

**实验报告**



**题目：Linux系统及其相关软件环境**

**班级：2021211321**

**学号：2021213586**

**姓名：郭栩源**

**学院：计算机学院**

**2022年10 月27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. SecureCRT或其他远程登陆工具（服务器：10.120.11.12）

SecureCRT是一款终端仿真程序，是Windows下登陆UNIX或者Linux服务器主机的软件。

1. Linux

Linux是一种免费使用和自由传播的类UNIX操作系统，用户可以自由修改其源代码，自由度很高。

1. Gcc编译器

Gcc原名为GNU C语言编译器（GNU C Compiler），是由GNU开发的编程语言编译器，现已被大多数类Unix操作系统（Linux，MacOS等）采纳为标准的编译器。

1. GDB调试工具

GDB是由GNU开发组织并发布的UNIX和类UNIX下的调试工具，能让用户在程序运行时观察程序的内部结构和内存使用情况。

1. Objdump命令反汇编

Objdump是以一种可阅读的格式让用户更多地了解二进制文件可能带有的附加信息的工具。

三、实验概况

简述实验内容和基本设想

实验内容一（15分）

在linux环境下，编辑课件中源程序（注意程序的完整性）（包含源程序的开发环境截图），采用gcc编译该程序（要求分别采用-o和-O参数，并比较两者性能，编译指令截图），采用gdb进行调试，让程序运行到for函数语句（调试截图），运用objdump工具生成汇编程序（给出main函数的汇编程序截图）

#include<stdio.h>

int main(void)

{

double counter;

double result;

double temp;

for(counter=0;counter<2000.0\*2000.0\*2000.0/20.0+2020;

counter+=(5-1)/4){

temp=counter/1979;

result=counter;

}

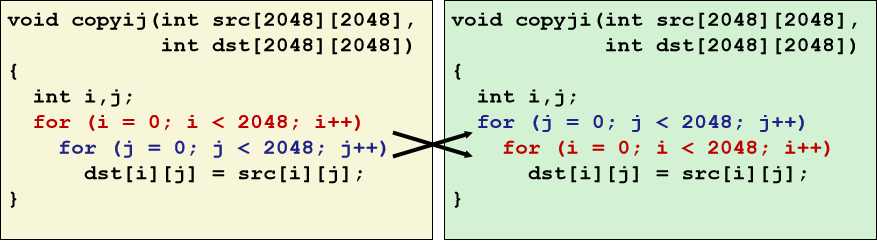
printf(Result is%lf\\n,result);

return 0；

}

实验内容二（15分）

在linux环境下，分别打印输出如下算法所需时间



分别设置不同优化参数，给出运行时间

实验内容三（30分）

现有int型数组a[i]=i-50,b[i]=i+y，其中y取自于学生本人学号2019211x\*y的个位。登录bupt1服务器，在linux环境下使用vi编辑器编写C语言源程序，完成数组a+b的功能，规定数组长度为100，函数名为madd（），数组a，b均定义在函数内，采用gcc编译该程序（不使用优化选项），

使用objdump工具生成汇编程序，找到madd函数的汇编程序，给出截图；

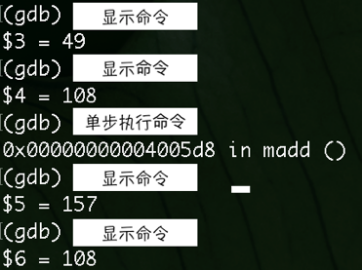
用gdb进行调试，练习如下gdb命令，给出截图；

gdb、file、kill、quit、break、delete、clear、info break、run、continue、nexti、stepi、disassemble、list、print、x、info reg、watch

找到a[i]+b[i]对应的汇编指令，指出a[i]和b[i]位于哪个寄存器中，给出截图；

使用单步指令及gdb相关命令，显示a[xy]+b[xy]对应的汇编指令执行前后操作数寄存器十进制和十六进制的值，其中x，y取自于学生本人学号2019211x\*y的百位和个位。

