What are and How to find a program's unused DSOs

Ravi Sankar Guntur
Contact: ravi.g@samsung.com
Samsung India
Agenda

✔ Background
✔ What constitutes application launch time?
✔ Known methods to improve launch times
✔ Results after applying “as-needed”
✔ Finding unused libraries
✔ Report of unused libraries
✔ Simulation results
✔ Source of unused DSOs
✔ Dependency graph of a DSO
✔ References
Background

➢ Part of System Team working on LiMO Platform in Samsung India, Bangalore.
➢ Improving application launch times is one of our objective.
➢ LiMO provides complete middleware and base applications.
What constitutes application launch time?

Time taken from touch event upto displaying application's first GUI window is its launch time.

- Application launch time can be divided in to three phases
  - Kernel
  - Dynamic Linker
  - Application Initialization code.
What constitutes application launch time?

Time taken from touch event upto displaying application first GUI window is its launch time.

➢ Application launch time can be divided in to three phases

➢ **Kernel**
  ➢ `fork()`, `execve()`
  ➢ Setting up process image

➢ **Dynamic Linker**
  ➢ Determine load dependencies
  ➢ Relocate the application and all its dependencies
  ➢ Initialize the application and dependencies in correct order

➢ **Application Initialization code.**
  ➢ Common stuff like, `g_type_init()`, `dbus_init()` etc
  ➢ App specific init stuff like default GUI windows
What constitutes application launch time?

Time taken from touch event upto displaying application first GUI window is its launch time.

➢ Application launch time can be divided into three phases

➢ Kernel
  ➢ fork(), execve()
  ➢ Setting up process image

➢ Dynamic Linker
  ➢ Determine load dependencies
  ➢ Relocate the application and all its dependencies
  ➢ Initialize the application and dependencies in correct order

➢ Application Initialization code.
  ➢ Common stuff like, g_type_init(), dbus_init() etc
  ➢ App specific init stuff like default GUI windows
Loading Dependencies

```
/mnt/nfs/git/tools/trimlib # strace -T ./foo
execve("./foo", ["./foo"], [/* 38 vars */) = 0 <0.001836>
brk(0)
    = 0x11000 <0.000053>
uname({sys="Linux", node="H2_SDK", ...}) = 0 <0.000054>
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory) <0.000084>
mmapping: NULL, 0x102, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, 3, 0) = 0x4001d000 <0.000050>
open("/usr/lib/libsys-assert.so", O_RDONLY) = 3 <0.000086>
read(3, ", 0x157f1111\00\00\00\000\000\0344\0\040\0\0\0", 512) = 512 <0.000063>
stat64(3, {st_mode=01644, st_size=15876, ...}) = 0 <0.000073>
mmapped: 48364, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x40026000 <0.000064>
protect(0x40026000, 28672, PROT_NONE) = 0 <0.000073>
mmapped: 0x40031000, 496, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x3) = 0x40031000 <0.000077>
close(3) = 0 <0.000047>
open("/etc/ld.so.preload", O_RDONLY) = 3 <0.000084>
read(3, ", 0x157f1111\00\00\00\000\000\0344\0\040\0\0\0", 512) = 512 <0.000063>
stat64(3, {st_mode=01644, st_size=60027, ...}) = 0 <0.000055>
mmapped: 0x40026000, 496, PROT_READ, MAP_PRIVATE, 3, 0) = 0x40026000 <0.000063>
close(3) = 0 <0.000046>
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory) <0.000063>
open("/usr/lib/libz.so.1", O_RDONLY) = 3 <0.000078>
read(3, ", 0x157f1111\00\00\00\000\000\0344\0\040\0\0\0", 512) = 512 <0.000057>
stat64(3, {st_mode=01644, st_size=86936, ...}) = 0 <0.000049>
mmapped: 118288, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x40041000 <0.000064>
protect(0x40041000, 28672, PROT_NONE) = 0 <0.000072>
mmapped: 0x4005d000, 496, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x14) = 0x4005d000 <0.000075>
close(3) = 0 <0.000046>
access("/lib/libgcc_s.so.1", O_RDONLY) = 3 <0.000072>
read(3, ", 0x157f1111\00\00\00\000\000\0344\0\040\0\0\0", 512) = 512 <0.000056>
mmapped: 4096, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x4001f000 <0.000056>
stat64(3, {st_mode=01644, st_size=422884, ...}) = 0 <0.000049>
mmapped: 74272, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x4005e000 <0.000064>
protect(0x4005e000, 32768, PROT_NONE) = 0 <0.000069>
mmapped: 0x40078000, 496, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0xa) = 0x40078000 <0.000075>
close(3) = 0 <0.000047>
access("/etc/ld.so.nohwcap", F_OK) = -1 ENOENT (No such file or directory) <0.000063>
open("/lib/libc.so.6", O_RDONLY) = 3 <0.000079>
read(3, ", 0x157f1111\00\00\00\000\000\0344\0\040\0\0\0", 512) = 512 <0.000055>
stat64(3, {st_mode=01644, st_size=1189428, ...}) = 0 <0.000048>
mmapped: 1225988, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x40071000 <0.000068>
protect(0x40071000, 32768, PROT_NONE) = 0 <0.000072>
mmapped: 0x40197000, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x11e) = 0x40197000 <0.000083>
mmapped: 0x4019a000, 9476, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x4019a000 <0.000072>
```
Loading Dependencies.. (cont)

