1 复习

算法设计 IPC
过程解析 进程调度, 磁盘寻道, 斜进, 可编程时钟?

没有名词

2 绪论

操作系统作用 扩展机器, 资源管理
内核态 操作系统具有对所有硬件的完全访问权, 可以执行机器能够运行的任何指令
用户态 只能使用机器指令中的一个子集

2.1 历史

3 进程和线程

程序 指令的有序集合
多道程序设计 CPU 在各进程虚拟 CPU 中来回切换

- 通常高级语言不允许直接访问硬件, 中断处理程序中必须有汇编

例 3.0.1 进程与程序区别
解

1. 程序是指令的有序集合 (静态); 进程是程序在处理机上的一次执行过程 (动态)
2. 程序可以**长期保存**, 而进程**暂时存在**则是具有生命周期 (动态的产生和消亡)
3. 不同的进程可以包含同一程序; 同一程序在执行中也可以产生多个进程
4. 进程包括程序、数据和进程状态信息

3.1 进程模型

进程, 顺序进程 正在运行的程序, 包括**程序计数器, 寄存器, 变量的当前值**.

- 进程是某种类型的活动。它有**程序**、**输入**、**输出**、**状态**
- 单处理器可以被几个进程所共享，使用某些调度算法

3.1.1 特点

- 顺序执行
  - 顺序性
  - 封闭性
  - 可再现性
- 并发
  - 间断（异步）性
  - 失去封闭性
  - 失去可再现性

3.1.2 创建

进程创建原因

1. 系统初始化 (init)
2. 正在运行的程序执行了创建进程的系统调用 (fork)
3. 用户请求创建一个新进程 (shell)
4. 一个批处理作业的初始化 (udev?)

守护进程, daemon 后台服务

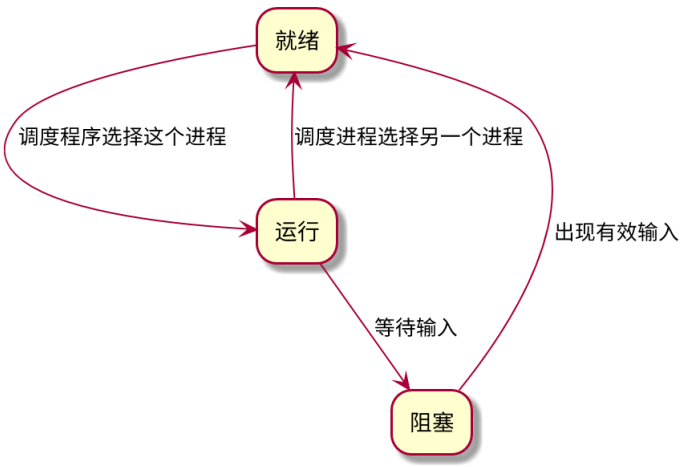
3.1.3 终止

1. 正常推出 (eof)
2. 错误推出 (panic)
3. 严重推出 (段错误?)
4. 被其他进程杀死 (KILL)

3.1.4 层次结构管理

1. 进程组 (tree)
2. 句柄 (token 控制)

3.1.5 进程状态



3.1.6 实现

操作系统维护一张**进程表: [进程表项: {程序计数器, 堆栈指针, 内存分配, 打开文件, 帐号, 调度信息}]**

3.1.7 多道程序

CPU 利用率 $= 1 - p^n$, p 为 I/O 等待时间占, n 称为多道程序道度

3.2 线程模型

- 操作系统能够进行运算调度的最小单位
- 进程作为资源分配单位
- 同一个地址空间多个控制线程

3.2.1 引入原因

1. 并行实体共享同一个地址空间和所有可用数据的能力
2. 比进程更容易创建、撤销
3. 性能的提高, 大量 I/O 的线程重叠进行, 加快运行
4. 多处理机系统，多线程可以实现真正的并行

3.3 IPC

竞争条件 共享数据, 结果取决于运行顺序
临界区 对共享内存进行访问的程序片段

3.3.1 互斥实现

1. 任何两个进程不能同时处于临界区
2. 不应对 CPU 的速度和数量做任何假设
3. 临界区外运行的进程不得阻塞其他进程
4. 不得使进程无限期等待进入临界区

3.3.1.1 忙等待

忙等待 连续测试等待
自旋锁 用于忙等待的锁

1. 屏蔽中断
2. 锁变量 (非原子)
3. **严格**轮换法, 违反 “临界区外运行的进程不得阻塞其他进程”
4. Peterson 解法

```
def lock():
    interested[current] = true
    turn = current
    while (turn == current and interested[other] == true):
        pass

def unlock():
    interested[current] = false
```

