Sysdev - System development with Scratchbox

Objective: creating another system from scratch, with a DirectFB interface, using the Scratchbox cross-compiling environment.

After this lab, you will

- be able to create a non trivial system from scratch, with elaborate applications and shared libraries.
- see how easy it can be to create an embedded system with the Scratchbox environment, which cross-compiling made as easy as native compiling.
- get more familiar with the DirectFB library and its capabilities.
- get familiar with compiling standard GNU/Linux libraries and tools from their sources.
- be able to use NFS to make the target system boot from a directory shared with the development host.
- be amazed by the capabilities of the qemu emulator!
- see how small the whole working system can be, once all development-only stuff is removed.

Environment setup

Since version 8.04, Ubuntu has extra protections against application and kernel vulnerabilities. However, they interfere with tools like Scratchbox.

In the `/etc/sysctl.conf` file, add the below lines at the end:

```
# Needed for Scratchbox
vm.vdso_enabled = 0
vm.mmap_min_addr = 4096
```

Now load the new values:

```
sudo /sbin/sysctl -p /etc/sysctl.conf
```

Setting `vm.mmap_min_addr` and `vm.vdso_enabled` in `/etc/sysctl.conf` is equivalent to writing the same values in `/proc/sys/vm/mmap_min_addr` and in `/proc/sys/vm/vdso_enabled`. `/etc/sysctl.conf` is the standard way to enforce any `/proc/sys/` settings in a permanent way.

Scratchbox installation

For all GNU/Linux distributions, Scratchbox tarballs are available at http://scratchbox.org/download/files/sbox-releases/apophis/tarball/. However, dedicated packages are available for the Debian and Ubuntu distributions. They make Scratchbox installation much easier, so we suggest you to use them in this lab.

First, we need to add the Scratchbox package repository, as Scratchbox has not been officially integrated into Ubuntu. To do so, add the following line to the `/etc/apt/sources.list` file:

```
deb http://scratchbox.org/debian/ apophis main
```

And run `apt-get update` to download the list of packages. Then, install the following packages using `apt-get install`:

- `scratchbox-core`
- `scratchbox-libs`

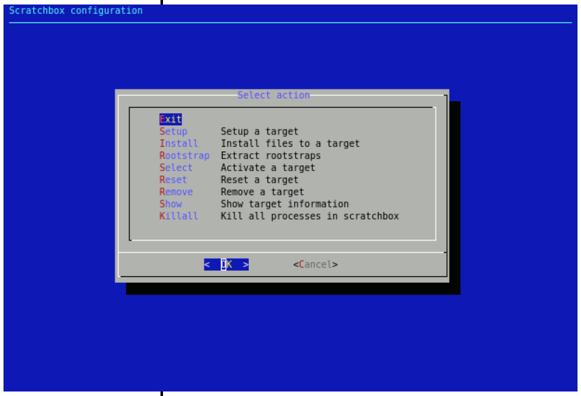
- scratchbox-devkit-cputransp
- scratchbox-toolchain-arm-gcc3.4-uclibc0.9.28

Now add an user:

```
sudo /scratchbox/sbin/sbox_adduser <your_username>
```

Again, accept the default settings.

You may have re-login to your machine, so that you get `sbox` group privileges needed for running Scratchbox. Under your regular account, you can check that this is done by issuing the `groups` command. The `sbox` group should be listed.



```
Scratchbox configuration

                    Select action
          Exit
          Setup      Setup a target
          Install    Install files to a target
          Rootstrap  Extract rootstraps
          Select     Activate a target
          Reset      Reset a target
          Remove     Remove a target
          Show       Show target information
          Killall    Kill all processes in scratchbox

              <  OK  >      <Cancel>
```

Scratchbox target creation

Enter the Scratchbox chroot environment:

```
/scratchbox/login
```

Now configure Scratchbox:

```
sb-menu
```

First set up a new target:

- Target name: `armdemo`
- Compiler: `arm-gcc3.4-uclibc0.9.28`
- Devkits: `cputransp`
- CPU-transparency method: `qemu-arm-0.8.2-sb2`
- Do you wish to install a rootstrap on the target: no
- Do you wish to install files to the target: yes
  Accept the default settings.
- Do you wish to select the target: yes

Your shell is then restarted with a fake home directory corresponding to the target you selected.

You can now only see the development environment and target files:

- On your real root file system, your fake home directory is actually stored in `/scratchbox/users/<user>/home/<user>`.
- Target files are stored in `/scratchbox/users/<user>/targets/armdemo`.

You can also notice that you just see the development tools provided by Scratchbox:

```
which vi
/scratchbox/tools/bin/vi
```

Plenty of other tools you are familiar with are not available (other editors, browsers, `rsync`...). You can still use them, but from outside the Scratchbox chroot.

