第三章　I/O系统

翁楚良

https://chuliangweng.github.io

2023 春　ECNU

第三章 I/O系统 提纲

- 3.1 I/O硬件原理
- 3.2 I/O软件原理
- 3.3 死锁
- 3.4 MINIX3 I/O概述
- 3.5 MINIX3 块设备
- 3.6 RAM盘
- 3.7 磁盘
- 3.8 终端

RAM盘

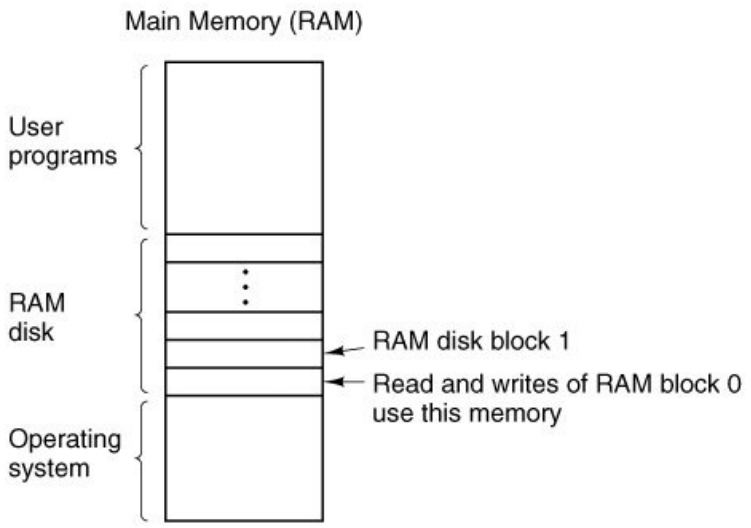
- 在内存中保留一部分存储区域，使其象普通磁盘一样使用，即RAM盘
  - RAM盘不提供永久存储，但可以快速访问

RAM盘硬件和软件

- 基本思想
  - 根据为RAM盘分配内存的大小，RAM盘被分成n块，每块的大小和实际磁盘块的大小相同。
  - 当驱动程序接收到一条读写一个数据块的消息时，它只计算被请求的块在RAM盘存储区的位置，并读出或写入该块。

数据的传输是通phys_copy汇编语言调用加以实现

一个RAM盘驱动程序可以支持多个RAM盘



MINIX3中RAM盘驱动概述

- RAM盘驱动对应六个RAM相关次设备
  - /dev/ram, /dev/kmem/, /dev/boot, /dev/mem/, /dev/null, /dev/zero
  - /dev/ram, /dev/mem, /dev/kmem, 和/dev/boot 由同一代码实现，区别在于它们作用的存储区域有所不同
    - 存储区域由数组ram_origin和ram_limit确定，通过次设备号进行索引

MINIX3中RAM盘的实现

- RAM盘驱动程序代码
  - 设备无关代码：driver.c实现主循环 (同其它磁盘驱动一样)
  - 设备相关代码：memory.c

```
/*===========================================
 *                    main
 *===========================================
PUBLIC int main(void)
{
/* Main program.  Initialize the memory dri
 m_init();
 driver_task(&m_dtab);
 return(OK);
}
```

```
/* Entry points to this driver. */
PRIVATE struct driver m_dtab = {
 m_name,     /* current device's name */
 m_do_open,  /* open or mount */
 do_nop,     /* nothing on a close */
 m_ioctl,    /* specify ram disk geometry */
 m_prepare,     /* prepare for I/O on a given minor device */
 m_transfer,    /* do the I/O */
 nop_cleanup, /* no need to clean up */
 m_geometry, /* memory device "geometry" */
 nop_signal,    /* system signals */
 nop_alarm,
 nop_cancel,
 nop_select,
 NULL,
 NULL
};
```

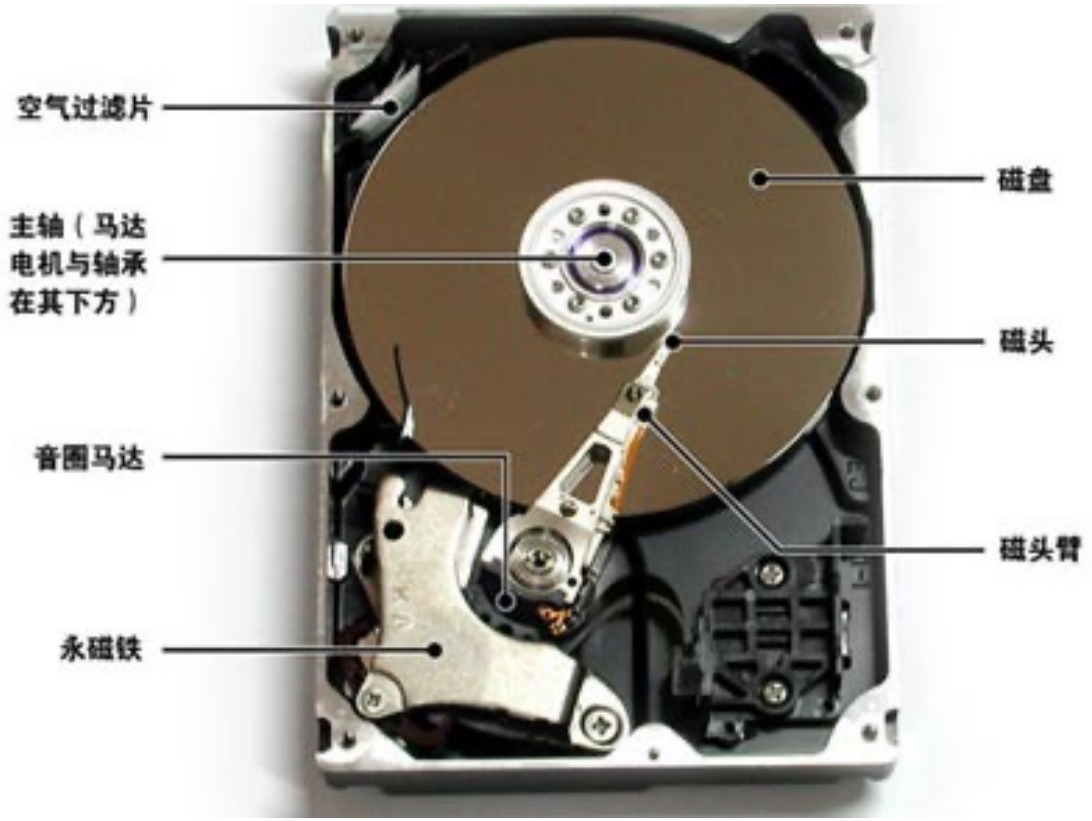
第三章 I/O系统 提纲

- 3.1 I/O硬件原理
- 3.2 I/O软件原理
- 3.3 死锁
- 3.4 MINIX3 I/O概述
- 3.5 MINIX3 块设备
- 3.6 RAM盘
- 3.7 磁盘
- 3.8 终端

