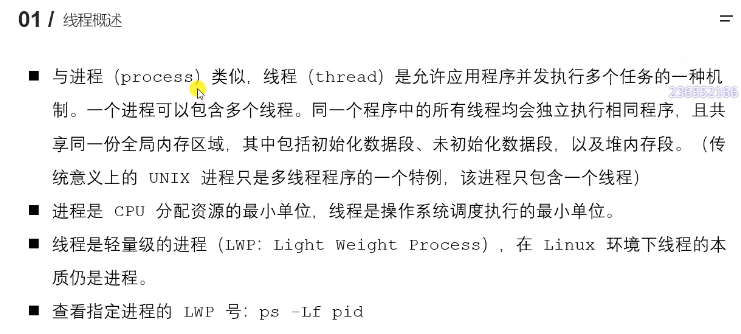
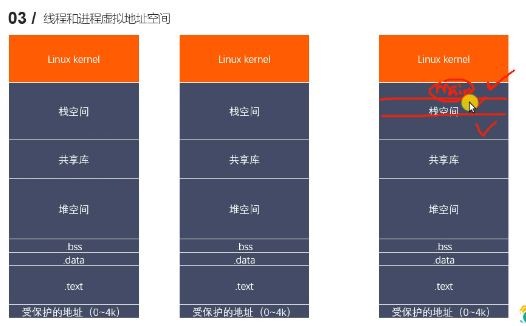
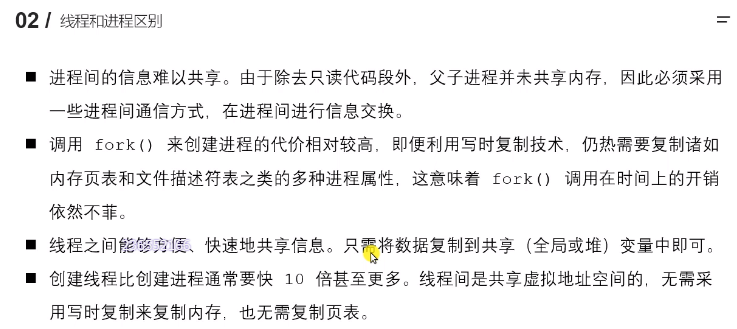
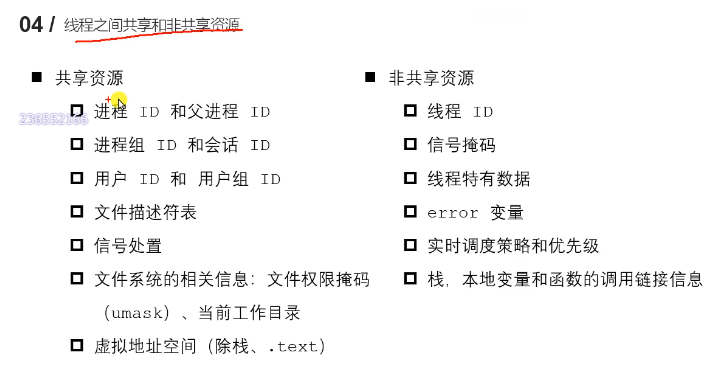
1. 线程概述

