

**计算机系统**

**大作业**

题 目 程序人生-Hello’s P2P

专 业 计算机专业

学　　 号 1190200501

班　　 级 1903002

学 生 林燕燕

指 导 教 师 郑贵滨

**计算机科学与技术学院**

**2021年6月**

**摘 要**

本论文通过hello.c程序，对在CSAPP课程中所学知识进行整理，在Ubuntu虚拟机Linux系统下进行所有操作，运用Linux系统的工具，分析hello程序的一生。

**关键词：**hello，计算机系统，linux，程序人生；

**（摘要0分，缺失-1分，根据内容精彩称都酌情加分0-1分）**

**目 录**

[第 1 章 概述 - 4 -](#_Toc75014076)

[1.1 Hello简介 - 4 -](#_Toc75014077)

[1.2 环境与工具 - 4 -](#_Toc75014078)

[1.3 中间结果 - 4 -](#_Toc75014079)

[1.4 本章小结 - 5 -](#_Toc75014080)

[第 2 章 预处理 - 6 -](#_Toc75014081)

[2.1 预处理的概念与作用 - 6 -](#_Toc75014082)

[2.2在Ubuntu下预处理的命令 - 6 -](#_Toc75014083)

[2.3 Hello的预处理结果解析 - 6 -](#_Toc75014084)

[2.4 本章小结 - 7 -](#_Toc75014085)

[第 3 章 编译 - 8 -](#_Toc75014086)

[3.1 编译的概念与作用 - 8 -](#_Toc75014087)

[3.2 在Ubuntu下编译的命令 - 8 -](#_Toc75014088)

[3.3 Hello的编译结果解析 - 8 -](#_Toc75014089)

[3.3.1 数据 - 8 -](#_Toc75014090)

[3.3.2 操作 - 9 -](#_Toc75014091)

[3.4 本章小结 - 12 -](#_Toc75014092)

[第 4 章 汇编 - 13 -](#_Toc75014093)

[4.1 汇编的概念与作用 - 13 -](#_Toc75014094)

[4.2 在Ubuntu下汇编的命令 - 13 -](#_Toc75014095)

[4.3 可重定位目标elf格式 - 13 -](#_Toc75014096)

[4.4 Hello.o的结果解析 - 15 -](#_Toc75014097)

[4.5 本章小结 - 16 -](#_Toc75014098)

[第 5 章 链接 - 17 -](#_Toc75014099)

[5.1 链接的概念与作用 - 17 -](#_Toc75014100)

[5.2 在Ubuntu下链接的命令 - 17 -](#_Toc75014101)

[5.3 可执行目标文件hello的格式 - 17 -](#_Toc75014102)

[5.4 hello的虚拟地址空间 - 18 -](#_Toc75014103)

[5.5 链接的重定位过程分析 - 19 -](#_Toc75014104)

[5.6 hello的执行流程 - 21 -](#_Toc75014105)

[5.7 Hello的动态链接分析 - 21 -](#_Toc75014106)

[5.8 本章小结 - 22 -](#_Toc75014107)

[第 6 章 hello进程管理 - 23 -](#_Toc75014108)

[6.1 进程的概念与作用 - 23 -](#_Toc75014109)

[6.2 简述壳Shell-bash的作用与处理流程 - 23 -](#_Toc75014110)

[6.3 Hello的fork进程创建过程 - 24 -](#_Toc75014111)

[6.4 Hello的execve过程 - 24 -](#_Toc75014112)

[6.5 Hello的进程执行 - 24 -](#_Toc75014113)

[6.6 hello的异常与信号处理 - 24 -](#_Toc75014114)

[6.7本章小结 - 27 -](#_Toc75014115)

[第 7 章 hello的存储管理 - 28 -](#_Toc75014116)

[7.1 hello的存储器地址空间 - 28 -](#_Toc75014117)

[7.2 Intel逻辑地址到线性地址的变换-段式管理 - 28 -](#_Toc75014118)

[7.3 Hello的线性地址到物理地址的变换-页式管理 - 28 -](#_Toc75014119)

[7.4 TLB与四级页表支持下的VA到PA的变换 - 29 -](#_Toc75014120)

[7.5 三级Cache支持下的物理内存访问 - 29 -](#_Toc75014121)

[7.6 hello进程fork时的内存映射 - 29 -](#_Toc75014122)

[7.7 hello进程execve时的内存映射 - 29 -](#_Toc75014123)

[7.8 缺页故障与缺页中断处理 - 30 -](#_Toc75014124)

[7.9动态存储分配管理 - 30 -](#_Toc75014125)

[7.10本章小结 - 30 -](#_Toc75014126)

[第 8 章 hello的IO管理 - 31 -](#_Toc75014127)

[8.1 Linux的IO设备管理方法 - 31 -](#_Toc75014128)

[8.2 简述Unix IO接口及其函数 - 31 -](#_Toc75014129)

[8.3 printf的实现分析 - 31 -](#_Toc75014130)

[8.4 getchar的实现分析 - 33 -](#_Toc75014131)

[8.5本章小结 - 33 -](#_Toc75014132)

[结论 - 34 -](#_Toc75014133)

[附件 - 35 -](#_Toc75014134)

[参考文献 - 36 -](#_Toc75014135)

# 概述

## 1.1 Hello简介

Hello的P2P过程：

在VSCode等编辑器中键入Hello代码得到hello.c程序；对程序进行预处理、编译、汇编、链接生成可执行目标程序；在shell中启动后，shell调用fork创建子进程；hello从Program转换为Process。

Hello的020过程：

然后shell为hello的execve映射虚拟内存，进入程序入口后开始载入物理内存；进入main函数执行目标代码，CPU为hello分配时间片以执行逻辑控制流；程序运行结束后，shell父进程回收hello进程，释放占用的内存，删除数据结构。

## 1.2 环境与工具

硬件环境

X64 CPU；1.6GHz；8G RAM；256G SSD Disk；1T HDD Disk

软件环境

Windows10 64位；Vmware 14pro；Ubuntu 20.04.2 LTS 64位

开发工具

Visual Studio Code 64位；vim/gpedit+gcc/as/ld/edb/readelf；

## 1.3 中间结果

|  |  |
| --- | --- |
| 文件的作用 | 文件名 |
| 预处理后的文件 | hello.i |
| 编译之后的汇编文件 | hello.s |
| 汇编之后的可重定位目标文件 | hello.o |
| 链接之后的可执行目标文件 | hello |
| hello.o 的 ELF格式 | elf.txt |
| hello.o 的反汇编代码 | hello\_o\_disass.s |
| hello的ELF 格式 | elf\_hello.txt |
| hello 的反汇编代码 | hello\_disass.s |

## 1.4 本章小结

本章简要介绍了hello的P2P，020过程，并列出了大作业的软硬件环境及开发工具，还列出了操作过程中产生的中间结果。

**（第1章0.5分）**

# 预处理

## 2.1 预处理的概念与作用

预处理又称预编译，（对于c/c++来说）预处理指的是在程序编译之前，根据以字符#开头的命令（即头文件/define等），修改原始的c程序。

预处理的主要作用如下：

●将源文件中以”include”格式包含的文件复制到编译的源文件中。

●用实际值替换用“#define”定义的字符串。

●根据“#if”后面的条件决定需要编译的代码。

## 2.2在Ubuntu下预处理的命令

命令：gcc -E hello.c -o hello.i



图 2.1 预处理命令

## 2.3 Hello的预处理结果解析

程序扩展为3065行，hello源程序出现在3047行，#include <stdio.h> #include <stdlib.h> #include <unistd.h>三个头文件消失，替代的是一大段代码，描述的是运行库在计算机中的位置，方便下一步翻译成汇编语言。

