Secure OS Layer Provided by Genode and POSIX API on a variant of L4 Microkernel

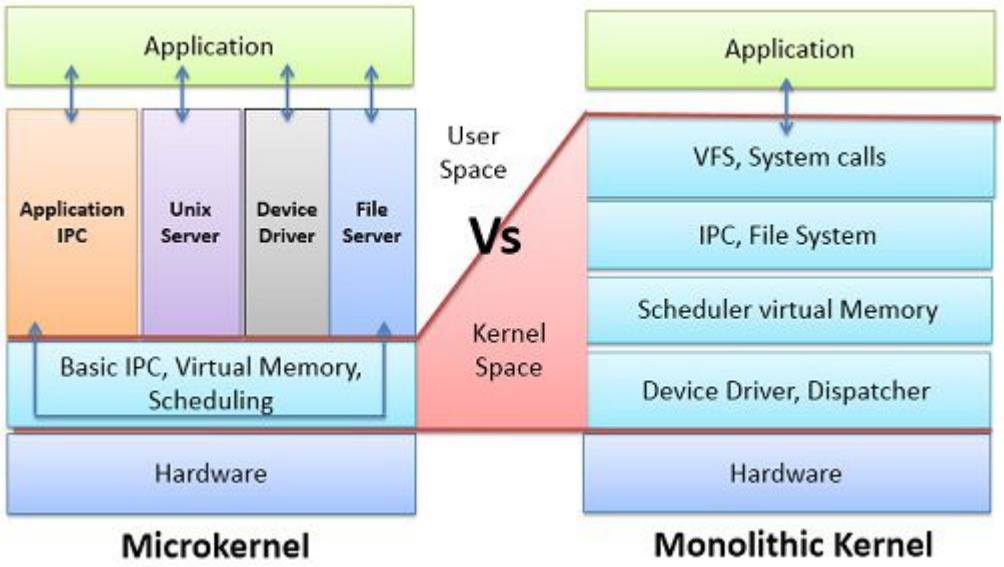
August 2018

Contents

1 Introduction

1.1 Objective

The kernel is a computer program that is the core of a computer's operating system, with complete control over everything in the system. On most systems, it is one of the first programs loaded on start-up (after the bootloader). It handles the rest of start-up as well as input/output requests from software, translating them into data-processing instructions for the central processing unit. It handles memory and peripherals like keyboards, monitors, printers, and speakers. There are two major design approaches to kernels - monolithic kernels and micro kernels. Monolithic kernel is a single large process running entirely in a single address space. It is a single static binary file. All kernel services exist and execute in the kernel address space. The kernel can invoke functions directly. Examples of monolithic kernel based OSs: Unix, Linux. In microkernels, the kernel is broken down into separate processes, known as servers. Some of the servers run in kernel space and some run in user-space. All servers are kept separate and run in different address spaces. Servers invoke "services" from each other by sending messages via IPC (Interprocess Communication). This separation has the advantage that if one server fails, other servers can still work efficiently. Examples of microkernel based OSs: Mac OS X and Windows NT.



In theory, a microkernel, by putting the bulk of the driver code into userspace, is more resilient against attacks: an attack against one driver can't easily be used to leverage access to the other drivers, or gain kernel-level privileges. Additionally, the reduced size of the kernel gives it a much smaller attack surface. Usually their capabilities are more restricted then those of a "general purpose" monolithic kernels and thus contain less bugs purely because their codebase is smaller.

1.2 Genode

Genode is a free and open-source operating system framework consisting of a microkernel abstraction layer for building highly secure special-purpose operating systems. It scales from embedded systems with as little as 4 MB of memory to highly dynamic general-purpose workloads. The framework aligns the construction principles of L4 with Unix philosophy. In line with Unix philosophy, Genode is a collection of small building blocks, out of which sophisticated systems can be composed. But unlike Unix, those building blocks include not only applications but also all classical OS functionalities including kernels, device drivers, file systems, and protocol stacks.

1.3 L4 Microkernel

L4 is a family of second-generation microkernels, generally used to implement Unix-like operating systems, but also used in a variety of other systems. Since its introduction, L4 has been developed for platform independence and also in improving security, isolation, and robustness. Sel4 is a secure variant of the L4 microkernel which aims to provide a basis for highly secure and reliable systems.

2 Tools Required

2.1 Hardware

- **NXP's IMX6Q-SABRESD** Board: Documentation about the board can be obtained from NXP's website
- USB Serial Cable

- Machine running Ubuntu 16.04 LTS
- FAT formatted SD-Card

2.2 Software

- U-Boot Bootloader
- Genode

  Source can be obtained from `https://github.com/genodelabs/genode`
- "gtkterm" or similar serial terminal
- Packages to be installed on Ubuntu:

  make, libSDL-dev, expect, qemu, genisoimage, byacc, autoconf2.64, autogen, bison, flex, g++, git, gperf, libxml2-utils, subversion, xsltproc, qemu-system-arm, texinfo, libncurses5-dev, libexpat1-dev

3 Procedure

The following section outlines the steps to be taken to boot the Secure OS on an IMX6Q-SABRESD board.

3.1 Setting up the bootloader

We will be using U-Boot. Das U-Boot (subtitled "the Universal Boot Loader" and often shortened to U-Boot) is an open source, primary boot loader used in embedded devices to package the instructions to boot the device's operating system kernel.