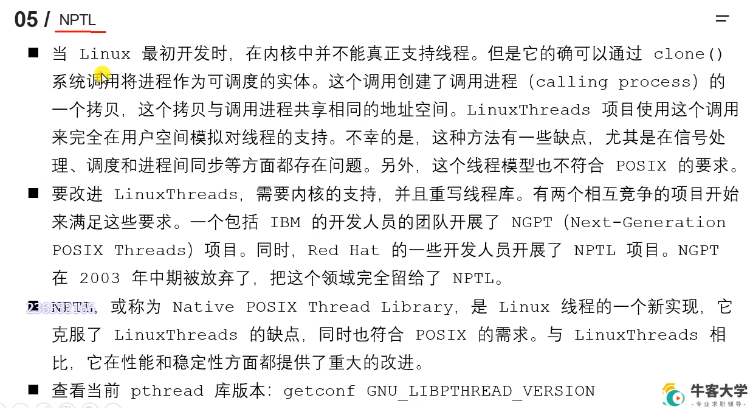


ps aux -> ps –LF pid：查看指定进程的线程。



线程虚拟地址空间发生变化（不共享）：栈空间，.text

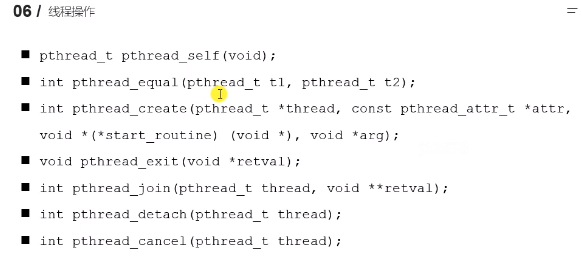


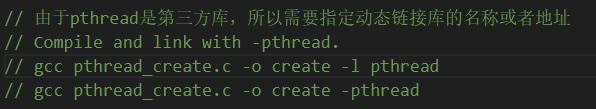


1. 创建线程

Linux环境下创建线程的方式：

1. 使用pthread库：pthread库是POSIX线程库的实现，是Linux下创建和管理线程的常用方式。
2. 使用C11标准的threads.h库：C11引入了一个新的线程库threads.h，该库提供了一组函数来创建和管理线程。
3. 使用C++11标准的 thread库：和C11相同





1. 终止线程

pthread库：

1、线程函数返回：线程函数可以通过返回来正常终止线程。当线程函数执行完毕并从函数中返回时，线程会自动终止。这是最常见的线程终止方式。如 return NULL；

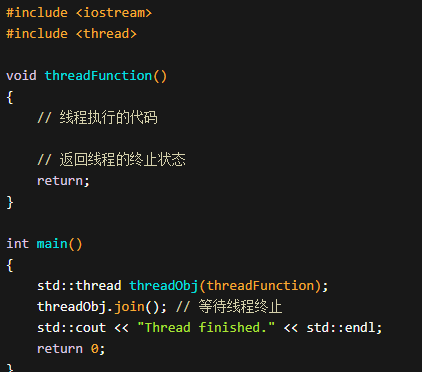
2、调用pthread\_cancel函数：可以使用pthread\_cancel函数向指定的线程发送取消请求，请求线程终止。被取消的线程需要在适当的位置检查取消请求并进行处理。

3、使用pthread\_exit函数：可以在任意位置调用pthread\_exit函数来终止当前线程。这会立即终止线程的执行，不会执行后续的代码。

thread库：

在C++11中，可以使用std::thread提供的方法来终止线程。然而，C++标准库没有提供直接的方式来终止线程，因为线程的终止可能导致资源泄露或不一致的状态。相反，C++11鼓励使用协作式的方式来终止线程。

1、通过退出线程函数：线程函数可以通过正常返回来终止线程。当线程函数执行完毕并从函数中返回时，线程会自动终止。如 return；



2、通过设置共享变量或标志位来控制线程终止：可以使用共享的标志位或变量来控制线程的执行流程。线程函数可以在适当的位置检查这个标志位或变量，并根据其值来决定是否终止线程。



3、使用std::condition\_variable来通知线程终止：可以使用条件变量来通知线程终止。线程函数在等待条件变量时，可以检查是否收到了终止的通知，如果收到通知，则终止线程。



1. 线程回收

1、在Linux下，当主线程退出后，运行中的子线程的资源不会自动回收。这意味着子线程继续执行，但其资源（如栈空间、线程控制块等）将不会被自动释放。

—使用可分离线程（Detached Thread）：在创建子线程时，可以通过设置线程属性将其设置为可分离线程。可分离线程在终止后会自动回收其资源，不需要其他线程调用pthread\_join来等待和回收子线程。

—其他子线程调用pthread\_join函数去释放子线程的资源。(不常用)。

2、在Linux下，在主线程退出前，主线程不会自动回收已经终止的子线程的资源。主线程可以通过调用pthread\_join来等待子线程的终止，并回收其资源。当子线程终止后，它的资源（如栈空间、线程控制块等）将保留在系统中，直到主线程调用pthread\_join来回收这些资源。

—使用pthread\_join等待子线程并回收资源：在主线程退出前，可以显式地调用pthread\_join来等待子线程的终止，并回收其资源。需要注意的是，使用pthread\_join等待子线程的终止可能会导致主线程阻塞，直到子线程终止。因此，确保在调用pthread\_join前主线程已经完成了需要处理的任务，或者可以在主线程中设置适当的超时机制来避免无限阻塞。

