
哈爾濱工業大學

計算機系統

大作業

題 目 程序人生-Hello's P2P

專 業 計算機大類

學 號 1190400818

班 級 1903009

學 生 蔣文祺

指導教師 吳銳

計算機科學與技術學院
2021 年 5 月

摘 要

通过跟踪 hello 程序的生命周期来开始对系统的学习--从它被程序员创建开始，到在系统上运行，输出简单的消息，然后终止。将沿着这个程序的生命周期，简要地介绍一些逐步出现的关键概念、专业术语和组成部分。后面的章节将围绕这些内容展开。

关键词：计算机系统；虚拟内存；编译原理；进程管理；

目 录

第 1 章 概述.....	- 4 -
1.1 HELLO 简介.....	- 4 -
1.2 环境与工具.....	- 4 -
1.3 中间结果.....	- 4 -
1.4 本章小结.....	- 4 -
第 2 章 预处理.....	- 5 -
2.1 预处理的概念与作用.....	- 5 -
2.2 在 UBUNTU 下预处理的命令.....	- 5 -
2.3 HELLO 的预处理结果解析.....	- 5 -
2.4 本章小结.....	- 5 -
第 3 章 编译.....	- 6 -
3.1 编译的概念与作用.....	- 6 -
3.2 在 UBUNTU 下编译的命令.....	- 6 -
3.3 HELLO 的编译结果解析.....	- 6 -
3.4 本章小结.....	- 6 -
第 4 章 汇编.....	- 7 -
4.1 汇编的概念与作用.....	- 7 -
4.2 在 UBUNTU 下汇编的命令.....	- 7 -
4.3 可重定位目标 ELF 格式.....	- 7 -
4.4 HELLO.O 的结果解析.....	- 7 -
4.5 本章小结.....	- 7 -
第 5 章 链接.....	- 8 -
5.1 链接的概念与作用.....	- 8 -
5.2 在 UBUNTU 下链接的命令.....	- 8 -
5.3 可执行目标文件 HELLO 的格式.....	- 8 -
5.4 HELLO 的虚拟地址空间.....	- 8 -
5.5 链接的重定位过程分析.....	- 8 -
5.6 HELLO 的执行流程.....	- 8 -
5.7 HELLO 的动态链接分析.....	- 8 -
5.8 本章小结.....	- 9 -
第 6 章 HELLO 进程管理.....	- 10 -
6.1 进程的概念与作用.....	- 10 -
6.2 简述壳 SHELL-BASH 的作用与处理流程.....	- 10 -
6.3 HELLO 的 FORK 进程创建过程.....	- 10 -
6.4 HELLO 的 EXECVE 过程.....	- 10 -
6.5 HELLO 的进程执行.....	- 10 -

6.6 HELLO 的异常与信号处理.....	- 10 -
6.7 本章小结.....	- 10 -
第 7 章 HELLO 的存储管理.....	- 11 -
7.1 HELLO 的存储器地址空间.....	- 11 -
7.2 INTEL 逻辑地址到线性地址的变换-段式管理.....	- 11 -
7.3 HELLO 的线性地址到物理地址的变换-页式管理.....	- 11 -
7.4 TLB 与四级页表支持下的 VA 到 PA 的变换.....	- 11 -
7.5 三级 CACHE 支持下的物理内存访问.....	- 11 -
7.6 HELLO 进程 FORK 时的内存映射.....	- 11 -
7.7 HELLO 进程 EXECVE 时的内存映射.....	- 11 -
7.8 缺页故障与缺页中断处理.....	- 11 -
7.9 动态存储分配管理.....	- 11 -
7.10 本章小结.....	- 12 -
第 8 章 HELLO 的 IO 管理.....	- 13 -
8.1 LINUX 的 IO 设备管理方法.....	- 13 -
8.2 简述 UNIX IO 接口及其函数.....	- 13 -
8.3 PRINTF 的实现分析.....	- 13 -
8.4 GETCHAR 的实现分析.....	- 13 -
8.5 本章小结.....	- 13 -
结论.....	- 14 -
附件.....	- 15 -
参考文献.....	- 16 -

.第 1 章 概述

.1.1 Hello 简介

使用 GNU 编译系统构造示例程序，在 shell 中用 gcc 将示例程序 ASCII 码源文件翻译成可执行目标文件（首先运行 C 预处理器，将 C 源程序 hello.c 翻译成一个 ASCII 码的中间文件 hello.i；接下来，驱动程序运行 C 编译器，将 hello.i 翻译成一个 ASCII 汇编语言文件 hello.s；然后，驱动程序运行汇编器，将 hello.s 翻译成一个可重定位目标文件 hello.o；最后，驱动程序运行链接器，将 hello.o 和一些必要的系统目标文件组合起来，创建一个可执行目标文件 hello）。

要运行可执行文件 hello，在 shell 输入名字：./hello，shell 调用加载器，它将可执行文件 hello 中的代码和数据复制到内存，然后将控制转移到这个程序的开头。

一旦目标文件 hello 中的代码和数据被加载到主存，处理器就开始执行 hello 程序的 main 程序中的机器语言指令。这些指令将目标字符串中的字节从主存复制到寄存器文件，再从寄存器文件中复制到显示设备，最终显示在屏幕上。

这里有两个并发的进程：shell 进程和 hello 进程。hello 进程终止后，操作系统恢复 shell 进程的上下文，并将控制权传回给它，shell 进程会等待下一个命令行输入。

.1.2 环境与工具

.1.2.1 硬件环境

Intel Core i7-8550U CPU @1.80GHz 2:00GHz;
16.0GB RAM;
256 GB Disk

.1.2.2 软件环境

Ubuntu 20.04 LTS 64 位（直接双系统，没有虚拟机）

.1.2.3 开发工具

CodeBlocks 64 位；vi/vim/gedit+gcc；edb-debugger

.1.3 中间结果

hello.i：C 预处理器产生的一个 ASCII 码的中间文件，用于分析预处理过程。

hello.s：C 编译器产生的一个 ASCII 汇编语言文件，用于分析编译的过程。

hello.o：汇编器产生的可重定位目标程序，用于分析汇编的过程。

hello：链接器产生的可执行目标文件，用于分析链接的过程。

hello.txt：hello 的反汇编文件，用于分析可执行目标文件 hello。

.1.4 本章小结

通过跟踪 hello 程序的生命周期来开始对系统的学习--从它被程序员创建开始，到在系统上运行，输出简单的消息，然后终止。将沿着这个程序的生命周期，简要地介绍一些逐步出现的关键概念、专业术语和组成部分。后面的章节将围绕这些内容展开。

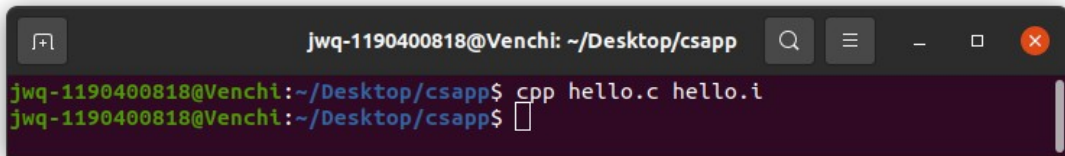
(第 1 章 0.5 分)

.第 2 章 预处理

.2.1 预处理的概念与作用

预处理器（cpp）根据以字符#开头的命令，修改原始的 C 程序。比如 hello.h 中的#include <stdio.h>命令告诉预处理器读取系统头文件 stdio.h 的内容，并把它直接插入程序文本中。结果就得到了另一个 C 程序，通常是以.i 作为文件扩展名。

.2.2 在 Ubuntu 下预处理的命令



```
jwq-1190400818@Venchi: ~/Desktop/csapp
jwq-1190400818@Venchi:~/Desktop/csapp$ cpp hello.c hello.i
jwq-1190400818@Venchi:~/Desktop/csapp$
```

.2.3 Hello 的预处理结果解析



```
1 # 1 "hello.c"
2 # 1 "<built-in>"
3 # 1 "<command-line>"
4 # 31 "<command-line>"
5 # 1 "/usr/include/stdc-predef.h" 1 3 4
6 # 32 "<command-line>" 2
7 # 1 "hello.c"
8
9
10
11
12
13 # 1 "/usr/include/stdio.h" 1 3 4
14 # 27 "/usr/include/stdio.h" 3 4
15 # 1 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 1 3 4
16 # 33 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 3 4
17 # 1 "/usr/include/features.h" 1 3 4
18 # 461 "/usr/include/features.h" 3 4
19 # 1 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 1 3 4
20 # 452 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 3 4
21 # 1 "/usr/include/x86_64-linux-gnu/bits/wordsize.h" 1 3 4
22 # 453 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 2 3 4
23 # 1 "/usr/include/x86_64-linux-gnu/bits/long-double.h" 1 3 4
24 # 454 "/usr/include/x86_64-linux-gnu/sys/cdefs.h" 2 3 4
25 # 462 "/usr/include/features.h" 2 3 4
26 # 485 "/usr/include/features.h" 3 4
27 # 1 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 1 3 4
28 # 10 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 3 4
29 # 1 "/usr/include/x86_64-linux-gnu/gnu/stubs-64.h" 1 3 4
30 # 11 "/usr/include/x86_64-linux-gnu/gnu/stubs.h" 2 3 4
31 # 486 "/usr/include/features.h" 2 3 4
32 # 34 "/usr/include/x86_64-linux-gnu/bits/libc-header-start.h" 2 3 4
33 # 28 "/usr/include/stdio.h" 2 3 4
34
35
36
37
38
39 # 1 "/usr/lib/gcc/x86_64-linux-gnu/9/include/stddef.h" 1 3 4
40 # 209 "/usr/lib/gcc/x86_64-linux-gnu/9/include/stddef.h" 3 4
```



```
3022
3023
3024
3025
3026
3027
3028
3029 extern int rpmatch (const char *__response) __attribute__ ((__nothrow__ , __leaf__))
    __attribute__ ((__nonnull__ (1)));
3030 # 957 "/usr/include/stdlib.h" 3 4
3031 extern int getsubopt (char **__restrict __optionp,
3032     char *const *__restrict __tokens,
3033     char **__restrict __valuep)
3034     __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (1, 2, 3))) ;
3035 # 1003 "/usr/include/stdlib.h" 3 4
3036 extern int getloadavg (double __loadavg[], int __nelem)
3037     __attribute__ ((__nothrow__ , __leaf__)) __attribute__ ((__nonnull__ (1)));
3038 # 1013 "/usr/include/stdlib.h" 3 4
3039 # 1 "/usr/include/x86_64-linux-gnu/bits/stdlib-float.h" 1 3 4
3040 # 1014 "/usr/include/stdlib.h" 2 3 4
3041 # 1023 "/usr/include/stdlib.h" 3 4
3042
3043 # 9 "hello.c" 2
3044
3045
3046 # 10 "hello.c"
3047 int main(int argc, char *argv[]){
3048     int i;
3049
3050     if(argc!=4){
3051         printf("Hello 1190400818 蒋文祺!\n");
3052         exit(1);
3053     }
3054     for(i=0;i<8;i++){
3055         printf("Hello %s %s\n", argv[1], argv[2]);
3056         sleep(atoi(argv[3]));
3057     }
3058     getchar();
3059     return 0;
3060 }
```

2.4 本章小结

预处理器实质是扩展宏

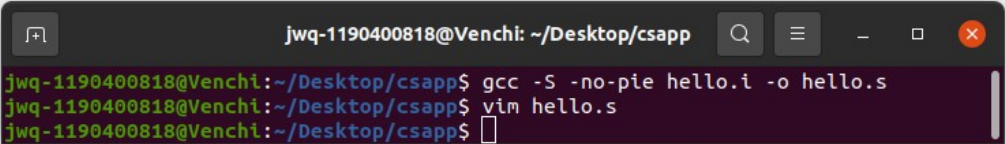
(第2章 0.5分)

.第 3 章 编译

.3.1 编译的概念与作用

编译器 cc1 将文本文件 hello.i 翻译成文本文件 hello.s，它包含一个汇编语言程序。

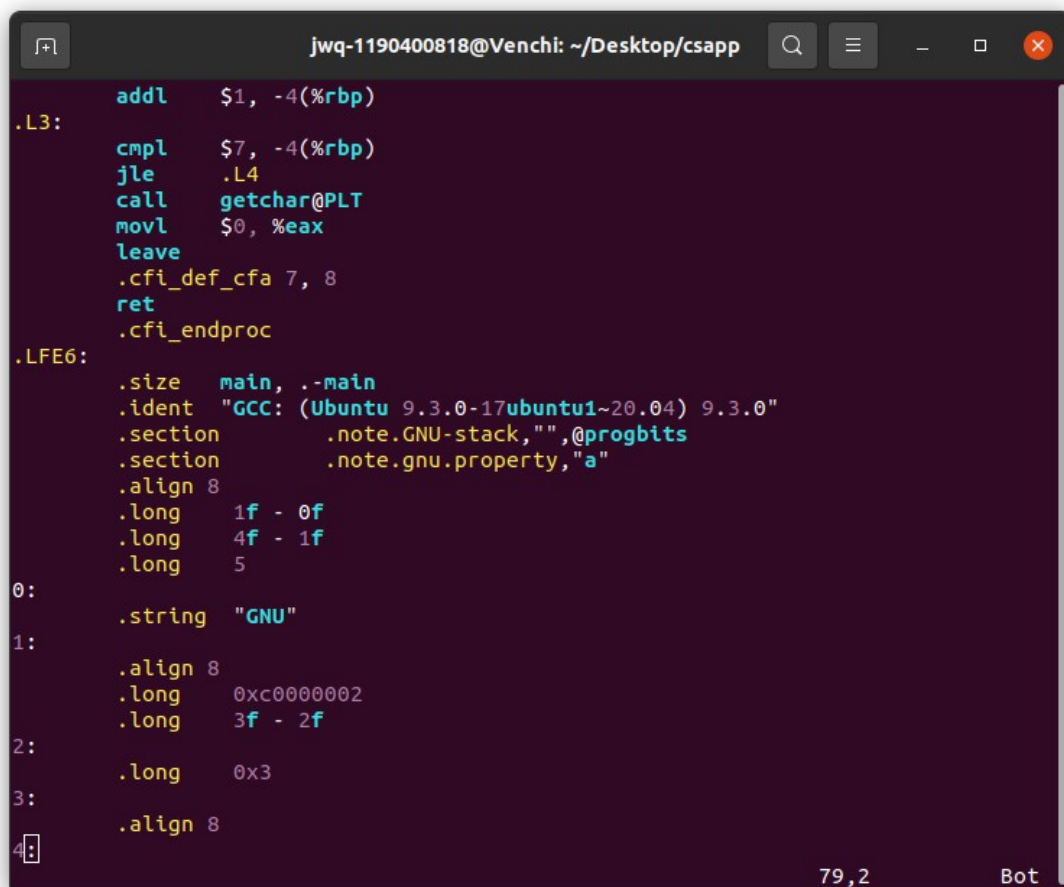
.3.2 在 Ubuntu 下编译的命令

A terminal window with a dark background and light text. The title bar shows the user 'jwq-1190400818@Venchi' and the current directory '~/Desktop/csapp'. The terminal contains three lines of text: the first line shows the command 'gcc -S -no-pie hello.i -o hello.s' being executed; the second line shows the command 'vim hello.s' being entered; the third line shows the prompt 'jwq-1190400818@Venchi:~/Desktop/csapp\$' with a cursor at the end.

```
jwq-1190400818@Venchi: ~/Desktop/csapp
jwq-1190400818@Venchi:~/Desktop/csapp$ gcc -S -no-pie hello.i -o hello.s
jwq-1190400818@Venchi:~/Desktop/csapp$ vim hello.s
jwq-1190400818@Venchi:~/Desktop/csapp$
```

