CS334 操作系统期中复习

本次整理的期中复习将会包括虚拟化（virtualization）以及并发（concurrency）的课件所有重点内容，希望可以取得一个比较好的成绩~由于是在夏季小学期上的课，如果对知识点的理解不是很深刻，还烦请在 issue 里面指出~

Lecture 1 Introduction to Operating Systems

冯诺依曼架构：

- 单一共享内存，同时存储程序以及数据
- 单一内存访问总线
- 一个计算单元
- 一个程序计数器（PC）

计算机系统分为四个部分：

- 硬件（提供计算资源，包括 CPU、内存、IO设备）
- 操作系统（控制管理硬件）
- 应用程序
- 用户

操作系统包括一个成为内核（kernel）的软件程序，也包括了一些 helper programs，如 shell，GUI，浏览器等等

OS 也被称为资源管理器（管理 CPU，内存，磁盘以及IO 设备），也称为控制程序（控制程序的执行）

操作系统执行三部分内容：

- 虚拟化（virtualization），虚拟化 CPU，虚拟化内存。
- 并发（concurrency)，跑多线程程序
- 持久化（persistence)，持久地保存易失设备中的数据

进程（process）：进程是一个正在执行的程序（program in execution）

进程的执行需要资源，例如 CPU，内存，IO以及文件

单线程的进程只有一个程序计数器，多线程的进程有多个程序计数器，每个线程一个。

进程管理（process management）：创建销毁用户以及系统的进程，提供进程的同步机制，提供进程通信机制

内存（memory）：DRAM 是用户使用的主存，CPU 在运行的时候只会直接与主存交互，所以数据以及程序都要放在主存当中。

内存管理（memory management)：确定内存所存放的内容，确实什么在内存当中以及谁在使用内存。分配以及回收内存。

存储管理（storage management)：将物理属性抽象成逻辑的存储单元——文件

文件以目录的形式管理

缓冲（buffering）：在运输数据的时候，暂时存储，用于成块运输

缓存（caching）：为了改善访问的性能，存储数据的一部分在更快的存储结构当中。

保护（protection）：控制进程或者用户的访问权限

安全（security）：保护系统免受内部或者外部的攻击（洪泛攻击，拒绝访问攻击等等）

Lecture 2 OS Basics

内核态以及用户态

用于 OS 的保护机制

一些特权指令（对硬件的操作）只能在内核态(kernel mode)执行

硬件支持：

- CPU 硬件提供 模式位（mode bit）
- 一些只能在内核态执行的操作
- 用户态到内核态的转换，同时保存用户态的PC
- 内核态向用户态的转换，并且恢复用户态的 PC

模态转换（Mode Transition）的三种类型

系统调用（system call）：需要系统提供的服务（例如 exit），在进程之外执行，没有地址。通过数字调用。类型如下：

- 进程控制（`fork()`,`exit()`,`wait()`）
- 文件管理（`open()`）
- 设备管理（`read()`,`write()`）
- 信息保持（`sleep()`）

- 通信（`pipe()`）
- 保护（`chmod()`）

中断（interrupt）：外部异步事件触发上下文切换（context switch），与用户进程独立。类型有抢占式调度等

陷阱或异常（trap of exception）：内部同步事件触发的上下文切换，如除零异常，非法地址访问等

内核结构（kernel structure）

宏内核（Monolithic Kernel）

将所有的特权操作都放在内核当中，例如访问 io 设备，访问内存以及硬件中断

宏内核非常大，包括了例如内存子系统，IO 子系统等等的许多组件，是 linux, Unix 的基础

微内核（Micro Kernel）

将更多的传统功能交付给用户进程，有更好的灵活性，安全性以及错误的容忍度

与保护（protection）相关的机制依然在内核当中，但是与资源管理(resource management)相关的策略交给了用户层面

优势:

- 由于内核代码比较少，所以更加稳定以及安全可靠
- 更好地支持并发以及分布式系统

劣势:

- 由于需要更多的进程间通信，导致更多上下文切换，使得更加慢

杂交内核（Hybrid Kernel）

宏内核和微内核的结合

外核（exokernel）

内核只负责安全，更加小

Hypervisor

虚拟机管理器，VMM

重点在于虚拟化以及隔离机制

虚拟机之间的资源分隔器

操作系统提供的服务

- 用户界面（CLI, GUI）
- 程序执行（系统可以将程序装载进入内存然后运行）
- IO 操作
- 文件系统管理（创建，搜索，删除文件）
- 通信（进程之间的通信服务，通过共享内存或者信息传递）
- 差错检测
- 资源分配
- 记录（使用的用户，以及资源的使用情况）
- 保护以及安全

