UNIX tools for exploring object files

Learn more about your system

William B. Zimmerly November 21, 2006

The programs that run on a UNIX® system follow a careful design known as the *object file format*. Learn more about the object file format and the tools that you can use for exploring object files found on your system.

The modern art of computer programming combines a special kind of human personality with a special set of tools to produce a rather ghostly product -- software -- that other human beings find useful. Computer programmers are detail-oriented folks who are able to deal with the difficulties of computers. Computers are exacting in their demands and don't tolerate deviation from these demands at all. No doubt about it, computers are difficult to program no matter what your personality, and many tools have been created to assist you in making the task easier.

In UNIX® and Linux®, everything is a file. You could say that the very sine qua non of UNIX and Linux programming is writing code to deal with files. Many types of files make up the system, but object files have a special design that provides for flexible, multipurpose use.

Object files are roadmaps that contain mnemonic symbols with attached addresses and values. The symbols are used for naming various sections of code and data, both initialized and uninitialized. They are also used for locating embedded debugging information and, just like the semantic Web, are fully readable by programs.

Tools of the trade

The tools of the computer programming trade begin with a code editor, such as vi or Emacs, with which you can type and edit the instructions you want the computer to follow to carry out the required tasks, and end with the compilers and linkers that produce the machine code that actually accomplishes these goals.

High-level tools, known as *Integrated Debugging Environments* (IDEs), integrate the functionality of individual tools with a common look and feel. An IDE can vastly blur the lines between editor, compiler, linker, and debugger. So for the purpose of studying and learning the system with greater depth, it's often advisable to work with the tools separately before working with the integrated suite. (**Note:** IDEs are often called *Integrated Development Environments*, too.)

The compiler transforms the text that you create in the code editor into an object file. The object file was originally known as an *intermediate* representation of code, because it served as the input to link editors (in other words, *linkers*) that finish the task and produce an executable program as output.

The transformation process that proceeds from code to executable is well-defined and automated, and object files are an integral link in the chain. During the transformation process, the object files serve as a map to the link editors, enabling them to resolve the symbols and stitch together the various code and data sections into a unified whole.

History

Many notable object file formats exist in the world of computer programming. The DOS family includes the *COM*, *OBJ*, and *EXE* formats. UNIX and Linux use *a.out*, *COFF*, and *ELF*. Microsoft® Windows® uses the *portable executable* (PE) format and Macintosh uses *PEF*, *Mach-O*, and others.

Originally, each type of computer had its own unique object file format but, with the advent of UNIX and other operating systems designed to be portable among different hardware platforms, some common file formats ascended to the level of a common standard. Among these are the *a.out*, *COFF*, and *ELF* formats.

Understanding object files requires a set of tools that can read the various portions of the object file and display them in a more readable format. This article discusses some of the more important aspects of those tools. But first, you must create a workbench and put a victim -- er, a patient -- on it.

The workbench

Fire up an xterm session, and let's begin to explore object files by creating a clean workbench. The following commands create a useful place to play with object files:

```
cd
mkdir src
cd src
mkdir hw
cd hw
```

Then, by using your favorite code editor, type the program shown in Listing 1 in the $HOME/src/hw directory, and call it *hw.c*.

Listing 1. The hw.c program

```
#include <stdio.h>

int main(void)
{
  printf("Hello World!\n");
  return 0;
}
```

This simple "Hello World" program serves as a patient to study with the various tools available in the UNIX arsenal. Instead of taking any shortcuts to creating the executable (and there are many shortcuts), you'll take your time to build and examine just the object file output.

File formats

The normal output of a `c` compiler is assembler code for whatever processor you specify as the target. The assembler code is input to the assembler, which by default produces the grandfather of all object files, the a.out file. The name itself stands for *Assembler Output.* To create the a.out file, type the following command in the xterm window:

```
cc hw.c
```

Note: If you experience any errors or the a.out file wasn't created, you might need to examine your system or source file (hw.c) for errors. Check also to see whether **cc** is defined to run your `c/c++` compiler.

Modern `c` compilers combine the compile and assemble steps into one step. You can invoke switches to see just the assembler output of the `c` compiler. By typing the following command, you can see what the assembler output from the `c` compiler looks like:

```
cc -S hw.c
```

This command has generated a new file -- hw.s -- that contains the assembler input text that you typically would not have seen, because the compiler defaults to producing the a.out file. As expected, the UNIX assembler program can assemble this type of input file to produce the a.out file.

UNIX-specific tools

Assuming that all went well with the compile and you have an a.out file in the directory, let's examine it. Among the list of useful tools for examining object files, the following set exists:

- **nm:** Lists symbols from object files.
- **objdump:** Displays detailed information from object files.
- **readelf:** Displays information about ELF object files.

The first tool on the list is nm, which lists the symbols in an object file. If you type the `nm` command, you'll notice that it defaults to looking for a file named *a.out* If the file isn't found, the tool complains. If, however, the tool did find the a.out file that your compiler created, it presents a listing similar to Listing 2.

Listing 2. Output of the nm command

```
08049594 A __bss_start
080482e4 t call_gmon_start
08049594 b completed.4463
08049498 d __CTOR_END__
08049494 d __CTOR_LIST__
```

```
08049588 D __data_start
08049588 W data_start
0804842c t __do_global_ctors_aux
0804830c t __do_global_dtors_aux
0804958c D __dso_handle
080494a0 d __DTOR_END__
0804949c d __DTOR_LIST__
080494a8 d _DYNAMIC
08049594 A _edata
08049598 A _end
08048458 T _fini
08049494 a __fini_array_end
08049494 a __fini_array_start
08048478 R _fp_hw
0804833b t frame_dummy
08048490 r __FRAME_END__
08049574 d _GLOBAL_OFFSET_TABLE_
         w __gmon_start__
08048308 T __i686.get_pc_thunk.bx
08048278 T _init
08049494 a __init_array_end
08049494 a __init_array_start
0804847c R _IO_stdin_used
080494a4 d __JCR_END__
080494a4 d __JCR_LIST__
         w _Jv_RegisterClasses
080483e1 T __libc_csu_fini
08048390 T __libc_csu_init
         U __libc_start_main@@GLIBC_2.0
08048360 T main
08049590 d p.4462
         U puts@@GLIBC_2.0
080482c0 T _start
```

The sections that contain executable code are known as *text sections* or *segments.* Likewise, there are data sections or segments for containing non-executable information or data. Another type of section, known by the *BSS* designation, contains blocks started by symbol data.

For each symbol that the `nm` command lists, the symbol's value in hexadecimal (by default) and the symbol type with a coded character precede the symbol. Various codes that you commonly see include **A** for *absolute,* which means that the value will not change by further linking; **B** for a symbol found in the BSS section; or **C** for common symbols that reference uninitialized data.

Object files contain many different parts that are divided into sections. Sections can contain executable code, symbol names, initialized data values, and many other types of data. For detailed information on all of these types of data, consider reading the UNIX man page on `nm`, where each type is described by the character codes shown in the output of the command.

Details, details . . .

