homework2 实验报告

一、实验名称

程序的编辑、编译和调试

二、实验目的

通过对源程序的编辑、编译、链接、调试，了解高级语言源程序和目标机器代码的不同表示及其相互转换；

熟悉开发环境和 GDB 调试命令

三、实验环境

ubuntu 14.4　64bits +gcc4.8

四、实验内容

1. 补全程序

简单选择排序函数 selectionSort()，其基本思想是每一趟从待排序的数据元素中选择最小（或最大）的一个元素作为首元素，直到所有元素排完为止。即第 1 趟，在待排序记录 a[1]~a[n]中选出最小的记录，将它与 a[1]交换；第 2 趟，在待排序记录 a[2]~a[n]中选出最小的记录，将它与 a[2]交换；以此类推，第 i 趟在待排序记录 a[i]~a[n]中选出最小的记录，将它与 a[i]交换，使有序序列不断增长直到全部排序完毕。

qjw@qjw-VirtualBox: ~/hw2

```
#include"selectionSort.h"
void selectionSort(int s[], int n){
        //TODO: write your code here
        int min;//记录最小值的位置
        int i;
        for(i=0;i!=n-1;i++){
                min=i;
                int j;
                for(j=i+1;j!=n;j++){
                        if(s[j]<s[min])
                                min=j;
                }//找到最小值下标

                if(min!=i){
                        int temp;
                        temp=s[i];
                        s[i]=s[min];
                        s[min]=temp;
                }//交换
        }
}
```

求和函数 add()就是循环累加

```
#include"add.h"
int add(int s[], int n){
        //TODO: write your code here
        int result=0;
        int i;
        for(i=0;i!=n;i++){
                result+=s[i];
        }
        return result;
}
```

printArray()也是循环遍历打印数组内容

main.c，首先调用 srand()函数，以系统时间作为生成随机数的种子，调用 rand()，赋给 randValue 1-10 的任意值，生成十个随机数赋给数组 a[]和 b[]。然后输出原始的随机数列。打印功能选项列表，读入选项后执行操作，1 对应选择排序，2 对应求和，3 对应打印数组，4 退出该程序。

2. 将源程序文件预处理、编译、汇编和链接生成可执行文件

两种方法：

① 使用 gcc 直接生成可执行文件

```
qjw@qjw-VirtualBox:~/hw2$ gcc -o main1 main.c selectionSort.c add.c printArray.c
```

② 先用 gcc -c 命令将所有.c 文件编译成.o 文件，再用 ld 命令进行链接，以生成可执行目标文件。其中用 ld 命令链接时要包含很多系统库，可以使用 gcc -v main.c 来查看需要哪些库

```
qjw@qjw-VirtualBox:~/hw2$ gcc -c main.c
qjw@qjw-VirtualBox:~/hw2$ gcc -c selectionSort.c
qjw@qjw-VirtualBox:~/hw2$ gcc -c add.c
qjw@qjw-VirtualBox:~/hw2$ gcc -c printArray.c
```

得到需要链接的系统库：

```
qjw@qjw-VirtualBox:~/hw2$ gcc -v main.c
```

```
 /usr/lib/gcc/x86_64-linux-gnu/4.8/collect2 --sysroot=/ --build-id --eh-frame-hdr -m elf_
x86_64 --hash-style=gnu --as-needed -dynamic-linker /lib64/ld-linux-x86-64.so.2 -z relro
/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/crt1.o /usr/lib/gcc/x86_64-li
nux-gnu/4.8/../../../x86_64-linux-gnu/crti.o /usr/lib/gcc/x86_64-linux-gnu/4.8/crtbegin.o
 -L/usr/lib/gcc/x86_64-linux-gnu/4.8 -L/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-
linux-gnu -L/usr/lib/gcc/x86_64-linux-gnu/4.8/../../../../lib -L/lib/x86_64-linux-gnu -L/
lib/../lib -L/usr/lib/x86_64-linux-gnu -L/usr/lib/../lib -L/usr/lib/gcc/x86_64-linux-gnu/
4.8/../../.. /tmp/ccKFFqEN.o -lgcc --as-needed -lgcc_s --no-as-needed -lc -lgcc --as-need
ed -lgcc_s --no-as-needed /usr/lib/gcc/x86_64-linux-gnu/4.8/crtend.o /usr/lib/gcc/x86_64-
linux-gnu/4.8/../../../x86_64-linux-gnu/crtn.o
```

```
/tmp/ccKFFqEN.o：在函数‘main’中：
```