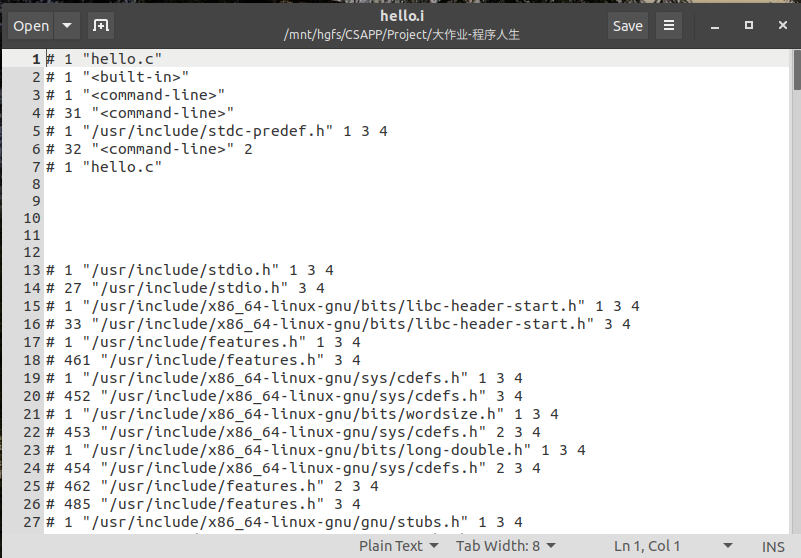


图 2.2 hello.i库文件



图 2.3 源代码

## 2.4 本章小结

本章主要介绍了预处理的概念和应用功能，同时将hello.c文件进行预处理生成hello.i文本文件，并对生成的hello.i文件进行分析，详细了解了预处理的内涵。

**（第2章0.5分）**

# 编译

## 3.1 编译的概念与作用

编译是利用编译程序从源语言编写的源程序产生目标程序的过程，是用编译程序产生目标程序的动作，把高级语言变成计算机可以识别的2进制语言。

作用：

把用高级程序设计语言书写的源程序，翻译成等价的计算机汇编或机器语言书写的目标程序，编译程序以高级程序设计语言书写的源程序作为输入，而以汇编或机器语言表示的目标程序作为输出。

注意：这儿的编译是指从 .i 到 .s 即预处理后的文件到生成汇编语言程序

## 3.2 在Ubuntu下编译的命令

命令：gcc -S hello.i -o hello.s



图 3.1 编译命令

## 3.3 Hello的编译结果解析

### 3.3.1 数据

1. 常量

字符串：如下两个函数中的字符串被存储在 .rodata节中



图 3.2 printf函数

存储如下：

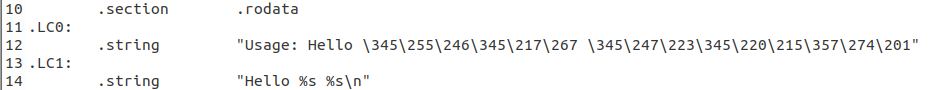


图 3.3 字符串常量存储

1. 变量
2. 全局变量

初始化的全局变量储存在.data节，sleepsecs全局变量被存放在.data节



图 3.4 sleepsecs存储位置

1. 局部变量

局部变量存储在寄存器或栈中。程序中有局部变量

在汇编代码中如下：



图 3.5 局部变量i存储位置

i被存储在栈中，-4(%rbp)的位置。

### 3.3.2 操作

1. 算术操作

在循环操作中，使用 i++ 自增操作，每次循环结束后对 i 加1，对栈上存储i 的位置加1；



图 3.6 i的自增操作

1. 关系操作
2. 程序第16行中判断argc是否等于3，源代码为：



图 3.7 源代码

汇编代码为：



图 3.8 汇编代码

1. 程序第21行中判断 i 是否小于10，源代码为：



图 3.9 源代码

汇编代码为，汇编优化为 i <= 9：



图 3.10 汇编代码

1. 控制转移

在使用比较关系操作进行判断后，程序将按判断结果经如下代码跳转至L2或L4，进入if语句或继续进入循环



图 3.11 if语句跳转



图 3.12 for语句跳转

1. 数组/指针/结构操作

主函数main的参数中有指针数组char \*argv[]，argv[0]指向输入程序的路径和名称，argv[1]和argv[2]分别表示两个字符串。

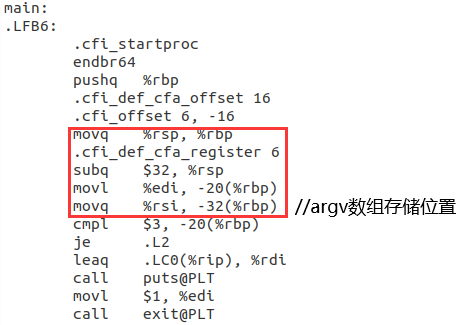


图 3.13 main函数参数存储

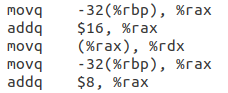


图 3.14 argv数组实现汇编代码

1. 函数操作
2. main函数

参数传递：传入参数argc,argv[]，分别用寄存器%edi和%rsi存储

函数调用：被系统启动函数调用

函数返回：设置%eax为0，并返回



图 3.15 main函数返回

1. printf函数
2. call puts@PLT

参数传递：传入字符串参数首地址

函数调用：if判断满足条件后被调用



图 3.16 call puts汇编代码

1. call printf@PLT

参数传递：传入 argv[1]和argc[2]的地址

函数调用：for循环中被调用



图 3.17 call printf汇编代码

1. exit函数

参数传递：传入参数1

函数调用：if判断满足条件后被调用



图 3.18 exit函数汇编代码

1. sleep函数

参数传递：传入参数全局变量sleepsecs

函数调用：for循环中被调用



图 3.19 sleep函数汇编代码

1. getchar函数

函数调用：在for循环结束后被调用



图 3.20 getchar函数汇编代码

## 3.4 本章小结

本章主要介绍了编译的概念以及过程，对hello进行编译转换为汇编代码。分析编译得到的文件，介绍汇编代码如何实现变量、常量、传递参数以及分支和循环。编译程序在确认所有的指令都符合语法规则之后，将其翻译成等价的中间代码表示或汇编代码表示。

