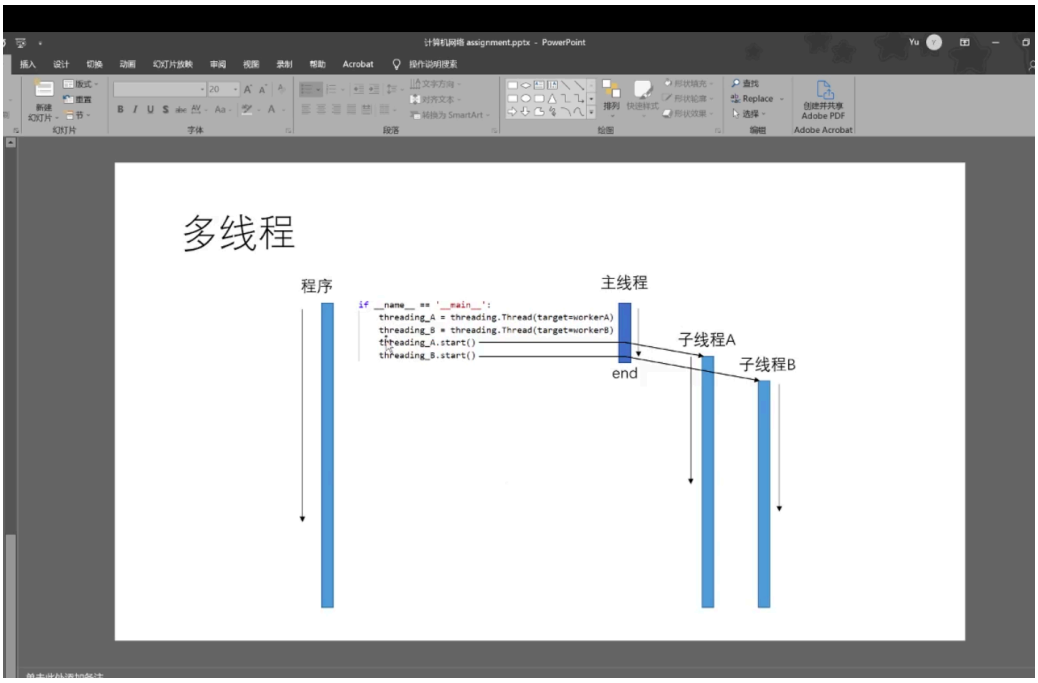
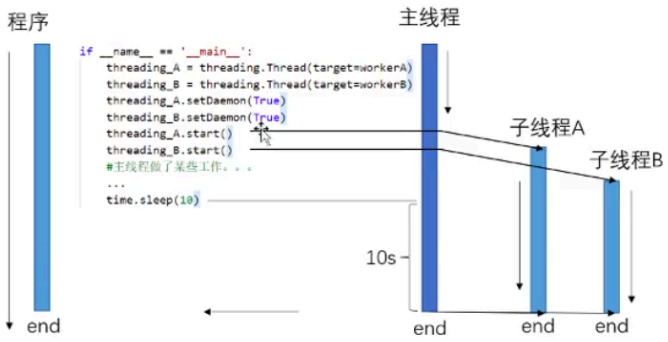
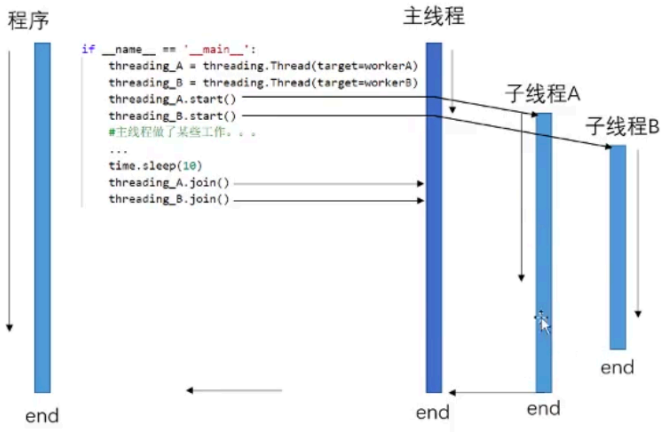


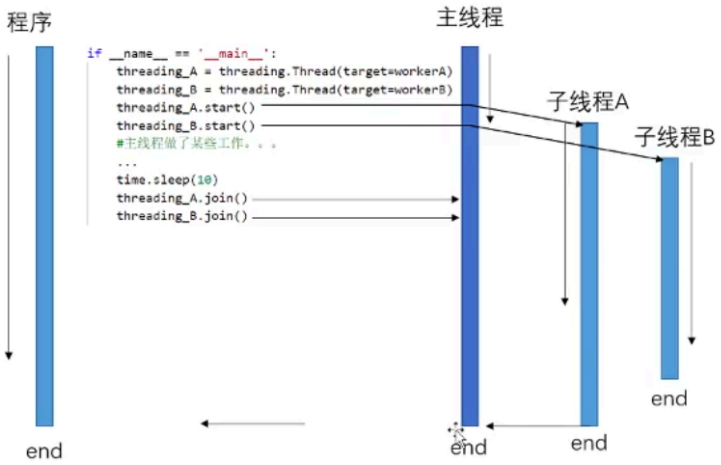
守护线程



多线程之间的同步



多线程之间的同步



线程的关闭

- 1. 通过添加用于表示关闭信号的变量
- 2. 通过设置子线程的子线程来控制

多线程同时使用一个资源

1. 什么是资源?