生成的/tmp/ccKFFqEN.o 即为 main.c 编译出来的 main.o 文件，也就是临时文件，手动链接时需要排除这些临时文件，将 collect2 换成 ld，加上-o main main.o add.o selectionSort.o printArray.o，后面部分复制粘贴刚刚查找出来的系统库即可，注意把临时文件删掉。

```
qjw@qjw-VirtualBox:~/hw2$ ld -o main main.o add.o selectionSort.o printArray.o --sysroot=
/ --build-id --eh-frame-hdr -m elf_x86_64 --hash-style=gnu --as-needed -dynamic-linker /l
ib64/ld-linux-x86-64.so.2 -z relro /usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linu
x-gnu/crt1.o /usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/crti.o /usr/lib/
gcc/x86_64-linux-gnu/4.8/crtbegin.o -L/usr/lib/gcc/x86_64-linux-gnu/4.8 -L/usr/lib/gcc/x8
6_64-linux-gnu/4.8/../../../x86_64-linux-gnu -L/usr/lib/gcc/x86_64-linux-gnu/4.8/../../..
/../lib -L/lib/x86_64-linux-gnu -L/lib/../lib -L/usr/lib/x86_64-linux-gnu -L/usr/lib/../l
ib -L/usr/lib/gcc/x86_64-linux-gnu/4.8/../../.. -lgcc --as-needed -lgcc_s --no-as-needed
-lc -lgcc --as-needed -lgcc_s --no-as-needed /usr/lib/gcc/x86_64-linux-gnu/4.8/crtend.o /
usr/lib/gcc/x86_64-linux-gnu/4.8/../../../x86_64-linux-gnu/crtn.o
```

注意到在 gcc -v 时出现一些错误

```
main.c:(.text+0x9b)：对‘printArray’未定义的引用
main.c:(.text+0x117)：对‘selectionSort’未定义的引用
main.c:(.text+0x12a)：对‘add’未定义的引用
main.c:(.text+0x159)：对‘printArray’未定义的引用
main.c:(.text+0x16f)：对‘printArray’未定义的引用
```

原因应该就是主要是.c 文件编译为.o 文件的时候并不需要函数的具体实现，只要有函数的原型即可。但是在链接为可执行文件的时候就必须要具体的实现了。所以链接以后错误即不再有。

（两种不同方式得到的可执行文件）

得到两种不同方式生成的可执行文件后，执行

```
qjw@qjw-VirtualBox:~/hw2$ ./main1

random array:     6    8    9    4    6   10    5    3    6    4

1.selection sort
2.sum
3.print result
4.exit
choose a number: 2

LibreOffice Impress 1

Done!


1.selection sort
2.sum
3.print result
4.exit
choose a number: 1

Done!


1.selection sort
2.sum
3.print result
4.exit
choose a number: 3
```

```
original array:     6    8    9    4    6   10    5    3    6    4

sorted array:       3    4    4    5    6    6    6    8    9   10

Done!


1.selection sort
2.sum
3.print result
4.exit
choose a number: 4

Done!

qjw@qjw-VirtualBox:~/hw2$
```

```
qjw@qjw-VirtualBox:~/hw2$ ./main

random array:     4    9   10    8    3    8    8    2    4    6

1.selection sort
2.sum
3.print result
4.exit
choose a number: 2

result of sum: 62

Done!


1.selection sort
2.sum
3.print result
4.exit
choose a number: 1

Done!


1.selection sort
2.sum
3.print result
4.exit
choose a number: 3

original array:     4    9   10    8    3    8    8    2    4    6

sorted array:       2    3    4    4    6    8    8    8    9   10
```

```
Done!


1.selection sort
2.sum
3.print result
4.exit
choose a number: 4

Done!

qjw@qjw-VirtualBox:~/hw2$
```

3. 使用 objdump 命令进行反汇编

预处理得到的.i 文件和编译得到的.s 文件都是可显示的文本文件，而汇编得到的可重定位目标文件.o 和最后链接得到的可执行文件都是机器语言目标文件，不是可以直接显示的文本文件，而是不可显示的二进制文件，所以需要反汇编，反汇编程序能将机器指令反过来翻译成汇编指令。