**（第3章2分）**

# 汇编

## 4.1 汇编的概念与作用

汇编是通过汇编器，把汇编语言翻译成机器语言的过程。

作用：通过汇编过程把汇编代码转换成计算机能直接识别的机器代码，把指令打包成可重定位目标程序的格式，并将结果保存在.o 目标文件中。

注意：这儿的汇编是指从 .s 到 .o 即编译后的文件到生成机器语言二进制程序的过程。

## 4.2 在Ubuntu下汇编的命令

命令：as hello.s -o hello.o



图 4.1 汇编命令

## 4.3 可重定位目标elf格式

1. 命令：readelf -a hello.o > ./elf.txt



图 4.2 生成elf文件命令

1. ELF文件头

包含了系统信息，编码方式，ELF头大小，节的大小和数量等等一系列信息。

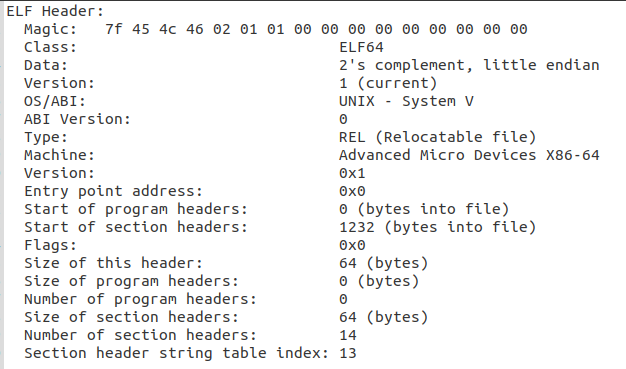


图 4.3 ELF文件头内容

1. 节头部表

描述了.o文件中出现的各个节的类型、位置、所占空间大小等信息。

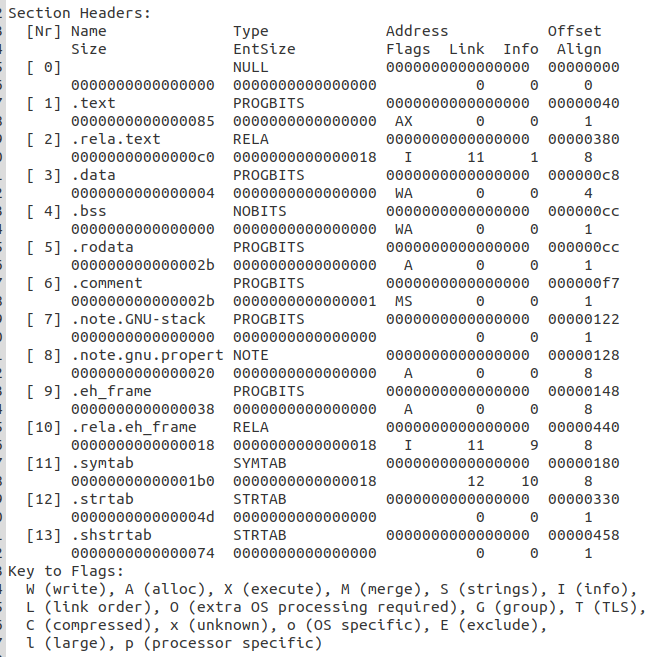


图 4.4 节头部表内容

1. 重定位节

表述了各个段引用的外部符号等，在链接时，需要通过重定位节对这些位置的地址进行修改。链接器会通过重定位条目的类型判断该使用什么养的方法计算正确的地址值，通过偏移量等信息计算出正确的地址。

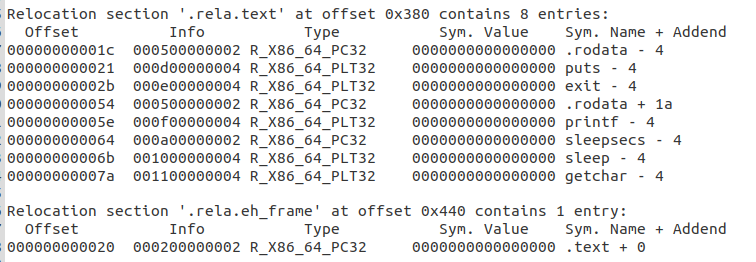


图 4.5 重定位节内容

1. 符号表

存放在程序中定义和引用的函数和全局变量的信息。

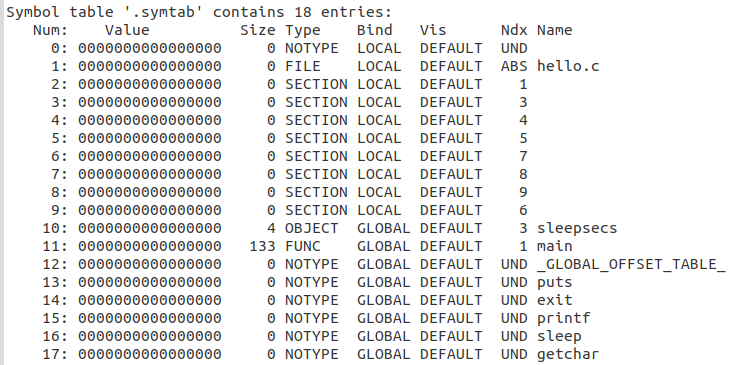


图 4.6 符号表内容

## 4.4 Hello.o的结果解析

命令：objdump -d -r hello.o > hello\_o\_disass.s

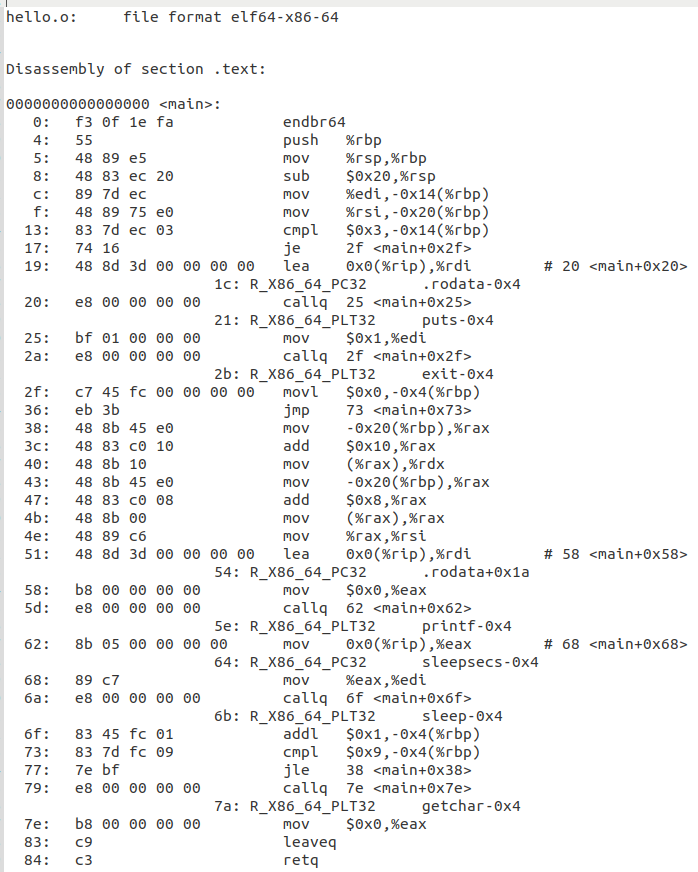


图 4.7 反汇编文件内容

分析hello.o的反汇编，并请与第3章的 hello.s进行对照分析：

1. 代码左边多了机器码；
2. hello.s中的操作数为十进制，hello.o反汇编代码中的操作数为十六进制；
3. call跳转指令，在hello.s文件中，直接加上跳转函数名，在反汇编文件中，加上了跳转的相对偏移地址，函数在链接之后才能确定执行的地址，因此在.rela.text节中为其添加了重定位条目。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。

