



A.1 字节序 (Byte Order)

A.2 ELF 常见段

A.3 常用开发工具命令行参考

A.1 字节序 (Byte Order)

“endian”这个词出自 Jonathan Swift 在 1726 年写的讽刺小说《格列佛游记》(Gulliver's Travels)。小人国的内战就源于吃水煮鸡蛋时究竟是从大头 (Big-Endian) 敲开还是从小头 (Little-Endian) 敲开, 由此曾发生过 6 次叛乱, 其中一个皇帝送了命, 另一个丢了王位。

在不同的计算机体系结构中, 对于数据 (比特、字节、字) 等的存储和传输机制有所不同, 因而引发了计算机领域中一个潜在但是又很重要的问题, 即通信双方交流的信息单元应该以什么样的顺序进行传送。如果达不成一致的规则, 计算机的通信与存储将会无法进行。日前在各种体系的计算机中通常采用的字节存储机制主要有两种: 大端 (Big-endian) 和小端 (Little-endian)。

首先让我们来定义两个概念:

MSB 是 Most Significant Bit/Byte 的首字母缩写, 通常译为最重要的位或最重要的字节。它通常用来表明在一个 bit 序列 (如一个 byte 是 8 个 bit 组成的一个序列) 或一个 byte 序列 (如 word 是两个 byte 组成的一个序列) 中对整个序列取值影响最大的那个 bit/byte。

LSB 是 Least Significant Bit/Byte 的首字母缩写, 通常译为最不重要的位或最不重要的字节。它通常用来表明在一个 bit 序列 (如一个 byte 是 8 个 bit 组成的一个序列) 或一个 byte 序列 (如 word 是两个 byte 组成的一个序列) 中对整个序列取值影响最小的那个 bit/byte。

比如一个十六进制的整数 0x12345678 里面:

0x12	0x34	0x56	0x78
------	------	------	------

0x12 就是 MSB (Most Significant Byte), 0x78 就是 LSB (Least Significant Byte)。而对于 0x78 这个字节而言, 它的二进制是 01111000, 那么最左边的那个 0 就是 MSB (Most Significant Bit), 最右边的那个 0 就是 LSB (Least Significant)。

Big-endian 和 little-endian 的区别就是 bit-endian 规定 **MSB** 在存储时放在低地址, 在传输时 **MSB** 放在流的开始; **LSB** 存储时放在高地址, 在传输时放在流的末尾。little-endian 则相反。例如: 0x12345678h 这个数据在不同机器中的存储是不同, 如表 A-1 所示。

表 A-1

	Big-Endian	Little-Endian
0 字节	0x12	0x78
1 字节	0x34	0x56

续表

	Big-Endian	Little-Endian
2 字节	0x56	0x34
3 字节	0x78	0x21

Little-Endian 主要用于我们现在的 PC 的 CPU 中,即 Intel 的 x86 系列兼容机;Big-Endian 则主要应用在目前的 Mac 机器中,一般指 PowerPC 系列处理器。另外值得一提的是,目前的 TCP/IP 网络及 Java 虚拟机的字节序都是 Big-endian 的。这意味着如果通过网络传输 0x12345678 这个整形变量,首先被发送的应该是 0x12,接着是 0x34,然后是 0x56,最后是 0x78。所以我们的程序在处理网络流的时候,必须注意字节序的问题。

big-endian 和 little-endian 的争论由来已久,计算机界对两种方式的优劣进行了长期的争论,争论双方相互不妥协(至今仍未完全妥协)。Danny Cohen 于 1980 年写的一篇名叫“On Holy Wars and a Plea for Peace”著名的论文形象地将双方比喻成《格列佛游记》小人国里征战的双方。从此以后这两个术语开始流行并且一直延用至今。

A.2 ELF 常见段

ELF 常见名如表 A-2 所示。

表 A-2

段名	说明
.bss	这个段里面保存了那些程序中用到的、基本上未初始化的数据。这个段在程序被运行时,在内存中会被清零。这个段本书不占用磁盘空间,它的属性为 SHT_NOBITS。具体请参照 3.3 节
.comment	这个段包含编译器版本信息
.data	这个段中包含的是程序中初始化的数据,主要是已初始化的全局变量、静态变量
.data1	与.data 类似
.debug	这个段中包含的是调试信息
.dynamic	动态链接信息。详见 7.5.2 节
.dynstr	动态链接时的字符串表,主要是动态链接符号的符号名。详见 7.5.3 节
.dynsym	动态链接时的符号表,主要用于保存动态链接时的符号。详见 7.5.3 节
.fini	程序退出时执行的代码,这些代码晚于 main 函数执行,多数被用作实现 C++全局析构。详见 11.4 节
.fini_array	包含一些程序或共享对象退出时须要执行的函数指针
.hash	符号表的哈希表,主要用于加快符号查找

