线 程

一口Linux

无线传感器网项目实战

公众号:一口Linux

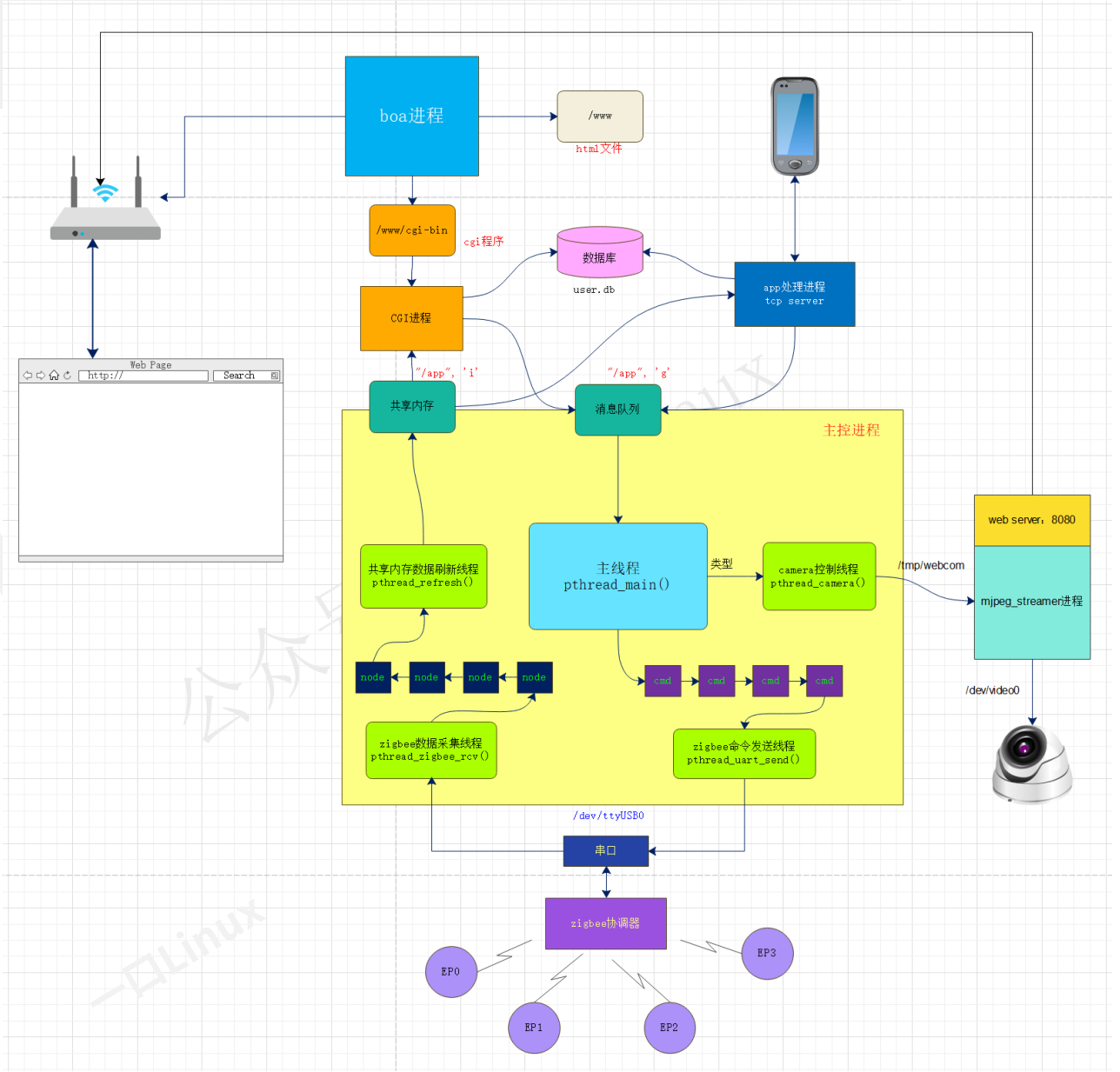
彭老师个人微信号

思考：

- 1. 多个进程有相同类型全局数组，要同步，如何高效操作?
- 2. 如果某个进程有动态链表，只要变化就要同步给多个进程，应该如何操作?
- 3. 对于一些要同时执行多个阻塞任务的进程要如何处理?
  【同时操作串口、摄像头、套接字、等待其他进程发送的消息队列等】