## 4.5 本章小结

了解汇编的概念和作用，汇编得到.o文件，然后分析了可重定位目标elf格式，然后使用objdump进行反汇编并与.s文件进行比较，更加深入地理解机器语言与汇编语言的关系。

**（第4章1分）**

# 链接

## 5.1 链接的概念与作用

链接是将各种不同文件的代码和数据部分收集并组合成一个单一文件的过程。

作用：

将源程序与为了节省空间而未编入的常用函数文件进行合并，生成可以正常工作的可执行文件。令分离编译成为可能，节省了大量的工作空间。

注意：这儿的链接是指从 hello.o 到hello生成过程。

## 5.2 在Ubuntu下链接的命令

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

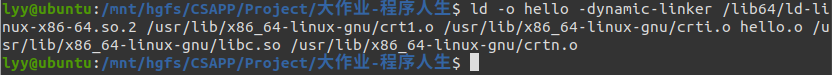


图 5.1 链接命令

## 5.3 可执行目标文件hello的格式

1. 命令：readelf -a hello > elf\_hello.txt
2. ELF文件头

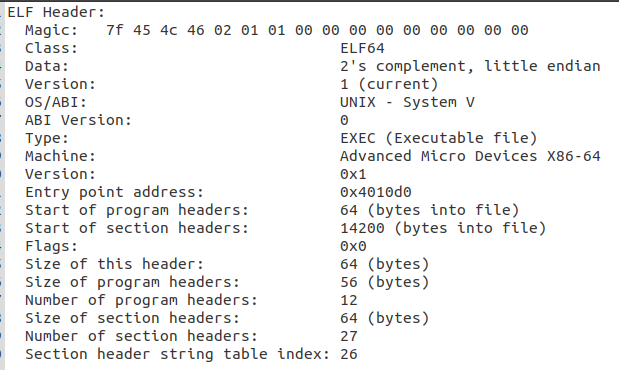
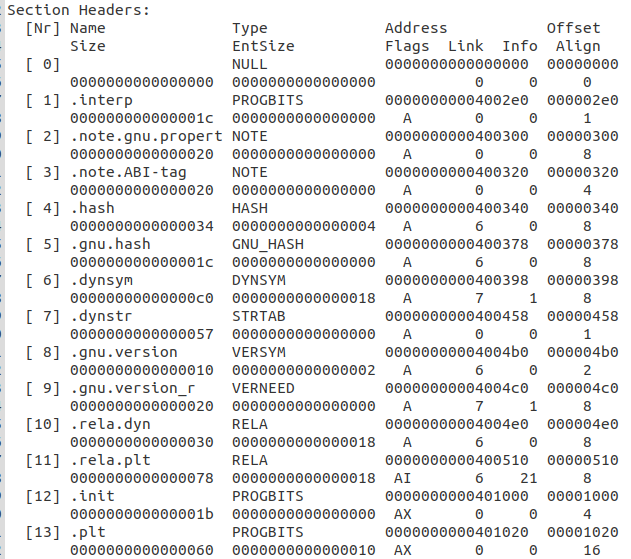


图 5.2 ELF文件头

1. 节头部表



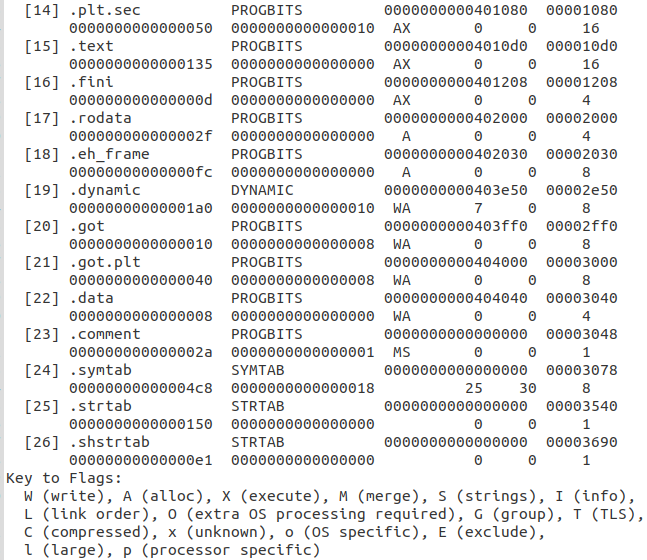


图 5.3 节头部表

1. 符号表

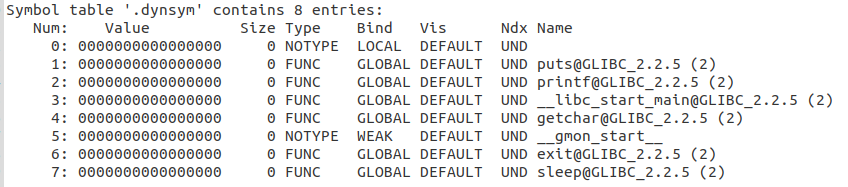


图 5.4 符号表

## 5.4 hello的虚拟地址空间

使用edb加载hello，查看本进程的虚拟地址空间各段信息，并与5.3对照分析说明。

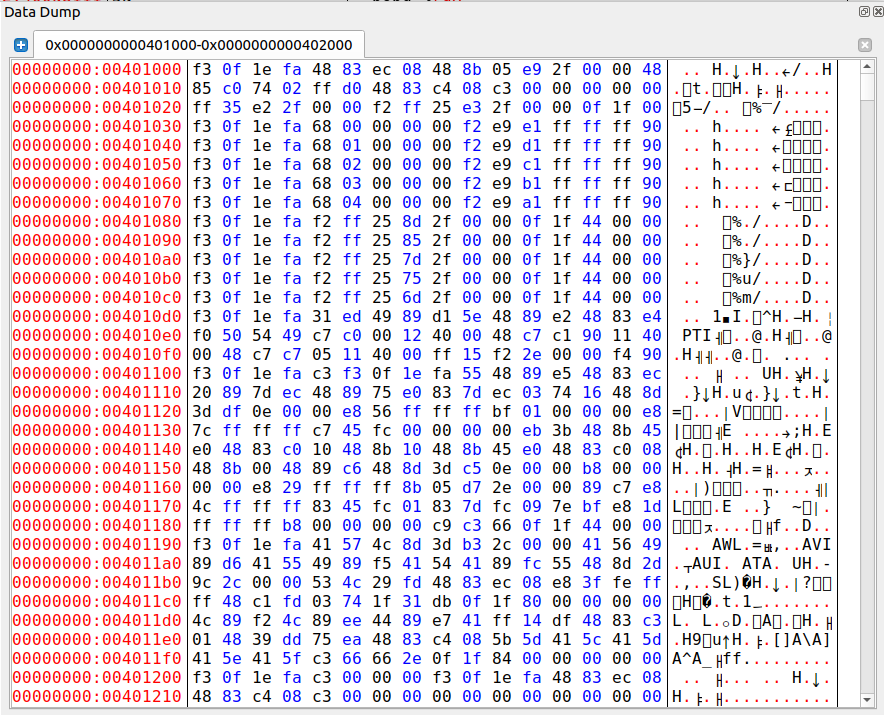


图 5.5 虚拟地址空间信息

查看 ELF 格式文件中的 Program Headers，它告诉链接器运行时加载的内容，并提供动态链接的信息。每一个表项提供了各段在虚拟地址空间和物理地址空间的各方面的信息。

