ELF Executable and Linkable Format

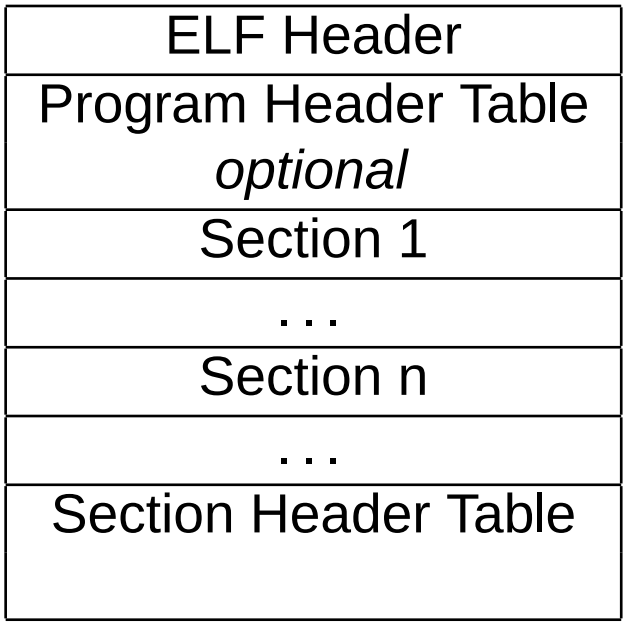
Hagen Paul Pfeifer
www.jauu.net

18. April 2005

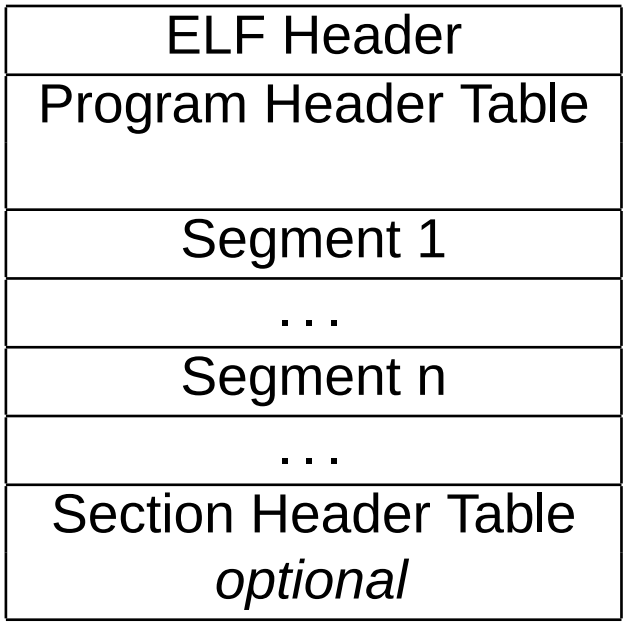
Prologue

- Binärformat
- ELF „Executable and Linkable Format“
- Entwickelt und veröffentlicht durch UNIX™ System Laboratories
- portabel (Plattformen, little ↔ big Endian . . .)
- Ziel: Reduzierung des Aufwands bei Portierung (z.B. keine festen Addressen)
- -shared als Richter für ld (DSO oder statisch)

Format



‹Representation Linking›



‹Representation Execution›

Sichtweisen

- benötigt beim Programmbau
- komplexer Aufbau
- Linking View
  - Betrachtung bei Linkvorgang
- Executation View
  - Betrachtung bei Programmausführung

Linking View

- unterteilt Datei in verschiedene Sektionen
  - Name und Typ
  - Rechte

Executation View

- relativ einfacher Aufbau
- Unterteilung in Segmente
- Beschreibt, was in Speicher eingeblendet werden muss
- Lage von Daten
- Segmente
  - Typ
  - gewünschte Speicheraddresse

- Rechte
- Größe in Datei/Speicher

Executable File

bash$ vim

- Dynamisch gelinkt? Statisch? → Programm Table (PT_INTERP)
- Laden des dymanischen Linkers
- do_mmap Userprogramm
- starten des Linkers

Object File Format

- reloctable file
  - verschiebbare Dateien, welche nach denm Linken ein auführbares Programm darstellen
- executable file
  - ausführbares Datei, welche Informationen über die Erzeugung eines Images enthält
- shared object file
  - kann von verschiedenen Programmen gleichzeitig genutzt werden (zwei mal linken)

Konzept der geteilten Bibliotheken

- Entwickler sind Sammler (spart Entwicklungszeit, reduziert Fehler)

  “lsof | wc -l“2100

- moderne OS können Code teilen => einmal physikalisch, mehrmals virtuell
- Kein neues Konzept (sk_buff, fork(),. . .)

  Jetzt kommt ELF ins Spiel

- Binärformat beschreibt Anwendungscode
- a.out und COFF als kränkelnde Vorväter

a.out

- nicht für shared libraries konzipiert
- keine Programmverlagerung zur Ladezeit möglich
- feste Ladeaddressen
- Konflikt vermeidung bei Addressvergabe (größere Dimensionierung)
- Stubs als Platzhalter für Addressen
- schnell(e | ste) Symbolauflösung

- Addressüberlappung nicht ausgeschlossen, Speicher stark fragmentiert

Statisch gegen Dynamisch | Round 1

- Statisch
  - selbstbeschreibend
  - kein externer Code/Daten notwendig
- Dynamisch
  - benutzt externe/n Code/Daten (shared libraries)
  - kleinere Programme
  - weniger Plattenspeicher
  - gewöhnlich

statisch gelinktes Programm

Programmablauf

- bekannte statische Einsprungsadresse
- Einblenden in den Speicher (do_mmap())
- Kontrolle an Programm (%EIP)