5. TSL (test and set lock, 原子版锁变量)

3.3.1.2 协助

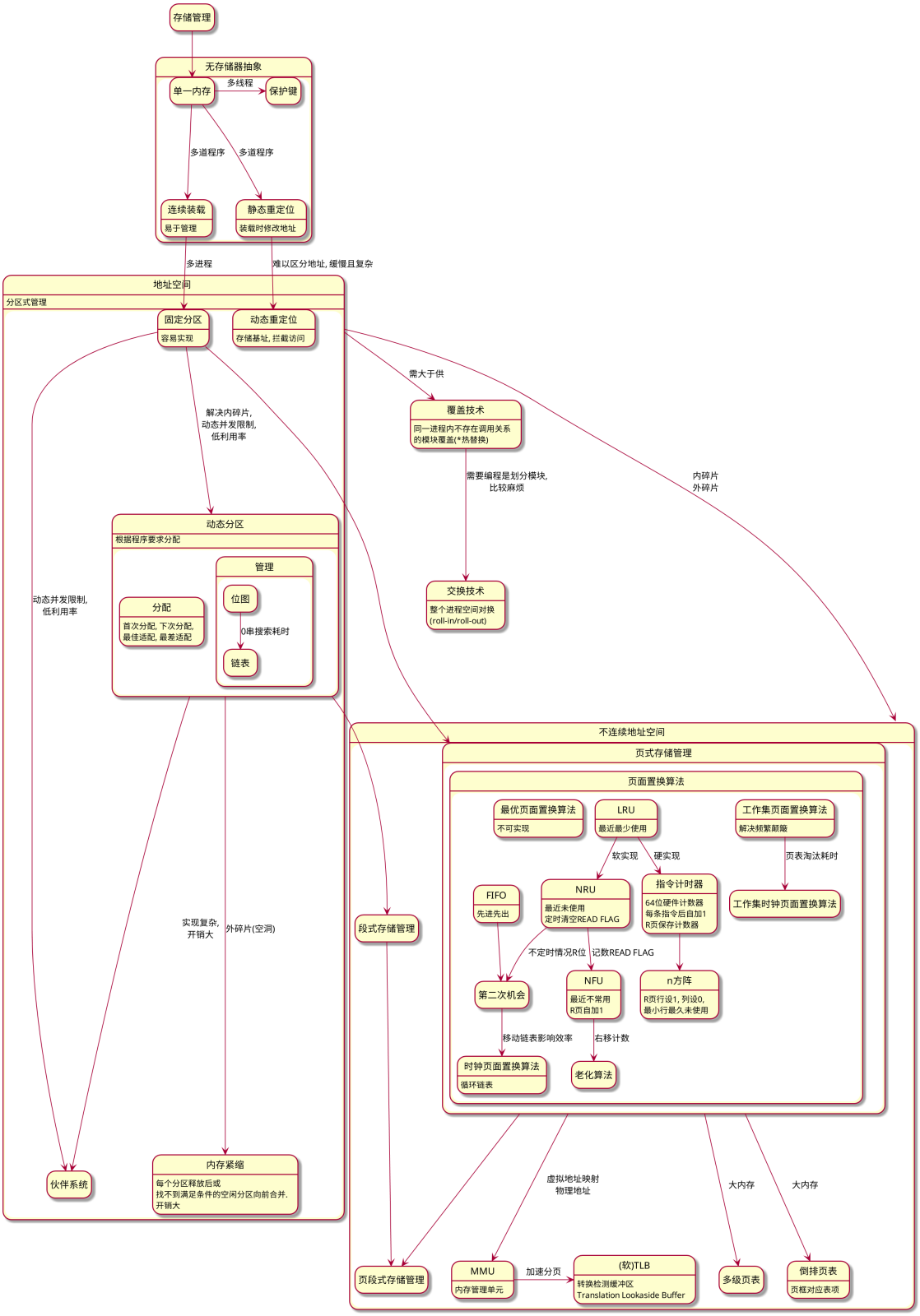
方法	wait(semaphore)	signal(semaphore)
睡眠和唤醒	`if (EMPTY) sleep(this)`	`if (NOT_EMPTY) wakeup(process)`
信号量 (原理)	`if (--s < 0) sleep(this)`	`if (++s <= 0) wakeup(sleeper[0])`
信号量	`down((int) semaphore)`	`up((int) semaphore)`
Edsger W. Dijkstra	`P((int) semaphore)`	`V((int) semaphore)`
mutex	`lock(mutex)`	`unlock(mutex)`

