操作系统原理笔记

陈鸿峥

2019.05*

目录

本课程使用的教材为William Stallings《操作系统—精髓与设计原理（第八版）》。其他参考资料包括Stanford CS140、CMU15-460、*Operating System Concepts (10th ed.)*。

关于计算机系统的内容在此不再赘述，详情参见**计算机组成原理**的笔记。

1 操作系统概述

1.1 概述

操作系统核心即怎么虚拟多几个冯诺依曼计算机出来给程序用。操作系统是控制应用程序执行的程序，是应用程序和计算机硬件间的接口（屏蔽硬件细节）。

这里先解释几个概念

- 并发：两件事情**可以**同时(simultaneously)发生，没有时间限制，$t_1 > t_2$，$t_1 < t_2$，$t_1 = t_2$都可
- 同步：两个事件有确定的时间限制
- 异步：两件事不知道何时发生

*Build 20190528

1.2 发展历史

- 串行处理(1940s)：没有OS，人工调度，准备时间长
- 简单批处理系统(1950s)：
  - 使用监控程序(monitor)，读入用户程序执行
  - 提供内存保护、计时器、特权指令、中断
  - 两种操作模式：用户态、内核态(mode)
  - 单道程序(uniprogramming)批处理：处理器必须等到IO指令结束后才能继续
- 多道程序批处理(1950s末)：多个作业同时进入主存，切换运行，**充分利用处理器**
- 分时系统(1961)：MIT CTSS(Compatible Time-Sharing System)，满足用户与计算机交互的需要，**减小响应时间**；多个交互作业，多个用户，把运行时间分成很短的时间片轮流分配
- 实时系统：专用，工业、金融、军事

现代的操作系统通常同时具有分时、实时核多道批处理的功能，因此被称为通用操作系统。而OS也不仅是在PC机上有，网络OS、分布式OS、嵌入式OS层出不穷。

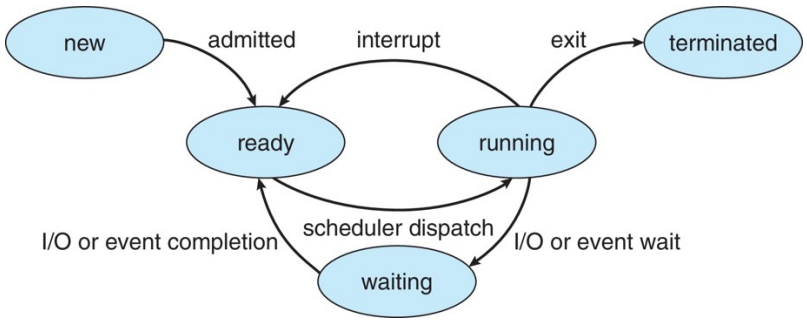
2 进程

进程(process)是**运行时(running/in execution)**程序的实例。

- 程序并发执行的特征（多道程序）：间断性、无封闭性、不可再现性（破坏冯顺序执行特性）
- 进程的特点：动态性、并发性、独立性、异步性
- 进程的作用
  - 提升CPU利用率：将多个进程重叠（一个进程IO时另一个计算）
  - 降低延迟(latency)：并发执行，不断切换，防止卡住