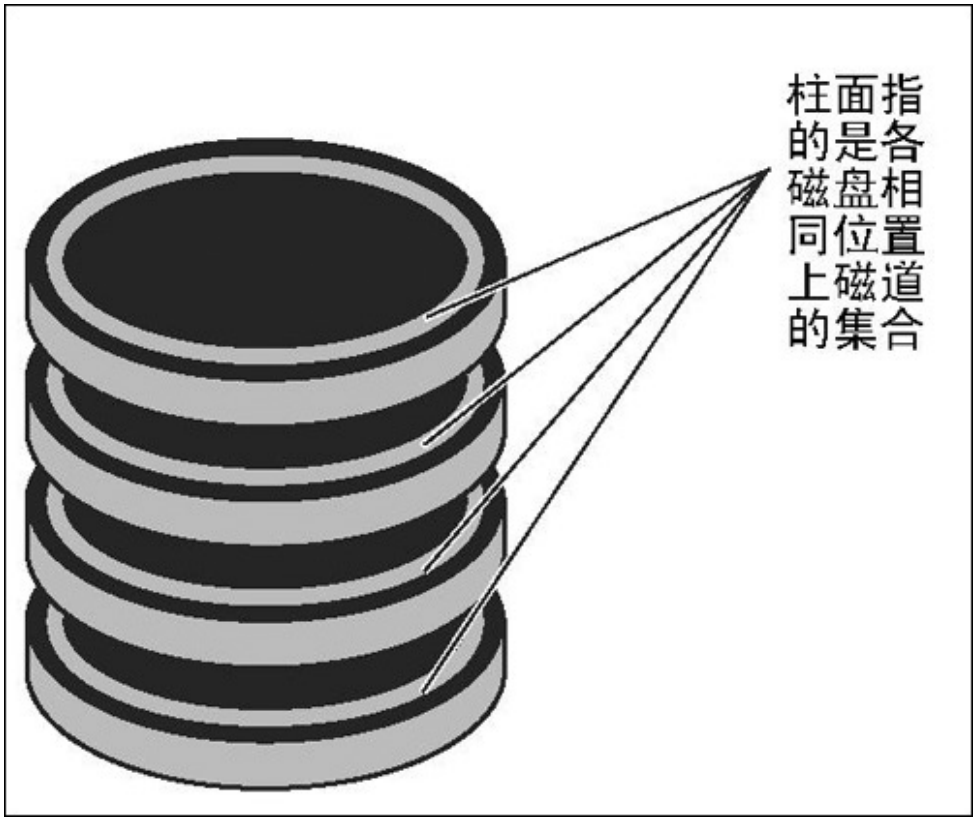
3.7 磁盘

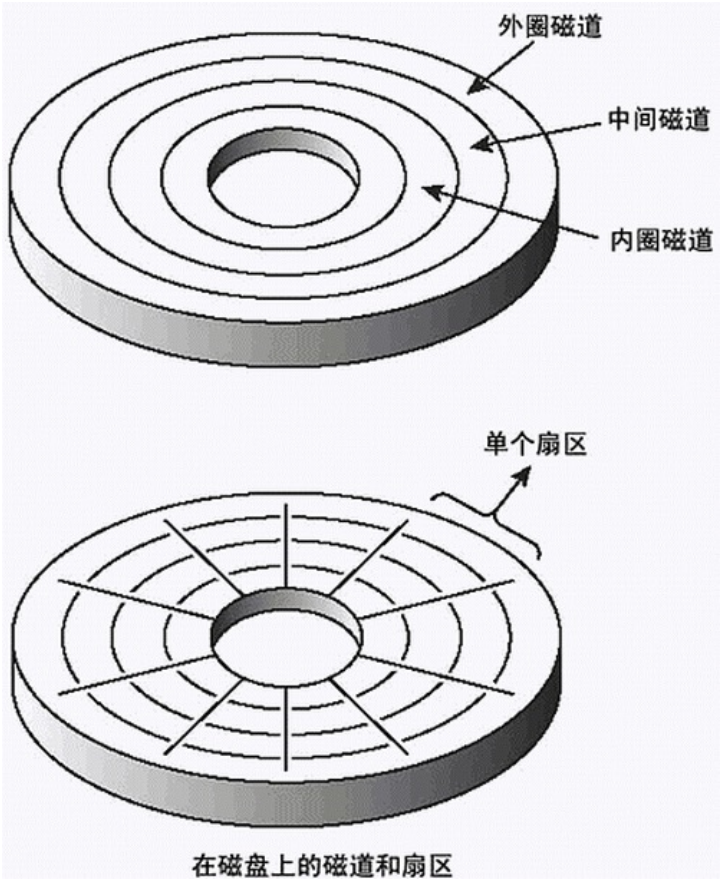
- 磁盘硬件
- RAID
- 磁盘软件
- MINIX3硬盘驱动简介
- MINIX3硬盘驱动实现
- 软盘处理

磁盘硬件



磁盘硬件



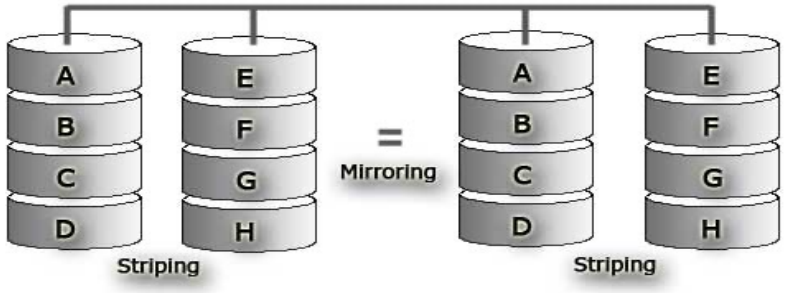


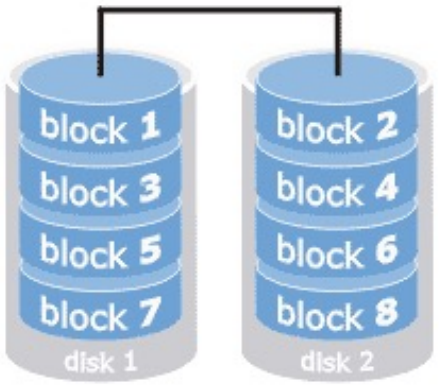
在磁盘上的磁道和扇区

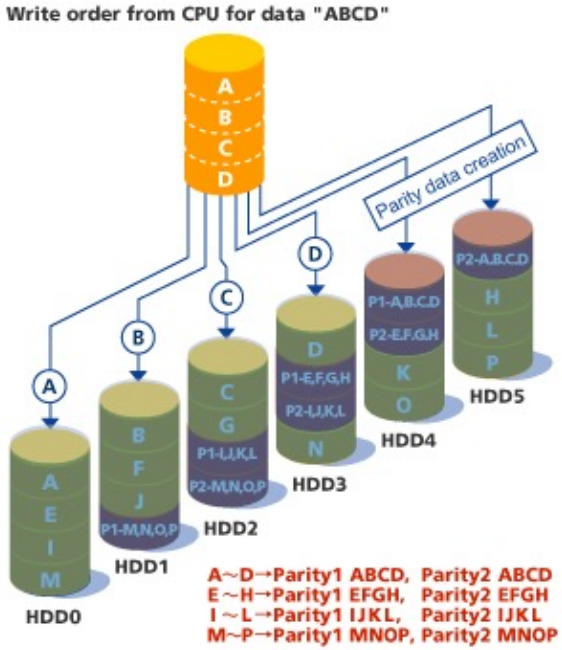
3.7 磁盘

- 磁盘硬件
- RAID
- 磁盘软件
- MINIX3硬盘驱动简介
- MINIX3硬盘驱动实现
- 软盘处理

RAID


A
B
C
D
E
F
G
H
=
Mirroring
Striping
Striping


RAID 0
striping
block 1
block 3
block 5
block 7
block 2
block 4
block 6
block 8
disk 1
disk 2


