

**计算机系统**

**大作业**

题 目 程序人生-Hello’s P2P

专 业 计算机科学与技术

学　　 号 2021113371

班　　 级 2103103

学 生 李富行

指 导 教 师 刘宏伟

**计算机科学与技术学院**

**2022年5月**

**摘 要**

HelloWorld是几乎全世界程序员编写的第一个程序，我们缓慢地输入HelloWorld 的几行代码，点击运行，惊喜地看到输出栏中的“Hello，World!”。但是很快我们就不再理会这个程序，去学习其他的语法。然而，在这个简单的hello中，囊括一个程序运行的所有过程，蕴含着无数计算机科学家的思想精华。

本文通过跟踪hello的一生，介绍了hello从代码编辑器到运行最后结束的过程，对计算机底层进行了较深的分析。

**关键词：**计算机系统；Linux；C语言；

**目 录**

[第1章 概述 - 4 -](#_Toc532238396)

[1.1 Hello简介 - 4 -](#_Toc532238397)

[1.2 环境与工具 - 4 -](#_Toc532238398)

[1.3 中间结果 - 4 -](#_Toc532238399)

[1.4 本章小结 - 4 -](#_Toc532238400)

[第2章 预处理 - 5 -](#_Toc532238401)

[2.1 预处理的概念与作用 - 5 -](#_Toc532238402)

[2.2在Ubuntu下预处理的命令 - 5 -](#_Toc532238403)

[2.3 Hello的预处理结果解析 - 5 -](#_Toc532238404)

[2.4 本章小结 - 5 -](#_Toc532238405)

[第3章 编译 - 6 -](#_Toc532238406)

[3.1 编译的概念与作用 - 6 -](#_Toc532238407)

[3.2 在Ubuntu下编译的命令 - 6 -](#_Toc532238408)

[3.3 Hello的编译结果解析 - 6 -](#_Toc532238409)

[3.4 本章小结 - 6 -](#_Toc532238410)

[第4章 汇编 - 7 -](#_Toc532238411)

[4.1 汇编的概念与作用 - 7 -](#_Toc532238412)

[4.2 在Ubuntu下汇编的命令 - 7 -](#_Toc532238413)

[4.3 可重定位目标elf格式 - 7 -](#_Toc532238414)

[4.4 Hello.o的结果解析 - 7 -](#_Toc532238415)

[4.5 本章小结 - 7 -](#_Toc532238416)

[第5章 链接 - 8 -](#_Toc532238417)

[5.1 链接的概念与作用 - 8 -](#_Toc532238418)

[5.2 在Ubuntu下链接的命令 - 8 -](#_Toc532238419)

[5.3 可执行目标文件hello的格式 - 8 -](#_Toc532238420)

[5.4 hello的虚拟地址空间 - 8 -](#_Toc532238421)

[5.5 链接的重定位过程分析 - 8 -](#_Toc532238422)

[5.6 hello的执行流程 - 8 -](#_Toc532238423)

[5.7 Hello的动态链接分析 - 8 -](#_Toc532238424)

[5.8 本章小结 - 9 -](#_Toc532238425)

[第6章 hello进程管理 - 10 -](#_Toc532238426)

[6.1 进程的概念与作用 - 10 -](#_Toc532238427)

[6.2 简述壳Shell-bash的作用与处理流程 - 10 -](#_Toc532238428)

[6.3 Hello的fork进程创建过程 - 10 -](#_Toc532238429)

[6.4 Hello的execve过程 - 10 -](#_Toc532238430)

[6.5 Hello的进程执行 - 10 -](#_Toc532238431)

[6.6 hello的异常与信号处理 - 10 -](#_Toc532238432)

[6.7本章小结 - 10 -](#_Toc532238433)

[结论 - 14 -](#_Toc532238451)

[附件 - 15 -](#_Toc532238452)

[参考文献 - 16 -](#_Toc532238453)

# 第1章 概述

## 1.1 Hello简介

根据Hello的自白，利用计算机系统的术语，简述Hello的P2P，020的整个过程。

**P2P**：在文本编辑器中将hello的代码输入，并保存为.c格式的文件，形成hello.c文件。这就是程序（Program），这是hello程序的生命周期的开始。在OS（例如Linux）中，通过交互式应用程序Shell，输入命令实现对hello.c从源文件到目标文件的转化。源程序通过cpp（预处理器）预处理，ccl（编译器）编译，as（汇编器）汇编，最后通过ld（链接器）链接生成hello可执行目标程序并将其保存到磁盘中。

**020**：在Shell运行该hello程序时，Shell调用fork函数创建子进程，并通过execve函数将hello程序载入并创建运行环境，比如分配虚拟内存，运行完成后，Shell回收该进程，释放内存空间。

## 1.2 环境与工具

**硬件环境**：X86-64 Intel i7 10th 16 GB RAM 512 GHD Disk

**软件环境**：Windows 10 VMware 16 Ubuntu 20.04 LTS

**开发调试工具**：GDB，EDB，Visual Studio Code，Vim，gcc

## 1.3 中间结果

1. **hello.c**：源代码文件。
2. **hello.i**：预处理后的文本文件。
3. **hello.s**：编译后的汇编文件。
4. **hello.o**：汇编后的可重定位文件。
5. **hello**：链接后的可执行文件。
6. **disa\_hello.s**：反汇编hello.o的汇编代码。
7. **disa\_hello\_2.s**：反汇编hello的汇编代码。