续表

段名	说明
.init	程序执行前的初始化代码, 这些代码早于 main 函数被执行, 多数时被用于实现 C++ 全局构造。详见 11.4 节
.init_array	包含一些程序或共享对象刚开始初始化时所须要执行的函数指针
.interp	包含了动态链接器的路径。详见 7.5.1 节
.line	包含了调试时用的行号信息, 主要表示机器代码与源代码行号之间的对应关系
.note	额外信息段, 编译器、链接器或操作系统厂商可能会在里面保存程序相关的额外信息, 这个属于平台相关的
.preinit_array	保存的是早于初始化阶段执行的函数指针数组, 这些函数会在 .init_array 的函数指针数组之前被执行
.rodata	只读数据段
.rodata.l	同 .rodat
.shstrtab	段名字符串表
.strtab	字符串表, 通常是符号表里的符号名所需要的字符串
.symtab	符号表, 这个段中保存的是链接时所需要的符号信息。详见 3.5 节
.tbss	这个段保存的是线程局部存储的未初始化数据。默认情况下, 每次进程启动新的线程时, 系统会产生一份 .tbss 副本并且将它的内容初始化为零
.tdata	这个段保存的是线程局部存储的初始化数据。默认情况下, 每次进程启动新的线程时, 系统会产生一份 .tdata 副本
.text	代码段, 存放程序的可执行代码。详见 3.3.1 节
.ctors	这个段保存的是全局构造函数指针。详见 11.4 节
.data.rel.ro	这个段保存的是程序的只读数据, 与 .rodata 类似, 唯一不同的是它在重定位时会被改写, 然后将会被置为只读
.dtors	这个段保存的是全局析构函数指针。详见 11.4 节
.eh_frame	这个段保存的是与 C++ 异常处理相关的内容
.eh_frame_hdr	这个段保存的是与 C++ 异常处理相关的内容
.gcc_except_table	语言相关数据
.gnu.version	符号版本相关。详见 8.2 节
.gnu.version_d	符号版本相关。详见 8.2 节
.gnu.version_r	符号版本相关。详见 8.2 节
.got.plt	这个段保存的是 PLT 信息, 详见 7.4 节
.jcr	Java 程序相关
.note.ABI-tag	用于指定程序的 ABI
.stab	调试信息
.stabstr	.stab 中用到的字符串

程序员的自我修养——链接、装载与库

A.3 常用开发工具命令行参考

A.3.1 gcc, GCC 编译器

- -E: 只进行预处理并把预处理结果输出。
- -c: 只编译不链接。
- -o <filename>: 指定输出文件名。
- -S: 输出编译后的汇编代码文件。
- -I: 指定头文件路径。
- -e name: 指定 name 为程序入口地址。
- -ffreestanding: 编译独立的程序, 不会自动链接 C 运行库、启动文件等。
- -finline-functions, -fno-inline-functions: 启用/关闭内联函数。
- -g: 在编译结果中加入调试信息, -ggdb 就是加入 GDB 调试器能够识别的格式。
- -L <directory>: 指定链接时查找路径, 多个路径之间用冒号隔开。
- -nostartfiles: 不要链接启动文件, 比如 crtbegin.o、crtend.o。
- -nostdlib: 不要链接标准库文件, 主要是 C 运行库。
- -O0: 关闭所有优化选项。
- -shared: 产生共享对象文件。
- -static: 使用静态链接。
- -Wall: 对源代码中的多数编译警告进行启用。
- -fPIC: 使用地址无关代码模式进行编译。
- -fPIE: 使用地址无关代码模式编译可执行文件。
- -XLinker <option>: 把 option 传递给链接器。
- -Wl <option>: 把 option 传递给链接器, 与上面的选项类似。
- -fomit-frame-pointer: 禁止使用 EBP 作为函数帧指针。
- -fno-builtin: 禁止 GCC 编译器内置函数。
- -fno-stack-protector: 是指关闭堆栈保护功能。
- -ffunction-sections: 将每个函数编译到独立的代码段。
- -fdata-sections: 将全局/静态变量编译到独立的数据段。

