

XOS 项目说明文档



1552648 陈鹤丹

1552633 韦尧

目录

一、开发环境	3
二、OS 介绍	3
1. 用户应用程序	3
(1) 科学计算器	3
(2) figlet.....	4
(3) 拼图游戏.....	5
(4) 实用万年历	6
2. 系统应用程序	6
(1) 进程状态表	6
(2) 文件列表.....	7
三、算法改进	7
四、总结	9

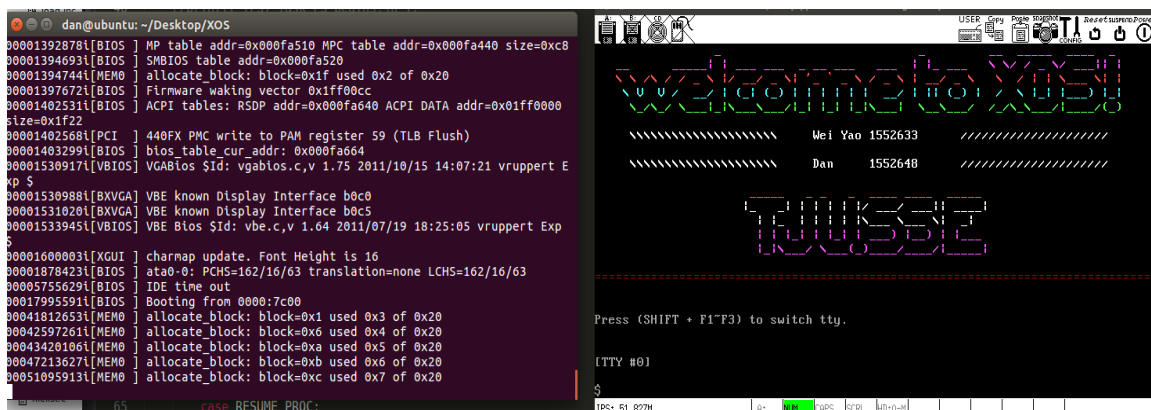
一、开发环境

项目采用 VMware 下 32 位 Ubuntu16.04 LTS 版本虚拟机进行开发，Bochs x86 Emulator 2.6 作为 XOS 运行虚拟机。如果需要运行 XOS 或者进行 makefile，建议使用 32 位系统进行，如果使用 64 位系统可能碰到意料之外的问题。

二、OS 介绍

XOS 是一款基于 Orange's 的操作系统，由同济大学软件学院陈鹤丹、韦尧开发。

在 Ubuntu 系统下，在 XOS 根目录打开终端，输入 bochs 指令，然后在终端中输入 c 并按回车，即可开始运行操作系统。



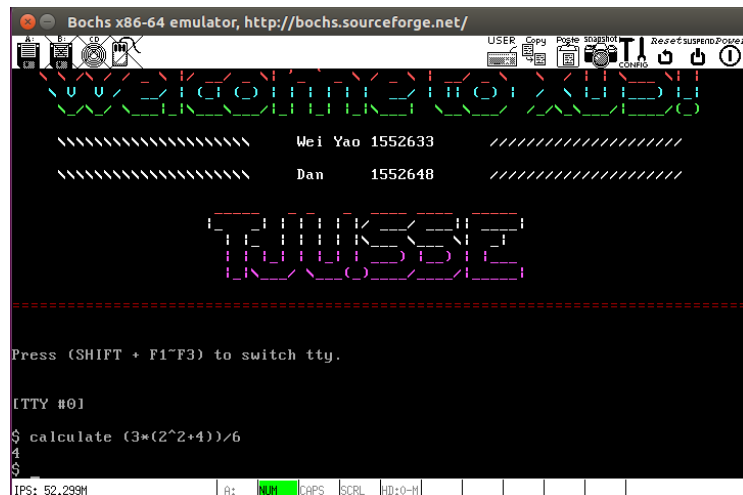
如图所示为 XOS 的开机界面，按 shift+F1~F3 在 3 个 TTY 之间进行切换。在每个 TTY 中均可以输入指令。注意：需要等 OS 输出符号 '\$' 之后再进行键盘操作，否则会出现错误。