➢ Six to eight expensive calls like open(), read(), mmap(), stat(), close() per DSO

➢ Typical GUI applications have large set of DSO dependencies.

➢ The more the number of participating Shared Libraries the more the time spent by dynamic linker to load those using open(), read(), mmap2(), close() calls.
**What constitutes application launch time?**

Time taken from touch event up to displaying application first GUI window is its launch time.

➢ Application launch time can be divided into three phases

➢ **Kernel**
  ➢ `fork()`, `execve()`
  ➢ Setting up process image

➢ **Dynamic Linker**
  ➢ Determine load dependencies
  ➢ **Relocate the application and all its dependencies**
  ➢ Initialize the application and dependencies in correct order

➢ Application Initialization code.
  ➢ Common stuff like, `g_type_init()`, `dbus_init()` etc
  ➢ App specific init stuff like default GUI windows
Relocation process
What constitutes application launch times?

Time taken to display default GUI window for every application is its launch time.

- Application launch time can be divided into three phases

  - **Kernel**
    - `fork()`, `execve()`
    - Setting up process image

  - **Dynamic Linker**
    - Determine load dependencies
    - Relocate the application and all its dependencies
    - **Initialize the application and dependencies in correct order**

  - Application Initialization code.
    - Common stuff like, `g_type_init()`, `dbus_init()` etc
    - App specific init stuff like default GUI windows
DSO constructors and destructors

Author of a DSO can write DSO initialization and deinitialization code.

➢ These init and deinit code can use symbols from other DSOs as well.
➢ Sorting of these init and deinit functions is a time consuming step.
What constitutes application launch times?

Time taken to display default GUI window for every application is its launch time.

➢ Application launch time can be divided into three phases

➢ Kernel
  ➢ fork(), execve()
  ➢ Setting up process image

➢ Dynamic Linker
  ➢ Determine load dependencies
  ➢ Relocate the application and all its dependencies
  ➢ Initialize the application and dependencies in correct order

➢ Application Initialization code.
  ➢ Common stuff like, g_type_init(), dbus_init() etc
  ➢ App specific init stuff like default GUI windows
**Application launch phases - summary**

- Application launch time can be divided into three phases

- **Kernel**
  - `fork()`, `execve()`
  - Setting up process image

- **Dynamic Linker** - The more DSOs, the more higher startup time
  - Determine load dependencies
  - Relocate the application and all its dependencies
  - Initialize the application and dependencies in correct order

- **Application Initialization code.** - Minimum required init code
  - Common stuff like, `g_type_init()`, `dbus_init()` etc
  - App specific init stuff like default GUI windows
Application launch time breakup (ms)

Breakup of startup time for dialer application
Few known methods to improve app launch time

➢ Readahead: File pre-fetching method

➢ Prelinking: Program which modifies ELF files, so that the time which dynamic linker needs for their relocation at startup significantly decreases

➢ Preloading: Reuse the relocation information for subsequent app launch.

