# Linux平台C程序GDB调试工具

实验名称：Linux平台C程序GDB调试工具

学时安排：2课时 指导老师：李赞

实验类别：验证型、设计型 实验要求：1人1组

学号： 姓名： 班级：

## 一、实验目的和任务

1. 本实验要求复习和掌握Linux平台的C程序开发过程。

2. 本实验要求掌握GDB调试工具的使用。

## 二、实验设备介绍

本实验主要用到的配置环境如下：

1.软件环境： win10操作系统，VMware workstation，ubuntu18。

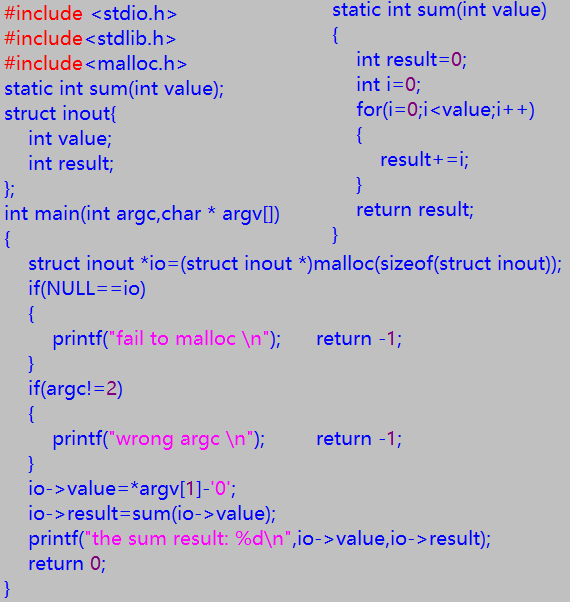
## 三、实验内容和步骤

### 3.1 Linux中的gdb调试工具介绍

GDB是UNIX和Linux平台下字符界面的调试工具，可以用它在linux上调试C和C++程序，它提供了以下的功能：

|  |  |
| --- | --- |
| gdb命令 | gdb命令 |
| 1.设置断点以及 | 10.生成内核转储文件 |
| 2.显示栈帧 | 11. attach到进程 |
| 3.显示变量、 | 12.反复执行 |
| 4.显示寄存器、 | 13.初始化文件 |
| 5.单步执行 | 14.设置源码目录 |
| 6.监视点、 | 15.TUI调试 |
| 7.改变变量的值。 | 16.Catchpoint |
| 8.监视点、 | 17.自定义脚本。 |
| 9.改变变量的值。 | 18.dump内存到文件 |

要使用GDB对源代码进行调试，在使用GCC工具编译时需加上-g开关。例如下面的源程序gdbsum.c：



首先使用-g开关的gcc命令：

gcc -o tt gdbsum.c –g

有两种运行用法，一是在编译成功后输入gdb + 编译后的文件名，二是在命令行中先输入gdb进入gdb调试模式，再输入file 加载要调试的程序。

gdb tt

|  |  |  |
| --- | --- | --- |
| 命令 | 解释 | 示例 |
| file | 加载被调试的可执行程序文件。 | (gdb) file ./tt |
| r | 运行被调试的程序。程序暂停在第一个可用断点处。 | (gdb) r |
| c | 继续执行被调试程序，直至下一个断点或程序结束。continue的缩写 | (gdb) c |
| b | <行号>设置断点。 | (gdb) b 8 |
| b | <函数名称>指定断点位置为函数名位置。 | (gdb) b main |
| b | \*<函数名称>将断点设置在"由编译器生成的prolog代码处"。 | (gdb) b \*main |
| b | \*<代码地址>将断点设置在"由编译器生成的代码处"。 | (gdb) b \*0x804835c |
| d | 删除指定编号的某个断点，或删除所有断点。 | (gdb) d |
| clear | 删除断点 |  |
| s | s: 执行一行源程序代码，如果此行代码中有函数调用，则进入该函数；(单步跟踪进入) | (gdb) s |
| bt | 显示栈帧 bt full可以显示局部变量 | (gdb) bt |
| n | n: 执行一行源程序代码，此行代码中的函数调用也一并执行。(单步跟踪) | (gdb) n |
| si,ni | 针对的是汇编指令的单步调试。 | (gdb) si (gdb) ni |
| p | 显示指定变量（临时变量或全局变量）的值。print | (gdb) p i |
| display | 设置程序中断后欲显示的数据及其格式。 | (gdb) display /i $pc |
| i | 显示寄存器信息 | (gdb) i r。 |
| q | 退出GDB调试环境。 | (gdb) q |
| watch | 监视点 | (gdb) watch |
| set variable | 修改变量的值 |  |
| help | 对GDB名种命令的解释说明。 | (gdb) help display |

查看栈信息 info

|  |  |
| --- | --- |
| 命令 | 解释 |
| info breakpoints | 查看断点信息 |
| info watchpoints | 查看观察点信息 |
| info program | 查看程序状态 |
| info stack | 程序调用栈信息 |
| info register | 查看寄存器值 |
| info threads | 线程信息 |
| info frame | 栈帧，信息比较复杂 |
| info args | main函数的参数及数值 |
| info line | 源代码在内存的地址 |
| info locals | 当前函数的局部变量信息 |
| info symbol | 显示全局变量信息 |
| info function | 显示所有函数名称 |
| info file | 当前调试的文件名 |

//008.c 使用gdb对程序调试

#include <stdio.h>

#include<stdlib.h>

#include<malloc.h>

static int sum(int value);

struct inout{

int value;

int result;

