Book OS

操作系统开发者手册

Operating system developer's manual

系统版本：Ver0.6.1

修订日期：2019/11/24

目录

系统简介 - 4 -
版权声明 - 4 -
第一章 开发环境的搭建 - 6 -
1.1 环境需求 - 6 -
1.2 环境原理 - 6 -
1.3 Windows 下的环境搭建 - 6 -
1.4 Linux 下的环境搭建 - 13 -
1.5 MacOS 下的环境搭建？？？ - 19 -
第二章 源码目录的解析 - 20 -
2.1 目录总览 - 20 -
2.1 Img 目录 - 20 -
2.2 Src 目录 - 21 -
2.2.1 arch 目录 - 22 -
2.2.2 drivers 目录 - 25 -
2.2.3 fs 目录 - 26 -
2.2.4 hal 目录 - 26 -
2.2.5 include 目录 - 26 -
2.2.6 init 目录 - 28 -
2.2.7 kernel 目录 - 29 -
2.2.7 mm 目录 - 31 -
2.2.7 share 目录 - 32 -
2.2.8 user 目录 - 32 -
2.2.8 其它文件 - 34 -
第三章 运行流程分析 - 35 -
3.1 引导加载流程 - 35 -
3.2 Arch 初始化流程 - 35 -
3.3 内核初始化流程 - 35 -
第四章 对未来的展望 - 38 -
4.1 计划中的内容 - 38 -
4.2 开发者的注意事项 - 38 -

后记.....- 41 -

系统简介

BookOS 是一个基于 x86 平台的 32 位操作系统，其版权属于 BookOS 开发者所有。经理了多个版本，在 0.6.1 这个版本中添加了许多新鲜的内容：平台架构，节点内存管理，虚拟内存，上下文切换的多任务，块设备框架和字符设备框架。使用了新的代码编写风格，有一种重获新生的感觉。XBook 与 BookOS 的关系：XBook 是 BookOS 的新内核，用一种新的方向去编写操作系统。

版权声明

我们采用开源的方式发布，其源码遵循 GPL-v3.0 协议。GPL-v3.0 协议声明如下（截取部分）：

GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program--to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you

these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

第一章 开发环境的搭建

1.1 环境需求

开发工具列表：gcc，nasm，ld，dd，rm，make
虚拟机：bochs，qemu，virtual box, vmware 等
当前版本：0.6.1
Git 地址：https://github.com/huzichengdevelop/XBook

1.2 环境原理

BookOS 采用 c 语言和汇编混合编写。靠近底层的内容，接口这些使用汇编来写，其它的采用 c 语言来写。也就是说 c 不能做的都用汇编来做。

Boot 和 loader 都是汇编语言编写，直接通过 nasm 生成二进制文件。

先用 gcc 把所有.c 源码文件和.asm 源码文件编译成.o 对象文件，然后再通过 ld 把.o 文件链接生成 ELF 格式的系统文件。

现在得到 boot.bin, loader.bin, kernel.elf 三个文件。

由于我们是在虚拟机里面测试运行系统，所以我们需要有虚拟磁盘（硬盘，软盘，光盘…）。在这里，为了简便，我们是用软盘作为引导盘。那么我们就需要把 boot，loader 和 kernel 写入到软盘中去。由于不通过文件系统加载，而是通过扇区加载的，所以直接用 dd 工具把这些文件写入到软盘中去。

磁盘镜像文件在 img 目录下面，软盘镜像是 a.img，无论在哪个虚拟机里面，只要配置了软盘，就可以引导启动系统了。

而对于硬盘，不同的虚拟机，虚拟硬盘格式不一样，bochs 和 qemu 的硬盘可以是 img 格式，而 virtual box 的硬盘不是 img，所以硬盘需要自行配置。不过，这里我们也提供了 c.img（bochs&qemu）,c.vhd（virtual box），这个是 IDE 硬盘的“主通道主盘”。

那么，我们需要有编译环境和运行环境。整个流程如下：

C&asm 源码->gcc&nasm 编译器->obj 对象文件->ld 链接器->操作系统可执行文件->dd 工具写入虚拟磁盘（a.img 软盘）->配置虚拟机环境->虚拟机运行

接下来，就开始配置开发环境。

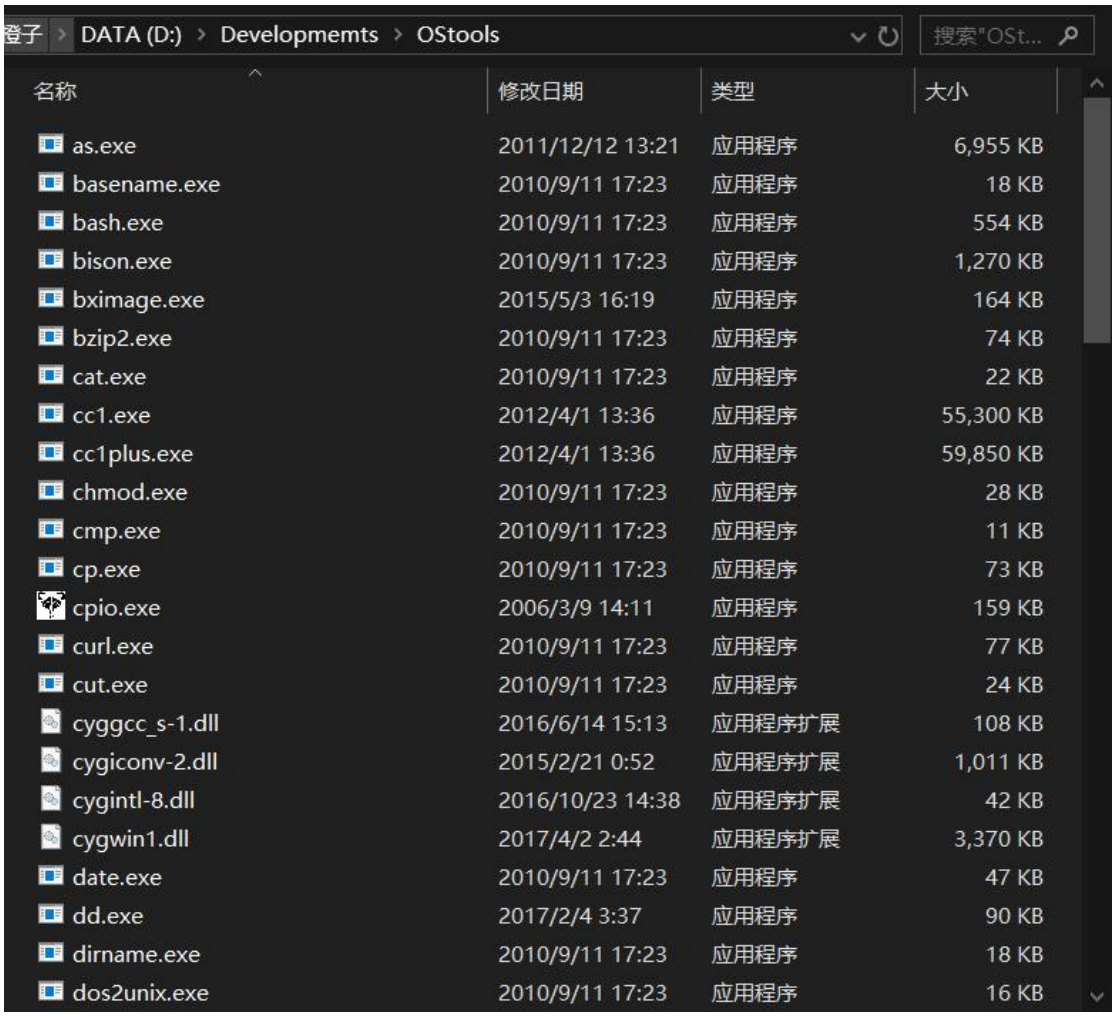
如果觉得配置 Qemu 或者 Bochs 麻烦的话，就直接下载 VirtualBox 就好了，那个下载安装后，可以用图形界面来配置，比较容易。你也可以使用你自己喜欢的虚拟机，我只是举个栗子。^_^

1.3 Windows 下的环境搭建

在 windows 的环境下需要下载一些有这些工具的环境。可以通过 Cygwin，Mingw 这两个软件得到这些工具。在这里，我自己提取了一些我们需要使用的工具，然后打包了，大家可以在这个链接去下载：

http://www.book-os.org/tools/BuildTools-v2.rar

下载好之后解压出来，解压到一个目录下面，我放到了 D:\Developmemts\OStools 这个目录。



登子 > DATA (D:) > Developmemts > OStools

名称	修改日期	类型	大小
as.exe	2011/12/12 13:21	应用程序	6,955 KB
basename.exe	2010/9/11 17:23	应用程序	18 KB
bash.exe	2010/9/11 17:23	应用程序	554 KB
bison.exe	2010/9/11 17:23	应用程序	1,270 KB
bximage.exe	2015/5/3 16:19	应用程序	164 KB
bzip2.exe	2010/9/11 17:23	应用程序	74 KB
cat.exe	2010/9/11 17:23	应用程序	22 KB
cc1.exe	2012/4/1 13:36	应用程序	55,300 KB
cc1plus.exe	2012/4/1 13:36	应用程序	59,850 KB
chmod.exe	2010/9/11 17:23	应用程序	28 KB
cmp.exe	2010/9/11 17:23	应用程序	11 KB
cp.exe	2010/9/11 17:23	应用程序	73 KB
cpio.exe	2006/3/9 14:11	应用程序	159 KB
curl.exe	2010/9/11 17:23	应用程序	77 KB
cut.exe	2010/9/11 17:23	应用程序	24 KB
cyggcc_s-1.dll	2016/6/14 15:13	应用程序扩展	108 KB
cygiconv-2.dll	2015/2/21 0:52	应用程序扩展	1,011 KB
cygintl-8.dll	2016/10/23 14:38	应用程序扩展	42 KB
cygwin1.dll	2017/4/2 2:44	应用程序扩展	3,370 KB
date.exe	2010/9/11 17:23	应用程序	47 KB
dd.exe	2017/2/4 3:37	应用程序	90 KB
dirname.exe	2010/9/11 17:23	应用程序	18 KB
dos2unix.exe	2010/9/11 17:23	应用程序	16 KB

接下来，还需要下载虚拟机。只是给出了链接地址，大家自行下载，如果链接下载不了，可以自己搜索其它下载地址。

QEMU 下载地址：https://qemu.weilnetz.de/

BOCHS 下载地址：https://www.psyon.org/projects/bochs-win32/

Virtaul Box 下载地址：https://www.virtualbox.org/wiki/Downloads

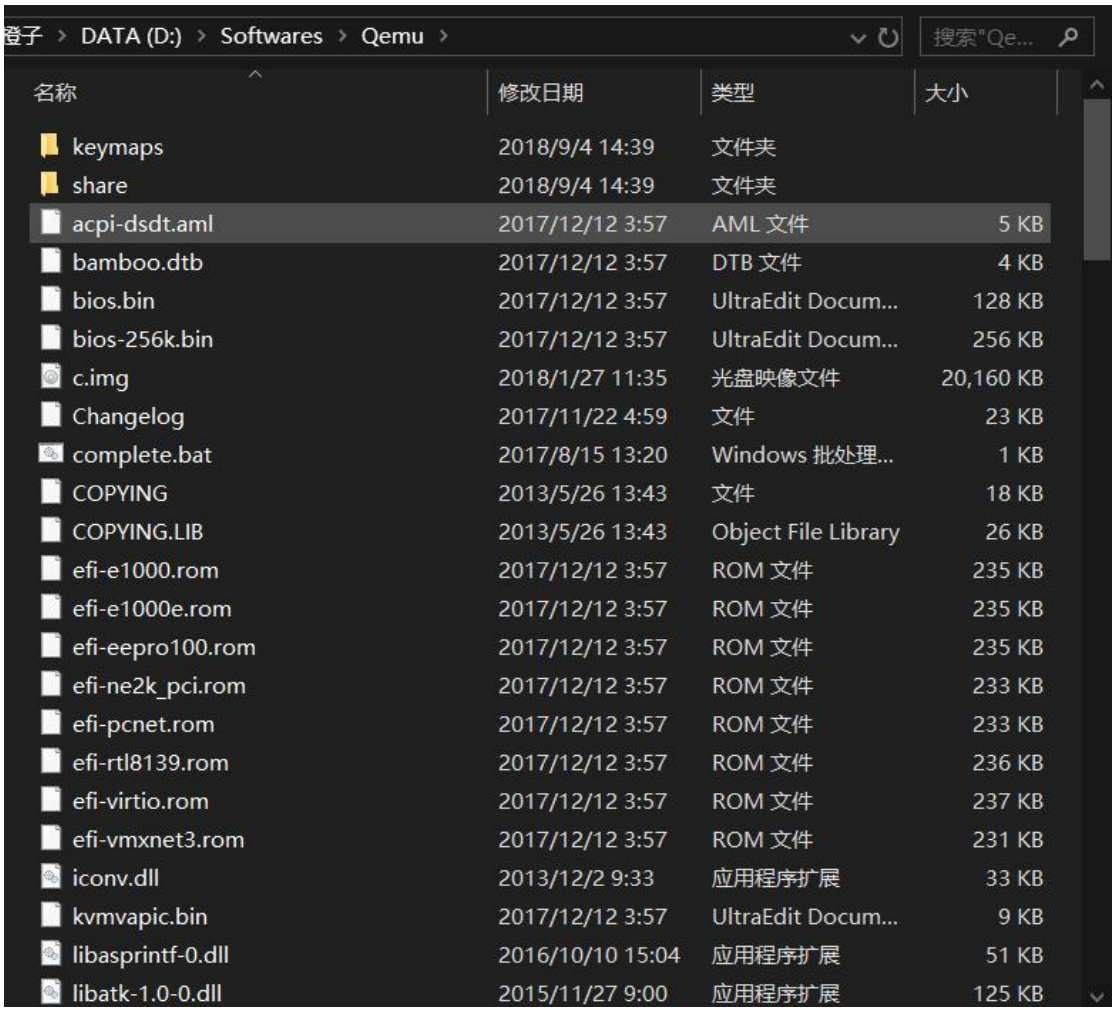
Vmware 下载地址:

https://my.vmware.com/cn/web/vmware/info/slug/desktop_end_user_computing/vmware_workstation_pro/15_0

