Indian Institute of Technology

Madras

Running TraceTool in Genode for sabrelite platform

Authors:

Nithin Joshua Stephen

(CS14M033)

Supervisor:

Prof. Kamakoti

04-Aug-2015

Introduction

All traditional operating systems follow a monolithic kernel architecture in which the entire functions of the operating system reside in a single kernel. The entire operating system functionalities is working in kernel address space. Since the entire functionalities are incorporated into the kernel, the size of the kernel tends to be huge. In contrast, in a microkernel architecture only the essential functionalities needed by the operating system is incorporated into the kernel. These functions work in kernel space and run in supervisor mode. These essential functions include low-level address space management, thread management, and inter-process communication. Other functions like device drivers for sound, network run in user space. This results in a significant reduction of size compared to monolithic kernels. Genode OS follows microkernel architecture with emphasis on security where security critical functions are seperated from the rest of the OS.

Architecture

In genode, a program(components) runs in a sandbox called protection domain. The components need to communicate with each other in a secure manner. These components interact with each other through remote procedural calls(RPC). Each component implements an RPC interface which allows other components to access its functions(capability). So a components is similar to an *object* in object oriented terms and capability to a *pointer*. So only if a component has a capability to other component, then only it can interact with it through remote procedural calls. Capabilities enable components to call methods of RPC objects provided by different protection domains.

In genode, every component other than the top level component has a parent. So when a child component is created, the parent assigns a fraction of its resources to the child. A component which provides a service(say GUI) announces the service to its parent. If a component requires a service, the component sends a session request to its parent. The component providing the service is termed server and the component availing the service is termed client. The client obtains the capability to the server and avails the service.

Code Structure

The genode code structure is organized as follows

- repos - This folder contains the core part of source code which is further subdivided into mainly
  - base - This tree contains code written by the creators of genode which can be used by developers to their liking.
  - base-hw - This tree contains code written for basic ARM hardware.
  - os - This tree contains the basic functionalities like device drivers and other system services.
  - libports - This directory contains ports of popular 3rd-party software to Genode.
  - ports - This directory contains ports of popular 3rd-party applications to Genode.
- tool - This directory contains tools for managing and using the source code for various platforms.

Compilation

To compile a custom component, we do a make on a run file(.run extension) with a location relative to a tree inside repos folder. If a run file say hello.run is located in repos/base/run we do *make run/hello*. The makefile first has to find the corresponding run file. This is specified in a build.conf file located in build/$platform/etc where $platform is the platform on which we try to build. Here it is hw_sabrelite. After locating the run file, it decides which all objects which it has to build. This is specified in attribute build_objs in the run file. To build an object, it scans for target.mk makefile which tells genode system which all source files needs to be compiled, which all headers files need to be included, libraries required and the name of the target file to be formed. A object to be build may be dependent on other libraries. These dependencies are generated by dep_prg.mk and dep_lib.mk residing in repos/base/mk folder. These dependencies are written to a dynamic makefile libdeps located in build/$platform/var folder. Then make is done on the libdeps file to complete the compilation phase. The build objects are then combined to form the boot image. The list of files to be combined is specified in the variable build_image. The build_image can contain other executables which are not created in our current compilation phase. These executables need to created prior running

our compilation. For example, the dynamic linker in genode(ld.lib.so) is to be created for complex components by executing *make test/ldso*.

TraceTool

TraceTool is developed by Duss Pirmin and is helpful in debugging genode programs. It reads the trace messages from the Genode OS framework and sends them over a terminal session. The essential components are

- Dut - Device under trace which contains timer functions for starting and stopping the trace session.
- TraceTool - The implementation of the actual trace session based on Model View Controller architecture.

The compilation of TraceTool is successful for hw_sabrelite platform while the creation of the final boot image is unsuccessful as it needs the executable nic_drv which is the NIC driver developed for the platform on Genode.

Work Done

- Studied advanced C++ concepts such as polymorphism, inheritance, virtual functions, templates and exceptions.
- Studied about architecture and build system of genode.
- Integrated sabrelite support(already implemented in an earlier version) to latest genode version.
- Studied about NIC driver implementation in genode for x86 platforms.

Future Work

NIC driver has been developed in genode for x86_32 and x86_64 platforms but not developed for ARM architecture. Since sabrelite uses ARM architecture, development of NIC driver is crucial for network related functionalties to be impemented in Genode for sabrelite platform.

References

[1] Norman Freske *Genode Operating System Framework.*

[2] https://github.com/genodelabs/genode

[3] https://github.com/trimpim/TraceTool/