Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

System Construction Course 2018,

Assignment 2

Felix Friedrich, ETH Zürich, 2018

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 a protocol for 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 folder shared/assignments/assignment2 to some working directory (or take this as your working 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 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!

1 Introductory example

Preparation

1. The tools required to build a Minos kernel are statically linked into the Oberon command line kernel (Windows: win32.oberon.exe, win64.oberon.exe; Linux: linux32.oberon linux64.oberon). Copy the appropriate file to file `oberon.exe` (Windows) or `oberon` (Linux).

2. Compile and link the Minos kernel using the command

   ```
   ./oberon execute MakeMinos.txt.
   ```

   The steps contained can also be executed one by one in the oberon shell:

   (a) Compile all modules of the Minos kernel, creating particularly compact and simple object files in a special format suited for Minos, as discussed in the lecture.

   ```
   Compiler.Compile -p=Minos Minos/RPI.Platform.Mos Minos/ARM.FPE64.Mod
       Minos/ARM.Builtins.Mod Minos/Heaps.Mos Minos/RPI.UartMin.Mos
       Minos/RPI.Kernel.Mos Minos/Utils.Mos Minos/Strings.Mos
       Minos/Device.Mos Minos/RPI.Uart.Mos Minos/Log.Mos
       Minos/SerialLog.Mos Minos/OFS.Mos Minos/OFSRamVolumes.Mos
       Minos/Tools.Mos Minos/Streams.Mos Minos/Modules.Mos Minos/Minos.Mos
       Minos/Out.Mos Minos/In.Mos
   ```

   (b) 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 initial code is compiled and linked using the following commands.

   ```
   Compiler.Compile -p=ARM Minos/RPI.MinimalInit.Mos
   Linker.Link -p=RPI --fileName=minimalinit.img MinimalInit
   ```

   (c) The prelinked initial code and the Minos modules are now linked together using a linker targeted towards the Minos object file format and runtime system

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

3. Store the kernel on the micro SD card.

4. **For windows users:** 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: With the physical connections being established, 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. Often problems result from privilege problems.

5. Open a connection from the oberon shell with the command `Zeus.Connect <portNumber>`[1] where `<portNumber>` has to be replaced by the respective COM-port number.

6. Check that the connection works by issuing a command to Minos:
   Execute `Zeus.Cmd Minos.ShowModules`.

[1] In Greek Mythology, Minos is one of the sons of Zeus, the king of all gods. No other host name would be more appropriate in view of the fact that the host (Zeus) can send commands to the client (Minos).

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 `/dev/ttyUSB0` (or some port different from 0). Using a terminal application such as minicom or putty, try to connect to the client board: when resetting the board (power off/on), you should see some text on the terminal. 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 or add the current user to the group associated with `/dev/ttyUSB0`

Tasks

(a) Create a new Oberon module called `Hello.Mos` 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 `Out.String("...")` to print a string. Use `Out.Ln()` for a newline character.

Compile the module with

```
Compiler.Compile -p=Minos Hello.Mos
```

Execute your code using `Zeus.Cmd Hello.World`

(b) Write a little program to blink the onboard LED. Consult module shared/assignments/assignment2/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.

Tasks

Implement (complete) 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 module 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 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 by calling the (already prepared) command

```
Minos.UnloadModule <moduleName>
```

in module Minos.Mos. .

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 .