3.4 进程调度

调度程序（算法） 多个进程同时竞争 CPU, 选择接下来运行的进程
可调度 总 CPU 时间占用率小于 1

1. 先来先服务 (fifo)
2. 最短作业优先 (`sort (<)`)
3. 最短剩余时间优先 (抢占式最短作业优先)
4. 时间片轮轮转调度
5. 优先级调度
6. CTSS: 1 2 3 8 16
7. 老化算法: `foldl (\acc x -> acc * a + x * (1-a))`

4 内存管理


存储管理
无存储器抽象
单一内存
多线程
保护键
多道程序
多道程序
连续装载
易于管理
静态重定位
装载时修改地址
多进程
难以区分地址, 缓慢且复杂
地址空间
分区式管理
固定分区
容易实现
动态重定位
存储基址, 拦截访问
解决内碎片,
动态并发限制,
低利用率
需大于供
覆盖技术
同一进程内不存在调用关系
的模块覆盖(*热替换)
需要编程是划分模块,
比较麻烦
内碎片
外碎片
交换技术
整个进程空间对换
(roll-in/roll-out)
动态分区
根据程序要求分配
管理
位图
0串搜索耗时
链表
分配
首次分配, 下次分配,
最佳适配, 最差适配
动态并发限制,
低利用率
不连续地址空间
页式存储管理
页面置换算法
最优页面置换算法
不可实现
LRU
最近最少使用
工作集页面置换算法
解决颠繁颠簸
软实现
硬实现
页表淘汰耗时
FIFO
先进先出
NRU
最近未使用
定时清空READ FLAG
指令计时器
64位硬件计数器
每条指令后自加1
R页保存计数器
工作集时钟页面置换算法
不定时情况R位
记数READ FLAG
第二次机会
NFU
最近不常用
R页自加1
n方阵
R页行设1, 列设0,
最小行最久未使用
移动链表影响效率
右移计数
时钟页面置换算法
循环链表
老化算法
段式存储管理
实现复杂,
开销大
外碎片(空洞)
伙伴系统
内存紧缩
每个分区释放后或
找不到满足条件的空闲分区向前合并.
开销大
虚拟地址映射
物理地址
大内存
大内存
页段式存储管理
MMU
内存管理单元
加速分页
(软)TLB
转换检测缓冲区
Translation Lookaside Buffer
多级页表
倒排页表
页框对应表项

储器管理程序 操作系统中管理存储器层次的部分

4.1 任务

1. 记录哪些内存在使用，哪些内存是空闲的
2. 在进程需要时为其分配存储空间，在进程使用完毕后释放存储空间
3. 当内存无法装入所有进程时，管理内存和磁盘间的交换

4.2 无存储器抽象

直接对应物理内存

4.2.1 变体

1. 操作系统位于 RAM 底部
2. 操作系统位于顶部 ROM 中
3. 设备驱动位于顶部 ROM, 操作系统位于 RAM 底部

4.3 地址空间

地址空间 (逻辑地址) 源程序经编译后目标程序所在的一个地址范围，通常从 0 开始

存储空间 (物理地址) 内存中的物理存储单元的集合

动态重定位 使用**基址寄存器**和**界限寄存器** (可选, 解决保护), 拦截各个进程访问内存指令

4.4 交换

把一个进程完整地调入内存，使该进程运行一段时间，然后把它存回磁盘

4.5 虚拟内存

程序、数据和堆栈的总量可能超过可用的物理内存的大小。操作系统把程序当前使用的部分保留在内存中，而把其他部分保存在磁盘上

4.6 分页

内存管理单元（MMU）将虚拟地址映射为物理地址

5 文件系统

进程 进程就是一个正在执行的程序
地址空间 一个进程可用于寻址内存的一套地址集合
文件 文件是进程创建的信息逻辑单元

5.1 布局

MBR Master Boot Record，主引导记录

[MBR: [..., 分区表], 分区: [引导块, 超级块, 空闲空间管理, i-node, 根目录, 文件...]]