就下载 1 个就可以了，我常用的是 qemu，由于 qemu 支持很多处理器，在这里我们只使用 i386，所以我把 i386 的那部分提取出来了。这样 qemu 的大小就小很多了。

精简后的 I386 的 QEMU：http://www.book-os.org/tools/ Qemu-i386.rar

这个下载后直接解压就可以使用了。我放到了 D:\Softwares\Qemu 这个路径。



登子 > DATA (D:) > Softwares > Qemu >

名称	修改日期	类型	大小
keymaps	2018/9/4 14:39	文件夹	
share	2018/9/4 14:39	文件夹	
acpi-dsdt.aml	2017/12/12 3:57	AML 文件	5 KB
bamboo.dtb	2017/12/12 3:57	DTB 文件	4 KB
bios.bin	2017/12/12 3:57	UltraEdit Docum...	128 KB
bios-256k.bin	2017/12/12 3:57	UltraEdit Docum...	256 KB
c.img	2018/1/27 11:35	光盘映像文件	20,160 KB
Changelog	2017/11/22 4:59	文件	23 KB
complete.bat	2017/8/15 13:20	Windows 批处理...	1 KB
COPYING	2013/5/26 13:43	文件	18 KB
COPYING.LIB	2013/5/26 13:43	Object File Library	26 KB
efi-e1000.rom	2017/12/12 3:57	ROM 文件	235 KB
efi-e1000e.rom	2017/12/12 3:57	ROM 文件	235 KB
efi-eepro100.rom	2017/12/12 3:57	ROM 文件	235 KB
efi-ne2k_pci.rom	2017/12/12 3:57	ROM 文件	233 KB
efi-pcnet.rom	2017/12/12 3:57	ROM 文件	233 KB
efi-rtl8139.rom	2017/12/12 3:57	ROM 文件	236 KB
efi-virtio.rom	2017/12/12 3:57	ROM 文件	237 KB
efi-vmxnet3.rom	2017/12/12 3:57	ROM 文件	231 KB
iconv.dll	2013/12/2 9:33	应用程序扩展	33 KB
kvmvapic.bin	2017/12/12 3:57	UltraEdit Docum...	9 KB
libasprintf-0.dll	2016/10/10 15:04	应用程序扩展	51 KB
libatk-1.0-0.dll	2015/11/27 9:00	应用程序扩展	125 KB

现在我们的编译环境和虚拟机都有了，还需要做的事情就是配置一下环境变量。

环境变量：把路径添加到环境变量后，在系统的任何路径都可以解析该路径下面的内容。

这个是我自己理解的意思，我们把路径的环境变量配置好之后，就可以在我们电脑的任何目录下面编译运行 BookOS 了。

如果你的电脑已经有 gcc，nasm 这些工具了就不需要配置这个 BuildTools，如果有虚拟机就不用配置这个 QEMU 了。有的话可以跳过这儿，没有就可以继续看看。

安装流程：

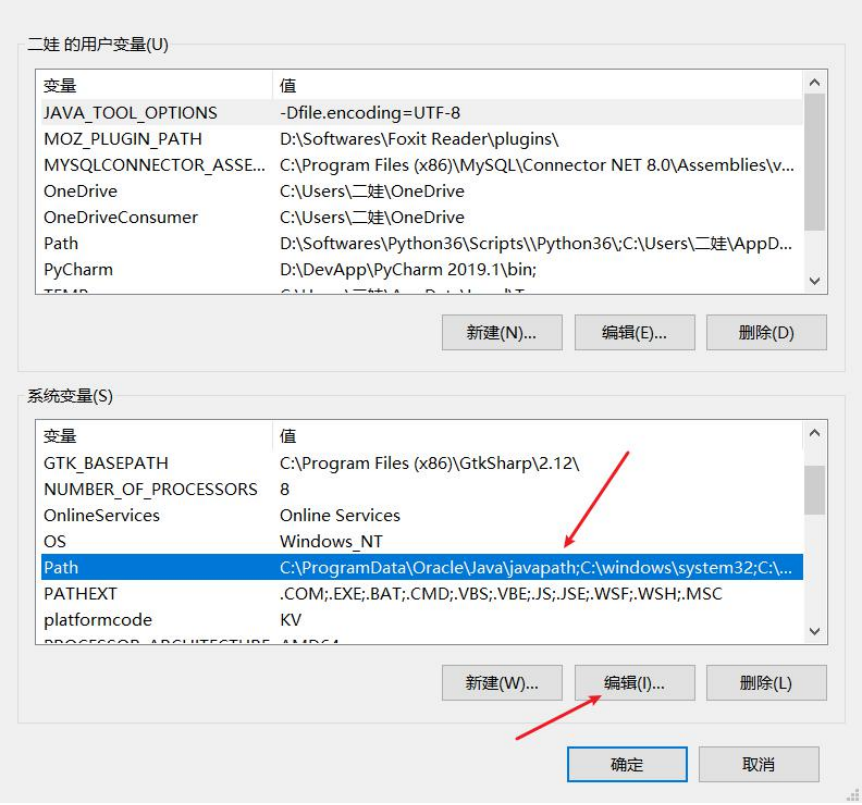
此电脑->鼠标右击->属性->高级系统设置->环境变量->系统变量的 Path->编辑->新建->把你的目录地址填进去

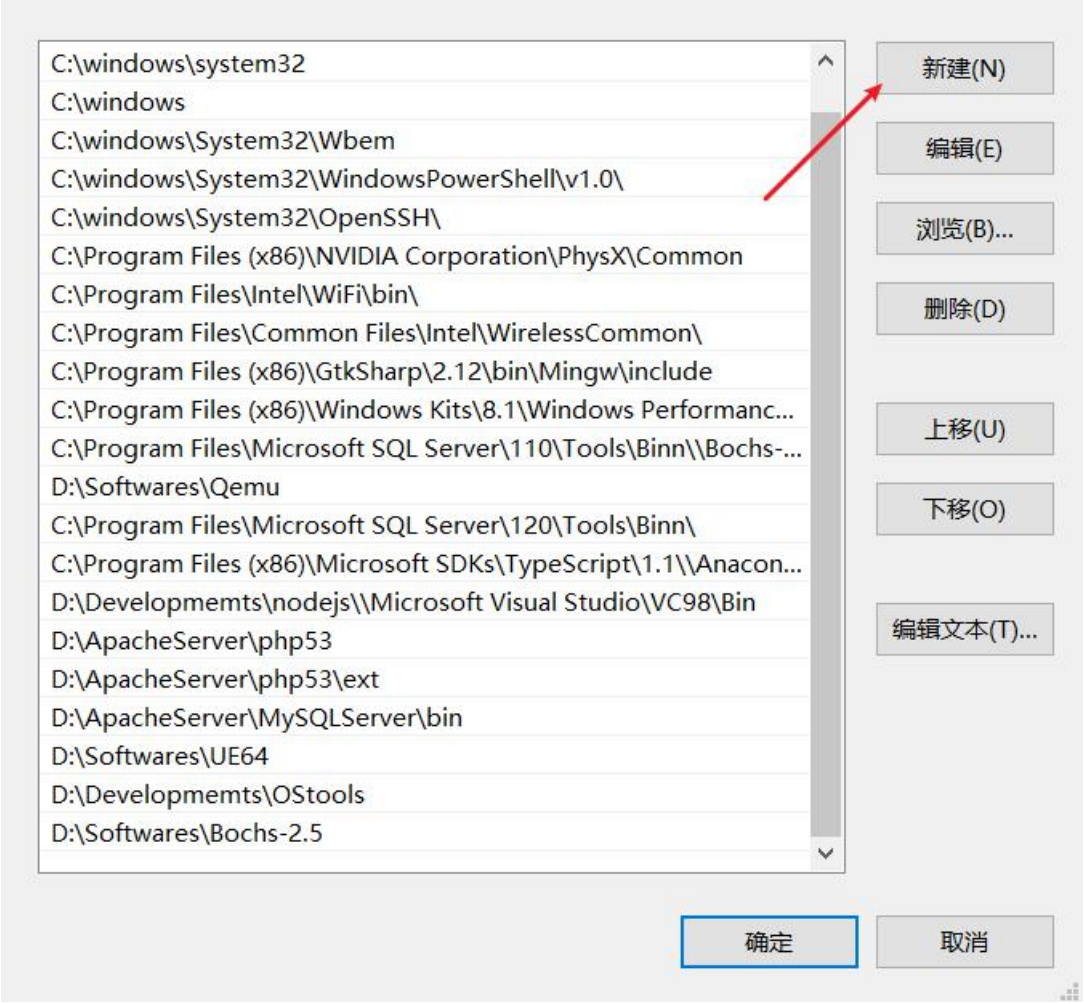
2 个目录都要填写，这样，才可以找到 tools 和 qemu。安装好后点击确定就好了。有图有真相！

打开(O)
固定到快速访问
管理(G)
从"开始"屏幕取消固定(P)
映射网络驱动器(N)...
断开网络驱动器的连接(C)...
创建快捷方式(S)
删除(D)
重命名(M)
属性(R)

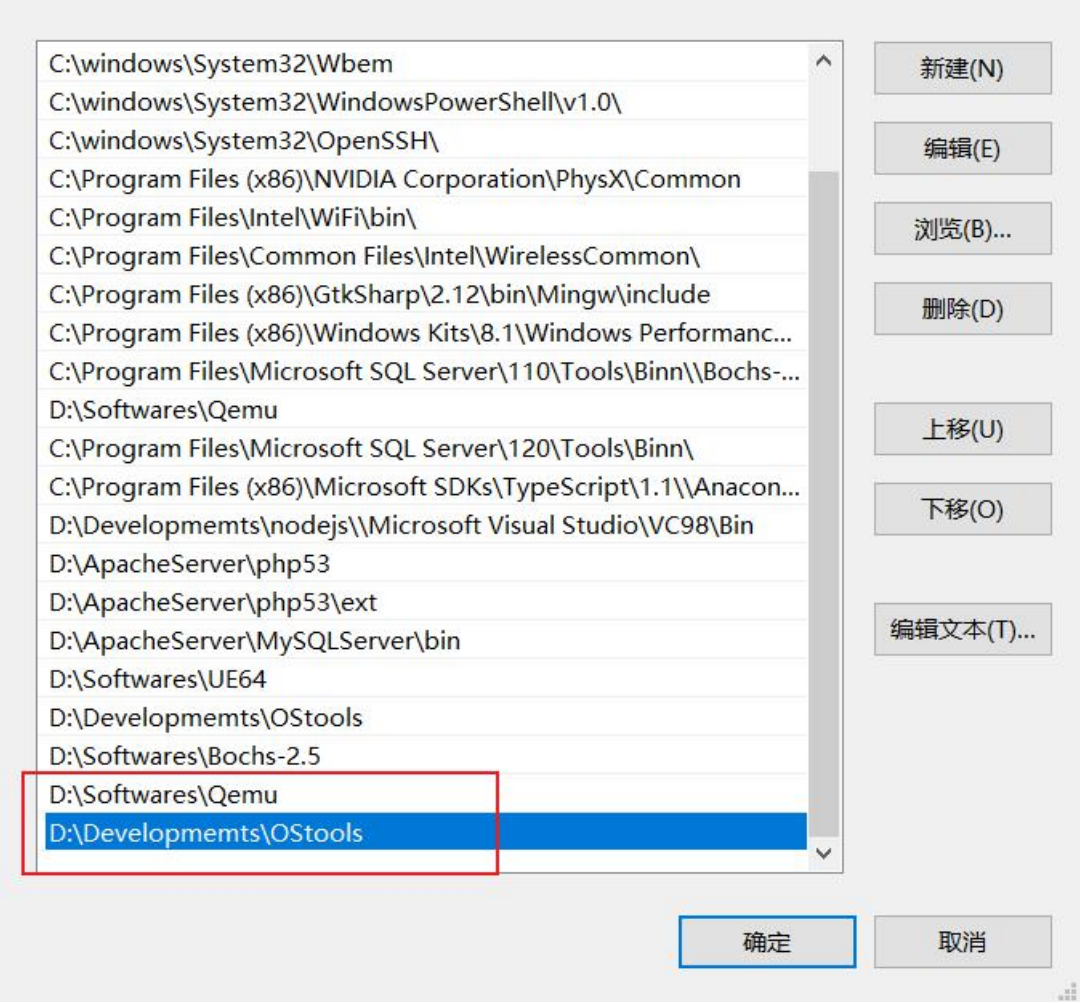
系统
控制面板 › 系统和安全 › 系统
搜索控制...
控制面板主页
设备管理器
远程设置
系统保护
高级系统设置
查看有关计算机的基本信息
Windows 版本
Windows 10 家庭中文版
© 2019 Microsoft Corporation。保留所有权利。
Windows 10
系统
处理器:
Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz 2.30 GHz
已安装的内存(RAM):
8.00 GB (7.81 GB 可用)
系统类型:
64 位操作系统，基于 x64 的处理器
笔和触控:
没有可用于此显示器的笔或触控输入
技术支持信息
另请参阅
安全和维护
计算机名、域和工作组设置

计算机名
硬件
高级
系统保护
远程
要进行大多数更改，你必须作为管理员登录。
性能
视觉效果，处理器计划，内存使用，以及虚拟内存
设置(S)...
用户配置文件
与登录帐户相关的桌面设置
设置(E)...
启动和故障恢复
系统启动、系统故障和调试信息
设置(T)...
环境变量(N)...
确定
取消
应用(A)

