## Initial Questions

1. **What is a kernel module, and how does it different from a regular library?**

   Kernel modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system. A module can be configured as built-in or loadable. To dynamically load or remove a module, it has to be configured as a loadable module in the kernel configuration.

2. **How to compile a kernel module?**

   Kernel modules need to be compiled a bit differently from regular userspace apps. Former kernel versions required us to care much about these settings, which are usually stored in Makefiles. Although hierarchically organized, many redundant settings accumulated in sublevel Makefiles and made them large and rather difficult to maintain. Fortunately, there is a new way of doing these things, called **kbuild**, and the build process for external loadable modules is now fully integrated into the standard kernel build mechanism.

   A sample makefile is shown as follows:

```
obj-m += hello-1.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
```

3. **Check `dadfs` for source code**

   The code is compiled and tested by running `./test.sh`.

```
loop13
     jbd           f0e6feec-8d98-402d-8b71-ac6fefec2be4
loop14
     jbd           e02a2a20-1cf7-4faf-b279-b56e926b7e76
loop15
     jbd           6454d3fc-d2f4-4873-b6f1-f28c31db7a46
loop16

sda
├─sda1
│    vfat         AECD-9B92                               511M     0% /boot/efi
├─sda2
│
└─sda5
     ext4         3cba45e7-70e0-4c1d-ba69-224b6838c5e5   15.6G    44% /
sr0
#make -C /lib/modules/5.11.0-38-generic/build M=/home/pastrol/Desktop/l7/dadfs modules
make -C /lib/modules/5.11.0-38-generic/build M=/home/pastrol/Desktop/l7/dadfs modules
make[1]: Entering directory '/usr/src/linux-headers-5.11.0-38-generic'
make[1]: Leaving directory '/usr/src/linux-headers-5.11.0-38-generic'
100+0 records in
100+0 records out
409600 bytes (410 kB, 400 KiB) copied, 0.00331365 s, 124 MB/s
10+0 records in
10+0 records out
10485760 bytes (10 MB, 10 MiB) copied, 0.0186852 s, 561 MB/s
mke2fs 1.45.5 (07-Jan-2020)
Discarding device blocks: done
Creating filesystem with 2560 4k blocks and 0 inodes
Filesystem UUID: c43b6e33-a6f6-4601-923a-1283ce9c2ef8
Superblock backups stored on blocks:

Zeroing journal device:
Super block written succesfully
root directory inode written succesfully
journal inode written succesfully
welcomefile inode written succesfully
inode store padding bytes (after the three inodes) written sucessfully
Journal written successfully
root directory datablocks (name+inode_no pair for welcomefile) written succesfully
padding after the rootdirectory children written succesfully
block has been written succesfully
```

1. **How are mutex defined and used?  How good is this approach?**

   Two mutex in total are defined.

   - `dadfs_super_block` : lock super block when action is taken on block.
   - `dadfs_inode` : lock super block when block inode is to be changed.

   Advantages:

   - Two different subtypes of mutex is used for better efficiency instead of just one type of mutex for every possible situation

2. **How is information shared between the kernel and user spaces?**

3. **Check `README.md` file for changes**.