Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

System Construction Course 2015,

Assignment 2

Felix Friedrich, ETH Zürich, 22.9.2015

Introduction

Oberon programs consist of modules that can be linked statically in a kernel or loaded dynamically when a kernel is already running. The systems that we deal with in this course support dynamic module loading. Moreover, Minos implements module loading over a serial connection.

In this exercise we will learn about the mechanism of linking and loading in some detail. We start with a little demonstrating example and then implement module unloading.

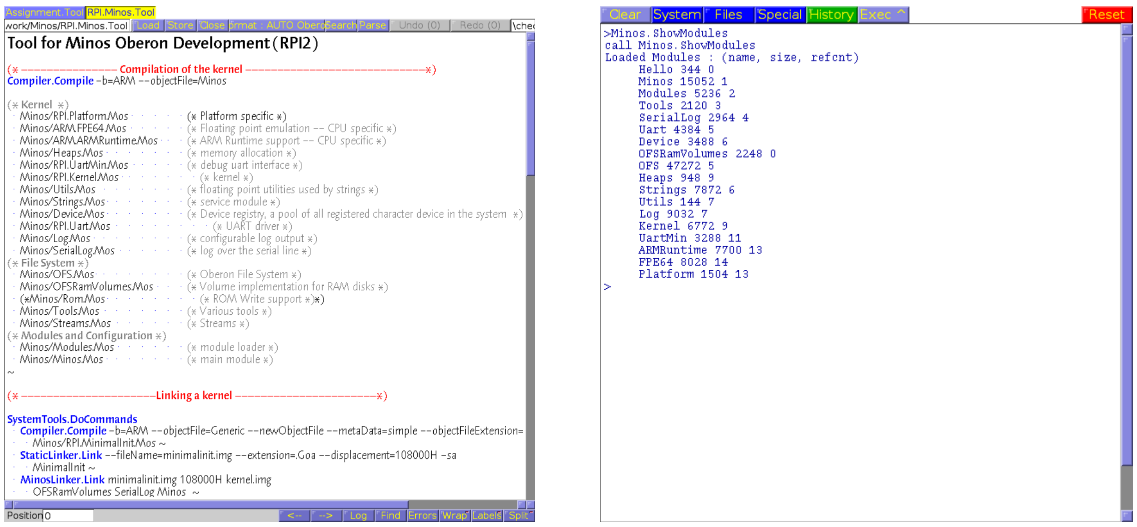
Lessons to Learn

- Learn to know the programming model of Oberon.
- Understand the concept of linking and loading.
- Get some insight into the module loading implementation on the Minos system.

Preparation

1. Update your repository
2. Copy the new files / directories from the `Work` folder located in (repo)/a2/ to your work directory.
3. For communication between the host PC and the Raspberry Pi we will use a serial connection. In order to provide the host with a connectivity to RS232 we use a special cable that contains a USB to serial conversion chip. At the one end of the cable you will find the UBS plug containing the conversion chip and on the other end there are four wires: red power, black ground, white RX into USB port, and green TX out of the USB port. The power pin provides 5V direct from the USB port and the RX/TX pins are 3.3V level for interfacing with the most common 3.3V logic level chipsets. If you intend to power the Raspberry Pi directly from your computer using this cable, do not connect the power cable used last week.