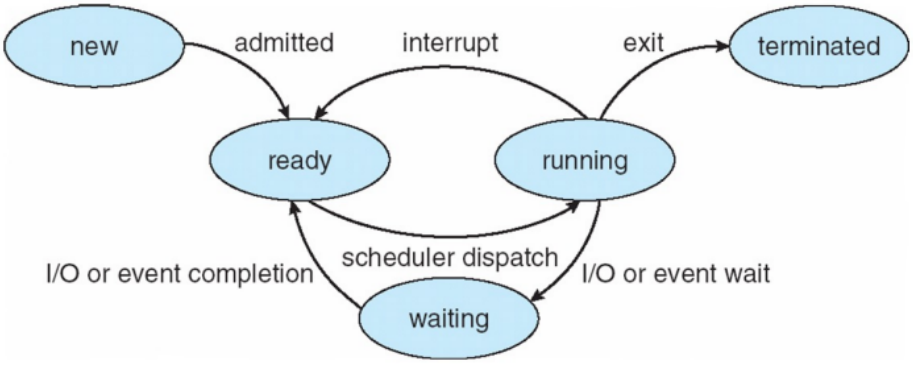
Lecture 3 Processes

进程是正在运行的程序。操作系统从磁盘中取出程序并且执行，操作系统会为该程序创建堆栈

通过不同的进程标识符（process ID）来标识不同的进程

进程的生命进程

处于 `ready` 状态的进程，在经过调度之后，将会进入 `running` 运行状态，在运行状态的程序会因为系统的中断（如时钟中断（interrupt））而回到 `ready` 准备状态；也会因为执行 `IO` 操作或者事件的等待而进入 `waiting` 等待状态，处于 `waiting` 等待状态的进程，完成了 `IO` 或者等待的事件完成了之后，便会回到 `ready` 准备状态。



System call 系统调用

与函数调用不同，系统调用通过数字被调用。系统调用是内核提供的一种内核操作的抽象。

通常，系统调用需要比一个索引更多的信息，有几种传参方法：

- 寄存器传参

- 块传参：通过传递一个存储有参数的块的块地址进入寄存器，通过寄存器将该块地址给 OS
- 栈传参

其中块传参以及栈传参都对参数的数量以及长度没有限制

进程的创建

父进程通过调用 `fork()` 创建子进程，子进程从 `fork()` 返回的位置开始执行，而不是从程序的起始处开始执行。

对于父进程来说，`fork()` 的返回值是子进程的进程号，而子进程的 `fork()` 返回值是 0

`fork()` 复制了所有父进程用户区域的数据（PC，程序代码，内存区域，开启的文件），但是没有复制如文件锁这类数据

地址区域全复制（局部变量，全局变量，动态分配的内存（堆），代码部分）

`exec()`，会直接将新的程序载入内存并且执行。会随着新程序的结束而直接结束，并不会回到老程序。

会将复制过来的地址空间做一些改变，例如清空局部变量，代码部分使用新的代码部分，动态分配的堆区域清空，全局变量重置。

`wait()`，将当前的程序挂起，等待任意一个子进程运行完毕之后，回到程序当中。当父进程没有子进程或者是子进程在父进程等待之前已经运行结束，则`wait()`会立即返回。

`exit()`，在进程调用 `exit()` 程序退出之后，会释放所有用户地址空间的内容，包括分配的内存以及代码区域。但是并没有释放进程表当中该进程的进程号，此时进程成为僵尸进程。内核此时会通知子进程的父进程这个消息，发送 `SIGCHLD`信号。

`exit()`系统调用将一个进程变成**僵尸进程**当：

- 一个进程调用了 `exit()`
- 一个进程从 `main`返回
- 一个进程反常地退出了，内核会主动调用 `exit()`

如果父进程没有在等待该子进程，父进程便会忽略这个 `SIGCHLD`信号。如果在等待，则内核会为该父进程注册一个信号处理 routine。这个 routine 接收并且移除该信号，之后从进程表以及任务列表当中移除该 已经结束的子进程的信息。

