

## • 共性问题

- `profile`和`memory_profile`装饰器同时使用
  - 当二者同时使用时，最外层的装饰器只能返回内层装饰器对应的`wrapper`一行代码的统计，和装饰器本身的设计有关，不应要求同时统计。
- 有些同学直接用函数调用的形式使用了装饰器  
`wrap(func)()`
  - 这样只作用在本次调用中，而装饰器`@`的形式是写在函数定义部分的，注意这一差别
- 装饰器对参数和返回值的处理
  - 原则是不改变函数的输入和输出
  - 装饰器应该返回目标函数的返回值
  - `results=func(*args,**kwargs)`
  - `return results`



School of Economics and Management, Beihang University

# 现代程序设计技术

赵吉昌

[jichang@buaa.edu.cn](mailto:jichang@buaa.edu.cn)

- 面向对象编程
  - 进程与线程
  - 多进程

- 并行与并发

- 并行 ( parallel )

- 在同一时刻，有多条指令在多个处理器上同时执行

- 并发 ( concurrency )

- 在同一时刻，只能有一条指令执行，但多个指令被快速轮转执行，使得在宏观上表现多个指令同时执行的效果

- 进程和线程

- 进程 ( process )

- 操作系统分配资源的基本单位

- 线程 ( thread )

- CPU调度和分派的基本单位

- 应用程序至少有一个进程

- 一个进程至少有一个线程

- 同一进程的多个线程可以并发执行

- 进程在执行过程中拥有独立的内存单元，而线程共享内存

- 多进程编程需要考虑进程间的通信 (IPC)

- 进程切换时，耗费资源较大

# 进程和线程的比较



| 对比维度    | 多进程                       | 多线程                   | 总结   |
|---------|---------------------------|-----------------------|------|
| 数据共享、同步 | 数据共享复杂，需要用IPC；数据是分开的，同步简单 | 共享进程数据，数据共享简单，但导致同步复杂 | 各有优势 |
| 内存、CPU  | 占用内存多，切换复杂，CPU利用率低        | 占用内存少，切换简单，CPU利用率高    | 线程占优 |
| 创建销毁、切换 | 创建销毁、切换复杂，速度慢             | 创建销毁、切换简单，速度很快        | 线程占优 |
| 编程、调试   | 编程简单，调试简单                 | 编程复杂，调试复杂             | 进程占优 |
| 可靠性     | 进程间不会互相影响                 | 一个线程出错将导致整个进程中止       | 进程占优 |
| 分布式     | 适用于多核、多机分布式               | 适用于多核分布式              | 进程占优 |

- 进程 ( process )
  - 进程号
    - `os.getpid()`
  - 孤儿进程
    - 子进程在父进程退出后仍在运行，会被`init`进程接管，`init`以父进程的身份处理子进程运行完毕后遗留的状态信息
  - 僵尸进程
    - 父进程创建子进程后，如果子进程退出，但父进程并没有调用`wait`或`waitoid`获取子进程的状态信息，那么子进程的进程描述符将一直保存于系统，对应的子进程称为僵尸进程
    - 僵尸进程无法通过`kill`命令来清除

- multiprocessing模块
  - 开启子进程并在其中执行定制任务
  - 提供Process、Queue、Pipe、Lock等关键组件
  - 支持进程间的通信与数据共享
  - 执行不同形式的同步
  - 处理僵尸进程



- Process

- 创建进程类

- `Process([group [, target [, name [, args [, kwargs]]]])`

- 实例对象表示一个子进程（尚未启动）

- 使用关键字的方式来指定参数

- `group`参数未使用，值始终为`None`
    - `target`表示调用对象，即子进程要执行的任务
    - `args`指定传给`target`调用对象的位置参数，元组形式，仅有一个参数时要有逗号
    - `kwargs`表示调用对象的字典参数
    - `name`为子进程的名称

- 在Windows中`Process()`必须放到`if __name__ == '__main__':`中

- Process
  - `p.start()`
    - 启动进程，并调用该子进程中的`p.run()`
  - `p.run()`
    - 进程启动时运行的方法，会调用`target`，自定义类定要实现该方法
  - `p.terminate()`
    - 强制终止进程`p`，但不进行任何清理操作
    - 如果`p`创建了子进程，该子进程将成为僵尸进程
    - `p`创建的锁也将不会释放，可能导致死锁
    - 一般不建议使用

- Process

- `p.is_alive()`

- 如果`p`仍然运行，返回`True`

- `p.join([timeout])`

- 主进程等待`p`终止（主进程处于等待状态，而`p`处于运行状态）
    - `timeout`是可选的超时时间
    - 只能`join`住`start`开启的进程，而不能`join`住`run`开启的进程

- Process
  - `p.daemon`
    - 默认值为`False`
    - 可以设置为`True`，但必须在`p.start()`之前设置，成为后台运行的守护进程，当`p`的父进程终止时，`p`也随之终止，且`p`不能创建新的子进程
  - `p.name`
    - 进程的名称
  - `p.pid`
    - 进程的`pid`
  - `p.exitcode`
  - 进程在运行时为`None`
    - 如果为`-N`，表示被信号`N`结束
  - `p.authkey`
    - 进程的身份验证键

