4 Threads & Concurrency

线程 (Thread)

- 当进程需要做的事情比较多的时候，一个进程管理事情就会非常复杂；因此将进程拆解成若干线程，再将每个线程交给一个核或处理器进行处理。这样可以很好的适应CPU的发展（多核、多处理器等），提高效率。
- 多线程的优点：
  - 响应时间更快 (Responsiveness)；
  - 资源共享 (Resource Sharing)；
  - 经济实用 (Economy)：线程需要的资源远远小于进程；
  - 可扩展 (Scalability)：更容易扩展到多核、多处理器CPU。
- 挑战：任务的划分；负载均衡；数据划分；数据依赖性；测试和调试。
- 分为用户线程 (User thread)、内核线程 (Kernel thread)。用户线程和内核线程的对应关系有：一对一、多对一、多对多。
  - 多对一 (many-to-one model)：许多用户线程共享一个内核线程；内核压力小但是内核线程完成时间可能会较长；
  - 一对一 (one-to-one model)：一个用户线程与一个内核线程对应；内核线程完成效率高，但是内核压力大，耗费更多资源；（Windows, Linux）；
  - 多对多 (many-to-many model)：多个用户线程与多个内核线程对应；映射复杂度增加；
  - 两层模型 (two-level model)：一部分选用一对一模型，一部分选用多对多模型。
- **PThreads** 进行实现。
- 超线程 (hyperthreading)：将线程的计算单元进行进一步的拆分。

并行方式

- *数据并行 (Data parallelism)*：将大数据拆分成若干小数据，每个小数据执行相同的任务；
- *任务并行 (Task parallelism)*：将同一任务拆分成不同小任务。

Amdahl's Law：见 CS359 笔记。

线程池 (Thread Pool)：将许多线程放在线程池中进行工作。

Fork-Join 并行 (Fork-Join Parallelism)：进程首先进行拆分 (fork) 成多个线程（执行任务），然后执行后合并 (join)。注意的是线程创建的需要的时间量级比进程低。

进程和线程的比较

- 进程是重量级 (heavy-weighted) 的，而线程是轻量级 (light-weighted) 的。进程需要所有的 PCB 信息包括堆、栈等等，而线程的许多数据是共享的；
- 进程的通讯方式 (IPC) 比较复杂，可以传递很多参数、数据；而进程的通讯方式 (signal，信号) 非常简单，但是只能传输简单信息 (signal)。
- 进程通过 `fork()` 创建，线程通过 `pthread_create()` 创建；
- 进程通过 `exec()` 执行，线程通过 `pthread_join()` 执行。
- 进程通过 `exit()` 退出，线程通过 `exit()` 或 `pthread_exit()` 退出；
- 一个轻量级进程 **(Light-Weighted Process, LWP)** 和线程基本相同；
- 进程的调度方式 (scheduling) 和线程基本一样。

线程通信 (signal) 分为同步通信和异步通信。

- 信号传输的分类
  - 点对点通信，将信号传输给固定的线程；

    - 将信号传输给这个进程中的所有线程；
    - 将信号传输给这个进程中的特定线程；
    - 可以定义一个**线程**来专门处理所有信号。
- 线程终止
    - 异步终止；
    - 定时终止：定时扫描并清除无用线程。

Thread-Local Storage (TLS) 可以给每个线程提供数据的一份拷贝，防止因为不同线程并发执行导致的公用数据修改问题。

Windows 线程分类

- ETHREAD (Executive Thread Block)：内核线程；“总管线程”，包含所有信息；
- KTHREAD (Kernel Thread Block)：内核线程；执行任务和调度；
- TEB (Thread Environment Block)：用户线程。

Linux线程：`tasks`，线程创建通过 `clone()` 系统调用，可以允许子进程共享父进程的一些数据（通过调用的不同参数设定）