

**实验报告**



**题目： 熟悉Linux系统及其相关软件环境**

**班 级： 2020211302**

**学 号： 2020211262**

**姓 名： 鄭毓恒**

**学 院： 计算机学院**

**2021年 10 月 31 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（10分）

1. Xshell 7（服务器：10.120.11.12）

XShell是一个强大的安全终端模拟软件，它支持SSH1,SSH2，以及Microsoft Windows平台的TELNET协议。XShell可以在Windows界面下用来访问远端不同系统下的服务器，从而比较好的达到远程控制终端的目的。

1. Linux

Linux，全称GNU/Linux，是一种免费使用和自由传播的类UNIX操作系统，其内核由林纳斯·本纳第克特·托瓦兹于1991年10月5日首次发布，它主要受到Minix和Unix思想的启发，是一个基于POSIX的多用户、多任务、支持多线程和多CPU的操作系统。它能运行主要的Unix工具软件、应用程序和网络协议。它支持32位和64位硬件。Linux继承了Unix以网络为核心的设计思想，是一个性能稳定的多用户网络操作系统。

1. Gcc编译器

GNU Compiler Collection，即 GNU 编译器套件，可以用来编译 C 语言程序，还可以处理 C++、Go、Objective -C 等多种编译语言编写的程序。

1. GDB调试工具

the GNU Project Debugger，是 GNU 发布的一个功能强大的 UNIX 程序调试工具，可以调试 Ada、C、C++、Objective-C 和 Pascal 等多种语言的程序。

1. Objdump命令反汇编

objdump命令是Linux下的反汇编目标文件或者可执行文件的命令，它以一种可阅读的格式让你更多地了解二进制文件可能带有的附加信息。

**大1班（1-4班）报告邮寄（最迟时间：2020年11月1日晚23：59）：**clavicle@bupt.edu.cn

**大2班（5-8班）报告邮寄（最迟时间：2020年11月3日晚23：59）：**

yangyyj98@bupt.edu.cn

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;

}

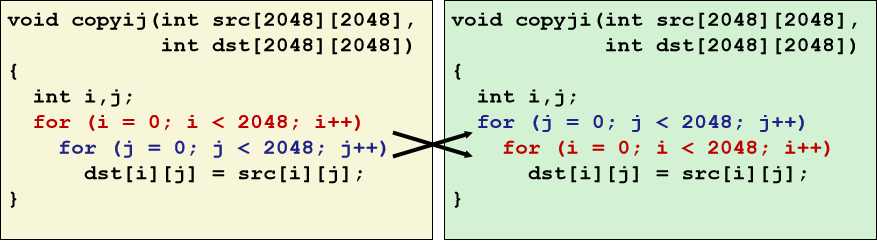
printf(Result is%lf\\n,result);