## 1.4 本章小结

本章对hello程序运行的P2P和020过程进行了简单的介绍，列出了此次大作业所使用的相关工具和软硬件环境，最后介绍了文中所用到的文件和作用。

# 第2章 预处理

## 2.1 预处理的概念与作用

**概念**：在预处理时，预处理器需要根据以字符#开头的命令，修改原始的C程序。其实，就是在对源程序做文本替换的操作。比如hello.c第一行的“#include<stdio.h>” 预处理时预处理器会读取stdio.h并把它插入到Hello程序中。

**作用**：在代码编写的过程是为了方便程序员而设计这些#开头的命令，而为了后续编译器的方便，需要对代码进行替换。

## 2.2在Ubuntu下预处理的命令

**预处理命令**：gcc -E hello.c -o hello.i



图1 预处理命令

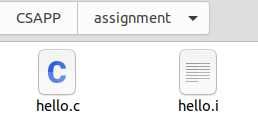


图2 文件列表

## 2.3 Hello的预处理结果解析

打开hello.i查看文件。

开头时引入外部库.h文件。



图3 hello.i文本文件 1

接下来是typedef进行数据类型名称替换。

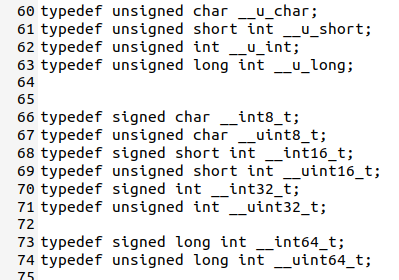


图4 hello.i文本文件 2

引入外部函数：

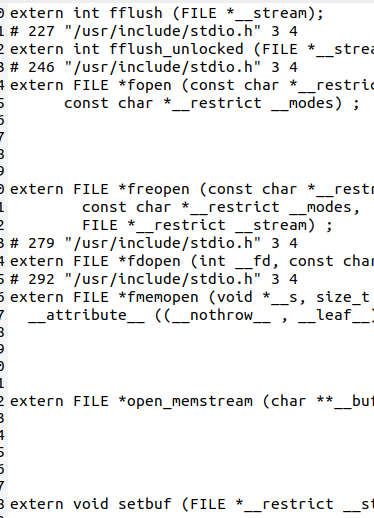


图5 hello.i文本文件 3

最后，是我们编写的main函数代码：

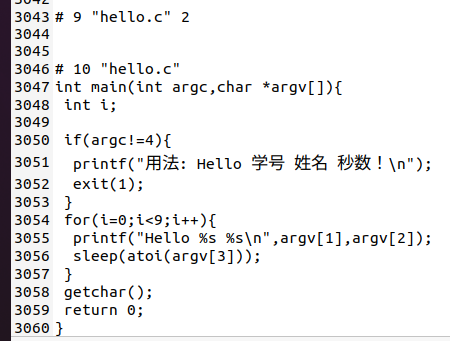


图6 hello.i文本文件 4

## 2.4 本章小结

本章介绍了hello.c的预处理过程，大致分析了预处理后形成的hello.i文件。可以知道，仅23行的.c文件预处理后的文件竟有3000行。如果编写一个hello程序需要3000行，这样的效率是极其低下的。这也就是预处理的意义：能让我们轻松写出可读性高，方便修改，利于调试的代码。

# 第3章 编译

## 3.1 编译的概念与作用

**概念**：编译是将预处理后的文本文件.i翻译为汇编语言的文本文件.s。

编译程序把一个源程序翻译成目标程序的工作过程分为五个阶段：词法分析，语法分析，语义检查和中间代码生成，代码优化，目标代码生成。主要是进行词法分析和语法分析，又称为源程序分析，分析过程中发现有语法错误，给出提示信息。

**作用**：将高级程序语言翻译为统一的，接近机器语言，对机器友好的汇编语言。

## 3.2 在Ubuntu下编译的命令

编译命令：gcc -S hello.i -o hello.s



图7 编译命令

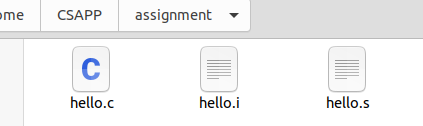


图8 文件列表

## 3.3 Hello的编译结果解析

编译生成了hello.s文件，为汇编代码，该小节将针对hello程序具体介绍编译器怎么处理C语言中各个数据类型以及各类操作的。

1. 常量
   1. 字符串常量：

"用法: Hello 学号 姓名 秒数！\n"

"Hello %s %s\n"。

汇编文件在开头为.LC0和.LC1中存放这两个字符串，其中一个汉字对应一个\xxx。这些常量存放在.rodata节，意为只读数据。

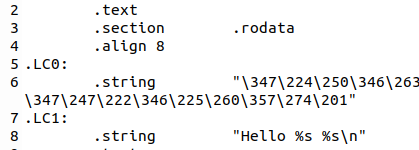


图9 hello.s文件 1

* 1. 立即数常量

如for循环中9这个常量：



图10 hello.s文件 2

1. 变量
   1. 局部变量

“int i”

局部变量存放在寄存器或栈中。

查看汇编代码：

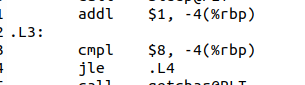


图11 hello.s文件 3

可以知道，i存放在栈中。

1. 算术操作

“i++”：

如上图，addl指令对i进行+1。

1. 赋值:

“i = 0”:



图12 hello.s文件 4

通过movl指令赋初值。

1. 比较语句：

比较通过cmp等指令来实现，根据两个操作数的差来设置条件码。

1. 数组/指针操作

“sleep(atoi(argv[3]))”：

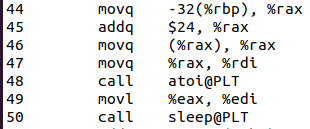


图13 hello.s文件 5

可以知道%rax先保存了argv[3]的地址。

1. 函数调用及返回：

如上图中调用的atoi函数，%rdi为参数，存放着字符串的首地址，%eax为返回值，为一个整数。

## 3.4 本章小结

本节主要介绍编译器通过编译由.i文件生成汇编语言的.s文件的过程，并分析了变量，赋值，循环等各类C语言的基本语句的汇编表示。汇编语言和高级语言很不同，即使是高级语言中一个简单的条件语句或者循环语句在汇编语言中都需要涉及到更多步骤来实现。学习汇编语言与编译，使我们能够真正的理解计算机底层的一些执行方法，有利于我们以后对程序的优化或调试。

# 第4章 汇编

## 4.1 汇编的概念与作用

**概念**：把汇编语言翻译成机器语言的过程称为汇编，汇编器同时将汇编程序（.s文件）打包成可重定位目标程序（.o文件）。这里的.o是二进制文件，而.s仍然是文本文件。

**作用**：通过汇编，汇编代码转化为了计算机能够完全理解的机器代码。

## 4.2 在Ubuntu下汇编的命令

汇编命令：gcc -c -m64 -no-pie -fno-PIC hello.s -o hello.o



图14 汇编命令

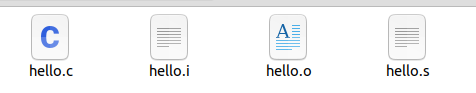


图15 文件列表

## 4.3 可重定位目标elf格式

分析hello.o的ELF格式，用readelf等列出其各节的基本信息，特别是重定位项目分析。

分析：

1. ELF头

命令：readelf -h -hello.o

ELF头首先以一个16字节的序列开始，这个序列描述了生成该文件的系统的 字的大小和字节顺序。剩下部分就如下图所示，列出了包含帮助链接器语法分析和解释目标文件的信息。其中包括ELF头的大小（64字节），目标文件的类型（REL可重定位文件），机器类型（AMD X86-64），节头部表的文件偏移，以及节头部表中条目的大小和数量。