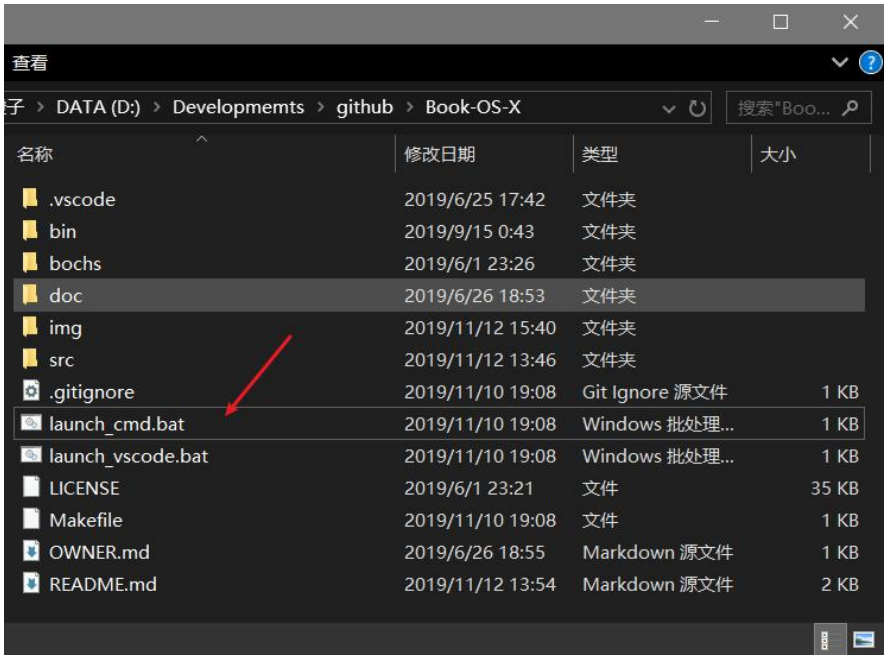

二娃 的用户变量(U)
变量
值
JAVA_TOOL_OPTIONS
-Dfile.encoding=UTF-8
MOZ_PLUGIN_PATH
D:\Softwares\Foxit Reader\plugins\
MYSQLCONNECTOR_ASSE...
C:\Program Files (x86)\MySQL\Connector NET 8.0\Assemblies\v...
OneDrive
C:\Users\二娃\OneDrive
OneDriveConsumer
C:\Users\二娃\OneDrive
Path
D:\Softwares\Python36\Scripts\\Python36\;C:\Users\二娃\AppD...
PyCharm
D:\DevApp\PyCharm 2019.1\bin;
新建(N)...
编辑(E)...
删除(D)
系统变量(S)
变量
值
GTK_BASEPATH
C:\Program Files (x86)\GtkSharp\2.12\
NUMBER_OF_PROCESSORS
8
OnlineServices
Online Services
OS
Windows_NT
Path
C:\ProgramData\Oracle\Java\javapath;C:\windows\system32;C:\...
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
platformcode
KV
新建(W)...
编辑(I)...
删除(L)
确定
取消


C:\windows\system32
C:\windows
C:\windows\System32\Wbem
C:\windows\System32\WindowsPowerShell\v1.0\
C:\windows\System32\OpenSSH\
C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common
C:\Program Files\Intel\WiFi\bin\
C:\Program Files\Common Files\Intel\WirelessCommon\
C:\Program Files (x86)\GtkSharp\2.12\bin\Mingw\include
C:\Program Files (x86)\Windows Kits\8.1\Windows Performanc...
C:\Program Files\Microsoft SQL Server\110\Tools\Binn\\Bochs-...
D:\Softwares\Qemu
C:\Program Files\Microsoft SQL Server\120\Tools\Binn\
C:\Program Files (x86)\Microsoft SDKs\TypeScript\1.1\\Anacon...
D:\Developmemts\nodejs\\Microsoft Visual Studio\VC98\Bin
D:\ApacheServer\php53
D:\ApacheServer\php53\ext
D:\ApacheServer\MySQLServer\bin
D:\Softwares\UE64
D:\Developmemts\OStools
D:\Softwares\Bochs-2.5
新建(N)
编辑(E)
浏览(B)...
删除(D)
上移(U)
下移(O)
编辑文本(T)...
确定
取消

C:\windows
C:\windows\System32\Wbem
C:\windows\System32\WindowsPowerShell\v1.0\
C:\windows\System32\OpenSSH\
C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common
C:\Program Files\Intel\WiFi\bin\
C:\Program Files\Common Files\Intel\WirelessCommon\
C:\Program Files (x86)\GtkSharp\2.12\bin\Mingw\include
C:\Program Files (x86)\Windows Kits\8.1\Windows Performanc...
C:\Program Files\Microsoft SQL Server\110\Tools\Binn\\Bochs-...
D:\Softwares\Qemu
C:\Program Files\Microsoft SQL Server\120\Tools\Binn\
C:\Program Files (x86)\Microsoft SDKs\TypeScript\1.1\\Anacon...
D:\Developmemts\nodejs\\Microsoft Visual Studio\VC98\Bin
D:\ApacheServer\php53
D:\ApacheServer\php53\ext
D:\ApacheServer\MySQLServer\bin
D:\Softwares\UE64
D:\Developmemts\OStools
D:\Softwares\Bochs-2.5
D:\Softwares\Qemu



现在环境变量配置好了之后，就需要打开源码，开始编译了。



下载源码后，双击 launch_cmd.bat，打开控制台。输入 make run 就可以编译运行了。

```
D:\Developmemts\github\Book-OS-X>make run
[LD] arch/x86/boot/built-in.o
[NASM] arch/x86/boot/boot.bin
[NASM] arch/x86/boot/loader.bin
[NASM] arch/x86/kernel/entry.o
[CC] arch/x86/kernel/arch.o
[NASM] arch/x86/kernel/x86.o
[CC] arch/x86/kernel/segment.o
[CC] arch/x86/kernel/gate.o
[NASM] arch/x86/kernel/interrupt.o
[CC] arch/x86/kernel/pic.o
[CC] arch/x86/kernel/cpu.o
[CC] arch/x86/kernel/cmos.o
[LD] arch/x86/kernel/built-in.o
[CC] arch/x86/mm/page.o
[CC] arch/x86/mm/bootmem.o
[CC] arch/x86/mm/ards.o
[CC] arch/x86/mm/phymem.o
[LD] arch/x86/mm/built-in.o
[CC] arch/x86/bus/pci.o
[LD] arch/x86/bus/built-in.o
[LD] arch/x86/built-in.o
[CC] init/main.o
[LD] init/built-in.o
[CC] kernel/bitmap.o
[CC] kernel/hal.o
[CC] kernel/debug.o
[CC] kernel/syscall.o
[CC] kernel/task.o
[CC] kernel/schedule.o
```

```
Machine   View
-> Gate descriptor <-
-> Tss <-
-> Physic Memory -> PCI
<-
<-
-> Kernel main -> Hal kernel  ¦- Registed hal-cpu
 ¦- Registed hal-ram
<-
-> Mem Cache <-
-> VM Area <-
-> Task -> VMSpace <-
<-
-> Work queue <-
-> Clock driver <-
-> CharDevice -> Test <-
<-
-> BlockDevice -> Ramdisk Driver <-
-> Ide Driver  ¦- Ide disks are 2
<-
<-
 ¦- execv: path root:init
I am parent, my pid is 0 my child is 6.
I am child, my pid is 6.
Execv shell.
```

1.4 Linux 下的环境搭建

系统环境： CentOS Linux release 7.6.1810 (Core)

首先，也是需要安装 gcc 和 nasm 编译器，其它的 linux 里面都默认自带，如果这两个编译器都有了的话，就不用安装了，如果没有，就需要自己安装。由于 linux 中版本太多，如果你是 linux 使用者，那么，想必你对如何安装这些软件是得心应手的。

其次，是安装虚拟机，这里，我拿 bochs 虚拟机做演示。

打开这个网页：https://sourceforge.net/projects/bochs/files/bochs/2.6.9/

https://sourceforge.net/projects/bochs/files/bochs/2.6.9/

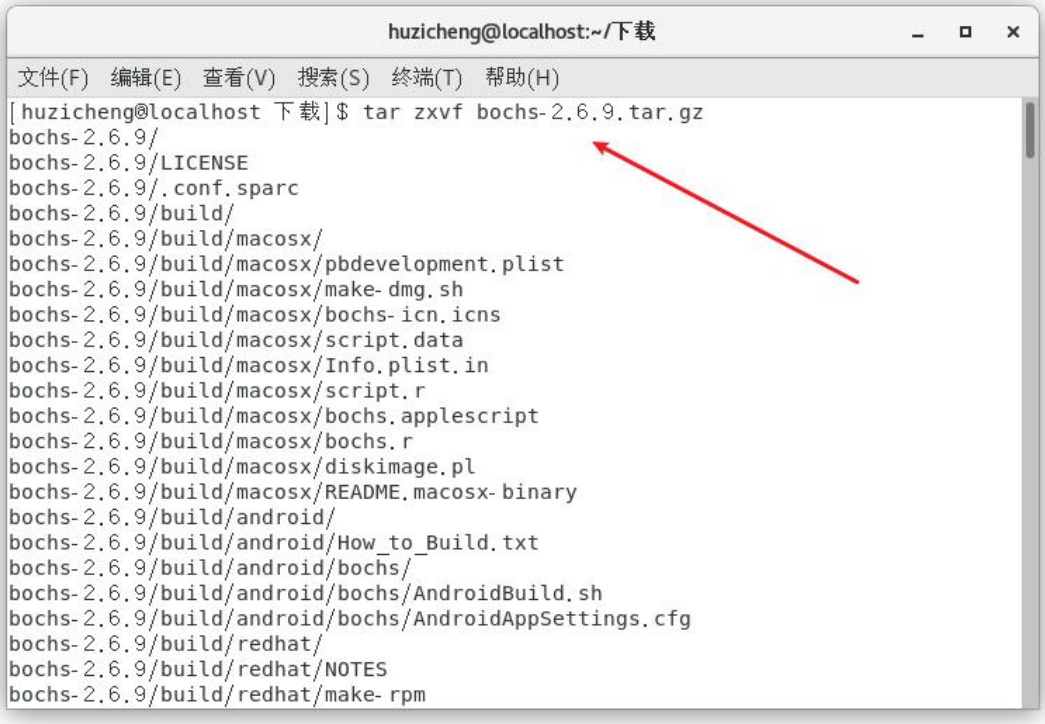
SOURCEFORGE Open Source Software Business Software Services Resources

Home / bochs / 2.6.9

Name	Modified	Size	Downloads / Week
Parent folder			
bochs-2.6.9-win64.zip	2017-04-17	3.0 MB	127
bochs-2.6.9-1.i586.rpm	2017-04-09	3.2 MB	94
bochs-p4-smp-2.6.9-win32.zip	2017-04-09	2.8 MB	19
Bochs-2.6.9.exe	2017-04-09	5.3 MB	498
bochs-2.6.9-msvc-src.zip	2017-04-09	5.7 MB	28
bochs-2.6.9-1.x86_64.rpm	2017-04-09	3.3 MB	6
README-bochs-2.6.9	2017-04-09	3.6 kB	10
bochs-2.6.9-1.src.rpm	2017-04-09	5.1 MB	3
bochs-2.6.9.tar.gz	2017-04-09	5.2 MB	158
Totals: 9 Items		33.7 MB	943

目前的最新版本好像是 2.6.9，我们下载它的源码来进行安装。

在进入下载目录，打开终端输入：tar zxvf bochs-2.6.9.tar.gz 就可以解压了。



接下来 cd　bochs-2.6.9 进入到目录里面。开始 configure、 make、 make install 三步曲。

1->配置，执行 configure，直接把这下面的内容复制进去就好了。注意，--prefix 选项是代表安装路径，你选择合适的路径进行安装。--enable 选项是一些调试器的打开与否。--with 是表示使用哪些图形库。

```
./configure \
--prefix=/your_path/bochs \
--enable-debugger \
--enable-disasm \
--enable-iodebug \
--enable-x86-debugger \
--with-x \
--with-x11
```

```
huzicheng@localhost:~/下载/bochs-2.6.9
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
[huzicheng@localhost bochs-2.6.9]$ ./configure --prefix=/your_path/bochs --enabl
e-debugger --enable-disasm --enable-iodebug --enable-x86-debugger --with-x --wit
h-x11
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking target system type... x86_64-unknown-linux-gnu
checking if you are configuring for another platform... no
checking for standard CFLAGS on this platform...
checking for gcc... gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... no
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether gcc accepts -g... yes
checking for gcc option to accept ISO C89... none needed
checking for g++... g++
checking whether we are using the GNU C++ compiler... yes
checking whether g++ accepts -g... yes
checking whether make sets $(MAKE)... yes
checking for a sed that does not truncate output... /usr/bin/sed
checking for grep that handles long lines and -e... /usr/bin/grep
checking for egrep... /usr/bin/grep -E
```

Configure 之后，会生成 Makefile 文件，我们需要根据这个文件来编译源码。

2->make，编译源码

输入 make 就好了

```
huzicheng@localhost:~/下载/bochs-2.6.9
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
config.status: creating cpu/cpudb/Makefile
config.status: creating cpu/fpu/Makefile
config.status: creating memory/Makefile
config.status: creating gui/Makefile
config.status: creating disasm/Makefile
config.status: creating instrument/stubs/Makefile
config.status: creating misc/Makefile
config.status: creating doc/docbook/Makefile
config.status: creating build/linux/bochs-dlx
config.status: creating bxversion.h
config.status: creating bxversion.rc
config.status: creating build/macosx/Info.plist
config.status: creating build/win32/nsis/Makefile
config.status: creating build/win32/nsis/bochs.nsi
config.status: creating host/linux/pcidev/Makefile
config.status: creating config.h
config.status: creating ltdlconf.h
config.status: ltdlconf.h is unchanged
[huzicheng@localhost bochs-2.6.9]$ make
cd iodev && \
make  libiodev.a
make[1]: 进入目录 “/home/huzicheng/下载/bochs-2.6.9/iodev”
g++ -c  -I.. -I./.. -I../instrument/stubs -I./../instrument/stubs -g -O2 -D_FILE
 OFFSET_BITS=64 -D_LARGE_FILES -pthread    devices.cc -o devices.o
```

Make 就会编译源代码。编译完成之后，需要把编译的安装到我们的系统中，输入 make install，主要要是 root 权限才可以安装，你必须切换到 root 才可以。

```
for i in /your_path/bochs/bin; do mkdir -p $i && test -d $i && test -w $i; done
mkdir: 无法创建目录"/your_path": 权限不够
```

没有使用 root 权限就会是这个样子。

```
[huzicheng@localhost bochs-2.6.9]$ su -
密码：
上一次登录：四 11月 14 15:37:49 CST 2019pts/0 上
[root@localhost ~]#
```

在这里，我切换到了 root，不同的系统切换方式可能不一样，你需要自己去查找。

```
[root@localhost ~]# cd /home/huzicheng/下载/bochs-2.6.9
```

