CSI 402 – Lecture 9

(Linkers and Loaders – Continued)

External Symbols in C

- Discussion of Handout 9.1.
- Compiler must also handle the `static` attribute for functions and variables.

SIC/XE – A Representation for EDT and ERT:

- EDT and ERT as part of object code itself.
- **Definition Record** (D-record) for EDT: Each D-record has a symbol and its relative address.
- **Reference Record** (R-record) for ERT:
  - R-record contains all the external symbols referenced by the module.
  - Addresses will appear in modifier records.
  - R-records are strictly unnecessary; they are useful in making the linking process more efficient.

Changes to SIC/XE Modifier Records

- Current format: Each modifier record contains starting byte address and number of bytes.
- Adequate for relocation but not for linking.
- New format: Each modifier record contains starting byte address, number of bytes, a flag and a local or external symbol.
- Flag can be '+' or '-'.
- New format can handle both relocation and linking.

Example: To be presented in class. (The example is based on Handout 8.3.)

Local and External Symbols

Example: Numbers shown below are LC values.

```
        FUNCT    CSECT
                 EXTDEF    BUFFER
                 EXTREF    SYMBTAB
0                LDA       #15
3                +ADD      BUFFER
7                +STA      SYMBTAB
                   .
                   .
        BUFFER   RESW      10
                 END
```

M-Records for the above module:

```
M   4   3   +   FUNCT
M   8   3   +   SYMBTAB
```

Local and External Symbols (continued)

Example (continued): LC value is shown for the relevant statement.

```
        READ    CSECT
                EXTREF    BUFFER
                  .
                  .
75              +STA      BUFFER
                  .
                  .
                END
```

An M-Record for the above module:

```
  M   76   3   +  BUFFER
```

Local and External Symbols (continued)

Notes:

- When the symbol `BUFFER` is referenced in the module where it is defined, the corresponding M-record uses the name of the module.
- When the symbol `BUFFER` is referenced in a module where it is an external symbol, the corresponding M-record uses the symbol BUFFER itself.

Need for the '-' Flag:

- Operand field may have an expression containing external symbols.
- These expressions are generally simple. (They contain only '+' and '-' operators).

Need for the '-' Flag (continued)

Example:

```
FUNCT     CSECT
          EXTREF    FIRST,LAST
            .
            .
SIZE      WORD      LAST-FIRST+1
```

- Suppose the LC value of `SIZE` is 70.
- The assembler initializes bytes 70, 71 and 72 to 0, 0 and 1 respectively.
- It writes the following M-records to the object file.

```
M   70   3   +  LAST
M   70   3   -  FIRST
```

Algorithm for a Linking Loader

- **Input:** Object code for each module organized as Header record, D-record, R-record, Text records, M-records and End record.
- **Assumptions:**
  - Only one module specifies a starting address in its End record.
  - Modules loaded successively with no gaps.
- Algorithm uses two passes.
- This version of algorithm ignores R-records.
- **Pass I:** Builds the External Symbol Table by combining the EDTs of the modules.
- **Pass II:** Resolves the external symbols, performs relocation and loading and starts execution.
- **Algorithm Outline:** Handout 9.2.
- **Reading assignment:** Figures 3.11(a) and 3.11(b) of [Beck].

Usefulness of R-records

- With the revised format for modifier records, R-records are strictly unnecessary.
- R-records can be used to improve the efficiency of the linking process.

Example:

- Suppose the External Symbol Table (EST) has 100 symbols.
- **Fact:** If we have a sorted EST with n entries, we can use binary search to look for a symbol; in the worst case, this uses $\lceil \log_2 (n+1) \rceil$ probes. (Each probe involves a call to a function such as `strcmp`.)
- For the above example, the number of probes needed for each search is $\lceil \log_2 100 \rceil = 7$.

Usefulness of R-records (continued)

Example (continued):

- Suppose a particular module references only 3 of the symbols, say `A`, `B` and `C`, of the EST. (We would know this from the R-record for the module.)
- Let there be 5 M-records each for the symbols `A`, `B` and `C`. (Total: 15 M-records.)

