2.1 今日目标

2.2 [重、难点]软件安装

- 离线安装：`sudo dpkg -i 软件.deb`
- 在线安装：`sudo apt install 软件名`
    - 为了提高下载速度, 可以切换镜像源

2.3 [重点]软件卸载

- 离线包卸载：`sudo dpkg -r 软件安装包名`
- 在线包卸载：`sudo apt remove 软件名`

2.4. [了解]多任务的介绍

- 概念：同一时间执行多个任务
- 多任务执行方式：
    - 并发: 同一时间交替运行
    - 并行: 同一时间多个运行

2.5 [重点]进程

- 概念：进程是操作系统进行资源分配的基本单位
- 作用：可以创建多个进程, 实现多任务

2.6 [重点]多进程的使用

- 进程使用的步骤：
    - 导入模块

      ```
      import multiprocessing
      ```

    - 创建子进程对象

      ```
      multiprocessing.Process(target=任务名)
      ```

    - 启动子进程

      ```
      子进程对象.start()
      ```

2.7 [重点, 理解]获取进程编号

- 获取进程的对象

  ```
  multiprocessing.current_process()
  ```

- 获取进程的编号

  ```
  os.getpid()
  ```

- 获取进程的父进程编号

  > `os.getppid()`

- 结束进程

  > `os.kill(进程编号, 9)`

2.8 [重点]进程执行带有参数的任务

- 进程的参数传递
  - `args`: 以元组方式传参
  - `kwargs`: 以字典方式传参

  ```
  sing_process = multiprocessing.Process(target=sing, args=(3, "吴东豹"))
  dance_process = multiprocessing.Process(target=dance, kwargs={"name": "舒棋", "count": 100})
  ```

2.9 [重点]进程的注意点

- 全局变量问题:

  > 创建子进程会对主进程资源进行拷贝，也就是说子进程是主进程的一个副本，好比是一对双胞胎，之所以进程之间不共享全局变量，是因为操作的不是同一个进程里面的全局变量，只不过不同进程里面的全局变量名字相同而已。

- 等待子进程:

  > 默认效果, 主进程会等待子进程结束, 再结束

- 守护进程:

  > 子进程对象.daemon = True

- 子进程结束:

  > 子进程对象.terminate()

2.10 [重点]线程

- 概念: 线程是cpu调度的基本单位
- 作用: 通过多线程, 实现多任务

2.11 [重点]多线程的使用

- 导入模块

  > `import threading`

- 创建线程对象

  > `threading.Thread(target=任务名)`

- 启动线程

> `线程对象.start()`

- 获取当前线程对象

> `threading.current_thread()`

2.12 [重点]线程执行带有参数的任务

- 线程执行带参数的任务：
  - 元组方式：

```
# 1) 元组传递参数 args=(5,) 按照顺序传递
sing_threading = threading.Thread(target=sing, args=(5,))
```

  - 字典方式：

```
# 2) 字典方式传递参数 kwargs={"count":5} 按照key名作为参数名传递
dance_threading = threading.Thread(target=dance, kwargs={"count": 3})
```

2.13 [重点]线程的注意点1

- 线程的执行顺序：无序
- 主线程会等待所有子线程结束：再结束
- 守护线程：

```
# 1.1 创建线程对象时，设置守护线程
# 1.2 线程对象.setDaemon(True)
# 1.3 线程对象.daemon = True
```

2.14 [重点]线程的注意点2

- 多线程之间共享全局变量：可以共享, 但是可能会遇到资源抢夺的问题, 导致数据异常
- 共享全局变量的问题：
  - 解决办法：
    - join(): 线程等待, 这种方式扔是单任务的
    - 互斥锁: 给全局变量修改的地方进行加锁. 只有数据处理的地方是单任务, 其他代码是多任务的

2.15 [重点]互斥锁

- 概念：对共享数据进行锁定，保证同一时刻只能有一个线程去操作。
- 使用：
  - 创建锁：`mutex = threading.Lock()`
  - 上锁：`mutex.acquire()`
  - 解锁：`mutex.release()`

2.16 [重点]死锁

- 概念: 没有及时解锁, 一直等待对方释放锁。
- 避免死锁：在适当的位置释放锁。

2.17 [重点]进程和线程对比

- 区别对比：
    - 概念
        - 进程是操作系统资源分配的基本单位
        - 线程是CPU调度的基本单位
    - 关系
        - 线程是依附在进程里面的，没有进程就没有线程。
    - 开销上:
        - 创建进程的资源开销要比创建线程的资源开销要大
    - 稳定性
        - 多进程开发比单进程多线程开发稳定性要强(多进程是真正的多任务, 资源是独立的)
    - 全局变量:
        - 进程不共享全局变量
        - 线程共享全局变量, 但是要注意资源竞争的问题, 需要通过互斥锁或者线程等待解决
- 优缺点对比：
    - 进程优缺点:
        - 优点：可以用多核
        - 缺点：资源开销大
    - 线程优缺点:
        - 优点：资源开销小
        - 缺点：不能使用多核

2.18 今日知识总结