学号2019211999，a[99]+b[99]单步执行前后的参考截图如下（实际命令未显示出）：



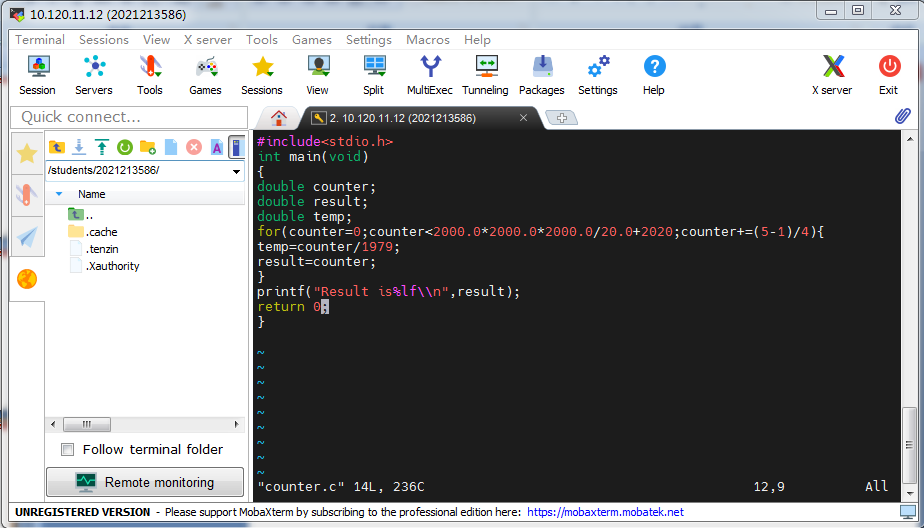
实验内容四（加分项，20分）

任选高复杂度算法（具体算法自选，类型分为高计算量类型和高内存需求类型2类算法），通过设置不同优化参数，分析算法的运行效率

四、实验步骤（60-80分）

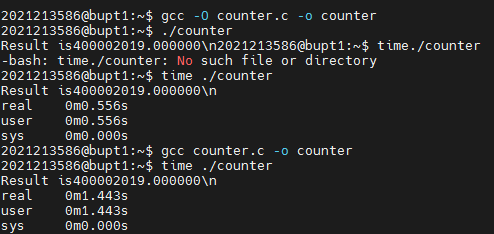
操作步骤+运行截图

**实验一实验步骤：**首先登录MobaXtrem，输入vi counter.c打开vi编辑器，同时新建源文件counter.c。按i键进入insert模式，并将上述代码复制到源文件中，如下图。

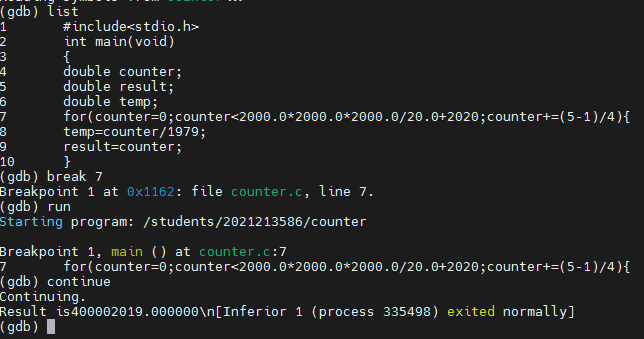


按esc进入command模式，输入：wq保存并退出vi编辑器。

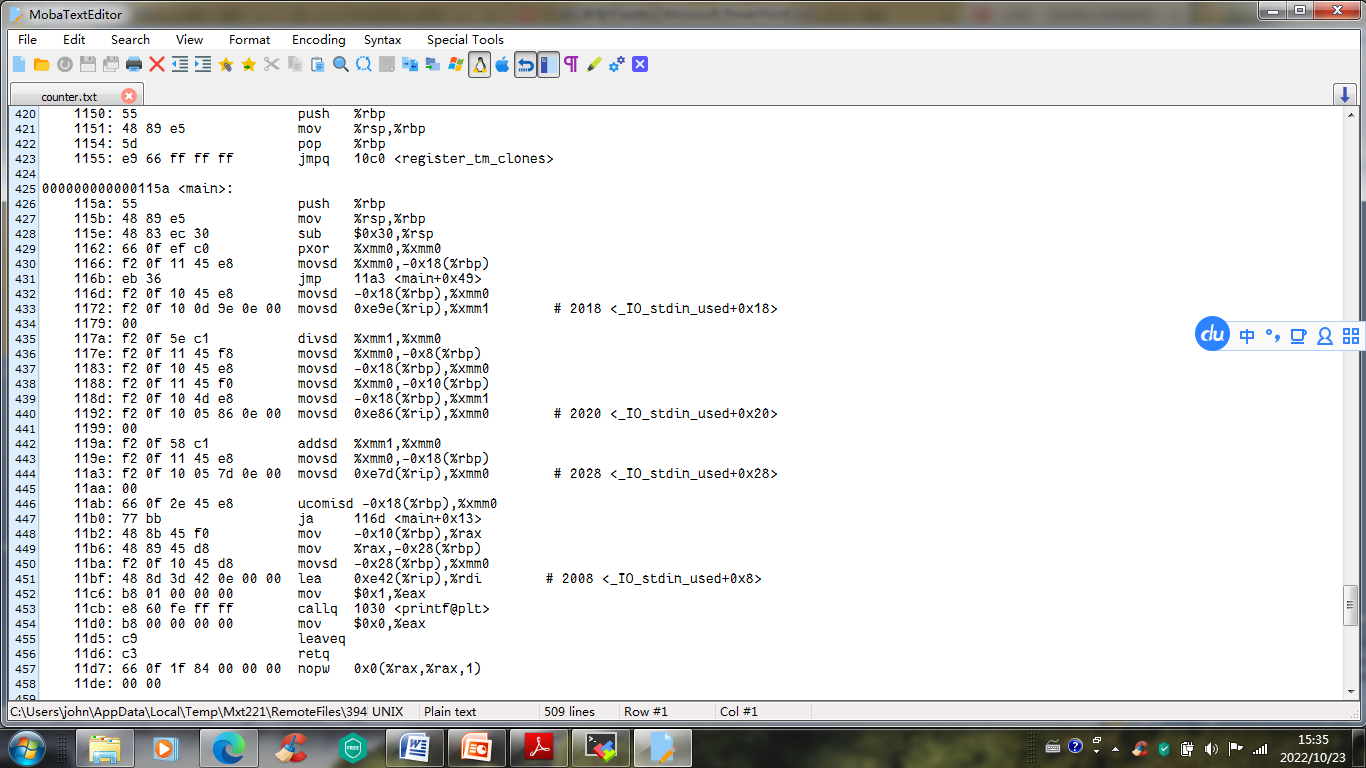
输入gcc counter.c -o counter用gcc编译counter.c,用time ./counter运行counter并输出运行时间。用gcc -O counter.c -o counter开启O1优化， 并运行，得到O0和O1优化的性能比较，如下图。并可知开启O1优化时运行时间更短，性能更高。



输入gcc -g counter.c -o counter编译源文件，输入gdb counter进入调试模式。输入list查看for循环所在行数，得知for循环所在行数为第7行，并用break 7在第七行设置断点。使用run运行counter，由于断点运行到for循环时终止，输入continue继续运行到结尾。

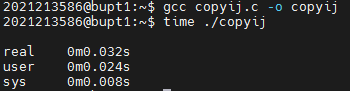


使用gcc -o counter counter.c,objudmp -s -d counter > counter.txt,反汇编并生成counter.txt汇编语言文本，如下图。

