

**实验报告**



**题目： Linux系统及其相关软件环境**

**班 级： 2021211321**

**学 号： 2021212171**

**姓 名： 杨晨**

**学 院： 计算机学院**

**2022年 10 月 16 日**

1. 实验目的
2. 熟悉 linux 操作的基本操作；
3. 掌握 gcc 编译方法；
4. 掌握 gdb 的调试工具使用；
5. 掌握 objdump 反汇编工具使用；
6. 熟悉理解反汇编程序（对照源程序与 objdump 生成的汇编程序）。

二、实验环境（5分）

简述使用的工具

1. MobaXterm远程登陆工具（服务器：10.120.11.12）
2. Linux
3. Gcc编译器
4. GDB调试工具
5. Objdump命令反汇编

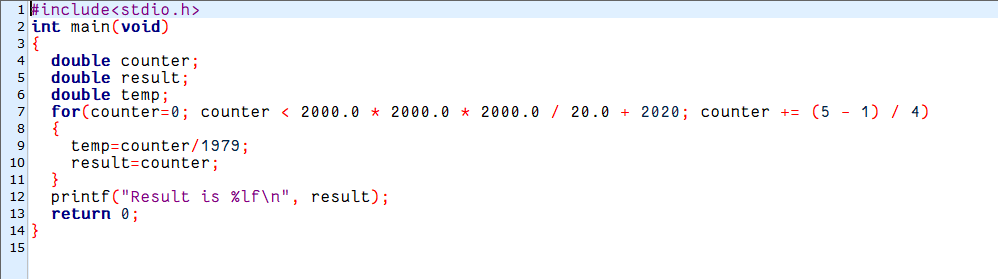
三、实验概况

四、实验步骤（60-80分）

操作步骤+运行截图

**实验内容一：**

首先新建一个文件，命名为optimize.c，将代码逐行敲入，保存





然后选择不同的参数进行编译，生成的可执行文件命名为optimize

选择**-o参数**进行编译（gcc optimize.c -o optimize）

程序执行时间（time ./optimize）

文本

描述已自动生成

选择**-O参数**进行编译（gcc -O optimize.c -o optimize）

程序执行时间（time ./optimize）

文本

描述已自动生成

接下来进行**gdb调试**（gdb）

因为for语句在在第7行，所以在第7行设置断点（b 7）

运行（run）

文本

描述已自动生成

观察变量counter的值的变化（watch counter）

继续执行到下一个断点（continue）

继续执行到下一个断点（continue）

很明显可以观察到counter的值的变化

文本

描述已自动生成

对optimize进行反汇编（objdump -d optimize）



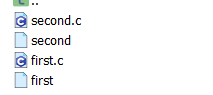
找到main函数的位置，即为main的汇编代码

图片包含 图形用户界面

描述已自动生成

**实验内容二：**

首先，新建2个文件，分别将两种算法对应的代码写入，并分别命名为first.c和second.c



不使用优化参数编译源代码（gcc first,c -o first）（gcc second.c -o second）

输出执行时间（time ./first）（time ./second）

文本

描述已自动生成

使用-O参数优化编译源代码（gcc -O first.c -o first）（gcc -O second.c -o second）

输出执行时间（time ./first）（time ./second）

文本

描述已自动生成

使用-O0参数（不优化）进行编译（gcc -O0 first.c -o first）（gcc -O0 second.c -o second）

输出执行时间（time ./first）（time ./second）

文本

描述已自动生成

使用-O1参数（同-O参数）进行编译（gcc -O1 first.c -o first）（gcc -O1 second.c -o second）

输出执行时间（time ./first）（time ./second）

文本

描述已自动生成

使用-O2参数进行编译（gcc -O2 first.c -o first）（gcc -O2 second.c -o second）

输出执行时间（time ./first）（time ./second）

文本

描述已自动生成

使用-O3参数进行编译（gcc -O3 first.c -o first）（gcc -O3 second.c -o second）

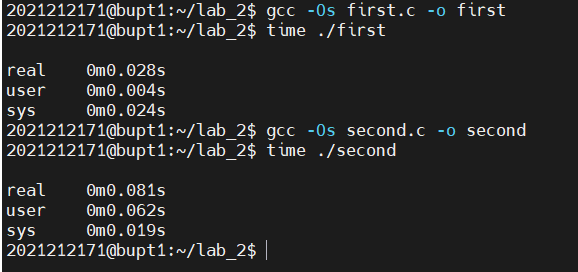
输出执行时间（time ./first）（time ./second）

文本

描述已自动生成

使用-Os参数进行编译（gcc -Os first.c -o first）（gcc -Os second.c -o second）

输出执行时间（time ./first）（time ./second）



实验内容三：

首先新建一个文件，完成a[i]+b[i]的算法，并命名为sum.c



采用gcc的-g选项进行编译（gcc -g sum.c -o sum）

对sum进行反汇编（objdump -d sum）



找到madd函数对应的汇编代码，截图如下

图形用户界面, 文本

描述已自动生成

