NTNU – Trondheim
Norwegian University of
Science and Technology

Tutorial Lecture for Exercise 3
TDT4258 Energy Efficient Computer Systems

Stefano Nichele
Department of Computer and Information Science
2013, March 15th

Exercise 2

- Deadline: today kl. 12:00 – It’s Learning
- (Brief) presentation to vit.ass (only selected groups). The presentations will be held in the lab.

  When?

NTNU – Trondheim
Norwegian University of
Science and Technology

Exercise 3

- Deadline: Friday 26th April, on It’s Learning

Lab hours with assistance:

Week 12-13: Ekskursjon – Påskeferie

Week 14: Thursday - Friday

Week 15: Monday - Tuesday

Week 16: Monday - Tuesday

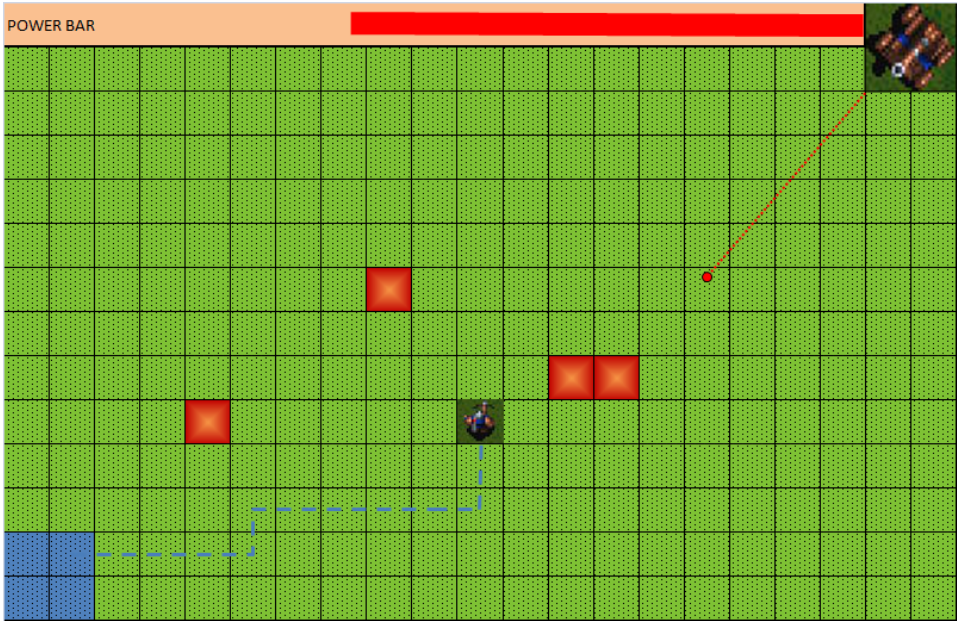
Week 17: Monday - Tuesday

Exercise 3

1. Write a Linux driver for the use of buttons and LEDs on the STK1000
   - Device driver: software layer between the applications and the actual device
     - they hide the details of how the device works
     - they make a particular piece of hardware respond to a well defined programming interface
     - can be built separately from the rest of the kernel and ”plugged in” when needed

2. Create a game (The Scorched Land Defence) that runs under Linux on STK1000

The Scorched Land Defence



Use your creativity
Create a very simple version of the game
No specific requirements on the implementation

Task requirements

- To be written in C language
- The game should run under Linux on STK1000
- Write your own drivers for buttons and LEDs
- Use existing drivers for sound card and LCD monitor

LINUX on STK1000

- Use SD card as ”hard disk”
- Linux kernel and file system on SD card
- Bootloader (*u-boot*) on the microcontroller

Communication with STK1000/Linux

- Serial
  - Cable between PC and STK1000 UART_A
  - Run *minicom –o* on your PC
- Network
  - Find the IP address of STK1000 (eg. With *ifconfig*)
  - *telnet ip-address*

IO devices in Linux

- IO devices are represented by special files in */dev* directory
- To make the I/O
  - Open (with the system call *open*) the file that represents the device to use
  - Execute *ioctl* call, if necessary
  - Read/write with *read* / *write* using *lseek* to switch position
  - Close the file (*close*)
- System calls are documented in man pages

  (e.g. *man 2 open*)

Compiling for AVR32-Linux

- Compiling takes place as before, except that we use programs with the prefix *AVR32-linux-* intead of *avr32-*

- *Avr32-linux-gcc*, *avr32-linux-gdb*, etc.