1. 连接已经终止的线程

Pthread\_join函数。

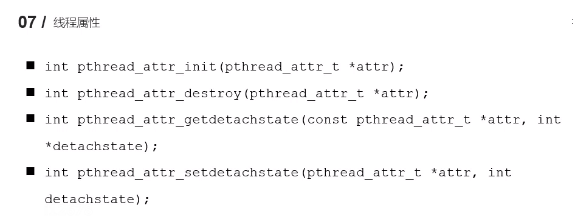
1. 线程分离

当子线程先于pthread\_detach()结束时，子线程的资源会等待到主线程结束时或整个进程结束时由操作系统自动回收，而不是在调用pthread\_detach()之后立即回收。

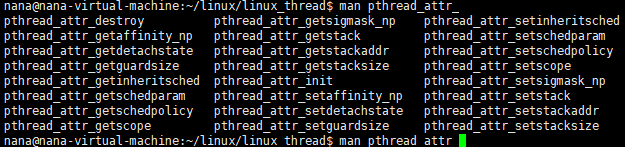
1. 线程取消

调用pthread\_cancel函数：可以使用pthread\_cancel函数向指定的线程发送取消请求，请求线程终止。被取消的线程需要在适当的位置检查取消请求并进行处理。但是并不会立即终止，而是执行到取消点（系统调用）(man pthreads)的时候才终止。一般是在主线程中使用该函数去取消子线程。