➢ All the above techniques assume that the application is linked with the right set of shared libraries and hence no unwanted libraries are loaded by them at runtime.

➢ As-needed: flag to force the linker to link in the produced binary only the libraries containing symbols actually used by the binary itself.
GNU Link editor option "--as-needed"

➢ link only the libraries containing symbols actually used by the binary itself.

➢ LDFLAGS="-Wl,--as-needed"

➢ Incorrect way of using "--as-needed"

➢ $ gcc -Wl,--as-needed -lm someunit1.o someunit2.o -o program

➢ in this case libm is considered before the object files and discarded independently from the content of the two.

➢ Correct way of using "--as-needed"

➢ $ gcc -Wl,--as-needed someunit1.o someunit2.o -lm -o program

➢ "--as-needed" can not solve the problem introduced by bad library design choice.
Improvement we got with “as-needed”

- Calendar
- File manager
- Life drive
- Music player
- Settings
- Boottime

Imp %
Finding unused dependencies?

➢ “ldd -u < ELF >” gives list of direct unused dependencies of an ELF file.

➢ Interested in finding how many shared libraries are actually used by each application out of total loaded at run time.
Auditing API for the GNU dynamic linker

➢ Auditing API allows an application to be notified when various dynamic linking events occur.

➢ Create a shared library that implements a standard set of function names.

➢ Set the environment variable LD_AUDIT to a list of shared libraries that implements event callbacks.
Few API of the GNU dynamic linker

unsigned int la_objopen(struct link_map *map, Lmid_t lmid, uintptr_t *cookie);
   - When a new shared object is loaded.

unsigned int la_objclose(uintptr_t *cookie);
   - After any finalization code for the object has been executed, before the object is unloaded.

uintptr_t la_symbind32(Elf32_Sym *sym, unsigned int ndx, uintptr_t *refcook, uintptr_t *defcook, unsigned int *flags, const char *symname);
   - When a symbol binding occurs
# Audit Events

<table>
<thead>
<tr>
<th>LD_AUDIT Event</th>
<th>Callback Action – Data update</th>
</tr>
</thead>
<tbody>
<tr>
<td>objopen</td>
<td>Library loaded</td>
</tr>
<tr>
<td>objcose</td>
<td>Library unloaded</td>
</tr>
<tr>
<td>symbind</td>
<td>Library used</td>
</tr>
<tr>
<td>preinit</td>
<td>Get timing info (diff of time from constructor execution to preinit execution)</td>
</tr>
</tbody>
</table>
How to use libaudit.so

1. Set LD_AUDIT
2. Launch program
3. Run through all use cases
4. Exit program
5. Report
Sample report for “dialer” application

