**0 序章：Powerint DOS 发展史，以及各内核版本被赋予的意义**

在2022年1月1日之前，Powerint DOS 的开发者仅有zhouzhihao一人，并且是实模式下的操作系统，在这之后，Powerint DOS便正式进入32位保护模式世界。并命名为Powerint DOS 386

**0.1为什么叫Powerint DOS** （待定 纯我（min0911）自己想的）

Power ：力量的英文

Int ：中断的缩写

由于目前这个系统的系统调用是通过中断实现的，所以，Powerint DOS的意思就是我们希望这个系统拥有强大的系统调用。

**0.2 发展史**

Powerint DOS 386（以下简称pdos x86）的前身是SimpleOS 一个十分简易的OS，由于不属于pdos x86 的正统系列，就不多说，简单实现了屏幕打印的一些功能（你们现在还可以在内核中看到他的身影！见screen.c）

最早的pdos x86版本是0.1a，一个，非常简陋的内核，并且与pdos x86的后续版本都不关联，这里也就不多说了。

0.4：基础输入输出

接着便是 0.4x，这里有个小贴士，pdos x86 的版本只有 （0.xa，0.xb，0.xc）三个，如果有特殊（例如0.4x）就说明这是在为后面的东西打基础。至于啥基础，也是底层函数那一类的，是什么我也不太清楚了

大概1月25号，我们将版本号改为了0.5a，开始了新一阶段的开发，这一阶段的开发成功为后面的顺利开发打下了坚实的基础，例如多任务，以及内存分页啥的。

0.5b，这时候就已经支持了多任务了，然后还有很多小程序，大家可以自己去把玩。

0.5c，支持内存分页，不连续物理内存映射成连续虚拟内存；读写软盘；VBE驱动，高分辨率色彩图形模式；以及大大小小很多改进（多任务API啥的）。

0.6a，一个新的阶段。我们现在的开发版本，打算写好PCI驱动，以及简单的网卡（AMD PCNET）驱动；分割内核文件，宏内核->微内核；完善进程间通信。

**0.3 序章中的环境搭建**

显然，要想来调试我们的内核，就必须要搭建好环境，大部分工具我们已经在z\_tools文件夹提供了，大家只要安装两个东西（其实只要安装一个就可以成功编译了，但是我这里考虑到大家还是需要调试这个内核，所以我将会要求大家安装两个东西）一个是GCC，这个我想大家应该无话可说，这是我们选用的现代化C编译器（推荐TDM-GCC10，不然可能会导致编译不通过），那么在哪里找呢？我这里推荐大家直接安装一个IDE: RedPanda Dev C++,这个IDE会附带TDM-GCC10，所以就不用劳烦大家自己找资源安装了，可以直接使用这个IDE安装，这个IDE还算好找，所以就不贴出下载地址了。

其实这时候你们已经可以编译了，只不过不能进行愉快的调试（简单来说就是调试很麻烦），我这里再叫大家安装一个软件：qemu虚拟机，这个软件也是直接在网上搜索就能下载到的。

**0.3.1 怎么启动？**（怎么进行调试内核）

您可以在命令提示符中，输入make img\_run（只编译内核并且启动qemu虚拟机）或者make full\_run（编译内核及其的应用程序并且启动qemu虚拟机）。运行完成之后您可能会发现，qemu窗口陷入了已暂停状态，那么我们要怎么让他运行起来呢？回到命令提示符窗口（对，就是你输入编译并运行指令的那个命令行窗口），你会发现GDB已经运行了，你可以直接输入ld（链接qemu虚拟机），和c（continue的缩写），这时候，你会发现，内核的信息已经打印出来了（会打印username:），你直接敲两个回车就好。然后是bootmenu，请您使用键盘按下1，即可看到内核打印出的加载信息，这时候，您只需要静静等待就好，然后就可以开始快乐（悲伤）调试啦！