通过“objdump -S main.o>main0.txt”命令显示可重定位目标文件 main.o

通过“objdump -S main>main.txt”命令显示可执行目标文件 main

```
0000000000000000 <main>:
   0:   55                      push   %rbp
   1:   48 89 e5                mov    %rsp,%rbp
   4:   48 83 c4 80             add    $0xffffffffffffff80,%rsp
   8:   c7 45 98 00 00 00 00    movl   $0x0,-0x68(%rbp)
   f:   bf 00 00 00 00          mov    $0x0,%edi
  14:   e8 00 00 00 00          callq  19 <main+0x19>
  19:   48 89 c7                mov    %rax,%rdi
  1c:   b8 00 00 00 00          mov    $0x0,%eax
  21:   e8 00 00 00 00          callq  26 <main+0x26>
```

```
0000000000400610 <main>:
  400610:       55                      push   %rbp
  400611:       48 89 e5                mov    %rsp,%rbp
  400614:       48 83 c4 80             add    $0xffffffffffffff80,%rsp
  400618:       c7 45 98 00 00 00 00    movl   $0x0,-0x68(%rbp)
  40061f:       bf 00 00 00 00          mov    $0x0,%edi
  400624:       e8 b7 ff ff ff          callq  4005e0 <time@plt>
  400629:       48 89 c7                mov    %rax,%rdi
  40062c:       b8 00 00 00 00          mov    $0x0,%eax
  400631:       e8 8a ff ff ff          callq  4005c0 <srand@plt>
```

（两段汇编代码均为节选）

注意到两段代码内容大致相同，只是在可重定位目标文件 main.o 中 main 函数的起始位置为 0，而在可执行文件 main 中 main 函数的起始位置为 0x400610。

4. 使用 GDB 命令进行各种调试

```
qjw@qjw-VirtualBox:~/hw2$ gcc -g -o main main.c selectionSort.c add.c printArray.c
qjw@qjw-VirtualBox:~/hw2$ gdb main
GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.3) 7.7.1
Copyright (C) 2014 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from main...done.
```

使用 l 命令列出部分源码，在循环处添加第一个断点，在三个功能函数处添加断点，info break 输出断点信息

```
(gdb) l
5
6       #define bool char
7       #define true 1
8       #define false 0
9       #define LENGTH 10
10
11      int main(){
12              int a[LENGTH],i;
13              int b[LENGTH];
14              int randValue=0;
(gdb)
15              srand(time(NULL));
16
17              for(i=0;i<LENGTH;i++){
18                      randValue=1+(int)rand()%LENGTH;
19                      a[i]=randValue;
20                      b[i]=a[i];
21              }
22
23              printArray(a,LENGTH,"\nrandom array: ");
24
(gdb) b 20
Breakpoint 1 at 0x40076a: file main.c, line 20.
```

```
(gdb) l
25              bool flag=true;
26              while(flag){
27                      printf("\n1.selection sort\n2.sum\n3.print result\n4.exit");
28                      printf("\nchoose a number: ");
29
30                      int number=0;
31                      scanf("%d",&number);
32
33                      int sum=0;
34                      switch(number)
(gdb)
35                      {
36                              case 1:
37                                      selectionSort(a,LENGTH);
38                                      break;
39                              case 2:
40                                      sum=add(a,LENGTH);
41                                      printf("\nresult of sum: %d\n",sum);
42                                      break;
43                              case 3:
44                                      printArray(b,LENGTH,"\noriginal array:\t");
(gdb) b selectionSort
Breakpoint 2 at 0x4008ae: file selectionSort.c, line 6.
(gdb) b add
Breakpoint 3 at 0x400993: file add.c, line 4.
(gdb) b printArray
Breakpoint 4 at 0x4009e0: file printArray.c, line 3.
(gdb) info break
Num     Type           Disp Enb Address            What
1       breakpoint     keep y   0x000000000040076a in main at main.c:20
2       breakpoint     keep y   0x00000000004008ae in selectionSort at selectionSort.c:6
3       breakpoint     keep y   0x0000000000400993 in add at add.c:4
4       breakpoint     keep y   0x00000000004009e0 in printArray at printArray.c:3
```