<table>
<thead>
<tr>
<th>Program</th>
<th>PID</th>
<th>Used</th>
<th>RefCount</th>
<th>Data Pages</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>3</td>
<td>1</td>
<td>/lib/ld-linux.so.3</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>No</td>
<td>0</td>
<td>1</td>
<td>/usr/lib/libsys-assert.so</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>1</td>
<td>1</td>
<td>/usr/lib/libappcore-efl.so.1</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>5</td>
<td>1</td>
<td>/usr/lib/libappcore-common.so.1</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>5</td>
<td>1</td>
<td>/usr/lib/libaul.so.0</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>18</td>
<td>1</td>
<td>/usr/lib/libdlog.so.0</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>417</td>
<td>18</td>
<td>/usr/lib/libeva.so.1</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>90</td>
<td>2</td>
<td>/usr/lib/libedje.so.1</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>466</td>
<td>3</td>
<td>/usr/lib/libelementary-ver-pre-svn-07.so.0</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>21</td>
<td>1</td>
<td>/usr/lib/libui-gadget.so.0.1.0</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>30</td>
<td>1</td>
<td>/usr/lib/libbundle.so.0</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>20</td>
<td>1</td>
<td>/usr/lib/libvconf.so.0</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>138</td>
<td>3</td>
<td>/lib/libpthread.so.0</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>841</td>
<td>5</td>
<td>/lib/libc.so.6</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>96</td>
<td>15</td>
<td>/usr/lib/libecore.so.1</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>66</td>
<td>2</td>
<td>/usr/lib/libecore_x.so.1</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>3</td>
<td>1</td>
<td>/usr/lib/libecore_input.so.1</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>15</td>
<td>1</td>
<td>/lib/libdl.so.2</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>3</td>
<td>1</td>
<td>/usr/lib/libsysman.so.0</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>36</td>
<td>1</td>
<td>/usr/lib/libobjc-2.0.so.0</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>No</td>
<td>0</td>
<td>3</td>
<td>/usr/lib/libsensor.so</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>136</td>
<td>1</td>
<td>/usr/lib/libglib-2.0.so.0</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>3</td>
<td>1</td>
<td>/usr/lib/liblua.so.0</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>193</td>
<td>4</td>
<td>/usr/lib/libX11.so.6</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>1</td>
<td>1</td>
<td>/usr/lib/libscreen-engine.so.0</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>19</td>
<td>1</td>
<td>/usr/lib/libdbus-glib-1.so.2</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>145</td>
<td>1</td>
<td>/usr/lib/libdbus-1.so.3</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>72</td>
<td>2</td>
<td>/usr/lib/libsqlite3.so.0</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>No</td>
<td>0</td>
<td>1</td>
<td>/usr/lib/libxdg_mime.so.1</td>
</tr>
<tr>
<td>dialer</td>
<td>2102</td>
<td>Yes</td>
<td>398</td>
<td>1</td>
<td>/usr/lib/libeina.so.1</td>
</tr>
</tbody>
</table>

Library coverage summary for dialer: 2102
Total libraries 119, Unused 28, Unused % 23.529411
Time spent in ld from AUDIT init to main() 91 milliseconds
Total Data Segment size of unused Libraries is 60 Pages of each 4K (Total 245760 Bytes)
Source code of the implementation

git: git://gitorious.org/slp-siso/trimlib.git
**Known issues with LD_AUDIT**

- LD_AUDIT does not work with Shared Libraries with no code in them.
- Example ICU-4.0 “libicudata.so”
- Error: “no PLTREL found in object /usr/lib/libicudata.so.40”
- Recompile after patching libicudata by sed'ing -nostdlib etc away sed -i -- "s/-nodefaultlibs -nostdlib/" config/mh-linux
Library coverage summary for few apps

Findings of libaudit.so utility for few sample set of applications
Summary of the findings

✔ Average percentage of unused libraries is around 25%

✔ Memory and launch time could be saved if unused libraries are not loaded at all.
Test results

With unused

without unused

Total Libs
Unused
Time
Dependency graph of libaudit.so.0.5
Dependency graph of “dialer” app
Source of unused DSOs
Learnings..

