# 第三章：输入/输出系统

主要内容：

- I/O硬件原理
- I/O软件原理
- 死锁
- MINIX I/O系统概述



# §3.1 I/O硬件原理

- 操作系统控制所有的输入/输出设备，它所实现的I/O功能为：
  - 向设备发布命令，捕获中断并进行错误处理，提供一个与其它部分无关的接口(设备无关性接口)。
- I/O设备特点：种类繁多，特性各异、操作方式的区别很大。



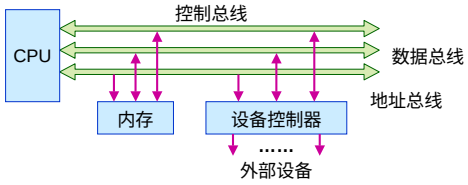
# 1．I/O设备的划分

- 按信息交换的方式分
  - **块设备(block devices)**：可以按块随机访问的设备。信息存储在可寻址的固定大小的数据块中，一般大小在512到32768字节不等。如磁盘。
  - **字符设备(character devices)**：按字符流方式操作的设备。字符设备无法编址。如打印机，Modem，鼠标等。
  - 其它：时钟，存储映像显示器等
- 按传输速率分：低速设备，中速设备，高速设备
- 按设备的共享方式：独占设备，共享设备，虚拟设备



# 2．I/O系统

- 主机I/O系统：使用专门的用于I/O的计算机完成I/O功能。
- 微型机I/O系统：I/O设备通过设备控制器连接到总线上。CPU不直接与I/O设备通信，而是与设备控制器进行通信，并通过它去控制相应的设备。设备控制器是处理机与设备之间的接口。





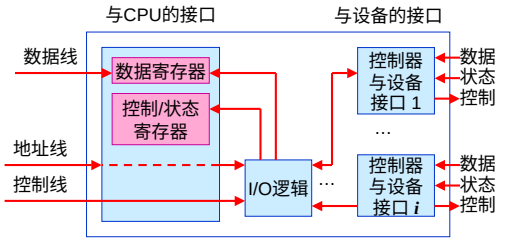

# 3．设备控制器

- I/O设备分两个部分：
  - 机械部分：设备本身
  - 电子部件：设备控制器，或适配器，它是处理机与设备之间的接口，控制设备完成相应的操作。
- CPU对设备控制器的控制主要通过读写设备控制器内对应的寄存器来完成，可以实现对设备的可编程控制，这是构建操作系统I/O管理系统的基础。
- 控制器和设备间的接口标准遵从标准接口的标准，比如ISO，IEEE，ANSI。这是一种低层次的标准，完成了对设备的具体控制。



# 设备控制器的组成

- 设备控制器与处理机的接口
- 设备控制器与设备的接口
- I/O逻辑



## 设备控制器功能

- 地址识别
- 接收和识别命令
- 设备状态的了解和报告
- 控制设备完成具体操作
  - 数据纠错，缓冲与交换 (数据信号)
  - 对设备控制信号的控制 (控制信号)
  - 设备状态的返回 (状态信号)
- 注意：设备控制器通过在它上面的接线器和设备的电缆线连接。一般可以控制不止一个设备。

7

## 4．数据交换的方式

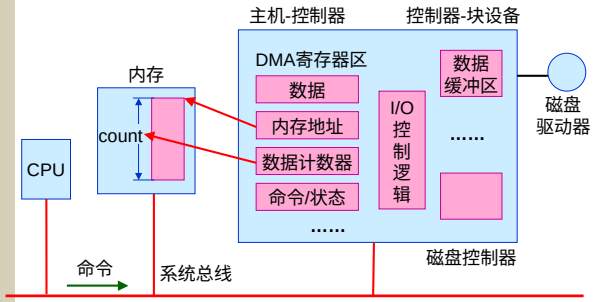
- **查询方式**：CPU直接利用I/O指令编程，实现数据的输入输出。分为：
  - 直接执行指令
  - 查询→等待→再传送
- **中断方式**：一旦设备就绪，设备控制发出中断通知CPU，进行中断处理程序，而在未就绪期间，CPU可以处理其它工作。
  - 软件要求：中断向量表、中断处理程序
  - 硬件要求：中断控制芯片、中断请求信号线
- 设想数据量很大