Even a simple Hello World program contains a vast array of details when it reaches the object file stage. The nm program is good for listing symbols and their types and values but, for examining in greater detail the contents of those named sections of the object file, more powerful tools are necessary.

Two of these more powerful tools are the objdump and readelf programs. By typing the following command, you can see an assembly listing of every section in the object file that contains

executable code. Isn't it amazing how much code the compiler actually generates for such a tiny program?

```
objdump -d a.out
```

This command produces the output you see in Listing 3. Each section of executable code is run when a particular event becomes necessary, including events like the initialization of a library and the main starting entry point of the program itself.

Listing 3: Output of the objdump command

```
a.out:     file format elf32-i386

Disassembly of section .init:

08048278 <_init>:
 8048278:       55                      push   %ebp
 8048279:       89 e5                   mov    %esp,%ebp
 804827b:       83 ec 08                sub    $0x8,%esp
 804827e:       e8 61 00 00 00          call   80482e4 <call_gmon_start>
 8048283:       e8 b3 00 00 00          call   804833b <frame_dummy>
 8048288:       e8 9f 01 00 00          call   804842c <__do_global_ctors_aux>
 804828d:       c9                      leave
 804828e:       c3                      ret
Disassembly of section .plt:

08048290 <puts@plt-0x10>:
 8048290:       ff 35 78 95 04 08       pushl  0x8049578
 8048296:       ff 25 7c 95 04 08       jmp    *0x804957c
 804829c:       00 00                   add    %al,(%eax)
        ...

080482a0 <puts@plt>:
 80482a0:       ff 25 80 95 04 08       jmp    *0x8049580
 80482a6:       68 00 00 00 00          push   $0x0
 80482ab:       e9 e0 ff ff ff          jmp    8048290 <_init+0x18>

080482b0 <__libc_start_main@plt>:
 80482b0:       ff 25 84 95 04 08       jmp    *0x8049584
 80482b6:       68 08 00 00 00          push   $0x8
 80482bb:       e9 d0 ff ff ff          jmp    8048290 <_init+0x18>
Disassembly of section .text:

080482c0 <_start>:
 80482c0:       31 ed                   xor    %ebp,%ebp
 80482c2:       5e                      pop    %esi
 80482c3:       89 e1                   mov    %esp,%ecx
 80482c5:       83 e4 f0                and    $0xfffffff0,%esp
 80482c8:       50                      push   %eax
 80482c9:       54                      push   %esp
 80482ca:       52                      push   %edx
 80482cb:       68 e1 83 04 08          push   $0x80483e1
 80482d0:       68 90 83 04 08          push   $0x8048390
 80482d5:       51                      push   %ecx
 80482d6:       56                      push   %esi
 80482d7:       68 60 83 04 08          push   $0x8048360
 80482dc:       e8 cf ff ff ff          call   80482b0 <__libc_start_main@plt>
 80482e1:       f4                      hlt
 80482e2:       90                      nop
 80482e3:       90                      nop

080482e4 <call_gmon_start>:
 80482e4:       55                      push   %ebp
 80482e5:       89 e5                   mov    %esp,%ebp
```

```
 80482e7:       53                      push   %ebx
 80482e8:       e8 1b 00 00 00          call   8048308 <__i686.get_pc_thunk.bx>
 80482ed:       81 c3 87 12 00 00       add    $0x1287,%ebx
 80482f3:       83 ec 04                sub    $0x4,%esp
 80482f6:       8b 83 fc ff ff ff       mov    0xfffffffc(%ebx),%eax
 80482fc:       85 c0                   test   %eax,%eax
 80482fe:       74 02                   je     8048302 <call_gmon_start+0x1e>
 8048300:       ff d0                   call   *%eax
 8048302:       83 c4 04                add    $0x4,%esp
 8048305:       5b                      pop    %ebx
 8048306:       5d                      pop    %ebp
 8048307:       c3                      ret

08048308 <__i686.get_pc_thunk.bx>:
 8048308:       8b 1c 24                mov    (%esp),%ebx
 804830b:       c3                      ret

0804830c <__do_global_dtors_aux>:
 804830c:       55                      push   %ebp
 804830d:       89 e5                   mov    %esp,%ebp
 804830f:       83 ec 08                sub    $0x8,%esp
 8048312:       80 3d 94 95 04 08 00    cmpb   $0x0,0x8049594
 8048319:       74 0c                   je     8048327 <__do_global_dtors_aux+0x1b>
 804831b:       eb 1c                   jmp    8048339 <__do_global_dtors_aux+0x2d>
 804831d:       83 c0 04                add    $0x4,%eax
 8048320:       a3 90 95 04 08          mov    %eax,0x8049590
 8048325:       ff d2                   call   *%edx
 8048327:       a1 90 95 04 08          mov    0x8049590,%eax
 804832c:       8b 10                   mov    (%eax),%edx
 804832e:       85 d2                   test   %edx,%edx
 8048330:       75 eb                   jne    804831d <__do_global_dtors_aux+0x11>
 8048332:       c6 05 94 95 04 08 01    movb   $0x1,0x8049594
 8048339:       c9                      leave
 804833a:       c3                      ret

0804833b <frame_dummy>:
 804833b:       55                      push   %ebp
 804833c:       89 e5                   mov    %esp,%ebp
 804833e:       83 ec 08                sub    $0x8,%esp
 8048341:       a1 a4 94 04 08          mov    0x80494a4,%eax
 8048346:       85 c0                   test   %eax,%eax
 8048348:       74 12                   je     804835c <frame_dummy+0x21>
 804834a:       b8 00 00 00 00          mov    $0x0,%eax
 804834f:       85 c0                   test   %eax,%eax
 8048351:       74 09                   je     804835c <frame_dummy+0x21>
 8048353:       c7 04 24 a4 94 04 08    movl   $0x80494a4,(%esp)
 804835a:       ff d0                   call   *%eax
 804835c:       c9                      leave
 804835d:       c3                      ret
 804835e:       90                      nop
 804835f:       90                      nop

08048360 <main>:
 8048360:       55                      push   %ebp
 8048361:       89 e5                   mov    %esp,%ebp
 8048363:       83 ec 08                sub    $0x8,%esp
 8048366:       83 e4 f0                and    $0xfffffff0,%esp
 8048369:       b8 00 00 00 00          mov    $0x0,%eax
 804836e:       83 c0 0f                add    $0xf,%eax
 8048371:       83 c0 0f                add    $0xf,%eax
 8048374:       c1 e8 04                shr    $0x4,%eax
 8048377:       c1 e0 04                shl    $0x4,%eax
 804837a:       29 c4                   sub    %eax,%esp
 804837c:       c7 04 24 80 84 04 08    movl   $0x8048480,(%esp)
 8048383:       e8 18 ff ff ff          call   80482a0 <puts@plt>
 8048388:       b8 00 00 00 00          mov    $0x0,%eax
 804838d:       c9                      leave
```

```
 804838e:       c3                      ret
 804838f:       90                      nop

08048390 <__libc_csu_init>:
 8048390:       55                      push   %ebp
 8048391:       89 e5                   mov    %esp,%ebp
 8048393:       57                      push   %edi
 8048394:       56                      push   %esi
 8048395:       31 f6                   xor    %esi,%esi
 8048397:       53                      push   %ebx
 8048398:       e8 6b ff ff ff          call   8048308 <__i686.get_pc_thunk.bx>
 804839d:       81 c3 d7 11 00 00       add    $0x11d7,%ebx
 80483a3:       83 ec 0c                sub    $0xc,%esp
 80483a6:       e8 cd fe ff ff          call   8048278 <_init>
 80483ab:       8d 83 20 ff ff ff       lea    0xffffff20(%ebx),%eax
 80483b1:       8d 93 20 ff ff ff       lea    0xffffff20(%ebx),%edx
 80483b7:       89 45 f0                mov    %eax,0xfffffff0(%ebp)
 80483ba:       29 d0                   sub    %edx,%eax
 80483bc:       c1 f8 02                sar    $0x2,%eax
 80483bf:       39 c6                   cmp    %eax,%esi
 80483c1:       73 16                   jae    80483d9 <__libc_csu_init+0x49>
 80483c3:       89 d7                   mov    %edx,%edi
 80483c5:       ff 14 b2                call   *(%edx,%esi,4)
 80483c8:       8b 45 f0                mov    0xfffffff0(%ebp),%eax
 80483cb:       83 c6 01                add    $0x1,%esi
 80483ce:       29 f8                   sub    %edi,%eax
 80483d0:       89 fa                   mov    %edi,%edx
 80483d2:       c1 f8 02                sar    $0x2,%eax
 80483d5:       39 c6                   cmp    %eax,%esi
 80483d7:       72 ec                   jb     80483c5 <__libc_csu_init+0x35>
 80483d9:       83 c4 0c                add    $0xc,%esp
 80483dc:       5b                      pop    %ebx
 80483dd:       5e                      pop    %esi
 80483de:       5f                      pop    %edi
 80483df:       5d                      pop    %ebp
 80483e0:       c3                      ret

080483e1 <__libc_csu_fini>:
 80483e1:       55                      push   %ebp
 80483e2:       89 e5                   mov    %esp,%ebp
 80483e4:       83 ec 18                sub    $0x18,%esp
 80483e7:       89 5d f4                mov    %ebx,0xfffffff4(%ebp)
 80483ea:       e8 19 ff ff ff          call   8048308 <__i686.get_pc_thunk.bx>
 80483ef:       81 c3 85 11 00 00       add    $0x1185,%ebx
 80483f5:       89 75 f8                mov    %esi,0xfffffff8(%ebp)
 80483f8:       89 7d fc                mov    %edi,0xfffffffc(%ebp)
 80483fb:       8d b3 20 ff ff ff       lea    0xffffff20(%ebx),%esi
 8048401:       8d bb 20 ff ff ff       lea    0xffffff20(%ebx),%edi
 8048407:       29 fe                   sub    %edi,%esi
 8048409:       c1 fe 02                sar    $0x2,%esi
 804840c:       eb 03                   jmp    8048411 <__libc_csu_fini+0x30>
 804840e:       ff 14 b7                call   *(%edi,%esi,4)
 8048411:       83 ee 01                sub    $0x1,%esi
 8048414:       83 fe ff                cmp    $0xffffffff,%esi
 8048417:       75 f5                   jne    804840e <__libc_csu_fini+0x2d>
 8048419:       e8 3a 00 00 00          call   8048458 <_fini>
 804841e:       8b 5d f4                mov    0xfffffff4(%ebp),%ebx
 8048421:       8b 75 f8                mov    0xfffffff8(%ebp),%esi
 8048424:       8b 7d fc                mov    0xfffffffc(%ebp),%edi
 8048427:       89 ec                   mov    %ebp,%esp
 8048429:       5d                      pop    %ebp
 804842a:       c3                      ret
 804842b:       90                      nop

0804842c <__do_global_ctors_aux>:
 804842c:       55                      push   %ebp
 804842d:       89 e5                   mov    %esp,%ebp
```

```
 804842f:       53                      push   %ebx
 8048430:       83 ec 04                sub    $0x4,%esp
 8048433:       a1 94 94 04 08          mov    0x8049494,%eax
 8048438:       83 f8 ff                cmp    $0xffffffff,%eax
 804843b:       74 12                   je     804844f <__do_global_ctors_aux+0x23>
 804843d:       bb 94 94 04 08          mov    $0x8049494,%ebx
 8048442:       ff d0                   call   *%eax
 8048444:       8b 43 fc                mov    0xfffffffc(%ebx),%eax
 8048447:       83 eb 04                sub    $0x4,%ebx
 804844a:       83 f8 ff                cmp    $0xffffffff,%eax
 804844d:       75 f3                   jne    8048442 <__do_global_ctors_aux+0x16>
 804844f:       83 c4 04                add    $0x4,%esp
 8048452:       5b                      pop    %ebx
 8048453:       5d                      pop    %ebp
 8048454:       c3                      ret
 8048455:       90                      nop
 8048456:       90                      nop
 8048457:       90                      nop
Disassembly of section .fini:

08048458 <_fini>:
 8048458:       55                      push   %ebp
 8048459:       89 e5                   mov    %esp,%ebp
 804845b:       53                      push   %ebx
 804845c:       e8 a7 fe ff ff          call   8048308 <__i686.get_pc_thunk.bx>
 8048461:       81 c3 13 11 00 00       add    $0x1113,%ebx
 8048467:       83 ec 04                sub    $0x4,%esp
 804846a:       e8 9d fe ff ff          call   804830c <__do_global_dtors_aux>
 804846f:       83 c4 04                add    $0x4,%esp
 8048472:       5b                      pop    %ebx
 8048473:       5d                      pop    %ebp
 8048474:       c3                      ret
```