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；

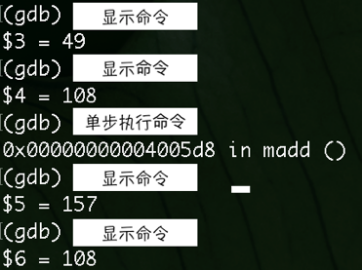
用gdb进行调试，练习如下gdb命令，给出截图；

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）：

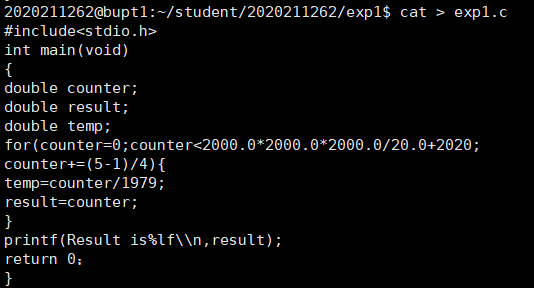


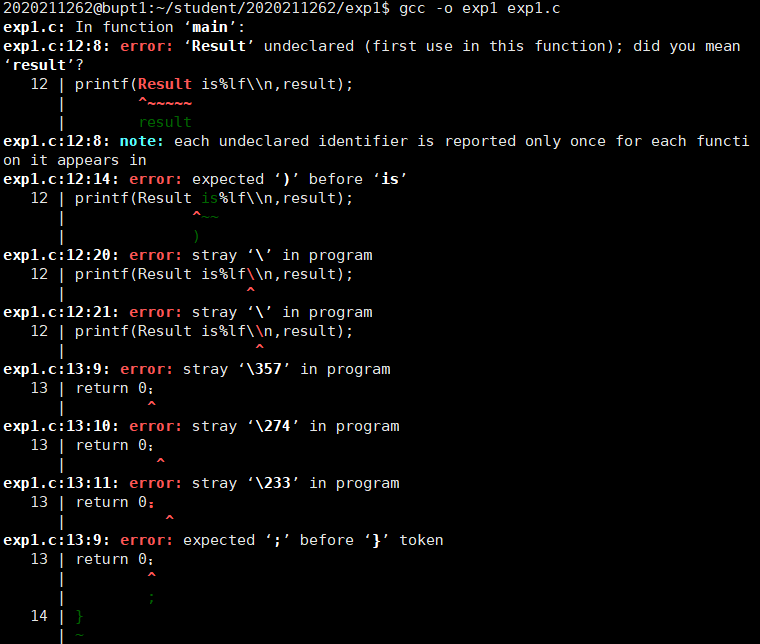
实验内容四（加分项，20分）

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

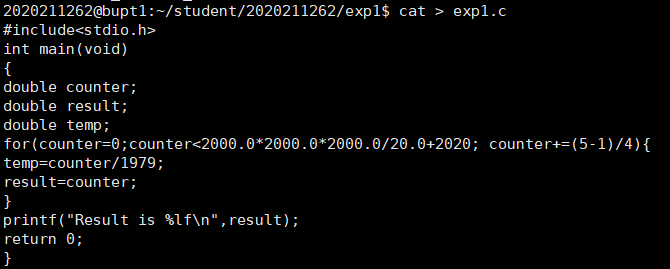
四、实验步骤（60-80分）

**实验内容一**：

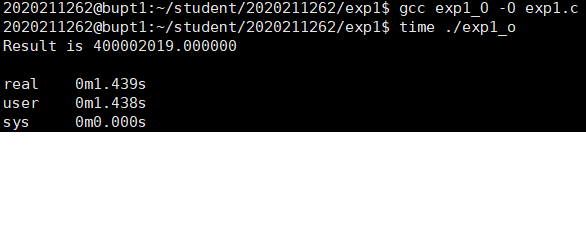


首先，先将源代码保存到exp1.c当中，然后进行编译。

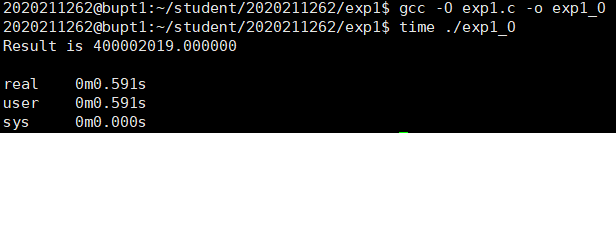
首次编译结果指出了以上语法错误，然后就错误对源代码进行更改。



代码更改如上，保存到exp1.c当中，进行第二次编译。

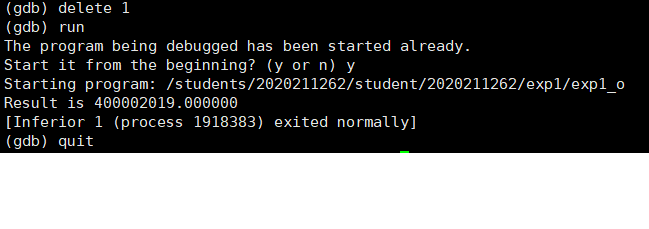
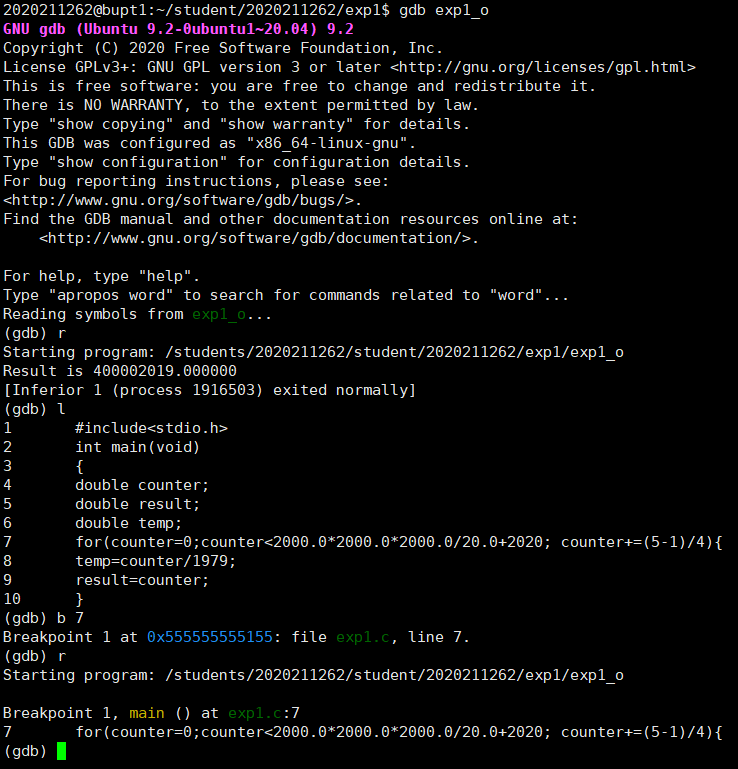


