ELF Relocatable Object File Format

ELF header (16 B)		Bootstrapping information for file
`.text`		Machine code of compiled module
`.rodata`		Read-only data (e.g., `printf` format strings, jump tables)
`.data`		Initialized global / static variables
`.bss`		Uninitialized static variables + those initialized to 0
`.symtab`		Symbol table
`.rel.text`		List of `.text` locations that need to be modified
`.rel.data`		List of `.data` locations that need to be modified
`.debug`	optional	Debugging symbol table
`.line`	optional	Mapping between source line #s and `.text` instructions
`.strtab`		String table for symbols in `.symtab`, `.debug`, and section names
Section Header Table		Fixed-size entries describing each section

The Tasks of The Linker

- Input: A collection $M = \{M_1, \dots, M_k\}$ of relocatable object modules.
  - Let module $M_i = (T_i, D_i, \Sigma_i)$, with T_i denoting its `.text` section, D_i its `.data` section, and Σ_i its symbol table.
  - Σ_i contains information about the symbols defined in or referenced by module M_i.

The Tasks of The Linker

- Input: A collection $M = \{M_1, \ldots, M_k\}$ of relocatable object modules.
  - Let module $M_i = (T_i, D_i, \Sigma_i)$, with T_i denoting its `.text` section, D_i its `.data` section, and Σ_i its symbol table.
  - Σ_i contains information about the symbols defined in or referenced by module M_i.
- Symbol resolution: Match each reference to a link-time symbol in each module with exactly one symbol definition from the set of defined symbols across all the modules in M.

The Tasks of The Linker

- Input: A collection $M = \{M_1, \dots, M_k\}$ of relocatable object modules.
  - Let module $M_i = (T_i, D_i, \Sigma_i)$, with T_i denoting its `.text` section, D_i its `.data` section, and Σ_i its symbol table.
  - Σ_i contains information about the symbols defined in or referenced by module M_i.
- Symbol resolution: Match each reference to a link-time symbol in each module with exactly one symbol definition from the set of defined symbols across all the modules in M.
- Relocation: Merge `.text` and `.data` sections from multiple relocatable object modules in a coordinated manner.
  - Create a single `.text` section T from $\{T_1, \dots, T_k\}$, and a single `.data` section D from $\{D_1, \dots, D_k\}$.
  - Adjust any text and data symbol references that need to change as a result of this merge.

The Tasks of The Linker

- Input: A collection $M = \{M_1, \dots, M_k\}$ of relocatable object modules.
  - Let module $M_i = (T_i, D_i, \Sigma_i)$, with T_i denoting its `.text` section, D_i its `.data` section, and Σ_i its symbol table.
  - Σ_i contains information about the symbols defined in or referenced by module M_i.
- Symbol resolution: Match each reference to a link-time symbol in each module with exactly one symbol definition from the set of defined symbols across all the modules in M.
- Relocation: Merge `.text` and `.data` sections from multiple relocatable object modules in a coordinated manner.
  - Create a single `.text` section T from $\{T_1, \dots, T_k\}$, and a single `.data` section D from $\{D_1, \dots, D_k\}$.
  - Adjust any text and data symbol references that need to change as a result of this merge.
- Output: The binary executable or shared object file.

Link-Time Symbols

- A link-time symbol in a relocatable object module is a procedure name, global variable, or `static` variable that is declared, defined, or referenced in the module.

Link-Time Symbols

- A link-time symbol in a relocatable object module is a procedure name, global variable, or `static` variable that is declared, defined, or referenced in the module.
- Method-local non-`static` variables are of no interest to the linker.
  - These are compile-time symbols. They are not even described in the module's symbol table.
  - The compiler binds such symbols to an offset in the activation record for the method in which the symbol is defined.
  - An activation record is instantiated on the run-time stack for each invocation of the method.

Link-Time Symbols

- A link-time symbol in a relocatable object module is a procedure name, global variable, or `static` variable that is declared, defined, or referenced in the module.
- Method-local non-`static` variables are of no interest to the linker.
  - These are compile-time symbols. They are not even described in the module's symbol table.
  - The compiler binds such symbols to an offset in the activation record for the method in which the symbol is defined.
  - An activation record is instantiated on the run-time stack for each invocation of the method.
- Procedure-local `static` variables are allocated in the `.data` or `.bss` sections of the module.
  - These are also compile-time symbols, but they are described in the module's symbol table as they require relocation.
  - The compiler ensures that there is only one definition per module for each procedure-local static variable, and creates a unique name for it.

