

[文章](#) [问答](#) [论坛](#) [东西](#) [休息室](#) [?](#)

Search for articles, questions,

[手表](#)

用 C/C++ 编写 16 位虚拟内核



阿什奇兰·巴特

2014 年 3 月 14 日 [警察](#)

评价我: 4.98/5 (80 票)

理解 FAT 文件系统和 C/C++ 内核编程

[下载源 - 7.4 KB](#)

介绍

在我之前的文章中，我只是简要介绍了如何编写引导加载程序。那很有趣也很有挑战性。我很喜欢它。但是在学习了如何编写引导加载程序之后，我想编写更好的东西，比如在其中嵌入更多功能。但随后加载程序的大小不断增加超过 512 字节，显然每次我用启动盘重新启动系统时，我都会看到错误“这不是可启动磁盘”。

文章的范围是什么？

在本文中，我将尝试简要介绍文件系统对我们的引导加载程序的重要性，并尝试编写一个虚拟内核，它除了显示用户输入文本的提示外什么都不做。为什么我要将引导加载程序嵌入到 FAT 格式的软盘中，它对我有什么好处。由于一篇文章太小而无法提及文件系统，我将尽我所能将其写得尽可能简短。

背景

- 如果您有任何语言的编程经验，这篇文章真的对您有很大帮助。尽管这篇文章似乎相当介绍性，但用汇编和 C 编写程序对于引导来说可能是一项艰巨的任务。如果您不熟悉计算机编程，那么我建议您阅读一些介绍编程和计算机基础知识的教程，然后再回到本文。
- 在整篇文章中，我将通过问答的方式向您介绍与计算机相关的各种术语。坦率地说，我会写这篇文章，就像我在向自己介绍这篇文章一样。进行了如此多的问答式对话，以确保我了解它在我日常生活中的重要性和目的。示例：您所说的计算机是什么意思？或者我为什么需要它们，因为我比它们聪明得多？

您也可以查看我以前的文章，以了解有关引导加载程序以及如何用汇编和 C 编写引导加载程序的基本概念。

这是链接。

<http://www.codeproject.com/Articles/664165/Writing-a-boot-loader-in-Assembly-and-C-Part><http://www.codeproject.com/Articles/668422/Writing-a-boot-loader-in-Assembly-and-C-Part>

内容是如何组织的？

这是本文主题的细分。

- 引导加载程序限制
- 从 Boot-loader 调用磁盘上的其他文件
- FAT文件系统
- FAT工作流程
- 开发环境
- 编写 FAT 引导加载程序
- 开发环境
- 小项目 - 编写 16 位内核
- 测试内核

引导加载程序限制

在之前的文章中，我尝试编写引导加载程序，在屏幕上打印彩色矩形后，我想将更多功能嵌入其中。但是，512 字节的大小对我来说是一个很大的限制，我不能更新引导加载程序的代码来做更多...

挑战如下

- 将更多功能方面的代码嵌入到引导加载程序中
- 将引导加载程序的大小限制为仅 512 字节。

我将如何处理上述问题？

下面小编给大家介绍一下。

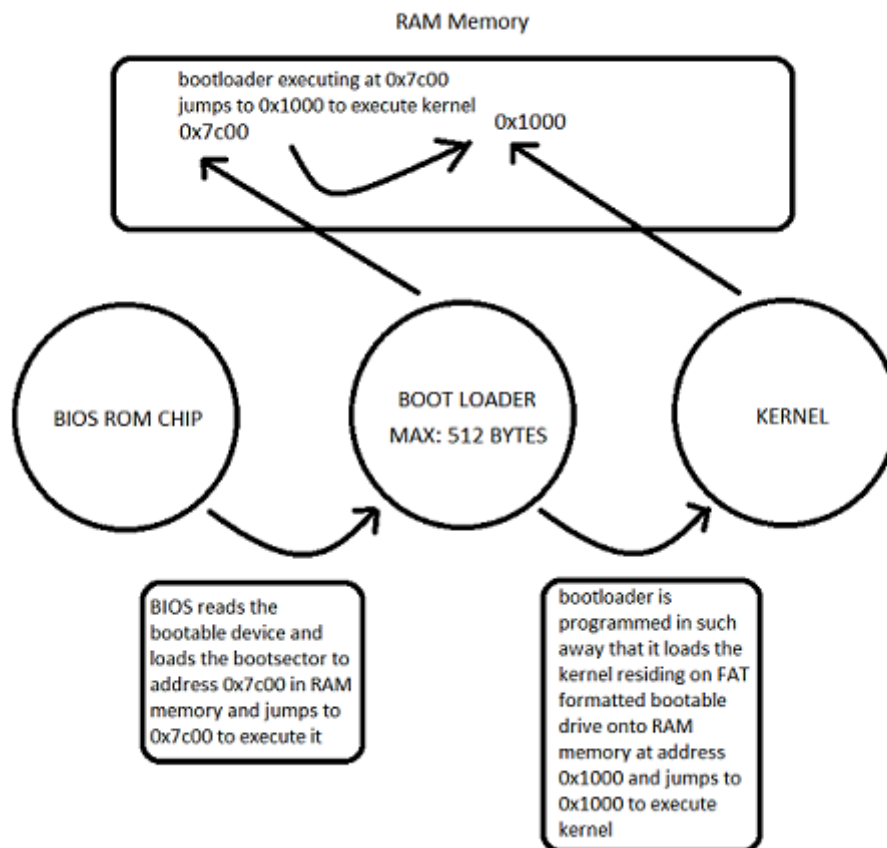
第1部分：

- 我将用 C 语言编写一个名为 kernel.c 的程序，确保我想要的所有额外功能都正确地写入其中。
- 编译并保存可执行文件为 kernel.bin
- 现在，将 kernel.bin 文件复制到可引导驱动器的第二个扇区。

第2部分：

在我们的引导加载程序中，我们所能做的就是将可引导驱动器的第二个扇区（kernel.bin）加载到 RAM 内存中，地址为 0x1000，然后从 0x7c00 跳转到位置 0x1000 以开始执行 kernel.bin 文件。

下面是你可以参考的图片以获得一个想法。



从引导加载程序调用磁盘上的其他文件

早些时候，我们知道我们实际上可以将控制权从 bootloader(0x7c00) 传递到其他磁盘文件（如 kernel.bin）在内存中的位置，然后继续进行。但我心中几乎没有疑问。

你知道kernel.bin文件在磁盘上会占用多少扇区吗？

我认为这很容易。我们所要做的就是以下

1 个扇区 = 512 字节

所以，如果 kernel.bin 的大小是 512 字节，那么它将占用 1 个扇区，如果大小为 1024 字节，则为 2 个扇区，依此类推...