.3.3 Hello 的编译结果解析

```
jwq-1190400818@Venchi: ~/Desktop/csapp
.file "hello.c"
.text
.section .rodata
.LC0:
.string "Hello 1190400818 \350\222\213\346\226\207\347\245\272\357\274\2
01"
.LC1:
.string "Hello %s %s\n"
.text
.globl main
.type main, @function
main:
.LFB6:
.cfi_startproc
endbr64
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $32, %rsp
movl %edi, -20(%rbp)
movq %rsi, -32(%rbp)
cmpl $4, -20(%rbp)
je .L2
leaq .LC0(%rip), %rdi
call puts@PLT
movl $1, %edi
call exit@PLT
.L2:
movl $0, -4(%rbp)
jmp .L3
.L4:
movq -32(%rbp), %rax
addq $16, %rax
movq (%rax), %rdx
movq -32(%rbp), %rax
addq $8, %rax
movq (%rax), %rax
movq %rax, %rsi
leaq .LC1(%rip), %rdi
movl $0, %eax
call printf@PLT
movq -32(%rbp), %rax
addq $24, %rax
movq (%rax), %rax
movq %rax, %rdi
call atoi@PLT
movl %eax, %edi
call sleep@PLT
addl $1, -4(%rbp)
.L3:
cmpl $7, -4(%rbp)
jle .L4
1,1-8 Top
```



```
jwq-1190400818@Venchi: ~/Desktop/csapp

    addl    $1, -4(%rbp)
.L3:
    cmpl    $7, -4(%rbp)
    jle     .L4
    call    getchar@PLT
    movl    $0, %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE6:
    .size    main, .-main
    .ident   "GCC: (Ubuntu 9.3.0-17ubuntu1~20.04) 9.3.0"
    .section .note.gnu-stack,"",@progbits
    .section .note.gnu.property,"a"
    .align   8
    .long    1f - 0f
    .long    4f - 1f
    .long    5
0:
    .string  "GNU"
1:
    .align   8
    .long    0xc0000002
    .long    3f - 2f
2:
    .long    0x3
3:
    .align   8
4:
```

79,2 Bot

3.3.1 数据：常量、变量(全局/局部/静态)、表达式、类型、宏

```
movl    %edi, -20(%rbp)
movq    %rsi, -32(%rbp)
movl    $1, %edi
movq    -32(%rbp), %rax
movq    (%rax), %rdx
movq    -32(%rbp), %rax
movq    (%rax), %rax
movq    %rax, %rsi
movl    $0, %eax
movq    -32(%rbp), %rax
movq    (%rax), %rax
movq    %rax, %rdi
```

```
movl    %eax, %edi
movl    $0, %eax
```

3.3.2 赋值 = ,逗号操作符, 赋初值/不赋初值

.LC0:

```
.string  "Hello 1190400818 \350\222\213\346\226\207\347\245\272\357\274\201"
```

.LC1:

```
.string  "Hello %s %s\n"
.text
.globl   main
.type    main, @function
```

3.3.3 数组/指针/结构操作: A[i] &v *p s.id p→id

```
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
cmpl    $7, -4(%rbp)
```

3.3.4 控制转移: if/else switch for while do/while ?: continue break

```
cmpl    $4, -20(%rbp)
je      .L2
leaq    .LC0(%rip), %rdi
call    puts@PLT
call    exit@PLT
```

3.3.5 函数操作: 参数传递(地址/值)、函数调用()、函数返回 return

```
.cfi_startproc
endbr64
pushq   %rbp

movq    %rsp, %rbp
.cfi_def_cfa_register 6
jmp     .L3
leaq    .LC1(%rip), %rdi
movl    $0, %eax
call    printf@PLT
call    atoi@PLT
call    sleep@PLT

jle     .L4
call    getchar@PLT
```

```
leave
.cfi_def_cfa 7, 8
ret
.cfi_endproc
```

3.3.6 算术操作：+ - * / % ++ -- 取正/负+- 复合 “+=”等

```
subq    $32, %rsp
addq    $16, %rax
addq    $8, %rax
addq    $24, %rax
addl$1, -4(%rbp)
```

3.4 本章小结

汇编代码非常接近于机器代码。与机器代码的二进制格式相比，汇编代码的主要特点是它用可读性更好的文本格式表示。理解汇编代码以及它与原始 C 代码的联系，是理解计算机如何执行程序的关键一步。

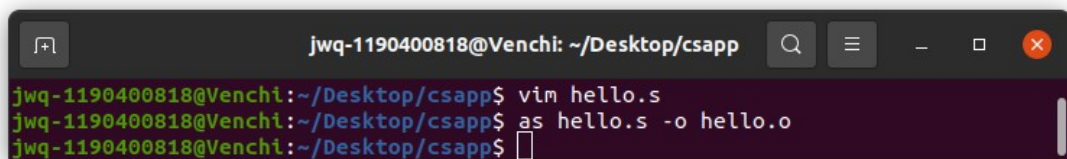
(第 3 章 2 分)

.第 4 章 汇编

.4.1 汇编的概念与作用

汇编器将 `hello.s` 翻译成机器语言指令，把这些指令打包成一种叫做可重定位目标程序的格式，并将结果保存在文件 `hello.o` 中。`hello.o` 文件是一个二进制文件。

.4.2 在 Ubuntu 下汇编的命令

A terminal window with a dark background and light green text. The window title is 'jq-1190400818@Venchi: ~/Desktop/csapp'. It shows three lines of commands: 'jq-1190400818@Venchi:~/Desktop/csapp\$ vim hello.s', 'jq-1190400818@Venchi:~/Desktop/csapp\$ as hello.s -o hello.o', and 'jq-1190400818@Venchi:~/Desktop/csapp\$' followed by a cursor.

```
jq-1190400818@Venchi: ~/Desktop/csapp
jq-1190400818@Venchi:~/Desktop/csapp$ vim hello.s
jq-1190400818@Venchi:~/Desktop/csapp$ as hello.s -o hello.o
jq-1190400818@Venchi:~/Desktop/csapp$
```

.4.3 可重定位目标 elf 格式

```

jq-1190400818@Venchi: ~/Desktop/csapp
jq-1190400818@Venchi:~/Desktop/csapp$ readelf -a hello.o
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00
  Class:                               ELF64
  Data:                               2's complement, little endian
  Version:                             1 (current)
  OS/ABI:                               UNIX - System V
  ABI Version:                           0
  Type:                                REL (Relocatable file)
  Machine:                             Advanced Micro Devices X86-64
  Version:                             0x1
  Entry point address:                  0x0
  Start of program headers:              0 (bytes into file)
  Start of section headers:             1224 (bytes into file)
  Flags:                                0x0
  Size of this header:                   64 (bytes)
  Size of program headers:               0 (bytes)
  Number of program headers:              0
  Size of section headers:               64 (bytes)
  Number of section headers:             14
  Section header string table index: 13

Section Headers:
[Nr] Name           Type           Address             Offset
     Size           EntSize          Flags  Link  Info  Align
[ 0]                  NULL              0000000000000000    00000000
     0000000000000000 0000000000000000          0  0    0
[ 1] .text            PROGBITS        0000000000000000    00000040
     0000000000000092 0000000000000000  AX      0  0    1
[ 2] .rela.text       RELA            0000000000000000    00000378
     00000000000000c0 0000000000000018  I      11  1    8
[ 3] .data            PROGBITS        0000000000000000    000000d2
     0000000000000000 0000000000000000  WA      0  0    1
[ 4] .bss             NOBITS          0000000000000000    000000d2
     0000000000000000 0000000000000000  WA      0  0    1
[ 5] .rodata          PROGBITS        0000000000000000    000000d2
     000000000000002b 0000000000000000  A      0  0    1
[ 6] .comment         PROGBITS        0000000000000000    000000fd
     000000000000002b 0000000000000001  MS      0  0    1
[ 7] .note.GNU-stack  PROGBITS        0000000000000000    00000128
     0000000000000000 0000000000000000          0  0    1
[ 8] .note.gnu.propert NOTE            0000000000000000    00000128
     0000000000000020 0000000000000000  A      0  0    8
[ 9] .eh_frame        PROGBITS        0000000000000000    00000148
     0000000000000038 0000000000000000  A      0  0    8
[10] .rela.eh_frame    RELA            0000000000000000    00000438
     0000000000000018 0000000000000018  I      11  9    8
[11] .symtab          SYMTAB          0000000000000000    00000180
     00000000000001b0 0000000000000018          12  10   8
[12] .strtab          STRTAB          0000000000000000    00000330
     0000000000000048 0000000000000000          0  0    1
[13] .shstrtab        STRTAB          0000000000000000    00000450
     0000000000000074 0000000000000000          0  0    1

Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),

```

```

jqw-1190400818@Venchi: ~/Desktop/csapp
L (link order), O (extra OS processing required), G (group), T (TLS),
C (compressed), x (unknown), o (OS specific), E (exclude),
l (large), p (processor specific)

There are no section groups in this file.

There are no program headers in this file.

There is no dynamic section in this file.

Relocation section '.rela.text' at offset 0x378 contains 8 entries:
  Offset          Info          Type          Sym. Value      Sym. Name + Addend
000000000001c    0005000000002 R_X86_64_PC32    0000000000000000 .rodata - 4
0000000000021    000c000000004 R_X86_64_PLT32    0000000000000000 puts - 4
000000000002b    000d000000004 R_X86_64_PLT32    0000000000000000 exit - 4
0000000000054    0005000000002 R_X86_64_PC32    0000000000000000 .rodata + 1a
000000000005e    000e000000004 R_X86_64_PLT32    0000000000000000 printf - 4
0000000000071    000f000000004 R_X86_64_PLT32    0000000000000000 atoi - 4
0000000000078    0010000000004 R_X86_64_PLT32    0000000000000000 sleep - 4
0000000000087    0011000000004 R_X86_64_PLT32    0000000000000000 getchar - 4

Relocation section '.rela.eh_frame' at offset 0x438 contains 1 entry:
  Offset          Info          Type          Sym. Value      Sym. Name + Addend
0000000000020    0002000000002 R_X86_64_PC32    0000000000000000 .text + 0

The decoding of unwind sections for machine type Advanced Micro Devices X86-64 i
s not currently supported.

Symbol table '.syntab' contains 18 entries:
  Num:   Value          Size Type   Bind   Vis    Ndx Name
  0: 0000000000000000    0 NOTYPE LOCAL DEFAULT UND
  1: 0000000000000000    0 FILE  LOCAL DEFAULT ABS hello.c
  2: 0000000000000000    0 SECTION LOCAL DEFAULT 1
  3: 0000000000000000    0 SECTION LOCAL DEFAULT 3
  4: 0000000000000000    0 SECTION LOCAL DEFAULT 4
  5: 0000000000000000    0 SECTION LOCAL DEFAULT 5
  6: 0000000000000000    0 SECTION LOCAL DEFAULT 7
  7: 0000000000000000    0 SECTION LOCAL DEFAULT 8
  8: 0000000000000000    0 SECTION LOCAL DEFAULT 9
  9: 0000000000000000    0 SECTION LOCAL DEFAULT 6
 10: 0000000000000000   146 FUNC  GLOBAL DEFAULT 1 main
 11: 0000000000000000    0 NOTYPE GLOBAL DEFAULT UND _GLOBAL_OFFSET_TABLE_
 12: 0000000000000000    0 NOTYPE GLOBAL DEFAULT UND puts
 13: 0000000000000000    0 NOTYPE GLOBAL DEFAULT UND exit
 14: 0000000000000000    0 NOTYPE GLOBAL DEFAULT UND printf
 15: 0000000000000000    0 NOTYPE GLOBAL DEFAULT UND atoi
 16: 0000000000000000    0 NOTYPE GLOBAL DEFAULT UND sleep
 17: 0000000000000000    0 NOTYPE GLOBAL DEFAULT UND getchar

No version information found in this file.

Displaying notes found in: .note.gnu.property
  Owner          Data size      Description
  GNU            0x00000010     NT_GNU_PROPERTY_TYPE_0
  Properties: x86 feature: IBT, SHSTK

```

4.4 Hello.o 的结果解析


```

jq-1190400818@Venchi: ~/Desktop/csapp
jq-1190400818@Venchi:~/Desktop/csapp$ objdump -d -r hello.o

hello.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
 0:  f3 0f 1e fa          endbr64
 4:  55                   push    %rbp
 5:  48 89 e5             mov     %rsp,%rbp
 8:  48 83 ec 20          sub     $0x20,%rsp
 c:  89 7d ec             mov     %edi,-0x14(%rbp)
 f:  48 89 75 e0          mov     %rsi,-0x20(%rbp)
13:  83 7d ec 04          cml     $0x4,-0x14(%rbp)
17:  74 16               je      2f <main+0x2f>
19:  48 8d 3d 00 00 00 00 lea     0x0(%rip),%rdi    # 20 <main+0x20>
                        1c: R_X86_64_PC32      .rodata-0x4
20:  e8 00 00 00 00      callq  25 <main+0x25>
                        21: R_X86_64_PLT32      puts-0x4
25:  bf 01 00 00 00      mov     $0x1,%edi
2a:  e8 00 00 00 00      callq  2f <main+0x2f>
                        2b: R_X86_64_PLT32      exit-0x4
2f:  c7 45 fc 00 00 00 00 movl    $0x0,-0x4(%rbp)
36:  eb 48               jmp     80 <main+0x80>
38:  48 8b 45 e0          mov     -0x20(%rbp),%rax
3c:  48 83 c0 10          add     $0x10,%rax
40:  48 8b 10             mov     (%rax),%rdx
43:  48 8b 45 e0          mov     -0x20(%rbp),%rax
47:  48 83 c0 08          add     $0x8,%rax
4b:  48 8b 00             mov     (%rax),%rax
4e:  48 89 c6             mov     %rax,%rsi
51:  48 8d 3d 00 00 00 00 lea     0x0(%rip),%rdi    # 58 <main+0x58>
                        54: R_X86_64_PC32      .rodata+0x1a
58:  b8 00 00 00 00      mov     $0x0,%eax
5d:  e8 00 00 00 00      callq  62 <main+0x62>
                        5e: R_X86_64_PLT32      printf-0x4
62:  48 8b 45 e0          mov     -0x20(%rbp),%rax
66:  48 83 c0 18          add     $0x18,%rax
6a:  48 8b 00             mov     (%rax),%rax
6d:  48 89 c7             mov     %rax,%rdi
70:  e8 00 00 00 00      callq  75 <main+0x75>
                        71: R_X86_64_PLT32      atoi-0x4
75:  89 c7               mov     %eax,%edi
77:  e8 00 00 00 00      callq  7c <main+0x7c>
                        78: R_X86_64_PLT32      sleep-0x4
7c:  83 45 fc 01          addl    $0x1,-0x4(%rbp)
80:  83 7d fc 07          cml     $0x7,-0x4(%rbp)
84:  7e b2               jle     38 <main+0x38>
86:  e8 00 00 00 00      callq  8b <main+0x8b>
                        87: R_X86_64_PLT32      getchar-0x4
8b:  b8 00 00 00 00      mov     $0x0,%eax
90:  c9                 leaveq
91:  c3                 retq
jq-1190400818@Venchi:~/Desktop/csapp$

