河南工业大学 Linux 基础与应用 实验报告

专业班级：软件 1305 班 学号：201316920311 姓名：田劲锋 指导老师：赵晨阳 评分：_____

实验题目：Linux下C语言编程

实验目的：（1）理解GCC；（2）理解程序维护工具make。

实验内容：

1. 简单的编写一个C语言程序，使用GCC编译该程序，要求按照以下步骤进行：预处理、编译、汇编、连接，并查看中间结果。
2. 编写两个C语言程序，其中一个程序的编译需要另一个程序的编译完成，试写出一个makefile文件，并测试。

实验步骤：

本次实验在 Mac OS X Yosemite 上运行，使用的 GCC 是 Apple LLVM 版本 6.1.0 (clang-602.0.49)（基于 LLVM 3.6.0svn）。

1. 这里我们编写了一个简单的相加程序。

Listing 1: aplusb.c

```c
#include <stdio.h>

int main(int argc, char* argv[])
{
    int a, b;
    scanf("%d%d", &a, &b);
    int c = a + b;
    printf("%d\n", c);
    return 0;
}
```

预处理文件过大这里不在列出，但是可以看一下编译出来的汇编代码（x64汇编）。

Listing 2: aplusb.s

```
        .section        __TEXT,__text,regular,pure_instructions
        .macosx_version_min 10, 10
        .globl  _main
        .align  4, 0x90
_main:                                  ## @main
        .cfi_startproc
## BB#0:
        pushq   %rbp
Ltmp0:
        .cfi_def_cfa_offset 16
Ltmp1:
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
Ltmp2:
        .cfi_def_cfa_register %rbp
        subq    $48, %rsp
        leaq    L_.str(%rip), %rax
        leaq    -20(%rbp), %rcx
        leaq    -24(%rbp), %rdx
        movl    $0, -4(%rbp)
```

```
21          movl    %edi, -8(%rbp)
22          movq    %rsi, -16(%rbp)
23          movq    %rax, %rdi
24          movq    %rcx, %rsi
25          movb    $0, %al
26          callq   _scanf
27          leaq    L_.str1(%rip), %rdi
28          movl    -20(%rbp), %r8d
29          addl    -24(%rbp), %r8d
30          movl    %r8d, -28(%rbp)
31          movl    -28(%rbp), %esi
32          movl    %eax, -32(%rbp)         ## 4-byte Spill
33          movb    $0, %al
34          callq   _printf
35          xorl    %esi, %esi
36          movl    %eax, -36(%rbp)         ## 4-byte Spill
37          movl    %esi, %eax
38          addq    $48, %rsp
39          popq    %rbp
40          retq
41          .cfi_endproc
42
43          .section        __TEXT,__cstring,cstring_literals
44  L_.str:                                 ## @.str
45          .asciz  "%d%d"
46
47  L_.str1:                                ## @.str1
48          .asciz  "%d\n"
49
50
51  .subsections_via_symbols
```

在Ltmp2段，进行了一系列弹栈操作之后，将操作数压入栈中，调用_scanf获取两个值到-20(%rbp)和-24(%rbp)中，相加后再压栈调用_printf输出出来。可以看到x86-64汇编使用了很多额外的寄存器。

如下，预处理、编译、汇编、连接并运行该程序：

```
$ ls
aplusb.c
$ gcc -E aplusb.c -o aplusb.i
$ ls
aplusb.c aplusb.i
$ gcc -S aplusb.c -o aplusb.s
$ ls
aplusb.c aplusb.i aplusb.s
$ gcc -c aplusb.s -o aplusb.o
$ ls
aplusb.c aplusb.i aplusb.o aplusb.s
$ gcc aplusb.o -o aplusb
$ ls
aplusb   aplusb.c aplusb.i aplusb.o aplusb.s
```

```
$ ./aplusb
12 34
46
```