   文件、内存、某个变量、缓冲区。。。。都是资源

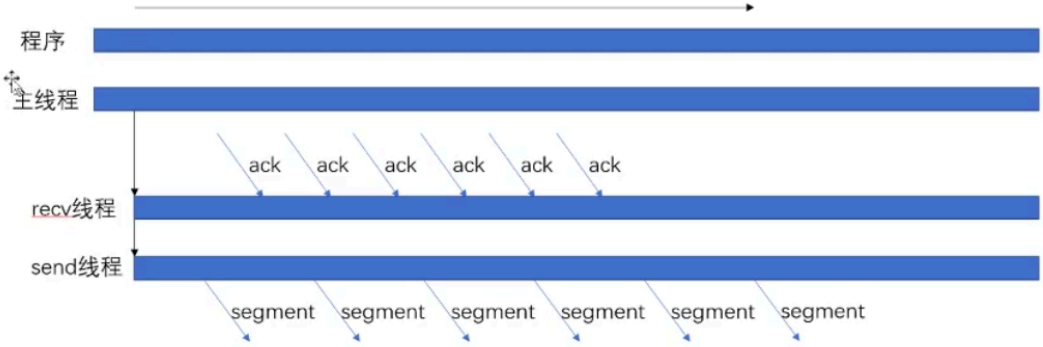
2. 多个线程同时使用一个资源会有问题吗?

   1. 如果多个线程只是读取资源的内容，不进行修改则不会有什么问题。
   2. 如果多个线程中，一个线程写，其他线程读，一般也不会有什么问题。
   3. 如果多个线程会对资源进行修改，那么一般就需要加锁。

3. 锁的作用是什么?