- “As Needed” linker option is a must for Embedded Devices
- 25% unused DSOs, even after applying “as-needed” to entire Platform.
- Unused DSOs should be avoided for better application launch time.
- Avoiding unused DSOs is not as simple as “rebuilding”
- Code changes may be required in few middleware DSOs
Reference


● How to Write Shared Libraries by Ulrich Drepper -
  http://www.akkadia.org/drepper/dsohowto.pdf

● Libelf - http://www.mr511.de/software/english.html

● How to use GNU Linker flag “--as-needed”-


● Computer Science from the Bottom Up -

● Graphviz documentaiton - http://www.graphviz.org/

● Manual page for “rt-ld-audit”

● Manual page for “ld.so”

● Manual page for “ldd”

● LiMO foundation web site - http://www.limofoundation.org
Questions ?
Backup slides
## Library usage statistics in SLP

<table>
<thead>
<tr>
<th>Command</th>
<th>Usage Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>161</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>156</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>138</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>135</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>122</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>121</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>120</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>119</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>113</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>103</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>102</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>102</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>99</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>95</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>94</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>93</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>85</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>82</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>81</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>79</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>53</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>52</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>45</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>44</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>43</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>42</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>40</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>35</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>35</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>34</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>33</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>32</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>28</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>27</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>26</td>
</tr>
<tr>
<td>/mnt/nfs/git/tools/trimlib # ./stats.sh</td>
<td>25</td>
</tr>
</tbody>
</table>

13/04/11  Samsung Electronics  37
Dependency graph of “dialer app”

```
Analyzing dependencies of: /opt/apps/deb.com.samsung.dialer/bin/dialer
Creating output as:    dialer.png

/opt/apps/deb.com.samsung.dialer/bin/dialer ===> libappcore-efl.so.1
/usr/lib/libappcore-efl.so.1 ===> libelementary-ver-pre-svn-07.so.0
/usr/lib/libelementary-ver-pre-svn-07.so.0 ===> libehal.so.1
/usr/lib/libehal.so.1 ===> libedbus.so.1
/usr/lib/libedbus.so.1 ===> libcore.so.1
/usr/lib/libcore.so.1 ===> libeina.so.1
/usr/lib/libeina.so.1 ===> librt.so.1
/lib/librt.so.1 ===> libc.so.6
/lib/libc.so.6 ===> ld-linux.so.3
/lib/librt.so.1 ===> libpthread.so.0
/lib/libpthread.so.0 ===> libc.so.6
/lib/libpthread.so.0 ===> ld-linux.so.3
/lib/librt.so.1 ===> ld-linux.so.3
/usr/lib/libeina.so.1 ===> libm.so.6
/lib/libm.so.6 ===> ld-linux.so.3
/lib/libm.so.6 ===> libc.so.6
/usr/lib/libeina.so.1 ===> libdl.so.2
/lib/libdl.so.2 ===> libc.so.6
/lib/libdl.so.2 ===> ld-linux.so.3
/usr/lib/libeina.so.1 ===> libgcc_s.so.1
/lib/libgcc_s.so.1 ===> libc.so.6
/usr/lib/libeina.so.1 ===> libpthread.so.0
/usr/lib/libeina.so.1 ===> libc.so.6
/usr/lib/libcore.so.1 ===> libgnutls-2.0.so.0
/usr/lib/libgnutls-2.0.so.0 ===> libgcc_s.so.1
/usr/lib/libgnutls-2.0.so.0 ===> libc.so.6
/usr/lib/libcore.so.1 ===> librt.so.1
/usr/lib/libcore.so.1 ===> libm.so.6
/usr/lib/libcore.so.1 ===> libeina.so.1
/usr/lib/libcore.so.1 ===> libgcc_s.so.1
/usr/lib/libcore.so.1 ===> libc.so.6
```

13/04/11
Who should be concerned?

➢ Embedded Developers who run their applications on Low-end processors and have limited main memory

➢ When you are told to optimize the application launch time. And you notice that most of the startup time is spend even before your “main” is called.

➢ As a System Engineer I am interested in each Process “Library Coverage”.
What are Shared Object Dependencies?

```
$ gcc -g foo.c /usr/lib/libz.a -o foo
```

➢ Link editor extracts archive library members and copies them into the output object file.

➢ These statically linked services are available during execution without involving the dynamic linker.