接下来，进入到 bochs 目录，输入 make install 安装 bochs。

```
root@localhost:/home/huzicheng/下载/bochs-2.6.9
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
 -V, --version  输出版本信息并退出

更多信息请参阅 su(1)。
[huzicheng@localhost bochs-2.6.9]$ su -
密码：
上一次登录：四 11月 14 15:37:49 CST 2019pts/0 上
[root@localhost ~]# make install
make: *** 没有规则可以创建目标“install”。 停止。
[root@localhost ~]# cd /user
-bash: cd: /user: 没有那个文件或目录
[root@localhost ~]# cd /home/huzicheng/下载/bochs-2.6.9
[root@localhost bochs-2.6.9]# make install
cd iodev && \
make  libiodev.a
make[1]: 进入目录 “/home/huzicheng/下载/bochs-2.6.9/iodev”
make[1]: “libiodev.a”是最新的。
make[1]: 离开目录 “/home/huzicheng/下载/bochs-2.6.9/iodev”
echo done
done
cd iodev/display && \
make  libdisplay.a
make[1]: 进入目录 “/home/huzicheng/下载/bochs-2.6.9/iodev/display”
make[1]: “libdisplay.a”是最新的。
make[1]: 离开目录 “/home/huzicheng/下载/bochs-2.6.9/iodev/display”
```

```
root@localhost:/home/huzicheng/下载/bochs-2.6.9
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
mkdir -p /your_path/bochs/share/man/man5
for i in bochs bximage bochs-dlx; do cat ./doc/man/$i.1 | sed 's/@version@/2.6.9
/g' | gzip -9 -c >  /your_path/bochs/share/man/man1/$i.1.gz; chmod 644 /your_pat
h/bochs/share/man/man1/$i.1.gz; done
for i in bochsrc; do cat ./doc/man/$i.5 | gzip -9 -c >  /your_path/bochs/share/m
an/man5/$i.5.gz; chmod 644 /your_path/bochs/share/man/man5/$i.5.gz; done
for i in /your_path/bochs/share/bochs;  do mkdir -p $i && test -d $i && test -w
$i; done
for i in bios/BIOS-bochs-* bios/VGABIOS* bios/SeaBIOS* bios/bios.bin-* ; do if t
est -f $i; then install -m 644 $i /your_path/bochs/share/bochs; else install -m
644 ./$i /your_path/bochs/share/bochs; fi; done
mkdir /your_path/bochs/share/bochs/keymaps
for i in ./gui/keymaps/*.map; do install -m 644 $i /your_path/bochs/share/bochs/
keymaps/; done
for i in /your_path/bochs/share/doc/bochs; do mkdir -p $i && test -d $i && test
-w $i; done
for i in CHANGES COPYING LICENSE README TODO misc/slirp.conf; do if test -f $i;
then install -m 644 $i /your_path/bochs/share/doc/bochs; else install -m 644 ./$
i /your_path/bochs/share/doc/bochs; fi; done
rm -f /your_path/bochs/share/doc/bochs/README
cat ./build/linux/README.linux-binary ./README > /your_path/bochs/share/doc/boch
s/README
install -m 644 ./.bochsrc /your_path/bochs/share/doc/bochs/bochsrc-sample.txt
[root@localhost bochs-2.6.9]#
```

现在你的 bochs 就安装好了。如果安装失败，请自行上网搜索。（ps：我也是在网上找的方法）

接下来，需要配置环境变量，只有这样，才能再任意路径执行 bochs 虚拟机。执行 vim ~/.bash_profile 修改文件中 PATH 一行，PATH=$PATH:$HOME/bin 之后添加 （注意以冒号分隔），wq 保存文件并退出，执行 source ~/.bash_profile 使其生效，这种方法只对当前登陆用户生效。

```
root@localhost:/home/huzicheng
文件(F) 编辑(E) 查看(V) 搜索(S) 终端(T) 帮助(H)
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin:/usr/bochs/bin

export PATH
~
~
~
~
~
~
~
~
~
~
"~/.bash_profile" 12L, 191C                    12,1          全部
```

```
[root@localhost huzicheng]# vim ~/.bash_profile
[root@localhost huzicheng]# source ~/.bash_profile
[root@localhost huzicheng]# bochs
========================================================================
                       Bochs x86 Emulator 2.6.9
               Built from SVN snapshot on April 9, 2017
                  Compiled on Nov 15 2019 at 20:24:00
========================================================================
00000000000i[      ] BXSHARE not set. using compile time default '/usr/bochs/sh
are/bochs'
------------------------------
Bochs Configuration: Main Menu
------------------------------

This is the Bochs Configuration Interface, where you can describe the
machine that you want to simulate.  Bochs has already searched for a
```

在执行配置过后，输入 bochs 就可以看到 bochs 运行起来了。

想要 bochs 运行我们的系统，需要配置 bochs，才能使它运行。在 linux 下面，需要修改的是源码中的 bochs/bochsrc.linux 这个配置文件。

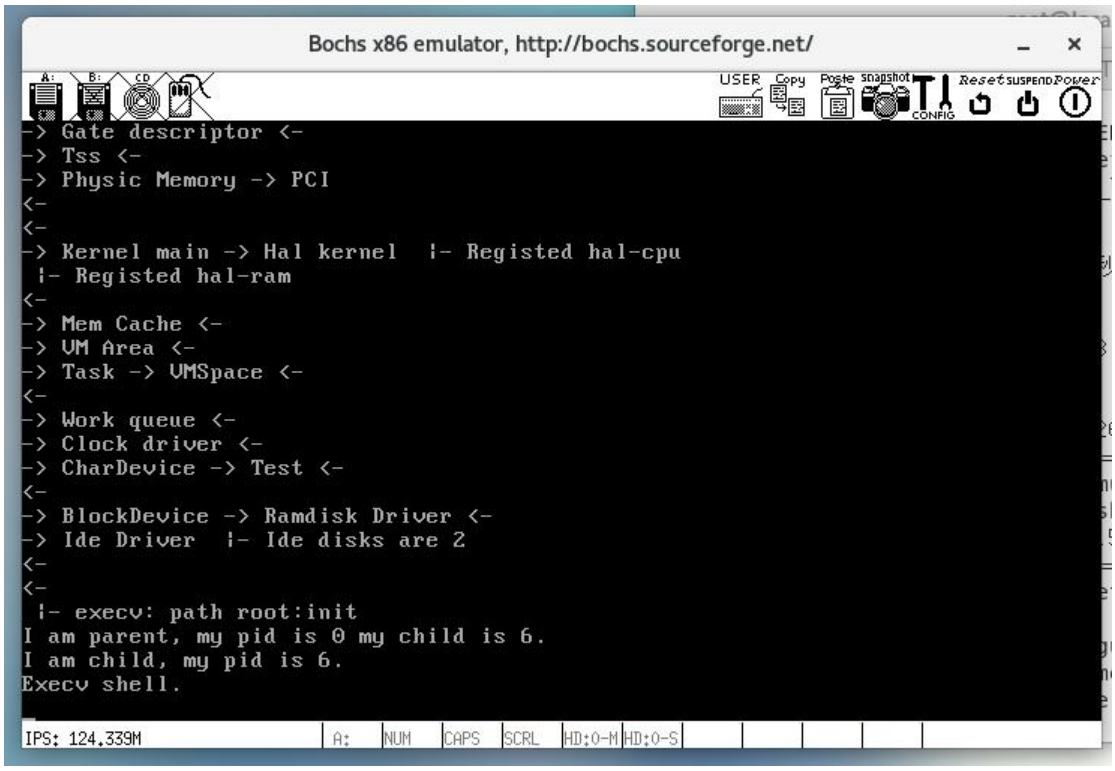
```
[root@localhost xbook]# ls
bin    doc  launch_cmd.bat     LICENSE    OWNER.md    src
bochs  img  launch_vscode.bat  Makefile   README.md
[root@localhost xbook]# cd bochs
[root@localhost bochs]# ls
bochsout.txt  bochsrc.linux  bochsrc.win
[root@localhost bochs]# vim bochsrc.linux
```

```
root@localhost:/usr/github/xbook/bochs
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
################################################################
# bochsrc.linux file for Book kernel disk image on linux
# develop environment.
################################################################

# how much memory the emulated machine will have
megs: 256

# filename of ROM images
romimage: file=/usr/bochs/share/bochs/BIOS-bochs-latest
vgaromimage: file=/usr/bochs/share/bochs/VGABIOS-lgpl-latest

#cpu: count=1, ips=10000000

# what disk images will be used
floppya: 1_44=../img/a.img, status=inserted

# hard disk
ata0: enabled=1, ioaddr1=0x1f0, ioaddr2=0x3f0, irq=14
ata0-master: type=disk, path="../img/c.img", cylinders=20, heads=16, spt=63
ata0-slave: type=disk, path="../img/d.img", cylinders=20, heads=16, spt=63

# choose the boot disk.
                                                        8,0-1         顶端
```

这里把 file 的路径修改成你系统的 bochs 里面的文件路径就可以了。

现在回到 master 目录下面，然后输入 make bochs_linux 就可以运行了。

```
root@localhost:/usr/github/xbook
文件(F)  编辑(E)  查看(V)  搜索(S)  终端(T)  帮助(H)
4096字节(4.1 kB)已复制，0.000152805 秒，26.8 MB/秒
记录了346+1 的读入
记录了346+1 的写出
177640字节(178 kB)已复制，0.000797636 秒，223 MB/秒
========================================================================
                        Bochs x86 Emulator 2.6.9
                Built from SVN snapshot on April 9, 2017
                   Compiled on Nov 15 2019 at 20:24:00
========================================================================
00000000000i[      ] BXSHARE not set. using compile time default '/usr/bochs/sh
are/bochs'
00000000000i[      ] reading configuration from ../bochs/bochsrc.linux
00000000000i[      ] installing x module as the Bochs GUI
00000000000i[      ] using log file ../bochs/bochsout.txt
========================================================================
Bochs is exiting with the following message:
[XGUI  ] POWER button turned off.
========================================================================
make[1]: *** [bochs_linux] 错误 1
make: *** [bochs_linux] 错误 2
[root@localhost xbook]# ls
bin    doc  launch_cmd.bat     LICENSE   OWNER.md   src
bochs  img  launch_vscode.bat  Makefile  README.md
[root@localhost xbook]# make bochs_linux
```

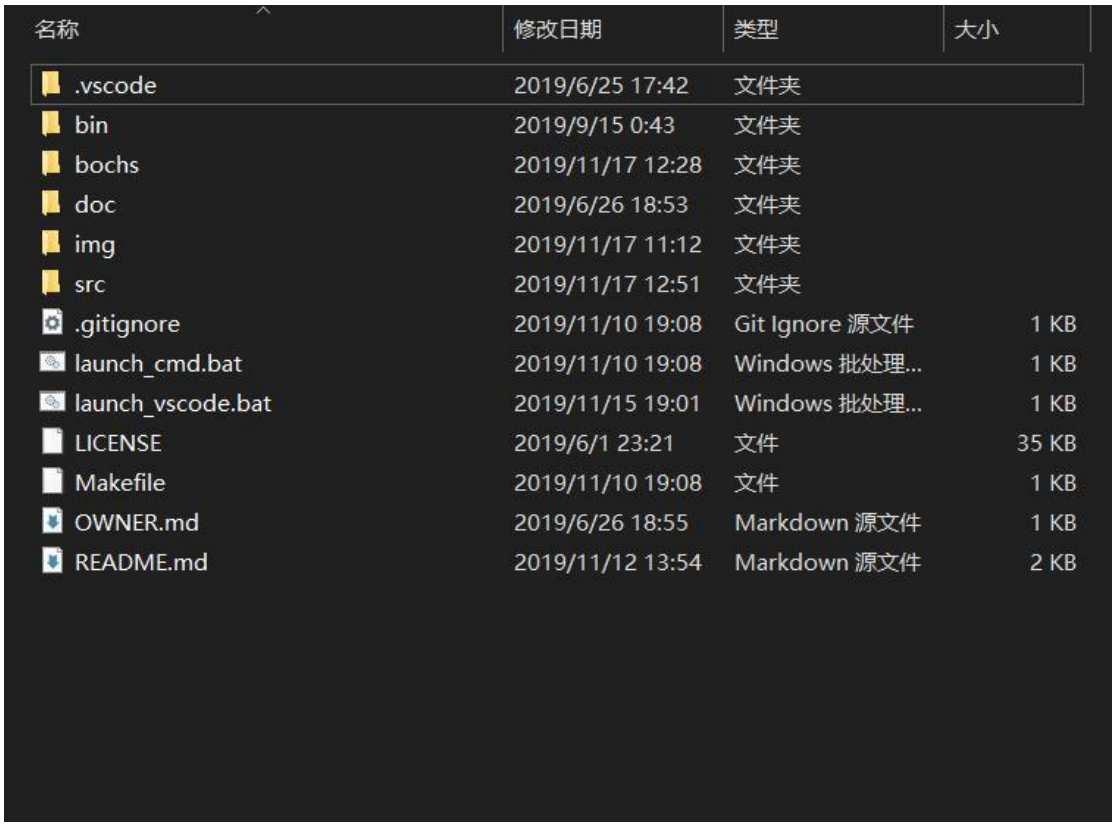


1.5 MacOS 下的环境搭建？？？

我没有苹果电脑，也不会使用，你可以根据前面两种环境的方法自己琢磨！

第二章 源码目录的解析

2.1 目录总览



名称	修改日期	类型	大小
.vscode	2019/6/25 17:42	文件夹	
bin	2019/9/15 0:43	文件夹	
bochs	2019/11/17 12:28	文件夹	
doc	2019/6/26 18:53	文件夹	
img	2019/11/17 11:12	文件夹	
src	2019/11/17 12:51	文件夹	
.gitignore	2019/11/10 19:08	Git Ignore 源文件	1 KB
launch_cmd.bat	2019/11/10 19:08	Windows 批处理...	1 KB
launch_vscode.bat	2019/11/15 19:01	Windows 批处理...	1 KB
LICENSE	2019/6/1 23:21	文件	35 KB
Makefile	2019/11/10 19:08	文件	1 KB
OWNER.md	2019/6/26 18:55	Markdown 源文件	1 KB
README.md	2019/11/12 13:54	Markdown 源文件	2 KB