1. 用户应用程序

(1) 科学计算器

经过我们对各种学长代码的阅读，发现一些学长的 OS 有计算器的功能，但是功能都比较简单，只能先输入一个数字，然后选择运算符，然后输入第二个数字进行运算。这让我们看到了很大的开发潜力，于是开发了这个科学计算器。

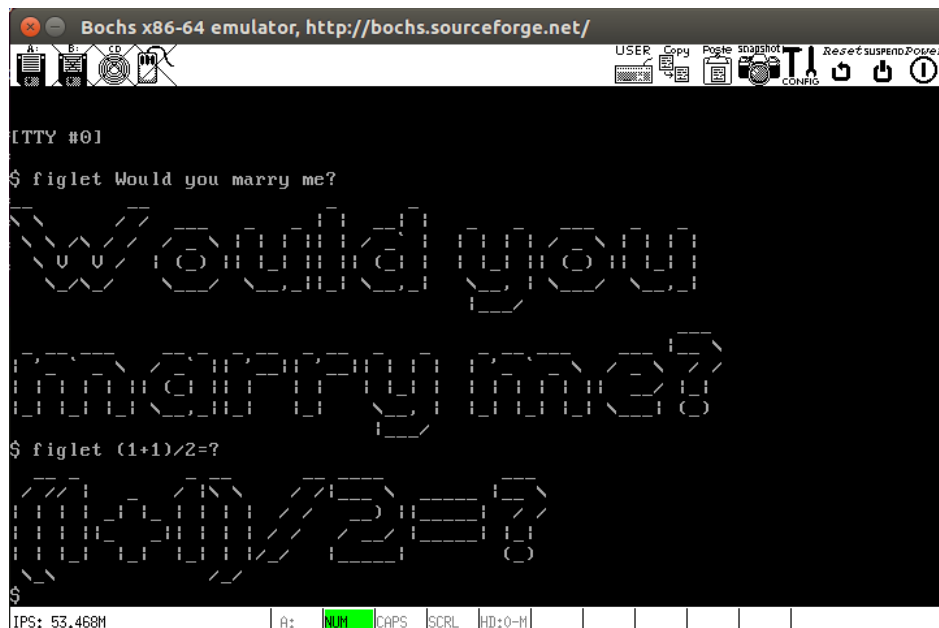
如图所示，只需要在命令行输入 `calculate + 算式`，即可直接获得运算结果。相比之前学长的计算器，更加直观，功能更加强大，支持四则运算、括号、乘方运算等功能。不过很可惜的是，XOS 并不支持浮点运算，因此运算只能在整数之间进行。当需要运算的内容较多时，可以通过切换 TTY 的方式在几个终端同时运算，不同 TTY 会分别保留各自的运算过程和结果，非常方便。



计算器算法采用将中缀表达式转换为逆波兰表达式再进行运算的方式，由于大一做过类似的项目，所以移植起来并没有遇到太多阻力，只是由于无法使用库函数，某些函数只能自己在本地实现，比如判断输入是否为数字的函数，就是用宏 `#define isdigit(c) ((c) >= '0' && (c) <= '9')` 来代替的。又例如由于无法使用 `math` 库，乘方函数 `pow` 也是在本地实现的。

(2) figlet

Figlet 是 Linux 系统下的一款非常有趣的程序，可以将输入的文字自动转换为字符画输出。XOS 的开机界面就是利用 figlet 进行设计的。为了致敬这一有趣的应用程序，我们决定将其移植到我们自己的 OS 上。