For a programmer who is fascinated by the low-level details of programming, this is a powerful tool for studying the output of compilers and assemblers. Details, such as those shown in this code, reveal a lot about how the native processor itself operates. When studied hand-in-hand with the processor manufacturer's technical documentation, you can glean valuable insights into how such things work to a greater degree because of the clarity of output from a functioning program.

Likewise, the readelf program can list the contents of the object file with similar lucidity. You can see this by typing the following command:

```
readelf -all a.out
```

This command produces the output shown in Listing 4. The ELF header shows a nice summary of all the section entries in the file. Before enumerating the contents of those headers, you can see how many there are. This information can be useful when exploring a rather large object file.

Listing 4. Output of the readelf command

```
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
```

```
  Entry point address:               0x80482c0
  Start of program headers:          52 (bytes into file)
  Start of section headers:          3504 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         7
  Size of section headers:           40 (bytes)
  Number of section headers:         34
  Section header string table index: 31

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048114 000114 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048128 000128 000020 00   A  0   0  4
  [ 3] .hash             HASH            08048148 000148 00002c 04   A  4   0  4
  [ 4] .dynsym           DYNSYM          08048174 000174 000060 10   A  5   1  4
  [ 5] .dynstr           STRTAB          080481d4 0001d4 00005e 00   A  0   0  1
  [ 6] .gnu.version      VERSYM          08048232 000232 00000c 02   A  4   0  2
  [ 7] .gnu.version_r    VERNEED         08048240 000240 000020 00   A  5   1  4
  [ 8] .rel.dyn          REL             08048260 000260 000008 08   A  4   0  4
  [ 9] .rel.plt          REL             08048268 000268 000010 08   A  4  11  4
  [10] .init             PROGBITS        08048278 000278 000017 00  AX  0   0  1
  [11] .plt              PROGBITS        08048290 000290 000030 04  AX  0   0  4
  [12] .text             PROGBITS        080482c0 0002c0 000198 00  AX  0   0  4
  [13] .fini             PROGBITS        08048458 000458 00001d 00  AX  0   0  1
  [14] .rodata           PROGBITS        08048478 000478 000015 00   A  0   0  4
  [15] .eh_frame         PROGBITS        08048490 000490 000004 00   A  0   0  4
  [16] .ctors            PROGBITS        08049494 000494 000008 00  WA  0   0  4
  [17] .dtors            PROGBITS        0804949c 00049c 000008 00  WA  0   0  4
  [18] .jcr              PROGBITS        080494a4 0004a4 000004 00  WA  0   0  4
  [19] .dynamic          DYNAMIC         080494a8 0004a8 0000c8 08  WA  5   0  4
  [20] .got              PROGBITS        08049570 000570 000004 04  WA  0   0  4
  [21] .got.plt          PROGBITS        08049574 000574 000014 04  WA  0   0  4
  [22] .data             PROGBITS        08049588 000588 00000c 00  WA  0   0  4
  [23] .bss              NOBITS          08049594 000594 000004 00  WA  0   0  4
  [24] .comment          PROGBITS        00000000 000594 000126 00      0   0  1
  [25] .debug_aranges    PROGBITS        00000000 0006c0 000088 00      0   0  8
  [26] .debug_pubnames   PROGBITS        00000000 000748 000025 00      0   0  1
  [27] .debug_info       PROGBITS        00000000 00076d 00022b 00      0   0  1
  [28] .debug_abbrev     PROGBITS        00000000 000998 000076 00      0   0  1
  [29] .debug_line       PROGBITS        00000000 000a0e 0001bb 00      0   0  1
  [30] .debug_str        PROGBITS        00000000 000bc9 0000bf 01  MS  0   0  1
  [31] .shstrtab         STRTAB          00000000 000c88 000127 00      0   0  1
  [32] .symtab           SYMTAB          00000000 001300 000520 10     33  63  4
  [33] .strtab           STRTAB          00000000 001820 0002d2 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings)
  I (info), L (link order), G (group), x (unknown)
  O (extra OS processing required) o (OS specific), p (processor specific)

There are no section groups in this file.

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x000e0 0x000e0 R E 0x4
  INTERP         0x000114 0x08048114 0x08048114 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x00494 0x00494 R E 0x1000
  LOAD           0x000494 0x08049494 0x08049494 0x00100 0x00104 RW  0x1000
  DYNAMIC        0x0004a8 0x080494a8 0x080494a8 0x000c8 0x000c8 RW  0x4
  NOTE           0x000128 0x08048128 0x08048128 0x00020 0x00020 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4

 Section to Segment mapping:
  Segment Sections...
```

```
   00
   01     .interp
   02     .interp .note.ABI-tag .hash .dynsym .dynstr .gnu.version
          .gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .eh_frame
   03     .ctors .dtors .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag
   06

Dynamic section at offset 0x4a8 contains 20 entries:
  Tag        Type                         Name/Value
 0x00000001 (NEEDED)                     Shared library: [libc.so.6]
 0x0000000c (INIT)                       0x8048278
 0x0000000d (FINI)                       0x8048458
 0x00000004 (HASH)                       0x8048148
 0x00000005 (STRTAB)                     0x80481d4
 0x00000006 (SYMTAB)                     0x8048174
 0x0000000a (STRSZ)                      94 (bytes)
 0x0000000b (SYMENT)                     16 (bytes)
 0x00000015 (DEBUG)                      0x0
 0x00000003 (PLTGOT)                     0x8049574
 0x00000002 (PLTRELSZ)                   16 (bytes)
 0x00000014 (PLTREL)                     REL
 0x00000017 (JMPREL)                     0x8048268
 0x00000011 (REL)                        0x8048260
 0x00000012 (RELSZ)                      8 (bytes)
 0x00000013 (RELENT)                     8 (bytes)
 0x6ffffffe (VERNEED)                    0x8048240
 0x6fffffff (VERNEEDNUM)                 1
 0x6ffffff0 (VERSYM)                     0x8048232
 0x00000000 (NULL)                       0x0

Relocation section '.rel.dyn' at offset 0x260 contains 1 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
08049570  00000506 R_386_GLOB_DAT    00000000   __gmon_start__

Relocation section '.rel.plt' at offset 0x268 contains 2 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