本项目使用场景

- 对于多个阻塞任务，必须使用多进程/多线程
- 所有的临界资源的操作都必须互斥：
  - 共享内存、消息队列、链表、串口、摄像头、数据库等
- 线程之间同步机制主要用的：
  - 互斥锁、条件变量

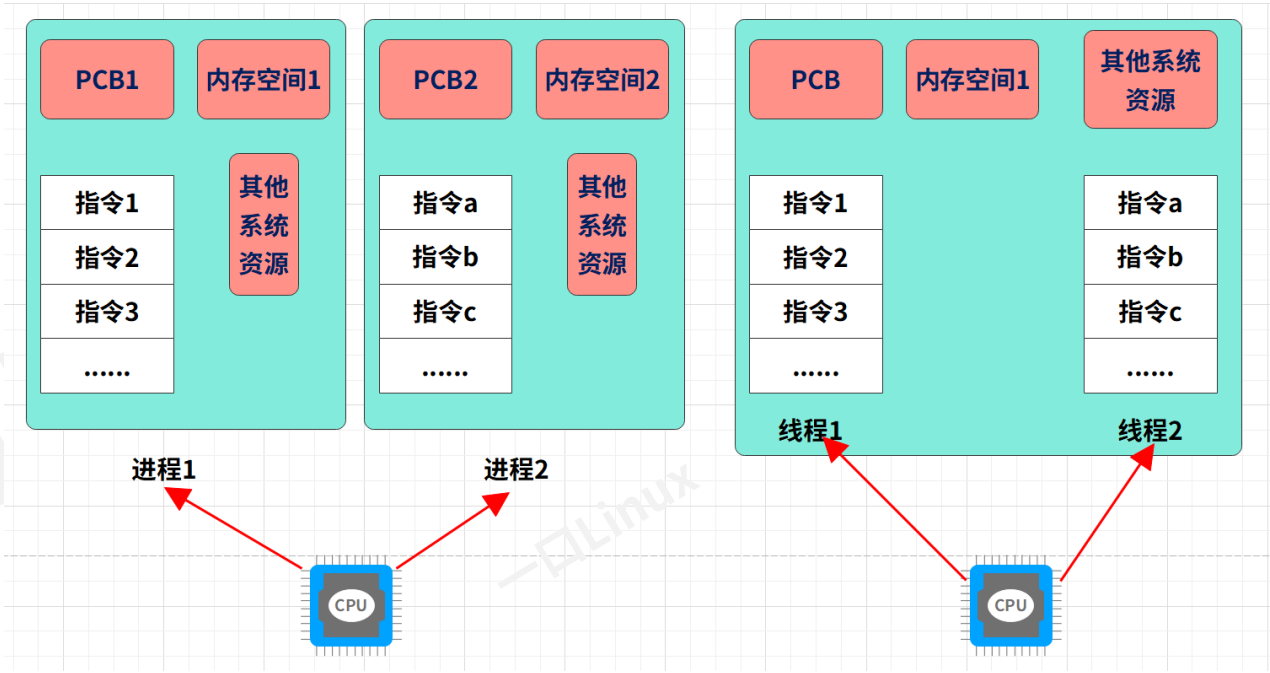


01

线程基础

线程基础

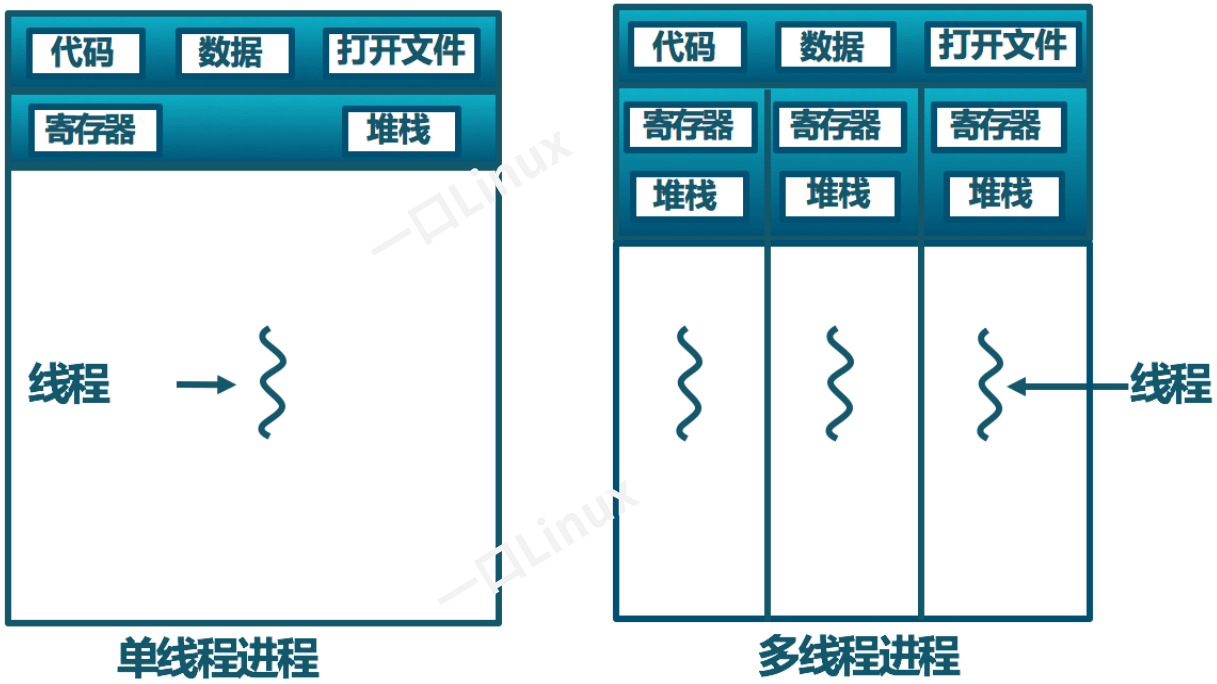
- 线程有时被称为轻量级进程(Lightweight Process，LWP），是程序执行流的最小单元。
- 线程是进程中的一个实体，是被系统**独立调度和分派的基本单位**，
- 线程自己不拥有系统资源，只拥有一点儿在运行中必不可少的资源，但它可与同属一个进程的其它线程**共享进程所拥有的全部资源**。



引入线程前，**进程**是**资源分配**的基本单位，也是**调度**的基本单位
引入线程后，**进程**是**资源分配**的基本单位，**线程**是**调度**的基本单位