8

## DMA方式(Direct Memory Access)

- DMA方式特点：
  - 数据传送基本单位是数据块
  - 所传送的数据直接依靠硬件从设备送入内存的，或者相反，传送期间不受CPU干预。
  - 仅在传送一个或多个数据块的开始和结束时，才需要CPU的干预，整块数据的传送是在控制器的控制下完成。
- DMA控制器：位于主存和I/O设备之间，它是一个数据传输控制硬件。一般的情况下，和设备控制器合在一起。

9




10

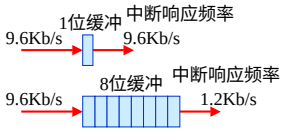
## DMA方式传送过程

- 初始化：分别设置内存起始地址和传送字节数到DMA地址寄存器和传送字节数计数器。中断允许位和启动位置1，启动设备。
- 进程进入阻塞状态，CPU运行其它进程。
- 输入设备挪用CPU总线周期，数据写入内存。
- 计数完成后发出中断信号，CPU转中断处理程序进行善后处理
- CPU返回被中断进程或调度新进程继续执行
- 缺点：设备管理复杂、提高成本……

11

## 5．数据缓冲

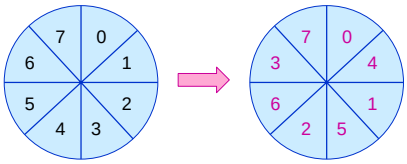
- 为什么要设置数据缓冲?
  - 速度不匹配、总线分配难并行、频繁中断
  - 例如：磁盘读取
  - 好处：避免对总线的持续占有，提高CPU和设备的并行程度。




12

## 6．磁盘交叉编址

- **交叉编址技术**：有意地跳过一些块以便为控制器留出时间供其将数据传送到内存的技术。
  - 为什么？
  - 磁盘控制器不能同时输入输出、磁盘数据流恒定






## §3.2 I/O软件原理

### 1．I/O软件的设计目标

- 实现设备无关性：指同一软件可以同时读出不同设备上的文件，而无需修改。
  - 实现统一命名，即能够使用某个简单的格式，如字符串或整数，来访问不同的设备。
- 选择和分配设备
  - 处理专用设备和共享设备
- 控制数据传输
  - 同步(阻塞)－异步(中断驱动)传输
- 错误处理



## I/O软件的层次

- I/O软件分为四个层次，自上往下分别是：
  - 用户层软件(高层)
  - 设备无关软件
  - 设备驱动程序
  - 中断处理程序(底层)



## 2．中断处理程序

- 中断(异步中断)和异常(同步中断)
  - 中断：系统不能确定事件发生的时间，例如：由外设产生的中断。
  - 异常：由CPU控制单元产生，一般由于指令执行过程中检测到错误引起，因此总发生在某个特殊指令的执行过程中。
- 中断处理程序位于I/O系统管理软件的低层，负责根据中断信号进行相关的处理。其具体操作由中断类型号决定。
- 中断信号被屏蔽，不直接到达进程，进程自动阻塞到被系统唤醒。



## 3．设备驱动程序

- 程序特点
  - 包括所有与设备相关的代码，一个设备驱动程序只能处理一种设备或一类设备
  - 使用特权级的I/O指令来控制设备，通常用汇编语言或系统编程语言编写
- 程序功能
  - 接收I/O操作指令和参数，并将抽象要求转为具体要求
  - 了解设备状态，传递有关参数，设置设备的工作方式
  - 如设备空闲，则启动设备，否则将请求挂在队列等待
  - 及时响应由控制器发来的中断，并调相应中断处理程序
  - 操作结束，进行检错，上传数据