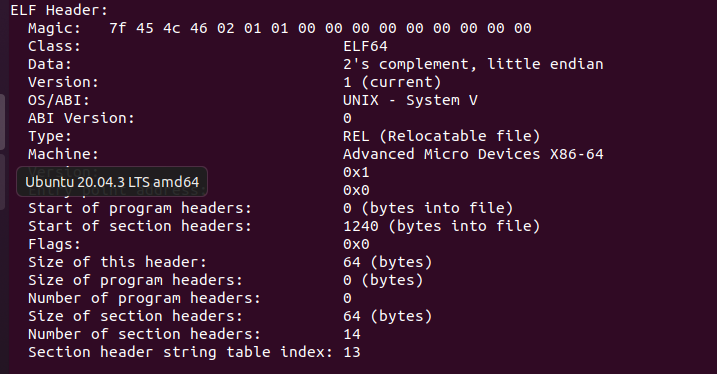


图16 ELF头

1. 节头部表

命令：readelf -S hello.o

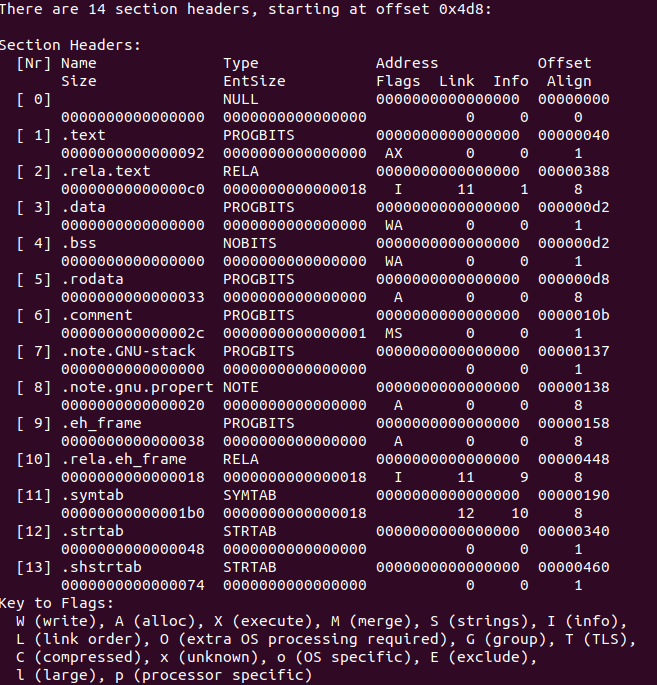


图17 节头部表

节头部表描述了 hello.o 中文件中各个节的语义，包括节的类型、位置和大小等信息

1. 符号表（.symtab）

命令：readelf -s hello.o

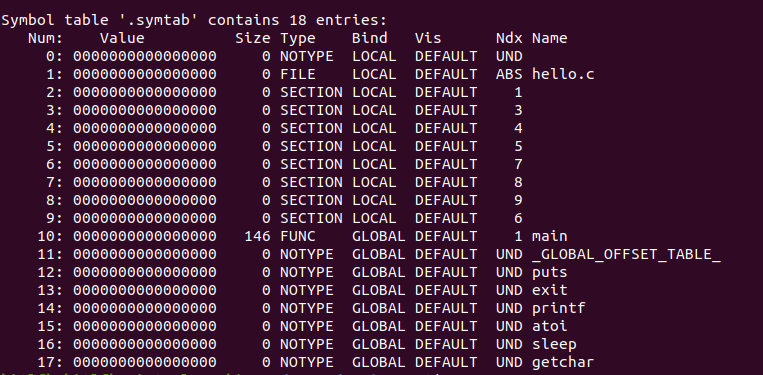


图18 符号表

可以看到.c文件中用到的main，atoi，exit，sleep函数。

1. 重定位条目

命令：readelf -r hello.o

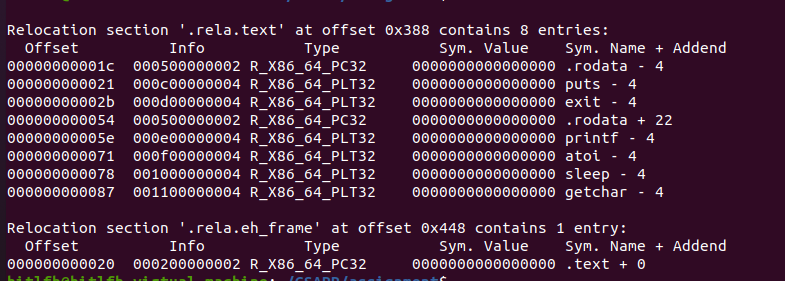


图19 重定位条目

当汇编器生成 hello.o 后，它并不知道数据和代码最终将放在内存中的什么位置。它也不知道这个模块引用的任何外部定义的函数或者全局变量的位置。所以，无论何时汇编器遇到对最终位置未知的目标引用，它就会生成一个重定位条目。重定位条目在链接时告诉链接器目标文件合并时如何修改应用。

重定位条目的类型这里有两种，R\_X86\_64\_32意思是重定位时使用一个32位的绝对地址的引用，通过绝对寻址，CPU直接使用在指令中编码的32位值作为有效地址，不需要进一步修改。R\_X86\_64\_PC32意思是重定位时使用一个32位PC相对地址的引用。一个PC相对地址就是据程序计数器的当前运行值的偏移量。

## 4.4 Hello.o的结果解析

命令：objdump -d -r hello.o > disa\_hello.s

生成hello.o的反汇编文件。

分析hello.o的反汇编，并与第3章的 hello.s进行对照分析。

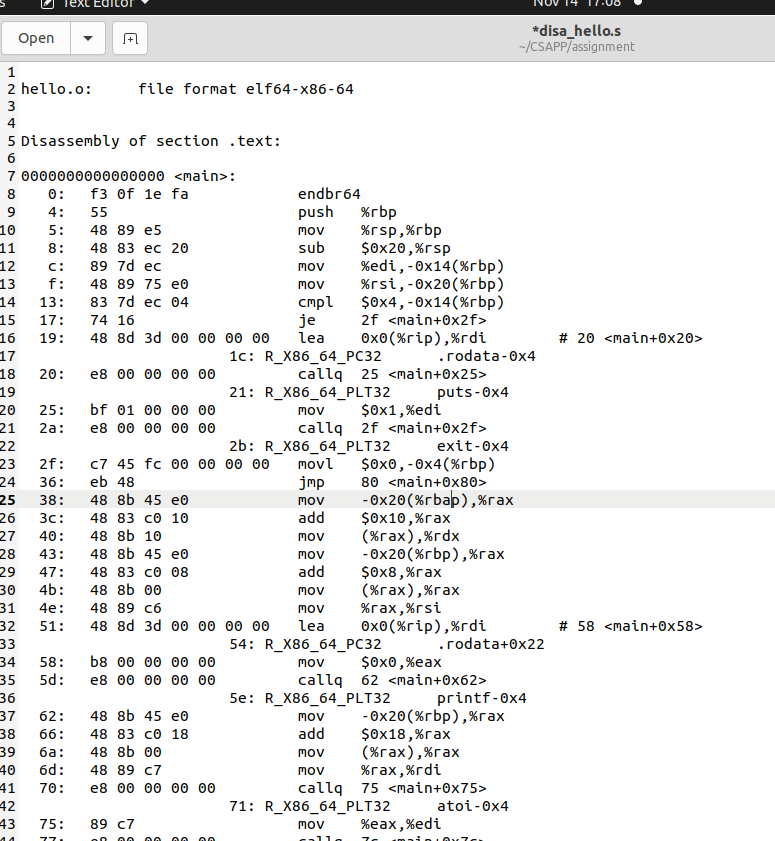


图20 disa\_hello.s文件内容

不同点：

1. 跳转命令：hello.s的跳转目的是.L1，.L2这样的符号来实现的，而disa\_hello.s中，是直接跳转到指令的地址。而这个地址在<>中注明了关于整个函数首地址的偏移量，这个偏移量是根据PC值和距跳转目的的偏移量算出来的。
2. 函数调用：hello.s函数调用后是函数名，而disa\_hello.s中call后有函数名，也有关于这个函数地址的信息，如重定位条目的类型。

