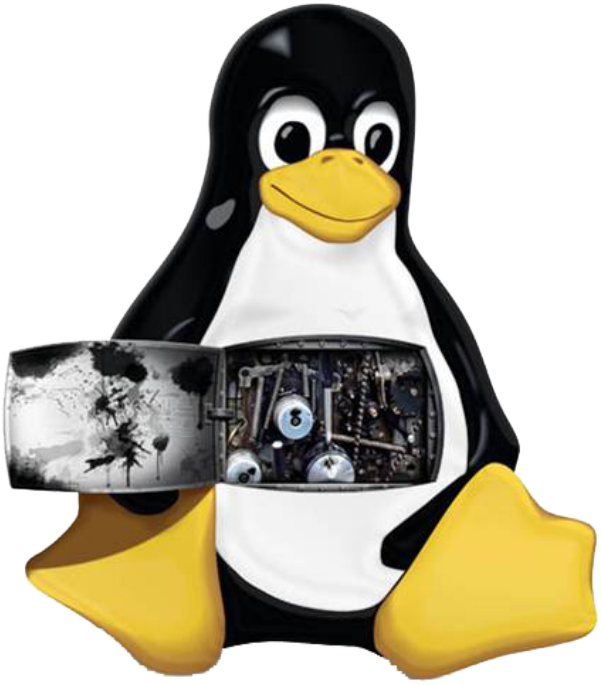
Linux Kernel Development (LKD)

Session 1



Loadable Kernel Modules (LKM): Laboratory

Paulo Baltarejo Sousa
pbs@isep.ipp.pt
2017

CISTER - Research Centre in Real-Time & Embedded Computing Systems
isep Instituto Superior de Engenharia do Porto
INESCTEC

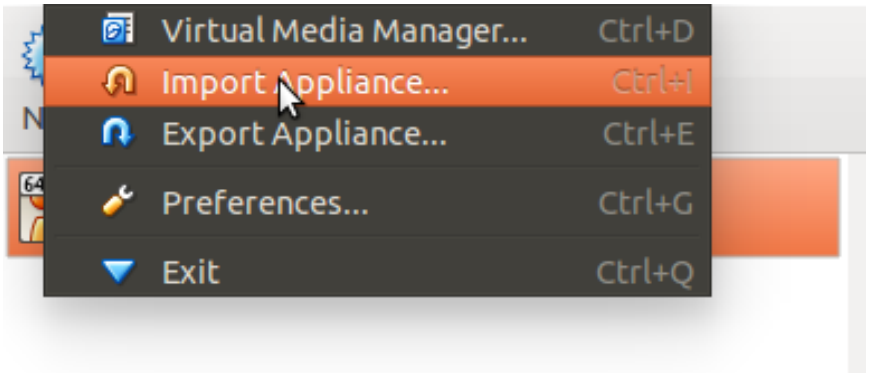
1 Introduction

In the Linux Kernel Development (LKD) module, we will use a virtual machine called `CISTER`. This virtual machine was created using the VirtualBox software running the operating system Linux with distribution Ubuntu 16.04 Linux (Xenial Xerus) and with the kernel version `4.10.0-28-generic`. This system is configured with only one user called `cister` and the password is also `cister`.

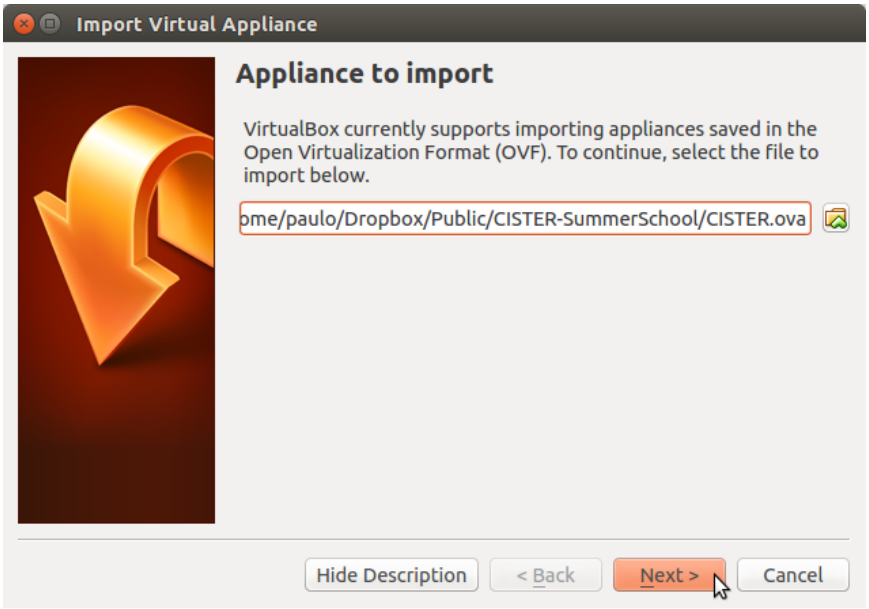
2 Get Virtual Machine

Download the virtual machine from `http://www.cister.isep.ipp.pt/summer2017/w1/CISTER.ova`