2. 这次我们考虑写一个计算数组元素和的程序。首先是头文件：

Listing 3: vector.h

```
#ifndef _VECTOR_H_
#define _VECTOR_H_

int int_sum(int a[], int len);

#endif
```

实现：

Listing 4: vector.c

```
#include "vector.h"

int int_sum(int a[], int len)
{
    int sum = 0;
    int i;
    for (i = 0; i < len; i++) {
        sum += a[i];
    }
    return sum;
}
```

在主程序中声明进来vector.h即可：

Listing 5: test.c

```
#include <stdio.h>
#include "vector.h"

int main(int argc, char* argv[])
{
    int n = 10;
    int a[10];
    int i;

    for (i = 0; i < n; i++) {
        a[i] = i;
    }

    int s = int_sum(a, n);
    printf("%d\n", s);

    return 0;
}
```

为了编译这个程序，我们编写了一个Makefile。顺便把上一题也写进去吧：

Listing 6: Makefile

```
OBJS1 = aplusb.o
PROB1 = aplusb

OBJS2 = vector.o test.o
PROB2 = test

default:
	$(MAKE) -r all

all:
	$(MAKE) -r $(PROB1)
	$(MAKE) -r $(PROB2)

%.i: %.c
	$(CC) -E $< -o $@

%.s: %.i
	$(CC) -S $< -o $@

%.o: %.s
	$(CC) -c $< -o $@

$(PROB1): $(OBJS1)
	$(CC) $^ -o $@

$(PROB2): $(OBJS2)
	$(CC) $^ -o $@

clean:
	-$(RM) *.i
	-$(RM) *.s
	-$(RM) *.o
	-$(RM) $(PROB1)
	-$(RM) $(PROB2)
```

该Makefile用到了一些通配符，也写了清理操作。

如图1展示了这两个程序的编译执行情况。

实验体会：

在UNIX/Linux上进行C语言编程是一件令人享受的事情。不管是gcc还是clang编译器都是非常优秀和好用的，这也为编译和调试提供了相当便利。从源代码到可执行文件的过程，在这个试验中也很好地复习到了。