More on Link-Time Symbols

- Useful Boolean function on symbols:
  - $\text{DEF}(s, m) = \textbf{true}$, if module m contains a definition of symbol s; **false**, otherwise.
  - $\text{USE}(s, m) = \textbf{true}$, if module m contains one or more references to symbol s; **false**, otherwise.
  - $\text{VISIBLE}(s) = \textbf{true}$, if s is a non-`static` symbol; **false**, otherwise.

More on Link-Time Symbols

- Useful Boolean function on symbols:
  - $\mathrm{DEF}\ (s, m)\ =\ \mathbf{true}$, if module m contains a definition of symbol s; **false**, otherwise.
  - $\mathrm{USE}\ (s, m)\ =\ \mathbf{true}$, if module m contains one or more references to symbol s; **false**, otherwise.
  - $\mathrm{VISIBLE}(s)\ =\ \mathbf{true}$, if s is a non-`static` symbol; **false**, otherwise.
- With reference to module m, link-time symbol s is said to be:
  - Local, if $\mathrm{DEF}(s, m) \wedge \neg\, \mathrm{VISIBLE}(s)$.
  - Global, if $\mathrm{DEF}(s, m) \wedge\ \mathrm{VISIBLE}(s)$.
  - External, if $\mathrm{USE}(s, m) \wedge\ \neg\, \mathrm{DEF}(s, m)$.

More on Link-Time Symbols

- Useful Boolean function on symbols:
  - $\text{DEF}\ (s, m) = \textbf{true}$, if module m contains a definition of symbol s; **false**, otherwise.
  - $\text{USE}\ (s, m) = \textbf{true}$, if module m contains one or more references to symbol s; **false**, otherwise.
  - $\text{VISIBLE}(s) = \textbf{true}$, if s is a non-`static` symbol; **false**, otherwise.
- With reference to module m, link-time symbol s is said to be:
  - Local, if $\text{DEF}(s, m) \wedge \neg\, \text{VISIBLE}(s)$.
  - Global, if $\text{DEF}(s, m) \wedge \text{VISIBLE}(s)$.
  - External, if $\text{USE}(s, m) \wedge \neg\, \text{DEF}(s, m)$.
- The linker resolves external symbol references with global symbol definitions across modules.
  - When the compiler encounters an external symbol, it generates a link-time symbol in the module's symbol table, and passes it on to the linker to resolve.
  - The linker tries to find a *unique* definition for this symbol reference among the symbol definitions across all the modules that it is linking.

Link-Time Symbols: Examples

main.c

```
int sum(int*, int);

int array[2] = {1,2};

int main(void) {
    int val = sum(array, 2);
    return val;
}
```

- The symbol `sum` is **external**.
- The symbol `array` is **global**.
- The symbol `main` is **global**.

sum.c

```
int sum(int *a, int n) {
    int s = 0;
    for (int i = 0; i < n; i++)
        s += a[i];
    return s;
}
```

- The symbol `sum` is **global**.

Link-Time Symbols: Examples

main.c

```
int sum(int*, int);

int array[2] = {1,2};

int main(void) {
    int val = sum(array, 2);
    return val;
}
```

- The symbol `sum` is **external**.
- The symbol `array` is **global**.
- The symbol `main` is **global**.

sum.c

```
int sum(int *a, int n) {
    int s = 0;
    for (int i = 0; i < n; i++)
        s += a[i];
    return s;
}
```

- The symbol `sum` is **global**.

sum1.c

```
static int sum(int *a, int n) {
    int s = 0;
    for (int i = 0; i < n; i++)
        s += a[i];
    return s;
}
```

- The symbol `sum` is **local**.

Rules for Unique Resolution

- The *global* symbols of module m are further divided into *strong* and *weak* symbols.
  - Function names and initialized variables are strong symbols.
  - Uninitialized variables are weak symbols.

Rules for Unique Resolution

- The *global* symbols of module m are further divided into *strong* and *weak* symbols.
  - Function names and initialized variables are strong symbols.
  - Uninitialized variables are weak symbols.

- Let s and w be the total number of strong and weak definitions matching an external reference r.
  - If $s = w = 0$, declare a link-time error: no definition found.
  - If $s > 1$, declare a link-time error: multiple strong definitions.
  - If $s = 1$, resolve r to the ***unique*** strong definition.
  - If $s = 0 \wedge w = 1$, resolve r to the ***unique*** weak definition.
  - If $s = 0 \wedge w > 1$, resolve r to an ***arbitrary*** weak definition.