## 4．设备无关软件

- 基本功能包括：执行适用于所有设备的常用I/O功能，并向用户层提供一个一致的接口。主要有8个功能：
- **设备命名(统一命名)**
  - 例如：主设备名，次设备名，路径

| 类型 | 保护 | 目录 | 属性 | 主设备号 | 次设备号 | 时间 | 设备名 |
|---|---|---|---|---|---|---|---|
| b | rw- rw- - - - | 1root | disk | 3 | 0 | May6 1998 | hda |
| b | rw- rw- - - - | 1root | disk | 3 | 1 | May6 1998 | hda1 |
| c | rw- - - - - - - | 1root | root | 4 | 0 | May6 1998 | tty0 |
| c | rw- rw- rw - | 1root | root | 1 | 3 | May6 1998 | zero |
| c | rw- rw- rw - | 1root | tty | 2 | 128 | May6 1998 | ptyx0 |

- 实现：系统利用保存在系统内的一张系统逻辑设备表来完成把逻辑设备引用映射到物理设备的功能。

| 逻辑设备名 | 物理设备名 | 驱动程序入口地址 |
|---|---|---|
| /dev/tty0 | 4 | 1024 |
| /dev/ptyx0 | 2 | 2046 |
| … | … | … |

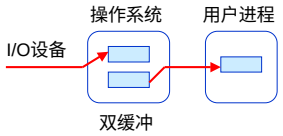
系统设备表

**设备保护**
- 实现：类似于对文件访问权限的限制，在UNIX中，通常采用rwx的权限机制。



**提供一个独立于设备的块大小－逻辑块**
- 功能：屏蔽不同存储设备的块大小差异(磁盘扇区)。

**缓冲**
- 字符设备： 用户输入字符流有可能快过CPU速度
- 块设备：速度慢于CPU，并且逻辑块大小与用户输入文件数据大小不一致




- 设想：追加20个字节到某一已存入磁盘的文件中……



**块设备的存储分配**
- 功能：定位空闲磁盘块
- 实现：用表或位图记录空闲盘块

**分配和释放独立设备**
- 系统设备表；设备控制表；设备控制器表
- 实现：利用对设备文件的操作进行申请

**错误报告**
- 实现：向上报告

**提供对设备驱动程序的统一接口**
- 功能：管理设备驱动程序



# 5．用户层软件

用户层软件：是指与用户程序链接在一起但与I/O功能有关的库例程，甚至是在核心外运行的完整程序。

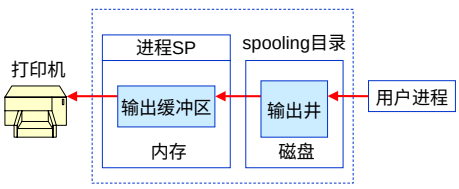
库函数(库例程)
- 作用：提供参数给相应的系统调用并调用之。
- 例如：count = write(fd，buffer，nbytes)；

**spooling系统**
- 利用程序分别模拟脱机输入、输出时的外围控制机功能，把低速I/O设备上的数据传送到高速磁盘上，或者把数据从磁盘传送到低速输出设备上。这样，在主机的直接控制下，实现脱机输入、输出功能的联机外围操作就称为SPOOLing(Simultaneaus Periphernal Operating On-line)，或称为假脱机操作。



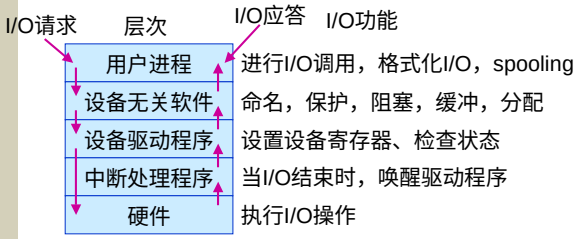
# SPOOLing系统的组成

例如打印机
- 特殊的spooling目录，竞争某一资源的进程把要操作的内容放入该目录：模拟输入、输出时的磁盘设备
- 输入、输出缓冲
- 一个专门的用于维护和操作该目录的进程