现在，重点是基于 kernel.bin 文件的大小，您必须对引导加载程序中的 kernel.bin 文件读取的扇区数进行硬编码。

这意味着以后如果你想通过频繁更新内核来升级内核，你还必须记住在引导加载程序中记录 kernel.bin 文件将占用的扇区数，否则内核会崩溃。

如果您想在可启动驱动器中添加除 kernel.bin 之外的更多文件，例如 office.bin、entertainment.bin、drivers.bin，您会怎么想？

在某些时候是的，您要做的就是继续将文件一个一个地附加到您的软盘上，在此过程中，您将不得不继续更新引导加载程序，了解每个文件在引导盘上的确切位置以及每个文件消耗的扇区数甚至更多。

但我想提醒您注意的一点是，慢慢地系统变得越来越复杂，不知何故我喜欢它。

你怎么知道你在引导扇区后——附加的文件是否是你想要的？

我们所做的就是从引导加载程序将相应的扇区加载到内存中，然后开始执行它。但这并不完美，还缺少一些东西。

有什么不见了？

我认为引导加载程序盲目地对每个文件一个一个地加载扇区，然后开始执行文件。但即使在引导加载程序尝试将文件加载到内存之前，也应该有一种方法来检查文件是否存在于引导磁盘上。

如果我错误地将错误的文件复制到引导盘的第二个扇区，然后更新引导加载程序然后运行，会发生什么情况？

我的系统只是崩溃了，用户会扔掉我的系统，因为这不是他想要的。

因此，在这种情况下，我只需要启动磁盘上的一个固定位置，所有文件名都像书中的索引一样写入。

我的引导加载程序将查询软盘的索引以查找文件名，如果文件名列在索引中，则继续将文件加载到内存中。

哇!!! 这很好，我喜欢它，因为它可以节省很多操作。

这消除了一些问题

早期的boot-loader盲目地用来加载硬编码在里面的扇区。

如果您不知道加载的文件是否正确，为什么要加载文件？

解决办法是什么？

我们所要做的就是组织上面列出的磁盘上的信息，然后开始组织数据，然后重新编程我们的引导加载程序，以便它可以真正有效地加载文件。

这种大规模组织数据的方式称为文件系统。有许多类型的文件系统，既有商业的，也有免费的。我将在下面列出其中的一些。

- 胖的
- FAT16
- FAT32
- NTFS
- 分机
- EXT2
- EXT3
- EXT4

FAT文件系统

在向您介绍文件系统之前，您需要了解一些术语。

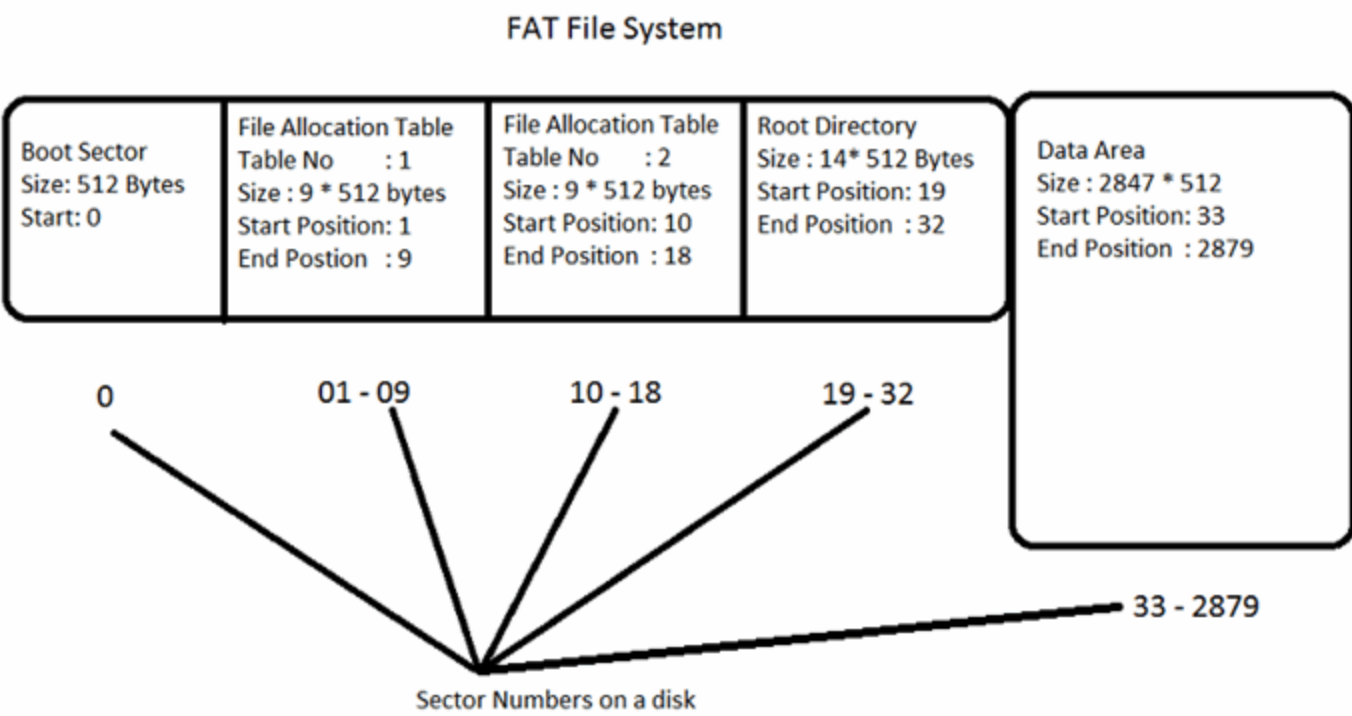
在 FAT 文件系统中，一个簇在存储介质上占用 1 个扇区，一个扇区占用 512 个字节。因此，在 FAT 格式的磁盘驱动器上，1 个簇相当于 1 个扇区。

簇和扇区是 FAT 文件系统上的最小单位。

为方便使用，FAT 文件系统分为四个主要部分，如下所示。

- 引导扇区
- 胖的
- 根目录
- 数据区

我尽量以图片的形式展示给大家，以便大家更好的理解。



现在让我向您介绍每个部分。

引导扇区：

FAT 格式磁盘上的引导扇区嵌入了一些与 FAT 相关的信息，因此每次将磁盘插入系统时，操作系统都会自动知道其文件系统。操作系统读取 FAT 格式磁盘的引导扇区，然后解析所需的信息，然后识别文件系统的类型，然后开始读取相应的内容。嵌入在引导扇区中的有关 FAT 文件系统的信息称为引导参数块。

启动参数块：

让我向您展示与引导扇区相关的引导参数块中的值。

Offset in Boot Sector	Offset in Boot Parameter Block	Length in bytes	Meaning	Values	Mandatory to specify
3 - 10	0 - 7	8	OEM Label	kirUX0.1	Not required
11 - 12	8 - 9	2	Total bytes per sector	512	YES
13	10	1	Total sectors per cluster	1	YES
14 - 15	11 - 12	2	Total reserved sectors	1	YES
16	13	1	Total FAT tables	2	YES
17 - 18	14 - 15	2	Total root directory entries	224	YES
19 - 20	16 - 17	2	Total sectors	2880	YES
21	18	1	Media Description	0xf0	YES
22 - 23	19 - 20	2	Total sectors per FAT	9	YES
24 - 25	21 - 22	2	Total sectors per track	18	YES
26 - 27	23 - 24	2	Total heads per cylinder	2	YES
28 - 31	25 - 28	4	Total hidden sectors	0	YES
32 - 35	29 - 32	4	Total big sectors	0	YES
36	33	1	boot drive identifier	0	YES
37	34	1	Total un-used sectors	0	YES
38	35	1	external boot signature	0x29	YES
39 - 42	36 - 39	4	serial number	"by	Not required
43 - 53	40 - 50	11	volume label	ASHAKIRAN B	Not required
54 - 60	51 - 57	8	file system type	FAT12	Not required
61 - 509		449	Boot Code		
510		1	Boot Signature	0X55	YES
511		1	Boot Signature	0XAA	YES

文件分配表：

该表的作用类似于一个包含文件的下一个簇值的链表。

从特定文件的 FAT 获得的簇值有两种用途。

- 确定文件结尾
 - 如果簇值在 0x0ff8 和 0x0fff 之间，则文件在其他扇区中没有数据（到达文件尾）。
- 确定文件数据所在的下一个扇区

笔记：

我在图片中提到了 FAT 表 1 和 2。您需要记住的是，一张表是另一张表的副本。如果一个表中的数据丢失或损坏，另一个表中的数据可以作为备份。这是引入两张表而不是一张表的纯粹意图。

