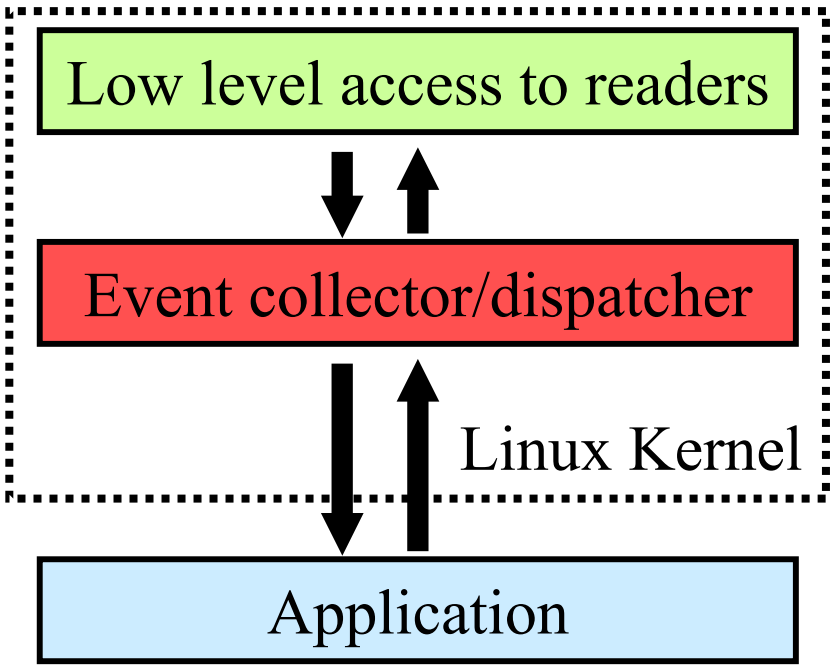
What's on the menu…

Software Comprehension and Maintenance
April 2005

RF-IDs in the Kernel -- Episode II: Revenge of the Modules

Achilleas Anagnostopoulos
(archie@istlab.dmst.aueb.gr)
Department of Management Science and Technology
Athens University Of Economics and Business

