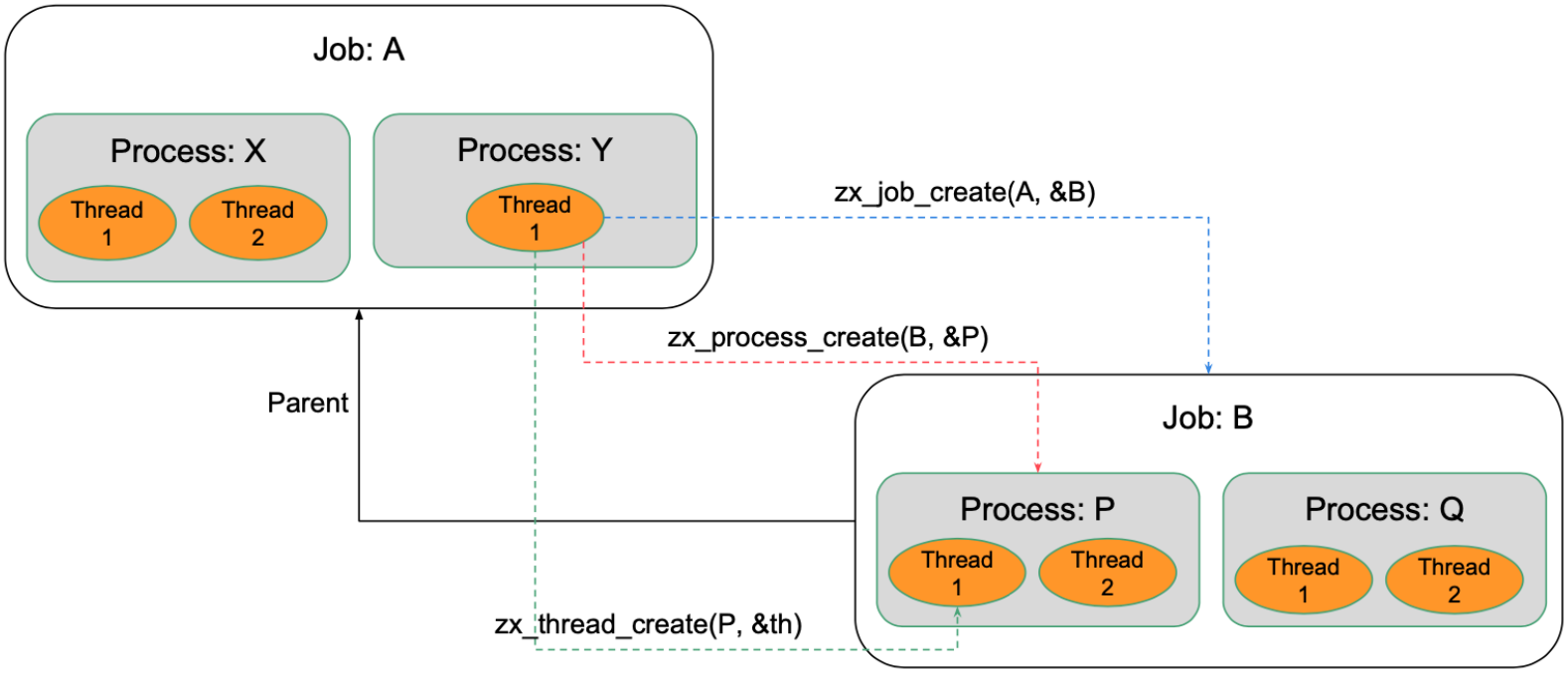
Zircon 内核对象简介：Task

王润基

2019.11.22

Task 相关对象

- Job：进程组
- Process：进程
- Thread：线程



Zircon Task 的主要特点

- 所有对象从零开始构造，没有 `fork`！（类 Windows）
- Job 具有父子关系，Process 和 Thread 之间是平级关系
- 三种对象的角色：
  - Job：权限控制 基本单位
  - Process：资源管理 基本单位（各种内核对象）
  - Thread：执行调度 基本单位

Job：进程组

- 对应实际的 应用（Application）
- 具有父子关系，子Job的权限 <= 父Job

Process：进程

- 对应传统 Unix 中的进程
- 拥有自己的资源（内核对象的 handle）

Thread：

- 对应经典的线程概念
- 创建后处于休眠态，需要手动开始执行

创建任务的典型流程

1. 创建一个子 Job，指定其权限
2. 在此 Job 中创建一个新 Process
3. 在此 Process 中创建一个新 Thread
4. 创建一个 Channel，向其中一端塞入要传给新进程的 handles
5. 启动线程，同时把 Channel 另一端作为参数传过去
6. 新线程运行后，从 Channel 中读取各种信息

Task 各对象 syscall 简介

- `SYS_job_*`
- `SYS_process_*`
- `SYS_thread_*`
- `SYS_task_*`：适用于 Job + Process + Thread

Job

- `create(parent_job, options) -> job`
  - 在 parent_job 中创建子 job
  - job 树有深度限制
- `set_policy(options, topic, policies)`
  - 设置 job 权限
  - 比较复杂，等到权限管理时再分析

Process

- `create(job, name, options) -> (proc, vmar)`
  - 在job中创建一个新进程
  - 进程不会自动运行，需要调用start
- `proc.start(thread, entry, stack, arg1, arg2)`
  - 在proc中启动线程thread，入口点entry，栈指针stack，handle arg1，参数arg2
  - 类似thread.start，用于启动第一个线程
  - arg1是个handle，move语义，如果失败则drop
- `exit(retcode)`
  - 退出当前进程，返回码retcode

Process

- `proc.read_memory(vaddr, buf) -> len`
  - 读取目标进程内存
  - buf.size() <= 64MB
- `proc.write_memory(vaddr, buf) -> len`
  - 写入目标进程内存

注：这两个syscall未来可能被移动到VMO中

Thread

- `create(proc, name, options) -> thread`
  - 在proc中创建一个新线程
  - 进程不会自动运行，需要调用start
- `thread.start(entry, stack, arg1, arg2)`
  - 启动线程，入口点entry，栈指针stack，参数arg1 arg2
  - entry函数必须在结束前调用exit
- `exit() -> !`
  - 退出当前线程

Thread

- `thread.read_state(kind, buf)`
  - 读取线程的某些寄存器，目标线程不能处于执行态
  - 读取内容平台相关，见 `zircon/syscalls/debug.h`
    - 通用（FS/GS）、浮点、向量、调试寄存器
    - single-step?
- `thread.write_state(kind, buf)`
  - 写入线程的某些寄存器，同上

类似 ptrace，用于实现 debugger

Tasks (Job + Process + Thread)

- `task.kill()`
  - 杀掉 task
- `task.suspend()`
  - 暂停执行
- `task.create_exception_channel(options) -> channel`
  - 为task创建异常处理channel
  - 发生异常时，channel会收到一个消息，包含exception handle和info，异常task 暂停执行
  - 异常处理结束后，关闭exception handle，异常task恢复执行
- `task.resume_from_exception(port, options)`
  - TODO