总目录

为了便于开发，操作系统不仅仅只有操作系统的代码，还有一些其它内容在这里面。

.vscode: vscode 编辑器的一些设定，不用管，用 vscode 编辑器打开这个目录会自动生成。

Bin: 可执行文件，例如 init，shell，test 这些程序放在里面。

Bochs: bochs 虚拟机的配置文件，想要用 bochs 虚拟机调试，就可以修改使用这里面的配置文件。

Doc: 帮助文档。目前只有命名规范。

Img: 虚拟磁盘镜像，软盘镜像，硬盘镜像。

Src: 操作系统源码

.gitignore:git 上传到 github 时会忽略那些格式的文件，一般是生成的中间对象文件.o 这种。

Launch_cmd.bat: 在 windows 开发环境下快速打开控制台

Launch_vscode.bat:在 windows 开发环境下快速打开 VS code

LICENSE: 开源协议，GPL V3

Makefile: 进入 src 目录编译系统的 makefile，原本是在 src 中才可以编译，现在在此目录下也可以编译。

OWNER.md: 所有者，即开发者，只要你的贡献到达一定程度，你的名字就会留在这里面。

README.md: 整个项目的信息介绍。

2.1 Img 目录

核心在于 img 目录和 src 目录，所以这里只对这 2 个目录进行简单的介绍。其它目录可

自行查看。

名称	修改日期	类型	大小
a.img	2019/11/17 12:51	光盘映像文件	1,440 KB
c.img	2019/11/15 13:36	光盘映像文件	10,080 KB
c.vhd	2019/9/11 21:57	Virtual Hard Disk	10,241 KB
d.img	2019/10/26 16:51	光盘映像文件	10,080 KB
raw.img	2019/1/7 1:59	光盘映像文件	10,080 KB
raw.vhd	2019/9/17 21:52	Virtual Hard Disk	10,241 KB
README.md	2019/5/29 20:49	Markdown 源文件	1 KB

Img 目录

Img 目录存放的是虚拟磁盘镜像，是放在虚拟机里面运行的。如果说虚拟机是对物理机的模拟，那么虚拟磁盘就是对物理磁盘的模拟。

磁盘有硬盘，软盘，光盘等。在这里，我们选择了软盘+硬盘的组合形式。也就是说，操作系统放到了软盘上，引导的时候选择软盘引导，通过软盘上的 boot 加载 loader，然后通过 loader 加载 kernel，这样，我们的操作系统就可以跑起来了。这些都放在了软盘上。我们后面需要在硬盘上建立一个文件系统，这样，硬盘上就可以储存文件，我们系统也可以访问这上面的文件。

其中 a.img 就是软盘镜像，c.img 就是硬盘镜像。Img 格式硬盘适用于 Qemu 和 Bochs 虚拟机。传统地，我们电脑上可以挂在 4 个硬盘，通过 2 个通道来连接，每个通道上有 2 个硬盘。也就是说，我们可以配置 4 个硬盘。4 个硬盘分别是主通道主盘，主通道从盘，从通道主盘，从通道从盘。而这里的 c.img 就是对应着主通道主盘，这个需要在配置的时候加以注意。除此之外，还有一个 c.vhd，这个是啥？其实也是硬盘，只是说，它是 virtual box 虚拟机下面的硬盘，不同的虚拟机支持的硬盘格式不一样，但是软盘却都可以是.img 格式的，所以 a.img 是各大虚拟机直接通用的。

除此之外，还有 d.img，它是主通道从磁盘，我们配置的时候尽量按着这个顺序来配置，你还可以添加 e,f 盘。

而 raw.img 和 raw.vhd 只是一个空的硬盘，没有被使用过，用来作为一个副本，有时候 c.img 写入数据后，内容混乱，你希望重新管理磁盘，就可以通过这 2 个磁盘创建一个副本，然后删掉原来的 c.img，把新建的副本名字修改成 c.img，这样就新的皇帝登基了。

2.2 Src 目录

总览：

可以发现，这个目录下面也有.vscode，可以不用管它，我都有想要删除它的冲动，但是想了一下，用 vscode 开发带来的便捷，还是保留吧。

arch（处理器架构有关，目前只有 x86-i386 处理器）

drivers（支持的驱动程序）

fs（支持的文件系统，目前处于测试阶段）

hal（简单的硬件抽象接口）

include（头文件，所有头文件都在这里面）

init（内核初始化的地方）

ipc（进程间通信，目前只是空目录）

kernel（内核的核心部分）

mm（内存管理）
share（共享库，内核和用户都可以使用）
user（用户程序库，给用户准备的系统调用接口）
cmd.bat（用于在 windows 环境下快速打开控制台）
FINISHED.md（实现了哪些功能，会在这里记录）
Makefile（代码编译的选项和命令）
Makefile.build（代码编译的规则）
README.md（目录介绍文件）

名称	修改日期	类型	大小
.vscode	2019/11/7 21:50	文件夹	
arch	2019/11/3 17:24	文件夹	
drivers	2019/11/17 17:27	文件夹	
fs	2019/11/17 17:27	文件夹	
hal	2019/11/17 17:27	文件夹	
include	2019/11/17 17:20	文件夹	
init	2019/11/17 17:27	文件夹	
ipc	2019/11/2 12:20	文件夹	
kernel	2019/11/17 17:27	文件夹	
mm	2019/11/17 17:27	文件夹	
share	2019/11/17 17:27	文件夹	
user	2019/11/17 17:27	文件夹	
cmd.bat	2019/5/29 20:49	Windows 批处理...	1 KB
FINISH.md	2019/11/17 0:50	Markdown 源文件	2 KB
makefile	2019/11/17 17:19	文件	3 KB
Makefile.build	2019/11/15 23:32	BUILD 文件	2 KB
README.md	2019/6/26 18:32	Markdown 源文件	1 KB

2.2.1 arch 目录

目前只有 x86-i386 处理器，所以只有一个目录，以后如果支持跨平台，这里面应该会有很多新的内容。

名称	修改日期	类型	大小
x86	2019/11/17 17:27	文件夹	

名称	修改日期	类型	大小
boot	2019/11/17 17:27	文件夹	
bus	2019/11/17 17:27	文件夹	
include	2019/11/3 17:24	文件夹	
kernel	2019/11/17 17:27	文件夹	
mm	2019/11/17 17:27	文件夹	
makefile	2019/11/12 13:45	文件	1 KB
MEMORY.md	2019/7/30 14:46	Markdown 源文件	4 KB

进入 x86 目录后，这里面有 boot（引导程序），bus（系统总线），include（头文件），kernel

（处理器内核），mm（与平台相关的内存管理），还有随处可见的 makefile，以及一个 MEMORY.md（物理内存分布情况）

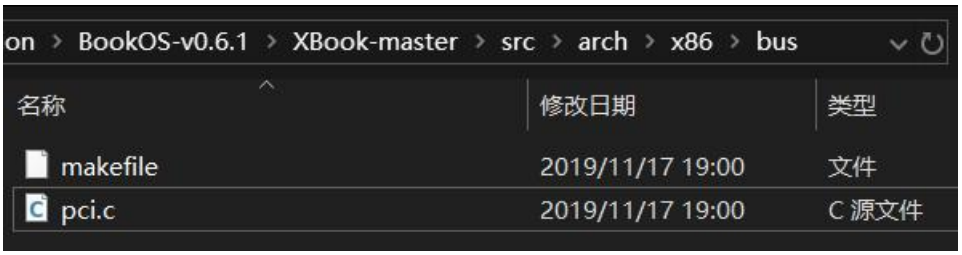
名称	修改日期	类型	大小
boot.asm	2019/6/2 15:06	ASM 文件	2 KB
loader.asm	2019/11/10 18:42	ASM 文件	14 KB
makefile	2019/11/12 13:45	文件	1 KB

src/arch/x86/boot/

boot.asm 是引导程序，是软盘引导，它会把 loader 程序从磁盘读入内存，并跳转到 loader 中去。

loader.asm 是加载器，主要做的事情是从磁盘加载内核文件到内存，并做一些实模式下面的初始化，然后切换到保护模式，并开启分页模式，最后跳转到内核的入口执行。（由于开启了分页模式，此时内核的地址就是虚拟地址）

makefile 就是把 boot.asm 和 loader.asm 编译成二进制文件。

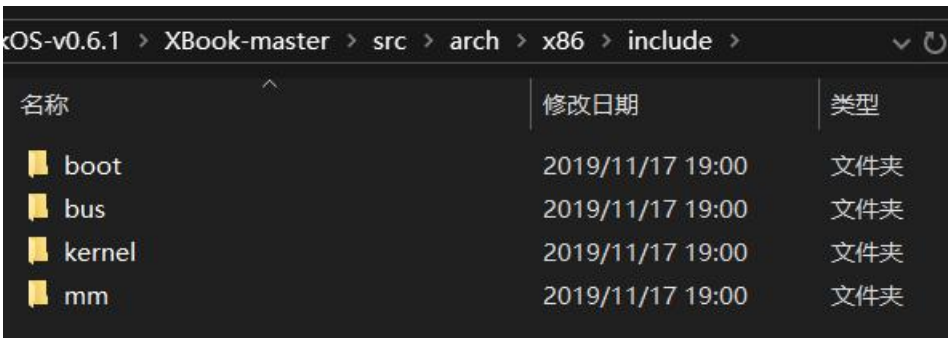


on > BookOS-v0.6.1 > XBook-master > src > arch > x86 > bus

名称	修改日期	类型
makefile	2019/11/17 19:00	文件
pci.c	2019/11/17 19:00	C 源文件

src/arch/x86/bus/

pci.c 是 PCI（PCI 是 Peripheral Component Interconnect(外设部件互连标准)的缩写，它是目前个人电脑中使用最为广泛的接口，几乎所有的主板产品上都带有这种插槽。）总线的一些内容，获取 PCI 设备的接口就在此处。

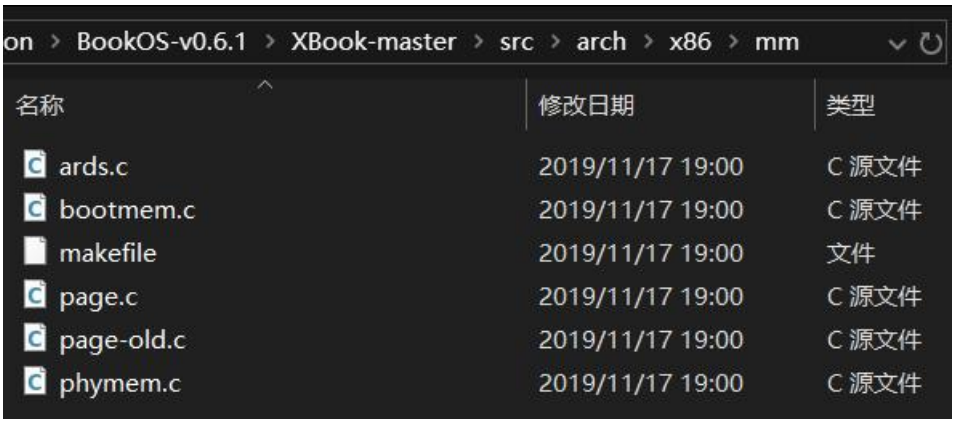


kOS-v0.6.1 > XBook-master > src > arch > x86 > include >

名称	修改日期	类型
boot	2019/11/17 19:00	文件夹
bus	2019/11/17 19:00	文件夹
kernel	2019/11/17 19:00	文件夹
mm	2019/11/17 19:00	文件夹

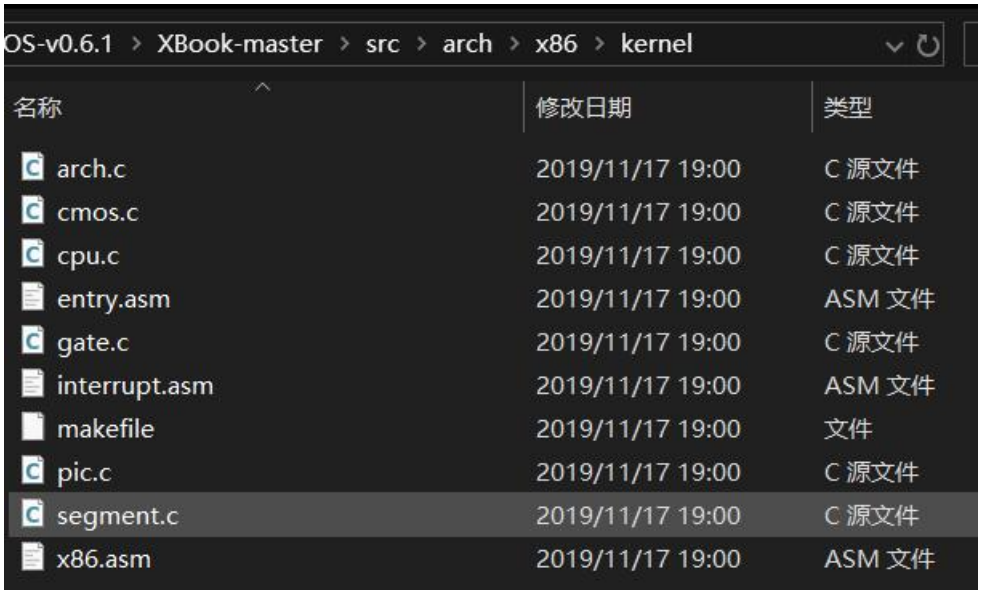
src/arch/x86/include/

这里面包含的是一些头文件，以后与平台相关的头文件都放到这个目录下面，方便引用，使结构也更加清晰。



src/arch/x86/mm /

mm 即内存管理的意思，与初始的内存管理相关的内容都在这里面。Ards.c 用于获取物理内存，bootmem.c 用于最开始连续的内存的分配，page.c 是分页管理，page-old.c 是老版本的分页管理方法，这里没有用上，但保留了下来，phymem.c 用于管理物理内存的分配。



src/arch/x86/kernel /

kernel 目录是某个平台的核心部分。Arch.c 是与平台相关的内容的初始化的地方，从 loader 加载完内核之后通过 entry.asm 的内容过后，就会跳到 arch.c 里面来做平台的初始化操作。Cmos.c 是用于获取时间的，cpu.c 是与 cpu 相关的内容，entry.asm 应该是内核的最开始的地方，是汇编的内容，通过这里跳转到 arch.c 里面。Gate.c 是与门相关的，如中断门，陷阱门，任务门等。Interrupt.asm 是与中断相关的汇编部分。Pic.c 是用于初始化 IRQ 中断的，也就是说 IRQ0-15 怎么与硬件连接，就会在这里面设定。Segment.c 是内存段的初始化与设定的地方，比如，内核的代码段，数据段，Tss 段，用户的代码段，数据段，都会在这里面初始化。X86.asm 就是与处理器相关的一些指令的封装，比如 In, Out, cli, sti 之类的。