Cross-compiling test

First, let's check that our cross-compiling environment works fine.

```
tar zxf /scratchbox/packages/hello-world.tar.gz
cd hello-world/
```

```
./autogen.sh
make
```

```
file hello
hello: ELF 32-bit LSB executable, ARM, version 1 (ARM),
dynamically linked (uses shared libs), not stripped
```

```
./hello
Hello World!
```

You can see that we could compile this simple program as if we were natively compiling. The configuration scripts didn't even notice. Not only cross-compiling is transparent, but executing programs for the target platform is transparent too, thanks to using `qemu` behind the scenes.

Compiling required libraries

We are going to compile programs based on the DirectFB graphical library. As written it is documentation, prerequisites are the `zlib`, `libpng`, `libjpeg` and `freetype` libraries.

Again, if you face trouble downloading sources from a given server, you can use our copies available in http://free-electrons.com/labs/sources.

Downloading with `wget` is much easier than from a browser: you don't have to select the right directory to save files to.

Download and compile `zlib 1.2.3` (http://www.zlib.net/):

```
./configure
make
make install
```

Download and compile `libpng 1.2.20` or later (http://www.libpng.org):

```
./configure
make
make install
```

Download and compile `libjpeg 6b` (http://www.ijg.org/):

```
./configure
make
make install-lib
```

Download and compile `freetype 2.3.5` (http://freetype.org):

```
./configure
make
make install
```

When no `--prefix` argument is given to the `configure` scripts, all compiled resources are installed in `/usr/local`. Check what was installed in your target directory.

Compiling the DirectFB library

Download DirectFB 1.0.1 from http://directfb.org.

Once you extracted the sources,
comment out line 1570 in `systems/fbdev/fbdev.c`:

```
//if (dfb_fbdev_compatible_format( var, 0, 5, 6, 5, 0, 11, 5, 0 ))

       return DSPF_RGB16;
```

This is a work-around for an apparent bug in qemu arm LCD emulation or in the corresponding kernel driver (see http://mail.directfb.org/pipermail/directfb-dev/2006-October/002364.html)

Configure and compile DirectFB as follows:

```
./configure --disable-x11 --with-gfxdrivers=none \
          --with-inputdrivers=keyboard,linuxinput,ps2mouse
make
make install
```

Of course, we cannot use any graphics hardware acceleration drivers here. With qemu, everything is eventually implemented in software (unless qemu managed to use capabilities of the host graphics card).

DirectFB and its programs use `pkg-config` to locate their resources. So, you need to set the `PKG_CONFIG_PATH` environment variable to signal where the DirectFB library and its headers were installed:

```
export PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
```

If you didn't do it, DirectFB programs would think that the DirectFB library is not installed.

Compiling the DirectFB examples

Download DirectFB-examples 1.0.0 (and not version 1.0.1) from http://directfb.org.

Configure and compile the programs:

```
./configure
make
make install
```

Compiling BusyBox

Still inside the chroot, have a look at `/bin`, `/usr/bin/`, `/sbin`... You can see that these directories are mostly empty!

To fill the target filesystem with standard Unix utilities, compile BusyBox 1.7.2 with the configuration file available in `/home/<user>/felabs/sysdev/scratchbox/data/`.

It's nice not to have to configure cross-compiling, isn't it?

Install it in the target directory by configuring the location for the `make install` command.

Completing the root filesystem

As in the previous lab, you need to add the device files which the applications are going to need: `/dev/console`, `/dev/tty5`, `/dev/null`.

Also add the device files needed by DirectFB: `/dev/fb0` (framebuffer), `/dev/tty0`, `/dev/tty1`, `/dev/tty2`, `/dev/tty3`, `/dev/input/mice` (all the mice in the system, merged in a single one).

How did we know which ones to create? Actually, just by waiting for error messages from the applications!

Create the `/proc/` and `/sys/` directories.

As in the previous lab, also add a `/etc/inittab` file and the corresponding `/etc/init.d/rcS`, doing the following things:

- Mounting `/proc` and `/sys`.
- Setting the `PATH` and `LD_LIBRARY_PATH` environment variables:
  ```
  export PATH=/usr/local/bin:/bin:/usr/bin:/sbin:/usr/sbin
  export LD_LIBRARY_PATH=/usr/local/lib:/lib:/usr/lib
  ```
- Starting a new interactive shell.

Also create the `/usr/local/lib/directfb-1.0-0/gfxdrivers` directory. Otherwise, DirectFB programs will keep complaining that this directory doesn't exist.

Booting the virtual board

At last! Your efforts will be rewarded.

Open a new user terminal and go to `/home/<user>/felabs/sysdev/scratchbox/`.

For our best convenience, we are not going to create a root filesystem out of the target files. Instead, we will boot directly from the target directory.

This way, whenever we make a change to the target files from the host, we won't have to update a target root filesystem image and reboot, as we had to do in the previous lab.