Write order from CPU for data "ABCD"
A
B
C
D
Parity data creation
HDD0
HDD1
HDD2
HDD3
HDD4
HDD5
A~D→Parity1 ABCD, Parity2 ABCD
E~H→Parity1 EFGH, Parity2 EFGH
I~L→Parity1 IJKL, Parity2 IJKL
M~P→Parity1 MNOP, Parity2 MNOP

3.7 磁盘

- 磁盘硬件
- RAID
- 磁盘软件
- MINIX3硬盘驱动简介
- MINIX3硬盘驱动实现
- 软盘处理

磁盘软件

- 读写一个磁盘块需要的时间由下面三个因素决定：
  - 寻道时间（将磁头臂移动到相应的柱面所需的时间）
  - 旋转延迟（相应的扇区旋转到磁头下面所需的时间）
  - 实际的数据传输时间。

磁盘IO时间

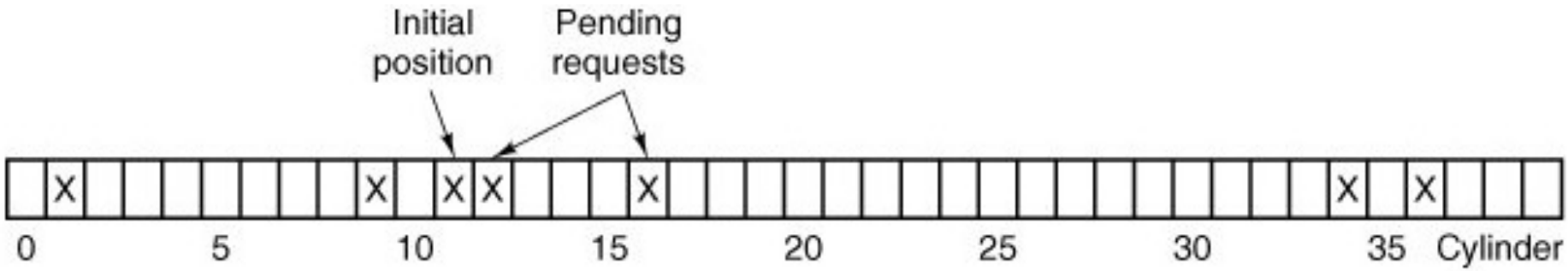
- 由于柱面定位时间在访问时间中占主要部分，合理组织磁盘数据的存储位置可提高磁盘I/O性能。

- 例子：读一个128KB大小的文件：
  - (1)文件由8个连续磁道(每个磁道32个扇区)上的256个扇区构成：
    - 20ms+(8.3ms+16.7ms)*8=220ms;
    - 其中，柱面定位时间为20ms，旋转延迟时间为8.3ms，32扇区数据传送时间为16.7ms；
  - (2)文件由256个随机分布的扇区构成：
    - (20ms+8.3ms+0.5ms)*256=7373ms;
    - 其中，1扇区数据传送时间为0.5ms；

- 随机分布时的访问时间为连续分布时的33.5倍。

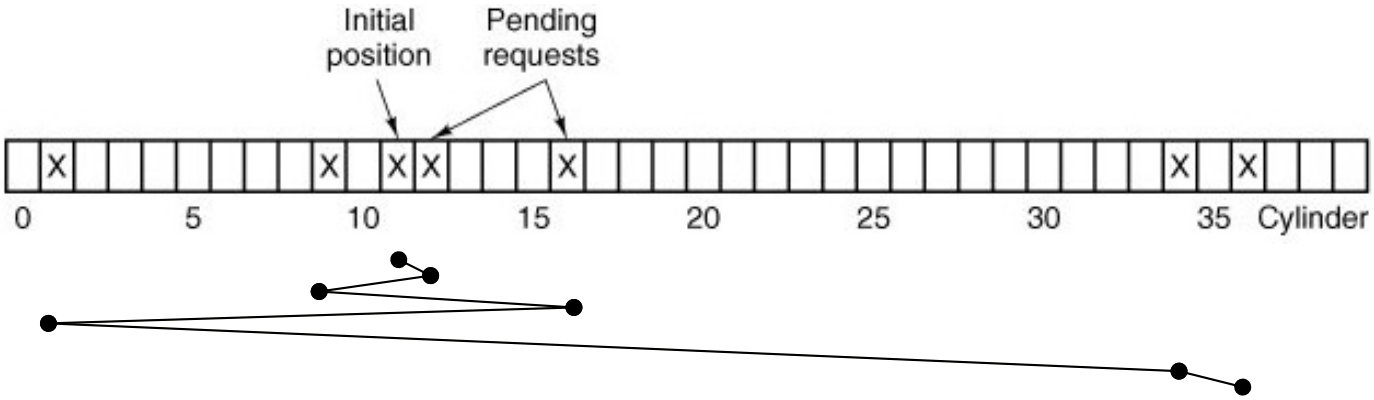
磁盘臂调度算法--FCFS算法

- 如果磁盘驱动程序每次接收一个请求并按照接收顺序执行，即先来先服务（FCFS）
  - 考虑一个具有四十个柱面的磁盘。假设一个请求到达请求读柱面11上的一个数据块，当对柱面11寻道时，又顺序到达了新请求要求寻道1、36、16、34、9和12，则它们被安排进入请求等待表，每一个柱面对应一个单独的链表。
  - 该算法中磁盘臂分别需要移动10、35、20、18、25和3个柱面，总共需要移动111个柱面。



