Shared libraries for the seL4 Kernel

Andreas Werner

06.08.2015

Outline

1. Instruction
2. Background
3. Analysis Shared Libraries
4. Shared libraries in seL4
5. Conclusions

Instruction

> *When a CERTAIN task is to preformed at several different place in a program, it is usually undesirable to repeat the codding in each place. To avoid this situation, the coding (called a subroutine) can be put into one place only, and a few extra instruction can be added to restart the outer program properly after the subroutine is finished. Transfer of control between subroutines and main programs is called subroutine linkage.*
>
> — Donald E. Knuth *The Art of Computer Programming*

Shared Libraries

Two Types of libraries exists:

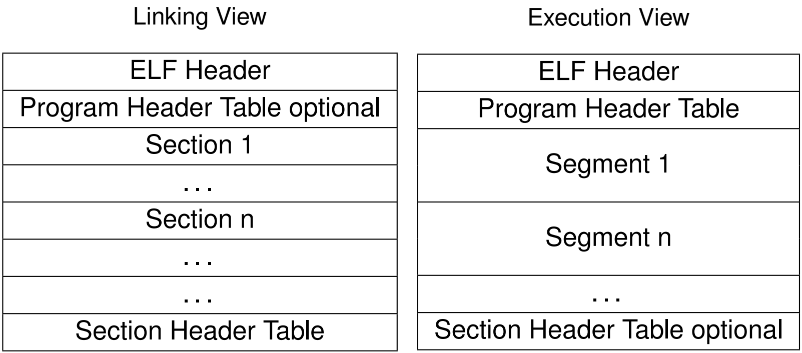
- Static libraries: Linked at Compile Time (also known as AR Archives(.a Files))
- Shared Libraries: Linked by dynamic linker at runtime(also known as Shared Objects(.so Files) or dynamic linked libraries (.dll))

seL4 Kernel has actually no Shared Libraries

Executable and Linking Format (ELF)

- Developed by Tool Interface Standard Committee at 1993
- Defined in Tool Interface Standard (TIS) Executable and Linking Format (ELF) Specification
- Goal: Standard for Executable and Library
- GCC compiler standard output

Structure of a ELF file



Linking View

- ELF Header
- Program Header Table optional
- Section 1
- . . .
- Section n
- . . .
- . . .
- Section Header Table

Execution View

- ELF Header
- Program Header Table
- Segment 1
- Segment n
- . . .
- Section Header Table optional

- Relocatable Files(.o Files): Object Files for Linking
- Program Files: Executable Files or Shared Librarys

ELF Header + Program Header Table

ELF Header contains:

- Type of File: Relocatable, Executable, Shared object file, . . .
- Machine type: x86, ARM, . . .
- Version
- Entry: Entrypoint for executable files
- Offsets of the Program Header Table and Section Header Table
- Size of ELF File

Program Header Table contains:

- Load Instructions
- Interpreter String
- Dynamic linking information
- Debug Information

Shared library

- Special Executable Files
- Global Offset Table(GOT) Section: contains the absolute addresses of the symbols
- procedure linkage table(PLT) Section: contain small routine to call the symbol

Dynamic Linker

- Interpreter Program called(/ lib /ld.so or / lib /ld–linux.so.2) instead of the original program
- Load executable memory segments.
- Load or Map Shared Library.
- Perform relocation(witting GOT entry) of executable.
- Start application.

seL4

- open source end-to-end proof of implementation correctness and security enforcement microkernel.
- developed by National Information Communications Technology Australia (NICTA) and General Dynamics Mission Systems
- based on L4 specification by Jochen Liedtke
- kernel implements
  - scheduling
  - minimal virtual memory mangement
  - interrupt handling
  - inter - process - communication
  - capabilities based right management

Playground

- one Version Static Linked(Standart)
- one Version with Shared Libraries(not working correctly)
- simple root task
  - Memory Management
  - Task Creation
  - IPC Communication
- simple application
  - Memory Management
  - IPC Communication
- Test on selfmade ARM Cortex -A5 with VF610 and on Phytec phyCORE®-Vybrid

Statically Linked

File	Library	Size
crtstuff.o	libc	80
common.o	libsel4platsupport	1272
__libc_start_main.o	libc	508
assert.o	libc	64
exit.o	libc	98
...	...	...
main.o	Application	1004
offset + padding		32772
Sum		121980

Dynamically Linked

File	Library	Size
crtstuff.o	libc	80
cpio.o	libcpio	896
main.o	Application	1004
Data	Data	4
ROData	Data	280
dynsym		509
dynstr		614
rel.plt		104
plt		176
dynamic		264
got		64
offset + padding		33585
	Sum	37476