如果以后要添加与某个特别的平台相关的内容，就需要在这个目录文件中添加对应的文件，接下来看一下怎么添加一个新文件。

这里以 src/arch/x86/kernel /目录为例子，先打开它的 makefile，会发现，是吧要添加的文件想要生成的对象文件直接加上去就行了。

```
makefile
obj-y   += entry.o
obj-y   += arch.o
obj-y   += x86.o
obj-y   += segment.o
obj-y   += gate.o
obj-y   += interrupt.o
obj-y   += pic.o
obj-y   += cpu.o
obj-y   += cmos.o
```

src/arch/x86/kernel /makefile

如果要添加一个 test.c 的文件，那么只需要在 makefile 里面添加其对应的对象文件即可。

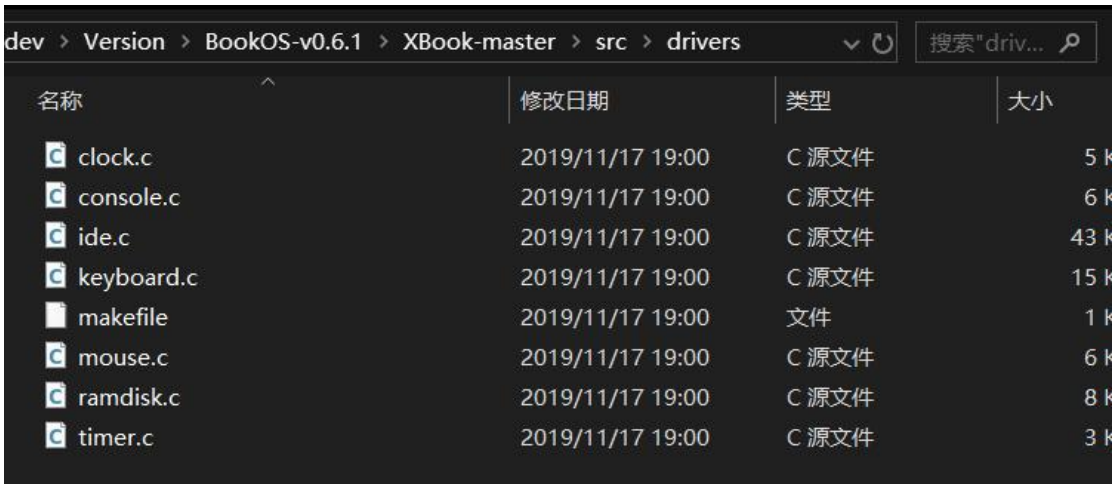
名称	修改日期	类型	大小
arch.c	2019/11/17 19:00	C 源文件	2 K
cmos.c	2019/11/17 19:00	C 源文件	2 K
cpu.c	2019/11/17 19:00	C 源文件	3 K
entry.asm	2019/11/17 19:00	ASM 文件	2 K
gate.c	2019/11/17 19:00	C 源文件	15 K
interrupt.asm	2019/11/17 19:00	ASM 文件	6 K
makefile	2019/11/23 11:47	文件	1 K
pic.c	2019/11/17 19:00	C 源文件	3 K
segment.c	2019/11/17 19:00	C 源文件	2 K
test.c	2019/11/17 19:00	C 源文件	3 K
x86.asm	2019/11/17 19:00	ASM 文件	4 K

```
makefile
obj-y   += entry.o
obj-y   += arch.o
obj-y   += x86.o
obj-y   += segment.o
obj-y   += gate.o
obj-y   += interrupt.o
obj-y   += pic.o
obj-y   += cpu.o
obj-y   += cmos.o
obj-y   += test.o
```

是不是很方便呢？如果是汇编文件的话，也是添加对应的.o 对象文件名就可以了。

2.2.2 drivers 目录

顾名思义，就是驱动的意思，也就是说，此目录下面存放的是一些硬件的驱动程序。每一个文件保存了一个驱动程序，如果要添加新的驱动程序，尽量保证单个文件存放单个驱动。



src/drivers

可以看到有 clock（时钟驱动），console（控制台驱动），ide（IDE 硬盘驱动），keyboard（键盘驱动），mouse（鼠标驱动-未实现），ramdisk（内存磁盘驱动），timer（定时器驱动）。

驱动程序就不详解了，如果需要添加一个新的驱动的话，那么就需要在这个目录添加一个对应的驱动程序。

2.2.3 fs 目录

文件系统目录，用于存放文件系统代码，目前新的文件系统还没有提上日程，在这个版本中，暂不讲述，待后面实现了之后，再进行一个简单的讲解。

2.2.4 hal 目录

HAL(Hardware Abstraction Layer)，硬件抽象层是位于操作系统 内核与硬件电路之间的接口层，其目的在于将硬件抽象化。最开始想的是把所有与硬件相关的内容都进行抽象，但是发现在有的地方，不太适合，所以不能全部都抽象，通过一些思考，以及实践，只保留 CPU 和 RAM 内存的一些抽象。

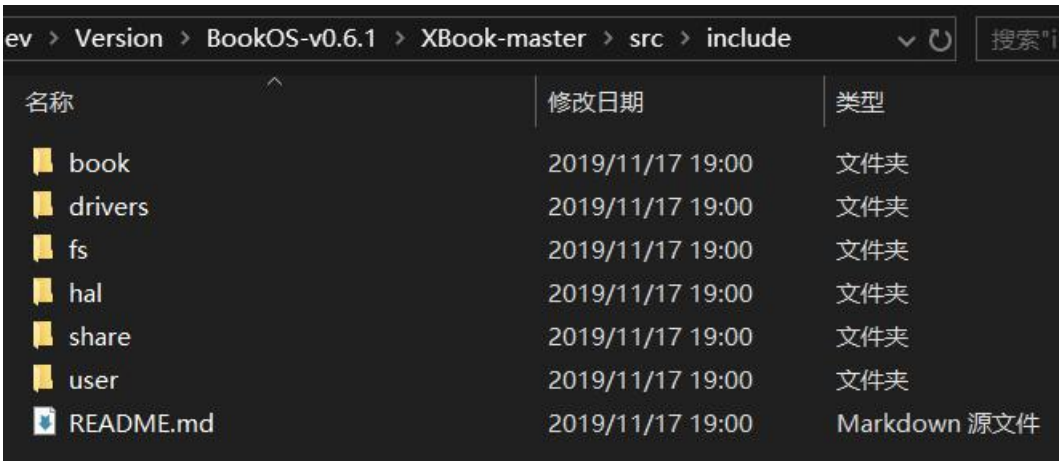
src/hal

核心在于，把底层硬件接口进行抽象化，总结规律，提供一套系统的，简洁的接口给调用者。

2.2.5 include 目录

只要遇到 include 的地方，我们就知道是与头文件相关的，内核的头文件是有 book（由于操作系统是 BookOS，所以核心部分也就用 book，这个是学得 linux 的，哈哈），drivers（驱动程序的头文件），fs（文件系统的头文件），hal（硬件抽象的头文件），share（系统和用户共享的头文件，可以在系统中使用，也可以在用户程序中使用），user（用户程序库的头文

件）组成。

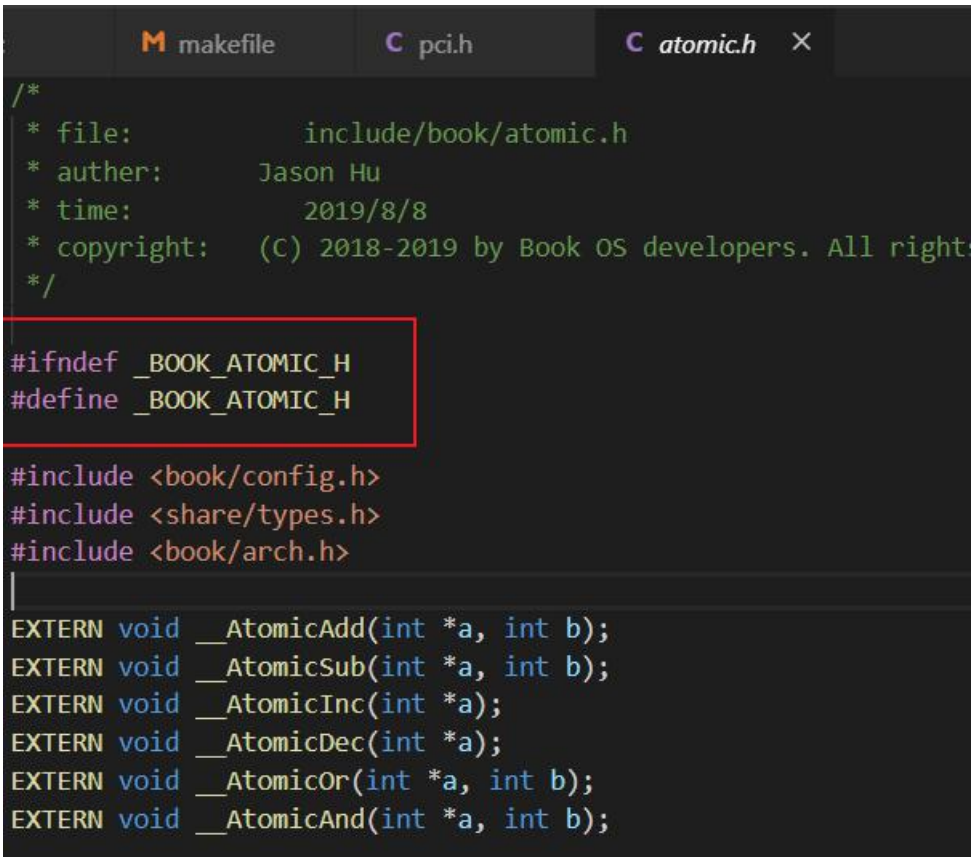


Src/include

在头文件的定义中，我们使用一个宏定义来保证它不会重复包含一个头文件。；例如：A 文件里面包含了 B 头文件，C 头文件里面包含了 B 头文件，当 A 头文件再包含 C 头文件的时候，B 文件会再次被加入到 A 文件,那么此时就会出现重复导入的问题。使用宏来解决。

```
#ifndef 头文件名
#define 头文件名
...头文件的内容添加到此处
#endif 结束当前宏
```

也就是说，当没有定义某个头文件的时候，才会定义它的内容，如果已经有某个头文件的内容之后，就不会再次导入，那么就解决了头文件重复导入的问题。

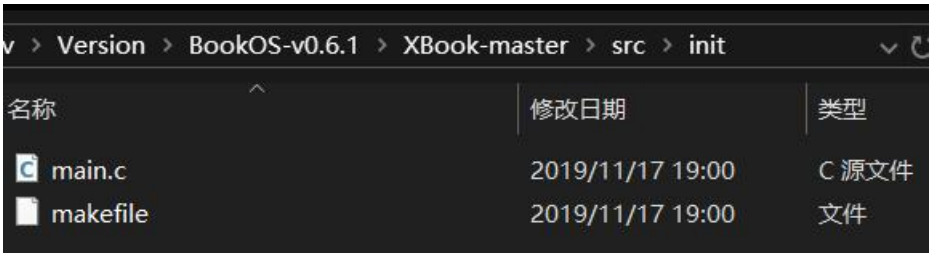


运用宏来包含头文件的内容，可以解决重复定义头文件的问题

对于这个头文件名的一个书写，统一一个格式：_XXX_XXX_H, _目录名字或属于某个领域的名字_头文件内容名_H.

例如：_BOOK_ATOMIC_H， _DRIVER_ATA_H， _SHARE_CONST_H， _USER_LIB_STDIO_H

2.2.6 init 目录



Src/init

这里面只有一个 main.c，这是内核的主程序，初始化的内容就会在里面。初始化虚拟内存管理，中断管理，初始化多任务，初始化驱动之类的。也就是说，操作系统内核的主要部分的初始化，与平台无关的内容。初始化完成之后，才能使用某一个模块（此模块非 linux 模块化的模块，只是说，一个独立的内容）的内容。注意先后顺序，如果需要调用其它模块的内容，得保证其它模块得先初始化。

如果是需要初始化使用中断的内容，就得把它添加到初始化中断之后，或者是打开中断之后才可以，不然你的驱动程序，就不能响应中断，或者是不能正确调用中断注册模块。

```
/* 初始化IRQ描述结构 */
InitIrqDescription();

/* 初始化软中断机制 */
InitSoftirq();

// 初始化多任务
InitTasks();

// 初始化多任务后，初始化工作队列
InitWorkQueue();

// 打开中断标志
EnableInterrupt();

//初始化时钟驱动
InitClockDriver();

/* 初始化ramdisk */
//InitRamdiskDriver();

/* 初始化字符设备 */
InitCharDevice();

/* 初始化块设备 */
InitBlockDevice();
```

2.2.7 kernel 目录

dev > Version > BookOS-v0.6.1 > XBook-master > src > kernel

名称	修改日期	类型
block	2019/11/23 11:22	文件夹
char	2019/11/23 11:22	文件夹
atomic.asm	2019/11/17 19:00	ASM 文件
bitmap.c	2019/11/17 19:00	C 源文件
debug.c	2019/11/17 19:00	C 源文件
device.c	2019/11/17 19:00	C 源文件
exit_wait.c	2019/11/17 19:00	C 源文件
hal.c	2019/11/17 19:00	C 源文件
interrupt.c	2019/11/17 19:00	C 源文件
ioqueue.c	2019/11/17 19:00	C 源文件
iostream.c	2019/11/17 19:00	C 源文件
makefile	2019/11/17 19:00	文件
mutex.c	2019/11/17 19:00	C 源文件
schedule.c	2019/11/17 19:00	C 源文件
sleep_wakeup.c	2019/11/17 19:00	C 源文件
softirq.c	2019/11/17 19:00	C 源文件
spinlock.c	2019/11/17 19:00	C 源文件
switch.asm	2019/11/17 19:00	ASM 文件
synclock.c	2019/11/17 19:00	C 源文件
syscall.c	2019/11/17 19:00	C 源文件
task.c	2019/11/17 19:00	C 源文件
workqueue.c	2019/11/17 19:00	C 源文件

Src/kernel

内核的核心目录，这里面都是最精华的地方。

Block 目录：块设备框架的目录，里面存放了块设备框架的内容。

sion > BookOS-v0.6.1 > XBook-master > src > kernel > block

名称	修改日期	类型	大小
blkdev.c	2019/11/17 19:00	C 源文件	5 K
block.c	2019/11/17 19:00	C 源文件	6 K
buffer.c	2019/11/17 19:00	C 源文件	9 K
disk.c	2019/11/17 19:00	C 源文件	5 K
elevator.c	2019/11/17 19:00	C 源文件	6 K
makefile	2019/11/17 19:00	文件	1 K
request.c	2019/11/17 19:00	C 源文件	9 K
requeue.c	2019/11/17 19:00	C 源文件	9 K