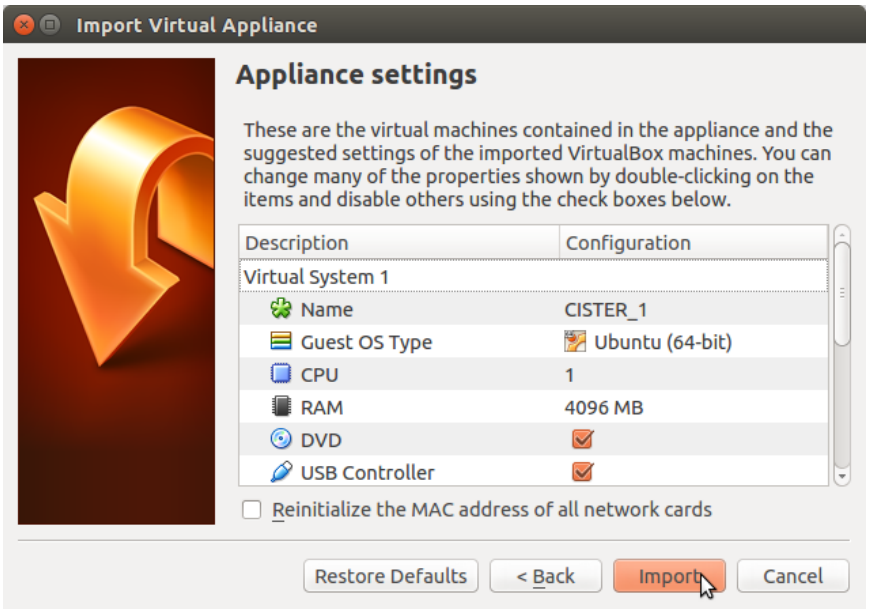
1. From VirtualBox main menu select `File > Import Appliance`.



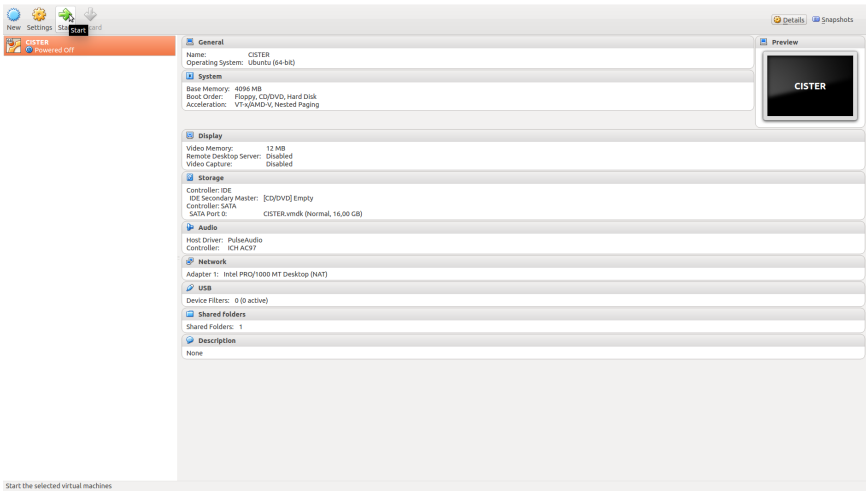
2. Select the `CISTER.ova` file and click on `Next`.