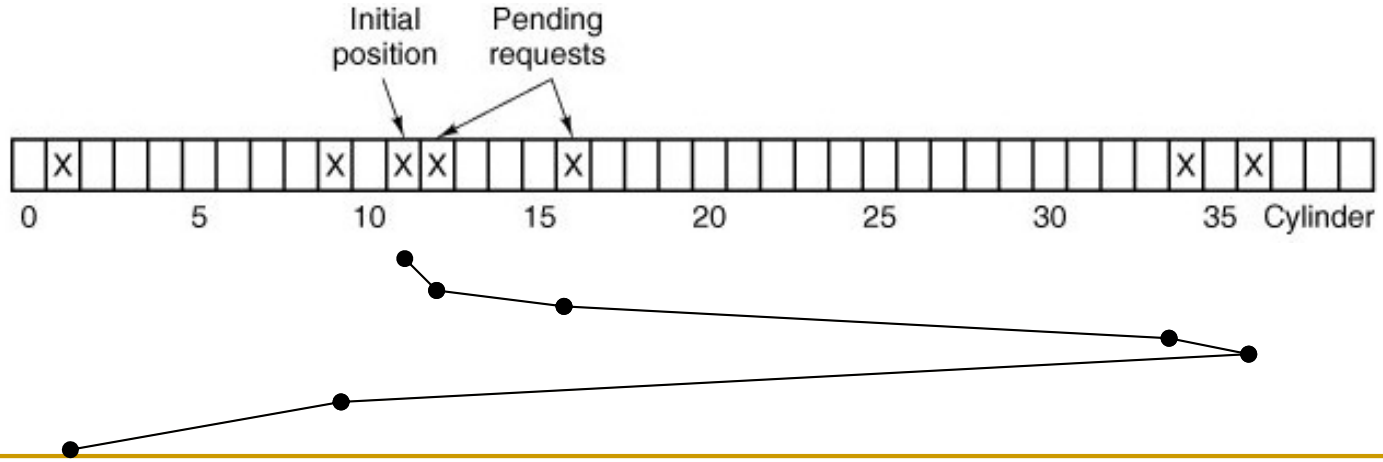
磁盘臂调度算法--最短寻道算法

- 在寻道时，选择与磁盘臂最接近的柱面请求，即最短寻道算法(SSF)
  - 考虑一个具有四十个柱面的磁盘。假设一个请求到达请求读柱面11上的一个数据块，当对柱面11寻道时，又顺序到达了新请求要求寻道1 、 36 、 16 、 34 、 9 和12，则它们被安排进入请求等待表，每一个柱面对应一个单独的链表。
  - 该算依次为12 、 9 、 16 、 1 、 34和36。按照这个顺序，磁盘臂分别需要移动1 、 3 、 7 、 15 、 33和2个柱面，总共需要移动61个柱面。



磁盘臂调度算法--电梯算法

- 电梯调度思想：电梯保持按一个方向运动，直到在那个方向上没有更远的请求为止，然后改变方向。
- 维护一个二进制位，即当前的方向：向上(外)或是向下(内)。
  - 当一个请求结束之后，磁盘驱动程序检查该位，如果是向上，磁盘臂移至下一个更高(向上)的等待请求。如果更高的位置没有请求，就翻转方向位。
  - 如果方向位设置为向下，同时存在一个低位置的请求，则移向该位置
- 假设初始方向位为向上，则各柱面获得服务的顺序是 12 、 16 、 34 、 36 、 9和1，磁盘臂分别移动1 、 4 、 18 、 2 、 27和8个柱面，总共移动60个柱面。

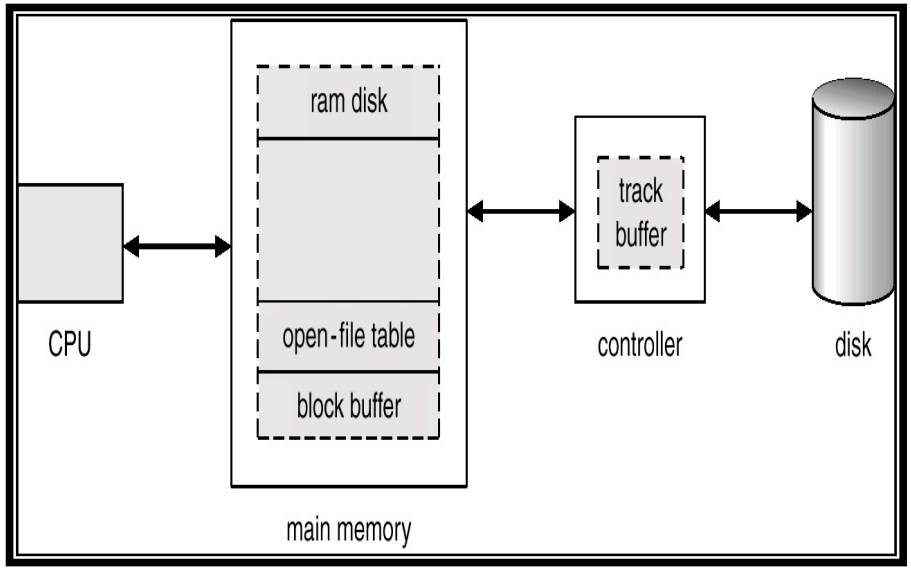


错误处理

- 磁盘驱动程序需尽可能地处理如下错误
  - 程序性错误（例如请求读写不存在的扇区）。
  - 暂时性校验和错（例如由磁头上的灰尘引起）。
  - 永久性校验和错（例如磁盘块的物理损坏）。
  - 寻道出错（例如磁盘臂应定位在第6柱面，但却到了第7柱面）。
  - 控制器错（例如控制器拒绝接受命令）。

每次一道缓冲

- 驱动程序需要把磁盘臂移到某个位置，那么读一个扇区还是读一条磁道都差不多
- 有些磁盘驱动程序通过维护一个秘密的每次一道的高速缓冲来充分利用这一特性，并且不被独立于设备的软件所感知。如果需要的扇区位于高速缓冲里，则不需要磁盘数据传输。
- 一些控制器将这个过程更进一步，在它们自己内部的存储器中实现每次一道缓冲，而对驱动程序透明。



3.7 磁盘

- 磁盘硬件
- RAID
- 磁盘软件
- MINIX3硬盘驱动简介
- MINIX3硬盘驱动实现
- 软盘处理

确盘驱动程序

- 主循环采用共享代码实现
  - DEV_OPEN：选择分区、子分区等
  - DEV_READ、DEV_WRITE、DEV_GATHER、DEV_SCATTER：准备和传输两阶段
    - do_rdwt、do_vrdwt → dp->dr_prepare、dp->dr_transfer
  - DEV_CLOSE：关闭
- 对于多个块的读写调度，由文件系统确定，由磁盘驱动根据请求队列，执行相应的数据读写操作。

硬盘驱动类型

- 针对硬盘异构性，几种处理方法
  - 为需要支持的每种硬盘控制器重新编译一个操作系统版本。
  - 在核心中编译几个不同的硬盘驱动程序，由核心在启动时自动决定使用哪一个。
  - 在核心中编译几个不同的硬盘驱动程序，提供一种方法使用户决定使用哪一个。

3.7 磁盘

- 磁盘硬件
- RAID
- 磁盘软件
- MINIX3硬盘驱动简介
- MINIX3硬盘驱动实现
- 软盘处理

MINIX3硬盘驱动实现

- 硬盘设备相关代码
  - drivers/at_wini/At_wini.c

```
/*===========================================================================*
 *                at_winchester_task                         *
 *===========================================================================*/
PUBLIC int main()
{
/* Set special disk parameters then call the generic main loop. */
  init_params();
  driver_task(&w_dtab);
  return(OK);
}
```

```
/* Entry points to this driver. */
PRIVATE struct driver w_dtab = {
  w_name,       /* current device's name */
  w_do_open,        /* open or mount request, initialize device */
  w_do_close,       /* release device */
  do_diocntl,   /* get or set a partition's geometry */
  w_prepare,    /* prepare for I/O on a given minor device */
  w_transfer,   /* do the I/O */
  nop_cleanup,      /* nothing to clean up */
  w_geometry,       /* tell the geometry of the disk */
  nop_signal,   /* no cleanup needed on shutdown */
  nop_alarm,    /* ignore leftover alarms */
  nop_cancel,       /* ignore CANCELs */
  nop_select,   /* ignore selects */
  w_other,      /* catch-all for unrecognized commands and ioctls */
  w_hw_int      /* leftover hardware interrupts */
};
```