Src/kernel/block

Blkdev.c 是块设备的抽象的内容，管理每一个块设备。

block.c 是块设备初始化的地方，如果一个是被是块设备，那么，我们需要把块设备的驱动程序在这里面进行初始化。

Buffer.c 是块的缓冲区，当访问一个块的时候，会把块的内容暂时读取到内存中，当下次访问的时候，直接从内存获取就可以了，就不用再从磁盘读取，可以在整体上，提高磁盘访问的效率。当写入一个磁盘块到设备的时候，只是记录该块为“脏”，就是说有新的数据写入，需要同步到磁盘，但不会立即同步，会有一个后台服务的内核线程，定期唤醒来同步磁盘。也可以手动 Sync 内存中的数据到磁盘上。

Disk.c 是磁盘管理，用于描述一个磁盘设备的信息，以及对他进行管理。Disk 是一个磁盘对应一个，blkdev 是一个磁盘可以对于多个，也就是说 disk numbers <= blkdev number。

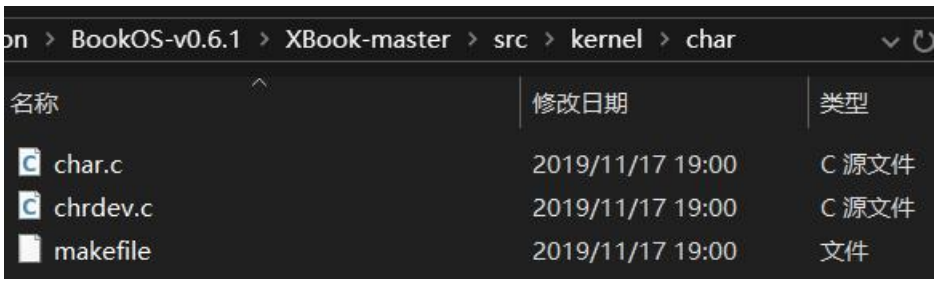
Elevator.c 是简单的电梯调度算法。用于对磁盘 io 进行排序，例如：请求一个磁盘读取，要操作 1,5,3 这 3 个扇区，从前往后移动磁盘的读取位置时，就会先读取 1，再去读取 5，再移动回来读取 3.如果有调度算法，那么，先把这个扇区排序成 1,3,5.那么，读取 1 之后，朝着 5 移动的过程中，读取了 3，最后读取 5，相比于第一种方法，那么就减少了回来再读取 3 这段时间，可以提升读写效率。但是，如果算法设计得有不好，那么，可能在进行算法排序的过程中，都把磁盘读取完了（比较夸张，但是有可能），这样就得不偿失。

Request.c 是请求。当需要发出请求信息的时候，会先把我们请求的内容封装成一个请求，传递给我们的磁盘驱动。驱动程序根据请求来做对应的处理。

Requeue.c 是请求队列。可能在处理一个请求的过程中，又产生了另外的请求，那么需要用一个队列来保存其它请求，当处理完当前请求过后，再从请求队列中获取下一个请求，知道处理完所有请求。

对于块设备驱动的开发，需要用到这些内容。开发者可以参考 ide 和 ramdisk 两个块设备驱动，来开发自己的块设备驱动。

Char 目录：字符设备框架的目录，里面存放了字符设备框架的内容。



Src/kernel/char

Char.c 是初始化字符驱动的地方，例如鼠标，键盘之类的驱动，都需要在此初始化。

Chrdev 是字符驱动抽象的内容，就是做了一个简单的接口。

回到 kernel 目录下面。

Atomic.asm 是原子操作函数处理的地方，常用语计数。

Bitmap.c 是位图操作，位图，是有 0 和 1 组成，某个位的状态只能是 0 或者 1，常用于把 0 表示为某个资源的空闲，把 1 表示为某个资源的分配。

Debug.c 是调试内核的一些函数，Panic 是系统运行出现严重故障，需要停在此处，相当于死亡。而 Spin 是类似的，只不过，Spin 常用于停在某处，而不是由于出错而停在某处，是开发者需要在某个地方调试，就停在那个地方，系统不继续往后面执行。Panic 用于出错的地方，而 Spin 用于调试的地方。

Device.c 就是设备的统一接口，在这里面为设备操作提供了一套接口，例如：DeviceOpen，DeviceClose,DeviceRead,DeviceWrite,DeviceIoctl 之类的。

Exit_wait.c 是用于进程的退出或者等待，好像只实现了 SysExit，退出运行，结束进程。

Hal.c 是硬件抽象层的核心，提供抽象的接口。

Interrupt.c 是用于中断管理的，提供中断注册和注销。

Ioqueue.c 是输入输出队列，运用同步锁来解决消费者和生产者问题。

Iostream.c 是输入输出流，用于把一段连续的内存虚拟成一个可以读写的字符流或者是块流。

Mutex.c 是互斥锁，用于进程间的互斥，保护临界资源。

Schedule.c 是进程间调度的实现。

Sleep_wakeup.c 是进程的休眠与唤醒。

Softirq.c 是软中断机制。

Spinlock.c 是自旋锁的实现。用于防止多个处理器竞争某个资源，由于我们目前是单处理器，所以说这个实现的内容意义不大。不过对于需要关闭或者打开中断的地方，可以使用它来进行中断状态的管理。保存状态，关闭中断->执行代码（临界区，不能被多个进程或者处理器访问）->恢复之前中断状态。

Switch.asm 是用于任务的上下文切换的汇编实现，上下文切换只能用汇编实现。

Synclock.c 同步锁，使用信号量来实现。同步的意思是，2 个任务做事情，需要按照某个顺序，任务 A 做完一半后，任务 B 才可以开始，任务 B 做完了过后，任务 A 才可以继续往后面执行。那么可以用同步锁来进行顺序的限定。

Syscall.c 是系统调用接口，把内核函数封装成系统调用，给用户一个接口，用户就可以通过调用这个接口，来调用系统函数。

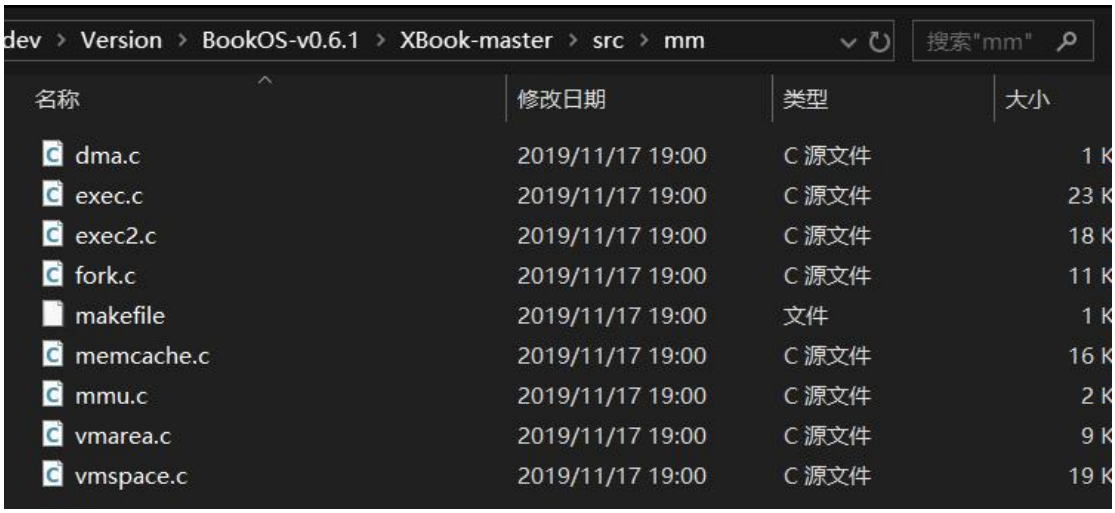
Task.c 是多进程和内核态多线程的实现。提供进程和 内核态线程的创建，退出等功能。

Workqueue.c 工作队列，是和中断相关的，用一个线程来做一个中断的处理。可以休眠。

内核这些内容，大多数都是与平台无关的，都是软件上的设计，所以说移植的时候，需要做对应的接口的对接。

2.2.7 mm 目录

Mm 不是美眉的意思，而是 memory management，内存管理的意思。



Src/mm

Dma.c 是直接内存访问，目前还没有实现。

Exec.c 是替换当前进程的镜像，相当于执行了一个新进程。由于文件系统尚未完善，该功能也是处于测试阶段。Exec2.c 是一个副本。

Fork.c 提供 fork 机制，主要做的就是创建一个当前进程的一个副本，相当于把自己复制一份，自己是父进程，创建了一个新的子进程，二者的区别在于 pid 不一样，其它内容都一样。如果和 exec 配合起来，就可以实现创建一个新的进程。

Memcache.c 提供动态内存管理，主要有 2 个接口，kmalloc 和 kfree，一个是分配内存，一个是释放内存。注意，kmalloc 默认最大只可以分配 256kb（可以在内核配置文件中配置，最大可以配置成 4MB），如果需要更大的空间，请使用 vmalloc。

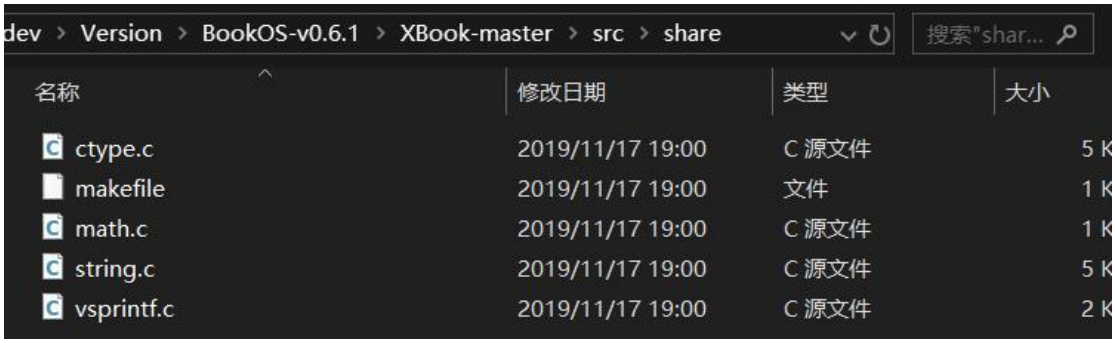
Mmu.c 是内存管理的入口，初始化的地方。

Vmarea.c 虚拟内存区域。用于提供大块内存分配，当 kmalloc 不能满足需要的时候，可以用 vmalloc 来分配，对应的释放 vfree。

Vmspace.c 是进程的虚拟地址空间，用户程序独有的区域，对于代码部分，用一个空间来描述，对于数据部分，也用一个空间来描述，与此同时，还有堆，栈也用空间来描述。可以实现访问虚拟地址缺页出错，并恢复的功能。

内存管理是操作系统的核心，因为所有的都围绕着内存来展开，需要提供一套比较稳定的管理机制，来使得系统更加问题。

2.2.7 share 目录



Src/share

共享目录，犹如现在的共享单车一样，大家都可以使用。这里的大家值得是系统和用户。

Ctype.c 实现的一些对字符的判断，例如：isspace，isalnum 之类的，我们系统也可以使用这些函数，用户程序也是可以使用浙西函数的。

Math.c 数学库，用于数学计算，目前只有 max,min,abs,pow，后面可以添加更多的内容进来。

String.c 用于字符串操作，以及内存的操作。Strcmp，strcpy，memset，memcpy 之类的。

Vsprintf.c 是格式字符串到一个缓冲区里面，可用于可变参数的格式化，常用于 printk 或者 printf 这种。

2.2.8 user 目录

用户的库函数，目前只有系统调用接口的封装。这些是用户程序可以直接调用的函数，可以借此来使用一些系统的功能。用户通过系统调用，从用户态陷入到内核态执行一些函数，然后再从内核态返回到用户态继续往后面执行。

sion > BookOS-v0.6.1 > XBook-master > src > user > lib

名称	修改日期	类型	大小
_brk.asm	2019/11/17 19:00	ASM 文件	1 K
_wait.asm	2019/11/17 19:00	ASM 文件	1 K
brk.c	2019/11/17 19:00	C 源文件	2 K
dir.asm	2019/11/17 19:00	ASM 文件	2 K
execv.asm	2019/11/17 19:00	ASM 文件	1 K
exit.asm	2019/11/17 19:00	ASM 文件	1 K
file.asm	2019/11/17 19:00	ASM 文件	2 K
fork.asm	2019/11/17 19:00	ASM 文件	1 K
getpid.asm	2019/11/17 19:00	ASM 文件	1 K
log.asm	2019/11/17 19:00	ASM 文件	1 K
makefile	2019/11/17 19:00	文件	1 K
malloc.c	2019/11/17 19:00	C 源文件	13 K
mmap.asm	2019/11/17 19:00	ASM 文件	1 K
msleep.asm	2019/11/17 19:00	ASM 文件	1 K
munmap.asm	2019/11/17 19:00	ASM 文件	1 K
printf.c	2019/11/17 19:00	C 源文件	1 K
sleep.asm	2019/11/17 19:00	ASM 文件	1 K

Src/user/lib

```
;----
;file:      user/lib/brk.asm
;auther:    Jason Hu
;time:      2019/8/10
;copyright: (C) 2018-2019 by Book OS developers. All rights reserved.
;----

[bits 32]
[section .text]

%include "syscall.inc"

; int _brk(void * addr);
global _brk
_brk:


    mov eax, SYS_BRK
    mov ebx, [esp + 4]
    int INT_VECTOR_SYS_CALL


    ret
```

这里，通过分析一下_brk.asm，来找出这些系统调用接口的规律。首先，需要导入

syscall.inc 的头文件，才能够使用对应的宏。_brk 是标签名，在这里用 global _brk 把它导出去，这个标签在 c 语言中就是函数名了。

Eax 寄存器保存的是系统调用的 id，表明是哪个系统调用。Ebx，ecx，esi，edi 可以用来传递参数，参数传入之后，再用 int 主动发出一个中断，INT_VECTOR_SYS_CALL 的值是 0x80，那么就相当于触发了一个 0x80 的中断，正好，这个中断是系统调用的中断，用于陷入内核中执行对应的内核函数。执行完后会返回到这里，继续执行 ret，就会返回调用_brk 的地方，继续往后面执行。

