3 Process

进程 （在操作系统中）正在执行的应用程序。由以下部分构成：

- 代码段 (text section) （可以理解为指令的集合）；
- 程序计数器 (program counter)；
- 栈 (stack) （中间值：临时变量、返回值等等）
- 数据段 (data section) （全局变量）
- 堆 (Heap) （内存空间的动态调整）

进程在内存中表示：（由下至上）代码段、数据段（已初始化的 (initialized)、未初始化的 (uninitialized)）、堆、（空闲区域）、栈（包括参数 (arguments)）。

状态 初始状态、运行状态、等待状态、准备状态、终止状态。

进程控制块（Process Control Block, PCB）

- 进程状态 (process state)
- 进程序号 (process number)
- 程序计数器 (program counter)
- 寄存器 (registers)
- 内存管理 (memory-management)
- 用户信息 (user data information)
- I/O状态信息 (I/O status information)

线程 (Thread) 进程的细分，见第4章。

进程的表示 在Linux中代码用 C 程序的结构体表示，包括 PID, state, 父进程, 子进程 等等；用链表联系起来。

进程调度 选择一系列的进程给 CPU 的处理器运行。进程调度中包括一些进程队列（如*准备队列 (ready queue)*（已处于准备状态、等待执行的进程队列），*等待队列 (wait queue)*（如等待 I/O 设备的进程队列），进程在各个队列中切换。

上下文切换 (context switch)：CPU从一个进程切换到另一个进程。

- 保存旧进程的状态信息（PCB信息）；
- 加载新进程的状态信息（PCB信息）。

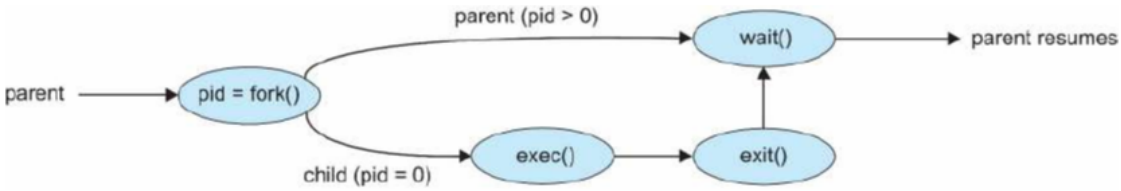
上下文 (context) 指 PCB 信息。

- PCB 信息越多，上下文切换时间越长；
- 硬件给每个 CPU 提供了许多寄存器，可以支持同时加载多个 PCB 信息。
- 上下文切换的频率不宜过高，否则系统的效率会较低。

系统的多进程

- 前端进程 (foreground)：和用户直接交互的进程（用户满意度直接相关，用户关心）；
- 后端进程 (background)：在内存中运行，但是暂时不进行显示（用户不是很关心）。
- 前端进程的优先级高于后端进程，因为前端进程的要求时间短，而后端进程一般没有时间要求。

进程的创建：通过父进程创建子进程。进程的创建形成了一棵树形结构（父进程-子进程关系）。



- 创建 PID。
- 资源共享选项：
  - 父进程和子进程共享所有资源；
  - 子进程共享父进程的部分资源；
  - 父进程和子进程不共享任何资源；
- 执行选项：
  - 父进程和子进程同时执行；
  - 先执行子进程，子进程执行完成后父进程接着执行。
- **Tips** 如果父进程和子进程共享较多资源，一般两个进程不能并行执行，因为共享的数据可能会出现冲突；一般共享数据较少的进程可以并行执行。
- 子进程的 pid = 0，父进程的 pid > 0。

进程的终止

- 可以通过 abort() 或 exit() 来终止进程。
- **Cascading termination**（强行终止）：父进程被终止后，其所有子孙进程都会被终止。
- 几种特殊情况：
  - 如果父进程终止了，但是子进程还在继续执行，则子进程为**孤儿**进程 (orphan)；
  - 如果子进程执行结束了，但是没有触发令父进程的 wait() 失效，则父进程为**僵尸**进程 (zombie)。

进程的分级：前端进程、可见进程、服务进程、后端进程、空进程。

进程间通信 (Interprocess Communication, IPC)

- 进程分为两类：独立进程和协作进程；
- 操作系统中大部分都是协作时进程，由于协作式进程具有如下优点：
  - 使得信息的共享更容易 (Information sharing)，提高内存使用效率；

    > [*Example*] Both process A and Process B need to access File C;

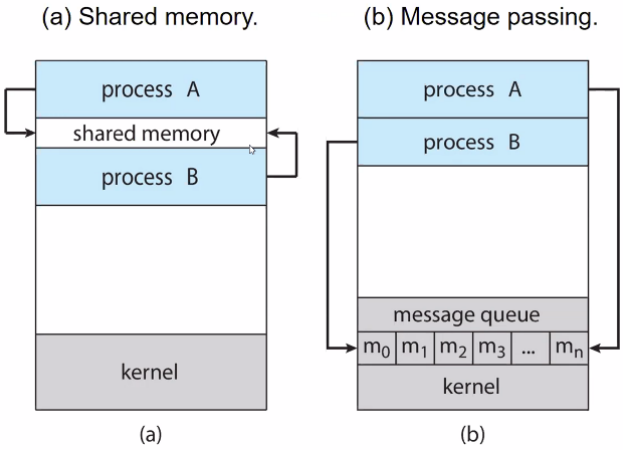
  - 加速计算 (Compuatation speedup)；

    > [*Example*] Process A need to compute $a+b$, while Process B need to compute $a+b+c$.

