操作系统部分内容总结

by 吴纵宇

第2章 操作系统的结构

- 机制与策略
  - 机制：需要提供什么功能
  - 策略：怎么实现这些功能

2.1 操作系统接口

- 至少提供两种类型的接口
  - 命令接口
    - 命令行界面Command-Line Interface，CLI
    - 图形用户界面Graphic-User Interface，GUI
  - 系统调用
- 命令行界面是通过shell界面来实现的
- shell是命令解释器，连接了用户与Linux内核
- shell并不是内核的一部分，只是一个应用程序

2.2 系统调用

- 操作系统的主要功能是管理硬件资源以及为开发人员提供良好的环境
- 内核提供一系列具备预定功能的多内核函数，通过一组称为**系统调用**的接口呈现给用户
- 多任务处理功能通常靠进程来实现
- **基址寄存器**和**界限寄存器**，可以防止越界
- **用户模式**与**内核模式**
- API为程序员提供了一系列函数库，而这些函数通常为程序员调用实际的系统调用
- API分类
  - Windows API
  - POSIX API
  - Java API
- 每个系统调用都有一个相关数字，而系统调用接口会根据这些数字来建立一个索引列表
- 系统调用向系统传递参数的三种方式
  - 直接通过寄存器
  - 参数存储在内存中

    - 参数压入堆栈并通过操作系统弹出

2.3 操作系统结构

- **整体式结构**
    - 首先确定总体功能，然后将功能分为若干个子功能，实现每个子功能的程序称为**模块**
    - 优点：效率高、接口简单
    - 缺点：没有可读性，可维护性差
- **层次式结构**
    - 各层之间的模块只有单向调用关系
    - 优点
        - 功能由无序性变为有序
        - 复杂依赖关系变为单向依赖
- **微内核**结构
    - 最基本的功能保存在内核，不需要在内核态执行的迁移到用户态。这些程序依赖于微内核来进行通信
    - 三种功能
        - 低级存储管理
            - 负责把每个虚拟页映射到一个物理页框
        - 进程间通信（Inter-Process Communication，IPC）
        - I/O和中断管理
    - 最大的问题是性能问题
- **模块化结构**
    - 面向对象、动态可加载模块
    - Linux就是模块化结构操作系统

2.4 虚拟机

- 软件模拟的、具有完整硬件系统功能的、运行在一个完全隔离环境中的完整计算机系统
- Hypervisor，也叫virtual machine monitor，VMM虚拟器监视器，是虚拟化技术的核心。是一种运行在物理服务器和操作系统之间的软件层
- Hypervisor分为两种
    - 裸机型
    - 宿主机型

第3章 进程与线程

3.1 进程基础

- **顺序执行**
  - 顺序性
  - 封闭性
  - 可再现性
- **并发执行**
  - 并发执行时的间断性
  - 没有封闭性，从而失去了不可再现性
- 为了使并发执行的程序具有可再现性，必须使并发执行的程序能够被管理、制约，并控制其执行速度。因此有了**进程**这个概念
- 进程的四个特点
  - 动态性
  - 并发性
  - 独立性
  - 异步性
- 进程是系统分配资源和调度的基本单位、
- 进程控制块PCB
- 进程三个基本状态
  - 运行态
  - 阻塞态
  - 就绪态
- 还有两种另外的状态
  - 初始态
  - 终止态
- 就绪和阻塞态的进程都是在内存中的
- 在具有虚拟存储管理的系统中，引入**挂起**这个概念

3.2 进程控制

- 程序状态字中有异味表示处理器的执行模式，通过这一位的改变进行执行模式的设置
- 原语
- 进程切换
- 引起进程切换的事件有
  - 中断
    - 时钟中断和I/O中断
  - 陷阱

    - 系统调用
- **进程创建**
    - 进程借助原语实现创建一个新进程
    - 进程树
    - 进程标识符pid，通常是一个整数值
    - 进程init是Linux系统中作为所有用户进程的根进程或父进程，其pid为1
    - Linux的**fork()**函数，用于从原进程中创建一个子进程。子进程中返回0；父进程中返回子进程id；出错返回1
    - 穿件新进程成功后，系统中出现两个基本完全相同的进程，这两个进程执行没有固定的先后顺序，哪个进程先执行要看系统的进程调度策略
- **进程的阻塞与唤醒**
    - 请求系统服务
    - 启动某种操作
    - 等待新的数据