程序包含：

PHDR 保存程序头表；

INTERP 指定在程序已经从可执行文件映射到内存之后，必须调用的解释器；

LOAD 表示一个需要从二进制文件映射到虚拟地址空间的段。其中保存了常量数据、程序的目标代码等；

DYNAMIC 保存了由动态链接器使用的信息；

NOTE 保存辅助信息；

GNU\_STACK，权限标志，用于标志栈是否是可执行；

GNU\_RELRO，指定在重定位结束之后哪些内存区域是需要设置只读。

## 5.5 链接的重定位过程分析

命令：objdump -d -r hello > hello\_disass.s

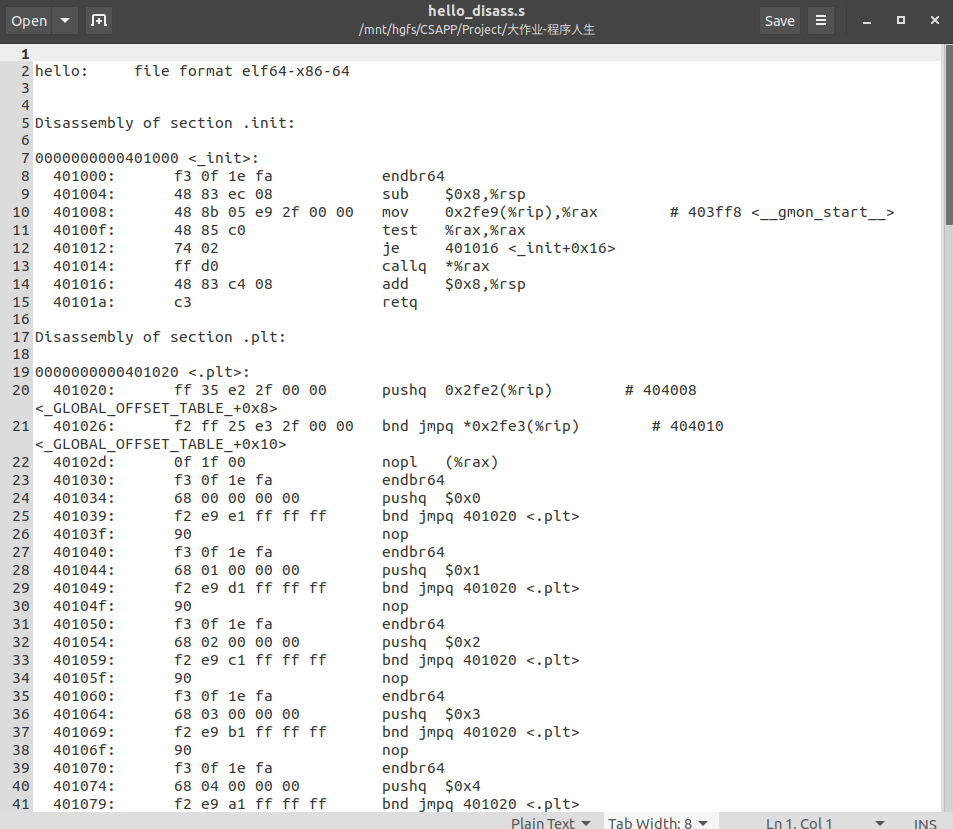


图 5.6 hello反汇编代码

* 分析hello与hello.o的不同：

1. 在hello.o中，地址为相对偏移地址；在hello中，地址为可以由CPU直接访问的虚拟内存地址；
2. hello的反汇编文件比hello.o的反汇编文件多了\_init，.plt，puts@plt，printf@plt，getchar@plt，exit@plt，sleep@plt，\_start，\_dl\_relocate\_static\_pie，\_\_libc\_csu\_init，\_\_libc\_csu\_fini，\_fini等节和需要用到的库函数；
3. hello.o将lea后的操作数置为0，并添加重定位条目。

* 链接的过程：

链接器将各个目标文件组装在一起，文件中的各个函数段按照一定规则累积在一起。

* 结合hello.o的重定位项目，分析hello中对其怎么重定位的：

1. 重定位节和符号定义，连接器将所有相同类型的节合并为同一类型的新的节，然后将运行内存地址赋给这个新的节和输入模块定义的每个节以及输入模块定义的每个符号；
2. 重定位节中的符号引用，连接器修改代码节和数据节中对每个符号的引用，使之指向正确的运行地址。

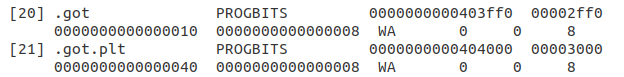
## 5.6 hello的执行流程

使用edb执行hello，说明从加载hello到\_start，到call main,以及程序终止的所有过程。请列出其调用与跳转的各个子程序名或程序地址。

|  |  |
| --- | --- |
| 子程序名 | 程序地址 |
| \_init | 0x 401000 |
| \_start | 0x4010d0 |
| \_\_libc\_csu\_init | 0x401190 |
| \_init | 0x 401000 |
| main | 0x401105 |
| puts@plt | 0x401080 |
| exit@plt | 0x4010b0 |
| \_fini | 0x401208 |