Flashback: What we are trying to accomplish

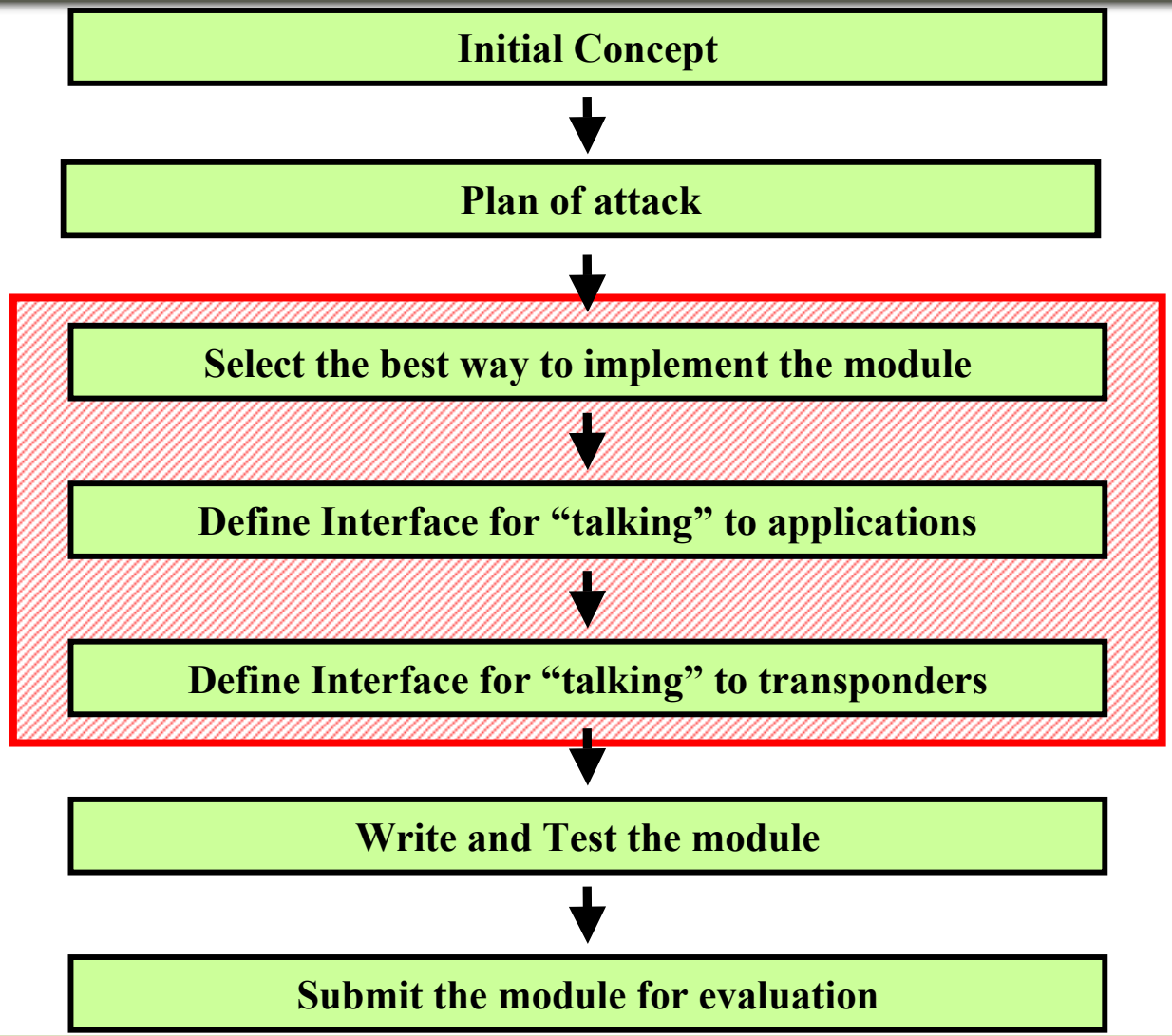


We need a unified way for reading and writing RF-ID tags from our own applications.

The proposed solution involves a specialized kernel module for linux to service this need.

The Roadmap or "Where are we now?"

We are here →



When merging code, size DOES matter!

A more accurate measurement of the C code in the kernel reveals the magic number 3.805.028

Interesting Questions:

- How do we configure and compile this spaghetti code?
- What happens when our module gets accepted for the next kernel version?
- How does a user app talk to our module?
- How does our module "talk" to the kernel?

