COMP3301 - Processes 速记表

模块	关键概念	一句话记忆
进程概念	进程 = 正在执行的程序，包含代码、数据段、堆与栈	“被加载到内存的程序才算活”
五种状态	new、ready、running、waiting、terminated	状态图像“泵”式循环
PCB	保存状态、寄存器、调度与内存信息	“进程身份证 + 行李清单”
调度三级	长程(Job)、中程(Memory swap)、短程(CPU)	长控数量，中调换页，短抢时片
上下文切换	保存旧 PCB，恢复新 PCB；硬件越好越快	切换 = 纯开销，别滥用
创建/终止	fork + exec 树状派生；exit/wait 回收资源	“父生子、子替身、退出要善后”
IPC 模型	共享内存 & 消息传递	“共享快，消息安全”
消息传递	`send/receive` 建链交换	两步走：先连后发
缓冲策略	0 / n / ∞ 容量队列	不缓冲就 rendezvous
POSIX 共享内存	`shm_open` ➔ `ftruncate` ➔ 写入	“文件式”共享，易跨进程
Sockets RPC Pipes	套接字端口、远程调用、匿名/命名管道	本机 127.0.0.1，端口 <1024 属系统
Linux task_struct	`pid`, `state`, `parent`, `mm`, `files`	内核用 C 结构体描述一切
移动多任务	iOS 单前台+受限后台；Android service 维持任务	节能 UI 限制，后台靠服务

核心脉络详解

1. **进程 vs. 程序**
   程序保存在磁盘是"静态文本"，只有被加载并拥有自身运行环境后才成为"进程"这一"动态实体" 。
2. **状态流转**
   状态机从 *new* 经 *ready* → *running* ↔ *waiting* → *terminated*，由调度与 I/O 事件驱动切换；教材中的椭圆-箭头图非常直观 。
3. **PCB（进程控制块）**
   PCB 记录所有现场信息，切换时由内核推入/弹出；越多字段意味着上下文切换开销越高 。
4. **调度器三分**
   - **长程**：控制系统并发度；决定新进程何时入内存。
   - **中程**：通过交换（swap）让暂停进程暂存磁盘，缓解内存压力。
   - **短程**：毫秒级频率，决定下一运行进程 。
5. **进程生命周期**
   `fork()` 复制地址空间，`exec()` 覆写新程序；`wait()` 收尸避免 zombie，若父进程先死则产生 orphan，由 init 接管 。
6. **上下文切换**
   OS 保存寄存器与程序计数器后再恢复目标 PCB；此过程纯粹"账务成本"，频率需权衡响应与效率 。
7. **IPC 双路线**
   - **共享内存**：速度快，需显式同步。
   - **消息传递**：封装好，代价是内核介入拷贝 。
     POSIX `shm_open` 用文件描述符映射共享段，示例代码演示创建-截断-写入完整流程 。
8. **通信细节**
   消息容量决定阻塞策略；直接通信需显式命名对方，间接通信依赖邮箱；缓冲三档从 0 到 ∞，影响生产者-消费者同步 、 。
9. **客户端-服务器范式**
   Socket 以 "IP:Port" 标识端点，RPC 通过 Stub 隐藏网络细节，Pipe 则提供同机进程流式通信 、 。
10. **Linux 实例**
    `task_struct` 揭示内核对进程的真实表示：父子链表、时间片、地址空间等域一应俱全 。

记忆口诀

> **"新就备，备就跑；跑等终，五态跑。"**
> **"长控数，中换人，短抢核；PCB 存档，切换慎用。"**

使用建议

- **答题框架**：先阐述进程定义→状态→PCB→调度→IPC，可快速覆盖核心。
- **举例**：说明 `fork-exec` 流程时可配合 Linux `task_struct` 字段以增强说服力。
- **拓展**：与前两章（OS 定义、结构）结合，把调度放回"资源分配器"大框里可形成闭环。