Consult the BCM2835 ARM Peripherals Technical Manual (page 102) in order to find out on which pins you have to connect the cable. The Minos kernel configures ports RXD0 and TXD0 as alternate function 0. Use slide number 45 from the course in order to find the GPIO-pin to pinout mapping. Before you connect, double check your findings with figure `https://svn.inf.ethz.ch/svn/lecturers/vorlesungen/trunk/syscon/2015/shared/documents/rpi/usbttl_wiring.jpg` in the repository. **You can physically destroy the cable and / or the RPI board when you connect it in a wrong way. Be careful! If in doubt, ask me before connecting!** Again: do not power the device from two sources!



```
Tool for Minos Oberon Development(RPI2)

(* --------------- Compilation of the kernel -----------------------------*)
Compiler.Compile -b=ARM --objectFile=Minos

(* Kernel *)
  Minos/RPI.Platform.Mos          (* Platform specific *)
  Minos/ARM.FPE64.Mos             (* Floating point emulation -- CPU specific *)
  Minos/ARM.ARMRuntime.Mos        (* ARM Runtime support -- CPU specific *)
  Minos/Heaps.Mos                 (* memory allocation *)
  Minos/RPI.UartMin.Mos           (* debug uart interface *)
  Minos/RPI.Kernel.Mos              (* kernel *)
  Minos/Utils.Mos                 (* floating point utilities used by strings *)
  Minos/Strings.Mos               (* service module *)
  Minos/Device.Mos                (* Device registry, a pool of all registered character device in the system *)
  Minos/RPI.Uart.Mos                  (* UART driver *)
  Minos/Log.Mos                   (* configurable log output *)
  Minos/SerialLog.Mos             (* log over the serial line *)
(* File System *)
  Minos/OFS.Mos                   (* Oberon File System *)
  Minos/OFSRamVolumes.Mos         (* Volume implementation for RAM disks *)
  (*Minos/Rom.Mos                   (* ROM Write support *)*)
  Minos/Tools.Mos                 (* Various tools *)
  Minos/Streams.Mos               (* Streams *)
(* Modules and Configuration *)
  Minos/Modules.Mos               (* module loader *)
  Minos/Minos.Mos                 (* main module *)
~

(* -------------------Linking a kernel ---------------------*)

SystemTools.DoCommands
  Compiler.Compile -b=ARM --objectFile=Generic --newObjectFile --metaData=simple --objectFileExtension=
    Minos/RPI.MinimalInit.Mos ~
  StaticLinker.Link --fileName=minimalinit.img --extension=.Goa --displacement=108000H -sa
    MinimalInit ~
  MinosLinker.Link minimalinit.img 108000H kernel.img
    OFSRamVolumes SerialLog Minos ~
```

```
>Minos.ShowModules
call Minos.ShowModules
Loaded Modules : (name, size, refcnt)
     Hello 344 0
     Minos 15052 1
     Modules 5236 2
     Tools 2120 3
     SerialLog 2964 4
     Uart 4384 5
     Device 3488 6
     OFSRamVolumes 2248 0
     OFS 47272 5
     Heaps 948 9
     Strings 7872 6
     Utils 144 7
     Log 9032 7
     Kernel 6772 9
     UartMin 3288 11
     ARMRuntime 7700 13
     FPE64 8028 14
     Platform 1504 13
>
```

Figure 1: Minos Build Commands (left) and terminal (right)

1 Introductory example

Preparation

Open file `Assignment2/Assignment.Tool` in the development environment (if not already open). Compile the components required for Minos.

Open file `Minos/Minos.Tool` in the development environment A2. **This tool contains all commands described in the following**. To build the Minos Boot Image containing the statically linked part of Minos (the "Kernel"), the kernel modules have to be compiled and linked. To compile the modules execute command

```
Compiler.Compile -b=ARM --objectFile=Minos ...
```

in Minos.Tool, cf. Figure 1. This compiles all kernel modules and generates object files in the format discussed in the lecture. In order to generate a runnable kernel some initial code has to be prepared that is executed before the kernel module bodies can run. Effectively this code sets up the initial stack and copies the kernel from address `0x8000` to address `0x108000` and jumps to the starting address of the kernel.

The following commands (also contained in `Minos.Tool`) generate this initial code:

```
Compiler.Compile -b=ARM --objectFile=Generic --newObjectFile --metaData=simple
    --objectFileExtension=.Goa --symbolFileExtension=.Sya Minos/RPI.MinimalInit.Mos ~
StaticLinker.Link --fileName=minimalinit.img --extension=.Goa --displacement=108000H -sa
    MinimalInit ~
```

Now the Minos boot image can be built using the command

```
MinosLinker.Link minimalinit.img 108000H kernel.img OFSRamVolumes SerialLog Minos  ~
```

Store the kernel on the micro SD card.

