PCB: process control blocks 进程控制块

TCB: thread control blocks 线程， TCB中存储每个线程的状态信息， TCB 和PCB最大的不同在于1-上下文切换时 TCB 地址空间不变（不需要换page table) 2.栈：一个线程一个栈，局部变量、返回值参数是栈上的thread-local

为什么使用多线程

并行可以加速，减少慢速 I/O 影响

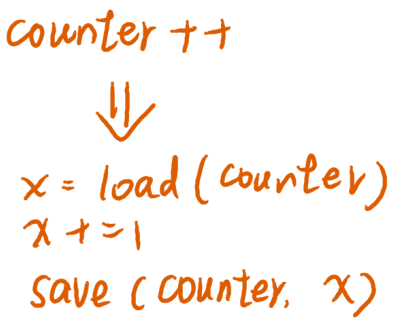
多线程的创建

p thread_join:等一个线程运行结束，教材26章第1个例子

一个线程创建之后可能立即被执行，也可能等一会儿被执行。执行顺序由OS scheduler 决定

共享数据： sharing data

例子266
问题核心：不受控制的多线程调度



比如，从一块内存读到eax 寄存器中，然后加1，这时候中断了，重新调度，执行另一个线程，counter 是3，回来，counter还是写3，BUG!

比如，从一块内存读到eax 寄存器中，然后加1，这时候中断了，重新
调度、执行另一个线程， counter 是3，回来， counter还是写3， BUG!

名词解释：

临界区：一部分代码可以访问共享数据，且不可以被多个线程并发执行

互斥：一个线程在临界区中执行，其他线程就不可以进入临界区，只有一个线程在临界区内

原子化：不可以在一条指令执行中被打断

synchronization primi tives: 同步原语帮助的使多线程以互斥同步的效果访问共享资源

数据竞争：多个线程几乎同时进入临界区，都尝试修改共享资源，导致意外结果

thread API

Lock

1. lock 是一个变量，
使用前先声明.初始化， lock （）的语义：尝试拿锁，没有其他线程持有锁，拿到，进临界区；否则不返回等待。unlock的语义，如果有其他线程等着拿锁，其中一个拿到，否则锁空闲

锁可以保证最多一个线程进临界区

2.评价锁
① 正确性：互斥
② 公平性：有没有
饥饿或饿死
③ 性能：

3、管理中断
早期仅通过开关中断，弊大于利。详见2 8-5节

4.自旋锁
① 实现 28.7节，同时
抢占式调度
② 评价：不能保证不会饥饿，所
以不公平

抢占式调度
② 评价：不能保证不会饥饿，所
以不公平
③ 性能 case 1单处理
器时，等 n-1个 cpu cycles.相当浪费。case z 多处理器，性能不错

5. Fetch and add
可用来实现 ticket looks,具体28.11。不会出现饥饿

6. just yield, baby

使用 yield 去让 OS重新调度一个线程执行， yield 是一个特别的系统调用，把running的线程变成ready,然后选择一个来执行。本质上是一次再调度

Lock based data structures

1.并发共享数据，比如全局变量：最简单的方法在临界区上下加锁

2. approximate counter:分成
local counter 和global counter

3. hand over locking
当遍历链表的，先抓住下一个节点的锁，再释放这个节点的锁，但实际上并不快，没啥实用价值

4. Michael and
Scott concurrent Queue 重点头尾各一个锁，提高并发

5-Premature optimiza tion 抛开workload谈优化， 等于耍流氓

Condition Variable条件变量

检查在执行中某个条件是否满足，比如子进程是否完成

概念解释

① 条件变量： 一个队列，当条件不满足的时候，把自己放进队列（在这个条件上等待）。当其他线程状态改变的，可以叫醒等待线程中的一个

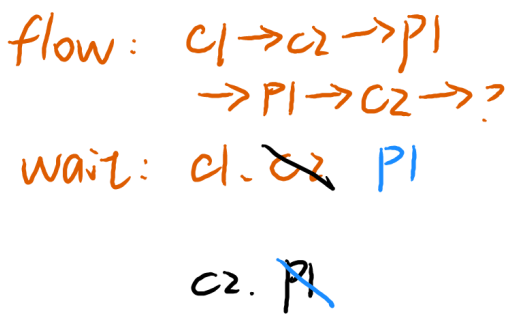
1.定义：两种操作wait 和 signal，其中wait 要和锁搭配使用，wait 释放锁并把当前线程加入等待队列