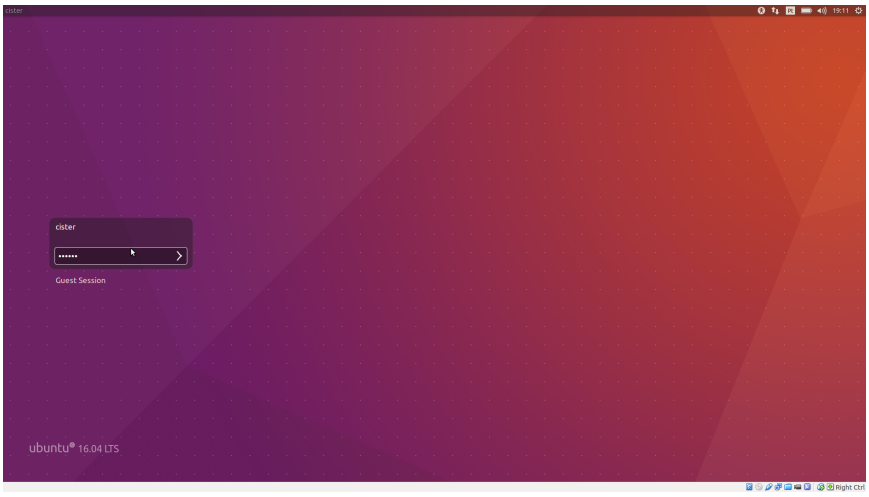
3. Check the Appliance Settings and click on `Import`.



4. Start the CISTER virtual Machine.

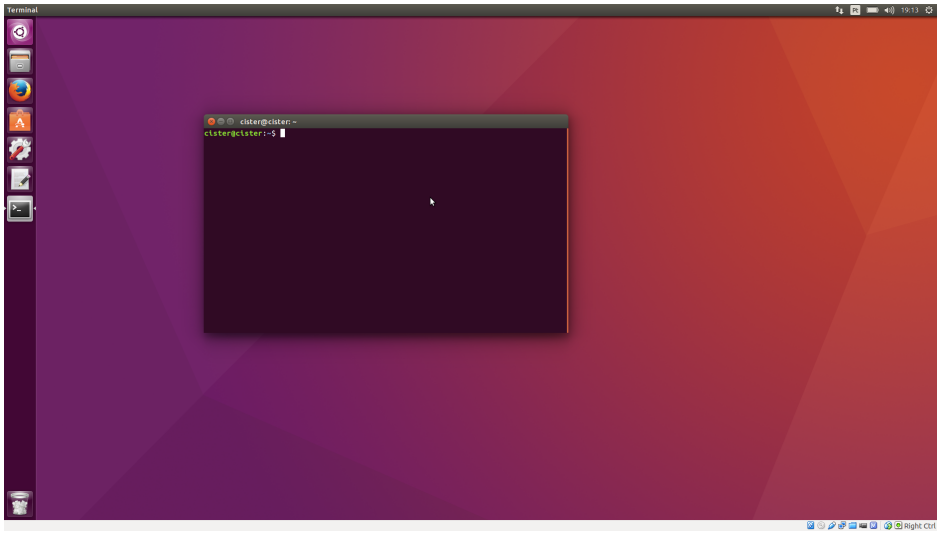


5. Type the password: `cister`



3 Tools and packages

In order to install the required tools and packages, open a terminal



and type the following linux commands:

```
> sudo apt-get update
> sudo apt-get install build-essential
```

4 Learning examples

Download the learning examples from `http://www.cister.isep.ipp.pt/summer2017/w1/S01-LKM.tar.gz`. In this compressed file you can find several source code modules and a `Makefile`. Extract it and enjoy!.
Feel free to inquiry the Lecturer.

4.1 `hello.c`

This is the simplest Linux kernel module.

1. For compiling it, type:
   ```
   > make
   ```
2. Insert it into the Linux kernel:
   ```
   > sudo insmod ./hello.ko
   ```
3. Check `printk` messages:
   ```
   > dmesg
   ```
4. Check list modules:
   ```
   > lsmod
   ```

5. Remove it from the Linux kernel:
   ```
   > sudo rmmod hello
   ```

6. Check `printk` messages:
   ```
   > dmesg
   ```

4.2 `hello2.c`

This is a basic module that create a new entry in the `proc` directory. Further, it uses `file_operations` structure.

1. Change `Makefile`:
   ```
   obj-m:=hello2.o
   ```

2. Compilation:
   ```
   > make
   ```

3. Insert it into the Linux kernel:
   ```
   > sudo insmod ./hello2.ko
   ```

4. Change `/proc/hello2` permissions:
   ```
   > sudo chmod 666 /proc/hello2
   ```

5. Perform read operation:
   ```
   > cat /proc/hello2
   > dmesg
   ```

6. Perform write operation:
   ```
   > echo "aa" > /proc/hello2
   > dmesg
   ```

7. Remove it from the Linux kernel:
   ```
   > sudo rmmod hello2
   ```

8. Check `printk` messages:
   ```
   > dmesg
   ```

4.3 `hello3.c`

This module allows to read and write messages from/to kernel.