The source can be obtained from `https://github.com/u-boot/u-boot`. Since IMX6Q-SABRESD uses ARM Instruction Set, we need a cross-compiler as we are working on an Intel machine. It can be downloaded from ubuntu's repository. The cross-compiler used in here is `arm-linux-gnueabihf-gcc`. The following steps need to be followed to bring up u-boot on the device.

Now a u-boot directory exists with all the latest sources. Type the following commands in the Ubuntu terminal.

```
$ cd u-boot
$ export CROSS_COMPILE=arm-linux-gnueabihf-
$ make mx6qsabresd_config
$ make
```

This should create a number of files, including `u-boot.imx`. Now copy the bootloader to your SD Card. It should reside at offset 1024B on the SD Card. Use the following command for the same.

```
$ dd if=u-boot.imx of=/dev/<your-sd-card> bs=1k seek=1;sync
```

3.2 Setting up Genode

After obtaining the source from Github, the toolchain must be setup - this toolchain is for cross compilation of Genode for IMX6Q-SABRESD. The script for setting up the toolchain is `/genode/tool/tool_chain`. Execute the script with `arm` as its command line parameter, as `$ ./tool_chain arm`. This sets up the toolchain in `/usr/local` directory. The `/genode` directory contains subdirectories like `/depot, /doc, /repos, /tool`.

Now we can begin with compiling Genode for IMX6Q-SABRESD. A script `/genode/tool/create_builddir` is used to set up the initial configuration for our hardware. When the script is executed without any parameters, it displays the available configurations.

We will choose `wand_quad` as it is the closest configuration for our available hardware. Certain modifications can be made to tailor the configuration for the hardware in hand. When the script is executed with command line parameter as wand quad, as `./create_builddir wand_quad` a directory is created at `/genode/build/wand_quad`.

3.3 Tweaks for IMX6Q-SABRESD

- `wand_quad` board has 2GB ram. The hardware at hand has 1GB ram. In `/repos/base/include/platform/imx6/drivers/board_base.h`

change the value of `RAM0_SIZE` from `0x80000000` to `0x20000000` or `0x40000000`.

• In `/genode/repos/base-hw/mk/spec-hw.mk` and `/genode/repos/base-hw/mk/spec-hw_wand_quad.mk` change the value of `NR_OF_CPUS` to `1`.

• In `/genode/build/etc/build.conf`, add the line
`RUN_OPT += $include image/uboot`
This is to enable uboot detect the Genode image and begin booting it.

3.4 Using Genode

Once the initial configuration settings have been modified, it can be built. Run the following commands in `/genode/build/wand_quad`

```
$ make init
$ make drivers
$ make
```

This creates a number of directories like `/app, /bin, /core, /drivers, /etc, /init, /lib, /server, /test, /var`. Make sure `/core` gets created. If it does not exist, run `make` again.

3.5 Compiling Genode over Sel4

We will use a script called `printf.run` that exists in `/repos/base/run` to demonstrate the working of Genode over Sel4. First we need to include capability for Genode to run on Sel4. The developers have already included basic functionality for the same. A directory `/repos/base-sel4` exists for this.

For the following step, install these packages from the Ubuntu repository - `python_future, python_tempita, python_ply, python_six`.

In `/genode/tool/ports` a script called `prepare_port` exists. Execute the script as shown below, from `/tool/ports` directory.

```
$./prepare_port sel4
$./prepare_port sel4_elfloader
```

In `/genode/build/wand_quad/etc/build.conf`, uncomment the following line `REPOSITORIES += $(GENODE_DIR)/repos/libports`. This is to include the libraries for sel4 in our Genode build.

Now navigate to `/genode/build/wand_quad` and execute
`$ make run/printf KERNEL=sel4`

Once the script executes successfully, a kernel image of Genode over sel4 gets built. Navigate to `/genode/build/wand_quad/var/run/printf` to find a file named `uImage`.

3.6 Running Genode over Sel4

This sections entails the steps to be followed to run Genode on the IMX6Q-SABRESD. Genode will be booted on the device using the SD-Card. Section 3.1 discussed about setting up the bootloader on the SD-Card. Now copy the `uImage` generated in the previous step to the SD-Card that has been formatted using FAT file system.

Connect the device to the Ubuntu machine using serial cables and open the desired serial terminal - gtkterm has been used here. In gtkterm, make sure the appropriate serial device has been selected, here `/dev/ttyUSB0` and the baud rate needs to be set to 1152000. Once this is done, insert the SD-Card into the device and boot it.

U-boot begins to load. Press any key to terminate auto-boot. The u-boot prompt should appear. Type the following command in the u-boot prompt, to boot the device from the SD-Card.

```
=> mmc dev 1
=> fatload mmc 1:1 0x12000000 uImage
=> bootm 0x12000000
```

`0x12000000` is the load address of the kernel for this device. It is obtained from the processor manual. The `uImage` is loaded from the FAT formatted

SD-Card to the device's memory. The command `bootm` boots the image that has been loaded into memory.

Here the kernel image has been booted from memory. Once the final required image has been obtained, the image can be flashed to the `emmc` chip of the device, to boot from it.

The following screenshot shows what the output looks like, once Genode is booted on the device.