- 设备无关代码
  - drivers/libdriver/driver.c

3.7 磁盘

- 磁盘硬件
- RAID
- 磁盘软件
- MINIX3硬盘驱动简介
- MINIX3硬盘驱动实现
- 软盘处理

软盘处理

- 简单的设备对应简单的控制器，为此操作系统就必须考虑更多的内容
  - 廉价的、缓慢的、低容量的软盘驱动器不值得配置硬盘所使用的复杂的集成控制器，因此驱动程序软件就不得不处理一些在硬盘中被隐藏于硬盘驱动器的操作。
    - 寻道（SEEK）操作需要显式地编程
  - 使软盘驱动程序复杂化的一些因素
    - 可移动介质
    - 多种磁盘格式
    - 电机控制

第三章 I/O系统 提纲

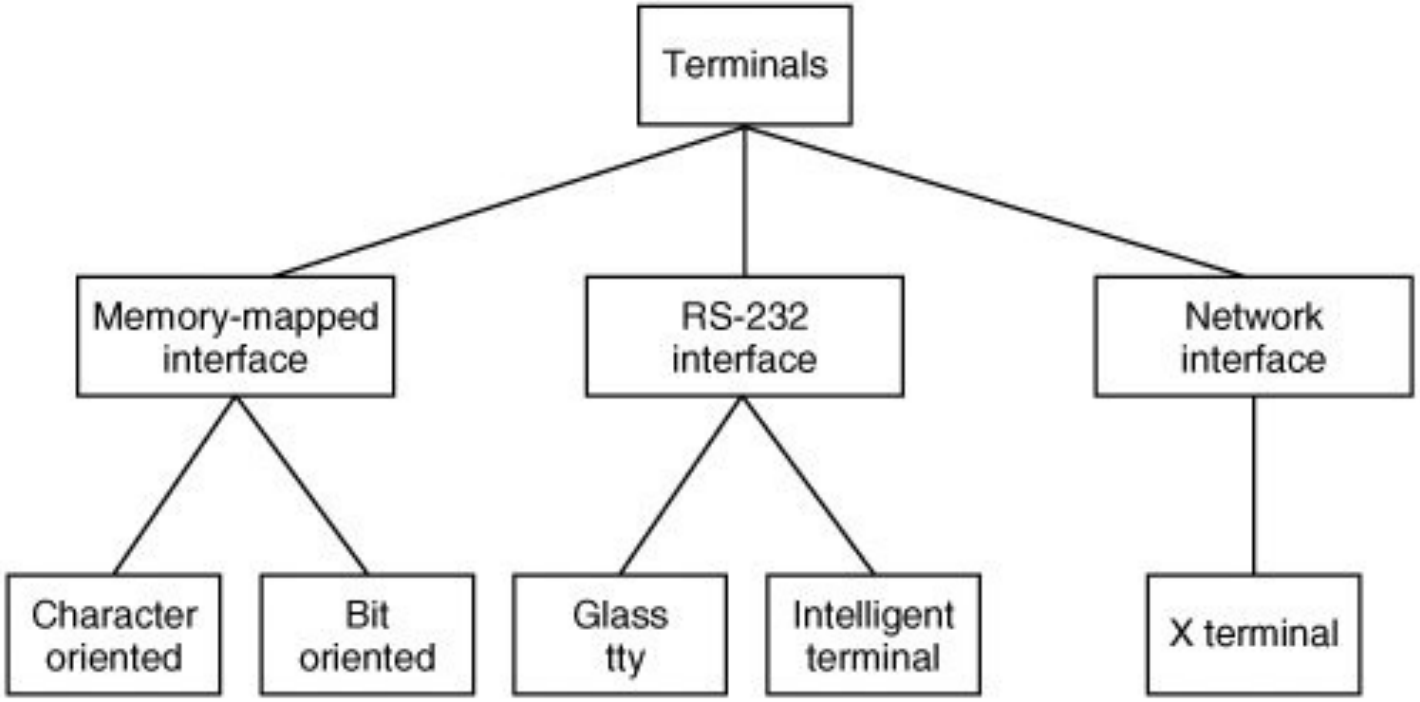
- 3.1 I/O硬件原理
- 3.2 I/O软件原理
- 3.3 死锁
- 3.4 MINIX3 I/O概述
- 3.5 MINIX3 块设备
- 3.6 RAM盘
- 3.7 磁盘
- 3.8 终端