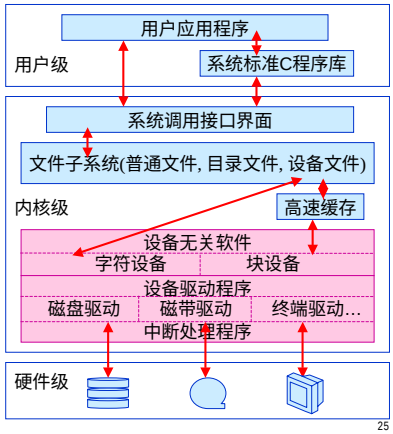





# 6．I/O软件的层次结构



## UNIX系统中用户程序,文件系统,I/O系统关系




25

## §3.3 死锁

### 1．死锁的概念

- 例1：进程：A、B；
  资源：绘图仪、CDROM
- **死锁**：若一个进程集合中的每一个进程都在等待只能由本集合中的另一个进程才能引发的事件，则这种情况被称为死锁。
- **死锁的根本原因**：死锁发生的根本原因是并发进程对资源的竞争。

26

### 2．资源

- **资源**：在计算机中需要排他访问的对象称为资源。简而言之，任何时刻只能由单个进程使用的对象就是资源，也叫**独占资源**。例如：绘图仪、CDROM、磁带机、进程表项。
- 使用资源的顺序：申请→使用→释放
- 例2：进程：A、B；
  资源：数据库中的记录R1、R2

27

- 资源的分类
  - **可剥夺式资源**：这类资源可以从拥有它的进程处剥夺而没有任何副作用。比如：内存。
  - **不可剥夺式资源**：这类资源无法在不导致相关计算失败的情况下将其从属主进程处剥夺。比如：打印机。
- 设想如果例1中一个资源换成内存……
- 死锁与不可剥夺资源有关，涉及可剥夺资源存在的潜在死锁，可以通过进程间重新分配资源化解

28

### 3．死锁原理

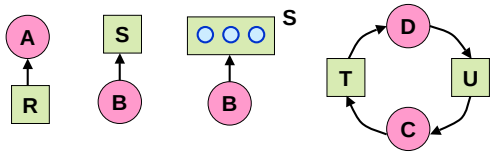
#### (1) 死锁的四个必要条件

- **互斥条件**：每一资源或者被分配给了一个进程，或者空闲。
- **保持和等待条件**：已分配到了一些资源的进程可以申请新的资源。
- **非剥夺条件**：已分配给一个进程的资源不可被剥夺，只能被占用它的进程显式地释放。
- **循环等待条件**：系统必然有一条由两个或两个以上进程组成的循环链，链中的每一个进程都在等待相邻进程占用的资源。

29

#### (2) 死锁模型

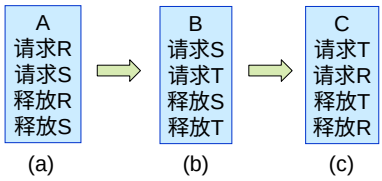
- **资源分配图**：1972年Holt提出的建立死锁四个必要条件的模型的有向图，使用其可以分析是否发生死锁。包括：
  - 两类结点：圆形－进程，方形－资源
  - 弧：从资源结点到进程结点－该资源已被占用，从进程到资源－进程在申请该资源并处于阻塞态




30

## (3) 死锁发生的诱因

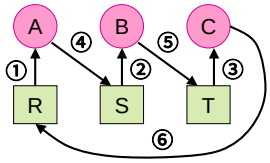
- 举例：进程A、B、C；资源R、S、T
  其中：A需要RS，B需要ST，C需要TR
- 顺序分配



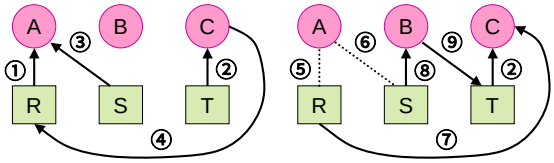

(a) (b) (c)