进行**gdb**调试（gdb）

文本

描述已自动生成

练习**file**命令，通过file命令加在sum程序进行调试（file sum）

文本

描述已自动生成

在第8行设置一个断点（b 8）

运行到断点处（run）

练习**kill**命令，终止正在调试的程序（kill）

文本

描述已自动生成

练习**break**命令，在madd函数设置一个断点（break madd）

练习**break**命令，在第10行设置一个断点（break 10）

练习**info break**命令，打印当前所有断点（info break）

练习**delete**命令，删除第1个断点（delete 1）

检验，打印当前所有断点（info break）

文本

描述已自动生成

练习**clear**命令，删掉第10行的断点（clear 10）

检验，打印当前所有断点（info break）

图形用户界面

中度可信度描述已自动生成

第5行设置断点，练习**run**和**continue**命令（b 5）

运行，执行到断点处（run）

执行到下一个断点（continue）

文本

描述已自动生成

在第12行设置一个断点（b 12）

练习**nexti**命令，发现nexti 是单步一条机器指令，不进入函数（多次执行nexti）

文本

描述已自动生成

练习**stepi**命令，发现stepi 是单步一条机器指令，包括进入函数（多次stepi）

文本

描述已自动生成

练习**disassemble**命令，显示当前函数的汇编代码（disassmeble）

图形用户界面, 文本

描述已自动生成

练习**disassemble [Function]**指定要反汇编的函数（disassemble main）

图形用户界面, 文本

描述已自动生成

练习**list**命令，显示源程序代码的内容，包括各行代码所在的行号（list）

文本

描述已自动生成

练习**print**命令，输出c[0]的值（print c[0]）

文本

描述已自动生成

练习**x**命令，显示存储c[0]的地址（十六进制）

蓝色的标志

描述已自动生成

练习**info reg**命令 ,显示当前各寄存器内的值（info reg）

文本

描述已自动生成

练习**watch**命令，观察c[1]值的变化（watch c[1]）

执行（continue）

文本

描述已自动生成

练习**quit**命令，退出gdb调试（quit）

文本

描述已自动生成

在a[i]+b[i]处设置断点（b 5）

运行到a[i]+b[i]处（run）

查看这里的汇编代码（disas）

图形用户界面, 文本

描述已自动生成

a[i]+b[i]的对应汇编语句为



故 a[i]寄存器%ecx，b[i]在寄存器%edx中

根据学号，X=1，Y=1，所以对a[1]和b[1]分析

先执行到a[1]+b[1]处（continue）

连续进行单步调试，以确保a[1]+b[1]即将执行（stepi）

输出寄存器内的值（十进制）（print $ecx）（print $edx）

输出寄存器内的值（十六进制）（print /x $ecx）（print /x $edx）

执行a[1]+b[1]后（stepi）

输出寄存器内的值（十进制）（print $ecx）（print $edx）

输出寄存器内的值（十六进制）（print /x $ecx）（print /x $edx）

文本

描述已自动生成

实验内容四：

选取的高计算量类型算法为矩阵乘法，高内存需求类型算法为递归求解斐波那契数。

上述两个算法分别实现后，命名为matrix.c和fibonacci.c。

接下来选择不同的参数进行编译，并输出执行时间。

（gcc -参数 XX.c -o XX）

（time ./XX）

对比分析，发现O0是运行效率最差的，而O1、O2、O3随着数字的增大，程序的执行效率也越来越高。

对于Os参数，在矩阵乘法的算法里，它的执行时间介于O0和O1之间；而在求解斐波那契数的算法里，它的执行时间介于O1和O2之间。

屏幕上写着字

描述已自动生成

文本

描述已自动生成

五、实验分析（20分）

实验时的工作思路、设想、效果等综合分析

**实验一思路：**

先将源代码敲入，检查无误后，命名为optimize.c，然后根据课件上的方法，选择不同的优化参数进行编译，输出运行时间。

对比不同的优化参数，可以发现，使用-O参数进行编译，程序的运行效率显著提高。猜想-O参数会对于编译过程有一定的优化。

查阅资料，发现-o参数对于编译过程是不进行优化的，而-O参数会进行一定的优化，-O等同于-O1，会在编译时占用稍微多的时间和相当大的内存，减少代码生成尺寸、缩短执行时间。