-o参数编译成功，随即运行程序，并以time指令得出运行时间。

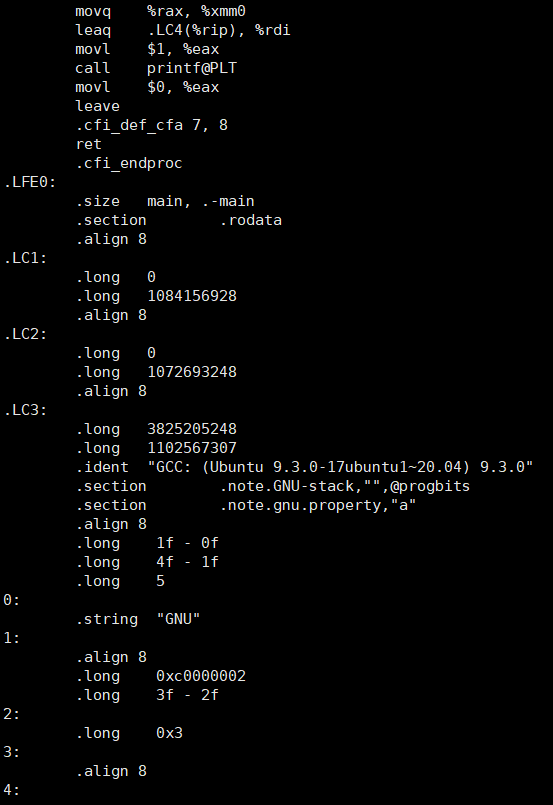
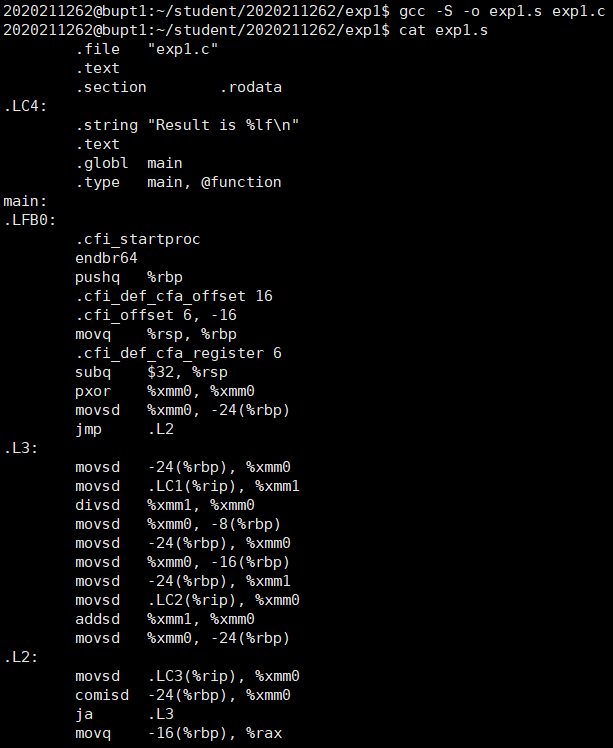


-O参数编译也成功，随即运行程序，并以time指令得出运行时间。

比较-o和-O参数编译所得的可执行文件各自的所需运行时间，发现-O参数编译所得的可执行文件的所需运行时间较-o参数快了将近一秒。-O参数编译的性能更好。接着进行gdb调试。

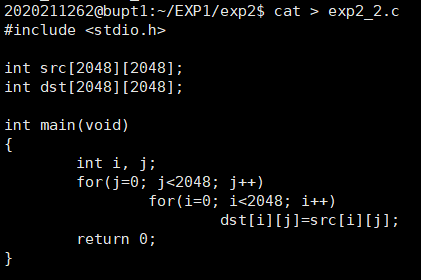
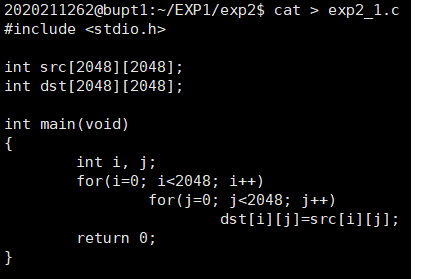


先用run指令对程序进行试运行，程序成功输出结果并退出。然后，用list指令查看程序源代码。接着，用break指令在for函数所在的第7行设置断点。再次以run运行程序，程序停在第7行的断点1，断点设置成功。gdb调试结束，以quit指令退出。