进程控制块（PCB）维护了进程的信息（进程状态、程序计数器、CPU 寄存器）

Thread

线程也是正在运行的程序的抽象

多线程程序有不止一个的执行点

每一个线程都有自己私有的执行状态（PC，私有的寄存器，私有的栈）

当 CPU 决定进行线程的切换的时候，也需要上下文切换。

在同一个进程里面的线程共享计算资源（地址空间，文件，信号等等）

使用线程的目的：

- 提高并行性
- 防止由于 IO 速度过慢导致的程序进程阻塞
- 允许资源的共享

Lecture 4 CPU Scheduling

CPU 调度会在线程发生下列四种情况发生

- 从运行状态切换到等待状态
- 从运行状态切换到准备状态
- 从等待状态切换到准备状态
- 终止

其中第二第三种都是抢占式（preemptive），第一种和第四种都是非抢占式（non-preemptive）

等待时间（waiting time）：进程在队列里等待的累计时间

轮转时间（turnaround time）：进程从进入到完成的累计时间

调度算法

- 最短工作优先（SJF），分为抢占式以及非抢占式
- 轮转算法（RR），优势在于及时响应
- 优先调度（Priority Scheduling），缺点在于有可能会进程饥饿，解决办法就是动态调整优先级（等待时间越久价值越高）
- 多队列优先调度

在有着相同 nice value 的情况下，IO bound 的进程比 CPU bound的进程有着更低的 vruntime, 即有着更高的优先级

Lecture 5 Synchronization

进程间通信

相同进程的线程彼此共享地址空间，多个线程共享全局变量，这样也使得线程之间的通信变得简单

竞态条件（race condition）

在多线程同时并发访问共享区域的时候会触发竞态条件

竞态条件意味着一个执行的运行结果取决于共享资源被访问的顺序

临界区域（critical section）

临界区域是一个会访问共享区域的代码片段

应该越紧越好，否则一个程序会有很大的可能去阻塞另一个程序

临界区域的实现：

- 首先要保证互斥（mutex exclusion）
- 有界限的等待（Bounded Waiting）
- 进步（progress）：当当前没有进程位于临界区域的时候，其他想要进入的进程应该最后可以进入临界区域

禁用中断（Disabling Interrupt）

在一个进程进入临界区域的时候，禁止其他进程进入临界区域

问题1：在单核的情况下，如果一个用户态的进程进入临界区域并无限循环，那么操作系统便无法重新获得执行力

问题2：在多核的情况下，不同核的进程完全可以同时访问临界区域

锁（lock）

自旋锁：通过检测共享变量的值来确定是自旋还是进入临界区域

Peterson 算法：通过设置`interested`区域来控制进入临界区域

信号量（semaphore）

有一个代表资源数量的整数以及一个等待列表

`sem_wait()`先给该整数减一，当该整数小于零的时候，进入睡眠状态，等待唤醒

`sem_post()`给整数加一，如果该整数小于等于零，就唤醒一个睡眠等待的进程

生产者消费者问题

实现注意：

- 需要使用两个信号量——避免消费者唤醒消费者或者生产者唤醒生产者
- 互斥量要紧挨着临界区域——避免死锁

哲学家就餐问题

通常会出现死锁问题，因为当所有哲学家都同时想要进餐的时候，当出现轮流拿起左手边的筷子时，大家都会在将要拿起右手边筷子的时候陷入等待，出现死锁

Lecture 6 Address Translation

多道编程（multiprogramming)

在一定时间内有多个进程准备运行，操作系统在他们之间切换调度

优势是能够时分共享计算机资源

地址空间（address space)

是进程对于内存的视角抽象

分为四个段：代码段，栈段，堆段

MMU

是CPU 中的内存管理单元（memory management unit)

将指令用到的虚拟地址转换成 DRAM 能够理解的物理地址

CPU 介入每次的内存访问

操作系统负责为正确的转换准备硬件

虚拟地址翻译

使用基址寄存器（Base）以及界限寄存器（Bound)

硬件支持：

- 更改 基址寄存器以及界限寄存器的特殊模式（privileged mode）
- 基址以及界限寄存器
- 注册异常处理的特殊指令

OS 支持

- 内存管理，分配回收内存
- 基址寄存器以及界限寄存器管理，在上下文切换的时候修改设置基址以及界限寄存器
- 异常处理，当异常发生时候需要执行的代码

基址以及界限寄存器会带来内部的碎片化，在堆和栈之间浪费内存

分段（segmentation）

为堆、栈、代码分别分配一堆基址以及界限寄存器