};

int main(int argc,char \* argv[])

{

struct inout \*io=(struct inout \*)malloc(sizeof(struct inout));

if(NULL==io)

{

printf("fail to malloc \n"); return -1;

}

if(argc!=2)

{

printf("wrong argc \n"); return -1;

}

io->value=\*argv[1]-'0';

io->result=sum(io->value);

printf("the sum result: %d\n",io->value,io->result);

return 0;

}

static int sum(int value)

{

int result=0;

int i=0;

for(i=0;i<value;i++)

{

result+=i;

}

return result;

}

用 "set args 参数值1 参数值2...."来设置main的输入参数

上机实验要求：请参照ppt演示内容，完成程序的编译，设置函数参数，设置断点，显示函数帧信息，查看变量值，查看函数的汇编指令。

### 3.2使用Gdb对calblank.c调试

下面的程序目标是统计给定字符串中空格的数量，但是该程序不能成功运行得到正确结果，参考ppt内容使用gdb来查找程序设计的错误原因。

//009.c 统计空格数量—程序存在bug

#include "stdio.h"

#include "stdlib.h"

#include "string.h"

void GetMemory(char \*p)

{

p = (char \*)malloc(100);

}

void CalcBlank(void)

{

char \*str = NULL;

char \*ptr = NULL;

unsigned int i = 0;

GetMemory((char \*)str);

strcpy(str, "Karl marx is born in Germany and German is his native language\n");

ptr = str;

while(('\0' != (\*ptr)))

{

if (\*ptr == ' ')

{

i++;

}

ptr++;

}

printf("the blank number is %d\n",i);

free(ptr);

}

int main(void)

{

CalcBlank();

return 0;

}

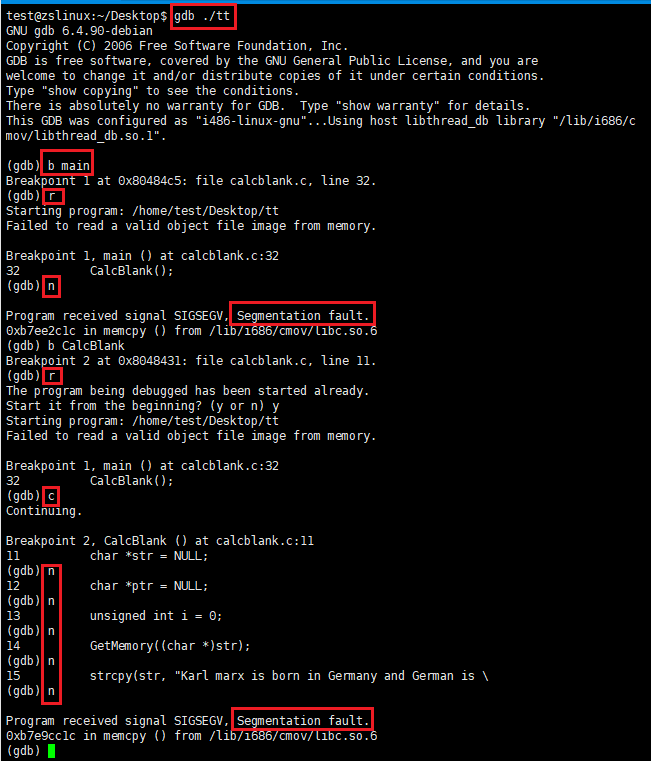
calbank.c源对文件中的空格进行统计，输入编译命令对calbank.c源程序进行编译：

gcc -o -g tt calcblank.c

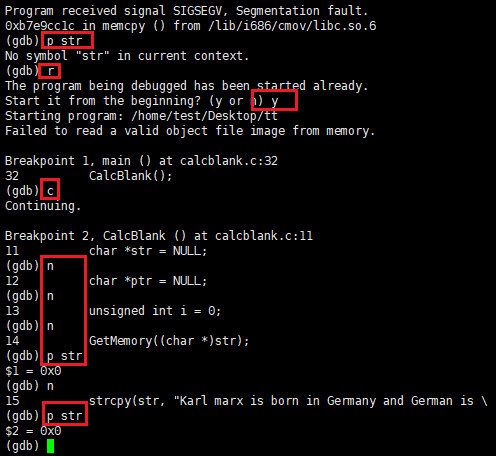
执行程序后有错误信息。



程序运行出现了Segmentation异常信息。使用gdb命令启动tt程序，在gdb环境下，b main命令添加断点在main函数处，r 命令开始运行，n 命令是单步执行代码，c 命令是运行到下一个断点处。



p str 命令显示变量str的当前值。



经过逐行调试，发现在执行strcpy函数时，str指针的值为 0x0，这说明str指针并没有分配到实际内存空间。原因是GetMemory函数进行内存分配时，字符串对应的内存地址因为函数传值的原因，没有传递回原字符串指针变量，因此面要对str执行取地址后再传入函数。实际调用为GetMemory(char \*\*p)，相应的函数其它地方也需要适当修改。最后正确的输出结果是：

the blank number is 11

上机作业要求：请参照ppt演示内容，使用gdb对该程序进行调试，截屏出现segmentation fault位置的图片，提交到学习通平台。

## 四、程序调试中遇到的问题和解决过程及运行结果

根据报告内容对指定程序进行调试，请首先排查程序出错的原因，最后得到正确的运行结果，并记录程序出错的代码及修正的内容。