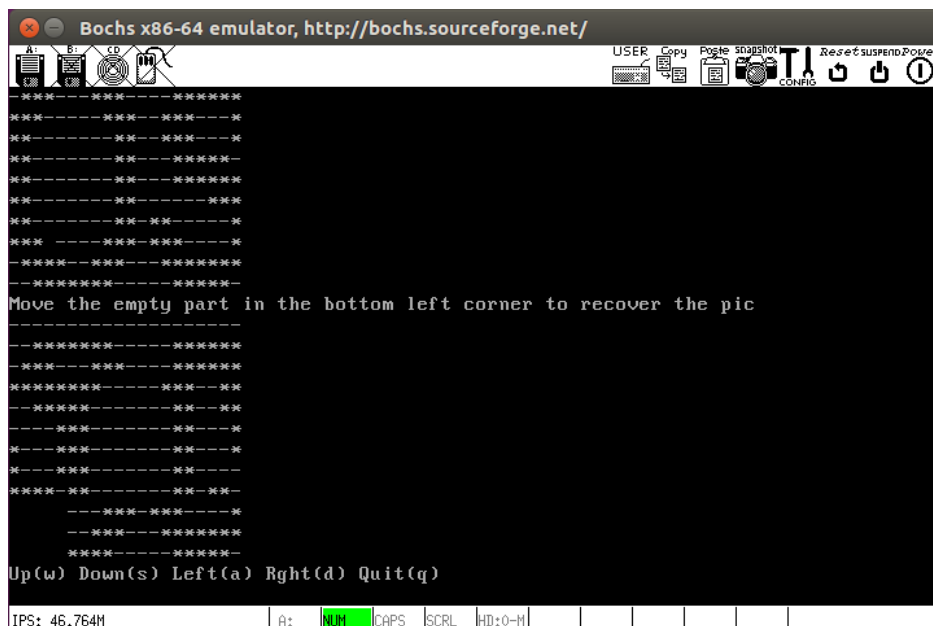
如图所示，移植版的 figlet 支持你能在键盘上直接打出来的任何字符，包括大小写英文字母、阿拉伯数字、!@#\$%^&*(){}[]<>等符号。你甚至可以尝试将表情 ^_^ :) 等输出，效果拔群。

如果您想在 Ubuntu 中使用 figlet，只需要在终端利用命令 `sudo apt-get install figlet` 安装 figlet，即可使用这一程序自由输出您的字符画了。使用方法与在 XOS 中相同，在终端输入 `figlet + 需要转换的字符串` 即可。

移植 figlet 确实耗费了我们不少的精力。首先，需要将键盘上近 100 个不同字符的字符画按行拆分，并写入 XOS 中的 figlet 程序，保存为本地字库。为了加快这一进度，我们甚至写了一个半自动的 C++ 程序，用于读取字符画并按行进行拆分。尽管如此，这一过程还是耗费了我们大量时间。第二个问题是输出。如果单输出一个字母是非常容易的，只需要按行循环输出字库中对应的字符即可。可是当输入的是一串字符时，就要把这些字符按行连起来输出，同时还需要考虑换行问题，因为如果不换行，当输出达到屏幕边界（bochs 中屏幕宽度为 80 个字符）时，输出结果便会变形，无法正常显示。（figlet 代码有 700 行之多）

（3）拼图游戏

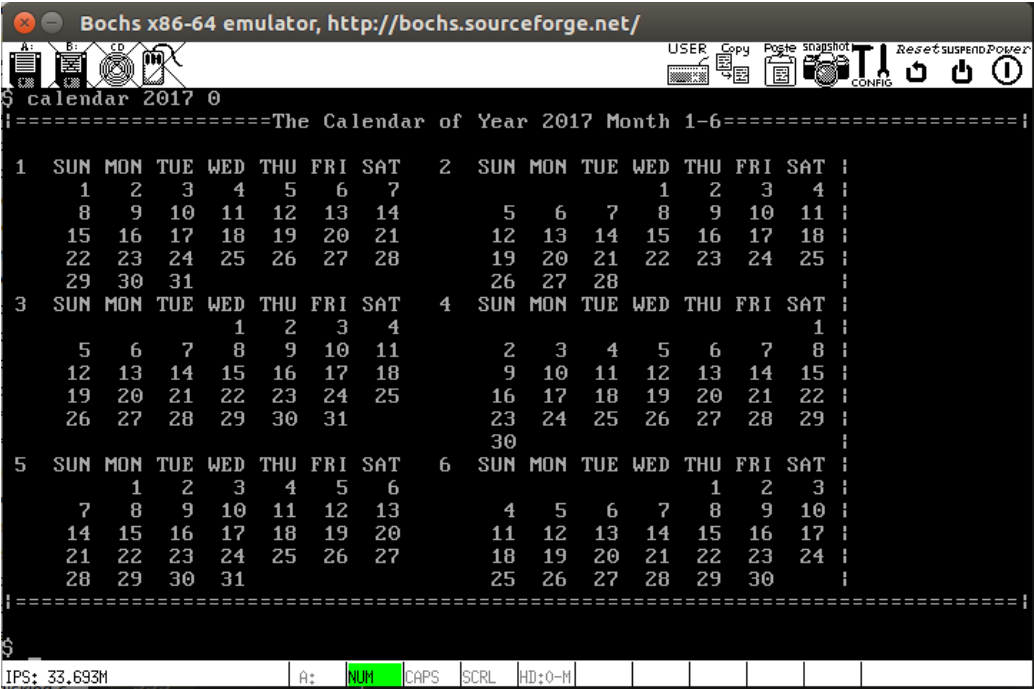
在命令行键入 `puzzle` 并按回车，即可开始拼图小游戏。游戏开始会给出拼图复原的样子和一个被打乱的拼图，用户需要通过输入 `wasd` 控制拼图中空缺位置的移动，从而最终复原拼图。