5.2 目录

1. 包含固定大小项的简单目录，目录项中有磁盘地址和属性
2. 目录中的每一项只是对 i-节点的引用
3. 目录下文件名
    1. 行方式, 指定文件项长度
    2. 堆方式, 指定文件名地址

5.3 连接

1. 硬链接: 目录项指向 I-节点, I-节点中记录链接数
2. 软链接: 路径

5.4 磁盘空间管理

5.4.1 文件

1. 连续分配: 简单, 性能好, 容易造成磁盘碎片
2. 链表分配: (解决碎片) 有效利用每个磁盘块, 顺序读取快, 随机读取非常慢

1. 内存表: 数组链表, -1 EOF

3. i-节点

5.4.2 磁盘块

把文件分成很多个连续的块

例 5.4.1 设磁盘每道有 128KB，旋转时间是 8.33 毫秒，平均寻道时间是 10 毫秒，若块大小是 K 字节，则数据传输率是多少
解 T=10+8.33/2+K/217*8.33(ms)

5.4.2.1 空闲块记录

1. 链表
2. 位图

5.4.3 备份

1. 所有的目录和已修改的文件相应的 I 节点号在位图中被标记
2. 其下没有包含被修改的子目录和文件的目录其 I 节点标记被删除
3. 扫描位图，转储所有被标记的目录
4. 扫描位图，转储所有被标记的文件

6 I/O

6.1 硬件原理

6.1.1 设备分类

使用可共享性分类

1. 独占设备: 任一指定的时刻**只能让一个进程使用的**设备 (打印机、磁带驱动器等)
2. 共享设备: 能够**同时让许多程序使用**的设备 (磁盘)
3. 虚拟设备: 设备本身是独占设备, 而经过某种技术处理, 可以把它**改造成共享设备**, 同时分配给多个进程.

信息组织方式

1. 块设备: **把信息存储在固定长度的块中**, 每块都有其自己的地址. 一般块的大小在 512 至 32768 字节之间. 块设备的基本特征是**每个块都能独立地读写**. 磁盘是最典型的块设备.
2. 字符设备: 字符设备发送或接收一个**字符流**, 不考虑任何块结构. 因此**不编址**, 没有任何**寻址操作**. 打印机、网络接口等是字符设备.

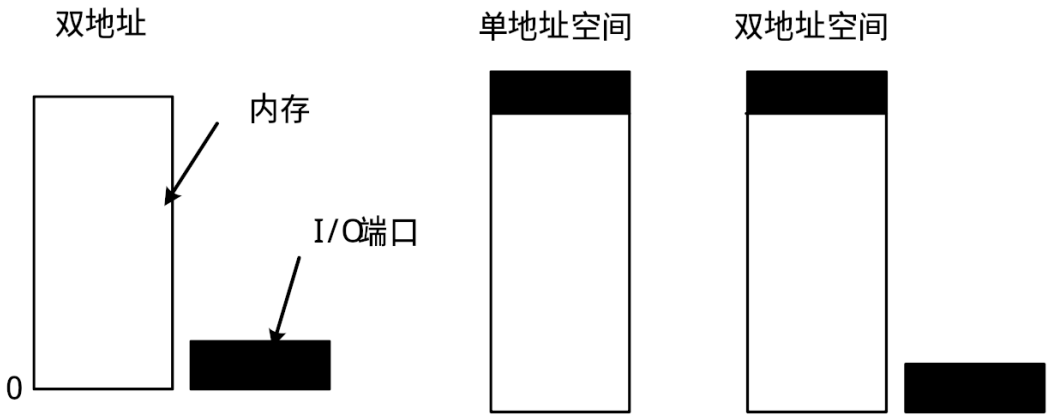
时钟 (定时中断) 不能按此分类

6.1.2 组成部分

1. 机械部分
2. 电子部分 (**设备控制器/适配器**, USB 主控等)
    - 很多控制器可以连接多个相同的设备

6.1.3 内存映射

通过内存映射与 CPU 通讯



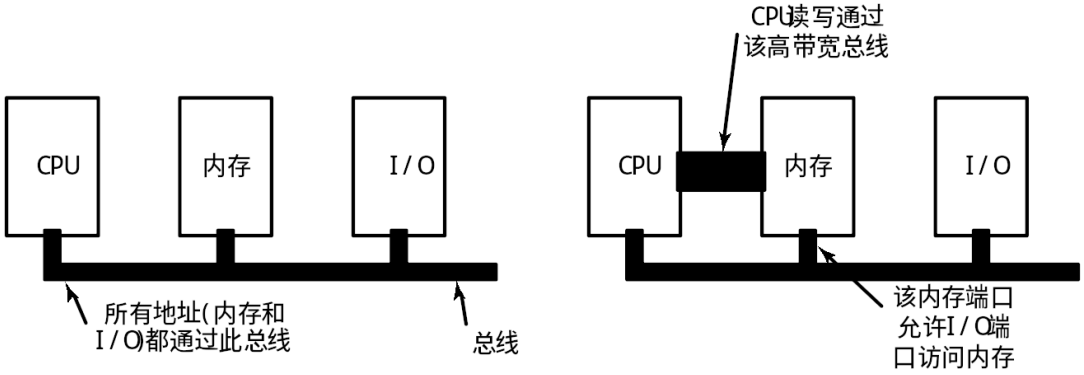
- 双地址: 分离内存和 I/O 端口地址, 独立读写指令 (IN OUT)
- 单地址: 控制寄存器映射至内存
- 混合

