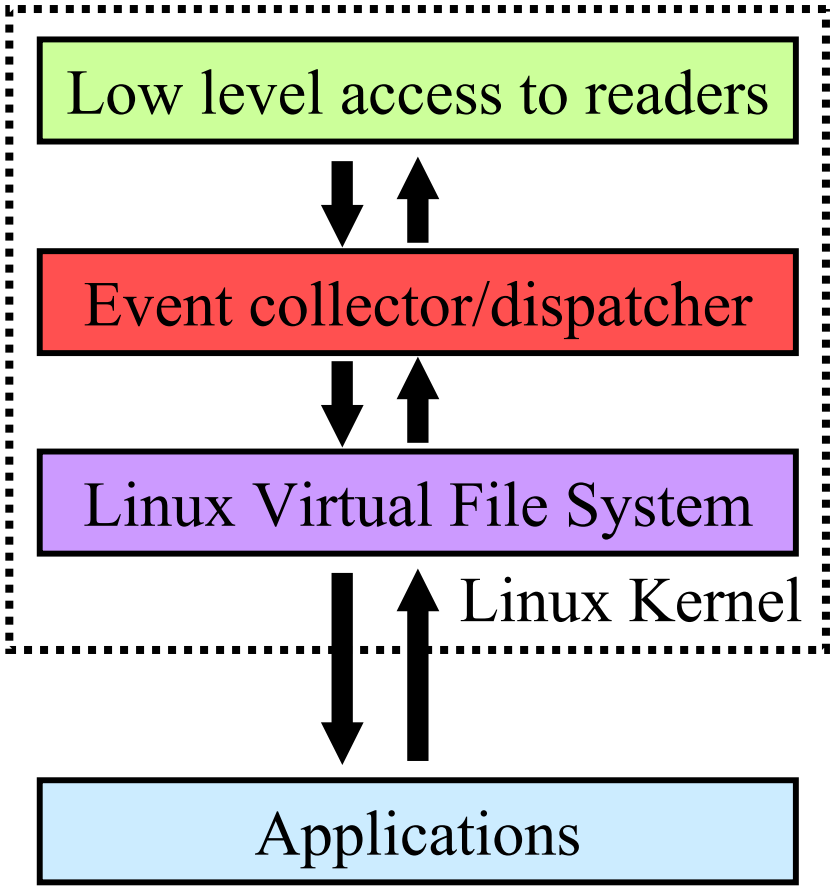
What's on the menu…

Software Comprehension and Maintenance
June 2005

RF-IDs in the Kernel -- Episode III: I want to File Away

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

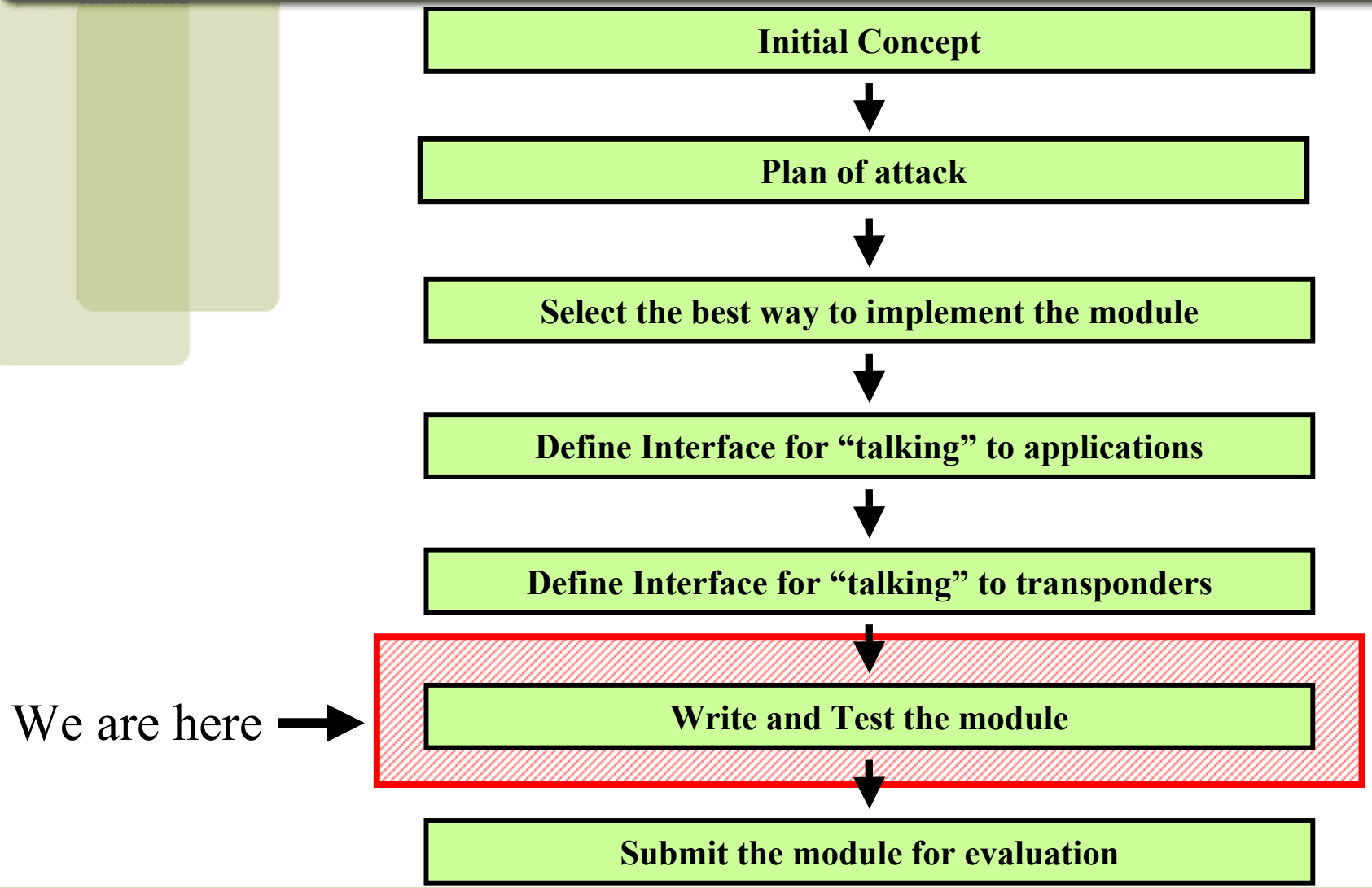
Flashback: What we are trying to accomplish



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.

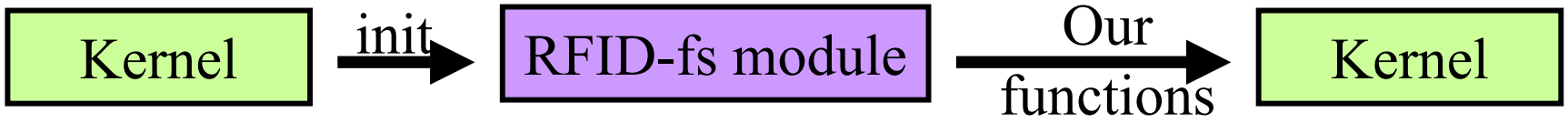
The Roadmap or "Where are we now?"



File Systems in Linux: The boring technical stuff (I)

The notion of the "**i-node**"

- A structure describing an entity in the filesystem and containing all its attributes (size,owner,timestamps…)
- The kernel provides **hooks** which the filesystem uses to map files and folders to i-nodes, create, move(rename) and delete files.
- When the kernel runs our module for the first time we supply a list with all the filesystem functions implemented by our module.