根目录：

根目录的作用类似于磁盘上所有文件名列表的索引。所以引导加载程序应该在根目录中搜索文件名，如果是肯定的，那么它可以在根目录中找到第一个簇，然后相应地加载数据。

现在从根目录找到第一个簇后，引导加载程序应该使用 FAT 表来查找下一个簇以检查文件的结尾。

数据区：

这是实际包含文件数据的区域。

一旦程序识别出文件的正确扇区，就可以从数据区中提取文件的数据。

FAT 工作流程

假设，假设我们的引导加载程序应该将 kernel.bin 文件加载到内存中，然后执行它。现在，在这种情况下，我们所要做的就是将以下功能编码到我们的引导加载程序中。

将前 11 个字节的数据与根目录表中从偏移量 0 开始的“kernel.bin”进行比较。

如果字符串匹配，则在根目录表中的偏移量 26 处提取“kernel.bin”文件的第一个簇。

现在您有了“kernel.bin”文件的起始簇。

您所要做的就是将集群转换为相应的扇区，然后将数据加载到内存中。

现在找到“kernel.bin”文件的第一个扇区后，加载到内存中，然后在文件分配表中查找文件的下一个簇，以检查文件是否仍有数据或已到达文件尾。

下图供大家参考。

开发环境

为了成功完成这项任务，我们需要了解以下内容。有关它们的更多信息，请参阅我之前的文章。

- 操作系统 (GNU Linux)
- 汇编程序 (GNU 汇编程序)
- 指令集 (x86 系列)
- 在 GNU 汇编器上为 x86 微处理器编写 x86 指令。
- 编译器 (C 编程语言 - GNU C 编译器 GCC)
- 链接器 (GNU 链接器 ld)
- 一个 x86 模拟器，如用于我们测试目的的 boch。

编写 FAT 引导加载程序

下面是用于在 FAT 格式的磁盘上执行 kernel.bin 文件的代码片段。

这是引导程序

文件名: stage0.S

C++

缩小▲ 复制代码

```

/*****
 *
 *
 *   Name       : stage0.S
 *   Date       : 23-Feb-2014
 *   Version    : 0.0.1
 *   Source     : assembly language
 *   Author     : Ashakiran Bhatte
 *
 *   Description: The main logic involves scanning for kernel.bin file on a
 *               fat12 formatted floppy disk and then pass the control to it
 *               for its execution
 *   Usage      : Please read the readme.txt for more information
 *
 *****/
.code16
.text
.globl _start;
_start:
    jmp _boot
    nop
    /*bios parameter block
    /*-----
    .byte 0x6b,0x69,0x72,0x55,0x58,0x30,0x2e,0x31 /* oem label
    .byte 0x00,0x02 /* total bytes per sector
    .byte 0x01 /* total sectors per cluster
    .byte 0x01,0x00 /* total reserved sectors
    .byte 0x02 /* total fat tables
    .byte 0xe0,0x00 /* total directory entries
    .byte 0x40,0x0b /* total sectors
    .byte 0xf0 /* media description
    .byte 0x09,0x00 /* size in of each fat table
    .byte 0x02,0x01 /* total sectors per track
    .byte 0x02,0x00 /* total heads per cylinder
    .byte 0x00,0x00, 0x00, 0x00 /* total hidden sectors
    .byte 0x00,0x00, 0x00, 0x00 /* total big sectors
    .byte 0x00 /* boot drive identifier
    .byte 0x00 /* total unused sectors
    .byte 0x29 /* external boot signature
    .byte 0x22,0x62,0x79,0x20 /* serial number
    .byte 0x41,0x53,0x48,0x41,0x4b,0x49 /* volume label 6 bytes of 11
    .byte 0x52,0x41,0x4e,0x20,0x42 /* volume label 5 bytes of 11
    .byte 0x48,0x41,0x54,0x54,0x45,0x52,0x22 /* file system type

    /* include macro functions */
#include "macros.S"

/* beginning of main code */
_boot:
    /* initialize the environment */
    initEnvironment

    /* Load stage2 */
    loadFile $fileStage2

```



```

/* infinite loop */
_freeze:
    jmp _freeze

/* abnormal termination of program */
_abort:
    writeString $msgAbort
    jmp _freeze

/* include functions */
#include "routines.S"

/* user-defined variables */
bootDrive : .byte 0x0000
msgAbort   : .asciz "*** FATAL ERROR ***"
#fileStage2: .ascii "STAGE2 BIN"
fileStage2: .ascii "KERNEL BIN"
clusterID  : .word 0x0000

/* traverse 510 bytes from beginning */
. = _start + 0x01fe

/* append boot signature */
.word BOOT_SIGNATURE

```

这是主要的加载程序文件，它执行以下操作。

- 通过调用 `initEnvironment` 宏初始化所有寄存器并设置堆栈。
- 调用 `loadFile` 宏将 `kernel.bin` 文件加载到地址 `0x1000:0000` 的内存中，然后将控制权交给它以进一步执行。

文件名：宏.S

这是一个包含所有预定义宏和宏功能的文件。

缩小▲ 复制代码

```

/*****
*
*
*      Name      : macros.S
*      Date       : 23-Feb-2014
*      Version    : 0.0.1
*      Source     : assembly language
*      Author     : Ashakiran Bhattar
*
*
*****/
/* predefined macros: boot loader */
#define BOOT_LOADER_CODE_AREA_ADDRESS      0x7c00
#define BOOT_LOADER_CODE_AREA_ADDRESS_OFFSET 0x0000

/* predefined macros: stack segment */
#define BOOT_LOADER_STACK_SEGMENT          0x7c00

#define BOOT_LOADER_ROOT_OFFSET            0x0200
#define BOOT_LOADER_FAT_OFFSET             0x0200

#define BOOT_LOADER_STAGE2_ADDRESS         0x1000
#define BOOT_LOADER_STAGE2_OFFSET          0x0000

/* predefined macros: floppy disk layout */
#define BOOT_DISK_SECTORS_PER_TRACK        0x0012
#define BOOT_DISK_HEADS_PER_CYLINDER      0x0002
#define BOOT_DISK_BYTES_PER_SECTOR        0x0200
#define BOOT_DISK_SECTORS_PER_CLUSTER     0x0001

/* predefined macros: file system layout */

```

```

#define FAT12_FAT_POSITION          0x0001
#define FAT12_FAT_SIZE              0x0009
#define FAT12_ROOT_POSITION        0x0013
#define FAT12_ROOT_SIZE            0x000e
#define FAT12_ROOT_ENTRIES        0x00e0
#define FAT12_END_OF_FILE          0x0ff8

/* predefined macros: boot loader */
#define BOOT_SIGNATURE              0xaa55

/* user-defined macro functions */
/* this macro is used to set the environment */
.macro initEnvironment
    call _initEnvironment
.endm
/* this macro is used to display a string */
/* onto the screen */
/* it calls the function _writeString to */
/* perform the operation */
/* parameter(s): input string */
.macro writeString message
    pushw \message
    call _writeString
.endm
/* this macro is used to read a sector into */
/* the target memory */
/* It calls the _readSector function with */
/* the following parameters */
/* parameter(s): sector Number */
/* address to load */
/* offset of the address */
/* Number of sectors to read */
.macro readSector sectorno, address, offset, totalsectors
    pushw \sectorno
    pushw \address
    pushw \offset
    pushw \totalsectors
    call _readSector
    addw $0x0008, %sp
.endm
/* this macro is used to find a file in the */
/* FAT formatted drive */
/* it calls readSector macro to perform this */
/* activity */
/* parameter(s): root directory position */
/* target address */
/* target offset */
/* root directory size */
.macro findFile file
    /* read fat table into memory */
    readSector $FAT12_ROOT_POSITION, $BOOT_LOADER_CODE_AREA_ADDRESS, $BOOT_LOADER_ROOT_OFFSET,
    $FAT12_ROOT_SIZE
    pushw \file
    call _findFile
    addw $0x0002, %sp
.endm
/* this macro is used to convert the given */
/* cluster into a sector number */
/* it calls _clusterToLinearBlockAddress to */
/* perform this activity */
/* parameter(s): cluster number */
.macro clusterToLinearBlockAddress cluster
    pushw \cluster
    call _clusterToLinearBlockAddress
    addw $0x0002, %sp
.endm
/* this macro is used to load a target file */
/* into the memory */
/* It calls findFile and then loads the data */