其它的系统调用都是类似的，可以自行查看。

2.2.8 其它文件

cmd.bat	2019/11/17 19:00	Windows 批处理...
FINISH.md	2019/11/17 19:00	Markdown 源文件
makefile	2019/11/17 19:00	文件
Makefile.build	2019/11/17 19:00	BUILD 文件
README.md	2019/11/17 19:00	Markdown 源文件

Src 中的非目录文件

Cmd.bat 是用于在 windows 环境下快速打开控制台。

FINISH.md 是记录完成了哪些功能的文档。

Makefile 和 makefie.build 是源码编译的规则。

README.md 是关于目录的介绍。

第三章 运行流程分析

3.1 引导加载流程

我们采用软盘作为引导盘，在 boot 中，只做了一件事情，那就是加载 loader，并跳转到 loader 中运行。

在 loader 中可以做更多事情，首先，把内核文件从磁盘加载到内存，软盘不再使用，就关闭软盘驱动，然后检测内存，把内存信息保存到固定的内存中。接下来就是设置保护模式，关中断，加载 gdt 表，打开 A20 总线，切换成保护模式，跳转到保护模式代码入口处（这里涉及到情况 CPU 流水线操作，直接 jmp 就可以了）

在保护模式代码中，先设置段选择子的值，设置栈顶。然后开启分页，把 elf 格式的内核文件读取到虚拟地址处，内核的虚拟地址是 0x80100000（在 makefile 中指定的），也就是高端虚拟地址 1M 的位置。接下来就是跳转到内核中去执行。

3.2 Arch 初始化流程

内核的入口是 src/x86/kernel/entry.asm 里面的_start 标签。在这里面先调用了初始化平台架构相关的，再进入到内核的主函数中去。先看进入平台架构做了什么。

```
;init all segment registeres
;设置一下段
mov ax, 0x10    ;the data
mov ds, ax
mov es, ax
mov fs, ax
mov gs, ax
mov ss, ax
mov esp, KERNEL_STACK_TOP - 4

;初始化平台架构
call InitArch                   ;into arch

;jmp $
;初始化平台成功，接下来跳入到内核中去执行
call main
```

Src/x86/kernel/entry.asm

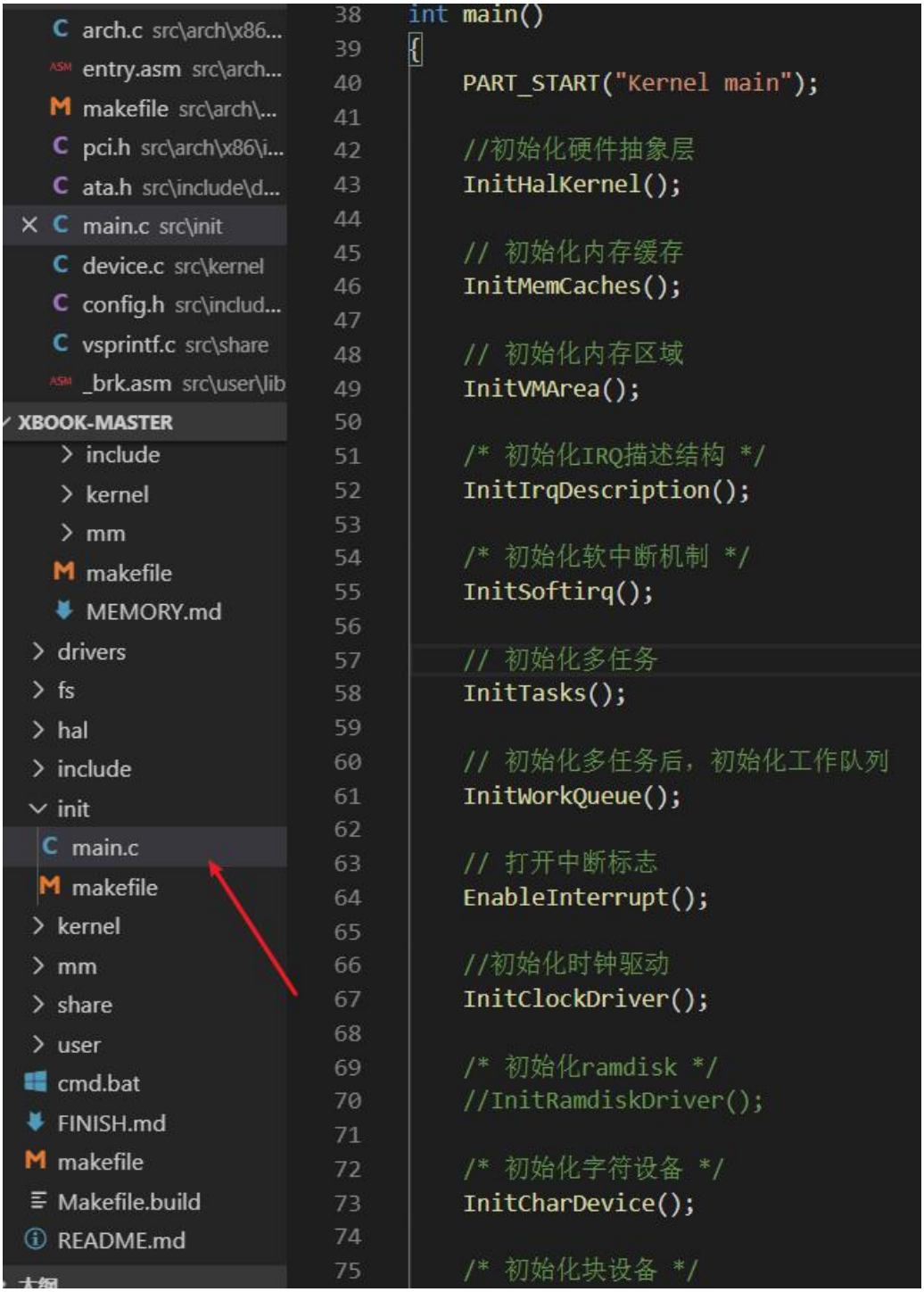
初始化平台架构位于 src/x86/kernel/arch 里面，首先初始化应急抽象层的环境，再初始化硬件抽象，由于需要显示输出一些内容，所以先初始化了控制台，这样就可以往上面输出信息了。接下来创建 CPU 和 RAM 的硬件抽象。

回到 InitArch 中，接下来初始化 CPU 相关信息，初始化段描述符，门描述符（中断），初始化 TSS，再初始化物理内存管理，初始化 PCI 接口。那么 arch 相关的初始化就结束了。从这里返回到 entry 中，然后调用 main，进入到真正的内核处。

3.3 内核初始化流程

内核的初始化代码位于 src/init/main.c 中，首先会初始化硬件抽象，由于调整后，目前里面啥也没做。然后初始化虚拟内存管理，也就是 MemCaches 和 VMArea，之后我们才可以使用 kmalloc。接下来是初始化软中断机制。接下来就是初始化多任务了，多任务中有内

核线程和进程，会把当前正在进行的这个“任务”当做主线程，直到打开始终中断之后，才会进行调度，才是真正的多任务（多个任务“同时”运行）。初始化多任务之后，我们的系统相当于有多个任务在运行。只是假象，其实每次 cpu 只能执行一个任务，只是说任务间的切换到一定的速度，看起来就流畅了。（就像动画原理一样，只要 1s 切换 24 帧，那么看起来动画就是流畅的）。



初始化多任务后，就是初始化工作队列，工作队列也是用于中断，但是需要有任务才行，所以就在初始化多任务后再初始化。

接下来就是打开中断，此后才可以响应中断。然后初始化时钟驱动，这样，内核就会在不知不觉中，默默的进行任务调度。只要我们前面创建了多个线程，那么这些线程现在都可

以得到运行的机会。

然后是初始化字符设备，目前只有键盘驱动。

再者，初始化块设备，目前有 IDE 硬盘和 Ramdisk 磁盘，分别初始化了对应的驱动。现在就可以访问磁盘了，读写扇区之类的。

```
/* 初始化块设备 */
InitBlockDevice();

/* 初始化文件系统 */
//InitFileSystem();

/* 初始化文件系统 */
//InitFileSystem();

//Spin("bofs test");

/* 加载init进程 */
InitFirstProcess("root:init", "init");
//BlockDeviceTest();

/* main thread 就是idle线程 */
while (1) {

    /* 进程默认处于阻塞状态，如果被唤醒就会执行后面的操作，
    知道再次被阻塞 */
    TaskBlock(TASK_BLOCKED);
    /* 打开中断 */
    EnableInterrupt();
    /* 执行cpu停机 */
    CpuHlt();
};
PART_END();
return 0;
```

有了磁盘驱动之后，就可以有文件系统，文件系统目前还处于测试阶段，所以先不谈。对于初始化 init 进程，也是需要文件系统的支持，所以现在里面相当于啥也没做。

接下来就会进入一个 while 循环，我把他当做 idle 线程。当没有任务运行的时候，就会唤醒它，它就会运行，这样，就不至于说整个系统一个任务都没有了，至少还有一个 idle。

Idle 先把自己阻塞，如果被唤醒后，就会打开中断，并进入 hlt（不消耗 cpu，啥也不做，不耗电，可以节约用电）。如果有其他任务产生，那么自己又会被阻塞。一直如此循环下去。

那么至此，内核也初始化完毕。就开始“群雄逐鹿”了。

第四章 对未来的展望

4.1 计划中的内容

文件系统：虚拟文件系统（不是 linux 的虚拟文件系统），FATfs，BFS。

进程间通信：管道 pipe，命名管道 fifo，共享内存，信号，信号量，消息队列，socket 进程间通信。

同步互斥机制： 读写锁，条件变量。

用户态线程：参考 pthread，并开发线程的同步与互斥机制

实现网卡驱动，实现网络协议栈，编写网络通信接口。

实现 vga 驱动，编写图形服务程序，编写图形界面。

实现鼠标驱动。实现鼠标+键盘+图形界面的用户交互。

实现声卡驱动，尝试播放简单的音乐。

支持 usb 接口。实现对 u 盘的数据控制。

支持 POSIX 标准。

开发基础的应用程序：简易浏览器，文本编辑器，图片查看器，视频播放，音乐播放，移植 gcc，nasm，bochs 等软件。

开发者可以选择自己感兴趣的部分进行开发。

4.2 开发者的注意事项

1.开发者应该先对源码有个大致的了解，把代码浏览一遍。

2.想一下自己要做什么内容，并在该领域进行深入探究。

3.注意代码编码风格，这里，我把代码风格贴出来，供大家参考。

```
我们 C 代码和汇编代码，我们分开说明！
## 汇编代码：
    1.标签都是大写字母开头，单词间没有间隔
    举例：LableStart，Entry，LoopForFoo123
    2.宏定义都是大写字母，单词间用下划线隔开
    举例：LOADER_START_SECTOR，FOO_MAX_NR，TIMES_2019_FAST
    3.其它的所有命名都是小写
    举例：  mov byte [0], al  dw 0xaa55
    汇编的规则就这些，其它的可以自我发挥！
## C 代码：
    1.函数命名：每个单词开头大写
    举例：Init，InitCpu，CreateWindow，AllocKernelMemoryPage，
WindowSetTitle
    2.结构体命名：每个单词开头大写
    举例：
    struct FooProgram
    {
        int foo;
    };
    3.宏定义都是大写字母，单词间用下划线隔开
```

```
    举例: MAX_DIR_NR, FOO_MAX_NR
    4.变量的命名,首字母小写,后面都大写。areYouLikeThis,isTrue,above80Right
    5.注意多实用空格，其它的可以自我发挥。
如果还有其它的需要添加，后面慢慢完善！
:)
## 注意
- 编码格式统一 utf-8

## 注释的重要性
注释可以用英文也可以用中文，看个人爱好
1.在文件开头必须添加注释，表明文件是什么，内容主要是：
文件：文件的全路径
作者：作者名字
日期：文件创建日期
版权：版权声明
其它内容可以自行添加
##
    汇编代码
    ;----
    ;file:      arch/x86/include/const.inc
    ;auther:    Jason Hu
    ;time:      2018/1/23
    ;copyright: (C) 2018-2019 by BookOS developers. All rights reserved
.
    ;----
    C 语言代码
    /*
    * file:     init/main.c
    * auther:   Jason Hu
    * time:     2019/6/2
    * copyright:    (C) 2018-2019 by Book OS developers. All rights res
erved.
    */
    makefile 代码
    #----
    #file:      arch/x86/boot/makefile
    #auther:    Jason Hu
    #time:      2019/6/2
    #copyright: (C) 2018-2019 by Book OS developers. All rights reserve
d.
    #----

2.在编写函数的时候要加上对函数的注释，主要内容是：
功能：函数的主要功能
```

```
参数：有哪些参数
返回值：返回哪些值，不同的值有啥意思
说明：需要提供的其它更多信息
##
    /**
    * SysMmap - 内存映射
    * @addr: 地址
    * @len: 长度
    * @prot: 页保护
    * @flags: 空间的标志
    *
    * 执行内存映射，使得该范围内的虚拟地址可以使用
    * 成功返回映射后的地址，失败返回 NULL
    */
```

这是直接从 doc/README.md 中复制出来的。可以仔细阅读以下。

4.有问题可以在 github 上发出问题，或者在 QQ 群中讨论，也可以给我发邮件。

5.开发方式，通过 github 合作开发，开发者需要先从上面获取最新内容，自己添加新内容，测试成功后，再推送到上面，并且要标明自己修改了什么内容。只有这样，才能保证最新内容是最佳的。

6.每一个有贡献的开发者，都会在开发者列表中出现自己的名字。

最后，希望大家一起能够做出一件伟大的事情。让我们站在巨人的肩膀上，成为后人的巨人。^_^

后记

由于本人能力有限，文档中可能还存在些许错误，希望大家能够指出，我将会及时修正。可能大家还会在开发中遇到问题，就可以联系我，我会更新解决方法到文档之中。也感谢大家对本系统的关注与支持。

编撰者：胡自成

联系方式

官网：www.book-os.org
电子邮箱：book_os@163.com
QQ 群：913813452

文档版本发布历史记录

2019/2/26	基于 BookOS v0.2	发布 v0.1 文档
2019/11/24	基于 XBook v0.6.1	发布 v0.2 文档