优点

- I/O 的设备驱动程序完全可以用 C 编写
- 使用内存映射 I/O, 无需特殊的保护机制来隔离 I/O 操作和用户进程, (避免这部分内存放入用户进程的虚拟内存)
- 使用内存映射 I/O, 每个可以引用内存的指令都同样可以引用控制寄存器

缺点

- (对于告诉内存缓存设计) 操作系统必须管理选择性高速缓存
- (单独的高速内存总线) I/O 设备无法直接查看内存地址, 需要采取特殊的措施, (连通内存和 I/O 设备)



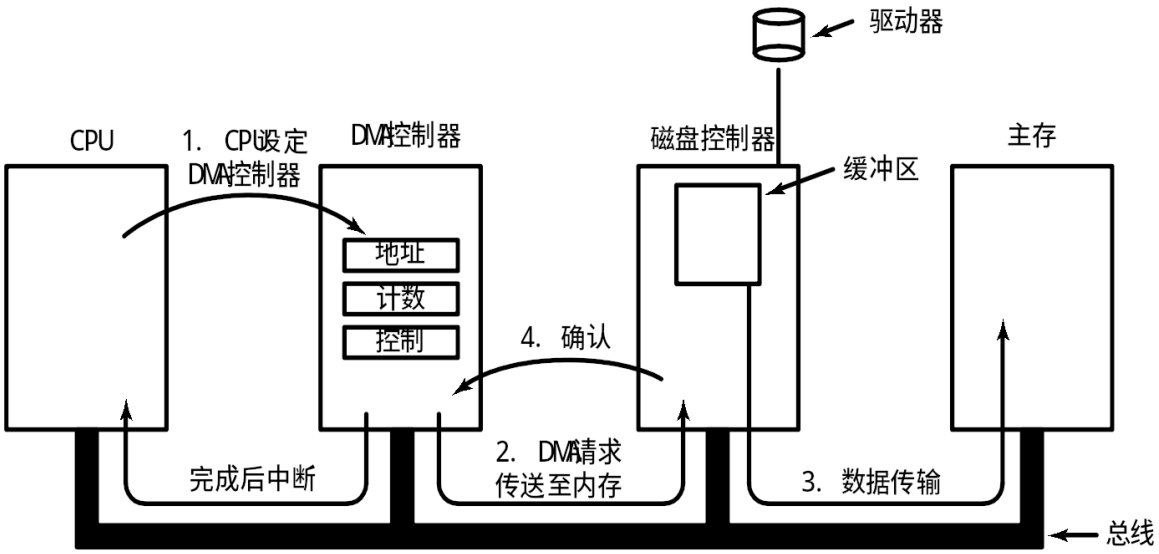
6.1.4 DMA

Direct Memory Access, 直接存储器存取, 允许某些计算机内部的硬件子系统 (计算机外设), 可以独立地直接读写系统内存, 而不需 CPU 介入处理

6.1.4.1 DMA 控制器组成

- 一个内存地址寄存器
- 一个字节计数寄存器
- 一个或多个控制寄存器 (指定要使用的 I/O 端口, 传送方向, 传送单位 (一次一个字节或一个字) 及一次突发传送中要传送的字节数)

6.1.4.2 工作流程



传送前预处理: 由 CPU 执行 I/O 指令, 对 DMA 控制器进行初始化和启动

1. CPU 对 DMA 控制器编程, 使之知道在哪里传输什么东西. DMA 控制器向磁盘控制器发出一个命令, 通知它从磁盘读数据写入其内部缓冲区, 并检查校验和. 当有效数据已经在磁盘控制器的缓冲区内时, DMA 就可以开始了

数据传送阶段: 由 DMA 控制器控制 (占用) 总线进行数据传送

2. DMA 控制器通过总线向磁盘控制器发出读请求来启动传输。
3. 写到内存是另一个标准的总线周期
4. 当写操作完成时, 磁盘控制器发送一个确认信号给 DMA 控制器, 同样也通过总线

后处理阶段: 数据传送结束, DMA 控制器向 CPU 发中断请求, 报告 DMA 操作结束. CPU 响应, 转入中断处理程序, 完成结束处理工作

5. 然后 DMA 控制器使其使用的内存地址加 1, 字节计数减 1. 如果字节计数仍然大于 0, 重复步骤 2 到 4, 直到计数为 0. 此时 DMA 发中断通知 CPU 传送已完成