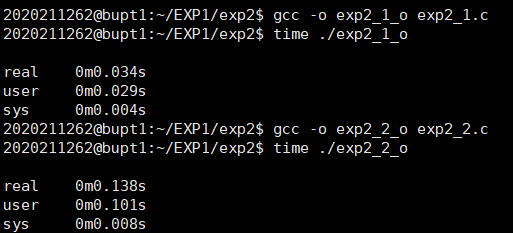


利用objdump工具生成exp1.c的汇编代码，可见main函数的汇编代码从.LFB0开始。

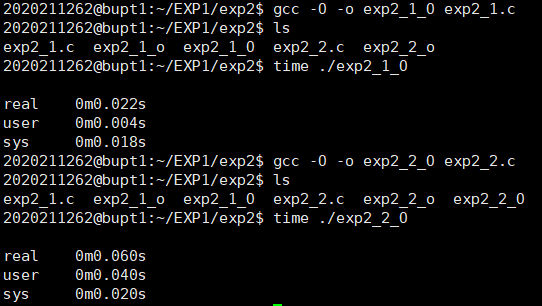
**实验内容二**：



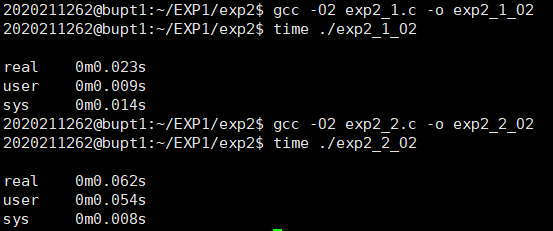
将两种算法分别输入至exp2\_1.c以及exp2\_2.c中。



首先使用-o参数编译并运行程序，可见exp2\_2.c所生成的可执行文件所需的运行时间更长。



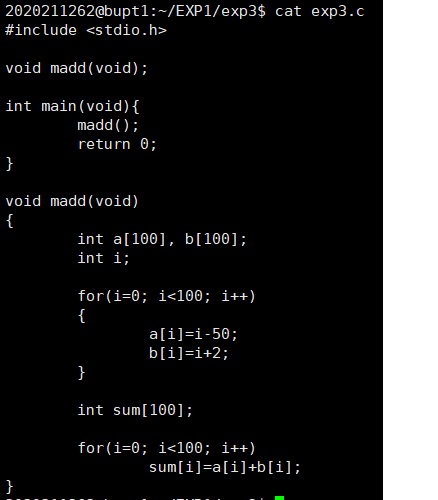
然后使用-O参数编译并运行程序，可见依然是exp2\_2.c所生成的可执行文件所需的运行时间更长。



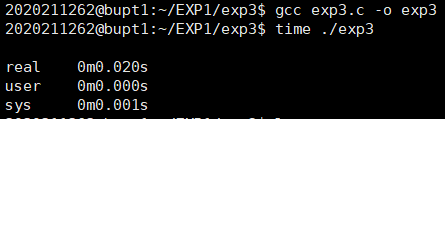
最后再使用-O2参数编译并运行程序，仍然是exp2\_2.c所生成的可执行文件所需的运行时间更长。

因此，列优先的算法在本次实验的环境中所需运行时间更长。

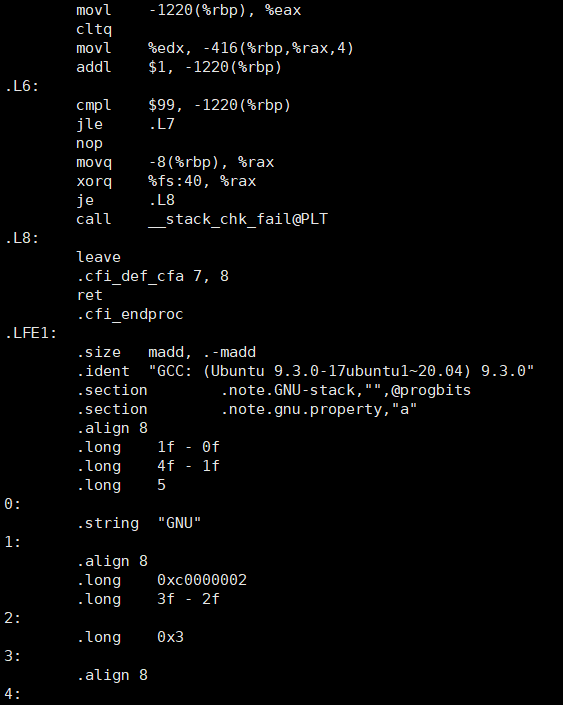
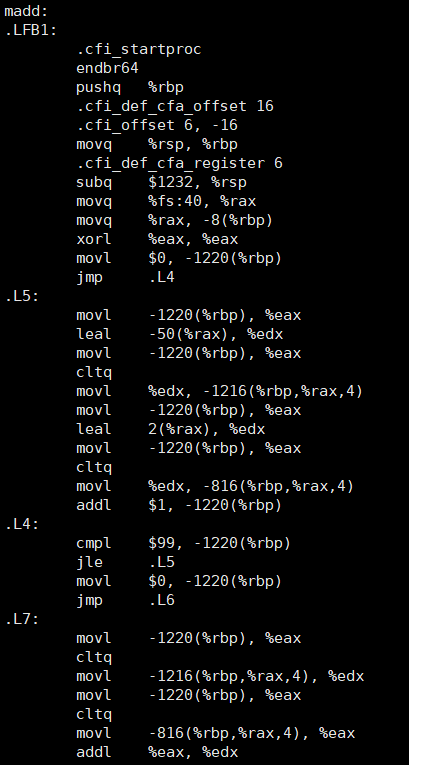
**实验内容三：**