## 4.5 本章小结

从汇编代码变为机器代码后，这个程序就可以真正被计算机理解。我们也可以利用反汇编工具查看一些二进制机器程序的汇编代码从而去破解或翻译，虽然反汇编后的代码与原本的汇编代码有所不同。

# 第5章 链接

## 5.1 链接的概念与作用

**概念**：链接是将各种代码和数据片段收集并组合成为了一个单一文件的过程，这个文件可被加载（复制）到内存并执行。链接可以执行于编译时，也可以执行于加载时，分别对应静态链接和动态链接。

**作用**：链接在软件开发中扮演着重要角色，因为它使得分离编译成为可能。我们可以将软件进行模块化设计，然后模块化编程，这样分组工作高效。而且，需要修改或者调试时，只需要修改某个模块，然后简单地重新编译它，并重新链接，而不是全部重新编译链接。

## 5.2 在Ubuntu下链接的命令

链接命令：

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



图21 链接指令

## 5.3 可执行目标文件hello的格式

1. ELF头

命令：readelf -h hello

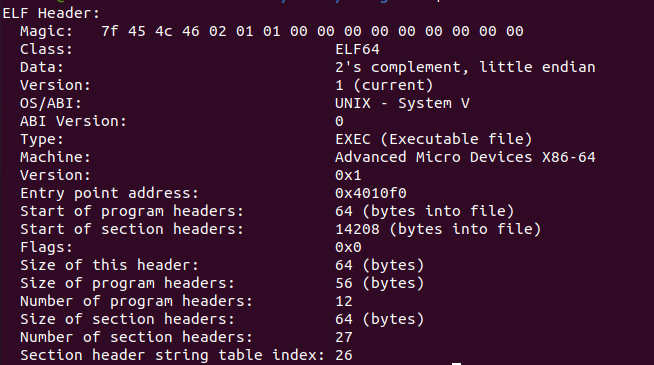


图22 ELF头

节头数量由14变为27。

1. 节头部表：

命令：readelf -S hello

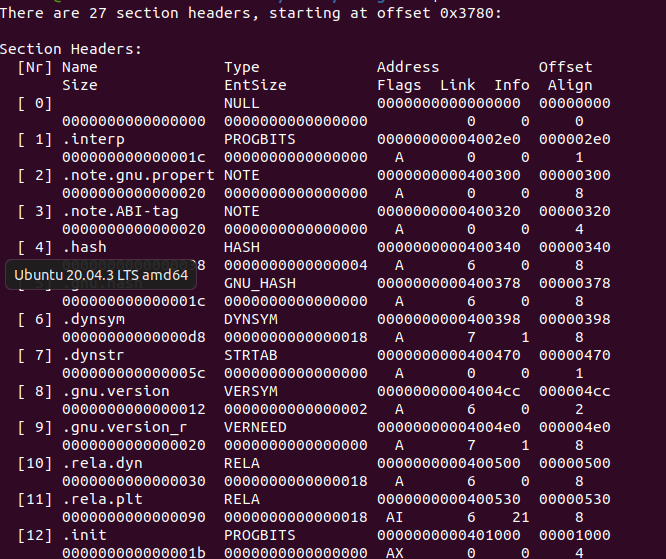


图23 节头部表-1

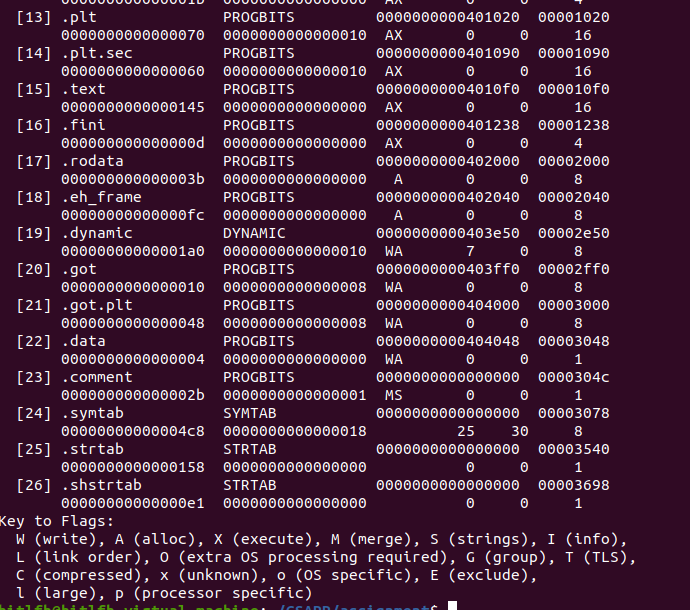


图24 节头部表-2

发现了一些动态链接所增加的段。

## 5.4 hello的虚拟地址空间

hello的虚拟地址从0x401000开始。

