Realtime embedded programming: Linux - Raspberry PI, lecture 1

Bernd Porr

1 Intro

- From low level ARM programming to operating system based programming (Linux!).
- Linux is now used on mobile phones (Android), in Blu ray players, cameras (Sony NEX) and many e-readers run Linux.
- In the previous lectures we programmed the ARM on the level of ports and had to take care of everything. Now we program it via high level commands avoiding accessing hardware.
- Advantages: no need to program low level routines, for example saving data on a memory stick. WIFI is plug and play and can directly accessed via IP protocols or sockets. Low level protocols such as SPI are easily accessed with standard high level read/write commands.
- Disadvantages: Linux creates an overhead in terms of computing power. Programming of interrupt based processing requires much more sophisticated programming but is easier to debug. Timing is usually achieved in userspace by high level timers which might not be precise enough for a certain task. Good for slow processes but not so good for faster processes.
- We use here the raspberry PI as an example of an embedded Linux system.

2 The raspberry PI

2.1 Features

- It's an ARM based computer running Linux.
- I/O: 2x USB, ethernet, video out (composite / HDMI / DVI), audio out, compact flash and a general purpose I/O header supporting SPI and I2C.
- It is an open system which has a large community.
- Its configuration is similar to a Blu ray player or media player.
- It's open source: all projects are published on the RPI web site.
- The operating system runs off a SD card which is also used to store data.

2.2 Shortcomings

The main shortcoming is the lack of analogue inputs at the RPI. It is not possible to measure, for example, temperature and plot it although the graphical output is excellent. We will attach our own AD converter to the RPI which is a preceision sigma delta converter.

3 Linux

- Developed by Linus Torvalds as an open source Unix operating system.
- Strictly just the "kernel" to be able to execute UNIX commands and programs.
- The Linux "kernel" is hosted at "www.kernel.org" which provides all basic I/O, filesystems, timers, multithreading, task scheduling, program execution and in general everything to be able to run a program without having to access hardware directly.
- UNIX tools: in order to communicate with the user Linux has a huge amount of command line tools such as "ls, cd, mkdir, ...". Many of them belong to the GNU project (GNU is NOT UNIX).

- A *distribution* is a bootable UNIX/Linux system containing a kernel, selection of UNIX tools and other programs (word processor, drawing programs, compilers, ...). The RPI is based on the "debian" distribution (www.debian.org). Many other distros are based on Debian (for examle UBUNTU). It has an ingenious package management system and an amazing bug tracking system.

- Linux distributions are usually maintained in one place. If the user wants to install software the user usually just selects a package and the distribution downloads it from a central repository.

- Programming languages: C is the language for the kernel. In userspace the standard language is C/C++ but virtually any language is supported (Python, Perl, Pascal, Fortran, assembler, ...).

- Graphical output: Linux is command line driven but has a graphical front end called X-windows system. Software for X is written usually using a library. Popular libraries are GTK or QT. We use QT in our demos. In contrast to Windows there are many different "looks", called Window managers. Popular ones are "gnome" or "K". In the lab we will not use any window manager. We will just plot directly on the screen.

4 Using the RPI

4.1 Startup

The RPI has its operating system on the CF card. Connect the monitor, keyboard, mouse, plug the CF card in its slot and connect the power. You will be greeted with a login. The user is "pi" and the password "raspberry". The screen should show you an IP address which you can log into via ssh.

4.1.1 Changing font size at boot up

The font might be very, very tiny. You can fix this by logging into the RPI and do "sudo -s", "cd /boot" and then start "nano config.txt". Uncomment "hdmi_safe=1". Press "Ctrl-X", "Y", enter and then "reboot". Alternatively you can mount the SD card on your Linux system and edit the file /boot/config.txt.

4.2 Command line programs

In general programs are called in the form "command -option number/parameter filename". Most commands have a so called "man page". For example, "man ls" shows you all options of the command that lists a directory. Press "q" to exit the man page. Use google to find cheat sheets and summaries of all commands.

4.3 User management and admin tasks (sudo)

The normal user "pi" is allowed to compile programs but not allowed to access low level peripherals such as the SPI port and it cannot install software from the distribution. To execute programs with admin rights you need to write "sudo" in front of them. For example, to install software from the distribution type "sudo dselect". You might be asked for your password again. If you need to execute many commands as admin you can switch to admin with "sudo -s" and then type "exit" once you are done.

4.4 Package management

Most programs can be directly downloaded from the distro which is always the safest way. For example "sudo apt-get install emacs" installs the emacs text editor. It automatically downloads it from the the distibution respository. Menu based package managers are "dselect" or "aptitude". So, to install the package manager "dselect" type: "apt-get install dselect".