由于无法使用随机数函数，也无法使用 `malloc` 函数，在移植游戏时做了部分阉割，拼图尺寸从可变动为固定（因为无法用 `malloc` 函数动态分配大小），拼图打乱算法由随机改为固定（因为无法用 `srand` 函数产生随机种子）。

(4) 实用万年历

由 08 级赵启云学长的万年历程序移植而来。在学长的万年历中，由于 bochs 屏幕尺寸过小，无法完整显示一年 12 个月的万年历。经过我们的改进，新增一个参数，用来选择显示上半年或者下半年的日历，不会出现显示不全导致某些月份看不到的问题。



使用方法为：在命令行键入 calendar + 年份 + 1/0，其中 0 表示上半年，1 表示下半年。

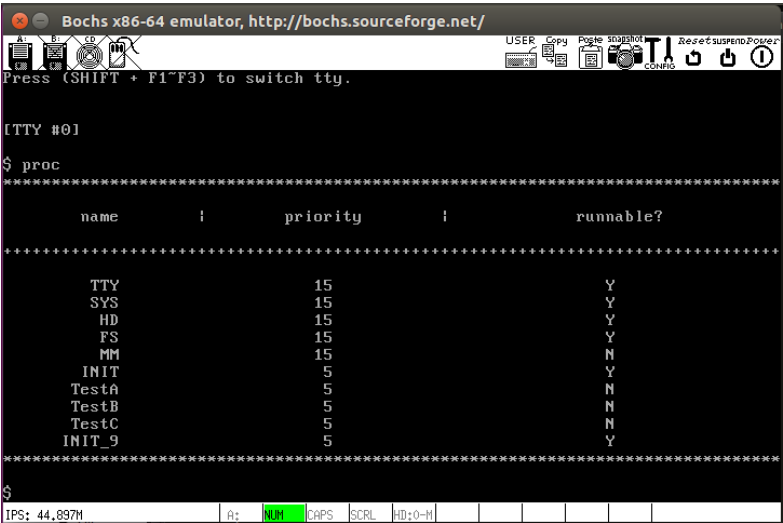
除了新增参数和微调显示方法，并未对学长的代码做过大更改，万年历算法保留不动。

2.系统应用程序

(1) 进程状态表

在命令行键入 proc，即可查看当前所有进程的状态信息，包括名称、优先级、是否活动，如右图所示。

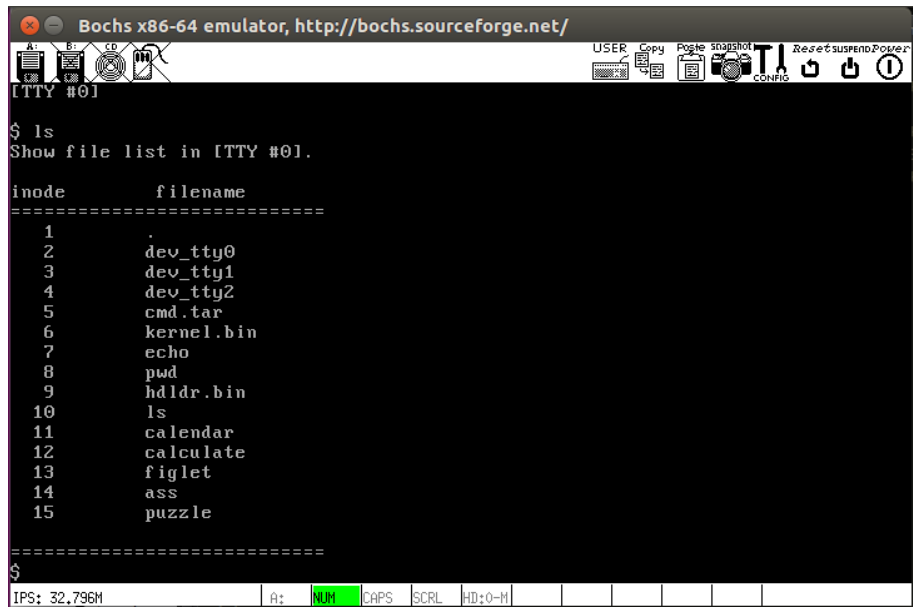
实现方法为，当判断命令为 proc 后，调用 showProc 函数，取出



进程表 proc_table 中保存的进程信息。由于 proc_table 已经在 global.h 中声明，所以直接使用即可，无需做其他额外操作。

(2) 文件列表

在命令行中键入 ls 并回车即可查看当前系统中的文件列表。利用用户级系统调用

A screenshot of a Bochs x86-64 emulator window. The title bar reads "Bochs x86-64 emulator, http://bochs.sourceforge.net/". The window contains