Have a look at the `/etc/exports` file configuring the NFS server to export the target directory.

Have a look at the `run_qemu` script in the current directory, and see how the NFS client (here, the Linux kernel) connects to the NFS server. In this script, make sure the path of the NFS exported directory contains your user name.

Now, run this script and see your emulated target boot!

Don't hesitate to show your problems (if any!) with your instructor. You can't be far from a working system!

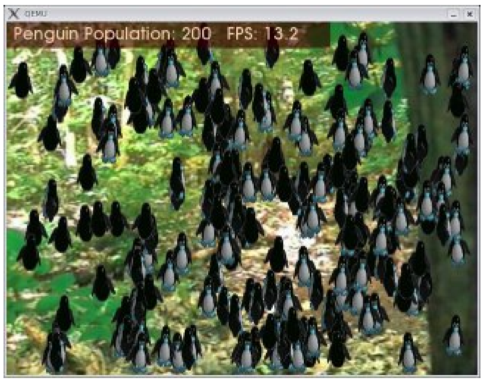
Testing the DirectFB example programs

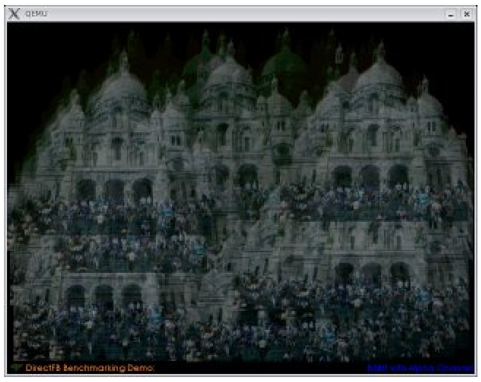
Once you started these programs, you can exit them through the `[q]` or `[Esc]` keys.

Some of them will miserably freeze your virtual target. In that case, you will have to restart qemu.

Here are our favorite programs:

- `df_andi`: a population of 200 penguins invading your screen.
- `df_dok`: benchmarking the performance of graphic primitives.
- `df_fire`: drawing a wall of fire.
- `df_input`: testing input drivers.
- `df_knuckles`: Napoleon's head trying to say something. Accelerated 3D graphics obviously making his message difficult to understand.
- `df_neo`: funny sprite animation.
- `df_palette`: an animated color palette.
- `df_window`: overlapping and moving translucent windows.
- `spacedream`: moving stars.





Congratulations! You built all this by yourself!

By studying the DirectFB examples, you should be able to easily create your own applications and interface for your real embedded systems. With Scratchbox, system development is much easier than you thought, isn't it?

Making your system smaller, ready for production

How big is your target filesystem? Wow!

Your system still contains stuff needed for development, but which can be removed when moving to production.

First, create a copy of your target development directory, and modify `/etc/exports` and `run_qemu` to use it instead.

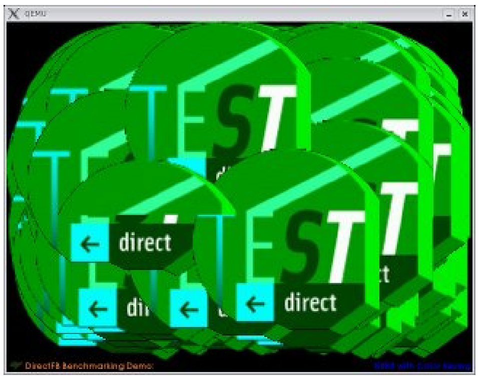
In the copy, remove the below directories:

- `/usr/include` (48 M), `/usr/local/include` (2.3M): C headers.
- `/usr/local/share/man` (236 K): manual pages.
- `*.a` library object files (13.2 M) and `*la` links to them: only needed for compiling. Remove them with (caution: do that in the chroot!):
  ```
  find . -name "*.a" -exec rm {} ';'
  ```

```
find . -name "*.la" -exec rm {} ';'
```

- `/usr/lib/libstdc++.so.6.0.3` (2.4 M): unused shared libraries (C++ not used in our case).
- `/usr/lib/libfakeroot` (60 K), `/usr/local/lib/pkgconfig` (28K): no longer needed in production.
- `/usr/bin/gdbserver`, `/usr/bin/strace` (304 K): programs no longer needed in production.
- `/usr/local/share/aclocal` (14K): just needed for development.

At the end, how small was your system?



Detecting unused files

Implement a mechanism to identify files which are not accessed at boot time, and through all the programs that you want to run.

Then, remove these files and check that your system still boots.

Ask your instructor if you don't have enough ideas, or if you have questions on how to implement these ideas.

End result

You can check how small the system could be by having a look at our page about this demo:
http://free-electrons.com/community/demos/qemu-arm-directfb/

You can go to this page to get updates and information about the techniques you can use to reduce the size and boot time of this demo.