r 命令运行程序在生成第一个随机数后中断，使用 info registers 命令输出所有寄存器内容，使用 info locals 指令可以输出局部变量

```
(gdb) r
Starting program: /home/qjw/hw2/main

Breakpoint 1, main () at main.c:20
20                      b[i]=a[i];
(gdb) info registers
rax            0x0      0
rbx            0x0      0
rcx            0x7      7
rdx            0x8      8
rsi            0x7fffffffdebc   140737488346812
rdi            0x7ffff7dd36e0   140737351857888
rbp            0x7fffffffdf60   0x7fffffffdf60
rsp            0x7fffffffdee0   0x7fffffffdee0
r8             0x7ffff7dd30c8   140737351856328
r9             0x7ffff7dd3140   140737351856448
r10            0x7fffffffdca0   140737488346272
r11            0x7ffff7a4dfb0   140737348165552
r12            0x400610 4195856
r13            0x7fffffffe040   140737488347200
r14            0x0      0
r15            0x0      0
rip            0x40076a 0x40076a <main+109>
eflags         0x202    [ IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
```

```
(gdb) info locals
a = {8, 32767, -134344704, 32767, -134225464, 32767, 0, 0, 1, 0}
i = 0
b = {-8352, 32767, 0, 0, 4196928, 0, 4195856, 0, -8128, 32767}
randValue = 8
flag = 0 '\000'
```

c 命令使程序继续运行到达下一个断点，进入第二轮循环，经过两次循环发现%rax 存储的是计数变量，%rcx 是生成的随机数，%rdx 是存入数组里的值，后面几次循环直接用 info registers rax rbx rcx rdx 打印这四个寄存器的内容

```
(gdb) c
Continuing.

Breakpoint 1, main () at main.c:20
20                      b[i]=a[i];
(gdb) info registers
rax            0x1      1
rbx            0x0      0
rcx            0x4      4
rdx            0x5      5
rsi            0x7fffffffdebc   140737488346812
rdi            0x7ffff7dd36e0   140737351857888
rbp            0x7fffffffdf60   0x7fffffffdf60
rsp            0x7fffffffdee0   0x7fffffffdee0
r8             0x7ffff7dd30cc   140737351856332
r9             0x7ffff7dd3140   140737351856448
r10            0x7fffffffdca0   140737488346272
r11            0x7ffff7a4dfb0   140737348165552
r12            0x400610 4195856
r13            0x7fffffffe040   140737488347200
r14            0x0      0
r15            0x0      0
rip            0x40076a 0x40076a <main+109>
eflags         0x202    [ IF ]
cs             0x33     51
ss             0x2b     43
ds             0x0      0
es             0x0      0
fs             0x0      0
gs             0x0      0
```

```
(gdb) c
Continuing.

Breakpoint 1, main () at main.c:20
20                      b[i]=a[i];
(gdb) info registers rax,rbx,rcx,rdx
Invalid register `rax,rbx,rcx,rdx'
(gdb) info registers rax rbx rcx rdx
rax            0x2      2
rbx            0x0      0
rcx            0x0      0
rdx            0x1      1
(gdb) c
Continuing.

Breakpoint 1, main () at main.c:20
20                      b[i]=a[i];
(gdb) info registers rax rbx rcx rdx
rax            0x3      3
rbx            0x0      0
rcx            0x5      5
rdx            0x6      6
(gdb) c
Continuing.

Breakpoint 1, main () at main.c:20
20                      b[i]=a[i];
(gdb) info registers rax rbx rcx rdx
rax            0x4      4
rbx            0x0      0
rcx            0x4      4
rdx            0x5      5
(gdb) c
Continuing.
```

中间可以用 p 指令打印变量的值，比如 p $rax 打印寄存器%rax 的值；或者用 x/s 0x40076a 显示内存。

```
Breakpoint 1, main () at main.c:20
20                      b[i]=a[i];
(gdb) info registers rax rbx rcx rdx
rax            0x5      5
rbx            0x0      0
rcx            0x1      1
rdx            0x2      2
(gdb) c
Continuing.

Breakpoint 1, main () at main.c:20
20                      b[i]=a[i];
(gdb) info registers rax rbx rcx rdx
rax            0x6      6
rbx            0x0      0
rcx            0x9      9
rdx            0xa      10
(gdb) c
Continuing.

Breakpoint 1, main () at main.c:20
20                      b[i]=a[i];
(gdb) info registers rax rbx rcx rdx
rax            0x7      7
rbx            0x0      0
rcx            0x7      7
rdx            0x8      8
(gdb) c
Continuing.

Breakpoint 1, main () at main.c:20
20                      b[i]=a[i];
(gdb) info registers rax rbx rcx rdx
rax            0x8      8
rbx            0x0      0
rcx            0x0      0
rdx            0x1      1
```

再次查看局部变量

```
(gdb) info locals
a = {8, 5, 1, 6, 5, 2, 10, 8, 1, 0}
i = 8
b = {8, 5, 1, 6, 5, 2, 10, 8, -8128, 32767}
randValue = 1
flag = 0 '\000'
```