**0.4 内核代码梳理**

Acpi.c ACPI电源管理

ASCFONT.ASM zhouzhihao做的ascii字符点阵字库

Fs.c 文件系统的实现

File.c 相当于是fs.c的一个子集吧

Floppy.c 软盘驱动（待改进，太慢）

Hd.c 硬盘驱动（待改进，只能写256个扇区）

Init.c 存放sysinit函数，进行系统的一些初始化操作（具体将会在内核初始化章节讲解）

Irq.c 中断的禁止与允许

Mem.c 内存管理

Gdtidt.c GDTIDT

Page.c 内存分页

Jpeg.c 感谢Kawai的代码^\_^

Screen.c 显示器绘图驱动（将在绘图一章详细讲解）

Graphic.c 顾名思义，图形，提供了部分绘制函数。以及一个UI（将在绘图一章详细讲解）

Task.c 多任务驱动

Main.c 存放内核的入口函数（KernelMain）

Mouse.c PS2鼠标及键盘驱动

Pak.c pak文件的支持（打包）

Pci.c PCI驱动

Pcnet.c AMD PCNET网卡驱动

Other.c 存放一些杂类函数

Timer.c 定时器驱动

Vga,c VGA显示卡驱动

VBE.c 超级VGA显示卡驱动

Shell.c 终端的实现，可以理解为所有驱动的一种封装（但其只是提供了解析服务）

Setup.c 安装到硬盘的程序（已经废弃，在HD.c修好之前，这个文件将一直处于废弃状态）

Sheet.c 屏幕刷新驱动（感谢Kawai的代码^\_^）

Fifo.c FIFO缓冲区驱动

Dma.c DMA驱动

CASM.c CASM汇编器（by min0911\_ & zhouzhihao）

Cmos.c CMOS驱动

Beep.c 蜂鸣器驱动

Acpi.c ACPI内存管理驱动

Input\_Stack.c 输入内容的缓冲区（栈）

Fdboot.asm 软盘引导程序

Hdboot.asm 硬盘引导程序

Boot/dosldr.asm DOSLDR引导器

Pcnet.c AMDpcnet网卡驱动

**1. Powerint DOS 内核介绍**

Powerint DOS 386是一个多任务操作系统，支持运行C语言程序，支持VBE、VGA

动。一个简易的内核，有一个简易的UI

**1.1 内核初始化**

当内核（kernel.bin）被加载完成时，会调用sysinit函数，那么这个函数做了什么呢，让我们来剖析剖析，这个函数位于kernel/src/init.c,内核首先初始化了全局描述符表（GDT）和中断描述符表（IDT），这两个东西也是支撑操作系统运行的巨大保障。然后会初始化8295A中断芯片，让中断能够被我们的操作系统所接受，然后通过io\_sti()函数开中断。Io\_sti函数其实就是汇编指令sti的一层调用，sti也就是允许中断进入，与之相反的就是cli指令，禁止中断进入，这里使用sti是因为在他的上两层调用（DOSLDR引导器）在进入保护模式之前，就调用了cli指令不允许中断进入，刚刚我们初始化完了pic和IDT，这里就可以直接打开中断，不会引发异常，然后这里调用ClearMaskIrq函数，允许几个中断产生，分别是定时器中断和键盘中断还有鼠标中断。

然后初始化鼠标和键盘独有的FIFO缓冲区，以便后续我们能读取到键盘输入的键值，以及鼠标的数据。

然后我们将初始化键盘电路，然后启用鼠标。然后让鼠标停止向fifo缓冲区发送信息。接着，初始化定时器，并且初始化内存，如果内存大小小于16MB就发出一个警报，如果小于9MB就不允许进入系统，循环调用蜂鸣器驱动，后初始化Task，并且读取软盘所有内容。到此时，只需要调用mt\_init进行task最后的初始化sysinit就返回了。

* 1. **sysinit之后内核干了什么事情？**

可以看到，Kernelc.c只是调用了sysinit，便死循环了。那么系统是如何进入shell呢？

这就要细说sysinit函数在初始化task的时候干了什么事情。Sysinit在初始化task的时候，在mt\_init之前，先在GDT中创建了三个任务：两个系统保留任务，一个就是shell（注：这里的系统保留任务并不是我们想创建，而是因为我们的进程调度器（mt\_switch……函数（具体定义在task.c））,似乎需要至少两个进程进行调度，所以在正式开始多任务之前，我们需要添加两个以上的任务给任务调度器进行调度使用，不然将会导致内核CPU爆出异常，内核崩溃）。

* 1. **shell函数**

shell函数就是用户操作界面（shell，也就是内核的壳）的入口，那么这个shell函数干了什么奇葩事情呢？且听我慢慢道来。

执行shell函数的时候，内核将会先切换到80x25文本模式（调用的是BIOS功能，更为通用，即使目前是处于VBE模式下，也可以顺利切换到VGA的文本模式），然后执行注册（或登录的操作，这就引出了内核与用户的第一次交互（登录/注册操作））