终端

- 终端硬件
- 终端软件
- MINIX3终端驱动程序概述
- 设备无关终端驱动程序
- 设备相关软件

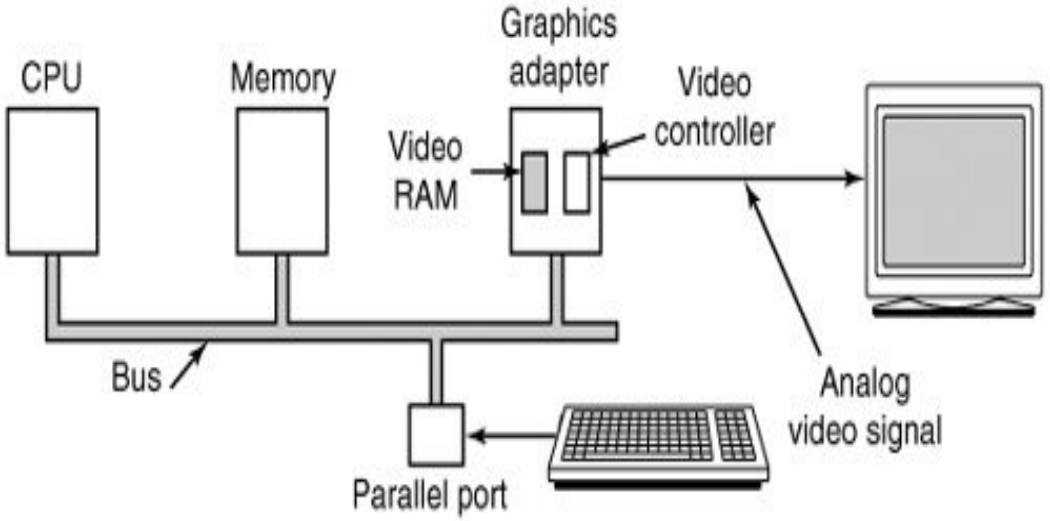
终端硬件

- 终端：用户与计算机交互的工具，包括键盘和显示器



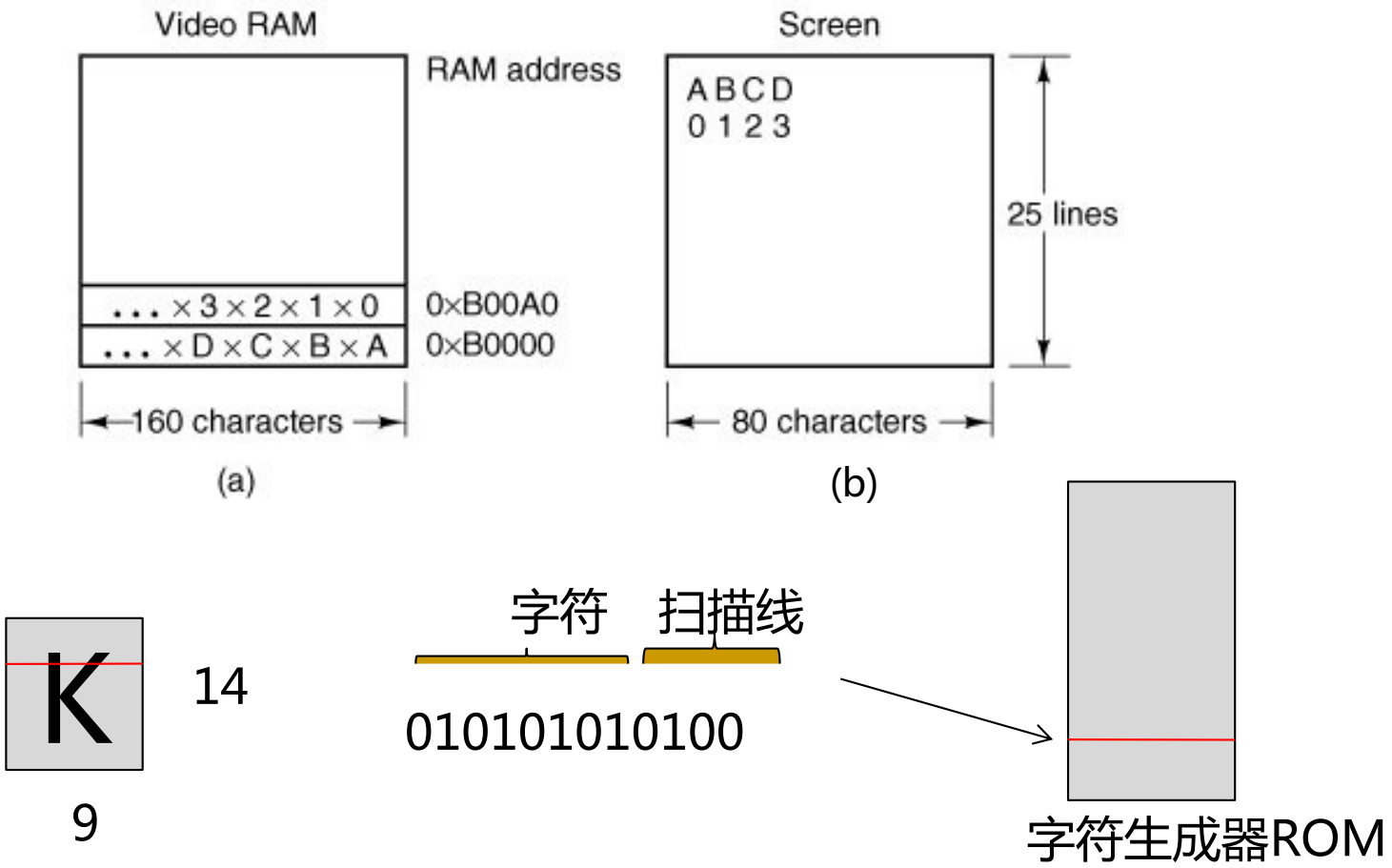
内存映射终端

- 视频RAM是计算机地址空间中的一部分，可以按地址进行访问
- 视频控制器：依据从视频RAM中取出的字符，产生视频信号



对于存储器映像显示器，键盘是与显示器分开的，它可能通过一个串行口或并行口和计算机相连。对于每一个键动作，产生CPU中断，键盘中断程序通过读I/O口取得键入的字符。

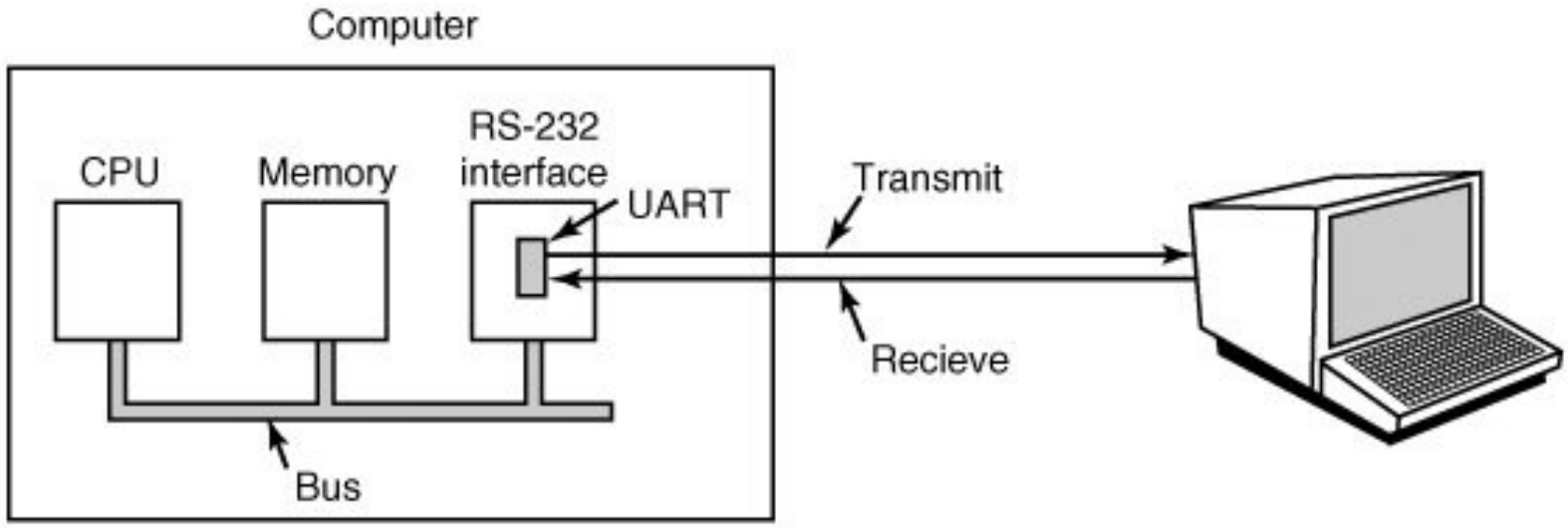
字符映射显示模式



另一种模式， 即位图模式显示，每个像素独立控制

RS-232终端

- RS－232终端是包括一个键盘和一个显示器的设备，通过一次传输一位的串行口与计算机通讯
- 为了向RS－232终端发一个字符，计算机必须一次传输一位，在字符前面加一起始位，后接一个或两个终止位为字符定界
- 一般传输速率为9600、19200或38400 bps



终端

- 终端硬件
- 终端软件
- MINIX3终端驱动程序概述
- 设备无关终端驱动程序
- 设备相关软件

输入软件

- 键盘驱动程序的工作模式
  - 生模式(非规范模式)
    - 驱动程序的工作仅仅是接收输入，并且不经任何修改就向上层传递

例：如果用户键入了dste而不是date，然后键入三个退格建和ate键来对此进行修正，最后键入一个回车键，那么用户程序将收到键入的全部11个ASCII码

  - 熟模式(规范模式)
    - 驱动程序处理行内的编辑，仅仅向用户程序传输正确的一行

输出软件

- RS－232终端
  - 输出被首先拷贝到缓冲，每一字符通过传输线送至终端
- 内存映像终端
  - 需要打印的字符在某一时刻从用户空间中取出，直接放入视频RAM中
  - 跟踪在视频RAM中当前位置，以便在那里打印可打印字符，然后向前移动输出位置