```

说明机器语言的构成，与汇编语言的映射关系。特别是机器语言中的操作数与汇编语言不一致，特别是分支转移函数调用等。

反汇编器使用的指令命名规则与 GCC 生成的汇编代码使用的有些细微的差别

示例中，省略很多结尾 ‘q’。

每个.o 文件都包含

- 1.一个文件头，包含代码段、数据段和 BSS 段的大小
- 2.一个代码段，包含机器指令
- 3.一个数据段，包含初始化全局变量和初始化静态局部变量
- 4.一个 BSS 段，包含未初始化全局变量和未初始化静态局部变量
- 5.代码中的指针以及数据和 BSS 中的偏移量的重定位信息
- 6.一个符号表，包含非静态全局变量、函数名称及其属性

.4.5 本章小结

elf 头	将连续的文件映射到运行时的内存段
.text	已编译程序的机器代码
.rodata	只读数据，比如 printf 语句中的格式串和开关语句的跳转表
.data	已初始化的全局和静态 C 变量
.bss	未初始化的全局和静态 C 变量
.symtab	一个符号表，它存放在程序中定义和引用的函数和全局变量的信息
.rel.text	.text 中的重定位信息
.rel.data	.data 中的重定位信息
.debug	gcc -g 情况下出现 调试符号表
.line	gcc -g 情况下出现 原始 C 源程序的行号和.text 节中机器指令之间的映射
.strtab	一个字符串表，其内容包括 .symtab 和 .debug 节中的符号表，以及节头部中的节名字。
节头目表	描述目标文件的节。

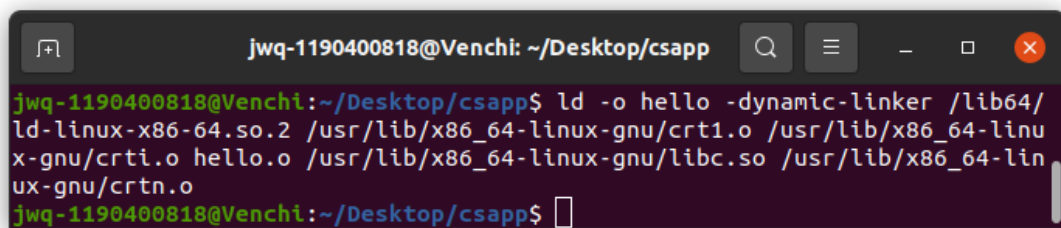
(第 4 章 1 分)

第 5 章 链接

.5.1 链接的概念与作用

共享库是致力于解决静态库缺陷的一个现代创新产物。共享库是一个目标模块，在运行或加载时，可以加载到任意的内存地址，并和一个在内存中的程序链接起来。这个过程称为动态链接，是由一个叫做动态链接器的程序来执行的。共享库也称为共享目标，在 Linux 系统中通常用 .so 后缀来表示。

.5.2 在 Ubuntu 下链接的命令



```
jwq-1190400818@Venchi: ~/Desktop/csapp
jwq-1190400818@Venchi:~/Desktop/csapp$ ld -o hello -dynamic-linker /lib64/ld-linux-x86-64.so.2 /usr/lib/x86_64-linux-gnu/crt1.o /usr/lib/x86_64-linux-gnu/crti.o hello.o /usr/lib/x86_64-linux-gnu/libc.so /usr/lib/x86_64-linux-gnu/crtn.o
jwq-1190400818@Venchi:~/Desktop/csapp$
```

.5.3 可执行目标文件 hello 的格式

```
jwq-1190400818@Venchi: ~/Desktop/csapp
jwq-1190400818@Venchi:~/Desktop/csapp$ readelf -a hello
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                               ELF64
  Data:                                   2's complement, little endian
  Version:                               1 (current)
  OS/ABI:                                UNIX - System V
  ABI Version:                           0
  Type:                                  EXEC (Executable file)
  Machine:                                Advanced Micro Devices X86-64
  Version:                                0x1
  Entry point address:                   0x4010f0
  Start of program headers:              64 (bytes into file)
  Start of section headers:              14208 (bytes into file)
  Flags:                                  0x0
  Size of this header:                    64 (bytes)
  Size of program headers:                56 (bytes)
  Number of program headers:              12
  Size of section headers:                64 (bytes)
  Number of section headers:              27
  Section header string table index: 26

Section Headers:
[Nr] Name           Type            Address          Offset
     Size           EntSize          Flags Link Info Align
[ 0]                NULL            0000000000000000 00000000
     0000000000000000 0000000000000000      0  0  0
[ 1] .interp          PROGBITS        00000000004002e0 000002e0
     000000000000001c 0000000000000000  A   0  0  1
[ 2] .note.gnu.propert NOTE            0000000000400300 00000300
     0000000000000020 0000000000000000  A   0  0  8
[ 3] .note.ABI-tag     NOTE            0000000000400320 00000320
     0000000000000020 0000000000000000  A   0  0  4
[ 4] .hash             HASH            0000000000400340 00000340
     0000000000000038 0000000000000004  A   6  0  8
[ 5] .gnu.hash          GNU_HASH        0000000000400378 00000378
     000000000000001c 0000000000000000  A   6  0  8
[ 6] .dynsym            DYNSYM          0000000000400398 00000398
     00000000000000d8 0000000000000018  A   7  1  8
[ 7] .dynstr            STRTAB          0000000000400470 00000470
     000000000000005c 0000000000000000  A   0  0  1
[ 8] .gnu.version       VERSYM          00000000004004cc 000004cc
```

.ELF Header:

Magic: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
 Class: ELF64
 Data: 2's complement, little endian
 Version: 1 (current)
 OS/ABI: UNIX - System V
 ABI Version: 0
 Type: EXEC (Executable file)

Machine: Advanced Micro Devices X86-64
 Version: 0x1
 Entry point address: 0x4010f0
 Start of program headers: 64 (bytes into file)
 Start of section headers: 14208 (bytes into file)
 Flags: 0x0
 Size of this header: 64 (bytes)
 Size of program headers: 56 (bytes)
 Number of program headers: 12
 Size of section headers: 64 (bytes)
 Number of section headers: 27
 Section header string table index: 26

Section Headers:

[Nr]	Name	Type	Address	Offset
	Size	EntSize	Flags Link Info Align	
[0]		NULL	0000000000000000	00000000
	0000000000000000	0000000000000000		0 0 0
[1]	.interp	PROGBITS	00000000004002e0	000002e0
	000000000000001c	0000000000000000	A	0 0 1
[2]	.note.gnu.propt	NOTE	0000000000400300	00000300
	0000000000000020	0000000000000000	A	0 0 8
[3]	.note.ABI-tag	NOTE	0000000000400320	00000320
	0000000000000020	0000000000000000	A	0 0 4
[4]	.hash	HASH	0000000000400340	00000340
	0000000000000038	0000000000000004	A	6 0 8
[5]	.gnu.hash	GNU_HASH	0000000000400378	00000378
	000000000000001c	0000000000000000	A	6 0 8
[6]	.dynsym	DYNSYM	0000000000400398	00000398
	00000000000000d8	0000000000000018	A	7 1 8
[7]	.dynstr	STRTAB	0000000000400470	00000470
	000000000000005c	0000000000000000	A	0 0 1
[8]	.gnu.version	VERSYM	00000000004004cc	000004cc
	0000000000000012	0000000000000002	A	6 0 2
[9]	.gnu.version_r	VERNEED	00000000004004e0	000004e0
	0000000000000020	0000000000000000	A	7 1 8
[10]	.rela.dyn	RELA	0000000000400500	00000500
	0000000000000030	0000000000000018	A	6 0 8
[11]	.rela.plt	RELA	0000000000400530	00000530
	0000000000000090	0000000000000018	AI	6 21 8
[12]	.init	PROGBITS	0000000000401000	00001000
	000000000000001b	0000000000000000	AX	0 0 4
[13]	.plt	PROGBITS	0000000000401020	00001020
	0000000000000070	0000000000000010	AX	0 0 16
[14]	.plt.sec	PROGBITS	0000000000401090	00001090
	0000000000000060	0000000000000010	AX	0 0 16
[15]	.text	PROGBITS	00000000004010f0	000010f0

```

0000000000000000145 0000000000000000 AX 0 0 16
[16] .fini PROGBITS 000000000000401238 00001238
0000000000000000d 0000000000000000 AX 0 0 4
[17] .rodata PROGBITS 000000000000402000 00002000
0000000000000002f 0000000000000000 A 0 0 4
[18] .eh_frame PROGBITS 000000000000402030 00002030
000000000000000fc 0000000000000000 A 0 0 8
[19] .dynamic DYNAMIC 000000000000403e50 00002e50
000000000000001a0 00000000000000010 WA 7 0 8
[20] .got PROGBITS 000000000000403ff0 00002ff0
00000000000000010 00000000000000008 WA 0 0 8
[21] .got.plt PROGBITS 000000000000404000 00003000
00000000000000048 00000000000000008 WA 0 0 8
[22] .data PROGBITS 000000000000404048 00003048
00000000000000004 00000000000000000 WA 0 0 1
[23] .comment PROGBITS 00000000000000000 0000304c
0000000000000002a 00000000000000001 MS 0 0 1
[24] .symtab SYMTAB 00000000000000000 00003078
0000000000000004c8 00000000000000018 25 30 8
[25] .strtab STRTAB 00000000000000000 00003540
000000000000000158 00000000000000000 0 0 1
[26] .shstrtab STRTAB 00000000000000000 00003698
000000000000000e1 00000000000000000 0 0 1

```

Key to Flags:

W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
L (link order), O (extra OS processing required), G (group), T (TLS),
C (compressed), x (unknown), o (OS specific), E (exclude),
l (large), p (processor specific)

There are no section groups in this file.

Program Headers:

Type	Offset	VirtAddr	PhysAddr
	FileSiz	MemSiz	Flags Align
PHDR	0x0000000000000040	0x000000000000400040	0x000000000000400040
	0x000000000000002a0	0x000000000000002a0	R 0x8
INTERP	0x000000000000002e0	0x0000000000004002e0	0x0000000000004002e0
	0x0000000000000001c	0x0000000000000001c	R 0x1
[Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]			
LOAD	0x00000000000000000	0x000000000000400000	0x000000000000400000
	0x000000000000005c0	0x000000000000005c0	R 0x1000
LOAD	0x000000000000001000	0x000000000000401000	0x000000000000401000
	0x00000000000000245	0x00000000000000245	R E 0x1000
LOAD	0x000000000000002000	0x000000000000402000	0x000000000000402000
	0x0000000000000012c	0x0000000000000012c	R 0x1000
LOAD	0x000000000000002e50	0x000000000000403e50	0x000000000000403e50
	0x000000000000001fc	0x000000000000001fc	RW 0x1000

```

DYNAMIC      0x00000000000002e50 0x000000000000403e50 0x000000000000403e50
              0x00000000000001a0 0x00000000000001a0 RW    0x8
NOTE         0x0000000000000300 0x000000000000400300 0x000000000000400300
              0x0000000000000020 0x0000000000000020 R     0x8
NOTE         0x0000000000000320 0x000000000000400320 0x000000000000400320
              0x0000000000000020 0x0000000000000020 R     0x4
GNU_PROPERTY 0x0000000000000300 0x000000000000400300
0x000000000000400300
              0x0000000000000020 0x0000000000000020 R     0x8
GNU_STACK    0x0000000000000000 0x0000000000000000 0x0000000000000000
              0x0000000000000000 0x0000000000000000 RW    0x10
GNU_RELRO    0x00000000000002e50 0x000000000000403e50 0x000000000000403e50
              0x00000000000001b0 0x00000000000001b0 R     0x1

```

Section to Segment mapping:

Segment Sections...

```

00
01  .interp
02                      .interp  .note.gnu.property  .note.ABI-
tag .hash .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rela.dyn .rela.plt
03  .init .plt .plt.sec .text .fini
04  .rodata .eh_frame
05  .dynamic .got .got.plt .data
06  .dynamic
07  .note.gnu.property
08  .note.ABI-tag
09  .note.gnu.property
10
11  .dynamic .got

```

Dynamic section at offset 0x2e50 contains 21 entries:

Tag	Type	Name/Value
0x0000000000000001	(NEEDED)	Shared library: [libc.so.6]
0x000000000000000c	(INIT)	0x401000
0x000000000000000d	(FINI)	0x401238
0x0000000000000004	(HASH)	0x400340
0x000000006ffffef5	(GNU_HASH)	0x400378
0x0000000000000005	(STRTAB)	0x400470
0x0000000000000006	(SYMTAB)	0x400398
0x000000000000000a	(STRSZ)	92 (bytes)
0x000000000000000b	(SYMENT)	24 (bytes)
0x0000000000000015	(DEBUG)	0x0
0x0000000000000003	(PLTGOT)	0x404000
0x0000000000000002	(PLTRELSZ)	144 (bytes)
0x0000000000000014	(PLTREL)	RELA
0x0000000000000017	(JMPREL)	0x400530
0x0000000000000007	(RELA)	0x400500

0x0000000000000008 (RELASZ)	48 (bytes)
0x0000000000000009 (RELAENT)	24 (bytes)
0x000000006ffffffe (VERNEED)	0x4004e0
0x000000006fffffff (VERNEEDNUM)	1
0x000000006ffffff0 (VERSYM)	0x4004cc
0x0000000000000000 (NULL)	0x0

Relocation section '.rela.dyn' at offset 0x500 contains 2 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
000000403ff0	000300000006	R_X86_64_GLOB_DAT		0000000000000000 __libc_start_main@GLIBC_2.2.5 + 0
000000403ff8	000500000006	R_X86_64_GLOB_DAT		0000000000000000 __gmon_start__ + 0

Relocation section '.rela.plt' at offset 0x530 contains 6 entries:

Offset	Info	Type	Sym. Value	Sym. Name + Addend
000000404018	000100000007	R_X86_64_JUMP_SLO		0000000000000000 puts@GLIBC_2.2.5 + 0
000000404020	000200000007	R_X86_64_JUMP_SLO		0000000000000000 printf@GLIBC_2.2.5 + 0
000000404028	000400000007	R_X86_64_JUMP_SLO		0000000000000000 getchar@GLIBC_2.2.5 + 0
000000404030	000600000007	R_X86_64_JUMP_SLO		0000000000000000 atoi@GLIBC_2.2.5 + 0
000000404038	000700000007	R_X86_64_JUMP_SLO		0000000000000000 exit@GLIBC_2.2.5 + 0
000000404040	000800000007	R_X86_64_JUMP_SLO		0000000000000000 sleep@GLIBC_2.2.5 + 0

The decoding of unwind sections for machine type Advanced Micro Devices X86-64 is not currently supported.