Compiling new kernel

- Source code published on the course page
- *make xconfig* or *make menuconfig* (can be omitted)
- *make*
- Compiled core in *arch/avr32/boot/images/uImage*, can be copied to */uImage* on the SD card
  - A complete file-system for the SD card is also given (on the course page)

Screen

- Uses framebuffer, /dev/fb0
- Data is written to /dev/fb0 ports on LCD screen
- Format:
  - 32 bit per pixel, 8 bit per color
  - The first row at the top
  - 320x240
- Can use *mmap* system call to display the screen to a table in memory

Audio

- Write audio data to */dev/dsp*
- Standard setup
  - One channel
  - 8bit per sample
  - Sample rate 8000Hz
- Can change setup with *ioctl*

Kernel modules

- The drivers should be created as kernel modules
- The driver should be the only part of the system that has direct access to the relevant PIO registers
- For common programs, LEDs and buttons are available via */dev/foobar*

Creating drivers

- 1st source of information: Linux Device Drivers (essential section 1-3 and 9)
- Compile kernel
- Write driver
- Compile the driver as a kernel module (ends up with foobar.ko)
- Boot up the kernel you compiled and load module
- Create a device file for the driver

Limits

- Standard library is not available
- I.e. No *printf*
- Linux kernel version: *printk*
- *Printk(KERN_INFO ”i = %d\n”, i);*
- *Printk(KERN_ALERT ”Minor damage\n”);*
- *dmesg (”display message” or ”driver message”, command that prints the message buffer of the kernel)*

Startup and shutdown of the module

- Create functions (interface between kernel and module):
  - static int __init foobar_init(void); //allocate, initialize
  - static int __exit foobar_exit(void); //deallocate
- Register it with:
  - module_init(foobar_init);
  - module_exit(foobar_exit);
- Init function is called when the module is loaded and exit function when it is removed

Major and minor number

- Device-files and drivers are connected together with two numbers called *major* and *minor* numbers
- Roughly: major identifies the driver (ie device type) and minor the specific device
- (Use *alloc_chrdev_region* to receive the major number, major and minor are used when creating a device-file)
- In */dev* try *ls -l*

File functions

- The driver contains implementations of file functions:
  - *static int foobar_open(struct inode *inode, struct file *filp);*
  - *static int foobar_release(struct inode *inode, struct file *filp);*
  - *static ssize_t foobar_read(struct file *filp, char __user *buff, ssize_t count, loff_t *offp);*
  - etc. (for example write, seek, ioctl...)

Register file functions

- Create a *struct file_operations* which has links to functions:

```
static struct file_operations foobar_fops = {
    .owner = THIS_MODULE,
    .open = foobar_open,
    .release = foobar_release,
    .read = foobar_read, // etc.
}
```

- Call *cdev_init* with the structure as argument (to tell the kernel how to use those functions)

Use of hardware (I/O ports)

- Need to request for access to hardware with *request_region*
- Otherwise, use the I/O ports in the same way as in exercise 2
- *release_region* when done

Compiling the kernel module

- Must have Linux source code available
- Use Linux build system with a small dose of magic
- See makefile

Loading and removal of the module

- Loading: *insmod foobar.ko*
- Removal: *rmmod foobar*
- List of loaded modules: *lsmod*

Create a device file

- Find major number in */proc/devices*
- *mknod /dev/foobar c major minor*
- *ls –l /dev/foobar* shows the major and minor number

Tips

- Start early. Biggest exercise this year.
- Play with u_boot to obtain the unique MAC-address of the card
- (If you want to use threads, build with –*pthread* flag)
- Make a simple ”hello world” module

Lykke til