```

```

/* of the respective file into the memory at */
/* address 0x1000:0x0000 */
/* parameter(s): target file name */
.macro loadFile file
    /* check for file existence */
    findFile \file

    pushw %ax
    /* read fat table into memory */
    readSector $FAT12_FAT_POSITION, $BOOT_LOADER_CODE_AREA_ADDRESS, $BOOT_LOADER_FAT_OFFSET,
    $FAT12_FAT_SIZE

    popw %ax
    movw $BOOT_LOADER_STAGE2_OFFSET, %bx
_loadCluster:
    pushw %bx
    pushw %ax

    clusterToLinearBlockAddress %ax
    readSector %ax, $BOOT_LOADER_STAGE2_ADDRESS, %bx, $BOOT_DISK_SECTORS_PER_CLUSTER

    popw %ax
    xorw %dx, %dx
    movw $0x0003, %bx
    mulw %bx
    movw $0x0002, %bx
    divw %bx

    movw $BOOT_LOADER_FAT_OFFSET, %bx
    addw %ax, %bx
    movw $BOOT_LOADER_CODE_AREA_ADDRESS, %ax
    movw %ax, %es
    movw %es:(%bx), %ax
    orw %dx, %dx
    jz _even_cluster
_odd_cluster:
    shrw $0x0004, %ax
    jmp _done
_even_cluster:
    and $0xffff, %ax
_done:
    popw %bx
    addw $BOOT_DISK_BYTES_PER_SECTOR, %bx
    cmpw $FAT12_END_OF_FILE, %ax
    jl _loadCluster

    /* execute kernel */
    initKernel
.endm
/* parameter(s): target file name */
/* this macro is used to pass the control of */
/* execution to the loaded file in memory at */
/* address 0x1000:0x0000 */
/* parameters(s): none */
.macro initKernel
    /* initialize the kernel */
    movw $(BOOT_LOADER_STAGE2_ADDRESS), %ax
    movw $(BOOT_LOADER_STAGE2_OFFSET), %bx
    movw %ax, %es
    movw %ax, %ds
    jmp $(BOOT_LOADER_STAGE2_ADDRESS), $(BOOT_LOADER_STAGE2_OFFSET)
.endm

```

初始化环境:

- 该宏用于根据需要设置段寄存器。
- 需要传递的参数数量为 none

用法: `initEnvironment`

写字符串:

- 此宏用于在屏幕上显示空终止字符串。
- 传递给它的参数是一个以空字符结尾的字符串变量。

用法: `writeString <字符串变量>`

读取扇区:

- 该宏用于从磁盘读取给定扇区, 然后将其加载到目标地址
- 传递所需的参数数量为 4。

用法: `readSector <扇区号>、<目标地址>、<目标地址偏移量>、<要读取的扇区总数>`

查找文件:

- 该宏用于检查文件是否存在。
- 需要传递的参数数量是 1

用法: `findFile <目标文件名>`

clusterToLinearBlockAddress:

- 此宏用于将给定的集群 ID 转换为扇区号。
- 需要传递的参数数量是 1

用法: `clusterToLinearBlockAddress <cluster ID>`

加载文件:

- 该宏用于将目标文件加载到内存中, 然后将执行控制权交给它。
- 需要传递的参数数量是 1

用法: `loadFile <目标文件名>`

初始化内核:

- 此宏用于将执行控制传递到 RAM 上的特定地址位置
- 传递所需的参数数量为无。

用法: `initKernel`

文件名: routines.S

自动售货机

缩小▲ 复制代码

```

/*****
 *
 *
 *   Name       : routines.S
 *   Date       : 23-Feb-2014
 *   Version    : 0.0.1
 *   Source     : assembly language
 *   Author     : Ashakiran Bhattar
 *
 *****/
/* user-defined routines */
/* this function is used to set-up the */

```

```

/* registers and stack as required */
/* parameter(s): none */
_initEnvironment:
    pushw %bp
    movw %sp, %bp
_initEnvironmentIn:
    cli
    movw %cs, %ax
    movw %ax, %ds
    movw %ax, %es
    movw %ax, %ss
    movw $BOOT_LOADER_STACK_SEGMENT, %sp
    sti
_initEnvironmentOut:
    movw %bp, %sp
    popw %bp
ret

/* this function is used to display a string */
/* onto the screen */
/* parameter(s): input string */
_writeString:
    pushw %bp
    movw %sp, %bp
    movw 4(%bp), %si
    jmp _writeStringCheckByte
_writeStringIn:
    movb $0x000e, %ah
    movb $0x0000, %bh
    int $0x0010
    incw %si
_writeStringCheckByte:
    movb (%si), %al
    orb %al, %al
    jnz _writeStringIn
_writeStringOut:
    movw %bp, %sp
    popw %bp
ret

/* this function is used to read a sector */
/* into the target memory */
/* parameter(s): sector Number */
/* address to load */
/* offset of the address */
/* Number of sectors to read */
_readSector:
    pushw %bp
    movw %sp, %bp

    movw 10(%bp), %ax
    movw $BOOT_DISK_SECTORS_PER_TRACK, %bx
    xorw %dx, %dx
    divw %bx

    incw %dx
    movb %dl, %cl

    movw $BOOT_DISK_HEADS_PER_CYLINDER, %bx
    xorw %dx, %dx
    divw %bx

    movb %al, %ch
    xchg %dl, %dh

    movb $0x02, %ah
    movb 4(%bp), %al
    movb bootDrive, %dl
    movw 8(%bp), %bx

```

```

    movw %bx    , %es
    movw 6(%bp) , %bx
    int $0x13
    jc  _abort
    cmpb 4(%bp) , %al
    jc  _abort

    movw %bp    , %sp
    popw %bp

ret

/* this function is used to find a file in */
/* the FAT formatted drive                */
/* parameter(s): root directory position  */
/*                target address          */
/*                target offset           */
/*                root directory size     */
/*                */
_findFile:
    pushw %bp
    movw %sp    , %bp

    movw $BOOT_LOADER_CODE_AREA_ADDRESS, %ax
    movw %ax    , %es
    movw $BOOT_LOADER_ROOT_OFFSET, %bx
    movw $FAT12_ROOT_ENTRIES, %dx
    jmp  _findFileInitValues

_findFileIn:
    movw $0x000b , %cx
    movw 4(%bp)  , %si
    leaw (%bx)   , %di
    repe cmpsb
    je  _findFileOut
_findFileDecrementCount:
    decw %dx
    addw $0x0020, %bx
_findFileInitValues:
    cmpw $0x0000, %dx
    jne  _findFileIn
    je   _abort
_findFileOut:
    addw $0x001a , %bx
    movw %es:(%bx), %ax
    movw %bp, %sp
    popw %bp
ret

/* this function is used to convert the given*/
/* cluster into a sector number              */
/* parameter(s): cluster number              */
/*                */
_clusterToLinearBlockAddress:
    pushw %bp
    movw %sp    , %bp
    movw 4(%bp) , %ax
_clusterToLinearBlockAddressIn:
    subw $0x0002, %ax
    movw $BOOT_DISK_SECTORS_PER_CLUSTER, %cx
    mulw %cx
    addw $FAT12_ROOT_POSITION, %ax
    addw $FAT12_ROOT_SIZE, %ax
_clusterToLinearBlockAddressOut:
    movw %bp    , %sp
    popw %bp
ret

```

_init环境:

- 该函数用于根据需要设置段寄存器。
- 需要传递的参数数量为 none

用法: 调用 _initEnvironment

_writeString:

- 此函数用于在屏幕上显示一个以空字符结尾的字符串。
- 传递给它的参数是一个以空字符结尾的字符串变量。

用法:

- pushw <字符串变量>
- 调用 _writeString
- 地址 \$0x02, %sp

读取扇区:

- 该宏用于从磁盘读取给定扇区, 然后将其加载到目标地址
- 传递所需的参数数量为 4。

用法:

- pushw <扇区号>
- pushw <地址>
- pushw <偏移量>
- pushw <totalsectors>
- 调用 _readSector
- 地址 \$0x0008, %sp

查找文件:

- 此函数用于检查文件是否存在。
- 需要传递的参数数量是 1

用法:

- pushw <目标文件变量>
- 调用 _findFile
- 地址 \$0x02, %sp

clusterToLinearBlockAddress:

- 此宏用于将给定的集群 ID 转换为扇区号。
- 需要传递的参数数量是 1

用法:

- pushw <集群ID>
- 调用 _clusterToLinearBlockAddress
- 地址 \$0x02, %sp

加载文件:

- 该宏用于将目标文件加载到内存中, 然后将执行控制权交给它。
- 需要传递的参数数量是 1

用法:

- pushw <目标文件>
- 调用 _loadFile
- 地址 \$0x02, %sp

文件名: stage0.ld

该文件用于在链接期间链接 stage0.object 文件。

[复制代码](#)

```
/*
 *
 *   Name       : stage0.ld
 *   Date       : 23-Feb-2014
 *   Version    : 0.0.1
 *   Source     : assembly language
 *   Author     : Ashakiran Bhattar
 *
 */
SECTIONS
{
    . = 0x7c00;
    .text :
    {
        _ftext = .;
    } = 0
}
```

文件名: bochsrc.txt

这是运行 bochs 模拟器所需的配置文件，用于测试目的。

[复制代码](#)

```
megs: 32
floppya: 1_44=../iso/stage0.img, status=inserted
boot: a
log: ../log/bochsout.txt
mouse: enabled=0
```

[复制代码](#)

小项目 - 编写 16 位内核

下面的文件是作为测试过程的一部分引入的虚拟内核的源代码。我们所要做的就是使用 make 文件编译源代码，看看它是否被引导加载程序加载。

带有龙图像的初始屏幕以文本形式显示，然后显示一个欢迎屏幕，然后是命令提示符，供用户输入任何内容。那里没有编写命令或实用程序来执行，但只是为了我们的测试目的，引入了这个内核，目前它一文不值。

文件名: kernel.c

C++

[缩小▲](#) [复制代码](#)

```
/*
 *
 *   Name       : kernel.c
 *   Date       : 23-Feb-2014
 *   Version    : 0.0.1
 *   Source     : C
 *   Author     : Ashakiran Bhattar
 *
 */
```



```

*
*   Description: This is the file that the stage0.bin loads and passes the
*               control of execution to it. The main functionality of this
*               program is to display a very simple splash screen and a
*               command prompt so that the user can type commands
*   Caution    : It does not recognize any commands as they are not programmed
*
*****/
/* generate 16 bit code */
__asm__(".code16\n");
/* jump to main function or program code */
__asm__("jmp $0x1000, $main\n");

#define TRUE 0x01
#define FALSE 0x00

char str[] = "$> ";

/* this function is used to set-up the */
/* registers and stack as required */
/* parameter(s): none */
void initEnvironment() {
    __asm__ __volatile__(
        "cli;"
        "movw $0x0000, %ax;"
        "movw %ax, %ss;"
        "movw $0xffff, %sp;"
        "cld;"
    );

    __asm__ __volatile__(
        "movw $0x1000, %ax;"
        "movw %ax, %ds;"
        "movw %ax, %es;"
        "movw %ax, %fs;"
        "movw %ax, %gs;"
    );
}

/* vga functions */
/* this function is used to set the */
/* the VGA mode to 80*24 */
void setResolution() {
    __asm__ __volatile__(
        "int $0x10" : : "a"(0x0003)
    );
}

/* this function is used to clear the */
/* screen buffer by splitting spaces */
void clearScreen() {
    __asm__ __volatile__(
        "int $0x10" : : "a"(0x0200), "b"(0x0000), "d"(0x0000)
    );
    __asm__ __volatile__(
        "int $0x10" : : "a"(0x0920), "b"(0x0007), "c"(0x2000)
    );
}

/* this function is used to set the */
/* cursor position at a given column */
/* and row */
void setCursor(short col, short row) {
    __asm__ __volatile__(
        "int $0x10" : : "a"(0x0200), "d"((row <= 8) | col)
    );
}

/* this function is used enable and */

```

```

/* disable the cursor */
void showCursor(short choice) {
    if(choice == FALSE) {
        __asm__ __volatile__(
            "int $0x10" : : "a"(0x0100), "c"(0x3200)
        );
    } else {
        __asm__ __volatile__(
            "int $0x10" : : "a"(0x0100), "c"(0x0007)
        );
    }
}

/* this function is used to initialize*/
/* the VGA to 80 * 25 mode and then */
/* clear the screen and set the cursor*/
/* position to (0,0) */
void initVGA() {
    setResolution();
    clearScreen();
    setCursor(0, 0);
}

/* io functions */
/* this function is used to get a chara*/
/* cter from keyboard with no echo */
void getch() {
    __asm__ __volatile__(
        "xorw %ax, %ax\n"
        "int $0x16\n"
    );
}

/* this function is same as getch() */
/* but it returns the scan code and */
/* ascii value of the key hit on the */
/* keyboard */
short getchar() {
    short word;

    __asm__ __volatile__(
        "int $0x16" : : "a"(0x1000)
    );

    __asm__ __volatile__(
        "movw %%ax, %0" : "=r"(word)
    );

    return word;
}

/* this function is used to display the*/
/* key on the screen */
void putchar(short ch) {
    __asm__ __volatile__(
        "int $0x10" : : "a"(0x0e00 | (char)ch)
    );
}

/* this function is used to print the */
/* null terminated string on the screen*/
void printString(const char* pStr) {
    while(*pStr) {
        __asm__ __volatile__(
            "int $0x10" : : "a"(0x0e00 | *pStr), "b"(0x0002)
        );
        ++pStr;
    }
}

```

```

/* this function is used to sleep for */
/* a given number of seconds */
void delay(int seconds) {
    __asm__ __volatile__(
        "int $0x15" : : "a"(0x8600), "c"(0x000f * seconds), "d"(0x4240 * seconds)
    );
}

/* string functions */
/* this function is used to calculate */
/* length of the string and then return */
/* it */
int strlen(const char* pStr) {
    int i = 0;

    while(*pStr) {
        ++i;
    }
    return i;
}

/* UI functions */
/* this function is used to display the */
/* logo */
void splashScreen(const char* pStr) {
    showCursor(FALSE);
    clearScreen();
    setCursor(0, 9);
    printString(pStr);
    delay(10);
}

/* shell */
/* this function is used to display a */
/* dummy command prompt onto the screen */
/* and it automatically scrolls down if */
/* the user hits return key */
void shell() {
    clearScreen();
    showCursor(TRUE);
    while(TRUE) {
        printString(str);
        short byte;
        while((byte = getchar())) {
            if((byte >> 8) == 0x1c) {
                putchar(10);
                putchar(13);
                break;
            } else {
                putchar(byte);
            }
        }
    }
}

/* this is the main entry for the kernel */
void main() {
    const char msgPicture[] =
        "
        ..                                     \n\r"
        ++`                                     \n\r"
        :ho.                                `.-/+/.  \n\r"
        `hh+.                                `:sds:  \n\r"
        -odds/-`                             .MNd/`  \n\r"
        .+ydm dyo/:--/yMMMMd/               \n\r"
        :+hMMMMNNNNMMddNMMh:`               \n\r"
        `:/+++/:-:ohmNNNNNNNNNNm+--+mMNd`  \n\r"
        ~+ooosdMMMMNNNNNNNNNNNNNNNNNNmNNMM/` \n\r"
        ~~~~ .+MMMMMMMMMMMMMMMMMMMMMMMMMMMMNNmho:.` \n\r"