Symbol table '.dynsym' contains 9 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	0000000000000000	0	NOTYPE	LOCAL	DEFAULT	UND	
1:	0000000000000000			0	FUNC		GLOBAL DEFAULT UND
puts@GLIBC_2.2.5 (2)							
2:	0000000000000000			0	FUNC		GLOBAL DEFAULT UND
printf@GLIBC_2.2.5 (2)							
3:	0000000000000000			0	FUNC		GLOBAL DEFAULT UND
__libc_start_main@GLIBC_2.2.5 (2)							
4:	0000000000000000			0	FUNC		GLOBAL DEFAULT UND
getchar@GLIBC_2.2.5 (2)							
5:	0000000000000000	0	NOTYPE	WEAK	DEFAULT	UND	__gmon_start__
6:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	atoi@GLIBC_2.2.5
(2)							
7:	0000000000000000	0	FUNC	GLOBAL	DEFAULT	UND	exit@GLIBC_2.2.5

(2)

8: 0000000000000000 0 FUNC GLOBAL DEFAULT UND
sleep@@GLIBC_2.2.5 (2)

Symbol table '.symtab' contains 51 entries:

Num:	Value	Size	Type	Bind	Vis	Ndx	Name
0:	0000000000000000	0	NOTYPE	LOCAL	DEFAULT		UND
1:	00000000004002e0	0	SECTION	LOCAL	DEFAULT	1	
2:	0000000000400300	0	SECTION	LOCAL	DEFAULT	2	
3:	0000000000400320	0	SECTION	LOCAL	DEFAULT	3	
4:	0000000000400340	0	SECTION	LOCAL	DEFAULT	4	
5:	0000000000400378	0	SECTION	LOCAL	DEFAULT	5	
6:	0000000000400398	0	SECTION	LOCAL	DEFAULT	6	
7:	0000000000400470	0	SECTION	LOCAL	DEFAULT	7	
8:	00000000004004cc	0	SECTION	LOCAL	DEFAULT	8	
9:	00000000004004e0	0	SECTION	LOCAL	DEFAULT	9	
10:	0000000000400500	0	SECTION	LOCAL	DEFAULT	10	
11:	0000000000400530	0	SECTION	LOCAL	DEFAULT	11	
12:	0000000000401000	0	SECTION	LOCAL	DEFAULT	12	
13:	0000000000401020	0	SECTION	LOCAL	DEFAULT	13	
14:	0000000000401090	0	SECTION	LOCAL	DEFAULT	14	
15:	00000000004010f0	0	SECTION	LOCAL	DEFAULT	15	
16:	0000000000401238	0	SECTION	LOCAL	DEFAULT	16	
17:	0000000000402000	0	SECTION	LOCAL	DEFAULT	17	
18:	0000000000402030	0	SECTION	LOCAL	DEFAULT	18	
19:	0000000000403e50	0	SECTION	LOCAL	DEFAULT	19	
20:	0000000000403ff0	0	SECTION	LOCAL	DEFAULT	20	
21:	0000000000404000	0	SECTION	LOCAL	DEFAULT	21	
22:	0000000000404048	0	SECTION	LOCAL	DEFAULT	22	
23:	0000000000000000	0	SECTION	LOCAL	DEFAULT	23	
24:	0000000000000000	0	FILE	LOCAL	DEFAULT		ABS hello.c
25:	0000000000000000	0	FILE	LOCAL	DEFAULT		ABS
26:	0000000000403e50	0	NOTYPE	LOCAL	DEFAULT	19	__init_array_end
27:	0000000000403e50	0	OBJECT	LOCAL	DEFAULT	19	__DYNAMIC
28:	0000000000403e50	0	NOTYPE	LOCAL	DEFAULT	19	__init_array_start
29:	0000000000404000	0	OBJECT	LOCAL	DEFAULT	21	
_GLOBAL_OFFSET_TABLE_							
30:	0000000000401230	5	FUNC	GLOBAL	DEFAULT	15	__libc_csu_fini
31:	0000000000404048	0	NOTYPE	WEAK	DEFAULT	22	data_start
32:	0000000000000000	0	FUNC	GLOBAL	DEFAULT		UND
puts@@GLIBC_2.2.5							
33:	000000000040404c	0	NOTYPE	GLOBAL	DEFAULT	22	_edata
34:	0000000000401238	0	FUNC	GLOBAL	HIDDEN	16	_fini
35:	0000000000000000	0	FUNC	GLOBAL	DEFAULT		UND
printf@@GLIBC_2.2.5							
36:	0000000000000000	0	FUNC	GLOBAL	DEFAULT		UND
__libc_start_main@@GLIBC_							

```

37: 0000000000404048  0 NOTYPE GLOBAL DEFAULT 22 __data_start
38: 0000000000000000      0 FUNC      GLOBAL DEFAULT  UND
getchar@@GLIBC_2.2.5
39: 0000000000000000  0 NOTYPE WEAK  DEFAULT UND __gmon_start__
40: 0000000000402000   4 OBJECT GLOBAL DEFAULT 17 __IO_stdin_used
41: 00000000004011c0 101 FUNC  GLOBAL DEFAULT 15 __libc_csu_init
42: 0000000000404050  0 NOTYPE GLOBAL DEFAULT 22 __end
43: 0000000000401120      5 FUNC      GLOBAL HIDDEN 15
_dl_relocate_static_pie
44: 00000000004010f0  47 FUNC  GLOBAL DEFAULT 15 _start
45: 000000000040404c  0 NOTYPE GLOBAL DEFAULT 22 __bss_start
46: 0000000000401125 146 FUNC  GLOBAL DEFAULT 15 main
47: 0000000000000000      0 FUNC      GLOBAL DEFAULT  UND
atoi@@GLIBC_2.2.5
48: 0000000000000000      0 FUNC      GLOBAL DEFAULT  UND
exit@@GLIBC_2.2.5
49: 0000000000000000      0 FUNC      GLOBAL DEFAULT  UND
sleep@@GLIBC_2.2.5
50: 0000000000401000  0 FUNC  GLOBAL HIDDEN 12 _init

```

Histogram for bucket list length (total of 3 buckets):

Length	Number	% of total	Coverage
0	0	(0.0%)	
1	0	(0.0%)	0.0%
2	1	(33.3%)	25.0%
3	2	(66.7%)	100.0%

Version symbols section '.gnu.version' contains 9 entries:

```

Addr: 0x00000000004004cc Offset: 0x0004cc Link: 6 (.dynsym)
000: 0 (*local*)      2 (GLIBC_2.2.5) 2 (GLIBC_2.2.5) 2 (GLIBC_2.2.5)
004: 2 (GLIBC_2.2.5) 0 (*local*)    2 (GLIBC_2.2.5) 2 (GLIBC_2.2.5)
008: 2 (GLIBC_2.2.5)

```

Version needs section '.gnu.version_r' contains 1 entry:

```

Addr: 0x00000000004004e0 Offset: 0x0004e0 Link: 7 (.dynstr)
000000: Version: 1 File: libc.so.6 Cnt: 1
0x0010: Name: GLIBC_2.2.5 Flags: none Version: 2

```

Displaying notes found in: .note.gnu.property

Owner	Data size	Description
GNU	0x00000010	NT_GNU_PROPERTY_TYPE_0

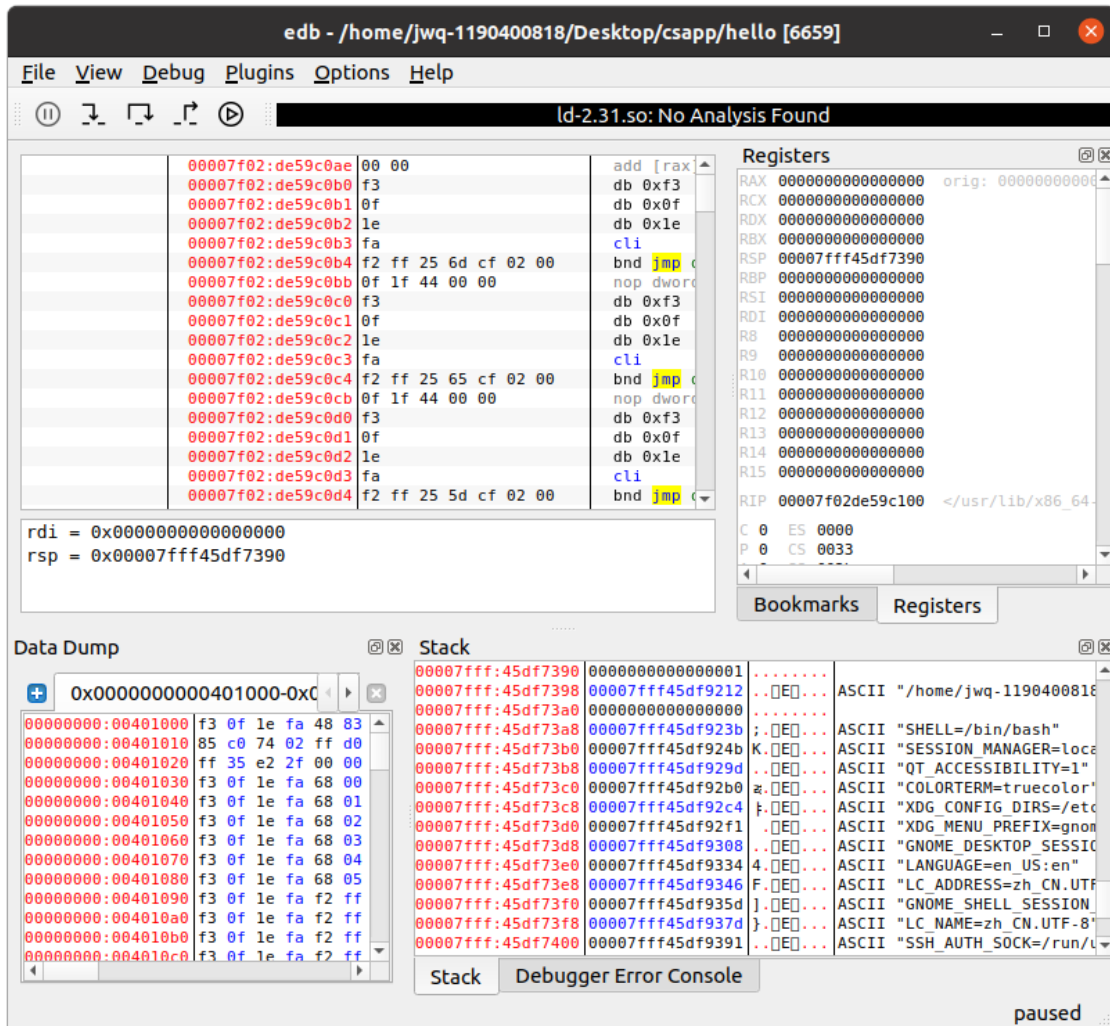
Properties: x86 feature: IBT, SHSTK

Displaying notes found in: .note.ABI-tag

Owner	Data size	Description
GNU	0x00000010	NT_GNU_ABI_TAG (ABI version tag)

OS: Linux, ABI: 3.2.0

.5.4 hello 的虚拟地址空间



.5.5 链接的重定位过程分析

```

jq-1190400818@Venchi: ~/Desktop/csapp
jq-1190400818@Venchi:~/Desktop/csapp$ objdump -d -r hello

hello:      file format elf64-x86-64

Disassembly of section .init:

0000000000401000 <_init>:
 401000:      f3 0f 1e fa      endbr64
 401004:      48 83 ec 08      sub    $0x8,%rsp
 401008:      48 8b 05 e9 2f 00 00  mov    0x2fe9(%rip),%rax      # 403ff8
<__gmon_start__>
 40100f:      48 85 c0      test   %rax,%rax
 401012:      74 02      je     401016 <_init+0x16>
 401014:      ff d0      callq  *%rax
 401016:      48 83 c4 08      add    $0x8,%rsp
 40101a:      c3      retq

Disassembly of section .plt:

0000000000401020 <.plt>:
 401020:      ff 35 e2 2f 00 00      pushq  0x2fe2(%rip)      # 404008 <_GL
OBAL_OFFSET_TABLE_+0x8>
 401026:      f2 ff 25 e3 2f 00 00      bnd jmpq *0x2fe3(%rip)      # 404010 <
_GLOBAL_OFFSET_TABLE_+0x10>
 40102d:      0f 1f 00      nopl   (%rax)
 401030:      f3 0f 1e fa      endbr64
 401034:      68 00 00 00 00      pushq  $0x0
 401039:      f2 e9 e1 ff ff ff      bnd jmpq 401020 <.plt>
 40103f:      90      nop
 401040:      f3 0f 1e fa      endbr64
 401044:      68 01 00 00 00      pushq  $0x1
 401049:      f2 e9 d1 ff ff ff      bnd jmpq 401020 <.plt>
 40104f:      90      nop
 401050:      f3 0f 1e fa      endbr64
 401054:      68 02 00 00 00      pushq  $0x2
 401059:      f2 e9 c1 ff ff ff      bnd jmpq 401020 <.plt>
 40105f:      90      nop
 401060:      f3 0f 1e fa      endbr64
 401064:      68 03 00 00 00      pushq  $0x3
 401069:      f2 e9 b1 ff ff ff      bnd jmpq 401020 <.plt>
 40106f:      90      nop
 401070:      f3 0f 1e fa      endbr64
 401074:      68 04 00 00 00      pushq  $0x4
 401079:      f2 e9 a1 ff ff ff      bnd jmpq 401020 <.plt>
 40107f:      90      nop
 401080:      f3 0f 1e fa      endbr64
 401084:      68 05 00 00 00      pushq  $0x5
 401089:      f2 e9 91 ff ff ff      bnd jmpq 401020 <.plt>
 40108f:      90      nop

Disassembly of section .plt.sec:

0000000000401090 <puts@plt>:
 401090:      f3 0f 1e fa      endbr64

```

hello: file format elf64-x86-64

Disassembly of section .init:

```

0000000000401000 <_init>:
401000: f3 0f 1e fa      endbr64
401004: 48 83 ec 08      sub    $0x8,%rsp
401008: 48 8b 05 e9 2f 00 00 mov     0x2fe9(%rip),%rax      # 403ff8
<__gmon_start__>
40100f: 48 85 c0         test   %rax,%rax
401012: 74 02           je     401016 <_init+0x16>
401014: ff d0          callq  *%rax
401016: 48 83 c4 08      add    $0x8,%rsp
40101a: c3             retq

```

Disassembly of section .plt:

```

0000000000401020 <.plt>:
401020: ff 35 e2 2f 00 00 pushq   0x2fe2(%rip)          # 404008
<_GLOBAL_OFFSET_TABLE_+0x8>
401026: f2 ff 25 e3 2f 00 00 bnd jmpq *0x2fe3(%rip)      # 404010
<_GLOBAL_OFFSET_TABLE_+0x10>
40102d: 0f 1f 00        nopl   (%rax)
401030: f3 0f 1e fa      endbr64
401034: 68 00 00 00 00   pushq  $0x0
401039: f2 e9 e1 ff ff ff bnd jmpq 401020 <.plt>
40103f: 90              nop
401040: f3 0f 1e fa      endbr64
401044: 68 01 00 00 00   pushq  $0x1
401049: f2 e9 d1 ff ff ff bnd jmpq 401020 <.plt>
40104f: 90              nop
401050: f3 0f 1e fa      endbr64
401054: 68 02 00 00 00   pushq  $0x2
401059: f2 e9 c1 ff ff ff bnd jmpq 401020 <.plt>
40105f: 90              nop
401060: f3 0f 1e fa      endbr64
401064: 68 03 00 00 00   pushq  $0x3
401069: f2 e9 b1 ff ff ff bnd jmpq 401020 <.plt>