Method I:

- Ignore the R-record and directly search the (EST) for each of the 15 M-records.
- Total number of probes in Method I $= 15 \times 7 = 105$.

Usefulness of R-records (continued)

Method II:

- The R-record of the module is:

```
R    A, B, C
```

- Search the EST and find the addresses of `A`, `B` and `C`. The number of probes used in this step $= 3 \times 7 = 21$.
- Create a smaller EST containing just these three symbols and their addresses.
- For each of the 15 M-records, search the smaller EST. Each search uses at most 3 probes (even when sequential search is used). So, the number of probes used in this step $= 15 \times 3 = 45$.
- Total number of probes used in Method II $= 21 + 45 = 66$.

Conclusion: Method II uses about 40% fewer probes than Method I.

Linking C programs with Libraries

- Functions from `<stdlib.h>` (e.g. `malloc`, `free`, `exit`) are linked automatically.
- Functions from `<stdio.h>` (e.g. `fprintf`, `fscanf`) are linked if `<stdio.h>` is included as a header.
- If functions from other libraries (e.g. `<math.h>`) are used, then the corresponding header must be included and the library must be specified as part of the `gcc` command.

Example: Suppose a program (say `prog.c`) uses the function `sqrt` from the math library.

- The program must include `<math.h>`.
- The Command for producing load module is:

```
% gcc   prog.c   -lm
```

Linking C programs with Libraries (continued)

Notes:

- The linker searches the libraries only after obtaining information about all the functions defined in the various files.
- This allows a user to redefine a library function. (If this is really necessary, it must be done with great care.)

Dynamic Linking:

- **Static Linking:** Linking done before execution.
- **Dynamic Linking:** Linking done during execution.
- Dynamic linking is useful when a program calls only a few of a large collection of routines from a library (e.g. a library containing error handling routines).
- Advantage: The size of the load module is smaller.
- Disadvantage: Additional runtime overhead.

Dynamic Linking (continued)

Steps used in Dynamic Linking:

1. A user program calls a routine (say `sqrt`) which is not part of the load module.
2. The runtime system makes a service request to the OS to load the requested function. The request is handled by a part of the OS called the "Dynamic Loader" (DL).
3. DL checks if the requested routine is already in memory; if not, loads the routine at an appropriate part of memory. DL also starts the execution of the routine.
4. When the routine completes, it returns control to DL.
   DL decides whether or not to leave the routine in memory or reclaim the memory.
5. DL passes the control back to the user program.

Dynamic Linking (continued)

Additional Remarks:

- **Binding:** Association of an attribute value with a name.
- Binding can happen at compile time, at load time or at execution time.
- With static linking, the binding between a function name and its address happens at load time (before execution begins). For a given execution, this binding does not change with time (static binding).
- With dynamic linking, the binding between a function name and its address may happen at execution time; this binding may also change with time (dynamic binding).

Bootstrap Loader

Issue: How to load the very first program into memory.

- A small part of memory is implemented as Read Only Memory (ROM) which contains a program. (This program is an absolute loader.)
- When power is turned on (or when the system is rebooted), the ROM program begins to execute. The program reads a block of bytes from a specific device (e.g. hard disk).
- The block (called the **boot block**) read in by the ROM program contains another program which can load a larger program.
- Thus, the program in the boot block loads a larger program, ..., and so on, until the OS itself is loaded.
- This sequence of loading larger and larger programs until the OS is loaded is called **bootstrapping**.
- Systems also perform "Power on self-tests" during booting.

Bootstrap Loader (continued)

Remarks:

- Some people refer to the ROM program as the bootstrap loader.
- Others refer to the program in the boot block as the bootstrap loader.

Reading Assignment: Figure 3.3 of [Beck].

Notes on the program in Figure 3.3 of [Beck]:

- It is a bootstrap loader (i.e., a simple absolute loader) for SIC/XE (that can be stored in ROM).
- It reads bytes from device F1 (hex) and loads the bytes from address 80 (hex).
- The device returns 04 (hex) when EOF occurs.
- At that time, the program causes a jump to location 80 (hex) to start the program that was just loaded.