IMPORTANT: If possible always install a package via the package manager and do not download tar/zips from the web!

4.5 Remote access

The raspberry pi has ethernet and connects to the network so that you can log into it from another computer. The standard way of doing it is via secure shell which is called "ssh". Just type "ssh username@computername/IP" and you will be prompted by a password to log in. If you want to have graphical X windows use the option "-X".

In the lab you first need to log into our server and then from there into the raspberry PI. The IP address of the server is on moodle. Every team has a login on the server. See moodle for login details.

4.6 Downloading software in source code

Open source software is usually distributed using the GPL. This forces people to distribute the source even with binaries so that other people can modify the software.

4.6.1 tar-ball or zip file

There is no browser on the raspberry pi but you can download tar/zips when you have direct link. Type:
"wget http://www.myfavsoftwarepage.com/mygreatprog.tar.gz" The program "wget" downloads any link to your directory. Firefox and other browsers provide you usually with the link (right click).

Convention is that all files are stored in a subdirectory:

```
bp1@bp1-x61:/tmp$ unzip qwt-example-master.zip
Archive:  qwt-example-master.zip
42df3004eb9909c3d49c93ed61b325b08e7e456a
   creating: qwt-example-master/
  inflating: qwt-example-master/QwtExample.pro
  inflating: qwt-example-master/README.md
  inflating: qwt-example-master/main.cpp
  inflating: qwt-example-master/window.cpp
  inflating: qwt-example-master/window.h
bp1@bp1-x61:/tmp$
```

Just unzip them and you are done. The other program is "tar" which takes as a command line "tar xzvf filename.tar.gz" or "tar xjvf filename.tar.bz".

4.6.2 checkout from a respository with git

All our programs are stored in a version control program. The standard program is called "git". To receive all files from the repository type "git clone LINK-PROVIDED-BY-GITHUB", for example:

```
git clone https://github.com/berndporr/qwt-example.git
```

and you should see:

```
Cloning into 'qwt-example'...
remote: Counting objects: 13, done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 13 (delta 4), reused 11 (delta 2)
Unpacking objects: 100% (13/13), done.
bp1@bp1-x61:~/tex/teaching/realtime$ cd qwt-example
bp1@bp1-x61:~/tex/teaching/realtime/qwt-example$
```

The git owner can give you permission to change files which you then commit to git by writing "git commit myChangedFile" and then you can upload them back onto github with the command "git push". Git will keep track of the changes and you can go back to previous versions.

4.7 Compiling a program

You should have all your code in a subdirectory after having unpacked it or downloaded via git. What you need for compilation is the "Makefile" which either already exists or needs to be generated. If "makefile" or "Makefile" exists just type "make" and the program will be compiled.

A makefile simply lists how the different source files depend on each other and how they should be compiled. One can generate their own Makefile just with an editor or use a tool which generates it. There are three tools available: autoconf, cmake and qmake.

Generally always read the README file in the program folder which should tell you what to do. "less" is a good program to read README files or just start the editor "nano".

4.7.1 Autoconf/automake/autogen

- Type "./configure" which generates the Makefile
- Type "make" which will read the "Makefile" and then will compile the program.

4.7.2 cmake

Cmake is a powerful cross platform Makefile generator. It reads a config file called "CMakeLists.txt" which can be created with every text editor. For a

standard user program the file will contain only two or three lines. The rest is done by cmake!

- Type "cmake ." which reads the CMakeLists.txt and generates the makefile.
- Type "make"

4.7.3 QT program (graphical output)

- Type "qmake" which generates the Makefile
- Type "make" to compile the program.

Make sure that you have the right QT libraries installed. QT5 is the up-to-date release. It's available for Windows, Mac and Linux.

4.8 Installing the binaries

Type "make install" or "sudo make install" to install the program. By convention any self compiled program will be installed in "/usr/local/bin" and self compiled libraries in "/usr/local/lib" and "/usr/local/include".

4.9 Running a program

4.9.1 Command line

If it's a command line program you can run it by "./myprogram" or you copy it to "/usr/local/bin" which then runs from any place by just typing "myprogram".

4.9.2 X-windows program

If you have a window manager running you can start it also with "./myprogram" and it will show up. However, in the lab we won't have the window manager running. You need to start it with "startx ./myprogram" or "initx ./myprogram". To keep it running in the background press "ctrl-alt-F1" to get back to the terminal (the graphical program runs in "ctrl-alt-F7").

Stop it by pressing ctrl-c or use the "kill" command to stop it from the terminal.