```

```

"                `omMMMMMMMMMMMMMMMMMMMMNdMNdNMMMMMMMMdo+-      \n\r"
"                .:oyMMMMMMMMMMMMMMMMMMNd/hMMd+ds-:h/-yMdydMNdNdNN+  \n\r"
"                -oosdMMMMMMMMMMMMMMMMd:` `yMM+.+h+.- /y `/m.:mmmN    \n\r"
"                -:` dMMMMMMMMMMMMMMMd.      `mMNo..+y/` . . -/.s    \n\r"
"                ` -MMMMMMMMMMMMMMMM-      -mMMmo-./s/.` `          \n\r"
"                `+MMMMMMMMMMMMMMMM-      .smMy:.` `+oo+//:-.`      \n\r"
"                .yNMMMMMMMMMMMMMMMd.      .+dmh+:. ` -:./+:.      \n\r"
"                y+-mMMMMMMMMMMMMMMm/`      ./o+-` .          \n\r"
"                :- :MMMMMMMMMMMMMMMMMmy/.`      \n\r"
"                ` `hMMMMMMMMMMMMMMMMMMNdS/.`      \n\r"
"                sNhMMMMMMMMMMMMMMMMMMMMMMNh+.      \n\r"
"                -d. :mMMMMMMMMMMMMMMMMMMMMMMNh:`      \n\r"
"                /. .hMMMMMMMMMMMMMMMMMMMMMMMMMh.      \n\r"
"                . `sMMMMMMMMMMMMMMMMMMMMMMMMMMN.      \n\r"
"                hMMMMMMMMMMMMMMMMMMMMMMMMMMMy      \n\r"
"                +MMMMMMMMMMMMMMMMMMMMMMMMMMh      \n\r"
"                ";
const char msgWelcome[] =
"                *****\n\r"
"                *                               *\n\r"
"                *           Welcome to kirUX Operating System           *\n\r"
"                *                               *\n\r"
"                *****\n\r"
"                *                               *\n\r"
"                *                               *\n\r"
"                *           Author : Ashakiran Bhattar               *\n\r"
"                *           Version: 0.0.1                             *\n\r"
"                *           Date   : 01-Mar-2014                       *\n\r"
"                *                               *\n\r"
"                *****";

initEnvironment();
initVGA();
splashScreen(msgPicture);
splashScreen(msgWelcome);

shell();

while(1);
}

```

先简单介绍一下功能：

初始化环境 ()：

- 这是用于设置段寄存器然后设置堆栈的函数。
- 所需的参数数量没有。
- 用法：initEnvironment();

设置分辨率 ()：

- 该功能用于设置视频模式为 80*25。
- 所需的参数数量没有。
- 用法：setResolution();

清除屏幕 ()：

- 此函数用于用空格填充屏幕缓冲区。
- 所需的参数数量是无
- 用法：clearScreen();

设置光标 ()：

- 此函数用于在屏幕上的给定位置设置光标位置。
- 所需的参数数量为 2。
- 用法：setCursor(column, row);

显示光标 () :

- 该函数用于根据用户的选择启用或禁用光标。
- 所需的参数数量为 1
- 用法: `showCursor(1);`

initVGA():

- 该函数用于将视频分辨率设置为80*25, 然后清屏, 最后将光标设置在屏幕上的 (0,0) 处。
- 所需的参数数量是无
- 用法: `initVGA();`

获取 () :

- 此函数用于从用户那里获得没有回声的击键。
- 所需的参数数量是无
- 用法: `getch();`

获取字符 () :

- 该函数用于返回键扫描码和相应的ascii码
- 所需的参数数量没有。
- 用法: `putchar();`

putchar():

- 该函数用于在屏幕上显示一个字符
- 所需的参数数量为 1。
- 用法: `putchar(character);`

打印字符串 () :

- 该函数用于返回键扫描码和相应的ascii码
- 所需的参数数量没有。
- 用法: `getchar();`

延迟 () :

- 此函数用于显示空终止字符串
- 所需的参数数量为 1
- 用法: `printString(null 终止的字符串变量);`

字符串长度 () :

- 此函数用于返回空终止字符串的长度
- 所需的参数数量为 1。
- 用法: `strlen(null 终止的字符串变量);`

飞溅屏幕 () :

- 此功能用于在屏幕上显示一些花哨的图像一段时间。
- 所需的参数数量为 1。
- 用法: `splashScreen(null 终止的字符串变量);`

贝壳 () :

- 此函数用于在屏幕上显示提示
- 所需的参数数量没有。
- 用法: shell();