   在某个线程拥有锁的时候，其他线程没法打断操作，近似让其原子化。

在具体的程序中 1-socket用于sender



在具体的程序中 1-socket用于sender

在具体的程序中 1-socket用于sender

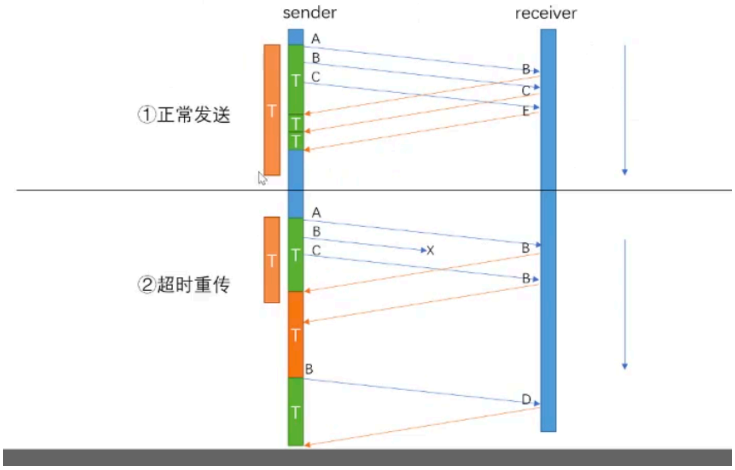
在具体的程序中 2-socket用于receiver

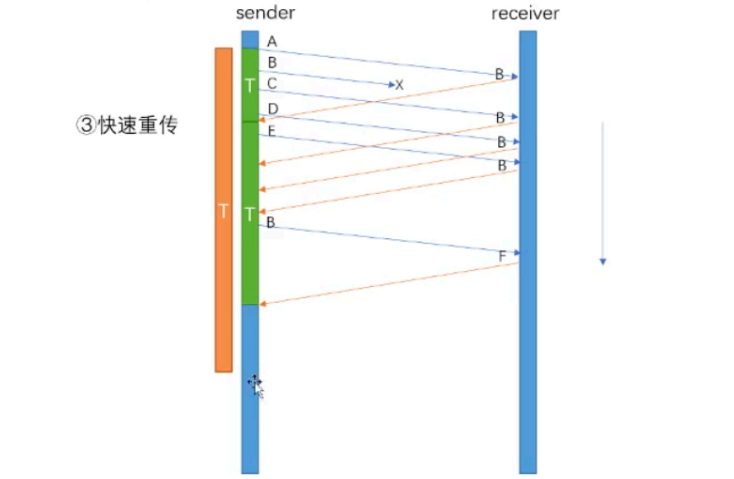
在具体的程序中 2-socket用于receiver

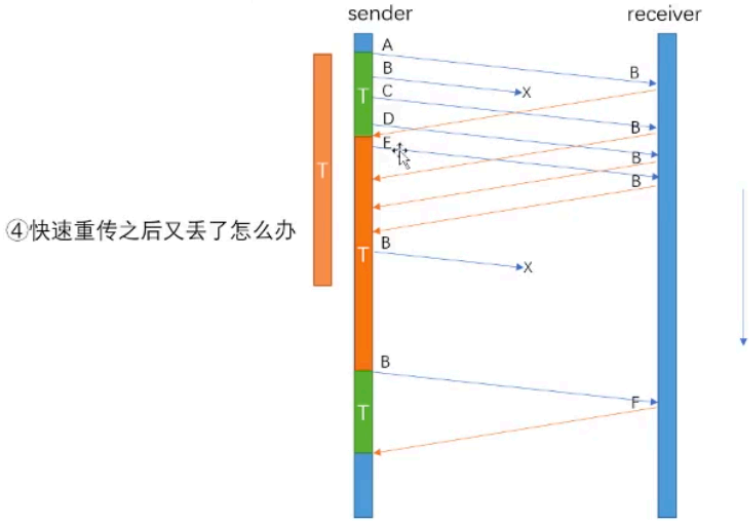
超时重传

- 超时重传中定时器的管理：
  - 1. 每次发送一个数据段的时候（不论是第一次发，还是重传的），如果此时定时器没有在运行状态，那么就重启它；如果在运行状态，就让它继续运行。
  - 2. 每当新的ack到达时，判断是否还有未ack的，如果有，那么重置定时器，如果没有，就删除定时器。
  - 3. 当所有的数据段都被ack了之后，关闭这个定时器。
- 当定时器触发时：
  - 重传还没被ack的段中，最早的一个。

超时重传


sender
receiver
①正常发送
②超时重传
A
B
C
X
D
E
T


sender
receiver
③快速重传
A
B
C
X
D
E
F
T


sender
receiver
④快速重传之后又丢了怎么办
A
B
C
X
D
E
F
T

超时重传

- 超时重传中定时器的管理：
  - 1. 每次发送一个数据段的时候（不论是第一次发，还是重传的），如果此时定时器没有在运行状态，那么就重启它；如果在运行状态，就让它继续运行。
  - 2. 每当新的ack到达时，判断是否还有未ack的，如果有，那么重置定时器，如果没有，就删除定时器。
  - 3. 当所有的数据段都被ack了之后，关闭这个定时器。
- 当定时器触发时：
  - 重传还没被ack的段中，最早的一个。

文件收发

read(*size=-1*)

从对象中读取 *size* 个字节并将其返回。作为一个便捷选项，如果 *size* 未指定或为 -1，则返回所有字节直到 EOF。在其他情况下，仅会执行一次系统调用。如果操作系统调用返回字节数少于 *size* 则此方法也可能返回少于 *size* 个字节。

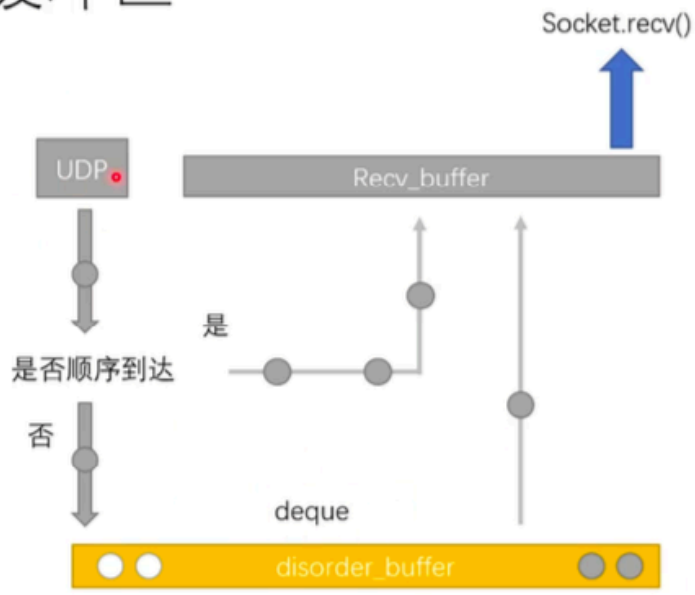
如果返回 0 个字节而 *size* 不为零 0，这表明到达文件末尾。如果处于非阻塞模式并且没有更多字节可用，则返回 None。

默认实现会转至 readall() 和 readinto()。

1. 新建PTP Socket
2. 连接接收方
3. 读取文件内容
4. 通过PTP Socket发送数据
5. 关闭文件
6. 关闭PTP Socket

1. 新建PTP Socket
2. 连接发送方
3. 通过PTP Socket接受数据
4. 写入文件
5. 关闭文件
6. 关闭PTP Socket

接受的缓冲区



发送的缓冲区