- 进程实现Demo

- 通过Process实现

- 通过继承Process实现

- Demo:mp\_error.py (on windows)

- Demo: mp.py

- 关于join的使用

- 先在主进程中启动所有子进程

- 然后在主进程中对所有子进程进行join

- 子进程都启动后再join, 启动后马上join会变成“串行”

- 虽然join仍会等等p1运行结束, 但其他子进程如p2, p3等仍在运行, 等p1运行结束后, 循环继续, p2, p3等可能也运行结束了, , 会迅速完成join的检验

- join花费的总时间仍然是耗费时间最长的子进程的运行时间

- 同步
  - 按预定的顺序先后执行
  - 调用后需要等到返回结果
  - 同步是保证多进程安全访问竞争资源的一种手段
- 异步
  - 与同步处理相对
  - 异步处理不用阻塞当前进程来等待处理完成，而是允许后续操作，并回调通知

- 临界资源
  - 一次仅允许一个进（线）程使用的资源称为临界资源
- 临界区
  - Critical Section
  - 存取临界资源的代码片段
  - 多进程要求进入空闲的临界区时，一次仅允许一个进（线）程进入
  - 如已有进（线）程进入临界区，则其它试图进入临界区的进（线）程需要等待
  - 进入临界区的进（线）程要在有限时间内退出

- 互斥量
  - mutex
  - 是一个仅处于两态之一的变量
    - 解锁
    - 加锁



- 进程的同步

- 实现机制

- 互斥锁 (Lock)
    - 信号量 (Semaphore)
    - 事件 (Event)
    - 条件 (Condition) [下周结合共享内存举例]

- Demo: `mpl.py` `mps.py` `mpe.py`

- 特征

- 可以用文件共享数据
      - 效率低
    - 加锁可以保证多个进程修改同一块数据时，同一时间只能有一个进程可以进行修改，即串行修改
      - 但需要自行加锁处理

- 基于消息的IPC通信机制
  - IPC
    - inter-process communication
  - 队列和管道

- **队列**

- `Queue([maxsize])`
- `q.put()`
  - 插入数据到队列，参数 `blocked` 默认为 `True`
  - 可能抛出 `Queue.Full` 异常
- `q.get()`
  - 从队列读取并删除一个元素
  - 可能抛出 `Queue.Empty` 异常
- 不可靠方法
  - `q.empty()`
  - `q.full()`
  - `q.qsize()`

- 生产者-消费者模型

- Demo: `mpq.py`

- 如何往队列中发送结束信号

- 生产者进程发送

- 消费者进程进行判断

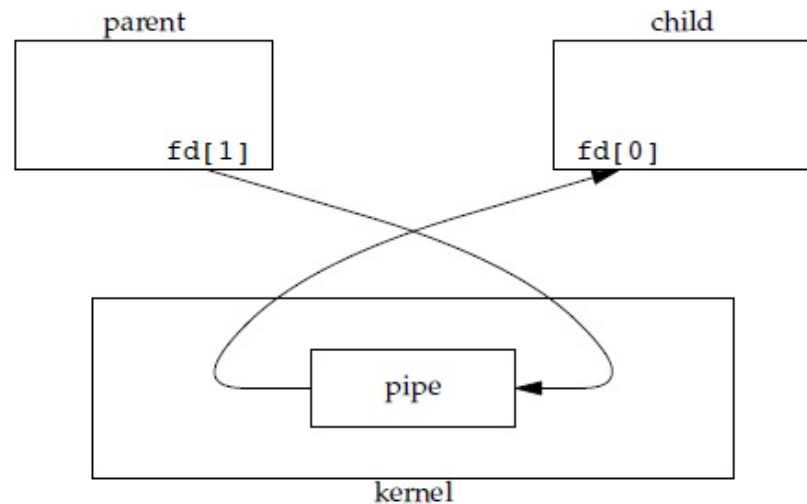
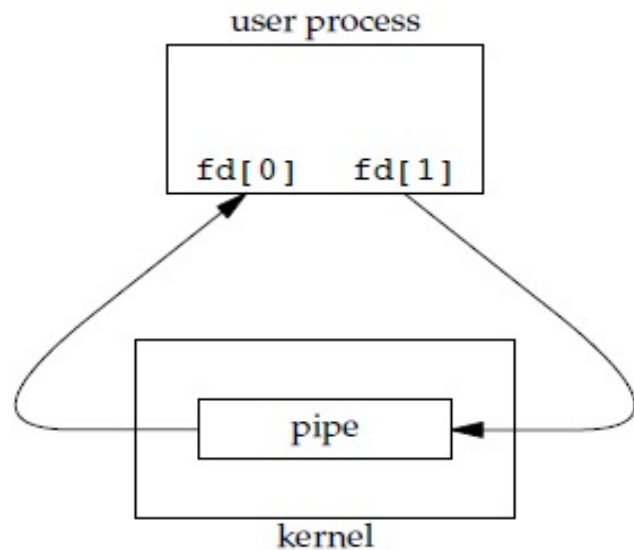
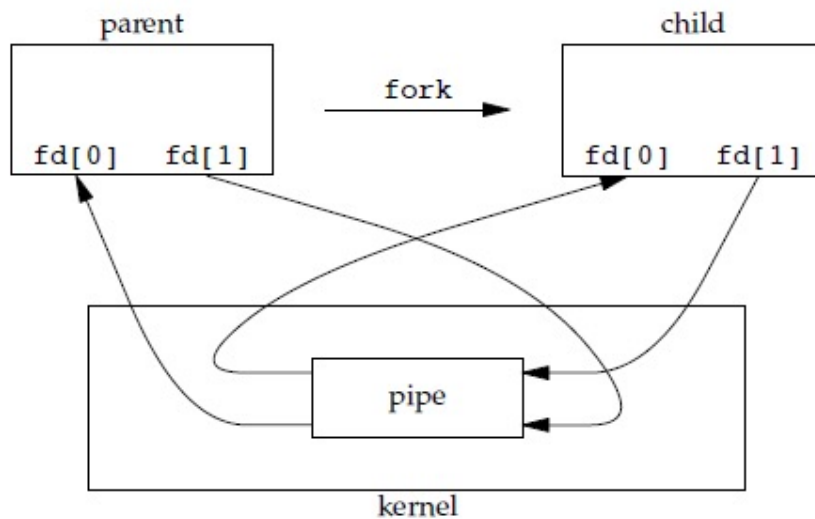
- 或主进程发送（注意要为每个消费者进程发送一个）