下面是引导加载程序加载的内核的屏幕截图。

测试内核

使用源代码：

附件是 sourcecode.tar.gz 文件，其中包含所需的源文件以及生成二进制文件所需的目录。

所以请确保您是系统的超级用户，然后开始将文件解压缩到一个目录或文件夹中。

确保安装 bochs-x64 模拟器和 GNU bin-utils 以进一步编译和测试源代码。

以下是从 zip 中提取文件后您将看到的目录结构。

应该有5个目录

- 垃圾桶
- 异
- 核心
- 日志
- 源文件

环境准备好后，请确保打开终端，然后运行以下命令

- cd \$(目录)/src
- make -f 生成文件测试
- 博克斯

截图供您参考：

屏幕 1：

这是内核执行时显示的第一个屏幕。



Bochs x86-64 emulator, <http://bochs.sourceforge.net/>

```

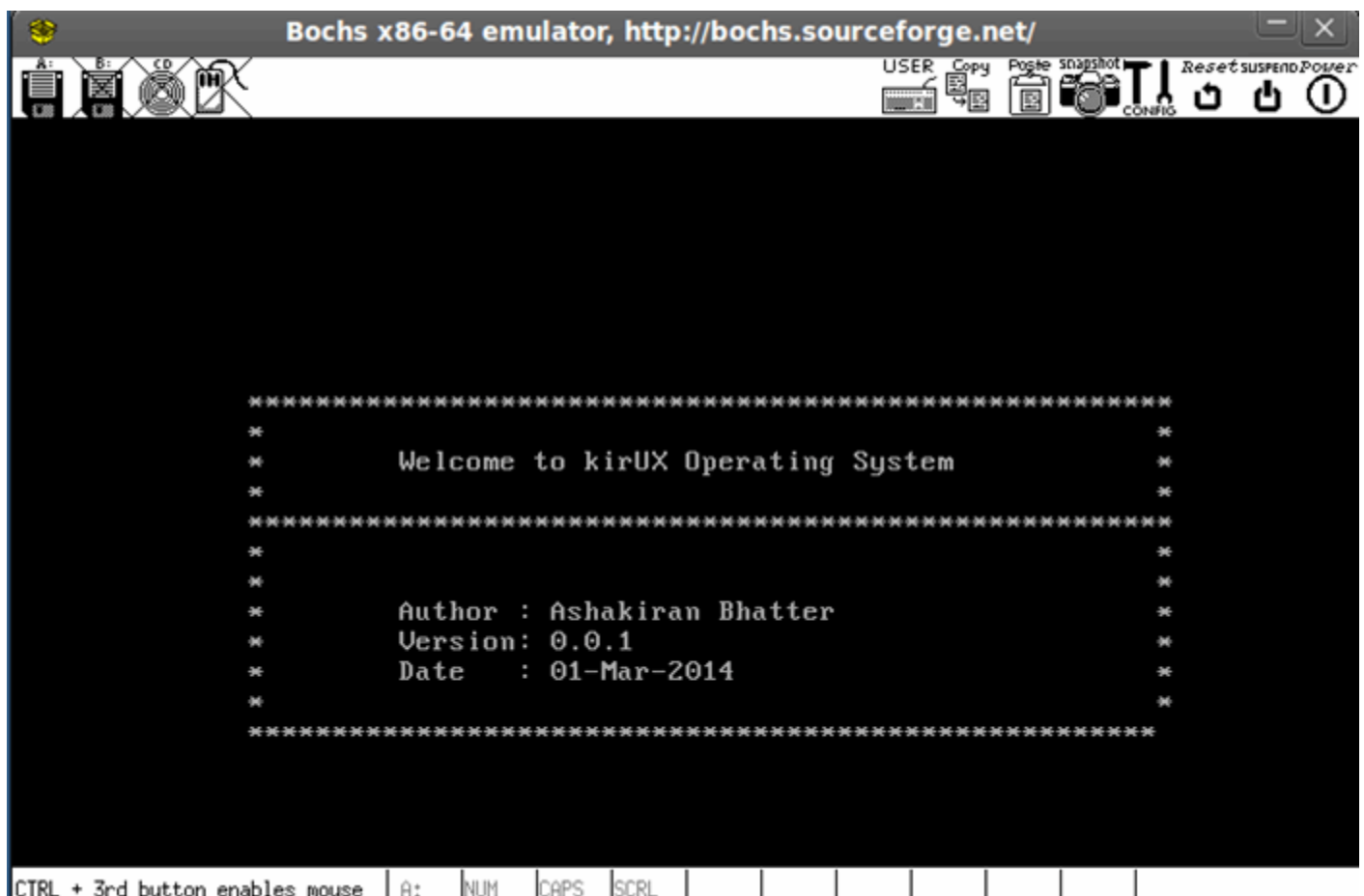
++
:ho.      .-/++/.
/hh+.      :sds:
-odds/-    .MNd/
.ydmdyo/:-/yMMMMd/
:hMMMMNNNNMMdNMMh:
-:/+++/:-:ohmNNNNNNNNNNm+-+mMNd
-+oo+osdMMMMMMMMMMMMMMMMMMMMmNMMm/
.mMMMMMMMMMMMMMMMMMMMMMMMMMMMMmho:
omMMMMMMMMMMMMMMMMMMMMMdydMMdNMMMMMMMMdo+-
:oyMMMMMMMMMMMMMMMMMNdO/hMMd+ds-:h/-yMdydMNdNdNN+
-oodMMMMMMMMMMMMMMMd: yMM+.h+.- /y /m.:mmmN
-: dMMMMMMMMMMMMMMMd. mMMNo..y/ . -/.s
-MMMMMMMMMMMMMMM- -mMMmo-./s/
+MMMMMMMMMMMMMM- .smMy: .-+oo+//:-.
.yNMMMMMMMMMMMMMMd. .+dmh+: .-:/+:.
y+-mMMMMMMMMMMMMMMm/ . /O+-
:- :MMMMMMMMMMMMMMmy/.
hMMMMMMMMMMMMMMNdS/.
sNhNMMMMMMMMMMMMMMMMMMMMNNh+.
-d. :mMMMMMMMMMMMMMMMMMMMMNNh:
/. hMMMMMMMMMMMMMMMMMMMMMMh.
sMMMMMMMMMMMMMMMMMMMMMMN.
hMMMMMMMMMMMMMMMMMMMMMMY
+MMMMMMMMMMMMMMMMMMMMMMh

```

CTRL + 3rd button enables mouse | A: | NUM | CAPS | SCRL | | | | | | | |

屏幕：2

这是内核的欢迎屏幕



Bochs x86-64 emulator, <http://bochs.sourceforge.net/>

```

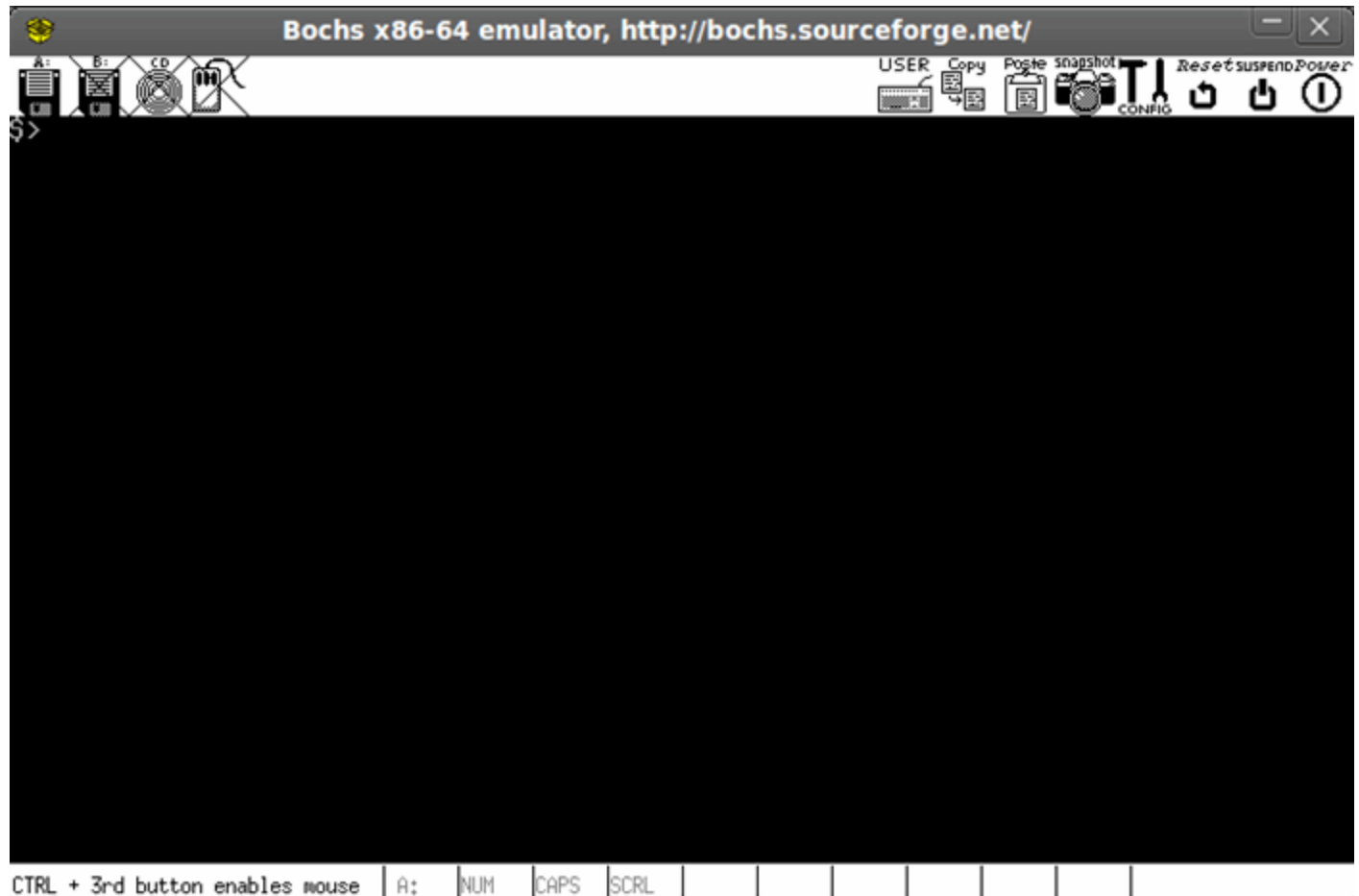
*****
*
*      Welcome to kirUX Operating System
*
*****
*
*      Author : Ashakiran Bhattar
*      Version: 0.0.1
*      Date   : 01-Mar-2014
*
*****

