel.
188.114900 fscjs: module license 'unspecified' taints kernel.
188.114908 Disabling lock debugging due to kernel taint
188.125523 ***system image***
pi@raspberrypi:~/Desktop/filesystem $ touch image
pi@raspberrypi:~/Desktop/filesystem $ mkdir FSCJS
pi@raspberrypi:~/Desktop/filesystem $ ls
FSCJS  fscjs.ko  fscjs.mod.o  image  modules.order
fscjs.c  fscjs.mod.c  fscjs.o  Makefile  Module.symvers
pi@raspberrypi:~/Desktop/filesystem $ sudo mount -o loop -t fscjs ./image ./FSCJS
```

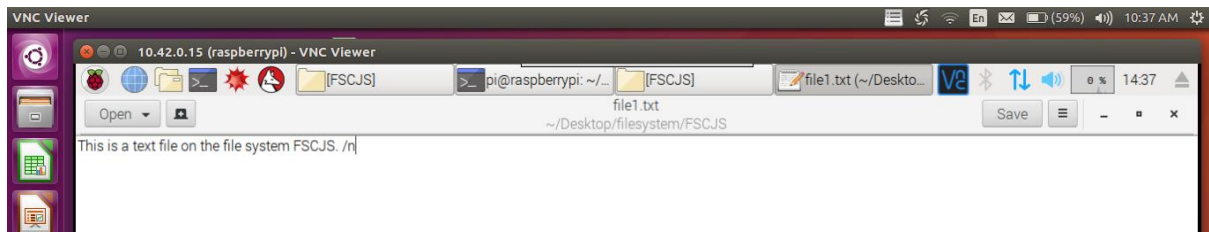
d. Creating a directory



```
10.42.0.15 (raspberrypi) - VNC Viewer
pi@raspberrypi: ~/Desktop/filesystem
pi@raspberrypi: ~/Desktop/filesystem
File Edit Tabs Help
pi@raspberrypi:~/Desktop/filesystem $ cat /proc/filesystems
nodev sysfs
nodev rootfs
nodev ramfs
nodev bdev
nodev proc
nodev cpuset
nodev cgroup
nodev cgroup2
nodev tmpfs
nodev devtmpfs
nodev configfs
nodev debugfs
nodev tracefs
nodev sockfs
nodev pipefs
nodev rpc_pipefs
nodev devpts
nodev ext3
nodev ext2
nodev ext4
nodev vfat
nodev msdos
nodev nfs
nodev nfs4
nodev autofs
nodev f2fs
nodev mqueue
nodev fuseblk
nodev fuse
nodev fusectl
nodev fscjs

pi@raspberrypi:~/Desktop/filesystem $ sudo su
root@raspberrypi:/home/pi/Desktop/filesystem# cd FSCJS
root@raspberrypi:/home/pi/Desktop/filesystem/FSCJS# mkdir dir1
root@raspberrypi:/home/pi/Desktop/filesystem/FSCJS# ls
dir1
root@raspberrypi:/home/pi/Desktop/filesystem/FSCJS#
```

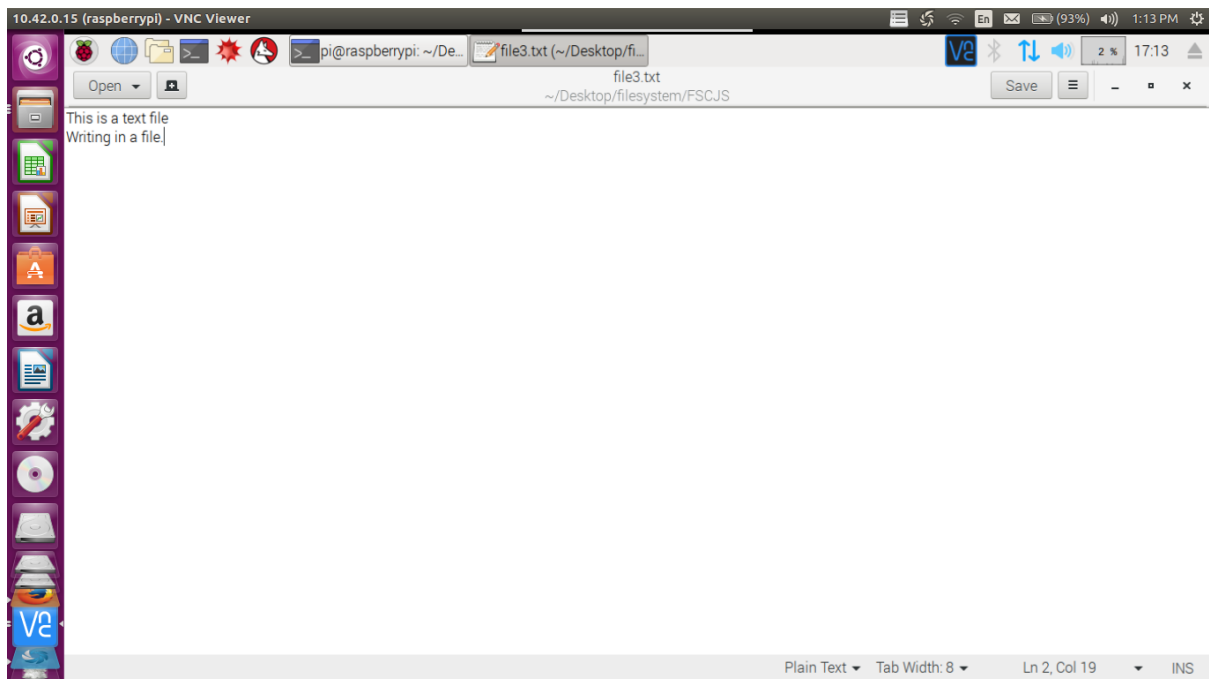
e. Creating and Reading file:



```
pi@raspberrypi:~/Desktop/filesystem/FSCJS $ ls
file1.txt
pi@raspberrypi:~/Desktop/filesystem/FSCJS $ head file1.txt
This is a text file on the file system FSCJS.
pi@raspberrypi:~/Desktop/filesystem/FSCJS $
```

f. Writing on file:

```
pi@raspberrypi:~/Desktop/filesystem/FSCJS $ echo "This is a text file" > file3.txt
pi@raspberrypi:~/Desktop/filesystem/FSCJS $ cat file3.txt
This is a text file
pi@raspberrypi:~/Desktop/filesystem/FSCJS $ gedit file3.txt
```



```
pi@raspberrypi:~/Desktop/filesystem/FSCJS $ cat file3.txt
This is a text file
Writing in a file.
pi@raspberrypi:~/Desktop/filesystem/FSCJS $
```

9. Summary:

We familiarized ourselves with creation of loadable kernel modules and main structures of the file system. We also wrote a real file system, which makes use of Linux's page caching mechanisms and mounted it on Raspberry Pi.

10. References

<https://www.tldp.org/LDP/lkmpg/2.6/html/lkmpg.html>

<https://kukuruku.co/post/writing-a-file-system-in-linux-kernel/>

http://www.cse.wustl.edu/~cdgill/courses/cse422/studios/01_RPi3_setup.html

http://www.cse.wustl.edu/~cdgill/courses/cse422/studios/02_welcome_linux.html

http://www.cse.wustl.edu/~cdgill/courses/cse422/studios/21_vfs.html