- `JoinableQueue([maxsize])`
  - 允许数据的消费者通知生成者数据已经被成功处理
    - 通过共享信号和条件变量来实现通知
  - `maxsize`
    - 队列中允许最大数据项数，省略则无大小限制
  - `q.task_done()`
    - 消费者使用此方法发出信号，表示`q.get()`的返回数据已经被处理，但如果调用此方法的次数大于从队列中删除数据的数量，将引发`ValueError`异常
  - `q.join()`
    - 生产者调用此方法进行阻塞，直到队列中所有的项目均被处理，阻塞将持续到队列中的每个数据均调用`q.task_done()`方法为止
  - Demo: `mpjq.py pq2.py`

- 管道 ( pipes)
  - FIFOs (named pipes) (Demo: 终端命令)
    - using regular files to communicate
    - the kernel takes care of synchronizing reads and writes
    - data is never actually written to disk (instead it is stored in buffers in memory)
    - the overhead of disk I/O is avoided.
    - A FIFO is part of the file system
  - Pipes
    - like FIFOs without the name
    - a read end and a write end
    - a read from a pipe *only gives end-of-file if all file descriptors for the write end of the pipe have been closed*
      - after a fork, whichever process is intending to do the reading (and thus not the writing) had best close the write end of the pipe
  - Processes communicating via pipes must be running on the same host
    - 补充：跨机需要通过socket

## • 管道

- 一般用于两个进程间通信
- 但也可以用于多个进程
- Demo: `piped.c` `pipes.c`



- 管道

- Pipe ( [duplex] )
- 在进程之间创建一条管道，并返回元组 ( conn1, conn2 )，其中conn1，conn2表示管道两端的连接对象
- conn1：读，接收消息
- conn2：写，发送消息
- 必须在产生Process对象之前产生管道
- duplex
  - 默认全双工，如果将duplex置为False，对于一个进程，只能通过conn1接收或者只能通过conn2发送



- 管道

- `conn1.recv()`

- 接收`conn2.send(obj)`发送的对象，如果没有消息可接收，`recv`方法会一直阻塞
    - 如果连接的另外一端（所有）已经关闭，那么`recv`方法会抛出`EOFError`

- `conn2.send(obj)`

- 通过连接发送对象，`obj`是与序列化兼容的任意对象

- `conn1.close()`

- 关闭连接

- Demo: `mpp.py`

# 本周作业



- MapReduce是利用多进程并行处理文件数据的典型场景。作为一种编程模型，其甚至被称为Google的“三驾马车”之一(尽管目前由于内存计算等的普及已经被逐渐淘汰)。在编程模型中，Map进行任务处理，Reduce进行结果归约。本周作业要求利用Python多进程实现MapReduce模型下的文档库( 搜狐新闻数据(SogouCS) ( 下载地址：<https://www.sogou.com/labs/resource/cs.php> ) , 注意仅使用页面内容，即新闻正文 ) 词频统计功能。具体地：
  - 1. Map进程读取文档并进行词频统计，返回该文本的词频统计结果。
  - 2. Reduce进程收集所有Map进程提供的文档词频统计，更新总的文档库词频，并在所有map完成后保存总的词频到文件。
  - 3. 主进程可提前读入所有的文档的路径列表，供多个Map进程竞争获取文档路径；或由主进程根据Map进程的数目进行分发；或者单独实现一个分发进程，与多个MAP进程通信。
  - 4. 记录程序运行时间，比较不同Map进程数量对运行时间的影响，可以做出运行时间-进程数目的曲线并进行简要分析。