- 分配方案2




- 分配方案3




- 死锁产生的诱因：进程推进顺序不当



## (4) 处理死锁的四种策略

- 忽略该问题
- 检测死锁并恢复
- 通过破坏四个必要条件来预防死锁发生
- 谨慎地对资源进行动态分配，避免死锁

## 4．鸵鸟算法

- 原理：忽略死锁
- 理由
- 补救措施：备份
- 效果：UNIX



## 5．死锁检测和恢复

- 方案1：检测资源图，如有**回路**，据需要撤销某些进程。
- 方案2：周期性检测进程是否**连续阻塞**超过一定时间，如有这样的进程则撤销。
- 用途：主要用于大型主机，尤其是批处理系统。



## 6．死锁的预防

- 破坏互斥条件
  - 方法：允许进程同时访问资源。对于那些需要独占的资源，可以采用spooling技术进行假脱机。
  - 缺点：有些资源的特性不适用spooling技术，此外spooling目录也是资源。
- 破坏保持并等待条件
  - 方法1：一次性申请分配足够的资源。
  - 缺点：进程的资源需求难预测，资源被使用的时刻和持续时间不确定，无法优化。
  - 方法2：申请新资源前先暂时释放所占有的资源，只有申请成功后，才收回原先占有的资源
- 破坏不可剥夺条件

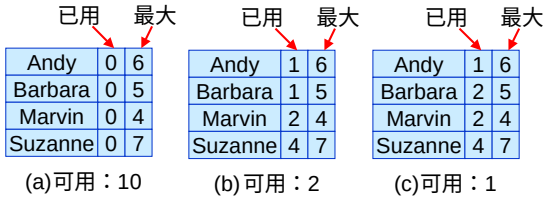


- 破坏循环等待条件
  - 方案1：任何时候一个进程只能拥有一种资源。
  - 缺点：有些情况下不可避免要同时使用两种以上资源。
  - 方案2：所有资源赋予全局编号，每个进程申请资源必须按编号顺序。
  - 缺点：资源数多，编号太大
  - 方案3：不允许进程申请编号比当前占有资源编号低的资源
  - 缺点：同上
- 总结：预防死锁的方法中施加了较强的限制条件，使它实现起来简单，但却严重损害了系统性能

# 7．死锁避免

- 思路：对每一次资源申请进行认真分析，判断是否可以安全分配。
- **(1) 单种资源的银行家算法(1965, Dijkstra)**
- 问题：一个小城镇的银行家，向一群客户分别承诺了一定的贷款额度。其保留资金小于总的最大贷款额度。如何才能保证每个客户都安全地获得贷款，确保他的信用呢?
- **安全与不安全状态**：安全状态是指系统能按某一种进程顺序来为每一个进程分配其所需的资源，直至最大需求，使每个进程都可以顺利完成。若系统不存在这样一个安全序列，则称系统处于不安全状态。



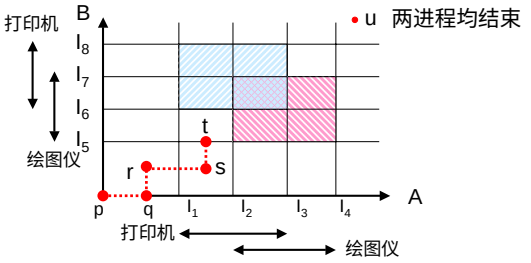



- 算法描述：对每一个请求进行检查，检查如果满足它是否会导致不安全状态。若是，则不满足该请求，否则便满足。
- **注意**：不安全状态并不一定导致死锁。



## (2) 资源轨迹图

- 系统中存在两个进程和两种资源。横纵坐标分别表示两个进程的执行过程。阴影区表示不可能的状态。




- 算法描述：对系统运行状态进行分析，检查系统出现死锁的区域和临界点，控制进程的运行，使不进入该区域



## (3) 多种资源的银行家算法

- 工作原理：判断进程的要求是否会导致不安全状态。只有当该资源要求不会导致不安全状态时，才满足，可以使系统总是处于安全状态。

| 进程 | 磁带机 | 绘图仪 | 打印机 | CD ROM |
|---|---|---|---|---|
| A | 3 | 0 | 1 | 1 |
| B | 0 | 1 | 0 | 0 |
| C | 1 | 1 | 1 | 0 |
| D | 1 | 1 | 0 | 1 |
| E | 0 | 0 | 0 | 0 |