```

```

40106f: 90          nop
401070: f3 0f 1e fa  endbr64
401074: 68 04 00 00 00  pushq $0x4
401079: f2 e9 a1 ff ff ff bnd jmpq 401020 <.plt>
40107f: 90          nop
401080: f3 0f 1e fa  endbr64
401084: 68 05 00 00 00  pushq $0x5
401089: f2 e9 91 ff ff ff bnd jmpq 401020 <.plt>
40108f: 90          nop

```

Disassembly of section .plt.sec:

```

0000000000401090 <puts@plt>:
401090: f3 0f 1e fa  endbr64
401094: f2 ff 25 7d 2f 00 00 bnd jmpq *0x2f7d(%rip) # 404018
<puts@GLIBC_2.2.5>
40109b: 0f 1f 44 00 00 nopl 0x0(%rax,%rax,1)

00000000004010a0 <printf@plt>:
4010a0: f3 0f 1e fa  endbr64
4010a4: f2 ff 25 75 2f 00 00 bnd jmpq *0x2f75(%rip) # 404020
<printf@GLIBC_2.2.5>
4010ab: 0f 1f 44 00 00 nopl 0x0(%rax,%rax,1)

00000000004010b0 <getchar@plt>:
4010b0: f3 0f 1e fa  endbr64
4010b4: f2 ff 25 6d 2f 00 00 bnd jmpq *0x2f6d(%rip) # 404028
<getchar@GLIBC_2.2.5>
4010bb: 0f 1f 44 00 00 nopl 0x0(%rax,%rax,1)

00000000004010c0 <atoi@plt>:
4010c0: f3 0f 1e fa  endbr64
4010c4: f2 ff 25 65 2f 00 00 bnd jmpq *0x2f65(%rip) # 404030
<atoi@GLIBC_2.2.5>
4010cb: 0f 1f 44 00 00 nopl 0x0(%rax,%rax,1)

00000000004010d0 <exit@plt>:

```

```

4010d0:  f3 0f 1e fa      endbr64
4010d4:  f2 ff 25 5d 2f 00 00  bnd jmpq *0x2f5d(%rip)      # 404038
<exit@GLIBC_2.2.5>
4010db:  0f 1f 44 00 00      nopl 0x0(%rax,%rax,1)

00000000004010e0 <sleep@plt>:
4010e0:  f3 0f 1e fa      endbr64
4010e4:  f2 ff 25 55 2f 00 00  bnd jmpq *0x2f55(%rip)      # 404040
<sleep@GLIBC_2.2.5>
4010eb:  0f 1f 44 00 00      nopl 0x0(%rax,%rax,1)

```

Disassembly of section .text:

```

00000000004010f0 <_start>:
4010f0:  f3 0f 1e fa      endbr64
4010f4:  31 ed            xor  %ebp,%ebp
4010f6:  49 89 d1          mov  %rdx,%r9
4010f9:  5e              pop  %rsi
4010fa:  48 89 e2          mov  %rsp,%rdx
4010fd:  48 83 e4 f0      and  $0xfffffffffff0,%rsp
401101:  50              push %rax
401102:  54              push %rsp
401103:  49 c7 c0 30 12 40 00 mov  $0x401230,%r8
40110a:  48 c7 c1 c0 11 40 00 mov  $0x4011c0,%rcx
401111:  48 c7 c7 25 11 40 00 mov  $0x401125,%rdi
401118:  ff 15 d2 2e 00 00  callq *0x2ed2(%rip)      # 403ff0
<__libc_start_main@GLIBC_2.2.5>
40111e:  f4              hlt
40111f:  90              nop

```

0000000000401120 <_dl_relocate_static_pie>:

```

401120:  f3 0f 1e fa      endbr64
401124:  c3              retq

```

0000000000401125 <main>:

```

401125:  f3 0f 1e fa      endbr64
401129:  55              push %rbp

```

```

40112a: 48 89 e5      mov  %rsp,%rbp
40112d: 48 83 ec 20    sub  $0x20,%rsp
401131: 89 7d ec      mov  %edi,-0x14(%rbp)
401134: 48 89 75 e0    mov  %rsi,-0x20(%rbp)
401138: 83 7d ec 04    cmpl $0x4,-0x14(%rbp)
40113c: 74 16         je   401154 <main+0x2f>
40113e: 48 8d 3d bf 0e 00 00 lea   0xebf(%rip),%rdi      # 402004
<_IO_stdin_used+0x4>
401145: e8 46 ff ff ff callq 401090 <puts@plt>
40114a: bf 01 00 00 00 mov  $0x1,%edi
40114f: e8 7c ff ff ff callq 4010d0 <exit@plt>
401154: c7 45 fc 00 00 00 00 movl $0x0,-0x4(%rbp)
40115b: eb 48         jmp  4011a5 <main+0x80>
40115d: 48 8b 45 e0    mov  -0x20(%rbp),%rax
401161: 48 83 c0 10    add  $0x10,%rax
401165: 48 8b 10      mov  (%rax),%rdx
401168: 48 8b 45 e0    mov  -0x20(%rbp),%rax
40116c: 48 83 c0 08    add  $0x8,%rax
401170: 48 8b 00      mov  (%rax),%rax
401173: 48 89 c6      mov  %rax,%rsi
401176: 48 8d 3d a5 0e 00 00 lea   0xea5(%rip),%rdi      # 402022
<_IO_stdin_used+0x22>
40117d: b8 00 00 00 00 mov  $0x0,%eax
401182: e8 19 ff ff ff callq 4010a0 <printf@plt>
401187: 48 8b 45 e0    mov  -0x20(%rbp),%rax
40118b: 48 83 c0 18    add  $0x18,%rax
40118f: 48 8b 00      mov  (%rax),%rax
401192: 48 89 c7      mov  %rax,%rdi
401195: e8 26 ff ff ff callq 4010c0 <atoi@plt>
40119a: 89 c7         mov  %eax,%edi
40119c: e8 3f ff ff ff callq 4010e0 <sleep@plt>
4011a1: 83 45 fc 01    addl $0x1,-0x4(%rbp)
4011a5: 83 7d fc 07    cmpl $0x7,-0x4(%rbp)
4011a9: 7e b2         jle  40115d <main+0x38>
4011ab: e8 00 ff ff ff callq 4010b0 <getchar@plt>
4011b0: b8 00 00 00 00 mov  $0x0,%eax
4011b5: c9           leaveq

```



```

4011b6:  c3                retq
4011b7:  66 0f 1f 84 00 00 00 nopw  0x0(%rax,%rax,1)
4011be:  00 00

00000000004011c0 <__libc_csu_init>:
4011c0:  f3 0f 1e fa      endbr64
4011c4:  41 57            push  %r15
4011c6:  4c 8d 3d 83 2c 00 00 lea     0x2c83(%rip),%r15      # 403e50
<_DYNAMIC>
4011cd:  41 56            push  %r14
4011cf:  49 89 d6         mov   %rdx,%r14
4011d2:  41 55            push  %r13
4011d4:  49 89 f5         mov   %rsi,%r13
4011d7:  41 54            push  %r12
4011d9:  41 89 fc         mov   %edi,%r12d
4011dc:  55              push  %rbp
4011dd:  48 8d 2d 6c 2c 00 00 lea     0x2c6c(%rip),%rbp      # 403e50
<_DYNAMIC>
4011e4:  53              push  %rbx
4011e5:  4c 29 fd         sub   %r15,%rbp
4011e8:  48 83 ec 08      sub   $0x8,%rsp
4011ec:  e8 0f fe ff ff   callq 401000 <_init>
4011f1:  48 c1 fd 03      sar   $0x3,%rbp
4011f5:  74 1f           je    401216 <__libc_csu_init+0x56>
4011f7:  31 db           xor   %ebx,%ebx
4011f9:  0f 1f 80 00 00 00 00 nopl  0x0(%rax)
401200:  4c 89 f2         mov   %r14,%rdx
401203:  4c 89 ee         mov   %r13,%rsi
401206:  44 89 e7         mov   %r12d,%edi
401209:  41 ff 14 df      callq *(%r15,%rbx,8)
40120d:  48 83 c3 01      add   $0x1,%rbx
401211:  48 39 dd         cmp   %rbx,%rbp
401214:  75 ea           jne   401200 <__libc_csu_init+0x40>
401216:  48 83 c4 08      add   $0x8,%rsp
40121a:  5b              pop   %rbx
40121b:  5d              pop   %rbp
40121c:  41 5c           pop   %r12

```

```
40121e: 41 5d      pop    %r13
401220: 41 5e      pop    %r14
401222: 41 5f      pop    %r15
401224: c3        retq
401225: 66 66 2e 0f 1f 84 00  data16 nopw %cs:0x0(%rax,%rax,1)
40122c: 00 00 00 00
```

0000000000401230 <__libc_csu_fini>:

```
401230: f3 0f 1e fa  endbr64
401234: c3        retq
```

Disassembly of section .fini:

0000000000401238 <_fini>:

```
401238: f3 0f 1e fa  endbr64
40123c: 48 83 ec 08      sub    $0x8,%rsp
401240: 48 83 c4 08      add    $0x8,%rsp
401244: c3        retq
```

.5.6 hello 的执行流程

ld-2.23.so!_dl_start

ld-2.23.so! dl_init

hello!_start

hello!__libc_start_main

libc-2.23.so!__libc_start_main

libc-2.23.so! cxa_atexit

hello!__libc_csu_init

hello!_init

libc-2.23.so!_setjmp

libc-2.23.so!_sigsetjmp

hello!main

hello!puts@plt

hello!exit@plt

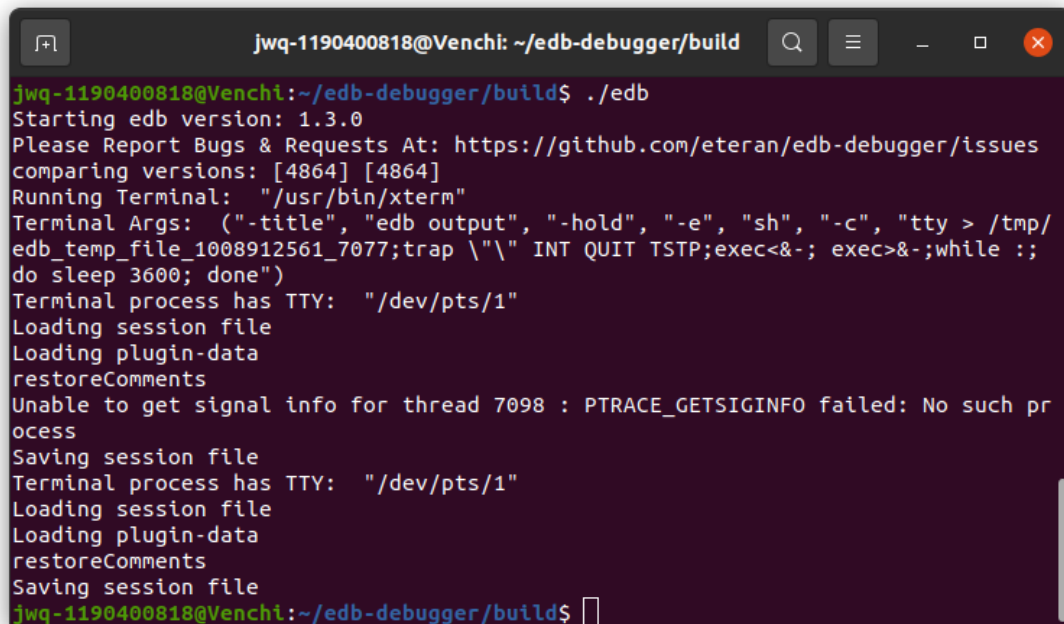
hello!printf@plt

hello!sleep@plt

hello!getchar@plt

ld-2.23.so!_dl_runtime_resolve_avx

libc-2.23.so!exit



```
jwq-1190400818@Venchi: ~/edb-debugger/build
jwq-1190400818@Venchi:~/edb-debugger/build$ ./edb
Starting edb version: 1.3.0
Please Report Bugs & Requests At: https://github.com/eteran/edb-debugger/issues
comparing versions: [4864] [4864]
Running Terminal: "/usr/bin/xterm"
Terminal Args: ("-title", "edb output", "-hold", "-e", "sh", "-c", "tty > /tmp/
edb_temp_file_1008912561_7077;trap \"\" INT QUIT TSTP;exec<&-; exec>&-;while ;;
do sleep 3600; done")
Terminal process has TTY: "/dev/pts/1"
Loading session file
Loading plugin-data
restoreComments
Unable to get signal info for thread 7098 : PTRACE_GETSIGINFO failed: No such pr
ocess
Saving session file
Terminal process has TTY: "/dev/pts/1"
Loading session file
Loading plugin-data
restoreComments
Saving session file
jwq-1190400818@Venchi:~/edb-debugger/build$
```

5.7 Hello 的动态链接分析

假设程序调用一个有共享库定义的函数。编译器没有办法预测这个函数的运行是地质，因为定义它的共享模块在运行时可以加载到任意位置。正常的办法是为该引用生成一条重定位记录，然后动态链接器在程序加载的时候解析它。GNU 编译系统使用了一种很有趣的技术来解决这个问题，称为延迟绑定，将过程地址的绑定推迟到第一次调用该过程时。