6.1.4.3 占用总线模式

1. (总线一字模式) **周期窃取** (cycle stealing), 因为设备控制器偷偷地潜入, 不定期地从 CPU 手中偷取总线周期, 稍许延迟 CPU 操作
2. (总线块模式) **突发模式** (burst mode), DMA 控制器告诉设备获取总线, 发出一系列传输, 然后释放总线, 效率更高

6.2 软件原理

6.2.1 程序控制 I/O

即忙等待. ~~用户进程控制内存/CPU 与外设的信息传递, 它利用 CPU 发命令启动设备, 并检测等待设备准备好 (忙等待), 即 CPU 控制完成所有的 I/O 操作~~

简单, 但 CPU 及所有外设串行工作 (单线程), 适合简单单片机

6.2.2 中断驱动 I/O

即中断异步流. ~~CPU 在 I/O 操作时, 将进程挂起, 转去做其他工作, 直至 I/O 完成后, 设备控制器发中断通知 CPU, 再唤醒进程~~

缺点: 控制器中缓冲区一般较小, 一旦装满就发中断, 造成**中断次数太多**, 中断需要花时间, 浪费相当多的 CPU 时间. (如打印机, 每个字符输出完毕都产生一个中断)

6.2.3 DMA

即 DMA 简化 CPU 操作. ~~类似于程序控制 I/O, 但是用 DMA 控制器代替 CPU 来完成所有工作~~

优点:

1. 操作均由硬件电路实现, 传输速度快

2. CPU 仅在初始化和结束时参与, 基本上不干预数据传送, 可减少 CPU 开销
3. CPU 与外设并行工作, 效率高

缺点: DMA 控制器的速度不如 CPU

6.3 软件设计

设备无关性, 统一命名, 错误处理, 同步异步, 缓冲, 共享/独占设备占用权限管理

1. 硬件: 实际 I/O 操作
2. 中断处理程序: 中断唤醒驱动程序
3. 设备驱动程序: 设定设备寄存器, 检查状态
4. 与设备无关的软件: 命名, 保护, 阻塞, 缓冲, 分配
5. 用户进程: I/O 调用, 格式化 I/O, 假脱机

6.3.1 中断处理程序

6.3.1.1 中断

中断 在计算机执行期间, 系统内发生任何不寻常或非预期的**急**需处理的事件, 使得 CPU 暂时中断当前正在运行的程序而**转**去执行相应的事件处理程序, 待处理**完**毕后又返回原来被中断处继续执行或调度新的进程。

中断源 引起中断发生的事件

中断请求 中断源向 CPU 发出的请求中断处理信号

中断响应 CPU 收到中断请求后转到相应的事件处理程序的过程

关中断/开中断 CPU 内部的 PSW 的中断允许位被清除/被设置, 不允许/允许 CPU 响应中断。用于保证某段程序执行的原子性

中断屏蔽 ~~在中断请求产生后，系统有选择地封锁一部分中断而允许另一部分仍能得到响应。有些具有最高优先级的中断不允许被屏蔽。~~

中断应该隐藏在操作系统底处，隐藏中断的最好方法就是启动 **I/O 操作的驱动程序阻塞自己**直到 I/O 完成并产生一个中断。

分类

- 中断源
  - 外中断 (中断)
  - 内中断 (陷阱)

区别:

- 陷阱通常由正在执行的现行指令引起，而中断则由与现行指令无关的中断源引起。
- 陷阱处理程序提供的服务为当前进程所用，而中断处理程序提供的服务则不为当前进程所用
- CPU 在执行完一条指令之后，下一条指令开始之前响应中断，而在一条指令执行中也可以响应陷阱。

6.3.1.2 处理过程

1. CPU 检查响应中断的条件是否满足：有来自于中断源的中断请求、CPU 允许中断。
2. 如果 CPU 响应中断，则 CPU 关中断，使其进入不可再次响应中断的状态
3. 保存被中断现场。为了在中断处理结束后能正确地返回到断点，必须保存 PSW、PC 等寄存器的值。
4. 分析中断原因，调用中断处理程序。系统一般都针对不同的中断源编制不同的中断处理子程序 (陷阱处理子程序)。这些子程序的入口地址存放在内存的特定单元中。不同的中断源也对应不同的 PSW，这些 PSW 放在相应的内存单元中，与中断处理子程序构成中断向量。系统根据中断向量表找到中断处理子程序的入口和对应的 PSW
5. 执行中断处理子程序
6. 退出中断，恢复被中断进程的现场或调度新进程占用 CPU。
7. 开中断，CPU 继续执行。

6.3.2 设备驱动程序

即硬件的软件接口.