已分配的资源

| 进程 | 磁带机 | 绘图仪 | 打印机 | CD ROM |
|---|---|---|---|---|
| A | 1 | 1 | 0 | 0 |
| B | 0 | 1 | 1 | 2 |
| C | 3 | 1 | 0 | 0 |
| D | 0 | 0 | 1 | 0 |
| E | 2 | 1 | 1 | 0 |

仍需要的资源

E=(6342)
P=(5322)
A=(1020)

E-总资源
P-已分配资源
A-剩余资源



- 步骤：
  - 检查右边矩阵是否有一行，其未被满足的设备数≤向量A(剩余资源)。如果找不到，则系统将死锁，因为任何进程都无法运行结束。
  - 若找到这样一行，则可以假设它获得所需的资源并运行结束，将该进程标记为结束，并将资源加到向量A上。
  - 重复以上步骤，直到所有进程都标记为结束。若达到所有进程都结束，则状态是安全的，否则将发生死锁。



- 例1：进程B申请一台打印机
  - 系统处于安全状态。
  - 某一安全序列为：D → A → E → C →B
- 例2：在满足进程B后，进程E继续申请一台打印机，系统是否处于安全状态？
- 优点：
  - 允许死锁必要条件中的互斥条件、保持和等待条件和不可剥夺条件存在。限制条件放松，资源利用程度提高。
- 缺点：
  - 要求每类资源的数量固定
  - 用户数要求不变
  - 要求事先说明最大资源要求
  - 没有考虑响应时间

## 8. 两阶段上锁法

- 第一阶段进程试图将其所需要的全部记录依次上锁，如成功，则在第二阶段进行数据更新和解锁。如不成功，则解开自己上的锁，等待下次重新申请。
- 缺点：仅适合于那些在第一阶段可以随时停止并重新执行的程序。

43

## §3.4 MINIX I/O系统概述

### 1. MINIX的中断处理

- 任务切换：包括从被中断进程的进程表项stackframe_s结构尾建立新堆栈，切换到核心堆栈，结束时再切换回进程表中的堆栈。

设备：将电信号送中断控制器

控制器：
中断CPU
发送中断设备的数字标识符

核心：
保存寄存器，转中断处理程序
执行驱动程序以读I/O设备
发送消息
重启动一个进程

调用程序：
将消息指针和目标存入CPU寄存器
执行软件中断指令

核心：
保存寄存器
发送和/或接收消息
重启动一个进程

44

## 2. MINIX的中断处理程序 (Interrupt Handlers)

- 磁盘中断处理程序
  - 程序见10981～10983行
  - 功能为读控制器状态寄存器的值，从而确定控制器的工作状态，把中断信号转为消息。
  - Interrupt函数将中断转换为向该设备所对应的系统任务发送一条消息。见06935～07003

```
/* acknowledge interrupt */
w_status = in_byte(w_wn->base + REG_STATUS);
interrupt(WINCHESTER);
return 1;
```

45

- 时钟中断处理程序
  - 过程：设一个按一定频率间隔递增的时钟变量pending_ticks；一个任务时间变量，记录上次运行时钟任务时的时间值，当前时间即两者的和。当时钟接收到一条消息被唤醒时，把pending_ticks值加到任务时间变量上，然后清零。最后检查相关变量，需要时向外发出消息。见11371~11469行。

```
pending_ticks += ticks;
now = realtime + pending_ticks;
if (tty_timeout <= now) tty_wakeup(now); /* possibly TTY */
pr_restart();                            /* possibly restart printer */
 ……
if (next_alarm <= now || sched_ticks == 1 &&
    bill_ptr == prev_ptr &&
    rdy_head[USER_Q] != NIL_PROC) {
      interrupt(CLOCK); /* clock task */
……
```

