The missing link

Nick Todd
Kernel RPE EMEA Oracle UK

Why?

What?

ld – the link editor

- The compiler output won't execute on its own.
- Concatenates predefined files to the compiler output.
- Interprets data from input files to produce a single output file.
- For a dynamic executable/shared object:
  - Creates a program header describing the different file segments.
  - Dynamic linking information.

ld command line

```
/usr/ccs/bin/ld ...
/opt/SS12.2/solstudio12.2/prod/lib/crti.o
/opt/SS12.2/solstudio12.2/prod/lib/crt1.o
...
 hello.o -o a.out
 -lc /opt/SS12.2/solstudio12.2/prod/lib/crtn.o
```

Dynamic executable

ELF HEADER
Program header table
interp
crti.o
crt1.o
hello.o
crtn.o
Data
Dynamic
…
Section header table (optional)

Running the executable

- exec() loads the executable into the bottom of the address space.
- The 'interpreter' (run time linker) is found by inspecting the program header.
- The shared object ld.so.1 is loaded into the top of the address space below the stack and control is passed to it.

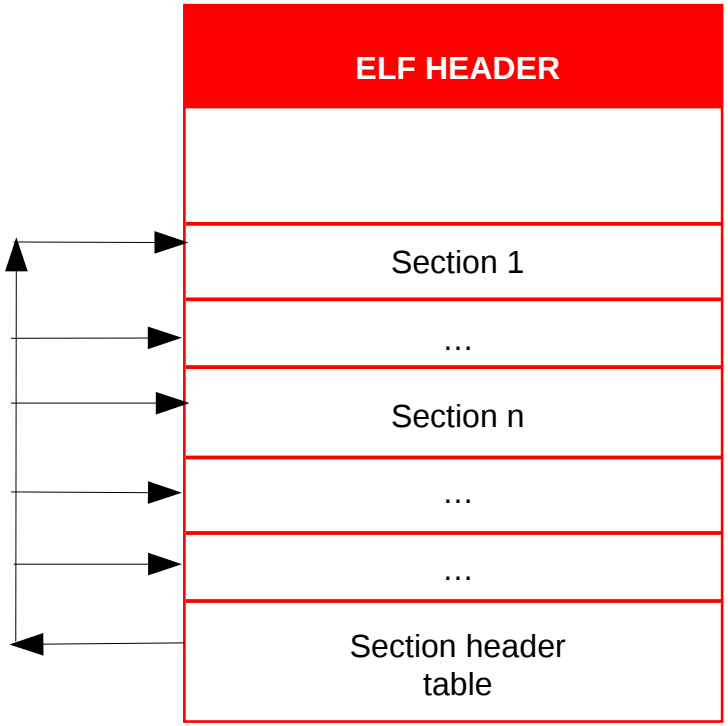
Running the executable (contd)

- The run time linker looks in the .dynamic section to find any shared objects it requires.
- Performs any relocations required by application.
- For each dependency the initialisation functions are run.
- Control is passed from ld.so.1 to _start() which calls main().
- During program execution performs lazy loading of shared objects.
- Access other objects via dlopen() and dlsym.()

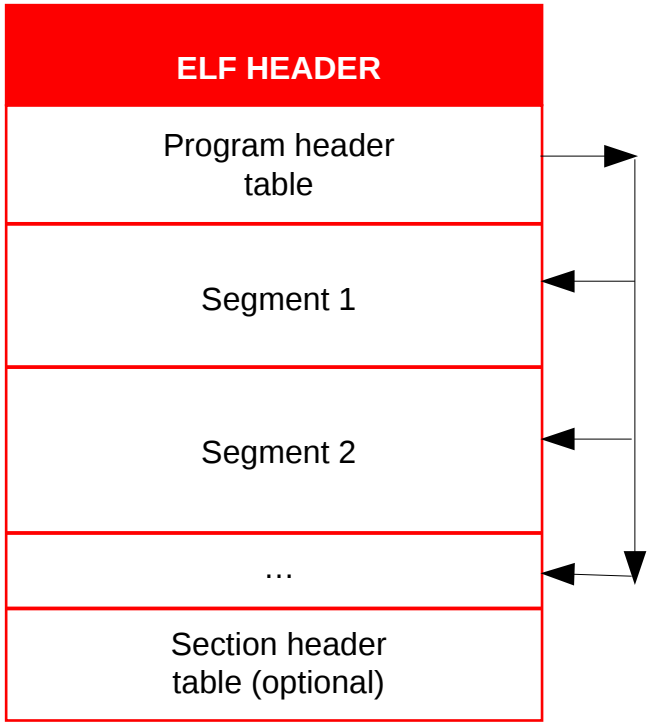
ELF files

- ELF – Executable and Linking Format
- Also known as Extensible and Linking Format
- Three types of file:
  - Dynamic executable
  - Relocatable
  - Shared object

ELF File Format



Relocatable



Executable/
shared object

elfdump(1)