A.3.2 ld, GNU 链接器

- -static: 静态链接。
- -l<libname>: 指定链接某个库。
- -e name: 指定 name 为程序入口。
- -r: 合并目标文件, 不进行最终链接。
- -L <directory>: 指定链接时查找路径, 多个路径之间用冒号隔开。
- -M: 将链接时的符号和地址输出成一个映射文件。
- -o: 指定输出文件名。
- -s: 清除输出文件中的符号信息。
- -S: 清除输出文件中的调试信息。
- -T <scriptfile>: 指定链接脚本文件。
- -version-script <file>: 指定符号版本脚本文件。
- -soname <name>: 指定输出共享库的 SONAME。
- -export-dynamic: 将全局符号全部导出。
- -verbose: 链接时输出详细信息。
- -rpath <path>: 指定链接时库查找路径。

A.3.3 objdump, GNU 目标文件可执行文件查看器

- -a: 列举.a 文件中所有的目标文件。
- -b bfdname: 指定 BFD 名。
- -C: 对于 C++ 符号名进行反修饰 (Demangle)。
- -g: 显示调试信息。
- -d: 对包含机器指令的段进行反汇编。
- -D: 对所有的段进行反汇编。
- -f: 显示目标文件文件头。
- -h: 显示段表。
- -l: 显示行号信息。
- -p: 显示专有头部信息, 具体内容取决于文件格式。
- -r: 显示重定位信息。
- -R: 显示动态链接重定位信息。

- -s: 显示文件所有内容。
- -S: 显示源代码和反汇编代码（包含-d 参数）。
- -W: 显示文件中包含有 DWARF 调试信息格式的段。
- -t: 显示文件中的符号表。
- -T: 显示动态链接符号表。
- -x: 显示文件的所有文件头。

A.3.4 cl, MSVC 编译器

- /c: 只编译不链接。
- /Za: 禁止语言扩展。
- /link: 链接指定的模块或给链接器传递参数。
- /Od: 禁止优化。
- /O2: 以运行速度最快为目标优化。
- /O1: 以最节省空间为目标优化。
- /GR 或/GR-: 开启或关闭 RTTI。
- /Gy: 开启函数级别链接。
- /GS 或/GS-: 开启或关闭。
- /Fa[file]: 输出汇编文件。
- /E: 只进行预处理并且把结果输出。
- /I: 指定头文件包含目录。
- /Zi: 启用调试信息。
- /LD: 编译产生 DLL 文件。
- /LDd: 编译产生 DLL 文件（调试版）。
- /MD: 与动态多线程版本运行库 MSVCRT.LIB 链接。
- /MDd: 与调试版动态多线程版本运行库 MSVCRTD.LIB 链接。
- /MT: 与静态多线程版本运行库 LIBCMT.LIB 链接。
- /MTd: 与调试版静态多线程版本运行库 LIBCMTD.LIB 链接。

A.3.5 link, MSVC 链接器

- /BASE:address: 指定输出文件的基地址。

- /DEBUG: 输出调试模式版本。
- /DEF:filename: 指定模块定义文件.DEF。
- /DEFAULTLIB:library: 指定默认运行库。
- /DLL: 产生 DLL。
- /ENTRY:symbol: 指定程序入口。
- /EXPORT:symbol: 指定某个符号为导出符号。
- /HEAP: 指定默认堆大小。
- /LIBPATH:dir: 指定链接时库搜索路径。
- /MAP[:filename]: 产生链接 MAP 文件。
- /NODEFAULTLIB[:library]: 禁止默认运行库。
- /OUT:filename: 指定输出文件名。
- /RELEASE: 以发布版本产生输出文件。
- /STACK: 指定默认栈大小。
- /SUBSYSTEM: 指定子系统。

A.3.6 dumpbin, MSVC 的 COFF/PE 文件查看器

- /ALL: 显示所有信息。
- /ARCHIVEMEMBERS: 显示.LIB 文件中所有目标文件列表。
- /DEPENDENTS: 显示文件的动态链接依赖关系。
- /DIRECTIVES: 显示链接器指示。
- /DISASM: 显示反汇编。
- /EXPORTS: 显示导出函数表。
- /HEADERS: 显示文件头。
- /IMPORTS: 显示导入函数表。
- /LINENUMBERS: 显示行号信息。
- /RELOCATIONS: 显示重定位信息。
- /SECTION:name: 显示某个段。
- /SECTION: 显示文件概要信息。
- /SYMBOLS: 显示文件符号表。
- /TLS: 显示线程局部存储 TLS 信息。

