COMP3301 - Threads 重点速查表

模块	关键概念	一句话记忆
为何要用线程	进程太重、线程轻巧，提升交互性与效率	“轻量小兵，敏捷多工”
线程带来的四大好处	响应快、共享易、开销省、能扩核	“快、易、省、扩”
多核编程挑战	划分任务、负载均衡、数据依赖、调试	“切块不易，Bug 难抓”
并发 vs 并行	单核靠时分，并行靠真多核	图示对比直观
Amdahl 定律	串行部分决定加速上限	“瓶颈在串行”
用户线程 / 内核线程	用户库维护 vs. 内核直接支持	“库快但受限，内核全能”
三种映射模型	Many-to-One、One-to-One、Many-to-Many	“多一、多多” 快速对比
线程库	Pthreads、Windows、Java	跨平台 API 基石
隐式并行	线程池、OpenMP、GCD	“工具自动并行”
线程池优势	复用、限流、策略灵活	“先备兵，再调度”
OpenMP 语法	`#pragma omp parallel/for`	编译器帮你分核
典型难题	fork/exec 语义、信号分派、取消、TLS	多线程版“七伤拳”
Linux 实现	`clone()` + `task_struct`	“任务(task)而非线程”

一、线程为何重要

- **动机**：现代应用往往包含显示刷新、后台下载、拼写检查等并行任务，线程允许它们**彼此独立又共享进程资源**，且创建/切换成本远低于进程。
- **四大收益**：
  1. **响应性**：界面不会因 I/O 阻塞而假死。
  2. **资源共享**：同一地址空间天然共享，免去显式 IPC。
  3. **经济性**：线程切换无需完整上下文，CPU/内存开销小。
  4. **可伸缩**：在多核中真正并行，充分榨干硬件。

二、多核与并行思维

- **挑战清单**：任务划分、负载平衡、数据切片、依赖消除与并行调试。
- **并发 vs. 并行**：单核靠调度让多个线程“看似同时”，多核才能实打实并行；教材中的时间片与两核示意图形象说明。
- **Amdahl**：若串行比例 S=25%，无论再加多少核，最高加速 = 1/S = 4；提示我们应先优化串行段。

三、线程的实现与映射

模型	描述	优劣
Many-to-One	多用户线程映射单内核线程，阻塞/不并行	简单，但多核利用率=0
One-to-One	一用户线程=一内核线程	并行好，线程数受内核限制
Many-to-Many	多用户线程 ⇆ 若干内核线程	兼顾灵活与并行，可动态调度内核线程

四、常用线程库

- **Pthreads**：POSIX 标准 API，Unix 家族通用；实现可在用户态或内核态。
- **Windows Threads**：Win32 API，CreateThread / _beginthreadex。
- **Java Threads**：由 JVM 管理，`extends Thread` 或实现 `Runnable`。

五、隐式并行技术

1. **线程池**：预生成固定线程等待任务，避免频繁创建并允许限制并发度；Windows API 已内建。
2. **OpenMP**：编译指令 `#pragma omp parallel/for` 让循环自动分片并行。
3. **Grand Central Dispatch (GCD)**：macOS/iOS 任务队列模型，按核心数动态派发。

六、常见线程编程坑

- **fork()/exec()**：不同 Unix 实现决定是复制所有线程还是仅调用者，需要留意。
- **信号处理**：多线程下可指定某线程专收信号亦可广播，设计不好易丢信号。
- **线程取消**：分为立即(asynchronous) 与 延迟(deferred)，必须保证资源回收。
- **TLS**：为每线程私有的全局数据，比函数局部变量生命期更长且线程隔离。

七、Linux 中的线程

- `clone()` 系统调用按 flag 选择共享/私有资源，本质上把线程称为 “task”，所有信息保存在 `task_struct`。

复习建议

1. **记口诀**：“快、易、省、扩”概括线程优势，“多一、多多”速辨三模型。
2. **画模型图**：把映射关系、栈/寄存器共享情况手绘一遍，有助答题。
3. **代码实操**：用 Pthreads 写个线程池，再加 `#pragma omp parallel for` 对比性能，体会 Amdahl 限制。

> 有了以上笔记，你可以在考试或项目中快速定位概念，结合上一章 **Processes** 的 PCB 与状态机，形成“进程-线程”完整知识链。