3.3 线程

- 线程又被称为轻型进程
- 线程是一个可以独立执行和调度的基本单位，但不再是拥有资源的独立单位、
- 每一个线程都有个独立的栈，还有独立的包含寄存器值、优先级和状态信息的**线程控制块**
- 线程与进程之间的区别和联系
    - 线程是调度和分配的基本单位，而进程是资源分配的基本单位
    - 具有更好的并发性
    - 线程创建、终止和切换时的系统开销比进程小
    - 线程之间通信无需调用内核从而提高了通信效率
- 线程分类
    - **内核级线程**
    - **用户级线程**
- 常见的三种线程库
    1. POSIX Thread
    2. Windows API
    3. Java

3.5 与进程或线程相关的其他技术

3.5.1 写时复制

- fork()与exec()函数
- 当使用写时复制技术时，仅复制任何已进场修改的页面，所有未修改的页面可以由父进程和子进程共享
- 只有可以修改的页面才需要标记为写时复制

3.5.2 线程池

- 服务器都具有一个共同点：单位时间内处理数目巨大的连接请求
- 线程池的出现着眼于减少管理线程的开销而产生的技术

3.5.3 进程间的远程通信

- IPC是指在不同进程之间传播或交换信息
  - 同一主机的进程间通信
  - 不同主机上的进程间通信（进程间远程通信）
    - 进程之间的远程通信是基于客户机/服务器的
    - 套接字Socket
    - 远程过程调用Remote Prcedure Call，RPC
      - 许多方面都类似IPC机制，并且通常建立在IPC之上
    - 远程方法调用Remote Method Invocation，RMI

第4章 进程同步

- 由于进程的执行具有异步性，如果考虑不周，进程之间相互影响，有可能造成无法估计的灾难性后果

4.1 进程的互斥

- 临界资源和程序的临界区
- 不论是硬件资源还是软件资源，多个进程必须互斥对它们进行访问
- ```
  do {
      进入区
          临界区
      退出区
          剩余区
  } while(TRUE)
  ```
- 系统需要满足的条件
  ```

    - 互斥
    - 有限等待
    - 空闲让进
- 互斥访问有两种方法
    - **硬件实现互斥**
        - **中断禁用**
            - ```
              while (true) {
                // 禁用中断
                临界区

                // 启用中断
              }
              ```
        - **专用机器指令**
            - 优点
                - 使用范围广
                - 使用简单
                - 可支持多个临界区
            - 缺点
                - 导致CPU空耗
                - 导致饥饿进程
    - **信号量实现互斥**
        - 基本原理：两个或多个进程可以通过简单的信号进行合作，一个进程可以被迫在某一个位置停止，直到它接收到一个特定的信号
        - 信号量包括一个整型值和一个等待队列queue，并且只能通过两个原语wait()、signal()操作来访问
              ```

```
struct semaphore {
    int value;
    struct QUETYPE *queue;
}

void wait(semaphore s)
{
    s.value = s.value-1;
    if (s.value<0)
        block(s.queue)
}
void signal(semaphore s)
{
    s.value = s.value + 1;
    if (s.value<=0)
        wakeupWai(s.queue);
}

do {
    wait(mutex);
        临界区
    signal(mutex)
}while(true)
```

4.2 进程的同步

- 一个进程受到另一个进程执行的直接制约，我们称之为进程间存在着同步关系
- 把系统中使用某一类资源的进程称为该资源的消费者，而把释放同类资源的进程称为该资源的生产者
- 一些应用
  - 独木桥问题
  - 取物问题
  - 进程的前驱后继问题

4.3 进程之间的通信

- 进程间控制信息的交换被称为**低级通信**
- 进程间大批量数据的交换称为**高级通信**
- 高级通信机制大概可以分为以下几类
  - **共享内存方式**

- **管道通信**
  - 管道是一个用于连接读进程和一个写进程，实现进程间通信的一种共享文件，称为pipe文件
  - 必须要有三个方面的能力
    1. 互斥
    2. 同步
    3. 确认对方是否存在
- **消息传递通信**
  - 直接通信方式与间接通信方式
  - send原语和receive原语
  - ```
    send(mailbox,letter)     // 把信件letter送到指定的信箱mailbox
    receive(mailbox,address) // 从指定信箱mailbox中取出一封信，存放到指定的地址address
    ```