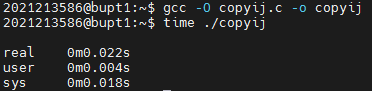


**实验二实验步骤：**首先输入vi copyij.c进入vi编辑器，并创建源代码copyij.c。将函数void copyij（int src[2048][2048],int dst[2048][2048]）复制到编辑器中，并编写main函数。分别使用gcc copyij.c -o copyij,gcc -O copyij.c -o copyij,gcc -O2 copyij.c -o copyij,gcc -O3 copyij.c -o copyij对copyij进行编译，并用time ./copyij输出运行时间。函数void copyji(int src[2048][2048],int dst[2048][2048])同理。运行结果如下：

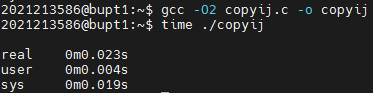
copyij函数使用O0优化：

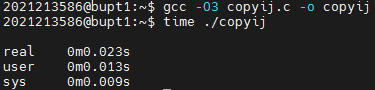


copyij函数使用O1优化：

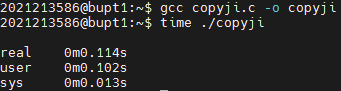


copyij函数使用O2优化：

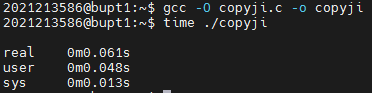


copyij函数使用O3优化：  
 

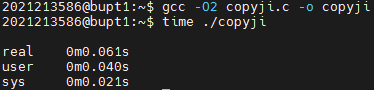
copyji函数使用O0优化：



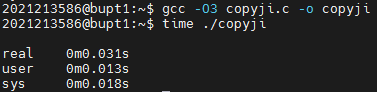
copyji函数使用O1优化：

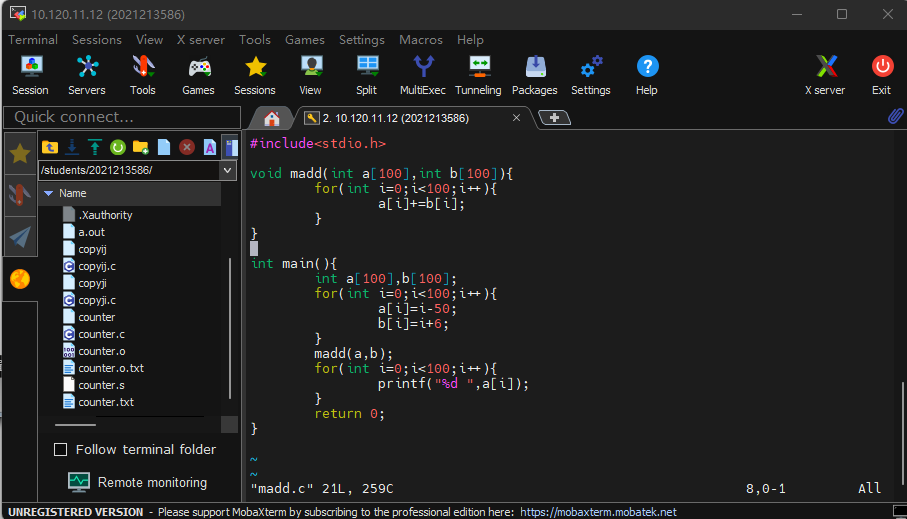