索引

- ABI (Application Binary Interface) 应用程序二进制接口, 115, 230
- Activate Record 活动记录, 287
- Address and Storage Allocation 地址和空间分配, 51
- API (Application Programming Interface) 应用程序编程接口, 9, 117
- ANSI (American National Standard Institute) 美国国家标准学会, 336
- Anonymous Virtual Memory Area 匿名虚拟内存区域, 166
- Assembly 汇编, 38
- Atomic 原子的, 25
- AWE (Address Windowing Extensions) 地址窗口扩展, 152
- Base Address 基地址, 175
- Base Index Scale Addressing 基址比例变址寻址, 47
- BFD (Binary File Descriptor Library) 二进制文件描述符库, 131
- Big-endian 大端, 66
- Binary Semaphore 二元信号量, 26
- Bootstrap 自举, 214
- BSS (Block Started by Symbol), 59
- Built-in Function 内置函数, 126
- Bus 总线, 6
- Byte Order 字节序, 66
- Calling Convention 调用惯例, 294
- Code Generator 代码生成器, 47
- Code Section 代码段, 58
- COFF (Common Object File Format) 通用对象文件格式, 134
- COM (Component Object Model) 组件对象模型, 275
- Common Block, Common 块, 111
- Compilation 编译, 38
- Condition Variable 条件变量, 27
- Context-free Grammar 上下文无关语法, 43
- Core Dump File 核心转储文件, 57
- COW (Copy-on-Write) 写时复制, 23
- CPU Bound, CPU 密集型, 22
- Critical Section 临界区, 26
- Data Section 数据段, 58
- Decorated Name 修饰后名称, 88
- Delayed Load 延迟载入, 264
- Dependency Ordering 依赖序列, 224
- Device Driver 硬件驱动, 12
- Disk Page 磁盘页, 17
- DLL Binding, DLL 绑定, 271
- DLL Hell, DLL 噩梦, 276
- DSO (Dynamic Shared Object) 动态共享对象, 183
- DWARF (Debug With Arbitrary Record Format) 通用调试记录格式, 95
- Dynamic Linker 动态链接器, 203
- Dynamic Linking 动态链接, 181
- Dynamic Linking Library 动态链接库, 56, 183
- Dynamic Semantic 动态语义, 44
- Dynamic Symbol Table 动态符号表, 206
- ELF (Executable Linkable Format) 可执行可连接格式, 56
- ELF Header, ELF 文件头, 69
- Entry Point 入口函数或入口点, 319

- Environment Subsystem 环境子系统, 409
- EXE (Executable) 可执行文件, 56
- Executable File 可执行文件, 57
- Execution View 执行视图, 164
- Exit Code 退出码, 126
- Explicit Run-time Linking 显式运行时链接, 221
- Export Function 导出函数, 206
- EAT (Export Address Table) 导出地址表, 258
- Export Forwarding 导出重定向, 261
- Export Table 导出表, 146
- Expression 表达式, 43
- FHS (File Hierarchy Standard) 文件层次结构标准, 241
- File Descriptor 文件描述符, 328
- Finite State Machine 有限状态机, 42
- Frame Pointer 帧指针, 288
- Free List 空闲链表, 312
- Function Level Linking 函数级别链接, 114
- Function Signature 函数签名, 88
- Global Symbol Interposition 全局符号介入, 192
- GOT (Global Offset Table) 全局偏移表, 194
- Grammar Parser 语法分析器, 43
- Handle 句柄, 328
- Hardware Specification 硬件规格, 10
- Heap 堆, 166
- Heap Manager 堆管理器, 310
- Hook 钩子, 293
- Image File 映像文件, 136
- Image Header 映像头, 136
- Import 导入, 206
- Import Address Table 导入地址数组, 263
- Import Function 导入函数, 206
- Import Library 导入库, 254
- Interface 接口, 9
- Intermediate Code 中间代码, 46
- Interrupt 中断, 388
- I/O Bound, I/O 密集型, 22
- ISR (Interrupt Service Routine) 中断处理程序, 389
- IVT (Interrupt Vector Table) 中断向量表, 389
- Kernel Mode 内核模式, 388
- Lazy Binding 延迟绑定, 184
- LBA (Logical Block Address) 逻辑扇区号, 13
- LWP (Lightweight Process) 轻量级进程, 19
- Library 库, 51
- Link Name 链接名, 235
- Link Time Relocation 链接时重定位, 190
- Linking 链接, 38, 50, 51
- Linking View 链接视图, 164
- LSB (Linux Standard Base) Linux 基础标准, 117
- Little-endian 小端, 66
- Load Time Relocation 装载时重定位, 190
- Load Ordering 装载序列, 224
- Lock 锁, 26
- LSB (Least Significant Bit/Byte) 影响最小的位/字节, 450
- Manifest, Manifest 文件, 277
- Manipulator 操纵符, 434
- Minor-revision Rendezvous Problem 次版本号交会问题, 236
- MMU (Memory Manager Unit) 内存管理单元, 18
- Module Definition File 模块定义文件, 124
- MSB (Most Significant Bit/Byte) 影响最大的位/字节, 450
- Multiprogramming 多道程序, 10
- Multi-tasking 多任务系统, 11
- Mutex 互斥量, 26
- Name Decoration 符号修饰, 87
- Name Mangling 符号改编, 87
- Name-Ordinal Table 名字序号对应表, 258
- Namespace 名称空间, 87
- Northbridge 北桥, 6
- Object File 目标文件, 51
- Ordinal Number 序号, 270
- Overlay 覆盖装入, 153
- Package 包, 50
- PAE (Physical Address Extension) 物理地址扩展, 152
- Page Fault 页错误, 17, 159
- Paging 分页, 17
- P-Code, P-代码, 46
- PE (Portable Executable) 可移植可执行文件, 134
- Physical Page 物理页, 17
- PIC (Position-independent Code) 地址无关代码, 190
- PIE (Position-Independent Executable) 地址无关可执行文件, 197
- PLT (Procedure Linkage Table) 过程链接表, 200
- Precompiled Header File 预编译头文件, 140
- Preemption 抢占, 22