不同设备的寄存器个数和命令含义都不同——设备驱动程序 (device driver) 是操作系统中唯一知道控制器有几个寄存器及它们用途的程序 (每个设备控制器都有一个或多个寄存器用来接受命令或表示设备的状态, 它们通过读写命令按址存取. 通常有数据寄存器, 状态, 操作方式寄存器等). 每个驱动程序通常处理一种设备, 或者最多处理一类紧密相关的设备.

分类 块设备, 字符设备

6.3.2.1 任务

接受来自上层的与设备无关软件的抽象请求, 并执行这个请求

1. 检查输入参数是否有效, 如有效, 将抽象请求转换成具体形式, 如磁盘的块号转换成磁道号、柱面号、扇区等
2. 检查设备是否空闲, 如在使用, 则请求排队
3. 根据请求完成的操作, 确定发送哪些命令, 向控制器中的寄存器写入命令, 并检查控制器是否收到命令

6.3.2.2 特点

1. 与 I/O 设备的硬件结构密切联系
2. 设备驱动程序中全部是依赖于设备的代码
3. 设备驱动程序是操作系统底层中唯一知道各种输入输出设备的控制器细节
4. 执行确定的缓冲区策略
5. 进行比寄存器接口级别层次更高的一些特殊处理, 如代码转换等

6.3.2.3 功能

1. 向有关输入输出设备的各种控制器发出控制命令, 并且监督它们的正确执行, 进行必要的错误处理
2. 对各种可能的有关设备排队, 挂起, 唤醒等操作进行处理
3. 执行确定的缓冲区策略
4. 进行比寄存器接口级别层次更高的一些特殊处理, 如代码转换等

5. 发出命令后, 由控制器控制命令的执行, 此时驱动程序

    1. 阻塞自己, 等待控制器完成操作后发中断唤醒
    2. 若操作完成几乎没有延时, 则驱动程序不阻塞

6. 操作完成后, 驱动程序要检查是否有错, 若一切正常, 驱动程序负责将数据送到与设备无关的软件层

7. 向调用者返回一些用于错误报告的状态信息. 若还有其它请求在排队, 则选择一个启动, 否则等待下一个请求

6.3.3 与设备无关的 I/O 软件

实现一般设备都需要的 I/O 功能, 并向用户层软件提供一个统一的接口

1. 对设备驱动程序统一的接口
2. 缓冲技术
3. 错误报告
4. 分配和释放独占设备
5. 提供与设备无关的数据单位: 字符数量, 块尺寸

6.3.3.1 对设备驱动程序统一的接口

1. 设备命名: 通过路径名寻址设备
2. 设备保护: 检查用户是否有权访问所申请设备

6.3.3.2 缓冲

传输速率, 时间约束, 不能直接送达目的地

1. 无缓冲
2. 用户空间缓冲
3. 内核空间单缓冲
4. 内核空间双 (轮询) 缓冲

缺点: 如果数据被缓冲多次, 性能就会遭受损失

6.3.3.3 错误报告

驱动程序无法处理的错误 (小部分), 错误处理的框架 (设备无关)

- 编程错误
- 实际的 I/O 错误类 (例如试图写一个被损坏的磁盘块，或者尝试读取一个关闭的摄像机, 若驱动程序无法处理)

6.3.3.4 分配和释放独占设备

- 接受/拒绝
- 阻塞队列

6.3.3.5 与设备无关的块大小

高层只需处理抽象的, 全部使用相同块容量的设备. 隐藏硬件区别

6.3.4 用户空间的 I/O 软件

- 库过程: 统一 I/O 接口库. ~~系统调用通常由库过程实现, 输入和输出的格式化由库过程完成~~
- 假脱机系统: 将操作内容暂存到独立空间. ~~Spooling 是在多道程序系统中处理专用 I/O 设备的一种方法. 解决方法就是创建一个特殊的进程称为端口监控程序 (daemon), 以及一个特殊的目录, 叫做 spooling 目录 (spooling directory). 为了要打印一个文件, 进程首先产生整个要打印的文件, 而且把它放到 spooling 目录中. 然后由端口监控程序, 它是允许使用打印机的特殊文件的唯一进程, 来打印目录中的文件~~