  - 模块化 (Modularity)；

    > [*Example*] Destruct a big process into several modules for parallelism.

  - 从工业界角度更方便 (Convenience)。
- 协作进程需要在进程间通信（IPC）。
- IPC 的两个模型：



- 共享内存 (shared memory)：安全性和隐私性好、去中心化管理（不用 kernel 管理、内核负担小）、区域自治，但是需要大量内存空间予以支持、效率偏低；
  - **生产者-消费者问题 (Producer-Comsumer Problem)**

    有两个进程同时对 buffer 操作，A 进程往 buffer 里增加数据，B 进程从 buffer 里消耗数据。两种方案：无限缓冲区（unbounded-buffer），有限缓冲区（bounded-buffer）。*有限缓冲区* 更为常用。
    - 等待 buffer 存在下一个空位再生产数据；
    - 等待 buffer 存在数据再消耗数据。
    - 可能存在一些问题，采用**进程同步**。
- 消息传递 (message passing)：效率高、中心化管理（kernel 统一管理）、节省内存空间，但是安全性、隐私性不如共享内存、内核负担和开销会加大（如果消息特别多内核可能无法承载）。
  - 一些实现问题：如何构造链接 (link)，链接是否可以连接多个进程，链接的容量，通过链接传送的消息是否定长，链接是单项还是双向等等。
  - 几种链接的实现方式：
    - 物理：共享内存、硬件总线、网络；
    - 逻辑：直接/间接、同步/异步、自动转发（收到消息直接转发）和缓冲转发（累计一定的消息再转发）

直接通信

- `send(P, message)` 将 message 发送给进程 P；
- `receive(Q, message)`，从进程 Q 接收 message。
- 链接的特点：
  - 链接是自动建立的（自动转发）；
  - 两个进程中有且仅有一个链接；
  - 链接通常是双向的。

间接通信

- 信息通过邮箱/端口 (mailbox / port) 发送和接收（邮箱可以理解为 buffer）；
- 每一个邮箱有一个独立的编号；
- 只有共享同一个邮箱的进程才能进行通信。
- 链接的特点：
  - 只有两个进程有共享邮箱，才有链接建立；
  - 一个链接可能涉及到许多进程；
  - 每对进程可能共享多个邮箱，从而有多个链接；
  - 链接可以是单项的，也可以是双向的。

- 操作：
  - 建立新的邮箱 / 端口（create mailbox）；
  - 通过邮箱发送 / 接受信息；
    - `send(A, message)` 将 message 发送给邮箱 A；
    - `receive(A, message)` 从邮箱 A 接收 message。
  - 回收邮箱（destroy mailbox）。
- 需要做很多的限定，以确认消息能够准确传递。

消息同步性

- **消息同步**：阻塞 (blocking) 操作 (Note: 阻塞 (blocking) 不等于拥塞 (congestion))
  - 阻塞发送 (blocking send)：发送者被阻塞，直到消息接收成功；
  - 阻塞接收 (blocking receive)：接收者被阻塞，直到消息可用。
- **消息异步**：非阻塞 (non-blocking) 操作
  - 非阻塞发送：发送者发送消息后继续操作；
  - 非阻塞接收：接收者接收一个可用消息或一个空消息 (null message)。

缓冲区 (Buffering)：链接的消息队列 (message queue)

- Zero capacity（零缓冲区）：发送者必须等待消息被接收者接收；
- Bounded capacity（有限缓冲区）：能够容纳 n 个消息的队列，如果缓冲区满，发送者必须等待至少一条消息被接收者接收；
- Unbounded capacity（无限缓冲区）：能够容纳无限个消息的队列，发送者不需要等待消息被接收者接收。

本地过程调用 (Local Procedure Call, LPC)：用于同一系统中的不同进程（通常进程连接着不同的设备），要求信息在设备间传输，则通过端口进行消息传输。（间接通讯）

管道 (Pipe)：用于在进程间传输大批量的数据。

- 通用管道（匿名管道）：一般传输数据量较大，用于父进程和子进程通信；支持双向传输；不过一般传输方式为单向（由于一个方向的数据已经占满管道）。
- 命名管道：传输的数据量比通用管道更大，不限于父子进程通信。大流量专线管道。

套接字 (Socket)：用于点对点的通信，间接通信。将IP地址和端口号封装起来形成 socket，通过电脑间的网络端口进行通信。

远程过程调用 (Remote Procedure Call, RPC)：用于网络中不同进程的通信，同样利用端口进行消息传输（间接通讯）。可能出现不稳定现象（重传）。