4.4 管程

- 进程等待队列与紧急等待队列
- 紧急等待队列的优先级应当高于入口等待队列的优先级

第5章 死锁

- 一组进程相互竞争系统资源而形成的“永久”阻塞现象

5.1 死锁的原理

- 多类资源，每类可以有多个资源
- 等价的资源组合在一起组合成一个“资源类”
- 死锁与环路关系
  - 资源分配图无环路，则一定无死锁
  - 资源分配图有环路，且每个资源类中只有一个资源，则环路的存在就意味着死锁的形成
  - 资源分配图有环路，但设计的资源类中有多个资源，则不一定构成死锁
- 死锁条件
  - 互斥
  - 占有且等待
    ```

    - 不可剥夺
    - 环路

5.2 死锁的处理方法

- 三种方法
    - **预防或者避免**
    - **预防**
        1. 破坏互斥（基本不考虑）
        2. 禁止占有且等待
            - 开始运行之前一次性获取所有资源
            - 只获得运行初期需要的资源，在运行过程中逐步释放分配到的且已经使用完成的资源
        3. 废除不可抢占
        4. 破坏循环等待条件
    - **避免**
        - 资源分配拒绝策略（银行家算法）
        - 安全状态和不安全状态
        - Resource，Available，Max，Allocation
    - **不采取措施**
    - **鸵鸟策略**
- 死锁的检测
    - 单体资源类
    - 多体资源类
        - 三个典型检测时机
            - 进程申请资源但却无法得到时
            - 定时监测
            - 系统资源利用率下降

5.3 死锁的解除

- 撤销所有死锁进程
- 恢复到前面定义的检查点
- 有选择的撤销死锁进程
- 剥夺资源

5.4 哲学家进餐问题

第6章 处理器调度问题

- 处理器调度是多道程序操作系统的基础

6.1 处理器调度算法的目标

- 衡量调度算法的指标包括
  - CPU利用率
  - 吞吐量
  - 周转时间
  - 等待时间
  - 响应时间
- 总体为两种策略
  - 非抢占式调度策略
  - 抢占式调度策略

6.2 分级调度

- 一个进程从创建到执行需要经历三级调度
  - 长程调度
  - 中程调度
  - 短程调度

6.2.1 长程调度

- 又称为**作业调度**、**接纳调度**
- 主要功能：根据作业控制块中的信息审查系统能否满足用户作业的资源需求，以及按照一定的算法从外存的后备队列中选取某些作业调度入内存，并且为它们创造进程、分配必要的资源，然后再把新创建的进程插入队列
- 典型的有先来先服务FCFS、最短作业优先SJF、最高响应比优先HRRN等

6.2.2 中程调度

- 提高内存的利用率和系统的吞吐量
- 对调功能

6.2.3 短程调度

略

6.3 常用的调度算法

6.3.1 先来先服务调度算法FCFS

- 最简单的非抢占的调度算法
- 甘特图Gantt
- “护航效果”

6.3.2 优先级调度算法

- 静态优先级和动态优先级
- 动态优先级：与进程占有CPU时间成反比，与在就绪队列中等待时间成正比

6.3.3 最短作业优先调度算法SJF

- Shortest-Job First
- 常用于长程调度（作业调度）
- 基于抢占式的SJF算法也被称为最短剩余时间优先调度算法

6.3.4 最高响应比优先调度算法

- HRRN，High Response Ratio First
- $R = \frac{W+S}{S}$

6.3.5 轮转调度算法

- Round Robin,RR
- 常用于分时系统中

6.3.6 多级反馈轮转调度算法

略

6.3.7 实时系统的调度算法

略

6.4 多处理器调度

- 非对称多处理AMP

- 对称多处理SMP

第7章 内存管理

7.1 内存相关基本概念

- 以字节为存储单位，内存地址空间是指对内存编码的范围。编译后会形成逻辑地址或者说虚地址

7.1.3 地址重定位

- **静态地址重定位**
  - 程序执行前，由装配程序完成的地址映射工作
  - 其只完成了一个首地址不同的连续地址变换
  - 不需要硬件支持
- **动态地址重定位*

7.1.4 程序链接

- 静态链接
- 装入时动态链接
- 运行时动态链接