- Preemptive 抢占式, 11
- Preprocessing 预处理, 38
- Priority Schedule 优先级调度, 21
- Process 进程, 11
- Program Header 程序头, 163
- Program Header Table 程序头表, 164
- Read-Write Lock 读写锁, 27
- Rebasing 基址重置, 190, 210
- Reentrant 可重入, 27
- Reference 引用, 81
- Relocatable File 可重定位文件, 56
- Relocation 重定位, 49, 51
- Relocation Entry 重定位入口, 53, 107
- Relocation Table 重定位表, 79, 106
- Replacement New 指定对象申请地址, 437
- Round Robin 轮转法, 21
- Runtime Library 运行时库, 52, 335
- RVA (Relative Virtual Address) 相对虚拟地址, 175, 251
- Scanner 扫描器, 42
- Scoping 范围机制, 237
- Section 节, 58
- Section Descriptor 段描述符, 75
- Section Table 段表, 59
- Section Header Table 段表, 69, 74
- Segment 段, 58
- Segmentation 分段, 15
- Semantic Analyzer 语义分析器, 44
- Semaphore 信号量, 26
- Shared Library 共享库, 230
- Shared Object File 共享目标文件, 57
- SMP (Symmetrical Mutil-Processing) 对称多处理器, 7
- SDK (Software Development Kit) 软件开发套装, 402
- Software Interrupt 软件中断, 10
- Source Code Optimizer 源代码级优化器, 45
- Southbridge 南桥, 6
- Stack 栈, 166
- Stack Frame 堆栈帧, 287
- Starvation 饿死, 22
- Static Linking Library 静态链接库, 56
- Static Semantic 静态语义, 44
- Static Shared Library 静态共享库, 189
- String Table 字符串表, 80
- Strong Reference 强引用, 93
- Strong Symbol 强符号, 92
- Subsystem 子系统, 409
- Symbol 符号, 49, 81
- Symbol Link 软链接, 233
- Symbol Resolution 符号决议, 51
- Symbol Table 符号表, 66, 81
- Symbol Versioning 基于符合的版本机制, 236
- Synchronization 同步, 26
- Syntax Tree 语法树, 43
- System Call 系统调用, 384
- System Call Interface 系统调用接口, 9
- System Service 系统服务, 402
- Target Code Optimizer 目标代码优化器, 47
- Task 任务, 23
- Time-Sharing System 分时系统, 11
- Time Slice 时间片, 21
- Thread 线程, 19
- TEB (Thread Environment Block) 线程环境块, 354
- Thread Priority 线程优先级, 21
- Three-address Code 三地址码, 46
- TLS (Thread Local Storage) 线程局部存储, 353
- Token 记号, 42
- Thread Schedule 线程调度, 21
- User Mode 用户模式, 388
- VDSO (Virtual Dynamic Shared Library) 虚拟动态共享库, 399
- Versioning 版本机制, 237
- Virtual Address 虚拟地址, 15
- Virtual Address Space 虚拟地址空间, 150
- Virtual Page 虚拟页, 17
- Virtual Section 虚拟段, 159
- VMA (Virtual Memory Address) 虚拟内存地址, 102
- VMA (Virtual Memory Area) 虚拟内存区域, 159
- Weak Reference 弱引用, 93
- Weak Symbol 弱符号, 92
- WoW (Windows On Windows), 409
- XMS (eXtended Memory Specification) 扩展内存标准, 153