With the physical connections being established, execute on the Host PC the commands `V24.Scan` folled by `Serials.Show`. Oberserving the log should make it possible to identify the COM port where the board is connected.

For linux users: it is required that you map the associated file (usually `/dev/ttyUSB0`) to a COM port number (e.g. 5) within A2 using the command `V24.Map 5 /dev/ttyUSB0`. If this does not work directly, then please consult the next paragraph.

Try the connection by opening a Minos host terminal using command `MinosHost.Open 5` (replace the appropriate COM port number).

Using the USB-to-Serial Cable in Linux

Having plugged in the Usb-/Serial cable, access to the serial connection may be immediately available via as `/dev/ttyUSB0` (or some port different from 0). Using a terminal application such as minicom or putty, try to connect to the client board. If this does not immediately work try the following.

```
admin@server ~ $ lsusb
Bus 002 Device 002: ID 090c:37a2 Silicon Motion, Inc. - Taiwan
Bus 005 Device 002: ID 067b:2303 Prolific Technology, Inc. PL2303 Serial Port
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

If you see (as in the example above) a Prolific to serial device, you can add the respective module to the kernel with the following command. Replace the vendor and product id by the one that you see from lsusb..

```
admin@server ~ $ sudo modprobe usbserial vendor=0x067b product=0x2303
Check if the installation worked by executing
admin@server ~ $ dmesg
[ 131.747013] USB Serial support registered for pl2303
[ 131.747038] pl2303 5-1:1.0: >pl2303 converter detected
[ 131.758965] usb 5-1: >pl2303 converter now attached to ttyUSB0
```

After having installed the usb-to-serial driver it may be necessary to either grant user access rights to the device (chmod) or run A2 with super user rights.

Tasks

Create a new Oberon module called `Hello.Mos`[1] and write a simple “Hello World” program, i.e. a program that prints “Hello World”. Do not link this module to the Minos kernel image, but download and link it dynamically at runtime. You can use the procedure `Log.SL("...")` to print a string followed by the newline character.

Open the Minos host terminal using command `MinosHost.Open <portnr>`. Commands are executed in the Minos-Host terminal using `SHIFT-ENTER`. Check that a connection is established by executing `Minos.ShowModules`. Then execute your code.

Write a little program to blink the onboard LED. Consult module Minos/RPI.Platform.Mos (procedure `LED`) in order to access the onboard LED.

2 Module unloading

Your Minos has no built-in mechanism to unload a module yet. Your task is to add this to Minos.

[1] using the Mos suffix makes sure that the compiler in the IDE generates object files for Minos

Tasks

Implement procedure `Unload(CONST name: ARRAY OF CHAR)` in module `Modules` to unload a currently loaded module from the system. Make sure a module is only unloaded if no other module imports it. Do not forget to adjust the reference counters of modules that are imported by the unloaded module. Please ignore the fact that memory cannot be de-allocated.

Remark: The variable root in `Modules.Mos` contains the root of a linked list of all currently loaded modules. You will find that currently the data structure `Modules.Module` is not prepared for keeping track of imported modules. However, changing the actual interface of the type `Modules.Module` would require a change of the linker. (Why?) Develop an alternative. You will find a hint in the file `Assignment2/Modules.Mos`. In general, in order to give some hints there are comments starting with `STUDENT` in the source code that guide you where you need to amend or modify the source code.

Check that your module unloading works properly.

Documents

- BCM2835 ARM Peripherals Technical Manual in the `documents/rpi` folder of the repository
- A2 Quickstart Guide in `documents/oberon` folder of the repository
- System Construction Lecture 2 slides from the course-homepage
  `http://lec.inf.ethz.ch/syscon`

More documents that are not strictly required for this exercise can be found in the repository .