08049580  00000107 R_386_JUMP_SLOT   00000000   puts
08049584  00000207 R_386_JUMP_SLOT   00000000   __libc_start_main

There are no unwind sections in this file.

Symbol table '.dynsym' contains 6 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000   378 FUNC    GLOBAL DEFAULT  UND puts@GLIBC_2.0 (2)
     2: 00000000   230 FUNC    GLOBAL DEFAULT  UND __libc_start_main@GLIBC_2.0 (2)
     3: 0804847c     4 OBJECT  GLOBAL DEFAULT   14 _IO_stdin_used
     4: 00000000     0 NOTYPE  WEAK   DEFAULT  UND _Jv_RegisterClasses
     5: 00000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__

Symbol table '.symtab' contains 82 entries:
   Num:    Value  Size Type    Bind   Vis      Ndx Name
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 08048114     0 SECTION LOCAL  DEFAULT    1
     2: 08048128     0 SECTION LOCAL  DEFAULT    2
     3: 08048148     0 SECTION LOCAL  DEFAULT    3
     4: 08048174     0 SECTION LOCAL  DEFAULT    4
     5: 080481d4     0 SECTION LOCAL  DEFAULT    5
     6: 08048232     0 SECTION LOCAL  DEFAULT    6
     7: 08048240     0 SECTION LOCAL  DEFAULT    7
     8: 08048260     0 SECTION LOCAL  DEFAULT    8
     9: 08048268     0 SECTION LOCAL  DEFAULT    9
    10: 08048278     0 SECTION LOCAL  DEFAULT   10
    11: 08048290     0 SECTION LOCAL  DEFAULT   11
    12: 080482c0     0 SECTION LOCAL  DEFAULT   12
```

```
    13: 08048458     0 SECTION LOCAL  DEFAULT   13
    14: 08048478     0 SECTION LOCAL  DEFAULT   14
    15: 08048490     0 SECTION LOCAL  DEFAULT   15
    16: 08049494     0 SECTION LOCAL  DEFAULT   16
    17: 0804949c     0 SECTION LOCAL  DEFAULT   17
    18: 080494a4     0 SECTION LOCAL  DEFAULT   18
    19: 080494a8     0 SECTION LOCAL  DEFAULT   19
    20: 08049570     0 SECTION LOCAL  DEFAULT   20
    21: 08049574     0 SECTION LOCAL  DEFAULT   21
    22: 08049588     0 SECTION LOCAL  DEFAULT   22
    23: 08049594     0 SECTION LOCAL  DEFAULT   23
    24: 00000000     0 SECTION LOCAL  DEFAULT   24
    25: 00000000     0 SECTION LOCAL  DEFAULT   25
    26: 00000000     0 SECTION LOCAL  DEFAULT   26
    27: 00000000     0 SECTION LOCAL  DEFAULT   27
    28: 00000000     0 SECTION LOCAL  DEFAULT   28
    29: 00000000     0 SECTION LOCAL  DEFAULT   29
    30: 00000000     0 SECTION LOCAL  DEFAULT   30
    31: 00000000     0 SECTION LOCAL  DEFAULT   31
    32: 00000000     0 SECTION LOCAL  DEFAULT   32
    33: 00000000     0 SECTION LOCAL  DEFAULT   33
    34: 00000000     0 FILE    LOCAL  DEFAULT  ABS abi-note.S
    35: 00000000     0 FILE    LOCAL  DEFAULT  ABS ../sysdeps/i386/elf/start
    36: 00000000     0 FILE    LOCAL  DEFAULT  ABS init.c
    37: 00000000     0 FILE    LOCAL  DEFAULT  ABS initfini.c
    38: 00000000     0 FILE    LOCAL  DEFAULT  ABS /build/buildd/glibc-2.3.6
    39: 080482e4     0 FUNC    LOCAL  DEFAULT   12 call_gmon_start
    40: 00000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    41: 08049494     0 OBJECT  LOCAL  DEFAULT   16 __CTOR_LIST__
    42: 0804949c     0 OBJECT  LOCAL  DEFAULT   17 __DTOR_LIST__
    43: 080494a4     0 OBJECT  LOCAL  DEFAULT   18 __JCR_LIST__
    44: 08049594     1 OBJECT  LOCAL  DEFAULT   23 completed.4463
    45: 08049590     0 OBJECT  LOCAL  DEFAULT   22 p.4462
    46: 0804830c     0 FUNC    LOCAL  DEFAULT   12 __do_global_dtors_aux
    47: 0804833b     0 FUNC    LOCAL  DEFAULT   12 frame_dummy
    48: 00000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
    49: 08049498     0 OBJECT  LOCAL  DEFAULT   16 __CTOR_END__
    50: 080494a0     0 OBJECT  LOCAL  DEFAULT   17 __DTOR_END__
    51: 08048490     0 OBJECT  LOCAL  DEFAULT   15 __FRAME_END__
    52: 080494a4     0 OBJECT  LOCAL  DEFAULT   18 __JCR_END__
    53: 0804842c     0 FUNC    LOCAL  DEFAULT   12 __do_global_ctors_aux
    54: 00000000     0 FILE    LOCAL  DEFAULT  ABS initfini.c
    55: 00000000     0 FILE    LOCAL  DEFAULT  ABS /build/buildd/glibc-2.3.6
    56: 00000000     0 FILE    LOCAL  DEFAULT  ABS hw.c
    57: 080494a8     0 OBJECT  LOCAL  HIDDEN   19 _DYNAMIC
    58: 08049494     0 NOTYPE  LOCAL  HIDDEN  ABS __fini_array_end
    59: 08049494     0 NOTYPE  LOCAL  HIDDEN  ABS __fini_array_start
    60: 08049494     0 NOTYPE  LOCAL  HIDDEN  ABS __init_array_end
    61: 08049574     0 OBJECT  LOCAL  HIDDEN   21 _GLOBAL_OFFSET_TABLE_
    62: 08049494     0 NOTYPE  LOCAL  HIDDEN  ABS __init_array_start
    63: 08048478     4 OBJECT  GLOBAL DEFAULT   14 _fp_hw
    64: 0804958c     0 OBJECT  GLOBAL HIDDEN   22 __dso_handle
    65: 080483e1    74 FUNC    GLOBAL DEFAULT   12 __libc_csu_fini
    66: 00000000   378 FUNC    GLOBAL DEFAULT  UND puts@@GLIBC_2.0
    67: 08048278     0 FUNC    GLOBAL DEFAULT   10 _init
    68: 080482c0     0 FUNC    GLOBAL DEFAULT   12 _start
    69: 08048390    81 FUNC    GLOBAL DEFAULT   12 __libc_csu_init
    70: 08049594     0 NOTYPE  GLOBAL DEFAULT  ABS __bss_start
    71: 08048360    47 FUNC    GLOBAL DEFAULT   12 main
    72: 00000000   230 FUNC    GLOBAL DEFAULT  UND __libc_start_main@@GLIBC_
    73: 08049588     0 NOTYPE  WEAK   DEFAULT   22 data_start
    74: 08048458     0 FUNC    GLOBAL DEFAULT   13 _fini
    75: 08049594     0 NOTYPE  GLOBAL DEFAULT  ABS _edata
    76: 08048308     0 FUNC    GLOBAL HIDDEN   12 __i686.get_pc_thunk.bx
    77: 08049598     0 NOTYPE  GLOBAL DEFAULT  ABS _end
    78: 0804847c     4 OBJECT  GLOBAL DEFAULT   14 _IO_stdin_used
    79: 08049588     0 NOTYPE  GLOBAL DEFAULT   22 __data_start
```

```
    80: 00000000     0 NOTYPE  WEAK   DEFAULT  UND _Jv_RegisterClasses
    81: 00000000     0 NOTYPE  WEAK   DEFAULT  UND __gmon_start__

Histogram for bucket list length (total of 3 buckets):
 Length  Number     % of total  Coverage
      0  0          (  0.0%)
      1  1          ( 33.3%)     20.0%
      2  2          ( 66.7%)    100.0%

Version symbols section '.gnu.version' contains 6 entries:
 Addr: 0000000008048232  Offset: 0x000232  Link: 4 (.dynsym)
  000:   0 (*local*)       2 (GLIBC_2.0)     2 (GLIBC_2.0)     1 (*global*)
  004:   0 (*local*)       0 (*local*)

Version needs section '.gnu.version_r' contains 1 entries:
 Addr: 0x0000000008048240  Offset: 0x000240  Link to section: 5 (.dynstr)
  000000: Version: 1  File: libc.so.6  Cnt: 1
  0x0010:   Name: GLIBC_2.0  Flags: none  Version: 2

Notes at offset 0x00000128 with length 0x00000020:
  Owner         Data size       Description
  GNU           0x00000010      NT_VERSION (version)
```