总是发的，可以叫醒等待线程中的一个

1.定义：两种操作wait 和 signal。其中wait 要和锁搭配使用， wait 释放锁并把当前线程加入等待队列(sleep 注意，这两步是一体的、原子的！ signal 叫醒一个线程，在 wait 返回时必须要reacquire lock

2.回到开头提出的等子线程问题，见3 0.3节。 **Hold the look when calling signal or wait**

3.生产者消费者问题：注意等待的条件、锁while 而不是 if。如果只有一个条件变量，



Semaphores

1、概念：

关于同步的原语，可以索要或归还一种资源

2. P 操作与 V
操作：P 可以看作要种资源， V 可以看作归还资源

3.读者写者问题

4、哲学家吃饭问题

Virtualization

1. process 进程：运行的

1. process 进程：运行的
程序

2. time sharing:
运行一个进程一段时间，然后选择另一个进程运行 ,营造出多个进程在 CPU上同时运行的假象。这也叫**虚拟化CPU**

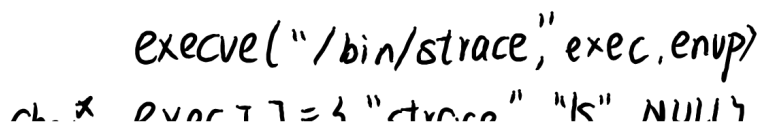
3.上下文切换：停止一个进程，把寄存器状态保存至 TCB中，然后调度另个进程，恢复寄存器现场，继续执行该进程。

4. machine state机器状态：
① 内存，地址空间
② 寄存器状态：
比如 PC,基址寄存器
③ 关于 I/O 的信息，
比如文件描述符

5.进程 API
① 创建
• 把代码和数据加
载，以某种可执行形式如 ELF。早期在这行之间全加载，现在
用虚存和分页。
˙分配栈和初始化栈， argc 和 argv。分配堆区。
˙文件描述符初始化， I/O 初始化。.从entry point 开始执行

6.进程状态：
① running: 执行中
② ready:拿到CPU就执行
③ Blocked:等资源或条件

7- API
① fork:用来创建新
进程， pìd 标记， fork 返回0是子进程，大于0是父进程
② wait:等另一个进
程完成，把自己放到等待队列
③ exec ve:
偷梁换柱



execve("/bin/strace", exec, envp)

char* exec[] = {"strace", "ls", NULL}

envp = environ 指针

Limited Direct Execution

1. performance & control
既要使 OS 保持对资源的控制，又要有良好性能

2-用户态与内核态：
① 用户态不能使用
全部硬件资源
② 内核态有全部
硬件的权限
两者之间可以使用自陷指令转换。中断向量表： trap table 开机的特权指令初始化中断向量表，使得中断到来时跳转到对应的例程

3.概念、第6章 13页

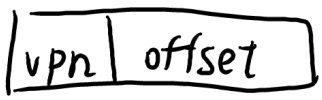
虚存和分页

1-分页机制的优点： flexibility,可以支持地址空间的抽象，不需要考虑栈和堆的方向simplicity:有利于简化空闲区域的管理

2页表：用于地址转换，页表是每个进程一个的数据结构，反置页表除外

VP	PP
0	1
1	3
2	11

2. address_translation
Vpn (virtual Page number) & offset



3. where is page table:

4.VA 到 PA 的转化：

调度算法

1.轮转时间：结束减到达

2. FCFS 导致
convoy effect 耗时短的在耗时长的后面等待

3-S J F: 短任务优先最优的！！！ 必须是以轮转时间为指标

4. 最短剩余时间优先

5-时间片轮转调度： RR。时间片越短响应时间越好，但上下文切换太频繁带来损失

5. MLPQ