首先，使用vi编辑器编写所需的C语言源程序，保存到exp3.c当中。

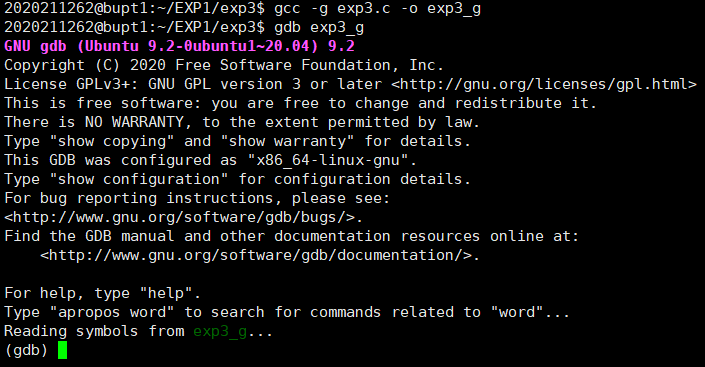


然后，使用gcc编译该程序，并试运行。程序编译和运行成功。

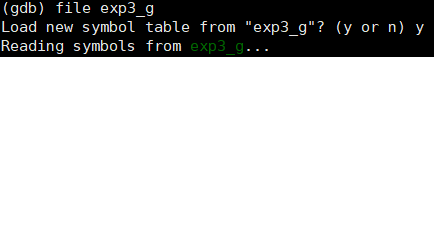


使用objdump工具生成汇编程序，以上两图为madd函数的汇编程序。

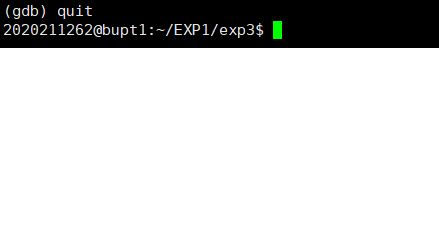
用gdb进行调试：



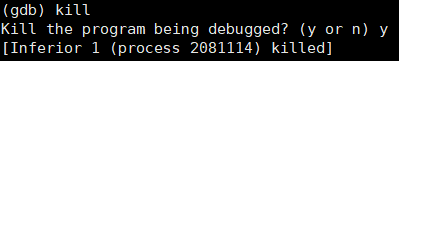
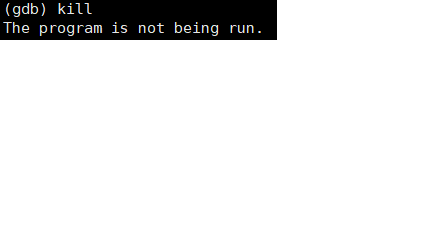
gdb命令成功。



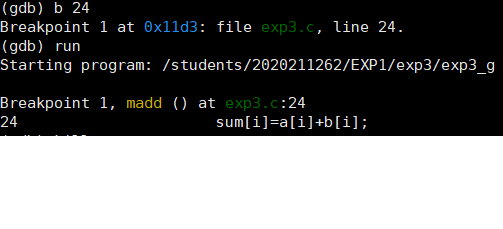
file命令成功。



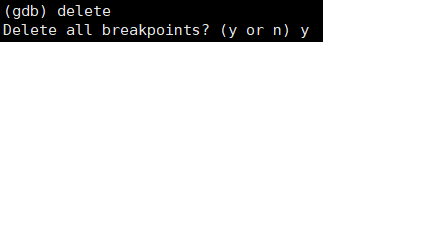
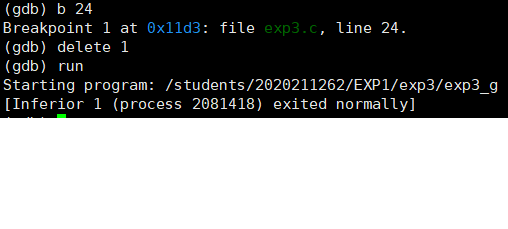
quit命令成功。