➢ Executables or Shared library can depend on another Shared Library for services.

```
$ gcc -g foo.c -lz -o foo
```

➢ The Link editor inserts supplied Shared library names as “NEEDED” list into the Object File.

➢ The dynamic linker loads the “NEEDED” shared object files to the process image for execution.

➢ Thus ELF standard defines a way for executable and shared object files to describe their specific dependencies in section called “dynamic section”.
What is Direct and Indirect dependency?

➢ When a Shared Object or Executable is linked against another Shared Object then we call it as “Direct dependency”

➢ Entry is made for each dependent Shared Object in the Object's Dynamic Section itself.

➢ Each Shared Object mentioned in the Dynamic Section of a given Object can further bring its own set of Dependency list. In this case we call it as indirect dependency list.

➢ $gcc -g -shared -fPIC foo.c -o libfoo.so -lz
➢ $gcc -g bar.c -o bar -lfoo

➢ foo is directly dependent on libbar.so and indirectly depends on libz.so.
**How to find Direct dependency?**

$ readelf -d <ELF File>

<table>
<thead>
<tr>
<th>Tag</th>
<th>Type</th>
<th>Name/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000001</td>
<td>(NEEDED)</td>
<td>Shared library: [libc.so.6]</td>
</tr>
<tr>
<td>0x000000c</td>
<td>(INIT)</td>
<td>0x80482bc</td>
</tr>
<tr>
<td>0x000000d</td>
<td>(FINI)</td>
<td>0x804849c</td>
</tr>
<tr>
<td>0x0000004</td>
<td>(HASH)</td>
<td>0x804818c</td>
</tr>
</tbody>
</table>

$ gcc -g foo.c /usr/lib/libz.a -o foo

<table>
<thead>
<tr>
<th>Tag</th>
<th>Type</th>
<th>Name/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000001</td>
<td>(NEEDED)</td>
<td>Shared library: [libc.so.6]</td>
</tr>
<tr>
<td>0x00000001</td>
<td>(NEEDED)</td>
<td>Shared library: [libz.so.1]</td>
</tr>
<tr>
<td>0x000000c</td>
<td>(INIT)</td>
<td>0x80482f0</td>
</tr>
<tr>
<td>0x000000d</td>
<td>(FINI)</td>
<td>0x80484cc</td>
</tr>
<tr>
<td>0x0000004</td>
<td>(HASH)</td>
<td>0x804818c</td>
</tr>
</tbody>
</table>

$ gcc -g foo.c -lz -o foo

Entry is made for libz.so
How to find Direct & Indirect dependency?

$ ldd <ELF file>

```
readelf -d /usr/lib/libdbus-glib-1.so.2.1.0
```

```
Dynamic section at offset 0x1be68 contains 28 entries:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Type</th>
<th>Name/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000001</td>
<td>(NEEDED)</td>
<td>Shared library: [libdbus-1.so.3]</td>
</tr>
<tr>
<td>0x00000001</td>
<td>(NEEDED)</td>
<td>Shared library: [libpthread.so.0]</td>
</tr>
<tr>
<td>0x00000001</td>
<td>(NEEDED)</td>
<td>Shared library: [libgobject-2.0.so.0]</td>
</tr>
<tr>
<td>0x00000001</td>
<td>(NEEDED)</td>
<td>Shared library: [libgthread-2.0.so.0]</td>
</tr>
<tr>
<td>0x00000001</td>
<td>(NEEDED)</td>
<td>Shared library: [librt.so.1]</td>
</tr>
<tr>
<td>0x00000001</td>
<td>(NEEDED)</td>
<td>Shared library: [libglib-2.0.so.0]</td>
</tr>
<tr>
<td>0x00000001</td>
<td>(NEEDED)</td>
<td>Shared library: [libc.so.6]</td>
</tr>
<tr>
<td>0x0000000e</td>
<td>(SONAME)</td>
<td>Library soname: [libdbus-glib-1.so.2]</td>
</tr>
<tr>
<td>0x0000000c</td>
<td>(INIT)</td>
<td></td>
</tr>
</tbody>
</table>
```

```
ldd /usr/lib/libdbus-glib-1.so.2.1.0
```

```
libdbus-1.so.3  =>  /lib/libdbus-1.so.3  (0x00a40000)
libpthread.so.0 =>  /lib/tls/i686/cmov/libpthread.so.0  (0x00f72000)
libgobject-2.0.so.0 =>  /usr/lib/libgobject-2.0.so.0  (0x00206000)
libgthread-2.0.so.0 =>  /usr/lib/libgthread-2.0.so.0  (0x00110000)
librt.so.1  =>  /lib/tls/i686/cmov/librt.so.1  (0x00fb0000)
libglib-2.0.so.0 =>  /lib/libglib-2.0.so.0  (0x00a88000)
libc.so.6  =>  /lib/tls/i686/cmov/libc.so.6  (0x00245000)
/lib/ld-linux.so.2  =>  /lib/ld-linux.so.2  (0x00e90000)
libpcre.so.3  =>  /lib/libpcre.so.3  (0x00bbd000)
```