然后将会调用文件系统（fs.c）查找软盘下面有没有autoexec.bat(批处理程序，每次启动操作系统，第一个执行的批处理文件（继承自MS-DOS）)

接着，就会进入到一个死循环里面，这个死循环的工作流程是：调用input函数，调用command\_run函数进行命令解析）

这就是shell函数的工作流程，接下来我们讲讲command\_run。

**1.4 command\_run函数的工作流程**

Command\_run函数的工作流程非常简单：

1. 跳过空格，要是都是空格直接返回。
2. 将所有字符转换成大写。
3. 判断命令
4. 结束

**1.5 Kernelc函数详解**

Kernelc.c的函数主要功能：维持系统的正常运转

**1.5.1 INT**

这可不是简单的int xxh，此函数是用于调用BIOS中断，例如VBE显卡功能，就是调用的此函数实现的。为什么使用BIOS中断，详见VBE显卡驱动章节

**1.5.2 sleep**

Sleep函数（换句话说是wait），接受一个参数（百分之一秒）

**1.5.3 sleep函数的实现**

Sleep函数的实现非常简单，其实就是在timer中创建了一个定时器，然后使用while循环等待定时器超时，一旦超时就释放这个定时器，然后返回（return）

**1.5.4 input函数**

Input函数等同于gets\_s函数。调用此函数，会等待用户输入一个以回车结尾的字符串，然后将这个字符串存储到一个缓冲区中。

**1.5.5 input函数目前支持的功能**

1. 基本输入

2. 左右移动方向键

3. 上下选择历史命令

**1.5.6 input函数发展历史**

1. （in Simple OS）21.11 input函数编码完成

2. 22.1 发现重大bug，input函数只返回一个地址为0的字符指针，十分不安全！更改为了接受两个参数（字符缓冲区地址和缓冲区的长度（或者说你想读取多少字节））

3. 22.2 更新左右移动光标并插入字符

4. 22.2 引入input\_stack，增加上下选择历史命令功能

5. 22.2 debug

6. 22.3 修复一个影响体验的bug（上下选择历史命令有时候很迟钝）

**1.5.7 input\_No\_El**

Input函数调用后，会进行换行操作，这个函数调整了这一功能，但是还有bug，待改进，所以在实际开发中请尽量不要使用它，或者您可以帮忙修复它。

**1.5.8 getch**

等待用户输入一个字符，并返回这个字符

**1.5.9 input\_char\_inSM**

等待用户按下键盘，并返回其扫描码

**1.5.10 kbhit**

这个函数是为一个程序而实现的（snake.bin），与conio.h中的\_kbhit函数相同，都是返回用户是否按下键盘（按下返回1（true），否则返回0（false））

**1.5.11 函数的实现**

我们首先设定了一个变量（Pressed），可以说是一个标志变量，当然，只拥有这个变量是不够的，我们会在关于键盘的中断处理程序中去设置这个变量的值为1（true），那么这个kbhit函数究竟是如何实现的呢？其实非常简单，就是检查这个变量是否为1，是的话返回（true）并且将这个变量的值初始化为0。如你所见，这个函数有很多缺陷，所以这也是我们需要更改的函数之一，希望你看了这个手册之后能够帮忙改进这一缺陷（0.5c）

**1.5.12 各种Get**

为什么需要get这类函数呢？或许有人会说，直接extern不就好了？？？因为有的变量是不希望别的函数进行一个修改，但是这个变量又需要别的函数读取来实现功能，所以才会有了Get函数，例如：get\_cons\_color这个函数，就是返回控制台的颜色信息

**1.5.13 各种中断处理函数**（这里就不详细说，在中断处理一章细讲）

**1.6 fs.c 我们的文件系统是怎么实现的？以及他的bug？将来呢？**

**1.6.1 文件系统的实现**

在fs,c下面，有一个:file\_search函数，可能有很多人不解，这个函数他好像是30天里面那个（就是那个【捂脸】，不想造轮子，不过也就这个），可能有人会说，这个函数只能读取硬盘根目录下的文件啊！哈，参数里面的finfo给我们提供了自由的空间，我们可以用软盘在内存中的地址+目录地址的形式来访问。（变量dictaddr就是当前处于的目录地址），例如0x002600就是根目录的目录地址，这个函数就可以通过第二个自由的参数访问所有目录下的文件。