copyji函数使用O2优化：



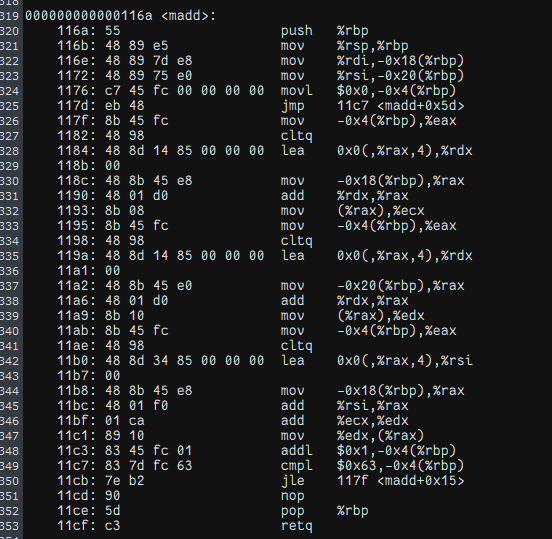
copyji函数使用O3优化：



**实验三实验步骤：**首先打开vi编写madd函数和调用madd函数的main函数，如下图：  


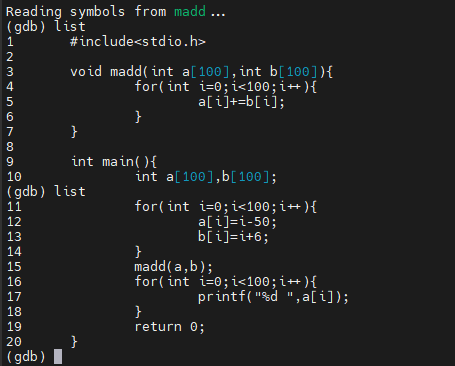
用gcc madd.c -o madd编译并用./madd运行该程序，发现程序正常运行。

用objdump反汇编该程序：输入objdump -s -d madd > madd.txt,生成汇编程序（以txt方式打开），从中找到madd函数的汇编程序如下图：



使用gdb进行调试：  
 首先输入gcc -g madd.c -o madd以调试模式编译madd.c，然后输入gdb madd进入gdb调试模式。

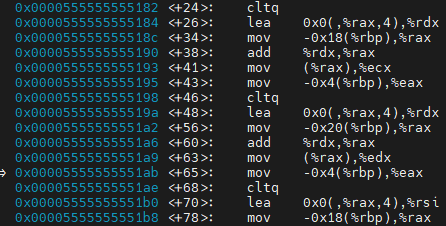
输入list查看程序行号（一个list显示十行，此处使用两次list命令）：

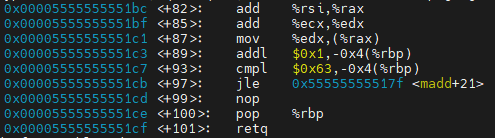


使用break madd在函数madd起始处设置断点。使用info break查看断点信息：



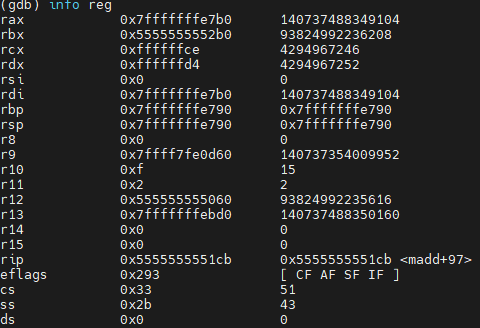
输入命令run，程序在第一个断点处停下。先使用disas命令反汇编madd函数，然后使用命令stepi（或nexti）逐指令执行函数madd，观察到语句a[i]+b[i]对应的汇编指令介于地址0x0000555555555182与0x00005555555551c1之间：