## 5.7 Hello的动态链接分析

在elf文件中：





进入edb查看：

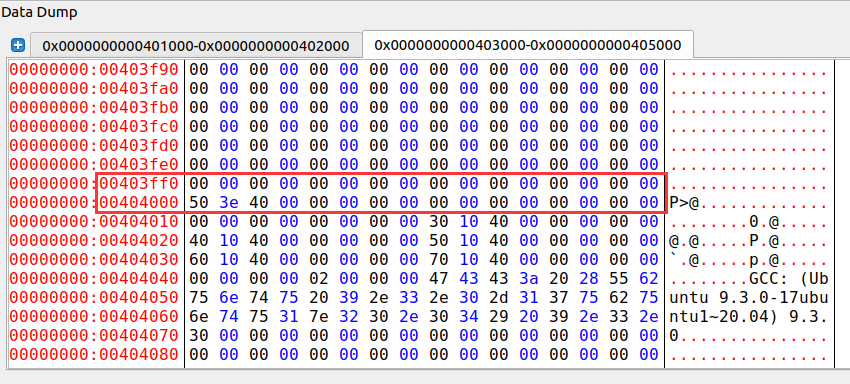


图 5.7 edb执行init之前的地址

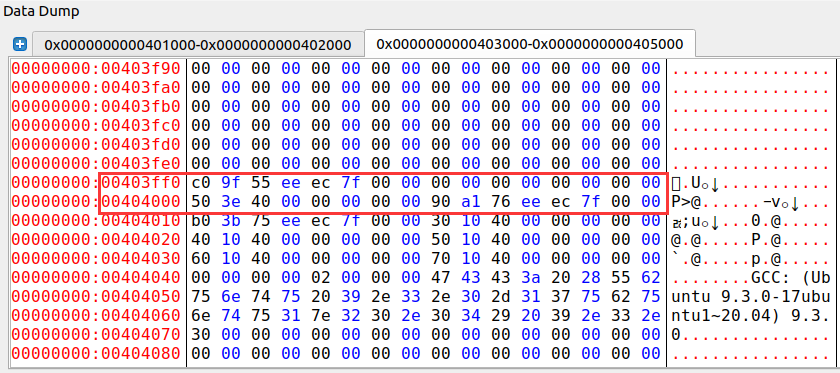


图 5.8 edb执行init之后的地址

一开始地址的字节都为0，调用\_init函数之后GOT内容产生变化，指向正确的内存地址，下一次调用跳转时可以跳转到正确位置。

## 5.8 本章小结

本章了解了链接的作用，对hello.o进行链接得到hello文件，并对hello进行分析，与hello.o的反汇编代码进行比较，更好的掌握重定位过程。

**（第5章1分）**

# hello进程管理

## 6.1 进程的概念与作用

概念：进程是一个执行中的程序的实例。

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。

## 6.2 简述壳Shell-bash的作用与处理流程

作用：

shell是一个命令解释器，它解释用户输入的命令并把它们送到内核，用于用户和系统的交互。

流程：

1. Shell首先从命令行中找出特殊字符（元字符），在将元字符翻译成间隔符号。元字符将命令行划分成小块tokens。Shell中的元字符如下：SPACE , TAB , NEWLINE , & , ; , ( , ) ,< , > , |
2. 程序块tokens被处理，检查看他们是否是shell中所引用到的关键字。
3. 当程序块tokens被确定以后，shell根据aliases文件中的列表来检查命令的第一个单词。如果这个单词出现在aliases表中，执行替换操作并且处理过程回到第一步重新分割程序块tokens。
4. Shell对~符号进行替换。
5. Shell对所有前面带有$符号的变量进行替换。
6. Shell将命令行中的内嵌命令表达式替换成命令；他们一般都采用$(command)标记法。
7. Shell计算采用$(expression)标记的算术表达式。
8. Shell将命令字符串重新划分为新的块tokens。这次划分的依据是栏位分割符号，称为IFS。缺省的IFS变量包含有：SPACE , TAB 和换行符号。
9. Shell执行通配符\* ? [ ]的替换。
10. shell把所有从处理的结果中用到的注释删除，並且按照下面的顺序实行命令的检查：A. 内建的命令B. shell函数（由用户自己定义的）C. 可执行的脚本文件（需要寻找文件和PATH路径）
11. 在执行前的最后一步是初始化所有的输入输出重定向。
12. 最后，执行命令。

## 6.3 Hello的fork进程创建过程

shell判断出不是内置命令后，加载可执行文件hello，通过fork创建子进程，子进程得到一份与父进程用户级虚拟地址空间相同的副本，还获得与父进程打开的文件描述符相同的副本，子进程与父进程的PID不同。fork被调用一次，但返回两次，父进程中返回子进程的PID，子进程返回0。

## 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文件，execve才会返回到调用程序。所以与fork一次调用返回两次不同，execve调用一次并从不返回。

## 6.5 Hello的进程执行

内核调用hello的进程开始进行，输出Hello与之前输入的内容，然后执行sleep函数，这个函数是系统调用，它请求让调用进程休眠。内核转而执行其他进程，这时就会发生一个上下文转换。2s后，又会发生一次进程转换，恢复hello进程的上下文，继续执行hello进程，重复这个过程。

循环结束后，后面执行到getchar函数，这时读取数据需要很长的时间，所以将会发生上下文切换转而执行其他进程，当数据被读取到缓存区后，会发生中断，使内核发生上下文切换，重新执行hello进程。

## 6.6 hello的异常与信号处理

hello执行过程中会出现4类异常：

|  |  |  |  |
| --- | --- | --- | --- |
| 类别 | 原因 | 异步/同步 | 返回行为 |
| 中断 | 来自I/O设备的信号 | 异步 | 总是返回到下一条指令 |
| 陷阱 | 有意的异常 | 同步 | 总是返回到下一条指令 |
| 故障 | 潜在可恢复的错误 | 同步 | 可能返回到当前指令 |
| 终止 | 不可恢复的错误 | 同步 | 不会返回 |