- User friendly way of inspecting an ELF file.
- elfdump <ELF-file> will show everything
- Many options to control output
- dump(1) is an alternative but is better for scripts

elfdump example

```
elfdump -p /bin/ls

Program Header[0]:
    p_vaddr:      0x8050034   p_flags:    [ PF_X PF_R ]
    p_paddr:      0           p_type:     [ PT_PHDR ]
    p_filesz:     0xe0        p_memsz:    0xe0
    p_offset:     0x34        p_align:    0

Program Header[1]:
    p_vaddr:      0x8050114   p_flags:    [ PF_R ]
    p_paddr:      0           p_type:     [ PT_INTERP ]
    p_filesz:     0x11        p_memsz:    0x11
    p_offset:     0x114       p_align:    0

            …
```

elfdump example contd.

```
Program Header[3]:
    p_vaddr:     0x8050000   p_flags:    [ PF_X PF_R]
    p_paddr:     0           p_type:     [ PT_LOAD ]
    p_filesz:    0x8328      p_memsz:    0x8328
    p_offset:    0           p_align:    0x10000

Program Header[4]:
    p_vaddr:     0x8069000   p_flags:    [ PF_W PF_R ]
    p_paddr:     0           p_type:     [ PT_LOAD ]
    p_filesz:    0x89e       p_memsz:    0x17d4
    p_offset:    0x9000      p_align:    0x10000

Program Header[5]:
    p_vaddr:     0x8069158   p_flags:    [ PF_X PF_W PF_R ]
    p_paddr:     0           p_type:     [ PT_DYNAMIC ]
    p_filesz:    0x198       p_memsz:    0
    p_offset:    0x9158      p_align:    0
```

ldd

- Inspect dependencies for an executable or shared object
- Options allow you to inspect search paths, unused dependencies
- Use pldd to inspect objects in a running process or core file.

ldd example

```
$ ldd -u /usr/bin/bash
        libsocket.so.1 =>  /lib/libsocket.so.1
        libresolv.so.2 =>  /lib/libresolv.so.2
        libnsl.so.1 =>     /lib/libnsl.so.1
        libgen.so.1 =>     /lib/libgen.so.1
        libc.so.1 =>       /lib/libc.so.1
        libcurses.so.1 =>  /lib/libcurses.so.1
        libmd.so.1 =>      /lib/libmd.so.1
        libmp.so.2 =>      /lib/libmp.so.2
        libm.so.2 =>       /lib/libm.so.2

  unused object=/lib/libgen.so.1
  unused object=/lib/libcurses.so.1
  unused object=/lib/libmd.so.1
  unused object=/lib/libmp.so.2
```

Debugging

- Works on ld and ld.so.1
- 'LD_DEBUG=help ls' or 'ld -Dhelp' to get list of options.
- Use LD_DEBUG=<option1, option2,...> against an executable object
- Use either:
  - ld -D<option1, option2,...> …
  - LD_OPTIONS=-D<option1, option2,...> …
- Use LD_DEBUG_OUTPUT=file to capture large amounts of data.

Performance

- Lazy loading
- Compile shared objects as PIC
- ld -z guidance
- Direct bindings
- Collapse mutiply defined strings.
- ELF symbol hashing.
- Stub objects

What's new in s10u10

- Large backport of Solaris 11 features and bug fixes.
- Stub objects.
- ld -z guidance option.
- Improved mapfile syntax option.
- Symbol capabilities.
- Cross linking capabilities.
- elfedit
- Improved archive rescanning.
- archive files > 2GB

What's new in s10u10 contd.

- ld and ELF utilities moved from /usr/ccs/bin to /usr/bin

Linux

- Uses ELF files
- s10u10 and s11 linker recognise a number of GNU ld options
- LD_DEBUG with smaller range of options.
- ldd(1)
- readelf(1) instead of elfdump(1)

Further Reading

- Linker and Libraries Guide
  - http://download.oracle.com/docs/cd/E18752_01/html/817-1984/
- Blogs by Rod Evans and Ali Bahrami
- Linkers and Loaders
  - John R. Levine
- Airs blog
  - Ian Lance Taylor
  - http://www.airs.com/blog/archives/date/2007/08/page/2
- Expert C Programming - Deep C Secrets
  - Peter Van Der Linden
  - Chapters 5 and 6

Questions?

Answers

- Does 'ld -z guidance' provide information as you go?
  - The guidance information is displayed after the ld command completes.
- If you build on s10u10, does it still rely on previous versions of libraries and relocatable objects?
  - The updated link editor and run time linker will be able to use and manage shared and relocatable objects built in previous versions of Solaris 10.

Answers contd.

- What can elfedit do?
  - See http://blogs.oracle.com/ali/entry/introducing_elfedit_a_tool_for an overview of elfedit. The elfedit man page also contains examples of changing a run path, removing a hardware capability path and reading information from an object.
- Are there any good or bad reasons to use/not use GCC/GNU linker?
  - The Oracle compilers, link editor and runtime linker are developed with the underlying knowledge of Solaris internals so should provide the best solution for applications running on Solaris.