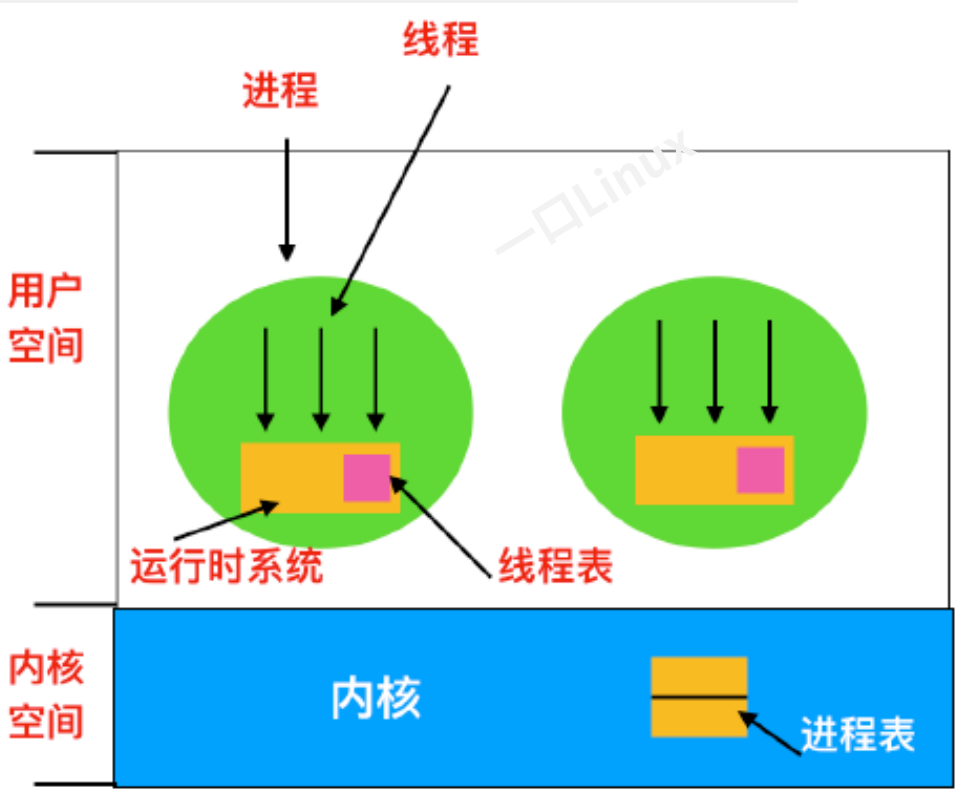
线程基础

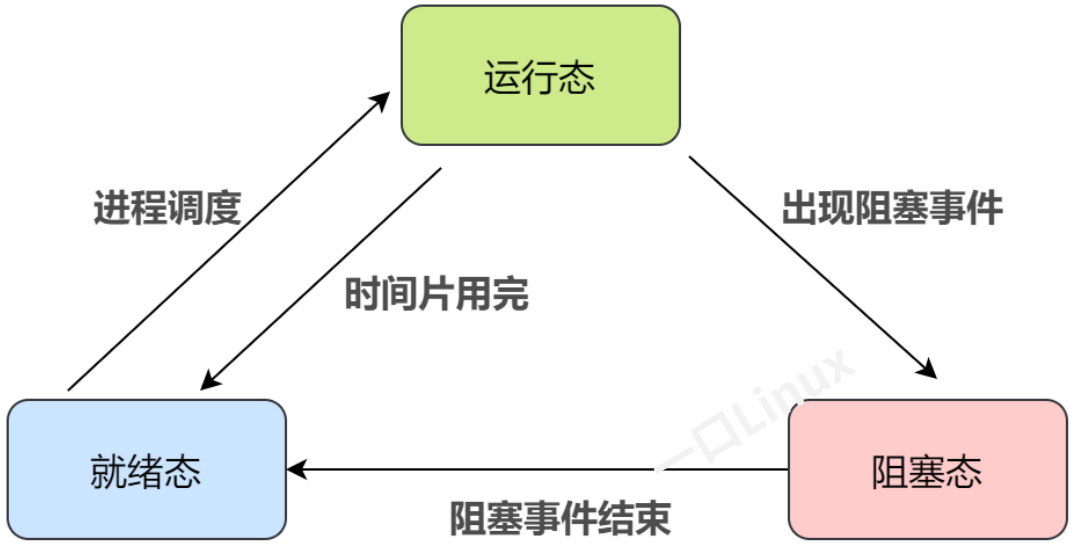
- 线程
  - 允许程序执行不止一个任务的机制
  - 并行执行
  - 受操作系统异步调度，是操作系统调度的最小单元，
  - 进程内的不同线程执行是同一程序的不同部分
- 主线程
  - 线程可由进程创建，操作系统在创建进程时会创建一个主线程，程序main()函数是主线程的入口。



线程基础

- 1. 由于进程的地址空间是私有的，因此在进程间上下文切换时，系统开销比较大，**在同一个进程中创建的线程共享该进程的地址空间，**
- 2.每一个程序都至少有一个线程，若程序只有一个线程，那就是程序本身。
- 3. 线程也有**就绪、阻塞和运行**三种基本状态。
- 4. 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**
- 5. 多线程通过第三方的线程库来实现 New POSIX Thread Library （NPTL）





