Executable and Linkable Format (ELF)

Why Executable Formats?

- All code in one file
  - But libraries!
- We need a way to combine files
  - Distribute as binary (“object files”)
- Linkers
- We need a way to control how our programs run
  - Memory permissions
  - Loading addresses
- Loaders
- We want PIE code! And shared libraries!
  - Dynamic linker (ld.so)

Why Executable Formats?

- Provide key metadata for running programs
  - Memory permissions
  - Loading addresses, custom interpreter, etc.
- Provide debugging assistance
  - Debug symbols
- Allow combining (linking) programs
  - Relocations
  - Function symbols

Common Executable Formats

- Executable and Linkable Format (ELF)
- Portable Executable (PE)
- Mach object file format (Mach-O)
- Organized Runtime Contents (ORC) - custom 595g file format!
  - Admittedly not so common...

How does ELF work?

- Reference: http://www.skyfree.org/linux/references/ELF_Format.pdf
  - Will be posted at http://cs595g.lockshaw.io/w20.html
- Your friend: readelf
  - readelf -S (sections)
  - readelf -l (segments) <- that's a lowercase L
  - readelf -h (headers)
  - readelf -a (everything)

ELF Headers

e_type

Name	Value	Meaning
ET_NONE	0	No file type
ET_REL	1	Relocatable file
ET_EXEC	2	Executable file
ET_DYN	3	Shared object file
ET_CORE	4	Core file
ET_LOPROC	0xff00	Processor-specific
ET_HIPROC	0xffff	Processor-specific

```
#define EI_NIDENT       16

typedef struct {
        unsigned char   e_ident[EI_NIDENT];
        Elf32_Half      e_type;
        Elf32_Half      e_machine;
        Elf32_Word      e_version;
        Elf32_Addr      e_entry;
        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;
```

e_entry Program entry point virtual address

Sections vs. Segment

- Object file != executable file
  - But they’re both ELFs
- Sections are chunks of programs we move around when linking
- Segments are how chunks of programs are loaded into memory
- No explicit mapping between sections and segments
- Both section table and segment table point into the overall contents
  - Need to maintain memory permissions
- To see mapping, run `readelf -l`

Sections

- Metadata stored in section table

`sh_flags` - permissions

```
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;
```

- `sh_type`

Name	Value
SHT_NULL	0
SHT_PROGBITS	1
SHT_SYMTAB	2
SHT_STRTAB	3
SHT_RELA	4
SHT_HASH	5
SHT_DYNAMIC	6
SHT_NOTE	7
SHT_NOBITS	8
SHT_REL	9
SHT_SHLIB	10
SHT_DYNSYM	11
SHT_LOPROC	0x70000000
SHT_HIPROC	0x7fffffff
SHT_LOUSER	0x80000000
SHT_HIUSER	0xffffffff

SHT_NOBITS - .bss

Some sections have no runtime effect: SHT_NOTE

Common Sections

Name	Type	Attributes
.bss	SHT_NOBITS	SHF_ALLOC + SHF_WRITE
.comment	SHT_PROGBITS	none
.data	SHT_PROGBITS	SHF_ALLOC + SHF_WRITE
.data1	SHT_PROGBITS	SHF_ALLOC + SHF_WRITE
.debug	SHT_PROGBITS	none
.dynamic	SHT_DYNAMIC	see below
.dynstr	SHT_STRTAB	SHF_ALLOC
.dynsym	SHT_DYNSYM	SHF_ALLOC
.fini	SHT_PROGBITS	SHF_ALLOC + SHF_EXECINSTR
.got	SHT_PROGBITS	see below
.hash	SHT_HASH	SHF_ALLOC
.init	SHT_PROGBITS	SHF_ALLOC + SHF_EXECINSTR
.interp	SHT_PROGBITS	see below
.line	SHT_PROGBITS	none
.note	SHT_NOTE	none
.plt	SHT_PROGBITS	see below
.rel*name*	SHT_REL	see below
.rela*name*	SHT_RELA	see below
.rodata	SHT_PROGBITS	SHF_ALLOC
.rodata1	SHT_PROGBITS	SHF_ALLOC
.shstrtab	SHT_STRTAB	none
.strtab	SHT_STRTAB	see below
.symtab	SHT_SYMTAB	see below
.text	SHT_PROGBITS	SHF_ALLOC + SHF_EXECINSTR

Symbols

```
typedef struct {
        Elf32_Word      st_name;
        Elf32_Addr      st_value;
        Elf32_Word      st_size;
        unsigned char   st_info;
        unsigned char   st_other;
        Elf32_Half      st_shndx;
} Elf32_Sym;
```

`st_name` - index into strtab

`st_info` - symbol type, binding

Name	Value
STT_NOTYPE	0
STT_OBJECT	1
STT_FUNC	2
STT_SECTION	3
STT_FILE	4
STT_LOPROC	13
STT_HIPROC	15

Name	Value
STB_LOCAL	0
STB_GLOBAL	1
STB_WEAK	2
STB_LOPROC	13
STB_HIPROC	15

STB_LOCAL
STB_GLOBAL
STB_WEAK

Relocations

How do we safely move sections around?

Name	Value	Field	Calculation
R_386_NONE	0	none	none
R_386_32	1	*word32*	S + A
R_386_PC32	2	*word32*	S + A - P
R_386_GOT32	3	*word32*	G + A - P
R_386_PLT32	4	*word32*	L + A - P
R_386_COPY	5	none	none
R_386_GLOB_DAT	6	*word32*	S
R_386_JMP_SLOT	7	*word32*	S
R_386_RELATIVE	8	*word32*	B + A
R_386_GOTOFF	9	*word32*	S + A - GOT
R_386_GOTPC	10	*word32*	GOT + A - P

```
typedef struct {
        Elf32_Addr      r_offset;
        Elf32_Word      r_info;
} Elf32_Rel;

typedef struct {
        Elf32_Addr      r_offset;
        Elf32_Word      r_info;
        Elf32_Sword     r_addend;
} Elf32_Rela;
```

Segments

```
typedef struct {
        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;
```

How does PIE work?

Name	Value	Meaning
ET_NONE	0	No file type
ET_REL	1	Relocatable file
ET_EXEC	2	Executable file
ET_DYN	3	Shared object file
ET_CORE	4	Core file
ET_LOPROC	0xff00	Processor-specific
ET_HIPROC	0xffff	Processor-specific