接下来根据课件上的方法，在for循环处设置断点，进行调试，观察循环变量counter的值的变化

选择对optimize的所有代码进行反汇编，找到main的位置，即找到了main的汇编代码

**实验二思路：**

将两个代码分别敲入，为了加以区分，分别命名为first.c和second.c，然后选择不同的优化参数进行编译，打印执行时间。

**实验三思路：**

编写代码，完成a+b功能，命名为sum.c，根据要求，选择不优化选项进行编译。

对所有代码进行反汇编，找到madd的位置，即找到了madd的汇编代码（同实验一的思路）。

分别按照要求练习gdb的各种命令。

为了找到a[i]+b[i]对应的汇编指令，先输出madd函数的汇编代码，再采用设置断点，单步机器代码逐个调试的方法。由于是加法，优先观察带有“add”的汇编代码。发现有若干个“add”。

为了确定a[i]+b[i]的位置，选择将若干个带有add代码的内寄存器内的值，进行输出，然后进行watch观测，stepi单步调试，最后确定下来a[i]+b[i]的汇编代码。回忆汇编代码的知识，相互印证，确定无误。

根据学号，要分析a[1]+b[1]。仍然是先设置断点，单步机器代码逐步推进，执行到a[1]+b[1]处，此时还未完成相加的操作，输出寄存器内的值；再进行一次单步机器代码推进，此时已完成了相加的操作，输出寄存器内的值。

**实验四思路：**

分别选择高计算量的算法——矩阵乘法，高内存需求类型算法——递归求斐波那契数，将二者代码敲入，命名。

选择不同的优化参数进行编译，输出执行时间。

六、实验总结（5分）

总结心得（包括遇到的困难，自己一些不成功的设计和设想）

**遇到的困难：**

在进行实验一的输出执行时间时，time ./XX中间没有打空格，导致无法输出执行时间。更正命令后成功解决。

在进行实验一的寻找main的汇编代码时，选择对目标文件的反汇编结果输出到txt文件中，在反汇编同时显示源代码，发现由于源代码和汇编代码掺杂在一起，难以直观找到main的汇编代码。更换方法后成功解决。

**一些设想：**

O0~O3的优化程度依次增强，但会牺牲程序的可调试性。所以在第3个实验里选择无优化参数进行编译。

**收获：**

学会了使用Linux指令进行文件的创建，修改，删除等操作

熟悉了gcc编译的基本命令，能够完成基本的编译操作

熟悉了gcd调试的基本命令，能够完成基本的调试操作

熟悉了objdump反汇编命令，可以完成对某一块代码的反汇编

了解了会汇编语言的执行逻辑，寄存器之间的调用关系

七、诚信声明（不签扣10分）

需要填写如下声明，并在底部给出手写签名的电子版。

在完成本次实验过程中，我没有与任何同学进行交流。

此外，我还参考了以下资料：

1. [GCC 参数详解 | 菜鸟教程 (runoob.com)](https://www.runoob.com/w3cnote/gcc-parameter-detail.html)
2. [Optimize Options (Using the GNU Compiler Collection (GCC))](https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html)

在我提交的程序中，还在对应的位置以注释形式记录了具体的参考内容。

我独立完成了本次实验除以上方面之外的所有工作，包括分析、设计、编码、调试与测试。

我清楚地知道，从以上方面获得的信息在一定程度上降低了实验的难度，可能影响起评分。

我从未使用他人代码，不管是原封不动地复制，还是经过某些等价转换。

我未曾也不会向同一课程（包括此后各届）的同学复制或公开我这份程序的代码，我有义务妥善保管好它们。

我编写这个程序无意于破坏或妨碍任何计算机系统的正常运行。

我清楚地知道，以上情况均为本课程纪律所禁止，若违反，对应的实验成绩将按照0分计。

（签名）文本, 白板

描述已自动生成