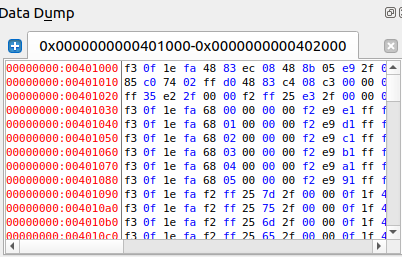


图25 EDB查看虚拟地址空间

比较readelf的段信息和edb的虚拟内存空间：

查看偏移0x3048的.data段

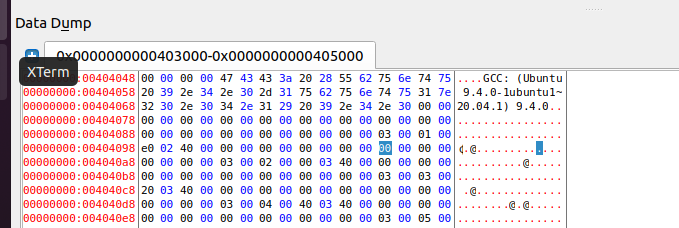


图26 EDB查看.data

## 5.5 链接的重定位过程分析

objdump -d -r hello > disa\_hello\_2.s分析与disa\_hello.s的不同

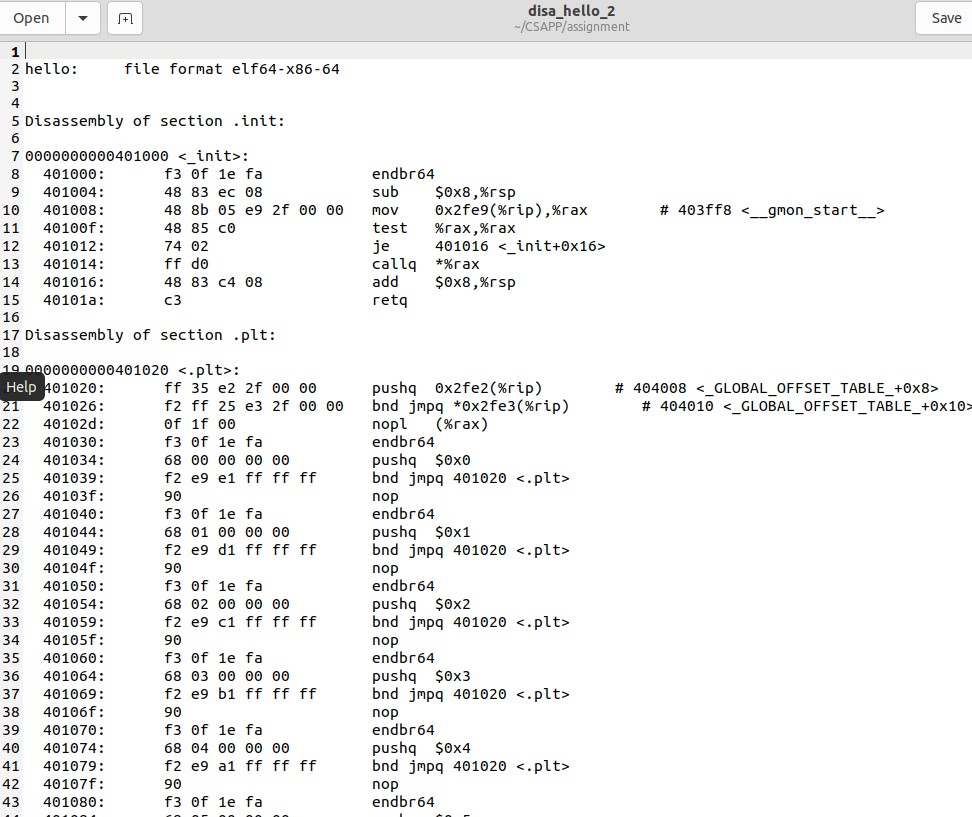


图27 查看disa\_hello\_2.s文件内容

1. 新增加的函数：

新增加了很多函数和它的实现汇编代码，如我们源代码中使用的printf，atoi，exit等。还有\_init初始化函数。

1. 新增加的节（Section）：

新增加了.init节，.plt节，.plt.sec。

1. 调用函数：

调用函数不再有重定位类型和偏移量，而是直接为函数的绝对地址和它的函数名。

链接：链接器（ld）将各个目标文件（各种.o文件）组装在一起，文件中的各个函数段按照一定规则累积在一起。从.o提供的重定位条目将函数调用和控制流跳转的地址填写为最终的地址。

## 5.6 hello的执行流程

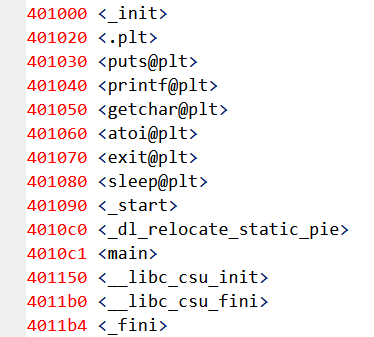


图28 hello的执行流程

## 5.7 Hello的动态链接分析

在进行动态链接前，首先进行静态链接，生成部分链接的可执行目标文件 hello。此时共享库中的代码和数据没有被合并到 hello 中。只有在加载 hello 时，动态链接器才对共享目标文件中的相应模块内的代码和数据进行重定位，加载共享库，生成完全链接的可执行目标文件。