信号处理方式：

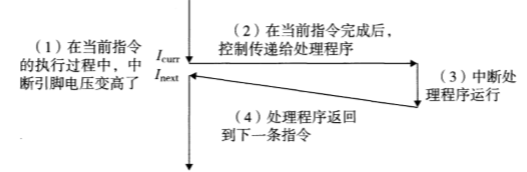


图 6.1 中断处理

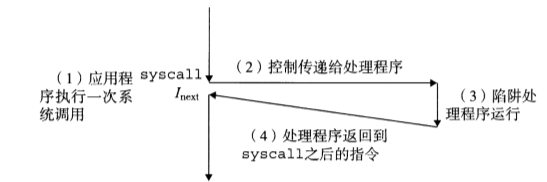


图 6.2 陷阱处理

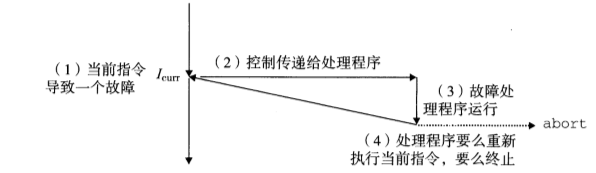


图 6.3 故障处理

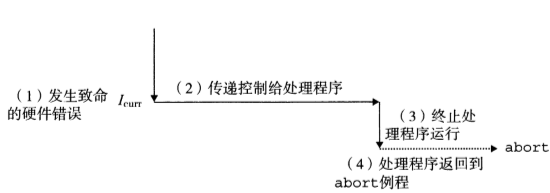


图 6.4 终止处理

1. 正常运行 回车

回车不影响运行，只是插入了空行。



图 6.5 回车运行结果

1. Ctrl-Z

进程收到SIGTSTP信号，暂时挂起，输入ps命令符查看PID发现hello的PID为12188，hello进程还没有被关闭，输入fg命令恢复后台进程。

输入kill命令后，进程被终止，输入ps命令进程hello不存在。

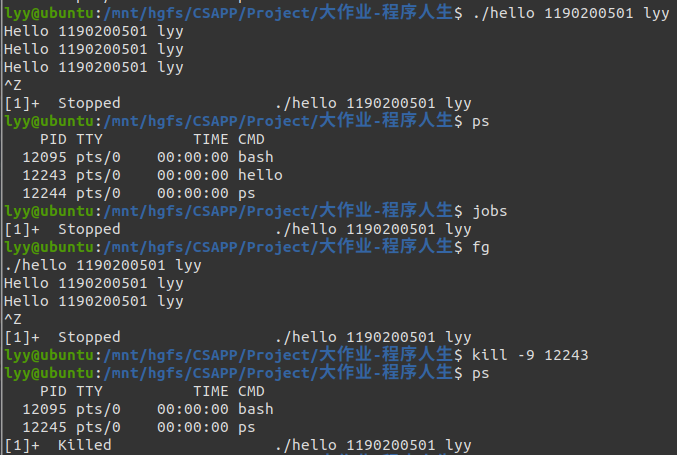


图 6.6 Ctrl-Z运行结果

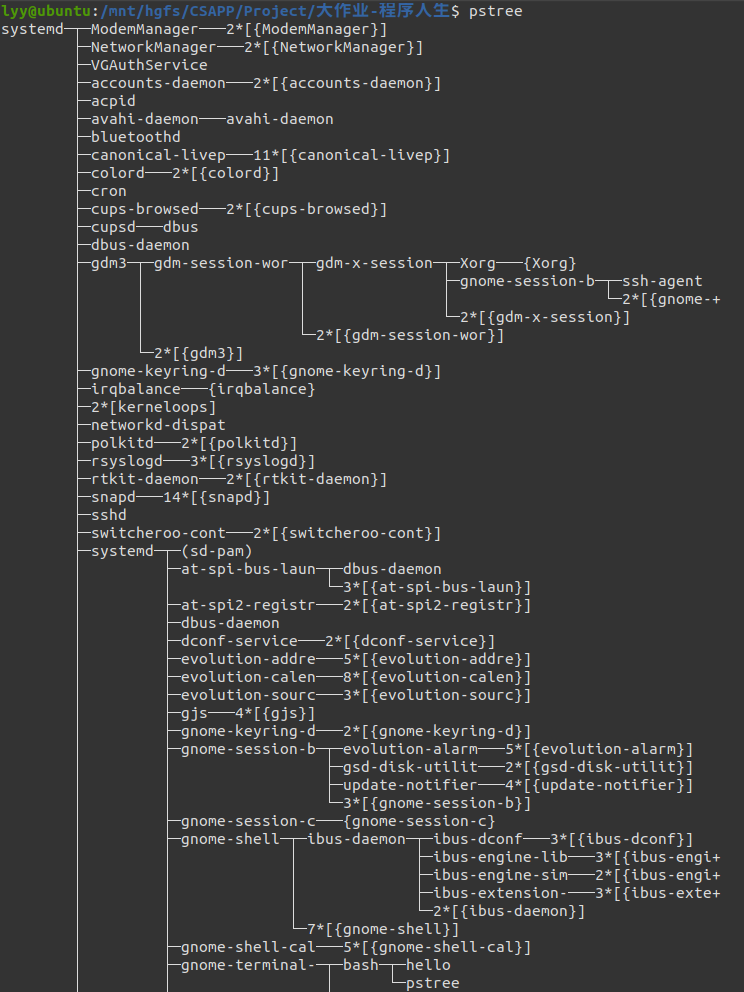


图 6.7 进程树

1. Ctrl-C

进程收到SIGINT信号，进程结束，输入ps命令没有hello。

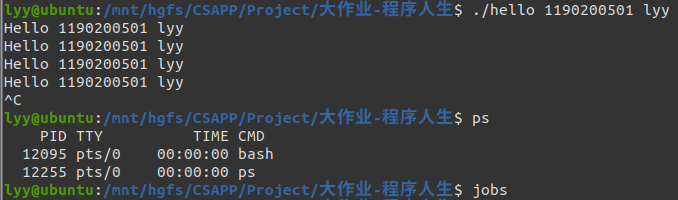


图 6.8 Ctrl-C运行结果

1. 中途乱按

输入的字符被保存在缓冲区，被认为是命令，在程序结束后执行。

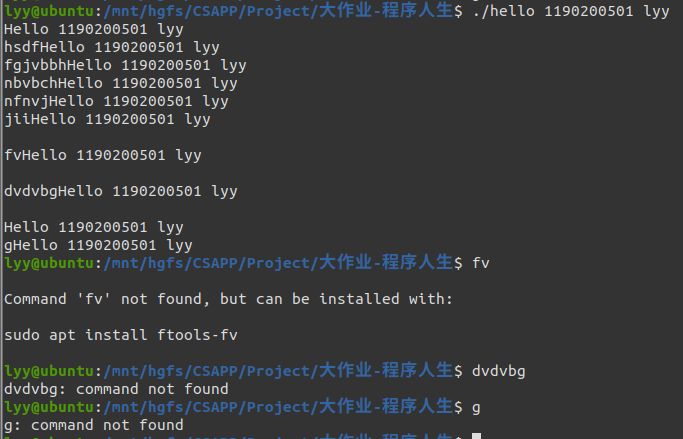


图 6.9 中途乱按运行结果

## 6.7本章小结

本章了解了hello进程的概念和shell的作用，分析了hello进程的执行过程以及fork和execve的作用，了解信号处理和hello进程如何在内核和前端中反复跳跃运行的。

**（第6章1分）**

# hello的存储管理

## 7.1 hello的存储器地址空间

* 逻辑地址：是由程序产生的与段相关的偏移地址部分。
* 线性地址：逻辑地址经过段机制后转化为线性地址，为(描述符:偏移量)的组合形式。分页机制中线性地址作为输入。
* 虚拟地址：有时也把逻辑地址叫做虚拟地址。
* 物理地址：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。每个字节都有唯一的物理地址。CPU通过地址总线的寻址，找到真实的物理内存对应地址。 CPU对内存的访问是通过连接着CPU和北桥芯片的前端总线来完成的。在前端总线上传输的内存地址都是物理内存地址。

## 7.2 Intel逻辑地址到线性地址的变换-段式管理

一个逻辑地址由两部份组成，段标识符: 段内偏移量。可以通过段标识符的前13位，直接在段描述符表中找到一个具体的段描述符，Base字段表示包含段的首字节的线性地址，也就是一个段的开始位置的线性地址。

1. 先检查段选择符中的TI字段，以决定段描述符保存在哪一个描述符表中；
2. 由于一个段描述符是8字节长，因此她在GDT或LDT内的相对地址是由段选择符的最高13位的值乘以8得到；
3. Base + offset，得到要转换的线性地址。

## 7.3 Hello的线性地址到物理地址的变换-页式管理

真正的物理地址需要经过线性地址一系列的映射才会到真正的物理地址处。一般是通过三级或者四级映射，即10-10-12或者2-9-9-12。

页表是一个存储索引号的表，每个元素是4B，对于10-10-12而言，第一个10bit用就可以表示就可以表示全，第二个也是如此，这样一来，一个一级页表是4 \* B 大小，也就是4KB的大小，一级页表都可以索引一个二级页表，一个二级页表也是4KB大小，二级页表大小是 \* 4KB ，占了4MB。

通过每一个页表可以确定下一个表的基地址，而线性地址中的每一个部分则是对这个地址所在的表的一个查询工作。

页式管理是一种内存空间存储管理的技术，页式管理分为静态页式管理和动态页式管理。将各进程的虚拟空间划分成若干个长度相等的页(page)，页式管理把内存空间按页的大小划分成片或者页面（page frame），然后把页式虚拟地址与内存地址建立一一对应页表，并用相应的硬件地址变换机构，来解决离散地址变换问题。页式管理采用请求调页或预调页技术实现了内外存存储器的统一管理。

## 7.4 TLB与四级页表支持下的VA到PA的变换

先将VA中的VPN分成三段，根据TLBT和TLBI，在TLB中寻找对应的PPN，如果没有找到，即为缺页，就需要到页表中去找。

接着将VPN分成更多段（4段），CR3是对应的L1PT的物理地址，然后一步步递进往下寻址，越往下一层每个条目对应的区域越小，寻址越细致，在经过4层寻址之后找到相应的PPN然后和VPO拼接起来得到PA。

## 7.5 三级Cache支持下的物理内存访问

得到物理地址PA之后，根据cache大小组数的要求，将PA拆分成CT（标记）、CI（索引）、CO（偏移量），用CI位进行索引，如果匹配成功且valid值为1，则为命中，根据偏移量在L1cache中取数；如果未命中就去二级和三级cache中重复以上步骤，命中后返回结果。