46

- 键盘中断处理程序
  - 键盘特性：存在半个字符。需要建缓冲队列收集输入的键码。
  - 过程：接收键盘上的每个中断，将其组织成键码队列。时钟中断不断监视变量tty_timeout，如果键码队列发生变化，则该变量变化，此时向终端任务发一条消息。收到消息后，终端任务取该字符。如果字符输入速度很快，可将若干字符积累起来放在一条消息中处理。

47

## 2. MINIX的设备驱动程序

- 程序特点：
  - 每一个设备驱动程序放在一个单独的源文件里。
  - 设备驱动程序之间，以及它们与文件系统之间可以通信，通信方式采用消息机制。
  - 有公用例程。
- 两层构造：设备相关的(device dependent)，设备无关的(device independent)。设备无关的放在一个文件里，有利于编程；而设备相关的属于它们各自的设备驱动程序，有利于灵活处理不同的硬件配置。

48

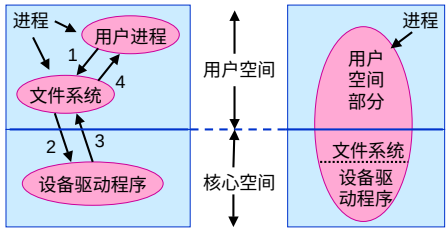
## (1) MINIX－模块化设备驱动程序

- 在MINIX中设备驱动程序是完整的进程，并且全部链入核心，共享一个公用的地址空间，层次分明，通过消息通信。

## (2) UNIX－整体式设备驱动程序

- 在UNIX中所有进程均分两部分：一个用户空间一个核心空间，通过陷入从用户到核心。所以其设备驱动程序只是能被进程的核心空间部分调用的核心过程。






MINIX－进程构造的系统　　UNIX－整体式系统

- 优缺点：MINIX的设计高度模块化，保持了较高的效率。各部分之间的接口定义清晰，更容易扩展到分布式系统。而UNIX则更高效，因为过程调用比消息传递要快。



# 3．MINIX中的块设备驱动程序

- 处理过程串行，不支持多道
- **(1) 公共部分**
- 消息结构

```
03135  typedef struct {
03136        int m_source；/* who sent Message */
03137        int m_type；/* what kind of M is it*/
03138        union {
03139          mess_1 m_m1；
03140          mess_2 m_m2；
03141          mess_3 m_m3；
03142          mess_4 m_m4；
03143          mess_5 m_m5；
03144          mess_6 m_m6；
03145        } m_u；
03146  } message；
```



# Driver Prototype

```
09009   /* Info about and entry points into the device dependent code. */
struct driver {
  _PROTOTYPE( char *(*dr_name), (void) );
  _PROTOTYPE( int (*dr_open), (struct driver *dp, message *m_ptr) );
  _PROTOTYPE( int (*dr_close), (struct driver *dp, message *m_ptr) );
  _PROTOTYPE( int (*dr_ioctl), (struct driver *dp, message *m_ptr) );
  _PROTOTYPE( struct device *(*dr_prepare), (int device) );
  _PROTOTYPE( int (*dr_schedule), (int proc_nr, struct iorequest_s
*request) );
  _PROTOTYPE( int (*dr_finish), (void) );
  _PROTOTYPE( void (*dr_cleanup), (void) );
  _PROTOTYPE( void (*dr_geometry), (struct partition *entry) );
09021   };
```