As you can see from this output, a huge amount of useful detail resides in the simple a.out Hello World file -- version information, histograms, multiple tables of various symbol types, and so on. Yes, one can spend a great deal of time learning about executable programs by exploring object files with just the few tools presented here.

In addition to all these sections, the compiler can place debugging information in the object files, and such information can be displayed as well. Type the following command and take some time to see what the compiler is telling you (if you're a debugging program, that is):

```
readelf --debug-dump a.out | less
```

This command produces the output shown in Listing 5. Debugging tools, such as GDB, read in this debugging information, and you can get the tools to display more descriptive labels (for example) than raw address values when disassembling code while it's running under the debugger.

Listing 5. Debugging information in the program

```
The section .debug_aranges contains:

  Length:                   28
  Version:                  2
  Offset into .debug_info:  0
  Pointer Size:             4
  Segment Size:             0

    Address  Length
    080482c0 34
  Length:                   52
  Version:                  2
  Offset into .debug_info:  10b
  Pointer Size:             4
  Segment Size:             0

    Address  Length
    08048308 4
    08048458 18
    08048278 11
```

```
    080482e4 36
  Length:                   44
  Version:                  2
  Offset into .debug_info:  19b
  Pointer Size:             4
  Segment Size:             0

    Address  Length
    08048308 4
    0804846f 6
    0804828d 2

Contents of the .debug_pubnames section:

  Length:                              33
  Version:                             2
  Offset into .debug_info section:     122
  Size of area in .debug_info section: 145

    Offset      Name
    121                 _IO_stdin_used

The section .debug_info contains:

  Compilation Unit @ offset 0x0:
   Length:        118
   Version:       2
   Abbrev Offset: 0
   Pointer Size:  4
 <0><b>: Abbrev Number: 1 (DW_TAG_compile_unit)
     DW_AT_stmt_list   : 0
     DW_AT_low_pc      : 0x80482c0
     DW_AT_high_pc     : 0x80482e2
     DW_AT_name        : ../sysdeps/i386/elf/start.S
     DW_AT_comp_dir    : /build/buildd/glibc-2.3.6/build-tree/glibc-2.3.6/csu
     DW_AT_producer    : GNU AS 2.16.91
     DW_AT_language    : 32769  (MIPS assembler)
  Compilation Unit @ offset 0x7a:
   Length:        141
   Version:       2
   Abbrev Offset: 20
   Pointer Size:  4
 <0><85>: Abbrev Number: 1 (DW_TAG_compile_unit)
     DW_AT_stmt_list   : 0x5b
     DW_AT_high_pc     : 0x80482e4
     DW_AT_low_pc      : 0x80482e4
     DW_AT_producer    : (indirect string, offset: 0x62): GNU C 3.4.6
     DW_AT_language    : 1      (ANSI C)
     DW_AT_name        : (indirect string, offset: 0x0): init.c
     DW_AT_comp_dir    : (indirect string, offset: 0x11): /build/buildd/...
 <1><9f>: Abbrev Number: 2 (DW_TAG_base_type)
     DW_AT_name        : (indirect string, offset: 0x90): unsigned int
     DW_AT_byte_size   : 4
     DW_AT_encoding    : 7      (unsigned)
 <1><a6>: Abbrev Number: 2 (DW_TAG_base_type)
     DW_AT_name        : (indirect string, offset: 0x54): unsigned char
     DW_AT_byte_size   : 1
     DW_AT_encoding    : 8      (unsigned char)
 <1><ad>: Abbrev Number: 2 (DW_TAG_base_type)
     DW_AT_name        : (indirect string, offset: 0x9d): short unsigned int
     DW_AT_byte_size   : 2
     DW_AT_encoding    : 7      (unsigned)
 <1><b4>: Abbrev Number: 2 (DW_TAG_base_type)
     DW_AT_name        : (indirect string, offset: 0x8b): long unsigned int
     DW_AT_byte_size   : 4
     DW_AT_encoding    : 7      (unsigned)
 <1><bb>: Abbrev Number: 2 (DW_TAG_base_type)
```

```
    DW_AT_name        : (indirect string, offset: 0x56): signed char
    DW_AT_byte_size   : 1
    DW_AT_encoding    : 6      (signed char)
 <1><c2>: Abbrev Number: 2 (DW_TAG_base_type)
    DW_AT_name        : (indirect string, offset: 0x7): short int
    DW_AT_byte_size   : 2
    DW_AT_encoding    : 5      (signed)
 <1><c9>: Abbrev Number: 3 (DW_TAG_base_type)
    DW_AT_name        : int
    DW_AT_byte_size   : 4
    DW_AT_encoding    : 5      (signed)
 <1><d0>: Abbrev Number: 2 (DW_TAG_base_type)
    DW_AT_name        : (indirect string, offset: 0x46): long long int
    DW_AT_byte_size   : 8
    DW_AT_encoding    : 5      (signed)
 <1><d7>: Abbrev Number: 2 (DW_TAG_base_type)
    DW_AT_name        : (indirect string, offset: 0x86): long long unsigned int
    DW_AT_byte_size   : 8
    DW_AT_encoding    : 7      (unsigned)
 <1><de>: Abbrev Number: 2 (DW_TAG_base_type)
    DW_AT_name        : (indirect string, offset: 0x4b): long int
    DW_AT_byte_size   : 4
    DW_AT_encoding    : 5      (signed)
 <1><e5>: Abbrev Number: 2 (DW_TAG_base_type)
    DW_AT_name        : (indirect string, offset: 0x90): unsigned int
    DW_AT_byte_size   : 4
    DW_AT_encoding    : 7      (unsigned)
 <1><ec>: Abbrev Number: 2 (DW_TAG_base_type)
    DW_AT_name        : (indirect string, offset: 0x5d): char
    DW_AT_byte_size   : 1
    DW_AT_encoding    : 6      (signed char)
 <1><f3>: Abbrev Number: 4 (DW_TAG_variable)
    DW_AT_name        : (indirect string, offset: 0xb0): _IO_stdin_used
    DW_AT_decl_file   : 1
    DW_AT_decl_line   : 25
    DW_AT_type        : <105>
    DW_AT_external    : 1
    DW_AT_location    : 5 byte block: 3 7c 84 4 8      (DW_OP_addr: 804847c)