a terminal window titled "[TTY #01]". Inside the terminal, the command "\$ ls" has been entered, and the output is a file list. The output is formatted as follows:

```
inode      filename
=====
1          .
2          dev_tty0
3          dev_tty1
4          dev_tty2
5          cmd.tar
6          kernel.bin
7          echo
8          pwd
9          hdlr.bin
10         ls
11         calendar
12         calculate
13         figlet
14         ass
15         puzzle
```

The terminal prompt "\$" is visible at the bottom. The Bochs window also shows various icons and a status bar at the bottom with information like "IPS: 32.796M".

PUBLIC int sendrec(int function, int src_dest, MESSAGE* p_msg);来实现。

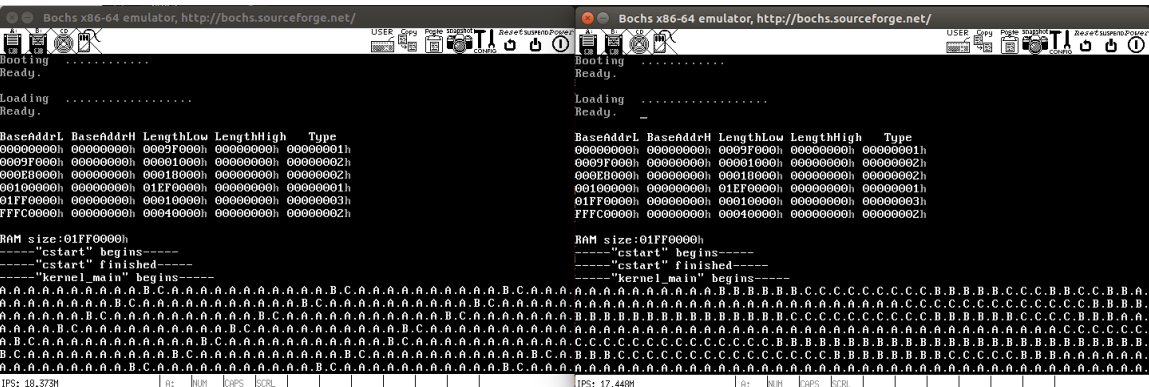
向文件系统进程发送 FILELIST 消息，进程收到消息后调用 do_filelist()函数来将文件信息显示在屏幕上。