- 主过程 (I/O过程)-完成硬件初始化后，驱动程序都调用driver_task。见代码09141~09200

```
PUBLIC void driver_task(dp)
struct driver *dp;      /* Device dependent entry points. */
{  int r, caller, proc_nr;
   message mess;
   init_buffer();       /* Get a DMA buffer. */
   while (TRUE) {
      receive(ANY, &mess);
      caller = mess.m_source;
      proc_nr = mess.PROC_NR;
      switch (caller) {
      case HARDWARE: continue; /* Leftover interrupt. */
      case FS_PROC_NR: break; /* The only legitimate caller. */
      default:
        printf("%s: got message from %d\n", (*dp->dr_name)(), caller);
        continue;       }
```



```
      switch(mess.m_type) {
         case DEV_OPEN:     r = (*dp->dr_open)(dp, &mess);  break;
         case DEV_CLOSE:    r = (*dp->dr_close)(dp, &mess); break;
         case DEV_IOCTL:    r = (*dp->dr_ioctl)(dp, &mess);   break;
         case DEV_READ:
         case DEV_WRITE:    r = do_rdwt(dp, &mess);           break;
         case SCATTERED_IO:  r = do_vrdwt(dp, &mess);      break;
         default:           r = EINVAL;               break;
      }
      /* Clean up leftover state. */
      (*dp->dr_cleanup)();

      mess.m_type = TASK_REPLY;
      mess.REP_PROC_NR = proc_nr;

      mess.REP_STATUS = r;
      send(caller, &mess);    /* send reply to caller */
   }
}
```

## (2) RAM

- RAM是内存中的虚拟盘。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。当驱动程序从文件系统接收到一个数据块的读写消息时，它只计算被请求的块在RAM盘存储区的位置，并读写该块，不涉及到软、硬盘。
- 主循环操作只有两步：
  - 准备m_prepare(09756~09768)
  - 调度操作m_schedule(09771~09823)

55

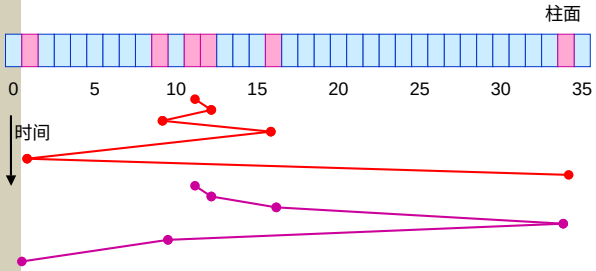
## (3) 磁盘

- 读写磁盘的时间因素
  - 寻道时间：将磁头臂移动到相应的柱面所需时间
  - 旋转时间：相应的扇区旋转到磁头下面所需时间
  - 实际数据传输时间
- 常用磁盘臂调度算法
  - **先到先服务算法**
  - **最短寻道算法**：总是选择和磁盘臂最接近的柱面请求。
  - **电梯算法**：保持按一个方向运动，直到该方向上没有更远的请求为止，然后改变方向。
    - 实现方法：维护一个二进制位，记录当前方向。
    - 改进，每次方向总是按一个方向运行

56

## 磁盘调度算法示意图

- 寻道请求：11，1，16，34，9，12




57

- 高速缓冲
  - 同磁道扇区缓冲
  - 双缓冲模式：读、写
  - 多驱动器
  - 每次一道(磁道)缓冲
- 出错处理
  - 程序性错误：例如请求读写不存在的扇区－终止
  - 暂时性校验和错：例如磁头上的灰尘引起－重复操作
  - 永久性校验和错：例如磁盘块的物理损坏－坏块替换
  - 寻道出错：例如定位6柱面，却到了7－复位重读
  - 控制器错：例如控制器拒绝接收命令－复位

58

## 4. MINIX中的死锁处理

- 方法：鸵鸟算法
- 由于MINIX不支持独占设备，所以死锁的发生只可能是在对隐含的共享资源进行访问时。而目前，没有一种已知的算法可以解决这种非显式地申请资源的问题。
- 补充：在发送消息的函数Mini_send()中，不允许接收消息的进程向调用进程发相反的请求消息，而只发应答消息。在I/O过程的调用函数call_task()中，对返回的ELOCKED信号进行进一步的处理。

```
07079    if (next_ptr == caller_ptr) return(ELOCKED);
```

59