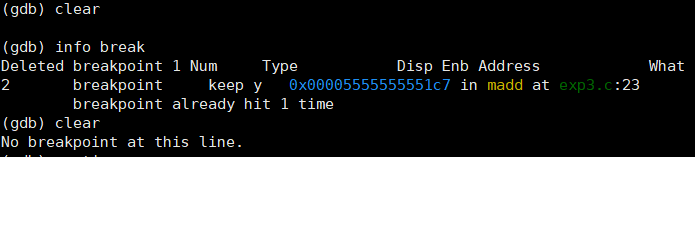
kill命令成功。左图为程序不在运行期间使用kill命令的结果，右图为程序运行期间的结果，如遇到断点后使用kill命令。



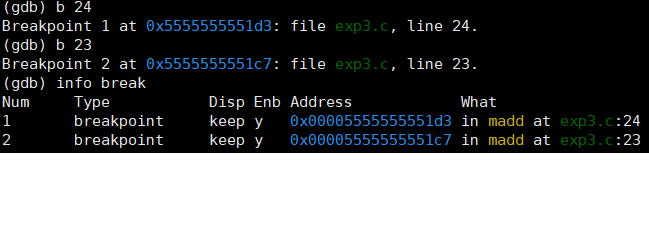
在第24行设置断点，然后运行程序，程序执行到第24行暂停。break命令成功。



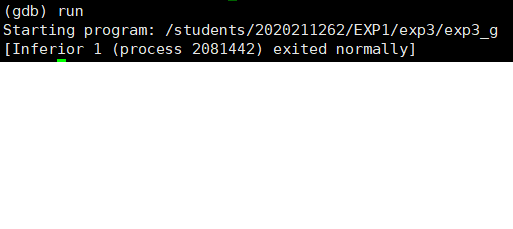
设置断点1后，再用delete 1命令将一号断点删除。随后程序运行不再停在断点1，delete命令成功。使用delete指令而不输入断点编号则可以删除所有断点。



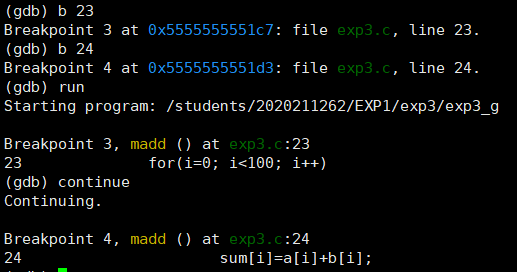
当程序运行到断点，在输入clear指令后，当前行的所有断点均被删除。clear命令成功。



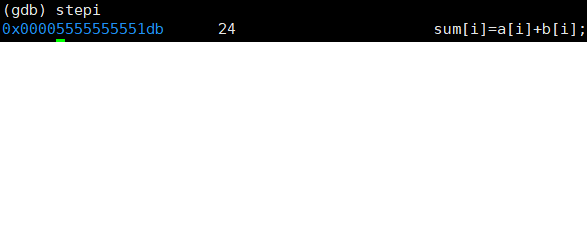
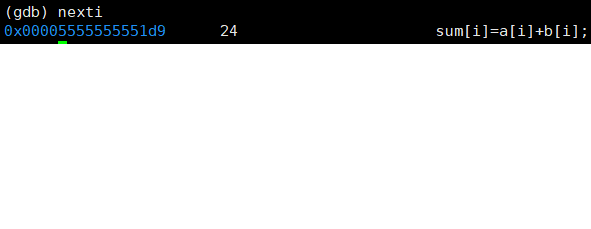
设置两个断点后，使用info break命令，输出两个断点的信息，info break指令成功。