Programmausführung

- execve()
- kein simples mmap(vim);
- ELF ist Struktur pur

ELF Konstrukt(abstrakt)

- Code (ausführbar, schreibgeschützt)
- Daten (nicht ausführbar, beschreibbar)
- Daten (nicht ausführbar, beschreibbar, nicht benötigt zum Programmstart)

whereis code? Definiert in Programm Header Table!

dynamisch gelinktes Programm

Programmablauf

- Datei öffnen
- mappen der LOAD Segmente in den Speicher
- dynamischen Linker starten und fd übergeben
- Kontrolle den Programm übergeben
- d. Linker überprüft die benötigten Bibliotheken
- erhält Information in der DYNAMIC Segment

- mapped Bibliotheken in den Speicher (z.B. DT_NEEDED)
- modifiziert Programm(für Zugriff auf Bibliotheksroutinen/Daten)
- Relocation

ELF Header (/include/linux/elf.h)

```
#define EI_NIDENT       16

typedef struct elf32_hdr{
    unsigned char e_ident[EI_NIDENT];
    Elf32_Half    e_type;
    Elf32_Half    e_machine;
    Elf32_Word    e_version;
    Elf32_Addr    e_entry;  /* Entry point */
    Elf32_Off     e_phoff;
    Elf32_Off     e_shoff;
    Elf32_Word    e_flags;
    Elf32_Half    e_ehsize;
    Elf32_Half    e_phentsize;
    Elf32_Half    e_phnum;
    Elf32_Half    e_shentsize;
    Elf32_Half    e_shnum;
    Elf32_Half    e_shstrndx;
} Elf32_Ehdr;
```

Program Header Table (/include/linux/elf.h)

```
typedef struct elf32_phdr{
    Elf32_Word    p_type;
    Elf32_Off     p_offset;
    Elf32_Addr    p_vaddr;
    Elf32_Addr    p_paddr;
    Elf32_Word    p_filesz;
    Elf32_Word    p_memsz;
    Elf32_Word    p_flags;
    Elf32_Word    p_align;
} Elf32_Phdr;
```

Beispiele für Headertable Typen

- LOAD
  - Teil soll in Speicher gemappt werden
- INTERP
  - enthält String welcher den dynamischen Linker bestimmt
- DYNAMIC
  - Zeiger zu Information für Linkprozess (.dynamic Section)

Program Header Table Interna

- Segmente mit PT_LOAD in p_type sind relevant
- p_offset und p_filesize bestimmen Standpunkt
- p_vaddr und p_memsize spezifizieren virtuelle Addresse
- p_memsize > p_filesize fülle mit Nullen (bss!)
- p_flags bestimmen Modus für Speicherseite (1 = exec;2 = write;4 = read)

- PT_LOAD werden in Speicher gemappt (vom Programm)
- dynamischer Linker wird auf die gleiche Weise vorbereitet
- PT_INTERP; String unter p_offset
- gemappt in den Speicher
- auxiliary vector wird auf Stack angelegt
- %eip wird nun auf e_entry im dynamischen Linker gesetzt

Start des Dynamischen Linkers

Grundsätzliches

- Bestimmung und Laden der Abhängigkeiten
- Neuanordnung der Anwendung und ihrer Abhängigkeiten
- Initialisierung der Anwendung und Abhängigkeiten in der richtigen Reihenfolge

Neuanordung (relocation) der Symbole

Zeitaufwendigste Teil eines d. Linkers(vgl. OpenOffice)
Neuanordung für alle Symbole, welche benötigt werden (Lazy Binding)

- Bestimmung des Hash Wertes für ein Symbol
- Vergleichen des “Hash Buckets “mit eben bestimmten Hashwert
- Offset für korrospondierenden Symbol-Namen bestimmen
- Vergleichen des Symbolnamens mit neuangeordneten Namen
- Stimmt Vergleich überein, Version überprüfen

- Bei Abweichungen im Vergleich neuen Hash Bucket aus Liste vergleichen

GOT und PLT

- Global Offset Table; Procedure Linkage Table
- -fPIC erzeugt positionsunabhängigen Code
- relative Variablen Adressen werden in GOT gespeichert
- relative Funktionsaddressen werden in PLT gespeichert

PLT

- .plt section
- erlaubt Programm den Aufruf von Funktionen die während der Compilier-zeit nicht vorhanden waren (z.B. printf()).
- Ist Sammlung von Functions Stubs
- bei relocation werden Stubs auf echte Addressen gesetzt
- Stichwort “lazy bindig “

GLT

- ähnlich wie PLT, aber nicht für Funktionen, sondern Variablen (z.B. errno)

Program Section Table (/include/linux/elf.h)

```
typedef struct {
    Elf32_Word    sh_name;
    Elf32_Word    sh_type;
    Elf32_Word    sh_flags;
    Elf32_Addr    sh_addr;
    Elf32_Off     sh_offset;
    Elf32_Word    sh_size;
    Elf32_Word    sh_link;
    Elf32_Word    sh_info;
    Elf32_Word    sh_addralign;
    Elf32_Word    sh_entsize;
} Elf32_Shdr;
```

Reloctable file

- enthält .text und .data Segmente welche für weiteres Linking geeignet sind.
- $\rightarrow$ executable oder shared object file

Tools

- readelf
- gdb
- ht-editor
- objdump
- strace
- ltrace

- nm
- ldd
- lsof
- Papier und Bleistift ;-)

Literatur

- Executable and Linkable Format (ELF) Manual ftp.intel.com (pub/tis/elf11g.zip)
- http://www.linuxjournal.com/article.php?sid=1060
- Kernel Quellen (lxr Quelltextbrowser)
- Manual Pages