## 7.6 hello进程fork时的内存映射

mm\_struct（内存描述符）：描述了一个进程的整个虚拟内存空间。

vm\_area\_struct（区域结构描述符）：描述了进程的虚拟内存空间的一个区间。

在用fork创建虚拟内存的时候，要经历以下步骤：

1. 创建当前进程的mm\_struct,vm\_area\_struct和页表的原样副本。
2. 两个进程的每个页面都标记为只读页面。
3. 两个进程的每个vm\_area\_struct都标记为私有，这样就只能在写入时复制。

## 7.7 hello进程execve时的内存映射

exceve函数加载和执行程序Hello，需要以下几个步骤：

1.删除已存在的用户区域。

2.创建新的私有区域（.malloc,.data,.bss,.text）。

3.创建新的共享区域（libc.so.data,libc.so.text）。

4.设置程序计数器PC，指向代码的入口点。

## 7.8 缺页故障与缺页中断处理

先判断这个缺页的虚拟地址是否合法，那么遍历所有的合法区域结构，如果这个虚拟地址对所有的区域结构都无法匹配，那么就返回一个段错误（segment fault）。接着查看这个地址的权限，判断一下进程是否有读写改这个地址的权限。若都不是，就是正常缺页，则选择一个页面牺牲然后换入新的页面并更新到页表。

## 7.9动态存储分配管理

基本方法：维护一个虚拟内存区域“堆”，分配器将堆视为一组不同大小的块的集合来维护，每个块要么是已分配的，要么是空闲的，需要时选择一个合适的内存块进行分配。

1. 记录空闲块，可以选择隐式空闲链表，显示空闲链表，分离的空闲链表和按块大小排序建立平衡树
2. 放置策略，可以选择首次适配，下一次适配，最佳适配
3. 合并策略，可以选择立即合并，延迟合并
4. 需要考虑分割空闲块的时机，对内部碎片的忍耐阈值.

## 7.10本章小结

本章介绍了hello 的存储器地址空间、 intel 的段式管理、 hello 的页式管理，在指定环境下介绍了 VA 到 PA 的变换，以及进程fork和execve内存映射的内容，还有缺页问题和动态存储分配管理的问题。

**（第7章 2分）**

# hello的IO管理

## 8.1 Linux的IO设备管理方法

设备的模型化：文件

文件的类型：

1. 普通文件(regular file)：包含任意数据的文件。
2. 目录(directory)(文件夹)：包含一组链接的文件，每个链接都将一个文件名映射到一个文件。
3. 套接字(socket)：用来与另一个进程进行跨网络通信的文件
4. 命名通道
5. 符号链接
6. 字符和块设备

设备管理：unix io接口

1. 打开和关闭文件
2. 读取和写入文件
3. 改变当前文件的位置

## 8.2 简述Unix IO接口及其函数

* 打开文件：open()函数

打开一个已经存在的文件，若文件不存在则创建一个新的文件。

* 关闭文件：close()函数

通知内核关闭这个文件，内核会释放文件打开时创建的数据结构，将描述符恢复到描述符池中。

* 读取文件：read()函数

从当前文件位置复制字节到内存位置，如果返回值<0则说明出现错误。

* 写入文件：write()函数

从内存复制字节到当前文件位置，如果返回值<0则说明出现错误。

* 改变文件位置：lseek()函数

文件开始位置为文件偏移量，应用程序通过seek操作，可设置文件的当前位置为k。

## 8.3 printf的实现分析

printf函数：

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;

}

所引用的vsprintf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);

}

vsprintf函数的作用是将所有的参数内容格式化之后存入buf，然后返回格式化数组的长度。

write函数是将buf中的i个元素写到终端的函数。

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.

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）

## 8.4 getchar的实现分析

getchar调用了一个read函数，将整个缓冲区都读到了buf里面，然后返回缓冲区的长度。如果buf长度为0，getchar才会调用read函数，否则是直接将保存的buf中的最前面的元素返回。

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

本章介绍了 Linux 的 I/O 设备的基本概念和管理机制，以及Unix I/O 接口及其函数，并且分析了printf 函数和 getchar 函数的工作过程。

**（第8章1分）**

# 结论

用计算机系统的语言，逐条总结hello所经历的过程。

你对计算机系统的设计与实现的深切感悟，你的创新理念，如新的设计与实现方法。

hello经历过程：

* 1. 预处理：将hello.c调用的所有外部的库拓展到hello.i文件中；
  2. 编译：将hello.i编译得到汇编代码文件hello.s；
  3. 汇编：将hello.s汇编成为二进制可重定位目标文件hello.o；
  4. 链接：将hello.o与可重定位目标文件和动态链接库链接成为可执行目标程序hello；
  5. 运行：shell中运行hello；
  6. 创建子进程：shell进程调用fork函数创建子进程；
  7. 运行程序：shell调用execve函数，execve调用启动加载器，映射虚拟内存，进入程序入口后程序开始载入物理内存，然后进入 main函数；
  8. 执行指令：CPU为其分配时间片，在一个时间片中，hello享有CPU资源，顺序执行自己的控制逻辑流；
  9. 访问内存：MMU将程序中使用的虚拟内存地址通过页表映射成物理地址；
  10. 动态申请内存：printf会调用malloc向动态内存分配器申请堆中的内存；
  11. 信号：如果运行途中键入Ctrl-C、Ctrl-Z则调用shell的信号处理函数分别停止、挂起；
  12. 结束：shell父进程回收子进程，内核删除为这个进程创建的所有数据结构。

感想：

要成为优秀的程序员或工程师，我们应该更加了解程序的底层实现，了解程序是如何从一个.c文件到可执行文件以及文件如何在计算机中运行的，只有了解学习了这些，我们才能更好的优化我们的程序，成为更好的工程师。

**（结论0分，缺失 -1分，根据内容酌情加分）**

# 附件

|  |  |
| --- | --- |
| 文件的作用 | 文件名 |
| 预处理后的文件 | hello.i |
| 编译之后的汇编文件 | hello.s |
| 汇编之后的可重定位目标文件 | hello.o |
| 链接之后的可执行目标文件 | hello |
| hello.o 的 ELF格式 | elf.txt |
| hello.o 的反汇编代码 | hello\_o\_disass.s |
| hello的ELF 格式 | elf\_hello.txt |
| hello 的反汇编代码 | hello\_disass.s |

**（附件0分，缺失 -1分）**

# 参考文献

[1] 林来兴. 空间控制技术[M]. 北京：中国宇航出版社，1992：25-42.

[2] 辛希孟. 信息技术与信息服务国际研讨会论文集：A集[C]. 北京：中国科学出版社，1999.

[3] 赵耀东. 新时代的工业工程师[M/OL]. 台北：天下文化出版社，1998 [1998-09-26]. http://www.ie.nthu.edu.tw/info/ie.newie.htm（Big5）.

[4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨：哈尔滨工业大学，1992：8-13.

[5] KANAMORI H. Shaking Without Quaking[J]. Science，1998，279（5359）：2063-2064.

[6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science，1998，281：331-332[1998-09-23]. http://www.sciencemag.org/cgi/ collection/anatmorp.

**（参考文献0分，缺失 -1分）**