还有dict\_search，与上面的file\_search相对，顾名思义，查找目录，参数与file\_search相同，这里就不多赘述了。

由于目前内核是使用的fat12文件系统，这里给大家普及一下fat12文件系统的知识：fileinfo结构体中，有一个成员叫：type，那么有人就要问了，这个type是什么呢？这个type是干什么的呢？

Type顾名思义，就是文件的属性，我们便是通过这个type来区分是文件还是文件夹，如果这个type是等于0x10，说明这是个文件夹，如果等于0x20说明这是个文件，文件和文件夹当然不同啦！所以相应的也有其对应的处理方式。

其实文件和文件夹非常类似，他们都可以通过一个相同的公式（在fopen函数中有对应的实现），来读取其的内容，不过内容有所不同。文件的内容就是文件，比如:hello world，而文件夹就不同了，记得我们之前提到过的0x2600这个根目录地址吗，文件夹的结构就和这个根目录地址一样，所以其实根目录也算是（或者说就是）一个文件夹了，0x2600就是他的地址，我们的文件系统也就是依靠这个理论来实现的。

Get\_File\_Address函数，大家在使用MS-DOS的时候，查找一个文件夹下面的文件，但不使用CD命令，一般都会这样（假设你要读取的文件是：other文件夹下的a.txt）:other/a.txt,所以我们的系统内核也必须支持这种查找文件的方式，要不然，连个DOS都算不上了！（笑），实现这个功能的主要函数也就是我们上面提到的Get\_File\_Address来实现的，这是函数的依赖也是上面提到过的file\_search啊dict\_search啊，这些函数来包装组成的函数，那么，接下来，我们就来讲讲这个函数是怎么实现的。

说白了就简单分为3步：

1. 保存原来的目录地址
2. 一步步跳过’/’或者反斜杠，并且更新地址
3. 到最后，如果dict\_search的返回结果为0，就尝试使用file\_seaech，如果正确运行，就返回，否则就返回0（没找到）

**1.6.2 file.c**

这个file.c是fs.c的番外扩展（就是效仿Windows啦）

**1.7 绘图**

看了这么多，你可能认为，这个DOS也只是个DOS，只有文字操作界面了吧！大！错！特！错 ！内核中（screen.c，以及graphic.c）提供了关于绘图的函数，那么有人会问了：“这个系统，不是运行在文本模式下的吗？？？你绘制个串串图啊”，这就要提到另外一个东西了，VGA驱动和VBE驱动（这两个将在显示驱动一章中讲解），你们可以通过SwitchTo320X200X256这样一个函数来将当前的显示模式切换到一个VGA图形模式，（显存被映射在0xA0000）您可以调用Draw\_Px进行画点。Draw\_Box来画方块,Draw\_Char来绘制字符，PUTCHINESE来绘制中文，注意：这些函数都只用于320X200模式的绘图，诺是您在其他模式（例如VBE模式），就不能使用这些函数，如果您处于256色环境，那么您可以调用有S后缀的函数（PUTCHINESE的S后缀版叫：PUTCHINESE0），如果你仔细看过VBE相关的功能，你会发现，VBE有真彩色模式，那么就不能适用于S版本了，但是您可以调用32（真彩色）后缀的函数。

**1.7.1 那么这些函数怎么用呢？**

**void Draw\_Px(int x, int y, int color);**

这是Draw\_Px函数的原型，这个函数用于在320X200图形模式下面画点（如何进入320X200图形模式？您可以看图形驱动的那一章节，也可以看上面说的内容，都有介绍如何进入这个图形模式）

这个函数的第一个参数用于接收这个像素点的在屏幕上的x（横坐标位置），第二个参数用于接收这个像素点在屏幕上的y（纵坐标位置），第三个参数则用于指定这个像素点的颜色（pal（调色板内的颜色，调色板可以自行设置，如何设置？请看图形驱动那一章节（VBE的256色模式也是通用的）））。

示例：（如何运行？您可以自行添加一个命令，也可以在Makefile中的CDEF中加上-D\_\_DEBUG\_\_然后转到shell.c，找到a\_test函数，您看到他的时候他应该是被一个#ifdef预处理器所包裹，您可以不用管，请直接键入您的代码，然后在命令提示符下面，输入：make img\_run（如果您使用的是Powershell，请输入：“./make img\_run”）如果出现了报错，请您查看序章中环境搭建的步骤，照着做一遍，如仍然无法正常编译，那请联系QQ:1474635462，然后进入到内核，输入a\_test（可能没有，请自行添加，如何添加（请看command\_run函数详解））命令，即可成功调用您刚刚写的代码）