进程控制块(Process Control Block, PCB)，在Unix是proc，在Linux是task_struct

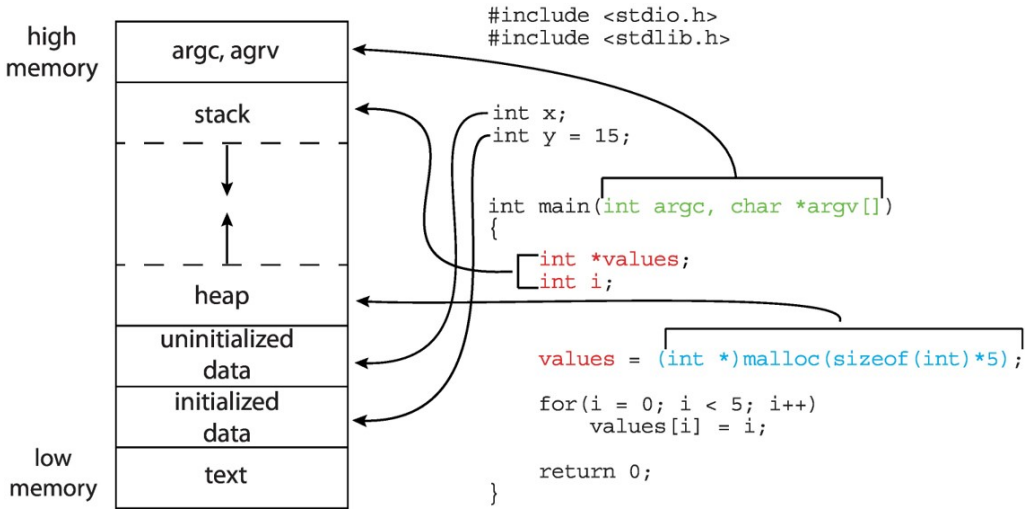
- 进程标识符/ID
- 状态：运行、就绪、等待/阻塞、新建、退出



- 优先级

- 程序计数器(PC)
- 内存指针：报错指向程序代码、相关数据和共享内存的指针
- 上下文数据(context)：进程被中断时寄存器中的数据
- IO状态信息
- 记账信息(accounting)：占用处理器时间、时钟数总和、时间限制等

内存组织：代码段、数据段、堆段、栈段（从小地址往大地址）



可以参考原始的UNIX论文[1]。

- 创建进程：fork、waitpid
- 删除进程：exit、kill
- 执行进程：execve

进程切换：扔掉堆、数据段，读入新代码段、数据段，重设栈段、堆段，初始化寄存器，开始执行

处理器上下文即处理器寄存器的内容　就绪 

运行　用户态核心态

shell：用户控制台，解释用户指令

OS自身代码

- 传统方式：响应中断，获得CPU控制权
- 进程方式

[1] http://www.scs.stanford.edu/19wi-cs140/sched/readings/unix.pdf

3 线程

4 并发

4.1 基本概念

- 原子(atomic)操作：不可分割
- 临界区(critical section)：不允许多个进程同时进入的一段访问**共享资源**的代码
- 死锁(deadlock)：两个及以上进程，因每个进程都在等待其他进程做完某事（如释放资源），而不能继续执行
- 活锁(livelock)：两个及以上进程，为响应其他进程中的变化，而不断改变自己的状态，但是没有做任何有用的工作
- 互斥(mutual exclusion)：当一个进程在**临界区访问共享资源**时，不允许其他进程进入访问
- 竞争条件(race condition, RC)：多个进程/线程读写共享数据，其结果依赖于它们执行的相对速度
- 饥饿(starvation)：可运行的进程长期未被调度执行

核心内容

$$\text{并发} \rightarrow \text{共享} \rightarrow \text{RC问题} \rightarrow \text{互斥}$$

共享数据的最终结果取决于进程执行的相对速度
同步：有明确的时间先后限制，两个或多个进程之间的操作存在时间上的约束（也包含互斥）

4.2 互斥

软件方法

- Dekker算法
- Peterson算法

硬件方法

- 关中断：限制处理器交替执行各进程的能力，不能用于多核
- TestSet(TS)指令：比较并交换，原子指令，一个指令周期内完成，不会被中断
- Exchange指令(x86`xchg`指令)：同上，适用于单核多核，多变量多临界区，但需要忙等待(busy waiting)/自旋等待(spin waiting)，可能饥饿或死锁

4.2.1 信号量(semaphore)

- 整数：可用资源数(≥ 0)，需要初始化
- P操作(`semWait`)：信号量的值**减1**（申请一个单位的资源），若信号量变为**负数**，则执行P操作进程阻塞，**让权等待**

- V操作(semSignal)：信号量的值**加1**（释放一个单位的资源），若信号量**不是正数**（绝对值=现被阻塞的进程数/等待队列的长度），则使一个因P操作被阻塞的进程解除阻塞（唤醒）

需要保证P操作和V操作的原子性

```
struct semaphore {
    int count;
    PCB *next;
} s;
void Init(s) {
    s.count = nr;
    s.next = NULL;
}
void P(semaphore s) {
    s.count--;
    if (s.count < 0)
        Block(CurruntProcess, s);
}
void V(semaphore s) {
    s.count++;
    if (s.count <= 0)
        WakeUp(s);
}
```

注意P操作是**小于0**，V操作**小于等于0**

信号量的优点是简单且表达能力强，用P、V操作可解决多种类型的同步/互斥问题，但不够安全，P、V操作使用不当会产生死锁。

二元信号量省空间，不能代表资源数量，要引入全局变量代表数量

信号量实现互斥

- 对于每一个RC问题，设一个信号量（向系统调用向内核调用），**初始化为**1
- 所有相关进程在进入临界区**之前**对该信号量进行**P操作**
- 出临界区**之后**进行**V操作**

信号量实现进程同步

- 对每一个同步关系都要设一个信号量，初值看具体问题
- P后续动作的前面
- V前驱动作的前面

组合动作：生产者-消费者问题

管程(monitor)：通过集中管理（封装同步机制与同步策略）以保证安全

消息传递

- send
- receive

读者-写者问题