Example: If the kernel wants to look up a file in our filesystem:

File Systems in Linux: The boring technical stuff (II)

Just how many hook-able functions are there? **MANY**

```
void (*read_inode) (struct inode *);
void (*write_inode) (struct inode *, int);
void (*put_inode) (struct inode *);
void (*delete_inode) (struct inode *);
void (*put_super) (struct super_block *);
void (*write_super) (struct super_block *);
int (*statfs) (struct super_block *, struct statfs *);
int (*remount_fs) (struct super_block *, int *, char *);
void (*clear_inode) (struct inode *);
void (*umount_begin) (struct super_block *);
ssize_t (*readv) (struct file *, const struct iovec *, unsigned long,
loff_t *);
……
```

```
loff_t (*llseek) (struct file *, loff_t, int);
ssize_t (*read) (struct file *, char *, size_t, loff_t *);
ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
int (*readdir) (struct file *, void *, filldir_t);
unsigned int (*poll) (struct file *, struct poll_table_struct *);
int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
int (*mmap) (struct file *, struct vm_area_struct *);
int (*open) (struct inode *, struct file *);
int (*flush) (struct file *);
int (*release) (struct inode *, struct file *);
int (*fsync) (struct file *, struct dentry *, int datasync);
int (*fasync) (int, struct file *, int);
…….
```

- Do I need to write ALL those functions myself? **Eeek**!!!
- Nope! You just have to supply the functions you **need** to use. If you are the kind of lazy programmer (like I am ☺) you should consider yourself lucky! The kernel writers even provide you with generic functions to do most of the boring filesystem chores.
- You can actually write an FS in about 245 LOC! (ramfs)

RFID-fs: How is it organized

file_data

char *name
int name_len

char type

inode* my_inode

tag_data

void *data
int data_len
long data_hash

bool is_dirty

reader *my_reader

tag_list

tag_data *item

tag_list *next
tag_list *prev

file_list

file_data *item

file_list *next
file_list *prev

inode_data

tag_data *my_tag

file_data *my_file

file_list *children

reader

Inode *my_inode
tag_list *my_tags

reader_callbacks

void *reader_data

reader_list

reader *item

reader_list *next
reader_list *prev

Speaking of Files and Folders

In RFID-fs there are three types of objects.

- **Tag Files** : They represent RFID-tags. You can read them. like ordinary files and depending on the transponder h/w, also write them.

- **Reader Dirs** : These are automatically allocated by the module when a reader module is enabled. Inside them you can find all tags in the reader's range. However, you cannot copy, rename, move or delete them.

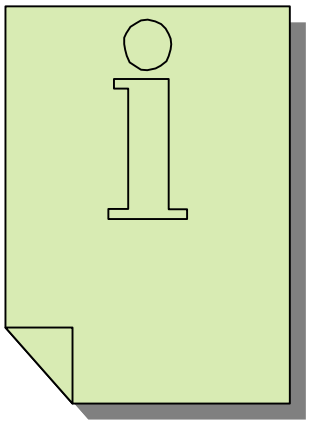
- **User Dirs** : These are directories the user may create for organizing his tags. You can do pretty much whatever you want with these folders.

The Problem: Detecting RFID-tags in the reader's range.

- How can we detect which tags are currently in the reader's range? Remember that if a tag cannot be read anymore, it has to be removed from the filesystem as well.
- The current solution involves polling readers at fixed intervals.
- Readers are generally "slow". Most are in the 10-100 tags/sec range. So, polling isn't really a bad solution! We use the kernel timer mechanism for polling hw every 600-700 ms.
- Before polling, we flag all tags as "dirty". Once a tag is read, we clear its "dirty" flag. Any tags remaining "dirty" are automatically removed.

Some final tips regarding kernel hacking…

- **Google is your best friend!**
- **Use the source, Luke!**
- **There are a lot of e-books out there describing the ins and outs of the linux kernel and its API**
- **When everything else fails, use a hammer…**



The END - Any Questions?

The complete source code of the rfid-fs module will be soon available online