SwitchTo320X200X256(); //切换到图形模式

Draw\_Px(0,0,0xf); //在原点位置处画点

getch(); //等待用户在键盘键入一个字符

SwitchToText8025(); //切换到文本模式

如果您认真键入了以上代码并且成功编译运行了内核与a\_test命令，那么您应该会在屏幕上的原点看到一个白色的的像素点。

**void Draw\_Char(int x, int y, char c, int color);**

如你所见，这是Draw\_Char的函数原型，第一个参数，传入的是需要显示字符的x（也就是相对于整个屏幕的横轴坐标），第二个呢，就是需要显示字符的y（也就是是相对于整个屏幕的纵轴坐标），用这两个参数，我们就能定位到屏幕上任何一个点，而第三个参数c，就是需要显示字符的ascii码（使用C语言的话，可以直接使用单引号括起来，更直观的得到字符的ascii码。）第四个参数color，相信已经很直观了，不过应该还是有朋友看不懂，我这里讲一下，就是显示这个字符需要什么颜色，这个颜色是一个8位值，也就是说，这个颜色的值最多只能是0-255，颜色的定义在vga.c的调色板中定义，大概是以RGB的形式定义的，大家可以很方便的咨询。

好的。相信大家已经迫不及待了，来写一个代码示例吧，添加a\_test命令（上面添加过的，就不用再次添加了，直接使用即可）。然后在里面键入代码

SwitchTo320X200X256(); //切换到图形模式

Draw\_Char(0,0,’A’,0xf); //再原点位置，画一个8x16大小的A字符

getch(); //等待用户在键盘键入一个字符

SwitchToText8025(); //切换到文本模式

好，我们运行一下，就能看到一个大大的A出现在原点，为什么这个A这么大呢？因为我们设置的显示模式是320x200，每个像素点是非常大的，在你如此高分辨率的显示屏上显示320x200，大肯定是必然的。那么要想小起来怎么办呢？请切换到VBE显示模式，并使用另一个函数，我们将在（VBE驱动这章详细讲解）

**1.8 创建进程及进程间通讯**

进程（task）又名任务，是操作系统实现最重要的板块之一。那么Powerint DOS 386又是怎么实现的呢？我该怎么去创建进程？以及怎么进行进程间的通讯呢？

首先，Powerint DOS 386有一个关于进程的结构体，我们通常叫做进程描述符。

struct TASK {

int sel, sleep, level;

struct TSS32 tss;

char name[32];

char running;

struct tty\* TTY;

struct FIFO8 \*keyfifo, \*mousefifo;

int fifosleep;

int cs\_base, ds\_base;

int alloc\_addr;

int esp0, esp1;

short ss1;

char \*memman;

struct IPC\_Header IPC\_header;

} \_\_attribute\_\_((packed));

让我们来详细讲解下这些变量。

sel -> 进程GDT选择子 sleep -> 进程休眠状态 level -> 进程优先级

tss -> 进程TSS描述符 running -> 进程运行状态，为结束标志

TTY –> 进程位于的控制台 keyfifo -> 进程的键盘缓冲区

mousefifo -> 进程的鼠标缓冲区 fifosleep -> 进程的硬件休眠状态

cs\_base -> 应用程序进程的CS基地址 ds\_base -> 应用程序进程的DS基地址

alloc\_addr -> 应用程序进程的内存缓冲区 esp0 -> 应用程序的系统堆栈指针

esp1 -> 应用程序的本身堆栈指针 ss1 -> 应用程序的本身SS

memman -> 应用程序进程的内存缓冲区的管理缓冲区

IPC\_header -> 进程通信头

**1.9 内存分页的实现**

Powerint DOS的内存分页是PAE的二级分页。

那么是如何实现的呢？这章我们将详细解析Powerint DOS的page.c

首先，我们要知道，分页是个啥，其实，分页就是作除法，通过分页机制，将一整个内存，分成无数个4kb的内存，一个4kb的内存就是一个页，每个页都有一个他独有的内存地址，用户只需要用一个线性地址，就可以通过这个线性地址访问每个页，那么，怎么描述线性地址呢？这就需要用到页表，通过页表，页表储存了所有页的信息（比如物理地址，比如这个页可不可写），有趣的是，页表本身，竟然也是一个页，光说，确实很枯燥，那么，画个图呢？