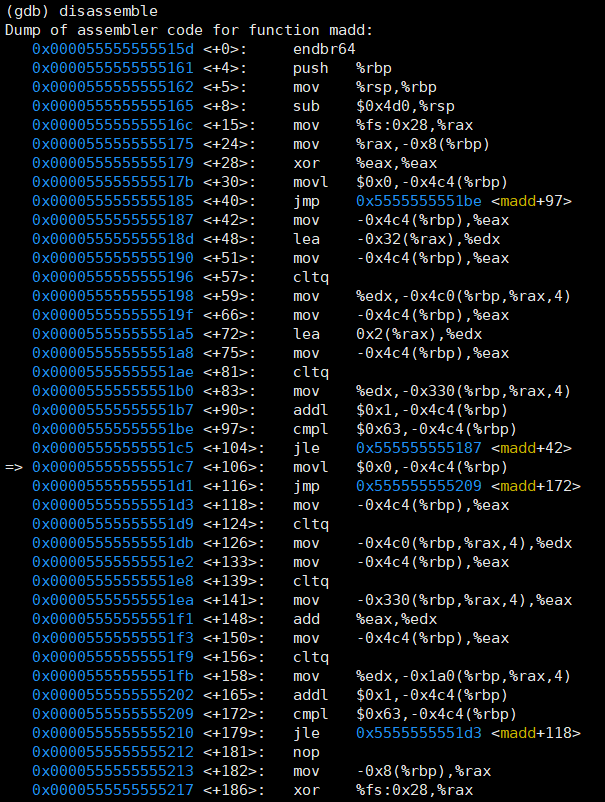
run命令成功。



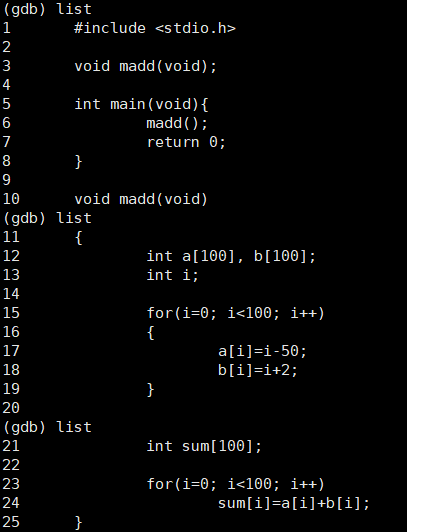
程序运行到断点后，程序暂停，然后使用continue命令，程序继续，continue命令成功。



使用nexti命令，程序执行下一条指令，nexti指令成功。使用stepi命令，程序接着执行stepi命令指定数量的指令，如无指定则执行一条，stepi指令成功。



