线程

1. 概念

是一个轻量级的进程，为了提高系统的性能引入线程

Linux 里同样用 task_struct 来描述一个线程。

线程和进程都参与统一的调度。

在同一个进程中创建的线程共享该进程的地址空间。

2. 线程和进程区别

共性：都为操作系统提供了并发执行能力

不同点：

调度和资源：线程是系统调度的最小单位，进程是资源分配的最小单位

地址空间方面：同一个进程创建的多个线程共享进程的资源；进程的地址空间相互独立

通信方面：线程通信相对简单，只需要通过全局变量可以实现，但是需要考虑临界资源保护的问题；进程通信比较复杂，需要借助进程间的通信机制(借助 3g-4g 内核空间)

安全性方面：线程安全性差一些，当进程结束时会导致所有线程退出；进程相对安全

3. 线程函数

3.1 创建线程

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                   void *(*start_routine) (void *), void *arg);
功能：创建线程
参数：thread：线程标识
      attr：线程属性， NULL：代表设置默认属性
      start_routine：函数名：代表线程函数
      arg：用来给前面函数传参
返回值：成功：0
        失败：错误码
```

3.4 获取线程号

```
pthread_t pthread_self(void);
功能：获取线程号
返回值：线程 ID
```

3.5 线程分离

```
int pthread_detach(pthread_t thread);
功能：让线程分离，线程退出让系统自动回收线程资源
```

练习：通过线程实现通信。主线程循环从终端输入数据，子线程循环将数据打印，当输入 quit 结束程序。

要求：先输入后输出

提示：标志位 int flag = 0;

4. 线程同步

4.1 概念

同步(synchronization)指的是多个任务(线程)按照约定的顺序相互配合完成一件事情

4.2 同步机制

通过信号量实现线程间同步。

信号量：由信号量来决定线程是继续运行还是阻塞等待，信号量代表某一类资源，其值表示系统中该资源的数量

信号量是一个受保护的变量，只能通过三种操作来访问：初始化、 P 操作(申请资源)、 V 操作(释放资源)

信号量的值为非负整数

4.3 特性

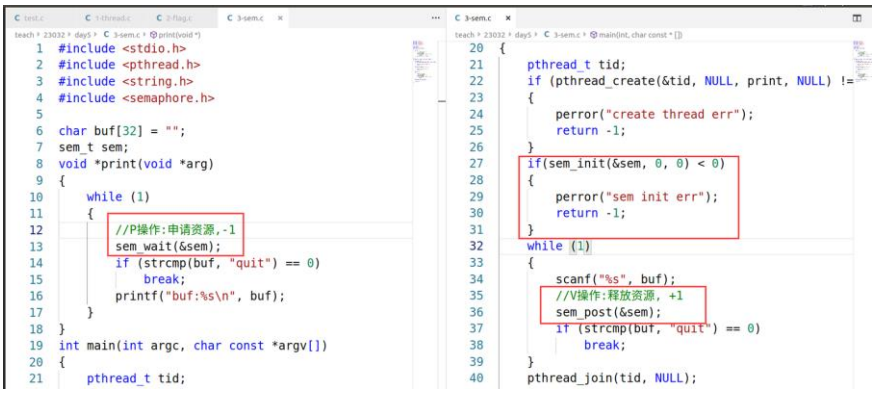
P 操作：

当信号量的值大于 0 时，可以申请到资源，申请资源后信号量的值减 1

4.4 函数

```
int  sem_init(sem_t *sem,  int pshared,  unsigned int value)
功能：初始化信号量
参数：sem：初始化的信号量对象
     pshared：信号量共享的范围(0：线程间使用    非 0:1 进程间使用)
     value：信号量初值
返回值：成功 0
       失败 -1
int  sem_wait(sem_t *sem)
功能：申请资源  P 操作
参数：sem：信号量对象
返回值：成功 0
       失败 -1
注：此函数执行过程，当信号量的值大于 0 时，表示有资源可以用，则继续执行，同时对信号量减 1；当信号量的值等于 0 时，表示没有资源可以使用，函数阻塞
int  sem_post(sem_t *sem)
功能：释放资源  V 操作
参数：sem：信号量对象
返回值：成功 0
       失败 -1
注：释放一次信号量的值加 1，函数不阻塞
```

举例：



```c
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <semaphore.h>

char buf[32] = "";
sem_t sem;
void *print(void *arg)
{
    while (1)
    {
        //P操作:申请资源,-1
        sem_wait(&sem);
        if (strcmp(buf, "quit") == 0)
            break;
        printf("buf:%s\n", buf);
    }
}
int main(int argc, char const *argv[])
{
    pthread_t tid;
    if (pthread_create(&tid, NULL, print, NULL) !=
    {
        perror("create thread err");
        return -1;
    }
    if(sem_init(&sem, 0, 0) < 0)
    {
        perror("sem init err");
        return -1;
    }
    while (1)
    {
        scanf("%s", buf);
        //V操作:释放资源, +1
        sem_post(&sem);
        if (strcmp(buf, "quit") == 0)
            break;
    }
    pthread_join(tid, NULL);
```

5.线程互斥

5.1 概念

临界资源：一次仅允许一个进程所使用的资源

临界区：指的是一个访问共享资源的程序片段

互斥：多个线程在访问临界资源时，同一时间只能一个线程访问

互斥锁：通过互斥锁可以实现互斥机制，主要用来保护临界资源，每个临界资源都由一个互斥锁来保护，线程必须先获得互斥锁才能访问临界资源，访问完资源后释放该锁。如果无法获得锁，线程会阻塞直到获得锁为止。

5.2 函数接口

```
int  pthread_mutex_init(pthread_mutex_t  *mutex,
pthread_mutexattr_t *attr)
功能：初始化互斥锁
参数：mutex：互斥锁
    attr:  互斥锁属性  //  NULL 表示缺省属性
返回值：成功 0
      失败 -1
int  pthread_mutex_lock(pthread_mutex_t *mutex)
功能：申请互斥锁
参数：mutex：互斥锁
返回值：成功 0
      失败 -1
注：和 pthread_mutex_trylock 区别：pthread_mutex_lock 是阻塞的；
pthread_mutex_trylock 不阻塞，如果申请不到锁会立刻返回
int  pthread_mutex_unlock(pthread_mutex_t *mutex)
功能：释放互斥锁
参数：mutex：互斥锁
返回值：成功 0
      失败 -1
int  pthread_mutex_destroy(pthread_mutex_t  *mutex)
功能：销毁互斥锁
参数：mutex：互斥锁
```

案例：全局数组 int a[10] = {};

t1: 循环倒置数组中元素

t2：循环打印数组中元素

5.3 死锁

是指两个或两个以上的进程/线程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去

死锁产生的四个必要条件：

1、互斥使用，即当资源被一个线程使用(占有)时，别的线程不能使用

2、不可抢占，资源请求者不能强制从资源占有者手中夺取资源，资源只能由资源占有者主动释放。

3、请求和保持，即当资源请求者在请求其他的资源的同时保持对原有资源的占有。

4、循环等待，即存在一个等待队列：P1 占有 P2 的资源，P2 占有 P3 的资源，P3 占有 P1 的资源。这样就形成了一个等待环路。

注意：当上述四个条件都成立的时候，便形成死锁。当然，死锁的情况下如果打破上述任何一个条件，便可让死锁消失。

作业：

1. 梳理今天内容
2. 课上代码敲两遍