6.4 I/O 实例: 盘

RAID 廉价磁盘冗余阵列
柱面斜进量 寻道时间内经过的扇区

6.4.1 磁盘臂调度算法

1. 先来先服务

2. 最短寻道时间优先
3. 电梯算法

6.4.2 可编程时钟

7 死锁

7.1 概念

7.1.1 资源

资源就是在任何时候都**只能被一个进程使用**的任何对象

可剥夺资源: 可以从拥有它的进程中剥夺而不会产生任何副作用 (存储器)

不可剥夺资源: (刻录机)

7.1.1.1 使用步骤

1. 请求资源
2. 使用资源
3. 释放资源

7.1.2 死锁

如果在一个进程集合中的每个进程都在等待只能由该集合中的另一个进程引发的事件，该组进程就被死锁

7.1.3 产生条件

- 互斥: ~~每个资源每次只允许一个进程使用或者空闲~~
- 占有和等待: 已占有某些资源的进程可以请求新的资源
- 非剥夺条件: ~~先前授予的资源无法强制地剥夺~~
- 循环等待条件: ~~至少有 2 个或以上进程构成循环链, 每个进程都在等待由循环链中下一个成员占有的资源~~

7.1.3.1 死锁建模

: 资源 R 被指派给进程 A

: 进程 A 请求/等待资源 R

7.2 鸵鸟算法

即视而不见

理由:

- 死锁极少发生
- 预防死锁的代价太高

UNIX 和 Windows 采用这方法, 这是方便性与正确性的平衡

7.3 死锁检测和恢复

- 允许死锁发生
- 尝试检测并恢复

7.3.1 单资源类型的死锁检测

每种资源只可分配一次, ~~每种类型只存在一个资源~~

构造资源图, 检测环路

7.3.2 多资源类型的死锁检测

每种资源只可分配多次, ~~每种类型存在多个资源~~

例 7.3.1 所有资源 $E=(4,2,3,1)$, 可用资源 $A=(2,1,0,0)$,
当前分配矩阵 $C=\begin{bmatrix}0&0&1&0\\2&0&0&1\\0&1&2&0\end{bmatrix}$, 请求矩阵 $R=\begin{bmatrix}2&0&0&1\\1&0&1&0\\2&1&0&0\end{bmatrix}$

解

1. 找到 P3 满足 R3=(2 1 0 0)<=A，标记 P3 并释放其资源：A=A+C3=(2 1 0 0)+(0 1 2 0)=(2 2 2 0)
2. 找到 P2 满足 R2=(1 0 1 0)<=A，标记 P2 并释放其资源：A=A+C2=(2 0 0 1)+(2 2 2 0)=(4 2 2 1)
3. 找到 P1 满足 R1=(2 0 0 1)<=A，标记 P1 并释放其资源：A=A+C1=(0 0 1 0)+(4 2 2 1)=(4 2 3 1)

结果: P1、P2 、P3 均被标记，不存在死锁

7.3.3 从死锁中恢复

- 剥夺法恢复, 很可能需要人工干预, 取决于资源类型
- 回退法恢复, 需要保存资源映像和资源状态
- 杀死进程法, 环路内环路外, 最好从头再运行没有副作用

7.4 死锁避免

安全状态 从安全状态出发，系统能够确保所有进程都能完成

7.4.1 银行家算法

优先选择距最大需求最近的客户

7.5 死锁预防

7.5.1 破坏互斥

某些设备 (例如打印机) 可以假脱机操作

- 只有打印机守护程序使用打印机资源
- 这样就可以消除打印机死锁

不是所有的设备都可以进行假脱机操作

原则:

- 不到万不得已不要指派资源
- 使得尽可能少的进程实际上要求资源

7.5.2 破坏占有和等待

要求进程在开始执行前请求所需资源, 这样进程在其需要时无需等待

问题

- 也许在开始运行时无法确知是否需要资源

- 也许需要的资源其他进程正在使用

变更

- 进程必须放弃所有资源
- 然后在需要时申请全部所需资源

7.5.3 破坏不可剥夺

剥夺, 通常不是有效的

7.5.4 破坏循环等待

- 保证每一个进程在任何时刻只能占用一个资源
- 提供所有资源一个全局编号, 所有申请必须按照编号的顺序