Brought in by libgobject-2.0.so.0
How Dependency information is supplied?

➢ Typical GUI applications have large set of shared object dependencies.

➢ pkg-config is a helper tool used when compiling applications and libraries.

➢ A package developer exports correct compiler options in a “.pc” file so that user can use it to expand to proper options rather than hard-coding.

➢ For instance glib exports compiler and linker options in its package config file located at “/usr/lib/pkgconfig/glib-2.0.pc”

➢ $ gcc `pkg-config --cflags --libs glib-2.0` gtk.c -fPIC -o gtk

```bash
ravi@ravi-desktop:~/sbox/gits/tools/trimlib$ cat /usr/lib/pkgconfig/glib-2.0.pc
prefix=/usr
exec_prefix=${prefix}
libdir=${exec_prefix}/lib
includedir=${prefix}/include
glib genmarshal=glib-genmarshal
gobject_query=gobject-query
glib_mkenums=glib-mkenums

Name: Glib
Description: C Utility Library
Version: 2.24.1
Libs: -L${libdir} -lglib-2.0
Libs.private:
Cflags: -I${includedir}/glib-2.0 -I${libdir}/glib-2.0/include
```

```bash
ravi@ravi-desktop:~/sbox/gits/tools/trimlib$ pkg-config --cflags --libs /usr/lib/pkgconfig/glib-2.0.p
I/usr/include/glib-2.0 -I/usr/lib/glib-2.0/include -lglib-2.0
```
Summary of Relocation costs

✔ Performance of each lookup depends ~ on the length of the hash chains and the number of objects in the lookup scope.

✔ Length of Hash chains depends on number of Symbols exported – tradeoff between Speed and hash table memory overhead.

✔ Relocation is at least O(rs); where r is number of relocations and s is the number of symbols defined in all objects.

✔ The more DSOs participate or the more symbols are defined in the DSOs, the longer the symbol lookup takes.

✔ Reducing the number of loaded objects increases performance.
Finding unused direct dependencies with 'ldd'

$ ldd -u <ELF File>

➢ Will give list of direct unused dependencies of the ELF file.

➢ Checks for dependency with in a “ELF File context” but not a Process context.

ELF 1 --\(\rightarrow\) ELF 2 --\(\rightarrow\) ELF 4
\hspace{2cm} | \hspace{2cm}
\hspace{2cm} \hspace{2cm} --\(\rightarrow\) ELF 5
\hspace{2cm} --\(\rightarrow\) ELF 3

➢ Increased usage of “dlopen” in service plugins. “ldd “ does not support dlopened libraries.
Source code for the libaudit.so tool to find unused DSOs

git: git://gitorious.org/slp-siso/trimlib.git
GNU GPL license.