Memory Consume of Shared Libraries

Library	Size
libc.so	599,36k
libsel4allocman.so	73,60k
libsel4platsupport.so	39,90k
libsel4.so	32,69k
libsel4muslcsys.so	85,20k
libsel4allocman.so	73,60k
libsel4simple.so	35,89k
libelf.so	39,58k
libplatsupport.so	52,07k
libsel4vspace.so	34,35k
libutils.so	35,05k
Sum	1108,23k

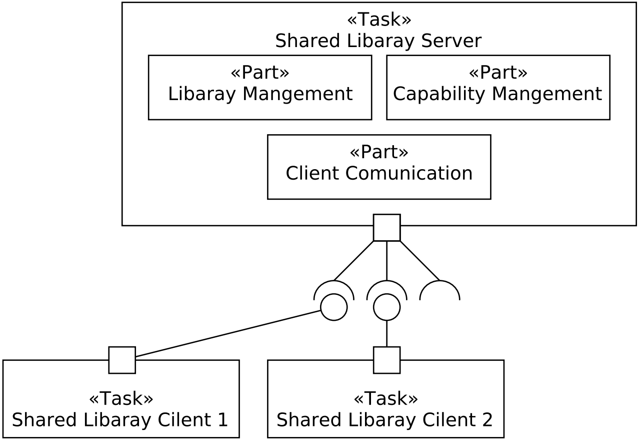
Approaches for Dynamic Loader

Tow Approaches for Dynamic Loader:

- rewrite *libmuslcs* Dynamic Loader (actual Linux Loader)
- integrate in *libsel4util*

Use second Approaches for Architecture

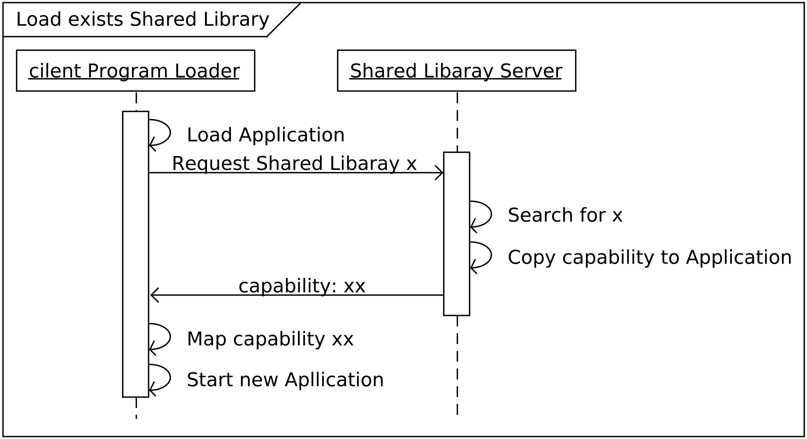
Architecture



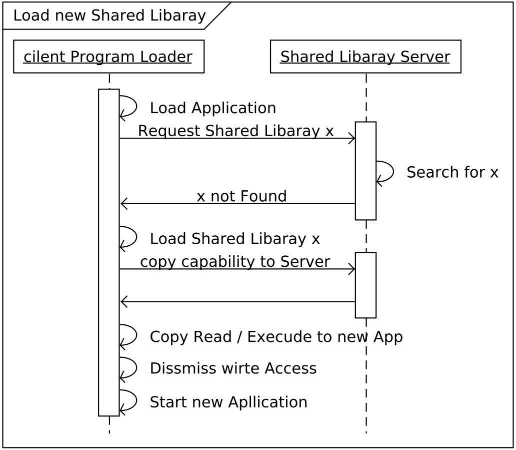
Client Requests

- get capability: Get capability to map a specific Shared Library
- add a new Library: Add a new Library to server
- remove Task: Remove a Task from the access to a Shared Library

Load exists Shared Library



Load new Shared Library



Possible Problems with this Architecture

- Server not started at start of root process.
- Possible Solutions:
  - root Task = Server
  - Interpreter like ELF Sysem (new Problems: no Filessystem Driver in Kernel)
  - root Task is statically linked

Conclusions

- Dynamic Linking provides big advantages but also big disadvantages
- many processes -> highly recommended to use dynamic linking
- hardware error in RAM: all Program corrupted that used library
- Tasks must highly trust in Server and loaded library

- paper does not elaborate an implementation
- implantation of a Dynamic Linker takes a lot of time
- Implementation can use the code of libmuslc or the GNU libc for some inspiration
- Easter to implement with used of libelf