The screenshot shows the EDB debugger interface. The main window displays assembly code for the `.plt` section. The registers window on the right shows the state of various registers, including `RAX`, `RDI`, `RSI`, `RBP`, `RSP`, `R15`, `R14`, `R13`, `R12`, `R11`, `R10`, `R9`, `R8`, `R7`, `R6`, `R5`, `R4`, `R3`, `R2`, `R1`, `R0`, `CR0`, `CR2`, `CR3`, `CR4`, `CR8`, `CR9`, `CR10`, `CR11`, `CR12`, `CR13`, `CR14`, `CR15`, `CR16`, `CR17`, `CR18`, `CR19`, `CR20`, `CR21`, `CR22`, `CR23`, `CR24`, `CR25`, `CR26`, `CR27`, `CR28`, `CR29`, `CR30`, `CR31`, `CR32`, `CR33`, `CR34`, `CR35`, `CR36`, `CR37`, `CR38`, `CR39`, `CR40`, `CR41`, `CR42`, `CR43`, `CR44`, `CR45`, `CR46`, `CR47`, `CR48`, `CR49`, `CR50`, `CR51`, `CR52`, `CR53`, `CR54`, `CR55`, `CR56`, `CR57`, `CR58`, `CR59`, `CR60`, `CR61`, `CR62`, `CR63`, `CR64`, `CR65`, `CR66`, `CR67`, `CR68`, `CR69`, `CR70`, `CR71`, `CR72`, `CR73`, `CR74`, `CR75`, `CR76`, `CR77`, `CR78`, `CR79`, `CR80`, `CR81`, `CR82`, `CR83`, `CR84`, `CR85`, `CR86`, `CR87`, `CR88`, `CR89`, `CR90`, `CR91`, `CR92`, `CR93`, `CR94`, `CR95`, `CR96`, `CR97`, `CR98`, `CR99`, `CR100`, `CR101`, `CR102`, `CR103`, `CR104`, `CR105`, `CR106`, `CR107`, `CR108`, `CR109`, `CR110`, `CR111`, `CR112`, `CR113`, `CR114`, `CR115`, `CR116`, `CR117`, `CR118`, `CR119`, `CR120`, `CR121`, `CR122`, `CR123`, `CR124`, `CR125`, `CR126`, `CR127`, `CR128`, `CR129`, `CR130`, `CR131`, `CR132`, `CR133`, `CR134`, `CR135`, `CR136`, `CR137`, `CR138`, `CR139`, `CR140`, `CR141`, `CR142`, `CR143`, `CR144`, `CR145`, `CR146`, `CR147`, `CR148`, `CR149`, `CR150`, `CR151`, `CR152`, `CR153`, `CR154`, `CR155`, `CR156`, `CR157`, `CR158`, `CR159`, `CR160`, `CR161`, `CR162`, `CR163`, `CR164`, `CR165`, `CR166`, `CR167`, `CR168`, `CR169`, `CR170`, `CR171`, `CR172`, `CR173`, `CR174`, `CR175`, `CR176`, `CR177`, `CR178`, `CR179`, `CR180`, `CR181`, `CR182`, `CR183`, `CR184`, `CR185`, `CR186`, `CR187`, `CR188`, `CR189`, `CR190`, `CR191`, `CR192`, `CR193`, `CR194`, `CR195`, `CR196`, `CR197`, `CR198`, `CR199`, `CR200`, `CR201`, `CR202`, `CR203`, `CR204`, `CR205`, `CR206`, `CR207`, `CR208`, `CR209`, `CR210`, `CR211`, `CR212`, `CR213`, `CR214`, `CR215`, `CR216`, `CR217`, `CR218`, `CR219`, `CR220`, `CR221`, `CR222`, `CR223`, `CR224`, `CR225`, `CR226`, `CR227`, `CR228`, `CR229`, `CR230`, `CR231`, `CR232`, `CR233`, `CR234`, `CR235`, `CR236`, `CR237`, `CR238`, `CR239`, `CR240`, `CR241`, `CR242`, `CR243`, `CR244`, `CR245`, `CR246`, `CR247`, `CR248`, `CR249`, `CR250`, `CR251`, `CR252`, `CR253`, `CR254`, `CR255`, `CR256`, `CR257`, `CR258`, `CR259`, `CR260`, `CR261`, `CR262`, `CR263`, `CR264`, `CR265`, `CR266`, `CR267`, `CR268`, `CR269`, `CR270`, `CR271`, `CR272`, `CR273`, `CR274`, `CR275`, `CR276`, `CR277`, `CR278`, `CR279`, `CR280`, `CR281`, `CR282`, `CR283`, `CR284`, `CR285`, `CR286`, `CR287`, `CR288`, `CR289`, `CR290`, `CR291`, `CR292`, `CR293`, `CR294`, `CR295`, `CR296`, `CR297`, `CR298`, `CR299`, `CR300`, `CR301`, `CR302`, `CR303`, `CR304`, `CR305`, `CR306`, `CR307`, `CR308`, `CR309`, `CR310`, `CR311`, `CR312`, `CR313`, `CR314`, `CR315`, `CR316`, `CR317`, `CR318`, `CR319`, `CR320`, `CR321`, `CR322`, `CR323`, `CR324`, `CR325`, `CR326`, `CR327`, `CR328`, `CR329`, `CR330`, `CR331`, `CR332`, `CR333`, `CR334`, `CR335`, `CR336`, `CR337`, `CR338`, `CR339`, `CR340`, `CR341`, `CR342`, `CR343`, `CR344`, `CR345`, `CR346`, `CR347`, `CR348`, `CR349`, `CR350`, `CR351`, `CR352`, `CR353`, `CR354`, `CR355`, `CR356`, `CR357`, `CR358`, `CR359`, `CR360`, `CR361`, `CR362`, `CR363`, `CR364`, `CR365`, `CR366`, `CR367`, `CR368`, `CR369`, `CR370`, `CR371`, `CR372`, `CR373`, `CR374`, `CR375`, `CR376`, `CR377`, `CR378`, `CR379`, `CR380`, `CR381`, `CR382`, `CR383`, `CR384`, `CR385`, `CR386`, `CR387`, `CR388`, `CR389`, `CR390`, `CR391`, `CR392`, `CR393`, `CR394`, `CR395`, `CR396`, `CR397`, `CR398`, `CR399`, `CR400`, `CR401`, `CR402`, `CR403`, `CR404`, `CR405`, `CR406`, `CR407`, `CR408`, `CR409`, `CR410`, `CR411`, `CR412`, `CR413`, `CR414`, `CR415`, `CR416`, `CR417`, `CR418`, `CR419`, `CR420`, `CR421`, `CR422`, `CR423`, `CR424`, `CR425`, `CR426`, `CR427`, `CR428`, `CR429`, `CR430`, `CR431`, `CR432`, `CR433`, `CR434`, `CR435`, `CR436`, `CR437`, `CR438`, `CR439`, `CR440`, `CR441`, `CR442`, `CR443`, `CR444`, `CR445`, `CR446`, `CR447`, `CR448`, `CR449`, `CR450`, `CR451`, `CR452`, `CR453`, `CR454`, `CR455`, `CR456`, `CR457`, `CR458`, `CR459`, `CR460`, `CR461`, `CR462`, `CR463`, `CR464`, `CR465`, `CR466`, `CR467`, `CR468`, `CR469`, `CR470`, `CR471`, `CR472`, `CR473`, `CR474`, `CR475`, `CR476`, `CR477`, `CR478`, `CR479`, `CR480`, `CR481`, `CR482`, `CR483`, `CR484`, `CR485`, `CR486`, `CR487`, `CR488`, `CR489`, `CR490`, `CR491`, `CR492`, `CR493`, `CR494`, `CR495`, `CR496`, `CR497`, `CR498`, `CR499`, `CR500`, `CR501`, `CR502`, `CR503`, `CR504`, `CR505`, `CR506`, `CR507`, `CR508`, `CR509`, `CR510`, `CR511`, `CR512`, `CR513`, `CR514`, `CR515`, `CR516`, `CR517`, `CR518`, `CR519`, `CR520`, `CR521`, `CR522`, `CR523`, `CR524`, `CR525`, `CR526`, `CR527`, `CR528`, `CR529`, `CR530`, `CR531`, `CR532`, `CR533`, `CR534`, `CR535`, `CR536`, `CR537`, `CR538`, `CR539`, `CR540`, `CR541`, `CR542`, `CR543`, `CR544`, `CR545`, `CR546`, `CR547`, `CR548`, `CR549`, `CR550`, `CR551`, `CR552`, `CR553`, `CR554`, `CR555`, `CR556`, `CR557`, `CR558`, `CR559`, `CR560`, `CR561`, `CR562`, `CR563`, `CR564`, `CR565`, `CR566`, `CR567`, `CR568`, `CR569`, `CR570`, `CR571`, `CR572`, `CR573`, `CR574`, `CR575`, `CR576`, `CR577`, `CR578`, `CR579`, `CR580`, `CR581`, `CR582`, `CR583`, `CR584`, `CR585`, `CR586`, `CR587`, `CR588`, `CR589`, `CR590`, `CR591`, `CR592`, `CR593`, `CR594`, `CR595`, `CR596`, `CR597`, `CR598`, `CR599`, `CR600`, `CR601`, `CR602`, `CR603`, `CR604`, `CR605`, `CR606`, `CR607`, `CR608`, `CR609`, `CR610`, `CR611`, `CR612`, `CR613`, `CR614`, `CR615`, `CR616`, `CR617`, `CR618`, `CR619`, `CR620`, `CR621`, `CR622`, `CR623`, `CR624`, `CR625`, `CR626`, `CR627`, `CR628`, `CR629`, `CR630`, `CR631`, `CR632`, `CR633`, `CR634`, `CR635`, `CR636`, `CR637`, `CR638`, `CR639`, `CR640`, `CR641`, `CR642`, `CR643`, `CR644`, `CR645`, `CR646`, `CR647`, `CR648`, `CR649`, `CR650`, `CR651`, `CR652`, `CR653`, `CR654`, `CR655`, `CR656`, `CR657`, `CR658`, `CR659`, `CR660`, `CR661`, `CR662`, `CR663`, `CR664`, `CR665`, `CR666`, `CR667`, `CR668`, `CR669`, `CR670`, `CR671`, `CR672`, `CR673`, `CR674`, `CR675`, `CR676`, `CR677`, `CR678`, `CR679`, `CR680`, `CR681`, `CR682`, `CR683`, `CR684`, `CR685`, `CR686`, `CR687`, `CR688`, `CR689`, `CR690`, `CR691`, `CR692`, `CR693`, `CR694`, `CR695`, `CR696`, `CR697`, `CR698`, `CR699`, `CR700`, `CR701`, `CR702`, `CR703`, `CR704`, `CR705`, `CR706`, `CR707`, `CR708`, `CR709`, `CR710`, `CR711`, `CR712`, `CR713`, `CR714`, `CR715`, `CR716`, `CR717`, `CR718`, `CR719`, `CR720`, `CR721`, `CR722`, `CR723`, `CR724`, `CR725`, `CR726`, `CR727`, `CR728`, `CR729`, `CR730`, `CR731`, `CR732`, `CR733`, `CR734`, `CR735`, `CR736`, `CR737`, `CR738`, `CR739`, `CR740`, `CR741`, `CR742`, `CR743`, `CR744`, `CR745`, `CR746`, `CR747`, `CR748`, `CR749`, `CR750`, `CR751`, `CR752`, `CR753`, `CR754`, `CR755`, `CR756`, `CR757`, `CR758`, `CR759`, `CR760`, `CR761`, `CR762`, `CR763`, `CR764`, `CR765`, `CR766`, `CR767`, `CR768`, `CR769`, `CR770`, `CR771`, `CR772`, `CR773`, `CR774`, `CR775`, `CR776`, `CR777`, `CR778`, `CR779`, `CR780`, `CR781`, `CR782`, `CR783`, `CR784`, `CR785`, `CR786`, `CR787`, `CR788`, `CR789`, `CR790`, `CR791`, `CR792`, `CR793`, `CR794`, `CR795`, `CR796`, `CR797`, `CR798`, `CR799`, `CR800`, `CR801`, `CR802`, `CR803`, `CR804`, `CR805`, `CR806`, `CR807`, `CR808`, `CR809`, `CR810`, `CR811`, `CR812`, `CR813`, `CR814`, `CR815`, `CR816`, `CR817`, `CR818`, `CR819`, `CR820`, `CR821`, `CR822`, `CR823`, `CR824`, `CR825`, `CR826`, `CR827`, `CR828`, `CR829`, `CR830`, `CR831`, `CR832`, `CR833`, `CR834`, `CR835`, `CR836`, `CR837`, `CR838`, `CR839`, `CR840`, `CR841`, `CR842`, `CR843`, `CR844`, `CR845`, `CR846`, `CR847`, `CR848`, `CR849`, `CR850`, `CR851`, `CR852`, `CR853`, `CR854`, `CR855`, `CR856`, `CR857`, `CR858`, `CR859`, `CR860`, `CR861`, `CR862`, `CR863`, `CR864`, `CR865`, `CR866`, `CR867`, `CR868`, `CR869`, `CR870`, `CR871`, `CR872`, `CR873`, `CR874`, `CR875`, `CR876`, `CR877`, `CR878`, `CR879`, `CR880`, `CR881`, `CR882`, `CR883`, `CR884`, `CR885`, `CR886`, `CR887`, `CR888`, `CR889`, `CR890`, `CR891`, `CR892`, `CR893`, `CR894`, `CR895`, `CR896`, `CR897`, `CR898`, `CR899`, `CR900`, `CR901`, `CR902`, `CR903`, `CR904`, `CR905`, `CR906`, `CR907`, `CR908`, `CR909`, `CR910`, `CR911`, `CR912`, `CR913`, `CR914`, `CR915`, `CR916`, `CR917`, `CR918`, `CR919`, `CR920`, `CR921`, `CR922`, `CR923`, `CR924`, `CR925`, `CR926`, `CR927`, `CR928`, `CR929`, `CR930`, `CR931`, `CR932`, `CR933`, `CR934`, `CR935`, `CR936`, `CR937`, `CR938`, `CR939`, `CR940`, `CR941`, `CR942`, `CR943`, `CR944`, `CR945`, `CR946`, `CR947`, `CR948`, `CR949`, `CR950`, `CR951`, `CR952`, `CR953`, `CR954`, `CR955`, `CR956`, `CR957`, `CR958`, `CR959`, `CR960`, `CR961`, `CR962`, `CR963`, `CR964`, `CR965`, `CR966`, `CR967`, `CR968`, `CR969`, `CR970`, `CR971`, `CR972`, `CR973`, `CR974`, `CR975`, `CR976`, `CR977`, `CR978`, `CR979`, `CR980`, `CR981`, `CR982`, `CR983`, `CR984`, `CR985`, `CR986`, `CR987`, `CR988`, `CR989`, `CR990`, `CR991`, `CR992`, `CR993`, `CR994`, `CR995`, `CR996`, `CR997`, `CR998`, `CR999`, `CR1000`, `CR1001`, `CR1002`, `CR1003`, `CR1004`, `CR1005`, `CR1006`, `CR1007`, `CR1008`, `CR1009`, `CR1010`, `CR1011`, `CR1012`, `CR1013`, `CR1014`, `CR1015`, `CR1016`, `CR1017`, `CR1018`, `CR1019`, `CR1020`, `CR1021`, `CR1022`, `CR1023`, `CR1024`, `CR1025`, `CR1026`, `CR1027`, `CR1028`, `CR1029`, `CR1030`, `CR1031`, `CR1032`, `CR1033`, `CR1034`, `CR1035`, `CR1036`, `CR1037`, `CR1038`, `CR1039`, `CR1040`, `CR1041`, `CR1042`, `CR1043`, `CR1044`, `CR1045`, `CR1046`, `CR1047`, `CR1048`, `CR1049`, `CR1050`, `CR1051`, `CR1052`, `CR1053`, `CR1054`, `CR1055`, `CR1056`, `CR1057`, `CR1058`, `CR1059`, `CR1060`, `CR1061`, `CR1062`, `CR1063`, `CR1064`, `CR1065`, `CR1066`, `CR1067`, `CR1068`, `CR1069`, `CR1070`, `CR1071`, `CR1072`, `CR1073`, `CR1074`, `CR1075`, `CR1076`, `CR1077`, `CR1078`, `CR1079`, `CR1080`, `CR1081`, `CR1082`, `CR1083`, `CR1084`, `CR1085`, `CR1086`, `CR1087`, `CR1088`, `CR1089`, `CR1090`, `CR1091`, `CR1092`, `CR1093`, `CR1094`, `CR1095`, `CR1096`, `CR1097`, `CR1098`, `CR1099`, `CR1100`, `CR1101`, `CR1102`, `CR1103`, `CR1104`, `CR1105`, `CR1106`, `CR1107`, `CR1108`, `CR1109`, `CR1110`, `CR1111`, `CR1112`, `CR1113`, `CR1114`, `CR1115`, `CR1116`, `CR1117`, `CR1118`, `CR1119`, `CR1120`, `CR1121`, `CR1122`, `CR1123`, `CR1124`, `CR1125`, `CR1126`, `CR1127`, `CR1128`, `CR1129`, `CR1130`, `CR1131`, `CR1132`, `CR1133`, `CR1134`, `CR1135`, `CR1136`, `CR1137`, `CR1138`, `CR1139`, `CR1140`, `CR1141`, `CR1142`, `CR1143`, `CR1144`, `CR1145`, `CR1146`, `CR1147`, `CR1148`, `CR1149`, `CR1150`, `CR1151`, `CR1152`, `CR1153`, `CR1154`, `CR1155`, `CR1156`, `CR1157`, `CR1158`, `CR1159`, `CR1160`, `CR1161`, `CR1162`, `CR1163`, `CR1164`, `CR1165`, `CR1166`, `CR1167`, `CR1168`, `CR1169`, `CR1170`, `CR1171`, `CR1172`, `CR1173`, `CR1174`, `CR1175`, `CR1176`, `CR1177`, `CR1178`, `CR1179`, `CR1180`, `CR1181`, `CR1182`, `CR1183`, `CR1184`, `CR1185`, `CR1186`, `CR1187`, `CR118`