 <1><105>: Abbrev Number: 5 (DW_TAG_const_type)
    DW_AT_type        : <c9>
  Compilation Unit @ offset 0x10b:
   Length:        140
   Version:       2
   Abbrev Offset: 86
   Pointer Size:  4
 <0><116>: Abbrev Number: 1 (DW_TAG_compile_unit)
    DW_AT_stmt_list   : 0x82
    DW_AT_name        : /build/buildd/glibc-2.3.6/build-tree/i386-libc/csu/crti.S
    DW_AT_comp_dir    : /build/buildd/glibc-2.3.6/build-tree/glibc-2.3.6/csu
    DW_AT_producer    : GNU AS 2.16.91
    DW_AT_language    : 32769  (MIPS assembler)
  Compilation Unit @ offset 0x19b:
   Length:        140
   Version:       2
   Abbrev Offset: 102
   Pointer Size:  4
 <0><1a6>: Abbrev Number: 1 (DW_TAG_compile_unit)
    DW_AT_stmt_list   : 0x12f
    DW_AT_name        : /build/buildd/glibc-2.3.6/build-tree/i386-libc/csu/crtn.S
    DW_AT_comp_dir    : /build/buildd/glibc-2.3.6/build-tree/glibc-2.3.6/csu
    DW_AT_producer    : GNU AS 2.16.91
    DW_AT_language    : 32769  (MIPS assembler)

Contents of the .debug_abbrev section:

  Number TAG
   1      DW_TAG_compile_unit    [no children]
```

```
    DW_AT_stmt_list    DW_FORM_data4
    DW_AT_low_pc       DW_FORM_addr
    DW_AT_high_pc      DW_FORM_addr
    DW_AT_name         DW_FORM_string
    DW_AT_comp_dir     DW_FORM_string
    DW_AT_producer     DW_FORM_string
    DW_AT_language     DW_FORM_data2
  Number TAG
   1      DW_TAG_compile_unit    [has children]
    DW_AT_stmt_list    DW_FORM_data4
    DW_AT_high_pc      DW_FORM_addr
    DW_AT_low_pc       DW_FORM_addr
    DW_AT_producer     DW_FORM_strp
    DW_AT_language     DW_FORM_data1
    DW_AT_name         DW_FORM_strp
    DW_AT_comp_dir     DW_FORM_strp
   2      DW_TAG_base_type    [no children]
    DW_AT_name         DW_FORM_strp
    DW_AT_byte_size    DW_FORM_data1
    DW_AT_encoding     DW_FORM_data1
   3      DW_TAG_base_type    [no children]
    DW_AT_name         DW_FORM_string
    DW_AT_byte_size    DW_FORM_data1
    DW_AT_encoding     DW_FORM_data1
   4      DW_TAG_variable    [no children]
    DW_AT_name         DW_FORM_strp
    DW_AT_decl_file    DW_FORM_data1
    DW_AT_decl_line    DW_FORM_data1
    DW_AT_type         DW_FORM_ref4
    DW_AT_external     DW_FORM_flag
    DW_AT_location     DW_FORM_block1
   5      DW_TAG_const_type    [no children]
    DW_AT_type         DW_FORM_ref4
  Number TAG
   1      DW_TAG_compile_unit    [no children]
    DW_AT_stmt_list    DW_FORM_data4
    DW_AT_name         DW_FORM_string
    DW_AT_comp_dir     DW_FORM_string
    DW_AT_producer     DW_FORM_string
    DW_AT_language     DW_FORM_data2
  Number TAG
   1      DW_TAG_compile_unit    [no children]
    DW_AT_stmt_list    DW_FORM_data4
    DW_AT_name         DW_FORM_string
    DW_AT_comp_dir     DW_FORM_string
    DW_AT_producer     DW_FORM_string
    DW_AT_language     DW_FORM_data2


Dump of debug contents of section .debug_line:

  Length:                      87
  DWARF Version:               2
  Prologue Length:             50
  Minimum Instruction Length:  1
  Initial value of 'is_stmt':  1
  Line Base:                   -5
  Line Range:                  14
  Opcode Base:                 13
  (Pointer size:               4)

 Opcodes:
  Opcode 1 has 0 args
  Opcode 2 has 1 args
  Opcode 3 has 1 args
  Opcode 4 has 1 args
  Opcode 5 has 1 args
```

```
 Opcode 6 has 0 args
 Opcode 7 has 0 args
 Opcode 8 has 0 args
 Opcode 9 has 1 args
 Opcode 10 has 0 args
 Opcode 11 has 0 args
 Opcode 12 has 1 args

The Directory Table:
 ../sysdeps/i386/elf

The File Name Table:
 Entry Dir     Time    Size    Name
 1     1       0       0       start.S

Line Number Statements:
 Extended opcode 2: set Address to 0x80482c0
 Advance Line by 64 to 65
 Copy
 Special opcode 38: advance Address by 2 to 0x80482c2 and Line by 5 to 70
 Special opcode 20: advance Address by 1 to 0x80482c3 and Line by 1 to 71
 Special opcode 39: advance Address by 2 to 0x80482c5 and Line by 6 to 77
 Special opcode 48: advance Address by 3 to 0x80482c8 and Line by 1 to 78
 Special opcode 24: advance Address by 1 to 0x80482c9 and Line by 5 to 83
 Special opcode 21: advance Address by 1 to 0x80482ca and Line by 2 to 85
 Advance Line by 24 to 109
 Special opcode 19: advance Address by 1 to 0x80482cb and Line by 0 to 109
 Special opcode 76: advance Address by 5 to 0x80482d0 and Line by 1 to 110
 Special opcode 77: advance Address by 5 to 0x80482d5 and Line by 2 to 112
 Special opcode 20: advance Address by 1 to 0x80482d6 and Line by 1 to 113
 Special opcode 21: advance Address by 1 to 0x80482d7 and Line by 2 to 115
 Special opcode 79: advance Address by 5 to 0x80482dc and Line by 4 to 119
 Special opcode 78: advance Address by 5 to 0x80482e1 and Line by 3 to 122
 Advance PC by 1 to 0x80482e2
 Extended opcode 1: End of Sequence


 Length:                      35
 DWARF Version:               2
 Prologue Length:             29
 Minimum Instruction Length:  1
 Initial value of 'is_stmt':  1
 Line Base:                   -5
 Line Range:                  14
 Opcode Base:                 13
 (Pointer size:               4)

Opcodes:
 Opcode 1 has 0 args
 Opcode 2 has 1 args
 Opcode 3 has 1 args
 Opcode 4 has 1 args
 Opcode 5 has 1 args
 Opcode 6 has 0 args
 Opcode 7 has 0 args
 Opcode 8 has 0 args
 Opcode 9 has 1 args
 Opcode 10 has 0 args
 Opcode 11 has 0 args
 Opcode 12 has 1 args

The Directory Table is empty.

The File Name Table:
 Entry Dir     Time    Size    Name
 1     0       0       0       init.c