继续到达下一个断点处因为后面 case 语句部分会进入函数查看，所以此处略过

```
(gdb) c
Continuing.

Breakpoint 1, main () at main.c:20
20                      b[i]=a[i];
(gdb) info registers rax rbx rcx rdx
rax            0x9      9
rbx            0x0      0
rcx            0x7      7
rdx            0x8      8
(gdb) c
Continuing.

Breakpoint 4, printArray (s=0x7fffffffdf00, n=10, str=0x400ac8 "\nrandom array: ")
    at printArray.c:3
3               printf("%s",str);
```

finish 跳过函数

```
(gdb) finish
Run till exit from #0  printArray (s=0x7fffffffdf00, n=10,
    str=0x400ac8 "\nrandom array: ") at printArray.c:6
random array:     8    5    1    6    5    2   10    8    1    8
main () at main.c:25
25              bool flag=true;
```

到达选择界面，选择 2，求和，stepi 单步执行进入函数 add()

```
1.selection sort
2.sum
3.print result
4.exit
choose a number: 2
```

```
Breakpoint 2, add (s=0x7fffffffdf00, n=10) at add.c:4
4               int result=0;
(gdb) stepi
6               for(i=0;i!=n;i++){
(gdb) stepi
0x00000000004009a1      6               for(i=0;i!=n;i++){
```

选择 1，选择排序，stepi 单步执行进入函数 selectionSort()

```
Breakpoint 1, selectionSort (s=0x7fffffffdf00, n=10) at selectionSort.c:6
6               for(i=0;i!=n-1;i++){
(gdb) stepi
0x00000000004008b5      6               for(i=0;i!=n-1;i++){
```

选择 3，输出数组，stepi 单步执行进入函数 printArray()

```
Breakpoint 3, printArray (s=0x7fffffffdf30, n=10, str=0x400b30 "\noriginal array:\t")
    at printArray.c:3
3               printf("%s",str);
(gdb) stepi
```

```
__printf (format=0x400b8e "%s") at printf.c:28
28      printf.c: 没有那个文件或目录.
```

bt 命令打印堆栈，可以看到当前正在 printf()函数中

```
(gdb) bt
#0  __printf (format=0x400b8e "%s") at printf.c:28
#1  0x00000000004009f6 in printArray (s=0x7fffffffdf30, n=10,
    str=0x400b30 "\noriginal array:\t") at printArray.c:3
#2  0x000000000040085a in main () at main.c:44
```

delete 命令可以用于删除断点

```
(gdb) delete
删除所有断点吗？ (y or n) y
(gdb) info break
No breakpoints or watchpoints.
```

q 命令退出 gdb

```
(gdb) q
qjw@qjw-VirtualBox:~/hw2$
```

五、问题分析

1) **分析同一个源程序在不同机器上生成的可执行目标代码是否相同。提示：从多个方面（如 ISA、OS 和编译器）来分析。**

不相同。ISA 规定了汇编语言形式和二进制机器码的格式，不同机器上的 C 语言程序在转换为可执行程序过程中，都是在不同 ISA 规定下的；不同操作系统在编译汇编链接时的 ELF 表不同;不同的编译器在数据对齐、库函数源文件、符号表的创建和解析方面不完全相同

2) **你能在可执行目标文件中找出函数 printf() 对应的机器代码段吗？能的话，请标示出来。**

不能，因为 printf()在动态链接库 libc.so 中，程序在链接时并不把这个库的函数的机器码链接到可执行程序中，而是在执行程序的时候才加载进来，所以不能找到。在单步执行时没有那个文件或目录也可以表明。

3）为什么源程序文件的内容和可执行目标文件的内容完全不同？

因为源文件是一种用高级语言编写的文件，它的主要作用是便于程序员理解代码的逻辑关系；而可执行文件是一种机器识别的文件。现在的计算机仍旧是二进制运算，无法识别高级语言，而编译器的作用就是将高级语言转换成机器语言，所以内容完全不同。

六、实验体会

这次实验前面补全程序部分比较简单，顺便复习了排序的几种算法。通过练习两种将 C 语言程序文件转化成生成可执行文件的方法，更好地明白了程序的预处理、编译、汇编、链接的过程。通过对比.c 文件和反汇编得到的汇编代码，认识到不同层次程序代码的不同，通过问题的思考发现了不同机器上生成的可执行文件不同的事实及其影响因素，明白了静态链接库和动态链接库的不同之处。后面又熟悉了 gcc 调试器的几种常用命令。总体来说没有遇到什么大问题，就是加深了对于课本第三章、第四章的理解。