当程序运行到madd函数，使用disassemble指令，得到当前函数的反汇编代码结果，disassemble指令成功。



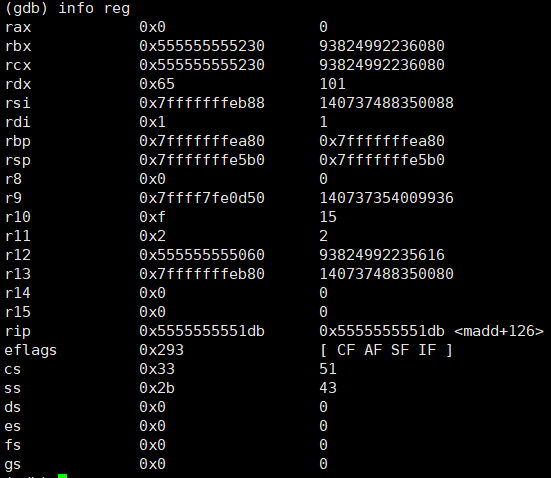
使用list命令，查看当前程序源代码，list指令成功。



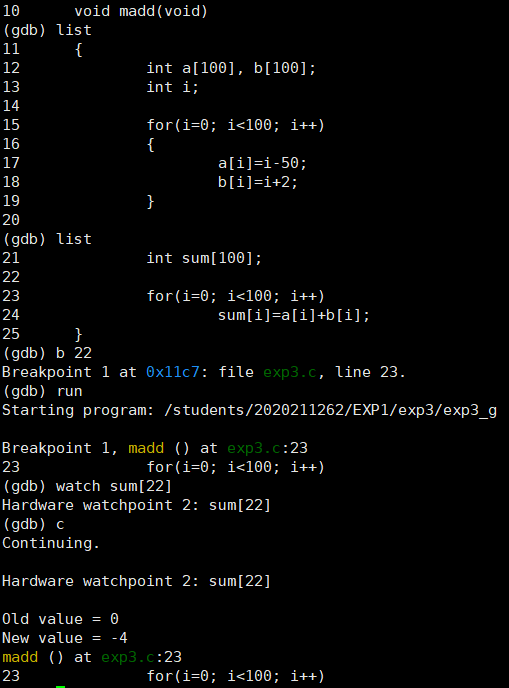
使用print命令，打印a[0]，print命令成功。



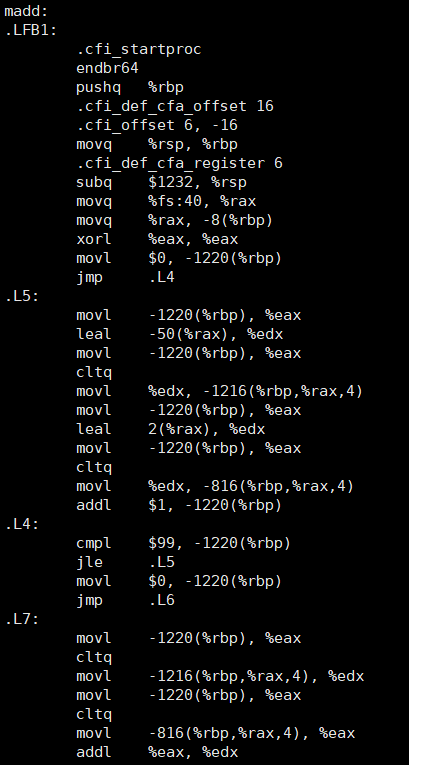
x命令成功。



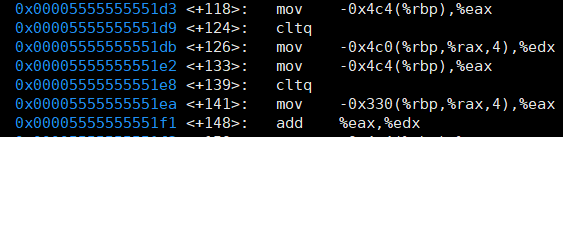
使用info reg命令，输出所有寄存器的值，info reg成功。



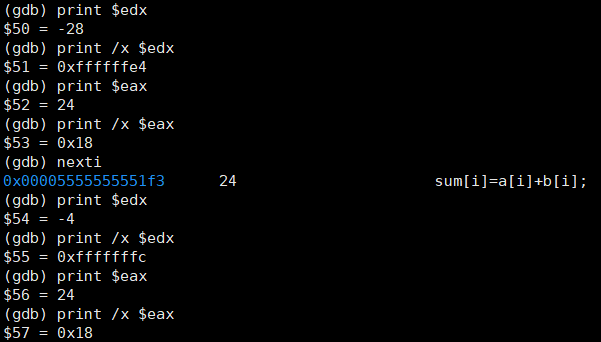
使用watch命令指定追踪变量sum[22]，程序运行时，sum[22]的值发生变化，程序暂停，watch命令成功。



在此前生成的madd函数的汇编代码中，找到了a[i]+b[i]对应的汇编指令，即上图最后一行。a[i]位于edx寄存器当中，b[i]位于eax寄存器当中。



上图为通过gdb调试中disassemble命令得到的madd函数的反汇编结果，显示了每条汇编指令的地址。使用单步指令以及continue, break, print等相关gdb指令，将程序进行至循环变量i=22.



本人学号为2020211262，a[22]+b[22]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如上。

五、实验分析（20分）

**实验内容一：**

由于所给程序源代码存在错误，实验时的工作思路是先利用gcc编译检测出出源代码中的错误再进行更改。当对正确的源代码使用gcc编译后，-O参数编译所得的程序运行时间比-o参数编译少，是因为-o参数相当于不对gcc编译做优化，而-O参数进行了优化。由于程序源代码正确，后面进行gdb调试以及objdump工具均成功。

**实验内容二**：

由于二维数组在存储时应该是以行优先的方式进行存储，因此设想先行后列的算法的所需时间应该较短。实验对gcc编译采用了-o, -O, -O2三种参数编译，每一种参数编译所得的程序的运行时间均是先行后列比先列后行短，符合设想。

**实验内容三**：

实验时的工作思路是先在madd函数利用循环对a和b数组进行初始化，然后利用循环完成数组a+b的工作，结果存储在sum数组中。后续使用gcc编译以及objdump工具生成汇编程序均成功。

gdb调试中，将题目指定的所有命令在各种情况以正确的方式运用了一次，均成功

使用objdump工具得到的汇编程序不足够，需利用gdb disassemble命令得到反汇编结果，利用显示出的每条汇编指令的地址协助阅读和辅助使用单步指令至a[22]+b[22]。由于先要让循环执行至i=22，先在循环内部设置断点，运行后重复利用continue指令进行下一次循环，同时需利用print输出i的值检查是否已经到达i=22。后续成功输出显示a[22]+b[22]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。

1. 实验总结（10分）

三个实验难度不大，但对了解linux系统的基本操作有极大帮助，有助于后续实验。通过这次实验，我更加熟悉linux操作的基本操作、掌握gcc编译方法、掌握gdb的调试工具使用、掌握objdump反汇编工具使用一级熟悉理解反汇编程序（对照源程序与objdump生成的汇编程序）。

在实验二中，曾遇到数组过大的问题，导致程序出现段错误。因此将src和dst数组更改为全局变量。

在实验三中，由于对汇编代码的了解不足，阅读代码找出对应指令时有困难。同时，gdb的info reg命令无法查看eax和edx两个运算数寄存器，导致本人以为这两个不属于寄存器的误解。

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、来自苏承烨同学的建议，在实验二中以全局变量定义src和dst数组。

2、来自赵维岗同学的建议，通过在循环内设置断点，利用continue命令进行下一次循环

此外，我还参考了以下资料：

1. 多个网站对实验所用工具的简介和定义

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）