Introduction to Computer System

PA 1

实验报告

姓名: 郑奘巍

学号: 171860658

邮箱: zzw@smail.nju.edu.cn

院系: 计算机科学与技术系

2018 年 9 月 24 日

第一部分　实验进度

我完成了所有内容。

第二部分　必答题

问题 0.1 假设你现在需要了解一个叫 *selector* 的概念, 请通过 *i386* 手册的目录确定你需要阅读手册中的哪些地方.

解答 我需要阅读 *CHAPTER 5, 5.1.3 Selectors* 的内容。

问题 0.2 理解基础设施

解答 根据假设，在没有实现简易调试器的情况下，每个 *bug* 需要 *20*30s=600s=10min* 时间进行排除。在 *500* 次编译中，调试的次数有 *500*90%=450* 次。假定每次调试消除一个 *bug*，则在调试上花费的时间为 *450*10min=4500min=75h*。

通过简易调试器，我们的效率提高了 $\frac{2}{3}$，则节省了 *50* 小时的时间。

如此看来，没有基础设施，我们可能会因为 *PA* 而挂科。

问题 0.3 查阅 *i386* 手册, 把需要阅读的范围写到你的实验报告里面

(a) EFLAGS 寄存器中的 *CF* 位是什么意思? *(b) ModR/M* 字节是什么? *(c) mov* 指令的具体格式是怎么样的?

解答 *(a)* 根据目录，首先阅读 *P33-P34* 关于 *EFLAGS* 和 *Status Flag* 的简介。然后在 *Appendix C, P419* 阅读 *CF* 位的解释。

(b) P241-P245 Instruction Format, ModR/M and SIB bytes

(c) P345-P347 两个 *MOV* 指令的具体格式

问题 0.4 *shell* 命令完成 *PA1* 的内容之后, *nemu/*目录下的所有*.c* 和*.h* 和文件总共有多少行代码? 你是使用什么命令得到这个结果的? 和框架代码相比, 你在 *PA1* 中编写了多少行代码? 除去空行之外, *nemu/*目录下的所有*.c* 和*.h* 文件总共有多少行代码?

解答 共 *4390* 行，使用命令：*find -regex ".*\.\(h\|c\)" | xargs cat | wc -l*

利用 *git checkout master* 回到框架代码状态，得到共 *3827* 行。相比较，我编写了 *563* 行的代码。

除去空行之后 *(*命令：*find -regex ".*\.\(h\|c\)" | xargs cat |grep -v $ | wc -l)*，框架代码共 *3109* 行，完成后 *3585* 行，编写了 *476* 行代码。

将该命令写入 *Makefile* 中，在*.PHONY* 后增加 *count*。注意 *$* 符号在 *Makefile* 中要用 *$$* 替代。

问题 0.5 使用 *man(vim?)* 打开工程目录下的 *Makefile* 文件，你会在 *CFLAGS* 变量中看到 *gcc* 的一些编译选项．请解释 *gcc* 中的*-Wall* 和*-Werror* 有什么作用？为什么要使用*-Wall* 和*-Werror?*

解答 *-Wall:* 打开 *gcc* 所有警告

-Werror: 将所有警告都当成错误

使用*-Wall* 和*-Werror* 是为了将潜在的 *fault* 直接转换为 *failure*。警告即意味着代码中的可能出错点或未定义行为，这些行为不一定会立即导致 *failure*，但可能在以后显现出来。当下处理掉这些可能错误的地方，提高了代码的可靠性。

第三部分　遇到的问题

1　配置 X Server 失败

在 PA1 一开始，配置 X Server 就给我了一个下马威。我在 Mac 电脑上使用 docker，利用 XQuartz2.7.11 死活无法运行马里奥。因为怕对之后的 PA 产生影响，我花了一天半的时间调这个问题，遇到了各种解决方案。如升级 XQuartz 到测试版本修复一个 bug，使用 nvidia-docker 等。但最后还是无功而返，只能跑路虚拟机。

虽然最后还是没有解决这个问题，但在 STFW 中还是了解了很多相关的知识，并一再看到成功的希望。

2 错误输入检测

在编写基础设施代码时，我发现不仅要完成基础功能，还要保证这些基础功能不会因错误的输入而导致运行错误。这是为了防止在以后 Debug 到一半时不小心手误使得功亏一篑。

检测错误的输入要考虑的情况还是蛮多的。一些简单的错误代码框架中已经处理了。还有如下错误：1. 参数类型错误 2. 参数个数错误 3. 取地址越界 4. watchpoint 中一原有值表达式计算中除 0

此外，为了模拟 gdb（也为了更好的单步执行），我还实现了不输入命令单按回车键可以输入上一条命令。

3 git commit message 编写

一开始就觉得我的提交信息写的很乱。在网上果然找到了有一些规范可以参考。于是我增加了.gitmessage 作为 git 提交信息的模板以及 icsgit 作为 git commit -a 的 alias。在书写时关于 scope，50/70 format 还不能完全掌握。但有了规范的提交信息，相信对以后的管理大有好处。

git 模板：

```
1       # header: <type>(<scope>): <subject>
2       # - type: feat, fix, docs, style, refactor, test, chore
3       # - scope: can be empty (eg. if the change is a global or difficult
    to assign to a single component)
4       # - subject: start with verb (such as 'change'), 50-character line
5       #
6       # body: 72-character wrapped. This should answer:
7       # * Why was this change necessary?
8       # * How does it address the problem?
9       # * Are there any side effects?
10      #
11      # footer:
12      # - Include a link to the ticket, if any.
13      # - BREAKING CHANGE
14      #
15      # If this commit is to revert before commit:
```

```
16          # revert: <before header>
17          # This revert commit <commit id>
18
```