终端

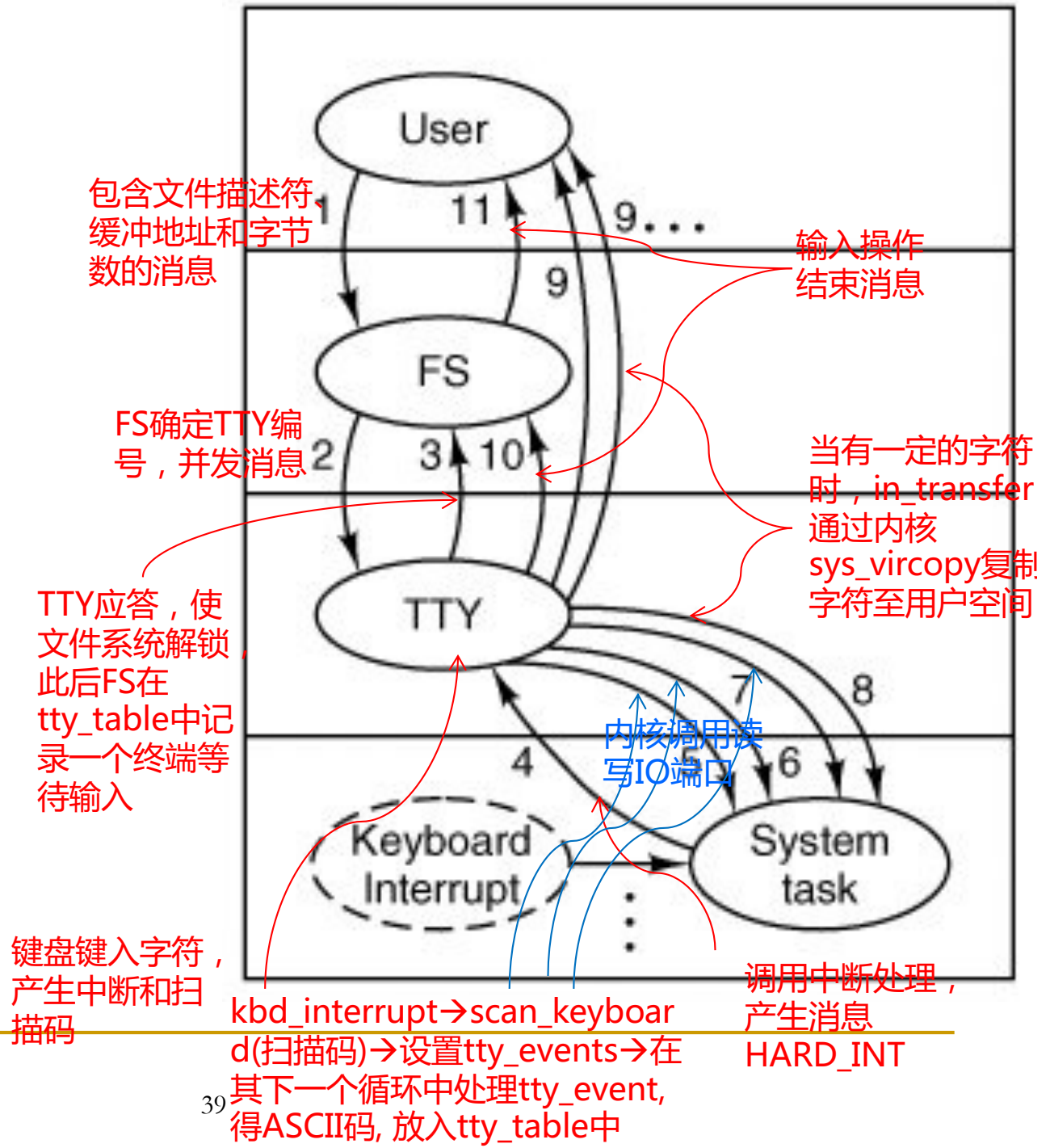
- 终端硬件
- 终端软件
- MINIX3终端驱动程序概述
- 设备无关终端驱动程序
- 设备相关软件

MINIX3终端驱动程序概述

- 终端驱动程序包含在几个文件中，构成了MINIX中最大的驱动程序。终端驱动程序既处理键盘，也处理显示器。
- 终端驱动程序接收七种消息类型：
  - 从终端读（来自FS，它代表用户进程）
  - 向终端写（来自FS，它代表用户进程）
  - 为IOCTL设置终端参数（来自FS，它代表用户进程）
  - 键盘中断发生（有键被按下或者释放）
  - 终止上一个请求（当信号发生时，来自文件系统）
  - 打开设备
  - 关闭设备