例如：查看.plt段

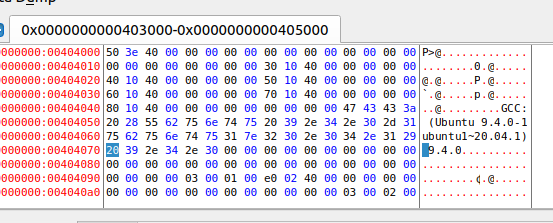


图29 EDB查看.plt段-1

运行后：

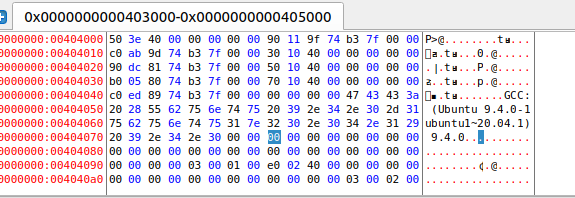


图30 EDB查看.plt段-2

## 5.8 本章小结

本章主要介绍了链接器如何将hello.o可重定向文件与动态库函数链接起来，分析了可重定位文件与可执行文件ELF的差异，并分析了重定位的过程。

# 第6章 hello进程管理

## 6.1 进程的概念与作用

**概念**：进程的经典定义是一个执行中程序的实例，系统的每个程序都运行在某个进程的上下文。上下文是由程序正确运行所需的状态组成的，这个状态包括存放在内存里的程序的代码和数据，它的栈，通用目的寄存器的内容，程序计数器，环境变量以及打开文件描述符的集合。进程是计算机科学中最深刻，最成功的概念。

**作用**：通过进程，我们会得到一种假象，好像我们的程序是当前唯一运行的程序，我们的程序独占处理器和内存，我们程序的代码和数据好像是系统内存中唯一的对象。而这些假象就是通过进程来实现的。

## 6.2 简述壳Shell-bash的作用与处理流程

Shell 是一种交互型的应用级程序，用户能够通过 Shell 与操作系统内核进行交互。

bash，全称为Bourne-Again Shell。它是一个为GNU项目编写的Unix shell。bash脚本功能非常强大，尤其是在处理自动循环或大的任务方面可节省大量的时间。bash是许多Linux平台的内定Shell。

处理流程：

1. Shell读取用户输入的命令。
2. Shell判断是否为shell内置命令，如果不是则认为是一个可执行文件。、
3. Shell构建参数和环境变量。
4. Shell通过fork创建子进程，再通过execve函数加载可执行文件。
5. Shell回收创建的子进程，循环。

## 6.3 Hello的fork进程创建过程

我们在Shell上输入./hello,这个不是一个内置的Shell命令，所以Shell会认为hello是一个可执行目标文件，通过条用某个驻留在存储器中被称为加载器的操作系统代码来运行它。当Shell运行一个程序时，父进程通过fork函数生成这个程序的进程。这个子进程几乎与父进程相同，子进程得到与父进程相同的虚拟地址空间（独立）的一个副本，包括代码，数据段，堆，共享库以及用户栈，并且和父进程共享文件。它们之间最大的不同是PID不同。

## Hello的execve过程

execve函数加载并运行可执行目标文件，且带参数列表argv和环境变量列表envp。在exevce加载了后，它调用启动代码，启动代码会设置栈，并将控制传递给新程序的主函数。

1. 删除已存在的用户区域
2. 映射私有区：为 hello 的代码、数据、.bss 和栈区域创建新的区域结构，所有这些区域都是私有的、写时才复制。
3. 映射共享区：比如 hello 程序与共享库 [libc.so](https://link.zhihu.com/?target=http%3A//libc.so/) 链接。
4. 设置 PC：exceve() 做的最后一件事就是设置当前进程的上下文中的程序计数器，使之指向代码区域的入口点

## 6.5 Hello的进程执行

**逻辑控制流**：一系列程序计数器PC的值的序列叫做逻辑控制流，进程是轮流使用处理器的，在同一个处理器核心中，每个进程执行它的流的一部分后被抢占（暂时挂起），然后轮到其他进程。

**时间片**：一个进程执行它的控制流的一部分的每一时间段叫做时间片。

**用户模式和内核模式**：处理器通常使用一个寄存器提供两种模式的区分，该寄存器描述了进程当前享有的特权，当没有设置模式位时，进程就处于用户模式中，用户模式的进程不允许执行特权指令，也不允许直接引用地址空间中内核区内的代码和数据；设置模式位时，进程处于内核模式，该进程可以执行指令集中的任何命令，并且可以访问系统中的任何内存位置。