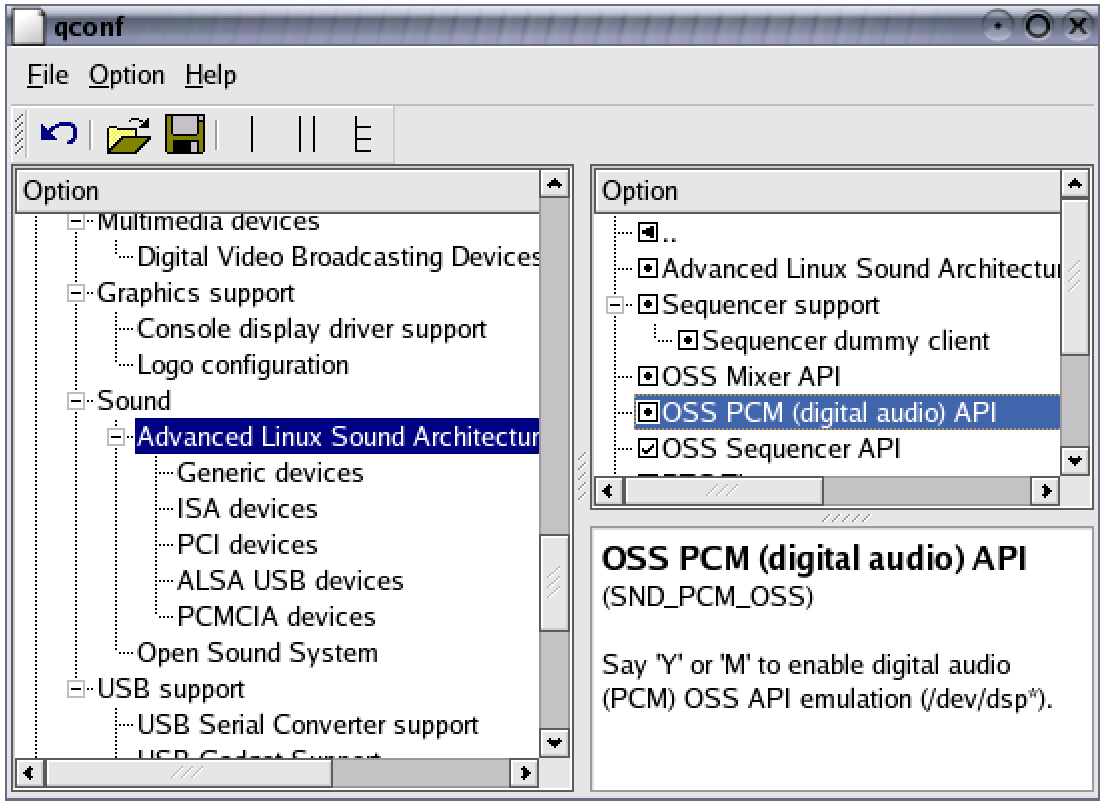
```
#!/bin/bash

find -name "*.c" | xargs sed
'
s/\/\/.*$//
s/\/\*.*\*\///g
/\/\*/{
    N
    s/\/\*.*\*\///g
}
/^[    ]*$/d
' | wc -l
```

DIY - Configuring and building the kernel

Forget about the console! Say Hi to graphical configuration!

Navigate to the folder where the source resides (usually /usr/src/linux) and type: **make xconfig**



Drivers may be compiled **directly into the kernel** OR as separate **modules** which are loaded on-demand.

Once you configured everything :

make modules
make bzImage

Then grab a coffee and wait :)

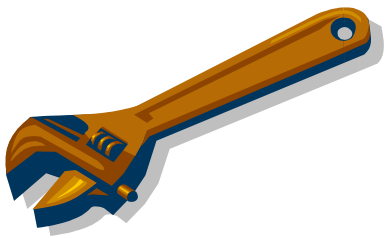
Our orphan module just found a home!

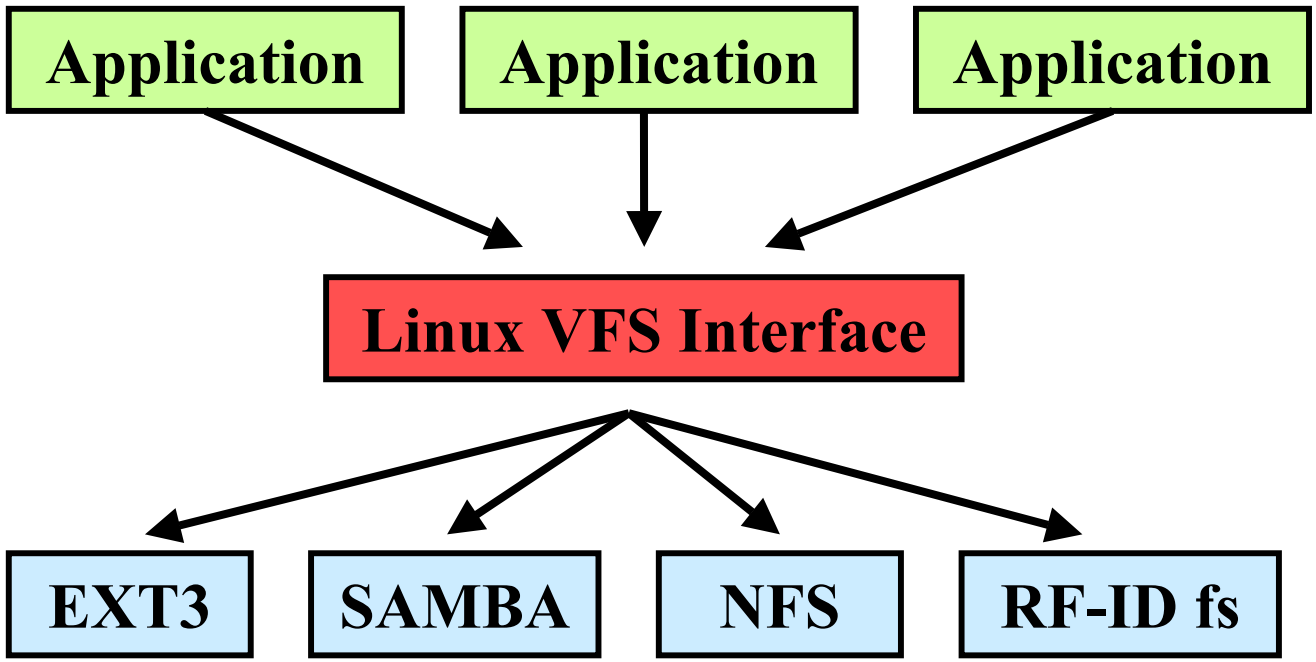
So, our module was accepted for the next version of the kernel! What does this mean?