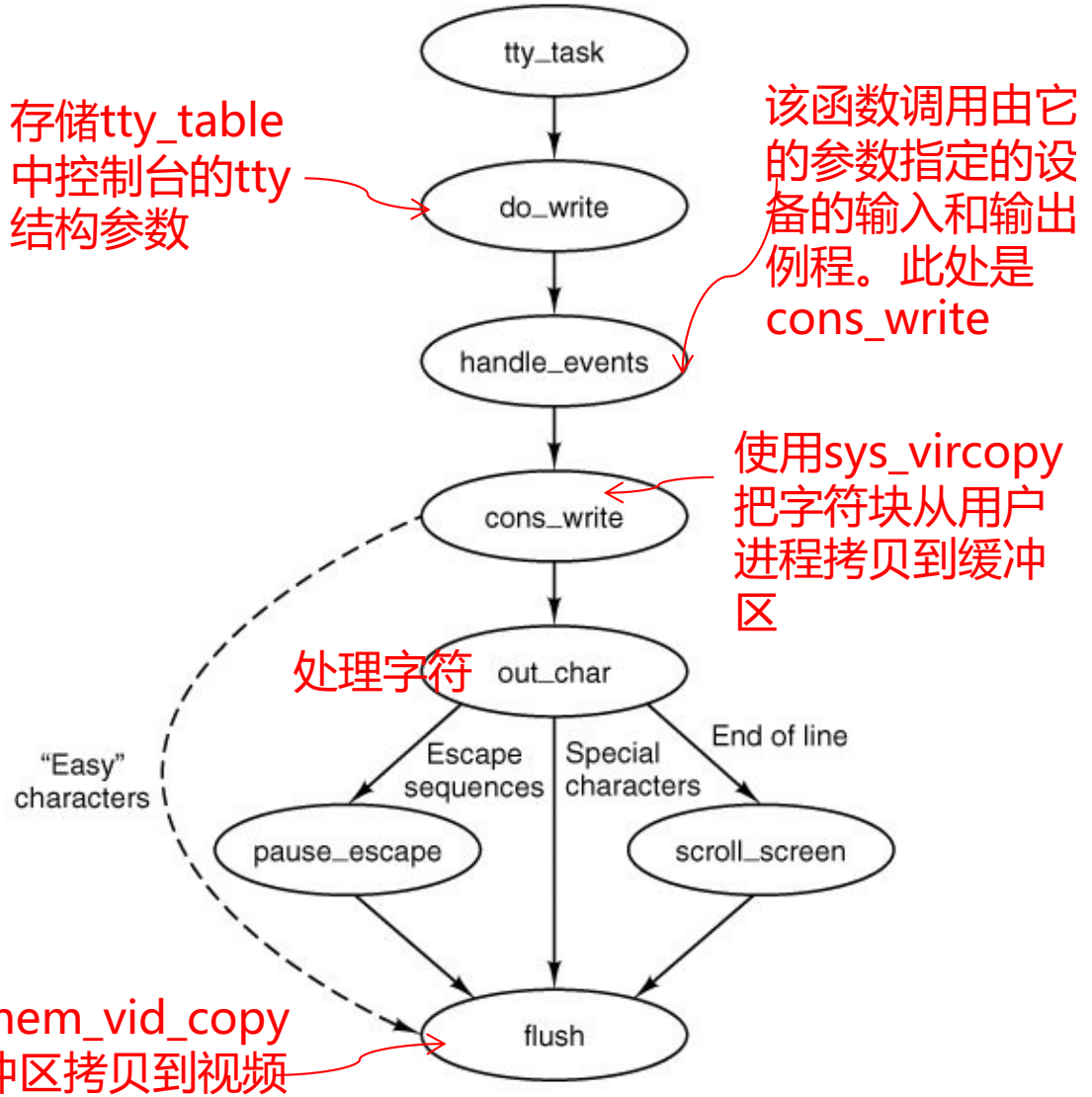
终端输入

- 终端驱动程序工作流程，以处理字符输入为例。
  - 当用户在控制台上登录时，系统为其创建一个shell，它用/dev/console作为标准输入、标准输出和标准出错。shell启动并试图通过调用库过程read读标准输入



终端输出

- 终端输出流程以处理输出字符串为例
  - 一个进程在试图显示时一般要调用printf。
  - printf调用WRITE向文件系统发送一条消息。该消息包含一个指向待显示字符序列的指针（不是字符序列本身）
  - 文件系统向终端驱动程序发送一条消息
  - 终端驱动程序取出字符并拷贝到视频RAM中。



终端

- 终端硬件
- 终端软件
- MINIX3终端驱动程序概述
- 设备无关终端驱动程序
- 设备相关软件

设备无关终端驱动程序

- 一个终端任务要支持几个不同类型的终端设备
- 数据结构tty
  - 每个终端设备都有一个这样的结构与之对应（控制台显示和键盘看作一个单一的终端）
- 主要的终端任务和设备无关的支持函数都在tty.c中定义

```
typedef struct tty {
  int tty_events;       /* set when TTY should inspect this line */
  int tty_index;        /* index into TTY table */
  int tty_minor;        /* device minor number */

  /* Input queue.  Typed characters are stored here until read by a program.
  u16_t *tty_inhead;        /* pointer to place where next char goes */
  u16_t *tty_intail;        /* pointer to next char to be given to prog */
  int tty_incount;      /* # chars in the input queue */
  int tty_eotct;        /* number of "line breaks" in input queue */
  devfun_t tty_devread;         /* routine to read from low level buffers *
  devfun_t tty_icancel;     /* cancel any device input */
  int tty_min;          /* minimum requested #chars in input queue */
  timer_t tty_tmr;      /* the timer for this tty */

  /* Output section. */
  devfun_t tty_devwrite;    /* routine to start actual device output */
  devfunarg_t tty_echo;         /* routine to echo characters input */
  devfun_t tty_ocancel;         /* cancel any ongoing device output */
  devfun_t tty_break;       /* let the device send a break */

  /* Terminal parameters and status. */
  int tty_position;     /* current position on the screen for echoing */
  char tty_reprint;         /* 1 when echoed input messed up, else 0 */
  char tty_escaped;         /* 1 when LNEXT (^V) just seen, else 0 */
  char tty_inhibited;       /* 1 when STOP (^S) just seen (stops output) */
  char tty_pgrp;        /* slot number of controlling process */
  char tty_openct;          /* count of number of opens of this tty */

  /* Information about incomplete I/O requests is stored here. */
  char tty_inrepcode;       /* reply code, TASK_REPLY or REVIVE */
  char tty_inrevived;       /* set to 1 if revive callback is pending */
  char tty_incaller;        /* process that made the call (usually FS) */
  char tty_inproc;      /* process that wants to read from tty */
  vir_bytes tty_in_vir;     /* virtual address where data is to go */
```

tty.c

```
if (tp < tty_addr(NR_CONS)) {
        scr_init(tp);
        tp->tty_minor = CONS_MINOR + s;
} else
if (tp < tty_addr(NR_CONS+NR_RS_LINES)) {
        rs_init(tp);
        tp->tty_minor = RS232_MINOR + s-NR_CONS;
} else {
        pty_init(tp);
        tp->tty_minor = s - (NR_CONS+NR_RS_LINES) + TTYPX_MINOR;
}
```

在console.c中

```
/* Fill in TTY function hooks. */
tp->tty_devwrite = cons_write;
tp->tty_echo = cons_echo;
tp->tty_ioctl = cons_ioctl;
```

具体终端处理函数

```
/*===========================================================================*
 *                tty_task                   *
 *===========================================================================*/
PUBLIC void main(void)
{
/* Main routine of the terminal task. */

  message tty_mess;       /* buffer for all incoming messages */
  unsigned line;
  int s;
  char *types[] = {"task","driver","server", "user"};
  register struct proc *rp;
  register tty_t *tp;

  /* Initialize the TTY driver. */
  tty_init();

  /* Get kernel environment (protected_mode, pc_at and ega are needed). */
  if (OK != (s=sys_getmachine(&machine))) {
    panic("TTY","Couldn't obtain kernel environment.", s);
  }

  /* Final one-time keyboard initialization. */
  kb_init_once();

  printf("\n");

  while (TRUE) {

      /* Check for and handle any events on any of the ttys. */
      for (tp = FIRST_TTY; tp < END_TTY; tp++) {
          if (tp->tty_events) handle_events(tp);
      }

      /* Get a request message. */
      receive(ANY, &tty_mess);

      /* First handle all kernel notification types that the TTY supports.
       *  - An alarm went off, expire all timers and handle the events.
       *  - A hardware interrupt also is an invitation to check for events.
       *  - A new kernel message is available for printing.
       *  - Reset the console on system shutdown.
       * Then see if this message is different from a normal device driver
       * request and should be handled separately. These extra functions
       * do not operate on a device, in constrast to the driver requests.
       */
      switch (tty_mess.m_type) {
```

终端

- 终端硬件
- 终端软件
- MINIX3终端驱动程序概述
- 设备无关终端驱动程序
- 设备相关软件

设备相关软件

- 尽管控制台显示器和键盘看作一个单一的终端，但支持它们的物理设备是完全独立的
  - 在一个标准的桌面系统中显示器使用一块插在底板上的适配卡。
  - 键盘由设计在母板上的电路支持，通过该电路与键盘内部的一个8位单片机接口。
  - 两个子设备需要完全独立的软件支持，即文件keyboard.c和console.c。

第三章 I/O系统 提纲

- 3.1 I/O硬件原理
- 3.2 I/O软件原理
- 3.3 死锁
- 3.4 MINIX3 I/O概述
- 3.5 MINIX3 块设备
- 3.6 RAM盘
- 3.7 磁盘
- 3.8 终端

作业

- 3, 4, 11, 12, 14, 15, 17, 22, 28