**上下文信息**：上下文就是内核重新启动一个被抢占的进程所需要的状态，它由通用寄存器、浮点寄存器、程序计数器、用户栈、状态寄存器、内核栈和各种内核数据结构等对象的值构成。

**上下文切换**：上下切换是由内核中调度器完成的，当内核调度新的进程运行后，它就会抢占当前进程，并进行1）保存以前进程的上下文2）恢复新恢复进程被保存的上下文，3）将控制传递给这个新恢复的进程 ，来完成上下文切换。

**Hello**：当hello运行到sleep函数时，是在用户模式，然后调用sleep函数，进入内核模式，内核处理休眠请求，并开始计时。当时间到了后，内核接受到恢复hello运行的信号，再进行调度，切换上下文，返回用户模式。

## 6.6 hello的异常与信号处理

1. 不停乱按，包括回车：

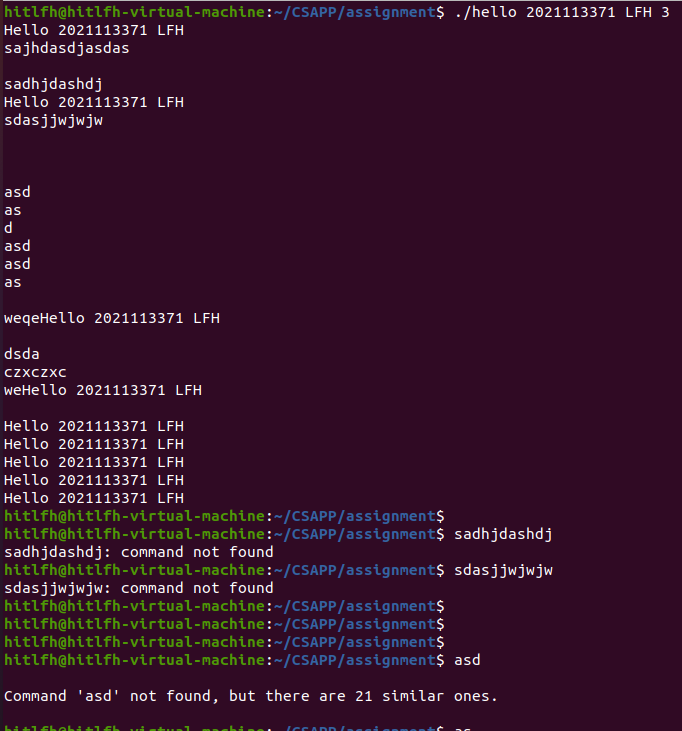


图31 乱按的现象

Shell会将回车前输出的字符串当作命令。

1. Ctrl + C

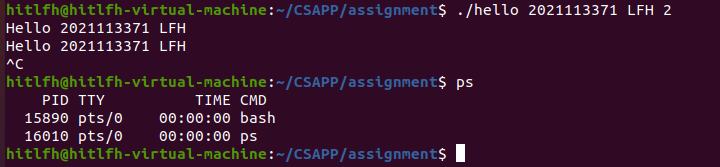


图32 Ctrl + C的现象

会立即终止进程，通过ps命令发现hello进程被回收。

1. Ctrl + Z

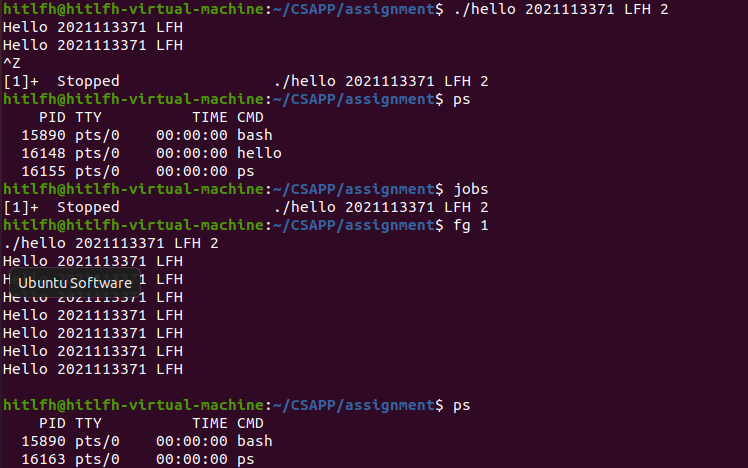


图33 Ctrl + Z的现象

会在后台停止，fg放到前台运行时，会输出剩下的7个字符串。

## 6.7本章小结

本章了解了hello进程的执行过程。在hello运行过程中，内核对其调度，异常处理程序为其将处理各种异常。每种信号都有不同的处理机制，对不同的shell命令，hello也有不同的响应结果。

# 结论

最终，我们在所学习的范围内，回顾了这段hello的一生。

1. 编写hello.c源代码程序。
2. hello.c预处理为hello.i。
3. hello.i编译为汇编语言hello.s。
4. hello.s汇编翻译为机器语言生成可重定位文件hello.o。
5. hello.o文件与动态库链接，生成可执行文件hello。
6. 在Shell中，输出./hello，创建进程。
7. 在子进程中，加载hello的上下文，映射内存。
8. 在hello运行的过程中，接收信号，进行信号处理。
9. 程序运行完成后，回收进程，释放内存。

会想起一开始的C语言学习，有时候报一些不是语法的错很不理解，有时候数组越界了却提示超时错误也很不理解。我们只是把从源代码到结果作为一个黑盒，而现在对于一个程序如何执行有了一个大体的认识，明白了之前的错误，但是还是有不清楚的地方，只学习到了第八章，还需要在以后的学习中继续努力。

# 附件

1. hello.c：源代码文件。
2. hello.i：预处理后的文本文件。
3. hello.s：编译后的汇编文件。
4. hello.o：汇编后的可重定位文件。
5. hello：链接后的可执行文件。
6. disa\_hello.s：反汇编hello.o的汇编代码。
7. disa\_hello\_2.s：反汇编hello的汇编代码。

# 参考文献

[1] Randal E.Bryant . 深入理解计算机系统[M]. 北京：机械出版社，2016.7

[2] https://blog.csdn.net/qq\_36314864/article/details/121250743

[3] https://blog.csdn.net/hzp020/article/details/83765267

[4] https://www.runoob.com/w3cnote/gcc-parameter-detail.html

[5] https://www.cnblogs.com/skywang12345/archive/2013/05/30/3106570.html