Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

THE A2 KERNEL AND ACTIVE OBERON LANGUAGE AT THE COMMAND LINE

FELIX FRIEDRICH

This is a brief description of the command line version of the A2 Kernel. In its minimal version it only contains the bare minimum to run the A2 kernel and to dynamically load and execute Active Oberon modules from a Windows or Linux command line.

1. Executables

The files `oberon.exe` (Windows PE File) or `oberon` (Linux ELF file) are executables that can be run as terminal application in Windows or Linux. During start of the executable the Active Oberon runtime system is initialized, memory management and scheduling is setup and the module loader is becoming effective. From here on the system behaves like a native A2 kernel, but embedded in and interacting with the surrounding Windows/Linux system.

The last module that is becoming active during boot time is the command shell (module `StdIOShell.Mod`). Parameters that are provided with the call of the executable, such as in

```
oberon Compiler.Compile Test.Mod
```

are interpreted as an Oberon command and are instantly executed by the kernel. This can imply module loading, either from prelinked modules in the executable image or from disk. Once the command is executed, the just started system shuts down again.

If `oberon` is called without parameters, the command line shell expects Oberon commands to be entered at the standard input. The shell can be exited using `exit` or by calling a terminating command such as `SystemTools.PowerDown`.

2. Built-in Functionality

The absolute minimal kernel provides functionality to load and execute modules. However, since the command line tool executables should be self-contained up to a certain degree, we additionally pre-linked the compiler, the static linker and Oberon interpreter. Moreover, we added shortcuts for compiling and linking, such that compiling a module for the Win32 Kernel can be done using

```
oberon compile -p=Win32G Test.Mod
```

Linking modules for, say a Raspberry Pi platform, can be done using

```
oberon link -p=RPI MyFiles
```

3. Implementation

The command line tool provides the minimal A2 kernel linked together with the module `StdIOShell`. StdIOShell redirects the standard input, standard error and standard output to the context of an A2 `Shell.Shell`. The bodies of all modules up to `StdIOShell` are executed during boot of the kernel. All other modules linked on top of `StdIOShell` are only initialized and added to the module list on demand. [1]

The minimal setup requires to compile and link the following modules:

In Windows:

```
StaticLinker.Link --fileFormat=PE32CUI --fileName=oberon.exe
    --extension=GofW --displacement=401000H
Runtime Trace Kernel32 Machine Heaps Modules Objects Kernel KernelLog
    Streams Commands Files WinFS Clock Dates Reals Strings Diagnostics
    BitSets StringPool ObjectFile GenericLinker Reflection GenericLoader
    Pipes Shell StdIOShell
```

In Linux:

```
StaticLinker.Link -p=Linux32G
Runtime Trace Glue Unix Machine Heaps Modules Objects Kernel KernelLog
    Streams Commands Files UnixFiles Clock Dates Reals Strings
    Diagnostics BitSets StringPool ObjectFile GenericLinker Reflection
    GenericLoader Pipes Shell StdIOShell
```

On top, any application code can be added that should be contained in the kernel. We added the Compiler, Linker, Interpreter and SystemTools.

```
Options Locks Debugging
StaticLinker FoxBasic FoxProgTools FoxScanner FoxSyntaxTree FoxGlobag
    FoxActiveCells FoxHardware FoxFormats FoxPrintout FoxParser
    FoxSemanticChecker FoxBackend FoxSections FoxFrontend Compiler
    FoxOberonFrontend FoxFingerPrinter FoxInterfaceComparison
    FoxTextualSymbolFile FoxBinarySymbolFile FoxBinaryCode
    FoxIntermediateCode FoxIntermediateBackend FoxCodeGenerators
    FoxBinaryObjectFile FoxGenericObjectFile FoxAMD64InstructionSet
    FoxAMD64Assembler FoxAMDBackend FoxAssembler FoxIntermediateAssembler
    FoxDisassembler FoxARMInstructionSet FoxARMAssembler FoxARMBackend
    FoxMinosObjectFile FoxIntermediateParser FoxIntermediateObjectFile
    FoxIntermediateLinker FoxTRMInstructionSet FoxTRMAssembler
    FoxTRMBackend FoxInterpreterBackend FoxTranspilerBackend
    FoxDocumentationScanner FoxDocumentationTree FoxDocumentationPrinter
    FoxDocumentationHtml FoxDocumentationParser FoxDocumentationBackend
DynamicStrings XMLObjects XML XMLScanner UTF8Strings XMLParser
    PersistentObjects FoxInterpreterSymbols FoxInterpreter
    InterpreterShell
CRC SystemVersion ProcessInfo0 ProcessInfo Plugins SystemTools
```

[1] This is a general feature of how we link modules using the generic object file format in order to be able to put a large number of modules into a monolithic block without being forced to initialize every module in the link order.