三、算法改进

针对 Oranges 进程调度算法，我们进行了改进。原本 Oranges 采用的是优先级调度算法，优先级越高的进程占用时钟周期数越多。但是这样的算法有一定的弊端：对于运行时间短但是优先级低的进程，可能很久得不到运行，导致系统效率低下。

为了解决这一问题，我们将算法改为高响应比优先调度算法，为每个进程引入动态优先权，并使作业的优先级随着等待时间的增加而提高，这样的算法有三大特点：①如果进程的等待时间相同，那么进程的运行时间越短，其优先级越高，因此该算法有利于短作业。②当进程的运行时间相同时，优先级取决于其等待时间，等待时间越长，优先级越高，因此该算法类似先来先服务。③对于运行时间长的进程，优先级随着等待时间的增加而提高，当其等待时间足够长，优先级便会升到很高，从而获得运行机会。

简言之，该算法既照顾了短进程，又考虑了进程到达的先后次序，不会使某进程长期得不到服务，因此，它实现了一种较好的折衷。



如图所示，左侧为未改进时的进程运行情况，右侧为改进后的进程运行情况，两个系统均采用ABC3个进程，进程优先级比为20：1：1。可以清楚的看到，未改进之前BC获得的运行机会很少，改进后BC获得的运行机会大大增加。当然，这种模拟和现实中的差别在于，现实中BC进程运行一段时间便停止了，不会输出满屏幕的BC，只是运行时间会提前；而在我们的模拟中，由于进程运行完之后会重置 ticks，所以会无限输出，看起来BC的运行时间太长了，实则不然。另一方面在Oranges中进程的运行时间(ticks)和优先级(priority)是同一个值，所以没有模拟运行时间低而优先级高/运行时间高而优先级低的情况，不过其实也都大同小异，通过我们的图片示例可以清晰的看到结果的不同，证明算法确实起作用了。

具体实现方式为，在clock_handler函数中，将未在运行的进程的ticks不断增加（增加速率可调，这将影响优先级变化速率），并且在某个进程运行结束后重置其ticks为priority的值，然后重新进行调度，运行schedule()函数。在schedule()函数中，根据ticks/priority的值计算出动态变化的新的优先级，并选出动态优先级最高的进程运行。值得注意的一点是，priority越低的进程，其动态优先级增加速率越快。例如，两个进程priority分别为50和100，经过50ticks之后，第一个进程的动态优先级为 $(50+50)/50=2$ ；第二个进程的动态优先级则为 $(100+50)/50=1.5$ 。这也说明该算法更加照顾优先级低而实际运行时间短的进程，从而保障各个进程的顺利运行。

编写算法过程中有一个小插曲——动态优先级的计算问题。由于XOS不支持浮点运算，通过直接计算比值获得动态优先级不是一个可行的方法。于是，我们采用分别保存分子分母，比较时通过交叉相乘进行比较的办法，巧妙绕过了这一限制。

注意：进程调度相关代码并未整合到XOS中，为了更加直观的显示结果，我们把它放到了进程调度文件夹中，请直接运行这个文件夹下的系统即可看到调

度过程。为了对比方便，在进程调度（原版）文件夹下有原版的调度算法，两个文件夹中进程的相关参数完全一致，可以方便看出改进后的算法与原版的区别。

四、总结

XOS 的开发基于《自己动手写操作系统》一书中的 Oranges 操作系统，并在其基础上进行了用户和系统级的改进，增加和优化了一些功能。

通过这次开发，我们不仅加深了对操作系统运行机制的理解：从 boot->loader->kernel 的整个过程，更深入探究了进程调度算法等具体问题，对操作系统的工作方式有了更直观的认识。略有遗憾的是由于之前没有学过汇编语言，对于书中保护模式一节的理解仍然不够到位，对开发过程造成了一定的阻碍。

尽管过程并不是一帆风顺，但是至少我们自己的操作系统 XOS 可以正常运行，这还是很令人开心的。在此要感谢《自己动手写操作系统》的作者于渊和同济大学软件学院 06、08 级同学的源码，为我们自己的开发提供了很多灵感。