链接可以在编译时由静态编译器来完成，也可以在加载时和运行时由动态链接器来完成。链接器处理称为目标文件的二进制文件，它有三种不同的形式：可重定位、可执行的和可共享的。可重定位的目标文件由静态链接器合并成一个可执行的目标文件，它可以加载到内存中并执行。共享目标文件（共享库）是在运行时由动态链接器链接和加载的，或者隐含地在调用程序被加载和开始执行时，根据需要在程序调用 `dlopen` 库的函数时。

(第 5 章 1 分)

.第 6 章 hello 进程管理

.6.1 进程的概念与作用

像 hello 这样的程序在现代系统上运行时，操作系统会提供一种假象，就好像系统上只有这个程序在运行。程序看上去是独占地使用处理器、主存和 I/O 设备。处理器看上去就像在不间断地一条接一条地执行程序中的指令，即该程序的代码和数据是系统内存中唯一的对象。这些假象是通过进程的概念来实现的。

.6.2 简述壳 Shell-bash 的作用与处理流程

shell 是一个交互型的应用级程序，它代表用户运行其他程序。bash 是 shell 的一个变种。shell 执行一系列的读/求值步骤，然后终止。读步骤读取来自用户的一个命令行。求值步骤解析命令行，并代表用户运行程序。

.6.3 Hello 的 fork 进程创建过程

父进程通过调用 fork 函数创建一个新的运行的子进程。新创建的子进程几乎但不完全与父进程相同。子进程得到与父进程用户级虚拟地址空间相同的但是独立的一份副本，包括代码和数据段、堆、共享库以及用户栈。子进程还获得与父进程任何打开文件描述符相同的副本，这就意味着父进程调用 fork 时，子进程可以读写父进程中打开的任何文件。父进程和新创建的子进程之间最大的区别在于它们有不同的 PID。

fork 在新的子进程中运行相同的程序，新的子进程是父进程的一个复制品。

.6.4 Hello 的 execve 过程

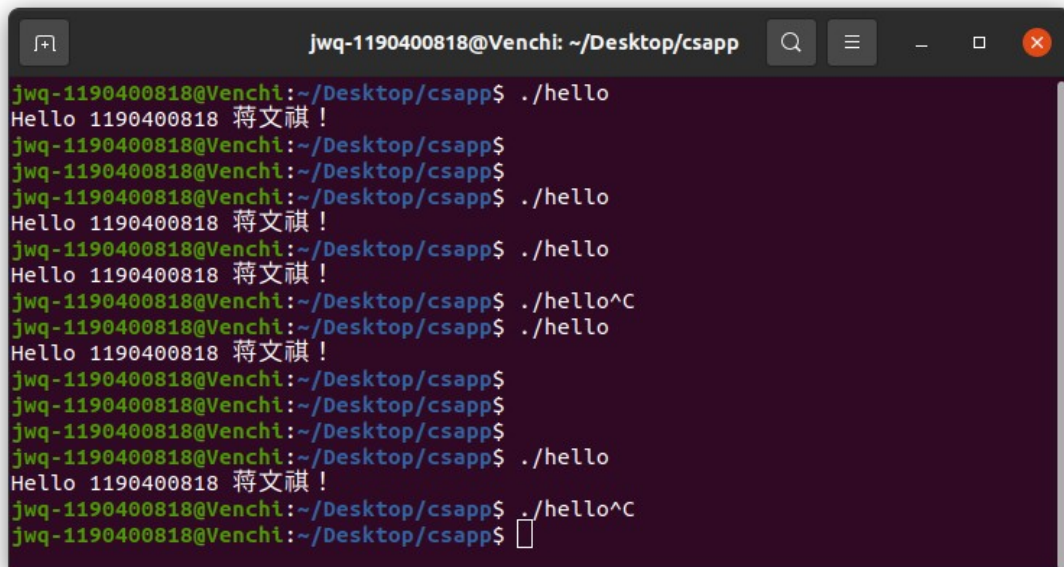
execve 函数在当前进程的上下文中加载并运行一个新程序。execve 函数加载并运行可执行目标文件 hello，且带参数列表 argv 和环境变量列表 envp。只有当出现错误时，例如找不到 hello，execve 才会返回到调用程序。所以，与 fork 调用一次返回两次不同，execve 调用一次并从不返回。

.6.5 Hello 的进程执行

运行应用程序代码的进程初始时是在用户模式中的。进程从用户模式变为内核模式的唯一方法是通过诸如中断、故障或者陷入系统调用这样的异常。

操作系统内核使用一种成为上下文切换的较高级形式的异常控制流来实现多任务。内核为每个任务维持一个上下文。上下文就是内核重新启动一个被抢占的进程所需的状态。在进程执行的某些时刻，内核可以决定抢占当前进程，并重新开始一个先前被抢占了的进程。当内核代表用户执行系统调用时，可能会发生上下文切换。中断也可能发生上下文切换。

.6.6 hello 的异常与信号处理



```
jqw-1190400818@Venchi: ~/Desktop/csapp
jqw-1190400818@Venchi:~/Desktop/csapp$ ./hello
Hello 1190400818 蒋文祺!
jqw-1190400818@Venchi:~/Desktop/csapp$
jqw-1190400818@Venchi:~/Desktop/csapp$
jqw-1190400818@Venchi:~/Desktop/csapp$ ./hello
Hello 1190400818 蒋文祺!
jqw-1190400818@Venchi:~/Desktop/csapp$ ./hello
Hello 1190400818 蒋文祺!
jqw-1190400818@Venchi:~/Desktop/csapp$ ./hello^C
jqw-1190400818@Venchi:~/Desktop/csapp$ ./hello
Hello 1190400818 蒋文祺!
jqw-1190400818@Venchi:~/Desktop/csapp$
jqw-1190400818@Venchi:~/Desktop/csapp$
jqw-1190400818@Venchi:~/Desktop/csapp$
jqw-1190400818@Venchi:~/Desktop/csapp$ ./hello
Hello 1190400818 蒋文祺!
jqw-1190400818@Venchi:~/Desktop/csapp$ ./hello^C
jqw-1190400818@Venchi:~/Desktop/csapp$
```

程序运行过程中键入 Ctrl-Z。键入 Ctrl-Z 会发送 SIGTSTP 信号给前台进程组的每个进程，结果是停止前台作业

使用 jobs 命令可以查看当前的作业

使用 ps 命令可以查看当前所有进程以及它们的 PID，进程包括 bash，hello 以及 ps

使用 pstree 命令将所有进程以树状图形式显示

使用 fg 命令可以使停止的 hello 进程继续在前台运行

使用 kill 命令可以给指定进程发送信号

.6.7 本章小结

进程的经典定义就是一个执行中程序的实例。系统中的每个程序都运行在某个进程的上下文中。每次用户通过向 shell 输入一个可执行目标文件的名字，运行程序时，shell 就会创建一个新的进程，然后在这个新的进程的上下文中运行这个可执行目标文件。

(第 6 章 1 分)

第 7 章 hello 的存储管理

7.1 hello 的存储器地址空间

逻辑地址是指由程序产生的与段相关的偏移地址部分。例如，在进行 C 语言指针编程中，可以使用 & 操作读取指针变量的值，这个值就是逻辑地址，是相对于当前进程数据段的地址。一个逻辑地址由两部份组成：段标识符和段内偏移量。

计算机系统的主存被组织成一个由 M 个连续的字节大小的单元组成的数组。每字节都有一个唯一的物理地址。地址空间是一个非负整数地址的有序集合：如果地址空间中的整数是连续的，称其为线性地址空间。

为了更加有效管理内存并且少出错，现代操作系统提供了一种对主存的抽象概念，虚拟内存。虚拟内存是硬件异常、硬件地址翻译、主存、磁盘文件和内核软件的完美交互，它为每个进程提供了一个大的、一致的和私有的地址空间。

7.2 Intel 逻辑地址到线性地址的变换-段式管理

逻辑地址由段标识符和段内偏移量两部分组成。段标识符由一个 16 位长的字段组成，称为段选择符。其中前 13 位是一个索引号，是对段描述符表的索引，每个段描述符由 8 个字节组成，具体描述了一个段后 3 位包含一些硬件细节，表示具体是代码段寄存器还是栈段寄存器还是数据段寄存器等。通过段标识符的前 13 位，可以直接在段描述符表中索引到具体的段描述符。每个段描述符中包含一个 Base 字段，它描述了一个段的开始位置的线性地址。将 Base 字段和逻辑地址中的段内偏移量连接起来就得到转换后的线性地址。

给定逻辑地址，看段选择符的最后一位是 0 还是 1，用于判断选择全局段描述符表还是局部段描述符表。再根据相应寄存器，得到其地址和大小。通过段标识符的前 13 位，可以在相应段描述符表中索引到具体的段描述符，得到 Base 字段，和段内偏移量连接起来最终得到转换后的线性地址。

7.3 Hello 的线性地址到物理地址的变换-页式管理

操作系统软件、MMU 中的地址翻译硬件和一个存放在物理内存中叫做页表的

数据结构，页表将虚拟页映射到物理页。每次地址翻译硬件将一个虚拟地址转换为物理地址时，都会读取页表。操作系统负责维护页表的内容，以及在磁盘与 DRAM 之间来回传送页。

.7.4 TLB 与四级页表支持下的 VA 到 PA 的变换

当页命中时，CPU 硬件执行步骤：

1. 处理器生成一个虚拟地址，并把它传给 MMU
2. MMU 生成 PTE 地址，并从高速缓存/主存请求得到它
3. 高速缓存/主存向 MMU 返回 PTE
4. MMU 构造物理地址，并把它传送给高速缓存/主存
5. 高速缓存/主存返回所请求的数据字给处理器

.7.5 三级 Cache 支持下的物理内存访问

硬件解码地址是，它也执行相似的任务。

开始时，MMU 从虚拟地址中抽取出 VPN，并检查 TLB，看它是否因为前面的某个内存引用缓存了 PTE 的一个副本。TLB 从 VPN 中抽取出 TLB 索引和 TLB 标记，命中，将缓存的 PPN 返回给 MMU。

如果 TLB 不命中，MMU 需要从主存中取出相应的 PTE。MMU 有了形成物理地址所需要的所有东西。它通过将来自 PTE 的 PPN 和来自虚拟地址的 VPO 连接起来，形成了物理地址。

MMU 发送物理地址给缓存，缓存从物理地址中抽取出缓存偏移 CO、缓存组索引 CI 以及缓存标记。

.7.6 hello 进程 fork 时的内存映射

当 fork 函数被当前进程调用时，内核为新进程创建各种数据结构，并分配给它一个唯一的 PID。为了给这个新进程创建虚拟内存，它创建了当前进程的 mm_struct、区域机构和页表的原样副本。它将两个进程中的每个页面都标记为只读，并将两个进程中的每个区域结构都标记为私有的写时复制。

当 fork 函数在新进程中返回时，新进程现在的虚拟内存刚好和调用 fork 时存

在的虚拟内存相同。当这两个进程中的任一个后来进行写操作时，写时复制机制就会创建新页面，因此，也就为每个进程保持了私有地址空间的抽象概念。

.7.7 hello 进程 `execve` 时的内存映射

`execve` 函数在当前进程中加载并运行包含在可执行目标文件 `hello` 中的程序，用 `hello` 程序有效替代了当前程序。加载并运行 `hello` 需要以下步骤：

1. 删除已存在的用户区域
2. 映射私有区域
3. 映射共享区域
4. 设置程序计数器

下一次调度这个进程时，它将从这个入口点开始执行。Linux 将根据需要换入代码和数据页面。

.7.8 缺页故障与缺页中断处理

处理缺页要求硬件和操作系统内核协作完成：

1. 2. 3. 与页命中时相同
4. PTE 中的有效位是零，所以 MMU 触发了一次异常，传递 CPU 中的控制到操作系统内核中的缺页异常处理程序
5. 缺页处理程序确定出物理内存中的牺牲页，如果这个页面已经被修改了，则把它换出到磁盘
6. 缺页处理程序页面调入新的页面，并更新内存中的 PTE
7. 缺页处理程序返回到原来的进程，再次执行导致缺页的指令。CPU 将引起缺页的虚拟地址重新发送给 MMU。因为虚拟页面现在缓存在物理内存中，所以就会命中。

.7.9 动态存储分配管理

动态内存分配器维护着一个进程的虚拟内存区域，称为 *heap*。堆是一个请求二进制零的区域，它紧接在为初始化的数据区域后开始，并向上生长。对于每个进程，内核维护着一个变量 *brk*，它指向堆的顶部。

分配器将堆视为一组不同大小的块的集合来维护。每个块就是一个连续的虚

拟内存片，已分配的或空闲的。已分配的块显式地保留为供应用程序使用。空闲块可用来分配。空闲块保持空闲，直到它显式地被应用程序分配。一个已分配的块保持已分配状态，直到它被释放，这种释放要么是应用程序显式执行的，要么是内存分配器自身隐式执行的。

.7.10 本章小结

虚拟内存是对主存的一个抽象。支持虚拟内存的处理器通过使用一种叫做虚拟寻址的间接形式来引用主存。处理器产生一个虚拟地址，在被发送到主存之前这个地址被翻译成一个物理地址。从虚拟地址空间到物理地址空间的地址翻译要求硬件和软件紧密合作。专门的硬件通过使用页表来翻译虚拟地址，而页表的内容是由操作系统提供的。

虚拟内存提供三个重要的功能。第一，它在主存中自动缓存最近使用的存放在磁盘上的虚拟地址空间的内容。虚拟内存缓存中的块叫做页。对磁盘上的页的引用会触发缺页，缺页将控制转移到操作系统中的一个缺页处理程序。该处理程序将页面从磁盘复制到主存缓存，如果必要，将写回被驱逐的页。第二，虚拟内存简化了内存管理，进而又简化了链接、在进程间共享数据、进程的内存分配以及程序加载。最后，虚拟内存通过在每条页表条目中加入保护位，从而简化了内存保护

(第7章2分)

.第 8 章 hello 的 IO 管理

.8.1 Linux 的 IO 设备管理方法

设备的模型化：文件

设备管理：unix io 接口

Linux 提供了少量的基于 Unix I/O 模型的系统级函数，它们允许应用程序打开、关闭、读和写文件，提取文件的元数据，以及执行 I/O 重定向。LINUX 的读和写操作会出现不足值，应用程序必须能正确地预计和处理这种情况。应用程序不应直接调用 UNIX I/O 函数，而应该使用 RIO 包，RIO 包通过反复执行读写操作，直到传送完成所有的请求数据，自动处理不足值。

.8.2 简述 Unix IO 接口及其函数

所有的 I/O 设备都被模式化为文件，而所有的输入输出都被当作对相应文件的读和写来执行，这使得所有的输入输出都能以一种统一且一致的方式来执行：

1. 打开文件：内核记录有关这个打开文件的所有信息，应用程序只需记住这个描述符。

2. LINUX SHELL 创建的每个进程开始时都有三个打开的文件：STDIN_FILENO\STDOUT_FILENO\STDERR_FILENO，它们可用来替代显式的描述符值。

3. 改变当前的文件位置。应用程序能通过执行 seek 操作，显式地设置文件的当前位置为 k。

4. 读写文件

5. 关闭文件

.8.3 printf 的实现分析

<https://www.cnblogs.com/pianist/p/3315801.html>

从 vsprintf 生成显示信息，到 write 系统函数，到陷阱-系统调用 int 0x80 或 syscall 等。

字符显示驱动子程序：从 ASCII 到字模库到显示 vram（存储每一个点的 RGB 颜色信息）。

显示芯片按照刷新频率逐行读取 vram，并通过信号线向液晶显示器传输每一个点（RGB 分量）。

研究 printf 的实现，首先来看看 printf 函数的函数体

```
int printf(const char *fmt, ...)
```

```
{
```

```
int i;
```

```
char buf[256];
```

```
va_list arg = (va_list)((char*)&fmt + 4);
```

```
i = vsprintf(buf, fmt, arg);
```

```
write(buf, i);
```

```
return i;
```

```
}
```

代码位置：D:/~/funny/kernel/printf.c

在形参列表里有这么一个 token：...