1. Change `Makefile`:
   ```
   obj-m:=hello3.o
   ```

2. Compilation:
   `> make`

3. Insert it into the Linux kernel:
   `> sudo insmod ./hello3.ko`

4. Change `/proc/hello3` permissions:
   `> sudo chmod 666 /proc/hello3`

5. Perform read operation:
   `> cat /proc/hello3`

6. Perform write operation:
   `> echo "LKM is cool" > /proc/hello3`

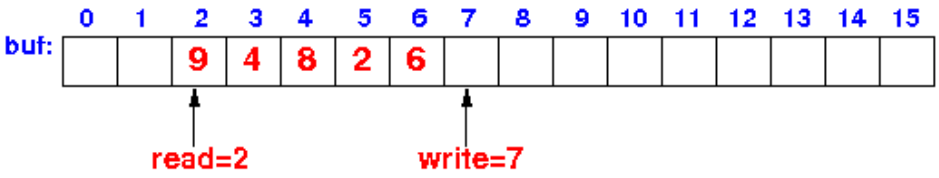
7. Perform read operation:
   `> cat /proc/hello3`

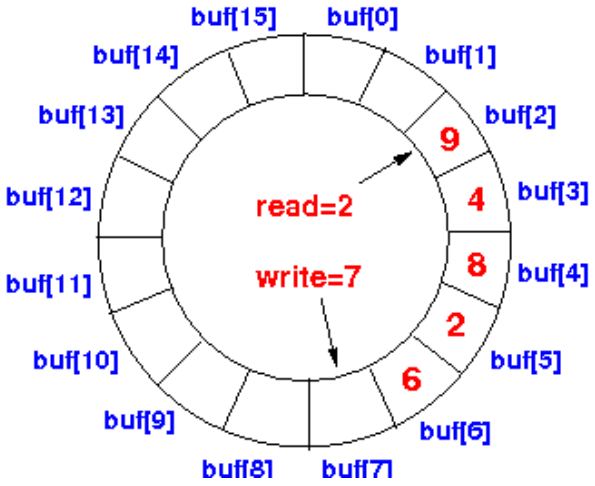
8. Remove it from the Linux kernel:
   `> sudo rmmod hello3`

9. Check `printk` messages:
   `> dmesg`

4.4 `ring_buffer.c`

This module implements a First In First Out (FIFO) ring buffer. A ring buffer is a container that uses a single, fixed-size buffer as if it was connected end-to-end.





1. Change `Makefile`:
   `obj-m:=ring_buffer.o`

2. Compilation:
   `> make`

3. Insert it into the Linux kernel:
   `> sudo insmod ./ring_buffer.ko`

4. Change `/proc/ring_buffer` permissions:
   `> sudo chmod 666 /proc/ring_buffer`

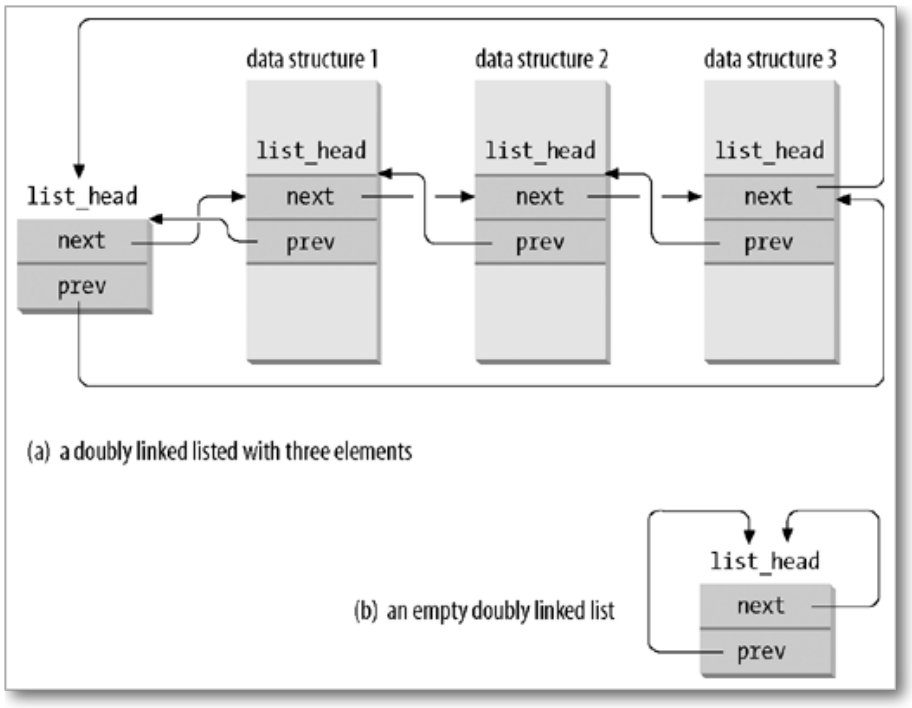
5. Test it by performing a set of write and read operations:
   ```
   > echo "1" > /proc/ring_buffer
   > echo "2" > /proc/ring_buffer
   > echo "3" > /proc/ring_buffer
   > echo "4" > /proc/ring_buffer
   > ...
   > cat /proc/ring_buffer
   > cat /proc/ring_buffer
   > cat /proc/ring_buffer
   ```

```
> cat /proc/ring_buffer
```