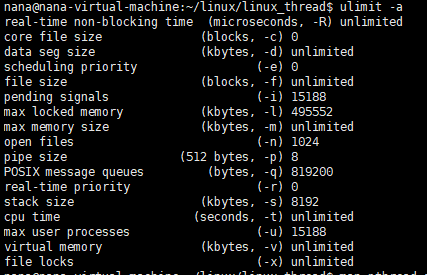
1. 线程属性



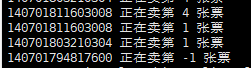
查看线程属性：man pthread\_attr\_+tab+tab



显示当前用户的资源限制情况：ulimit –a

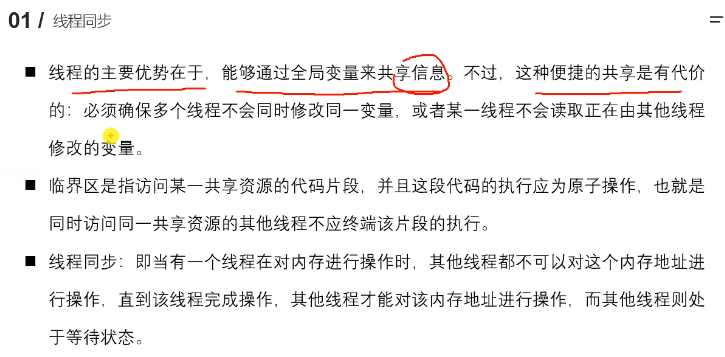


1. 线程同步



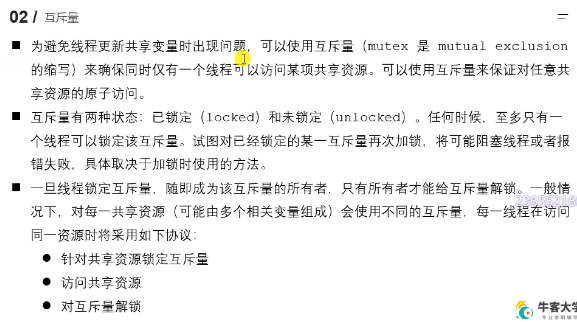
为什么会出现以上问题？最终有可能会卖-2张票。（三个子线程售票）。

没有设置线程同步。解决方法：互斥锁；读写锁；条件变量；信号量；

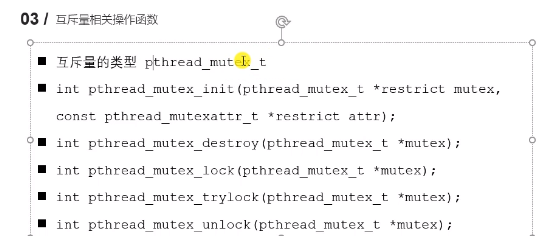


临界区是访问共享资源的代码段！！！！！！

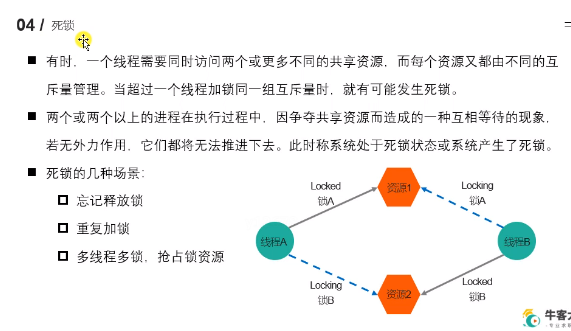
1. 互斥锁



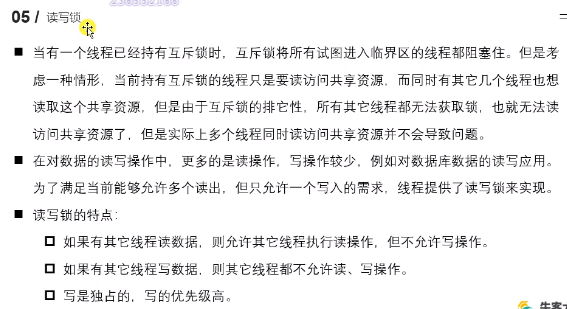
sudo apt-get install manpages-posix-dev。

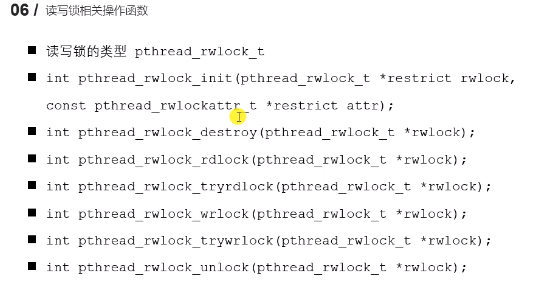


访问互斥量的时候可能出现死锁。



十一、读写锁





十二、生产者和消费者模型

生产者与生产者之间是互斥的，生产者与消费者之间是同步的，消费者与消费者之间是互斥的。

需要通过条件变量、信号量去建立生产者和消费者的通信。没有条件变量或信号量的模型中，无法实现生产者与消费者之间的同步。

一对一模型：

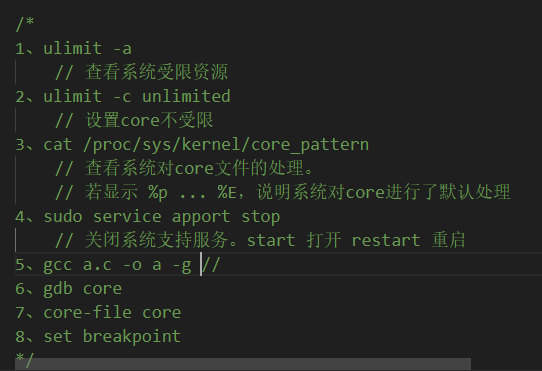
容器满的时候，生产者阻塞；容器空的时候，消费者阻塞。

多对多模型：

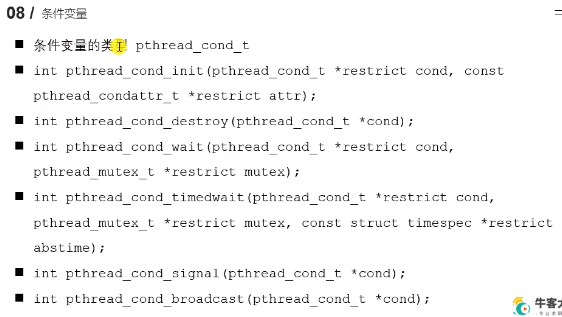
互斥锁：多个生产者不能同时访问容器，多个消费者也不能同时访问容器。（生产者之间互斥，消费者之间互斥）

读写锁：多个生产者不能同时访问容器，多个消费者可以同时访问容器；或者；多个生产者可以同时访问容器，多个消费者不能同时访问容器。（生产者之间互斥，消费者之间同步；生产者之间同步，消费者之间互斥）。

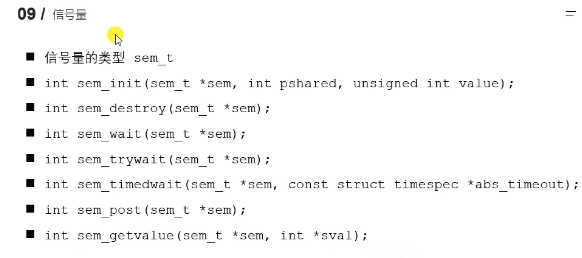
十三、生成 core 文件的问题



十四、条件变量



十五、信号量



只是用信号量不能保证线程安全问题。需要和互斥锁联合使用。