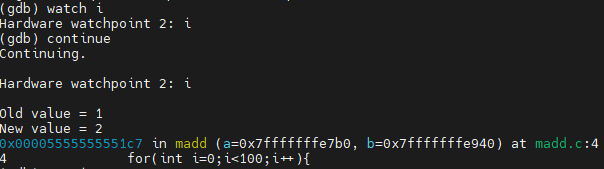


由此可知，a[i]与b[i]分别位于寄存器%rcx与%rdx。

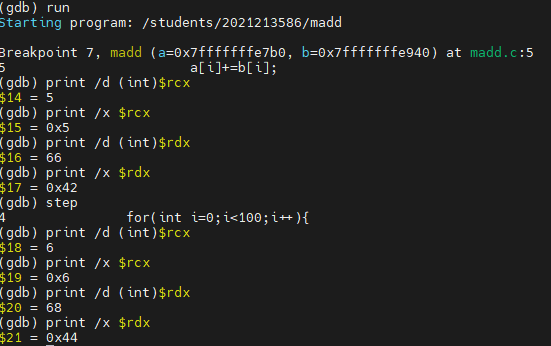
同时，使用info reg命令能够显示出各寄存器信息：



使用watch i可以监视变量i，当i的值发生变化时程序暂停，同时监视点也可以使用delete命令删除：



设置条件断点：break 5 if i==56，run。程序在执行a[56]+=b[56]前，使用命令print /d (int)$rcx显示寄存器%rcx的值，%rdx同理，适用step运行c语言语句a[i]+=b[i]后，再使用print，结果如下图（十六进制将/d改为/x即可）：



显示完后使用kill直接结束进程。

**实验四实验步骤：**

高内存需求

#include<stdio.h>

#include<stdlib.h>

int main(){

const int SIZE=100,NUM=1e7;  
 int \*p;

for(int i=0;i<NUM;i++){  
 p=(int\*)malloc(SIZE\*sizeof(int));

}

return 0;

}

代码思路：通过malloc不断向系统申请内存，每次申请100\*sizeof(int),申请1e7次。分别使用O0，O1，O2优化，运行效率如下图：

文本

描述已自动生成

可知优化等级越高，运算效率越高。

高计算量类型

#include<stdio.h>

int Fibonacci(int n){

if(n==0||n==1) return 1;

return Fibonacci(n-1)+Fibonacci(n-2);

}

int main(){

printf("%d\n",Fibonacci(40));

return 0;

}

代码思路：斐波那契数列不断递归调用，计算量较大，分别使用O0，O1，O2，O3运行结果如下图：

文本

描述已自动生成

文本

描述已自动生成

由实验结果可知，优化等级越高，程序运行效率越高。

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：实验一的主要目的是熟悉开发环境。在vi中编辑完counter.c后，使用gcc编译，生成可执行文件counter。使用gdb可以进行调试。使用objdump可以根据counter生成对应的汇编语言程序。

实验二：实验二使用控制变量的方法，通过控制函数类型和优化等级，可以得出结论：1、优化等级越高，程序运行速度越快；2、copyij运行效率高于copyji，原因是在执行copyij时，dst和src变化时地址是连续的。

实验三：实验三的目的是熟悉gdb的使用和理解汇编语言程序。通过观察可知变量所处寄存器，并可通过print指令验证。实验时还掌握了条件断点的设置方法。

实验四：编写高内存需求主要考虑多次调用malloc，编写高计算量程序时主要考虑多次递归调用。然后使用不同优化，并记录时间。

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）：

本次实验让我更深入地了解了Linux操作系统，学习了gdb和objdump的使用。在实验过程中，实验二的copyij和copyji由于数组a[2048][2048]较大，在未使用全局变量的调价下出现了segmentation fault，导致使用time ./输出程序运行时间时出错。后以改正。

有时在使用O0优化和O1优化时，运行时间差距不明显，甚至O0优化运行速度高于O1优化，目前还没有深入研究该问题。

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，我曾分别与以下各位同学就以下方面做过交流：

1、简单描述交流内容，例如：来自\*\*\*的建议，采用\*\*\*方式\*\*\*

2、

此外，我还参考了以下资料：

1. <https://baike.baidu.com/item/securecrt/8900957?fr=aladdin>
2. <https://baike.baidu.com/item/Linux?fromModule=lemma_search-box>
3. <https://baike.baidu.com/item/gcc/17570?fromModule=lemma_search-box>

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）