```

CTRL + 3rd button enables mouse | A: | NUM | CAPS | SCRL | | | | | | | |

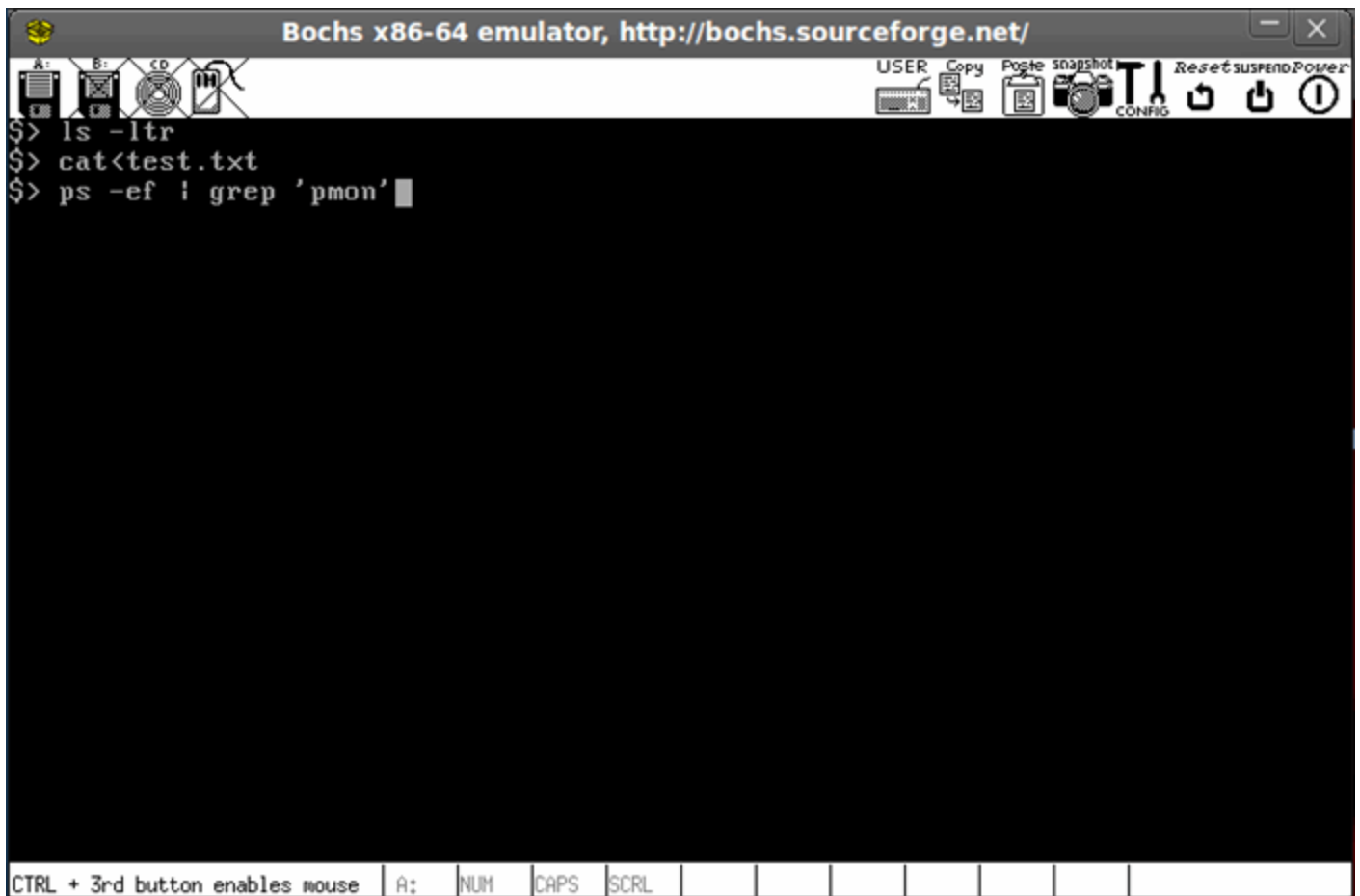
屏幕：3

这是我尝试在屏幕上显示的命令提示符，以便用户可以输入一些文本。



屏幕：4

这是用户输入的命令的屏幕截图，当用户按回车键时，屏幕会根据需要滚动。



```
$> ls -ltr
$> cat <test.txt
$> ps -ef | grep 'pmon'
```

如果您遇到任何问题，也请告诉我。我很乐意提供帮助。

结论：

我希望这篇文章能让您了解文件系统的使用及其在操作系统中的重要性。另外，希望本文能帮助您编写引导加载程序来解析文件系统以及如何使用 C/C++ 编写 16 位内核。如果您喜欢代码，您可以尝试编辑代码，然后尝试将更多功能嵌入其中。

这样做应该很有趣。再见：)

执照

本文以及任何相关的源代码和文件均根据[The Code Project Open License \(CPOI\)](https://www.codeproject.com/Articles/737545/Writing-a-bit-dummy-kernel-in-C-Cplusplus)获得许可

分享

关于作者



阿什奇兰·巴特



软件开发人员
美国 🇺🇸

手表
该会员

Ashakiran 来自印度海得拉巴，目前在美国担任软件工程师。他是一名业余程序员，喜欢编写代码。

评论和讨论

添加评论或问题



电子邮件提醒

Search Comments



第一 页上一页 下一页

很棒的信息! 🌟

南山 25-May-19 0:52

在 bochs 中没有显示 anythink 🌟

Member 13871550 25-Jul-18 16:55

海湾合作委员会版本? 🌟

Manoj Sharma 22-Apr-16 10:22

回复: 海湾合作委员会版本? 🌟

leon de boer 10-Nov-16 1:55

我的5票 🌟

raddevus 27-Dec-15 22:03

我的5票 🌟

srilekhamenon 20-Mar-15 13:05

链接器错误 🌟

Member 11233277 14-Nov-14 16:39

回复: 链接器错误 🌟

leon de boer 9-Nov-16 18:36

我的5票 🌟

fredatcodeproject 8-Jul-14 1:52

文件系统 🌟

Rye Salvador 17-Jun-14 18:15

回复: 文件系统 🌟

AshakiranBhat 18-Jun-14 2:13

回复: 文件系统 🌟

Rye Salvador 18-Jun-14 12:51

回复: 文件系统 🌟

AshakiranBhatter 18-Jun-14 14:08

回复: 文件系统

Rye Salvador 18-Jun-14 16:15

回复: 文件系统

AshakiranBhatter 22-Jun-14 9:40

回复: 文件系统

Rye Salvador 23-Jun-14 17:19

很高兴看到一篇关于 OS 核心的文章

balasubramanian.m 6-May-14 18:19

回复: 很高兴看到一篇关于 OS 核心的文章

AshakiranBhatter 11-May-14 0:20

回复: 很高兴看到一篇关于 OS 核心的文章

balasubramanian.m 3-Jul-14 2:04

回复: 很高兴看到一篇关于 OS 核心的文章

AshakiranBhatter 3-Jul-14 10:37

这是什么类型的内核?

Member 10696213 14-Apr-14 7:44

Re: 这是什么类型的内核?

AshakiranBhatter 14-Apr-14 8:19

真正的“低级”工作。

Paulo Zemek 10-Apr-14 7:49

我的5票

Southmountain 10-Apr-14 0:04

为什么要挂载磁盘映像?

Nickolakis 22-Mar-14 3:22

刷新

123 下一页

一般

新闻

建议

问题

错误

答案

笑话

赞美

咆哮

管理员

使用Ctrl+Left/Right 切换消息, Ctrl+Up/Down 切换主题, Ctrl+Shift+Left/Right 切换页面。