这个是可变形参的一种写法。

当传递参数的个数不确定时，就可以用这种方式来表示。

很显然，我们需要一种方法，来让函数体可以知道具体调用时参数的个数。

先来看 printf 函数的内容：

这句：

```
va_list arg = (va_list)((char*)&fmt + 4);
```

va_list 的定义：

```
typedef char *va_list
```

这说明它是一个字符指针。

其中的：(char*)&fmt + 4 表示的是...中的第一个参数。

如果不懂，我再慢慢的解释：

C 语言中，参数压栈的方向是从右往左。

也就是说，当调用 printf 函数的适合，先是最右边的参数入栈。

fmt 是一个指针，这个指针指向第一个 const 参数 (const char *fmt) 中的第一个元素。

fmt 也是个变量，它的位置，是在栈上分配的，它也有地址。

对于一个 char * 类型的变量，它入栈的是指针，而不是这个 char * 型变量。

换句话说：

你 sizeof(p) (p 是一个指针，假设 p=&i, i 为任何类型的变量都可以)

得到的都是一个固定的值。（我的计算机中都是得到的 4）

当然，我还要补充的一点是：栈是从高地址向低地址方向增长的。

ok!

现在我想你该明白了：为什么说(char*)&fmt + 4) 表示的是...中的第一个参数的地址。

下面我们来看看下一句：

```
i = vsprintf(buf, fmt, arg);
```

让我们来看看 vsprintf(buf, fmt, arg)是什么函数。

```
int vsprintf(char *buf, const char *fmt, va_list args)
{
    char* p;
    char tmp[256];
    va_list p_next_arg = args;

    for (p=buf; *fmt; fmt++) {
        if (*fmt != '%') {
            *p++ = *fmt;
            continue;
        }

        fmt++;

        switch (*fmt) {
            case 'x':
                itoa(tmp, *((int*)p_next_arg));
                strcpy(p, tmp);
                p_next_arg += 4;
                p += strlen(tmp);
                break;
            case 's':
                break;
            default:
                break;
        }
    }

    return (p - buf);
}
```

我们还是先不看看它的具体内容。

想想 printf 要干什么吧

它接受一个格式化的命令，并把指定的匹配的参数字符串格式化输出。

ok，看看 i = vsprintf(buf, fmt, arg);

vsprintf 返回的是一个长度，我想你已经猜到了：是的，返回的是要打印出来的字符串的长度
其实看看 printf 中后面的一句：write(buf, i);你也该猜出来了。

write，顾名思义：写操作，把 buf 中的 i 个元素的值写到终端。

所以说：vsprintf 的作用就是格式化。它接受确定输出格式的格式字符串 fmt。用格式字符串对个数变化的参数进行格式化，产生格式化输出。

我代码中的 vsprintf 只实现了对 16 进制的格式化。

你只要明白 vsprintf 的功能是什么，就会很容易看懂上面的代码。

下面的 write(buf, i); 的实现就有点复杂了

如果你是 os，一个用户程序需要你打印一些数据。很显然：打印的最底层操作肯定和硬件有关。所以你就必须得对程序的权限进行一些限制：

让我们假设个情景：

一个应用程序对你说：os 先生，我需要把存在 buf 中的 i 个数据打印出来，可以帮我么？

os 说：好的，咱俩谁跟谁，没问题啦！把 buf 给我吧。

然后，os 就把 buf 拿过来。交给自己的小弟（和硬件操作的函数）来完成。

只好通知这个应用程序：兄弟，你的事我办的妥妥当当！（os 果然大大的狡猾 ^_^）

这样 应用程序就不会取得一些超级权限，防止它做一些违法的事。（安全啊安全）

让我们追踪下 write 吧：

```
write:
    mov eax, _NR_write
    mov ebx, [esp + 4]
    mov ecx, [esp + 8]
    int INT_VECTOR_SYS_CALL
```

位置：d:~/kernel/syscall.asm

这里是给几个寄存器传递了几个参数，然后一个 int 结束

想想我们汇编里面学的，比如返回到 dos 状态：

我们这样用的

```
mov ax, 4c00h
int 21h
```

为什么用后面的 int 21h 呢？

这是为了告诉编译器：号外，号外，我要按照给你的方式（传递的各个寄存器的值）变形了。

编译器一查表：哦，你是要变成这个样子啊。no problem！

其实这么说并不严紧，如果你看了一些关于保护模式编程的书，你就会知道，这样的 int 表示要调用中断门了。通过中断门，来实现特定的系统服务。

我们可以找到 INT_VECTOR_SYS_CALL 的实现：


```
init_idt_desc(INT_VECTOR_SYS_CALL, DA_386IGate, sys_call, PRIVILEGE_USER);
```

位置：d:~/kernel/protect.c

如果你不懂，没关系，你只需要知道一个 int INT_VECTOR_SYS_CALL 表示要通过系统来调用 sys_call 这个函数。（从上面的参数列表中也该能够猜出大概）

好了，再来看看 sys_call 的实现：

```
sys_call:
```

```
call save
```

```
push dword [p_proc_ready]
```

```
sti
```

```
push ecx
```

```
push ebx
```

```
call [sys_call_table + eax * 4]
```

```
add esp, 4 * 3
```

```
mov [esi + EAXREG - P_STACKBASE], eax
```

```
cli
```

```
ret
```

位置：~/kernel/kernel.asm

一个 call save，是为了保存中断前进程的状态。

靠！

太复杂了，如果详细的讲，设计到的东西实在太多了。

我只在乎我所在乎的东西。sys_call 实现很麻烦，我们不妨不分析 funny os 这个操作系统了先假设这个 sys_call 就一单纯的小女孩。她只有实现一个功能：显示格式化了了的字符串。

这样，如果只是理解 printf 的实现的话，我们完全可以这样写 sys_call：

```
sys_call:
```

```
;ecx 中是要打印出的元素个数
```

```
;ebx 中的是要打印的 buf 字符数组中的第一个元素
```

```
;这个函数的功能就是不断的打印出字符，直到遇到：'\0'
```

```
;[gs:edi]对应的是 0x80000h：0 采用直接写显存的方法显示字符串
```

```
xor si,si
```

```
mov ah,0Fh
```

```
mov al,[ebx+si]
```

```
cmp al,'\0'
```

```
je .end
```

```

mov [gs:edi],ax
inc si
loop:
sys_call

.end:
ret

```

ok!就这么简单!

恭喜你，重要弄明白了 printf 的最最底层的实现!

如果你有机会看 linux 的源代码的话，你会发现，其实它的实现也是这种思路。

freedos 的实现也是这样

比如在 linux 里，printf 是这样表示的：

```

static int printf(const char *fmt, ...)
{
    va_list args;
    int i;

    va_start(args, fmt);
    write(1,printfbuf,i=vsprintf(printfbuf, fmt, args));
    va_end(args);
    return i;
}

```

va_start

va_end 这两个函数在我的 blog 里有解释，这里就不多说了

它里面的 vsprintf 和我们的 vsprintf 是一样的功能。

不过它的 write 和我们的不同，它还有个参数：1

这里我可以告诉你：1 表示的是 tty 所对应的一个文件句柄。

在 linux 里，所有设备都是被当作文件来看待的。你只需要知道这个 1 就是表示往当前显示器里写入数据

在 freedos 里面，printf 是这样的：

```

int VA_CDECL printf(const char *fmt, ...)
{
    va_list arg;
    va_start(arg, fmt);
    charp = 0;
    do_printf(fmt, arg);
    return 0;
}

```

看起来似乎是 do_printf 实现了格式化和输出。

我们来看看 do_printf 的实现：

```
STATIC void do_printf(CONST BYTE * fmt, va_list arg)
```

```
{
    int base;
    BYTE s[11], FAR * p;
    int size;
    unsigned char flags;

    for (;*fmt != '\0'; fmt++)
    {
        if (*fmt != '%')
        {
            handle_char(*fmt);
            continue;
        }

        fmt++;
        flags = RIGHT;

        if (*fmt == '-')
        {
            flags = LEFT;
            fmt++;
        }

        if (*fmt == '0')
        {
            flags |= ZEROSFILL;
            fmt++;
        }

        size = 0;
        while (1)
        {
            unsigned c = (unsigned char)(*fmt - '0');
            if (c > 9)
                break;
            fmt++;
            size = size * 10 + c;
        }

        if (*fmt == 'l')
        {
            flags |= LONGARG;
            fmt++;
        }
    }
}
```

```
}

switch (*fmt)
{
case '\0':
    va_end(arg);
    return;

case 'c':
    handle_char(va_arg(arg, int));
    continue;

case 'p':
    {
        UWORD w0 = va_arg(arg, unsigned);
        char *tmp = charp;
        sprintf(s, "%04x:%04x", va_arg(arg, unsigned), w0);
        p = s;
        charp = tmp;
        break;
    }

case 's':
    p = va_arg(arg, char *);
    break;

case 'F':
    fmt++;
    /* we assume %Fs here */
case 'S':
    p = va_arg(arg, char FAR *);
    break;

case 'i':
case 'd':
    base = -10;
    goto lprt;

case 'o':
    base = 8;
    goto lprt;

case 'u':
    base = 10;
    goto lprt;

case 'X':
```

```
case 'x':
base = 16;

lprt:
{
long currentArg;
if (flags & LONGARG)
currentArg = va_arg(arg, long);
else
{
currentArg = va_arg(arg, int);
if (base >= 0)
currentArg = (long)(unsigned)currentArg;
}
ltob(currentArg, s, base);
p = s;
}
break;

default:
handle_char('?');

handle_char(*fmt);
continue;

}
{
size_t i = 0;
while(p[i]) i++;
size -= i;
}

if (flags & RIGHT)
{
int ch = ' ';
if (flags & ZEROSFILL) ch = '0';
for (; size > 0; size--)
handle_char(ch);
}
for (; *p != '\0'; p++)
handle_char(*p);

for (; size > 0; size--)
handle_char(' ');
}
va_end(arg);
}
```

这个就是比较完整的格式化函数

里面多次调用一个函数：handle_char

来看看它的定义：

```
STATIC VOID handle_char(COUNT c)
{
    if (charp == 0)
        put_console(c);
    else
        *charp++ = c;
}
```

里面又调用了 put_console

显然，从函数名就可以看出来：它是用来显示的

```
void put_console(int c)
{
    if (buff_offset >= MAX_BUFSIZE)
    {
        buff_offset = 0;
        printf("Printf buffer overflow!\n");
    }
    if (c == '\n')
    {
        buff[buff_offset] = 0;
        buff_offset = 0;
#ifdef __TURBOC__
        _ES = FP_SEG(buff);
        _DX = FP_OFF(buff);
        _AX = 0x13;
        __int__(0xe6);
#elif defined(I86)
        asm
        {
            push ds;
            pop es;
            mov dx, offset buff;
            mov ax, 0x13;
            int 0xe6;
        }
#endif
    }
    else
    {
        buff[buff_offset] = c;
        buff_offset++;
    }
}
```

```
}
```

.8.4 getchar 的实现分析

异步异常-键盘中断的处理：键盘中断处理子程序。接受按键扫描码转成 ascii 码，保存到系统的键盘缓冲区。

getchar 等调用 read 系统函数，通过系统调用读取按键 ascii 码，直到接受到回车键才返回。

getchar 的源代码：

```
int getchar(void)

{

    static char buf[BUFSIZ];

    static char* bb = buf;

    static int n = 0;

    if(n == 0)

    {

        n = read(0, buf, BUFSIZ);

        bb = buf;

    }

    return(--n >= 0)?(unsigned char) *bb++ : EOF;

}
```

getchar 函数会从 stdin 输入流中读入一个字符。调用 getchar 时，会等待用户输入，输入回车后，输入的字符会存放在缓冲区中。第一次调用 getchar 时，需要从

键盘输入，但如果输入了多个字符，之后的 `getchar` 会直接从缓冲区中读取字符。`getchar` 的返回值是读取字符的 ASCII 码，若出错则返回-1。

异步异常-键盘中断的处理：键盘中断处理子程序。接受按键扫描码转成 `ascii` 码，保存到系统的键盘缓冲区。

`getchar` 等调用 `read` 系统函数，通过系统调用读取按键 `ascii` 码，直到接受到回车键才返回。

.8.5 本章小结

LINUX 内核使用三个相关的数据结构来表示打开的文件。描述符表中的表项指向打开文件表中的表项，而打开文件表中的表项又指向 V-NODE 表中的表项。每个进程都有它自己单独的描述符表，而所有的进程共享同一个打开文件表和 V_NODE 表。理解这些结构的一般组成就能时我们清楚地理解文件共享和 I/O 重定向。

(第 8 章 1 分)

.结论

处理器读取并解释存放在主存里的二进制指令。系统中的存储设备划分为层次结构。

操作系统内核时应用程序和硬件之间的媒介。它提供三个基本的抽象：文件是对 I/O 设备的抽象；虚拟内存是对主存和磁盘的抽象；进程是处理器、主存和 I/O 设备的抽象。

.附件

hello.i: C 预处理器产生的一个 ASCII 码的中间文件, 用于分析预处理过程。

hello.s: C 编译器产生的一个 ASCII 汇编语言文件, 用于分析编译的过程。

hello.o: 汇编器产生的可重定位目标程序, 用于分析汇编的过程。

hello: 链接器产生的可执行目标文件, 用于分析链接的过程。

hello.txt: hello 的反汇编文件, 用于分析可执行目标文件 hello。

.参考文献

为完成本次大作业你翻阅的书籍与网站等

- [1] <https://www.cnblogs.com/pianist/p/3315801.html>
- [2] <https://blog.csdn.net/lxcshax/article/details/117875520#t49>
- [3] RANDALE.BRYANT, DAVIDR.O‘HALLARON. 深入理解计算机系统[M]. 机械工业出版社, 2011.
- [4] <https://www.cnblogs.com/clover-toeic/p/3851102.html>
- [5] <https://www.cnblogs.com/pianist/p/3315801.html>

(参考文献 0 分，缺失 -1 分)