```

```
 Line Number Statements:

  Length:                      169
  DWARF Version:               2
  Prologue Length:             80
  Minimum Instruction Length:  1
  Initial value of 'is_stmt':  1
  Line Base:                   -5
  Line Range:                  14
  Opcode Base:                 13
  (Pointer size:               4)

 Opcodes:
  Opcode 1 has 0 args
  Opcode 2 has 1 args
  Opcode 3 has 1 args
  Opcode 4 has 1 args
  Opcode 5 has 1 args
  Opcode 6 has 0 args
  Opcode 7 has 0 args
  Opcode 8 has 0 args
  Opcode 9 has 1 args
  Opcode 10 has 0 args
  Opcode 11 has 0 args
  Opcode 12 has 1 args

 The Directory Table:
  /build/buildd/glibc-2.3.6/build-tree/i386-libc/csu

 The File Name Table:
  Entry Dir     Time    Size    Name
  1     1       0       0       crti.S

 Line Number Statements:
  Extended opcode 2: set Address to 0x8048308
  Advance Line by 64 to 65
  Copy
  Special opcode 48: advance Address by 3 to 0x804830b and Line by 1 to 66
  Advance PC by 1 to 0x804830c
  Extended opcode 1: End of Sequence

  Extended opcode 2: set Address to 0x8048458
  Advance Line by 46 to 47
  Copy
  Special opcode 20: advance Address by 1 to 0x8048459 and Line by 1 to 48
  Special opcode 34: advance Address by 2 to 0x804845b and Line by 1 to 49
  Special opcode 20: advance Address by 1 to 0x804845c and Line by 1 to 50
  Special opcode 76: advance Address by 5 to 0x8048461 and Line by 1 to 51
  Special opcode 90: advance Address by 6 to 0x8048467 and Line by 1 to 52
  Advance PC by 3 to 0x804846a
  Extended opcode 1: End of Sequence

  Extended opcode 2: set Address to 0x8048278
  Advance Line by 31 to 32
  Copy
  Special opcode 20: advance Address by 1 to 0x8048279 and Line by 1 to 33
  Special opcode 34: advance Address by 2 to 0x804827b and Line by 1 to 34
  Special opcode 48: advance Address by 3 to 0x804827e and Line by 1 to 35
  Advance PC by 5 to 0x8048283
  Extended opcode 1: End of Sequence

  Extended opcode 2: set Address to 0x80482e4
  Advance Line by 10 to 11
  Copy
  Special opcode 20: advance Address by 1 to 0x80482e5 and Line by 1 to 12
  Special opcode 34: advance Address by 2 to 0x80482e7 and Line by 1 to 13
  Special opcode 20: advance Address by 1 to 0x80482e8 and Line by 1 to 14
```

```
 Special opcode 76: advance Address by 5 to 0x80482ed and Line by 1 to 15
 Special opcode 90: advance Address by 6 to 0x80482f3 and Line by 1 to 16
 Special opcode 48: advance Address by 3 to 0x80482f6 and Line by 1 to 17
 Special opcode 90: advance Address by 6 to 0x80482fc and Line by 1 to 18
 Special opcode 34: advance Address by 2 to 0x80482fe and Line by 1 to 19
 Special opcode 34: advance Address by 2 to 0x8048300 and Line by 1 to 20
 Special opcode 35: advance Address by 2 to 0x8048302 and Line by 2 to 22
 Special opcode 48: advance Address by 3 to 0x8048305 and Line by 1 to 23
 Special opcode 20: advance Address by 1 to 0x8048306 and Line by 1 to 24
 Special opcode 20: advance Address by 1 to 0x8048307 and Line by 1 to 25
 Advance PC by 1 to 0x8048308
 Extended opcode 1: End of Sequence


 Length:                      136
 DWARF Version:               2
 Prologue Length:             80
 Minimum Instruction Length:  1
 Initial value of 'is_stmt':  1
 Line Base:                   -5
 Line Range:                  14
 Opcode Base:                 13
 (Pointer size:               4)

Opcodes:
 Opcode 1 has 0 args
 Opcode 2 has 1 args
 Opcode 3 has 1 args
 Opcode 4 has 1 args
 Opcode 5 has 1 args
 Opcode 6 has 0 args
 Opcode 7 has 0 args
 Opcode 8 has 0 args
 Opcode 9 has 1 args
 Opcode 10 has 0 args
 Opcode 11 has 0 args
 Opcode 12 has 1 args

The Directory Table:
 /build/buildd/glibc-2.3.6/build-tree/i386-libc/csu

The File Name Table:
 Entry Dir     Time    Size    Name
 1     1       0       0       crtn.S

Line Number Statements:
 Extended opcode 2: set Address to 0x8048308
 Advance Line by 33 to 34
 Copy
 Special opcode 48: advance Address by 3 to 0x804830b and Line by 1 to 35
 Advance PC by 1 to 0x804830c
 Extended opcode 1: End of Sequence

 Extended opcode 2: set Address to 0x804846f
 Advance Line by 18 to 19
 Copy
 Special opcode 48: advance Address by 3 to 0x8048472 and Line by 1 to 20
 Special opcode 20: advance Address by 1 to 0x8048473 and Line by 1 to 21
 Special opcode 20: advance Address by 1 to 0x8048474 and Line by 1 to 22
 Advance PC by 1 to 0x8048475
 Extended opcode 1: End of Sequence

 Extended opcode 2: set Address to 0x804828d
 Advance Line by 9 to 10
 Copy
 Special opcode 20: advance Address by 1 to 0x804828e and Line by 1 to 11
 Advance PC by 1 to 0x804828f
```

```
  Extended opcode 1: End of Sequence


Contents of the .debug_str section:

  0x00000000 696e6974 2e630073 686f7274 20696e74 init.c.short int
  0x00000010 002f6275 696c642f 6275696c 64642f67 ./build/buildd/g
  0x00000020 6c696263 2d322e33 2e362f62 75696c64 libc-2.3.6/build
  0x00000030 2d747265 652f676c 6962632d 322e332e -tree/glibc-2.3.
  0x00000040 362f6373 75006c6f 6e67206c 6f6e6720 6/csu.long long
  0x00000050 696e7400 756e7369 676e6564 20636861 int.unsigned cha
  0x00000060 7200474e 55204320 332e342e 36202855 r.GNU C 3.4.6 (U
  0x00000070 62756e74 7520332e 342e362d 31756275 buntu 3.4.6-1ubu
  0x00000080 6e747532 29006c6f 6e67206c 6f6e6720 ntu2).long long
  0x00000090 756e7369 676e6564 20696e74 0073686f unsigned int.sho
  0x000000a0 72742075 6e736967 6e656420 696e7400 rt unsigned int.
  0x000000b0 5f494f5f 73746469 6e5f7573 656400   _IO_stdin_used.
```