- It will be distributed with all subsequent versions of the kernel.
- We have the responsibility of **maintaining** our module, adding new features, fixing bugs and supplying any relevant documentation to be included in the kernel docs.
- **"You break it, you fix it!"**
  If changes to another module "break" our own module, we don't have to do anything! It's up to the person who submitted the changes in the first place to make it work. The same rule also applies to our own module as well ;-)

A sneak-peek into the future (I)

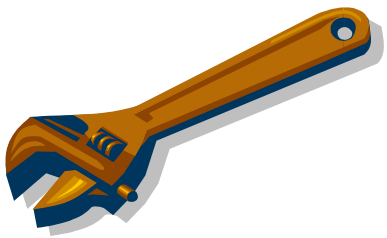
- Our module will provide a **virtual file system.**
- A **virtual file system** is an abstraction to the real file system and offers a unified interface to user applications.





A sneak-peek into the future (II)

- In our proposed File System, RF-ID tags are mapped as special files which I like to call "**R-Files**".
- You should be able to perform the following operations on R-Files:
  - **Rename** them so you may identify special tags easily.
  - **Copy** to other file systems. This involves the creation of a normal file whose content is the RF-ID tag's content.
  - You should be able to organize RF-ID files into **folders**.
  - You can **read** them, **write** them and **use** them in shell scripts just as any other ordinary file.
  - However, **deleting** R-Files will not be supported. What does "*delete a RF-ID tag*" mean anyway?



That's all for now…

For more info:

Linux.com - Writing Your Own Loadable Kernel Module

http://howtos.linux.com/howtos/Module-HOWTO/x811.shtml&e=747

Kernel Hacking: An Introduction to Linux Kernel Programming

http://www.kernelhacking.org

Info on VFS on Linux (Parts I and II)

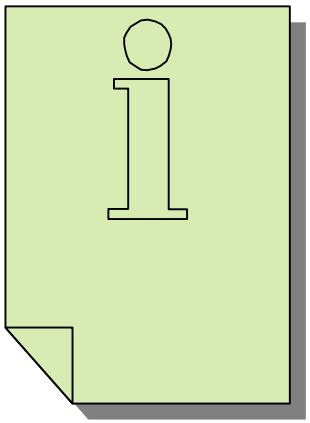
http://www.freeos.com/articles/3851/
http://www.freeos.com/articles/3838/

Devfs (Device File System) FAQ

http://www.atnf.csiro.au/~rgooch/**linux**/docs/devfs.html

An Overview of the Proc Filesystem

http://linuxgazette.net/issue46/fink.html



Any Questions?