6. Remove it from the Linux kernel:
   ```
   > sudo rmmod ring_buffer
   ```
7. Check `printk` messages:
   ```
   > dmesg
   ```

4.5 `fifo_queue.c`

This module implements a FIFO queue through a linked list.



1. Change `Makefile`:
   ```
   obj-m:=fifo_queue.o
   ```
2. Compilation:
   ```
   > make
   ```
3. Insert it into the Linux kernel:
   ```
   > sudo insmod ./fifo_queue.ko
   ```

4. Change `/proc/fifo_queue` permissions:
   ```
   > sudo chmod 666 /proc/fifo_queue
   ```

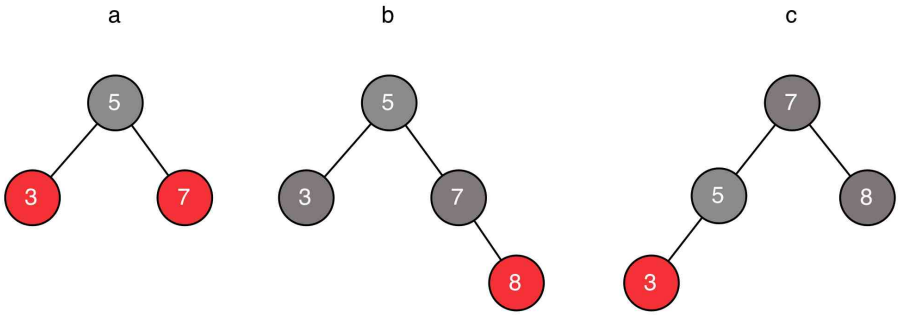
5. Test it by performing a set of write and read operations:
   ```
   > echo "1" > /proc/fifo_queue
   > echo "2" > /proc/fifo_queue
   > echo "3" > /proc/fifo_queue
   > echo "4" > /proc/fifo_queue
   > ...
   > cat /proc/fifo_queue
   > cat /proc/fifo_queue
   > cat /proc/fifo_queue
   > cat /proc/fifo_queue
   ```

6. Remove it from the Linux kernel:
   ```
   > sudo rmmod fifo_queue
   ```

7. Check `printk` messages:
   ```
   > dmesg
   ```

4.6 `sorted_queue.c`

This module implements a sorted queue through a Red-Black Tree.



1. Change `Makefile`:
   ```
   obj-m:=sorted_queue.o
   ```

2. Compilation:
   ```
   > make
   ```

3. Insert it into the Linux kernel:
   ```
   > sudo insmod ./sorted_queue.ko
   ```

4. Change `/proc/sorted_queue` permissions:
   ```
   > sudo chmod 666 /proc/sorted_queue
   ```

5. Test it by performing a set of write and read operations:
   ```
   > echo "1" > /proc/sorted_queue
   > echo "4" > /proc/sorted_queue
   > echo "3" > /proc/sorted_queue
   > echo "2" > /proc/sorted_queue
   > ...
   > cat /proc/sorted_queue
   > cat /proc/sorted_queue
   > cat /proc/sorted_queue
   > cat /proc/sorted_queue
   ```

6. Remove it from the Linux kernel:
   ```
   > sudo rmmod sorted_queue
   ```

7. Check `printk` messages:
   ```
   > dmesg
   ```

5 Assignment

1. Write the pseudo-code of all functions of the `ring_buffer.c`, `fifo_queue.c` and `sorted_queue.c`;

2. Update the `ring_buffer.c`, `fifo_queue.c` and `sorted_queue.c` in order to support concurrency.