不过说起来，OS X自带的llvm clang编译器和Linux自带的gcc还是有不少区别啊，而且编译出来的汇编代码也有微妙的区别呢。

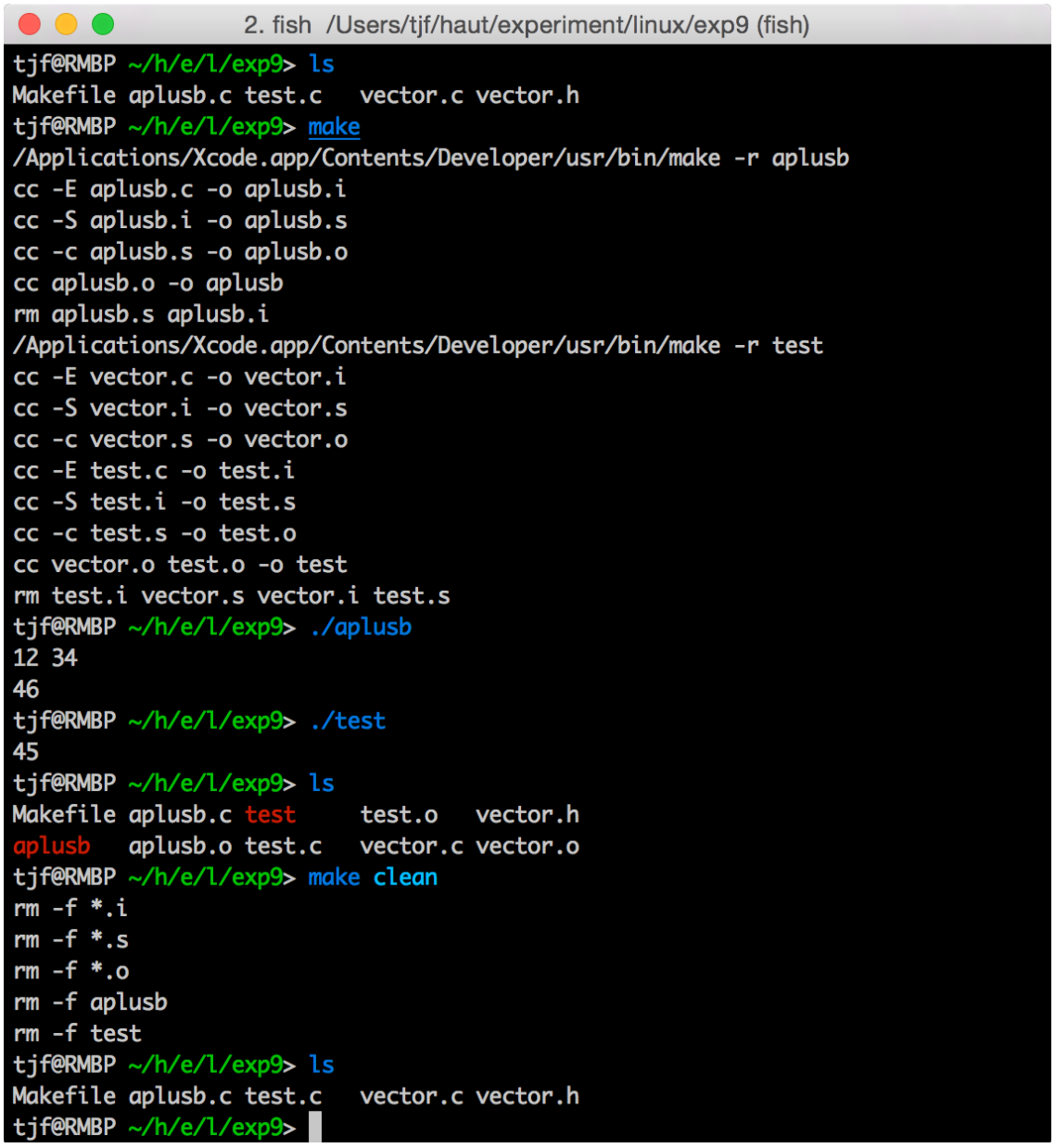
2. fish /Users/tjf/haut/experiment/linux/exp9 (fish)
tjf@RMBP ~/h/e/l/exp9> ls
Makefile aplusb.c test.c vector.c vector.h
tjf@RMBP ~/h/e/l/exp9> make
/Applications/Xcode.app/Contents/Developer/usr/bin/make -r aplusb
cc -E aplusb.c -o aplusb.i
cc -S aplusb.i -o aplusb.s
cc -c aplusb.s -o aplusb.o
cc aplusb.o -o aplusb
rm aplusb.s aplusb.i
/Applications/Xcode.app/Contents/Developer/usr/bin/make -r test
cc -E vector.c -o vector.i
cc -S vector.i -o vector.s
cc -c vector.s -o vector.o
cc -E test.c -o test.i
cc -S test.i -o test.s
cc -c test.s -o test.o
cc vector.o test.o -o test
rm test.i vector.s vector.i test.s
tjf@RMBP ~/h/e/l/exp9> ./aplusb
12 34
46
tjf@RMBP ~/h/e/l/exp9> ./test
45
tjf@RMBP ~/h/e/l/exp9> ls
Makefile aplusb.c test test.o vector.h
aplusb aplusb.o test.c vector.c vector.o
tjf@RMBP ~/h/e/l/exp9> make clean
rm -f *.i
rm -f *.s
rm -f *.o
rm -f aplusb
rm -f test
tjf@RMBP ~/h/e/l/exp9> ls
Makefile aplusb.c test.c vector.c vector.h
tjf@RMBP ~/h/e/l/exp9>

图 1: 编译执行情况