Executable files are object files

In the UNIX world, executable files *are* object files, and you can examine them as you did the a.out file. It is a useful exercise to change to the /bin or /local/bin directory and run `nm`, `objdump`, and `readelf` over some of your most commonly used commands, such as `pwd`, `ps`, `cat`, or `rm`. Often when you're writing a program that requires a certain functionality that one of the standard tools has, it's useful to see how those tools actually do their work by simply running `objdump -d <command>` over it.

If you're so inclined to work on compilers and other language tools, you'll find that time spent studying the various object files that make up your computer's system is time well spent. A UNIX operating system has many layers, and the layers that the tools examining its object files expose are close to the hardware. You can get a real feel for the system in this way.

Conclusion

Exploring object files can greatly deepen your knowledge of the UNIX operating system and provide greater insight into how the software is actually assembled from source code. I encourage you to study the output of the object file tools described in this article by running them over the programs found in the /bin or /local/bin directories on your system and seek out system documentation that your hardware manufacturer provides.

Related topics

- Executable file formats: Visit Wikipedia to learn more about executable file formats.
- Executable and Linking Format (ELF): Visit the University of California-Davis site for more information.
- AIX and UNIX articles: Check out other articles written by William Zimmerly.
- IBM trial software: Build your next development project with software for download directly from developerWorks.
- AIX 5L Wiki: A collaborative environment for technical information related to AIX.