多线程与多进程区别

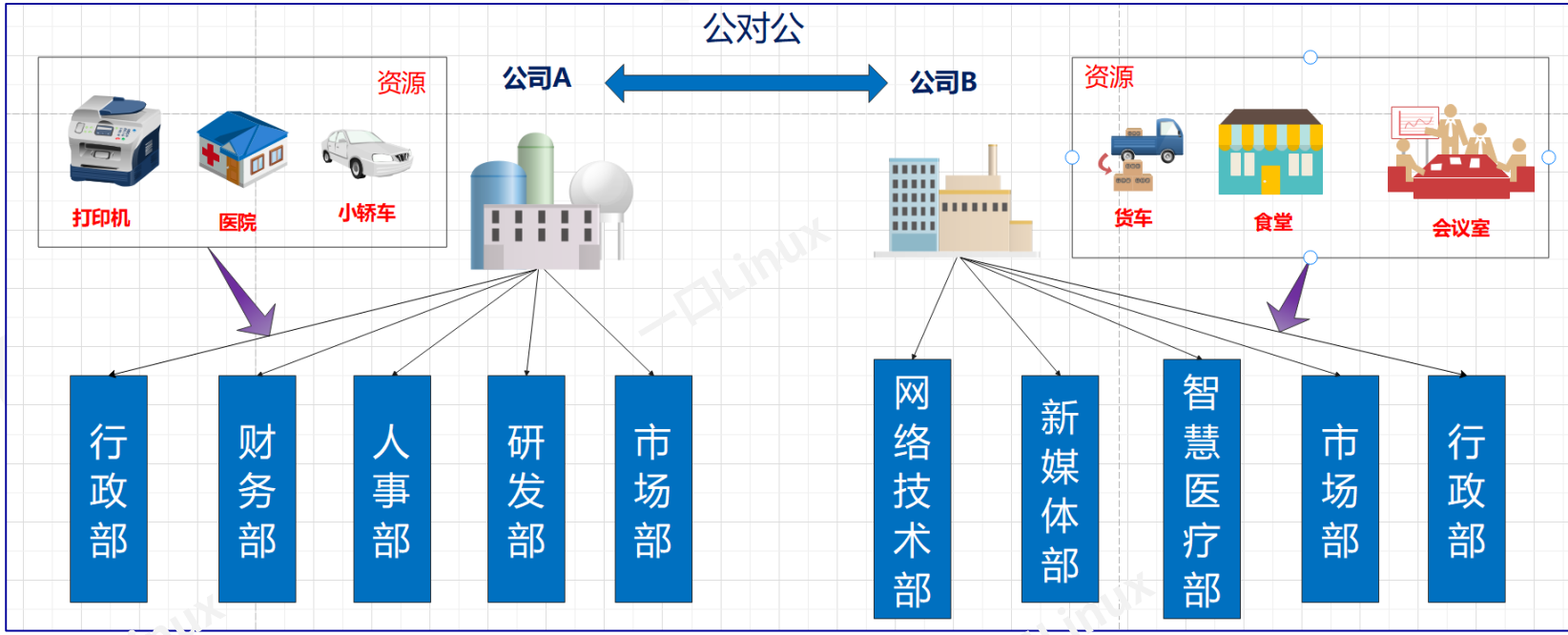
进程 ⟶ 公司

线程 ⟶ 部门

资源 ⟶ 全局变量、文件描述符…

线程间通信程 ⟶ 部门对部门

进程间通信程 ⟶ 公对公



公司A的市场部如果想用公司B的货车?

多线程优点

- 1. 经济实惠
  - 分配的资源少、线程切换比进程切换快、维护线程的开销小
- 2 资源共享
  - 线程共享它们所属进程的存储器和资源。
- 3 提高了响应速度
  - 允许程序在它的一部分被阻塞或正在执行一个冗长 的操作时持续运行
- 4 提高了多处理机体系结构的利用率
  - 在多CPU机器中，多线程提高了并发性

线程共享的资源

- 一个进程中的多个线程共享以下资源
  - 可执行的指令
  - 静态数据
  - 进程中打开的文件描述符
  - 信号处理函数
  - 当前工作目录
  - 用户ID
  - 用户组ID

线程私有的资源

- 每个线程私有的资源如下
  - 线程ID (TID)
  - PC(程序计数器)和相关寄存器
  - 堆栈
    - 局部变量
    - 返回地址
  - 错误号 (errno)
  - 信号掩码和优先级
  - 执行状态和属性

02

多线程编程
-线程创建

多线程编程

- NPTL线程库中提供了如下基本操作
  - 创建线程
  - 删除线程
  - 控制线程

- 线程间同步和互斥机制
  - 信号量
  - 互斥锁
  - 条件变量

多线程函数-pthread_create

所需头文件	#include <pthread.h>
函数原型	int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void * (* routine)(void *), void *arg)
函数参数	thread：创建的线程
	attr：指定线程的属性，NULL表示使用缺省属性
	routine：线程执行的函数
	arg: 传递给线程执行的函数的参数
函数返回值	成功：0
	出错：-1

PTHREAD_CREATE_DETACHED
分离线程没有被其他的线程所等待，自己运行结束了，线程也就终止了，马上释放系统资源

PTHREAD _CREATE_JOINABLE
线程的默认属性是非分离状态，这种情况下，原有的线程等待创建的线程结束。只有当pthread_join()函数返回时，创建的线程才算终止，才能释放自己占用的系统资源。

多线程函数-pthread_join

所需头文件	#include <pthread.h>
函数原型	int pthread_join(pthread_t thread, void **value_ptr)
函数参数	thread：要等待的线程
	value_ptr：指针*value_ptr指向线程返回的参数
函数返回值	成功：0
	出错：-1

线程执行完后如果不join的话，线程的资源会一直得不到释放而导致内存泄漏！

多线程函数-pthread_exit、pthread_cancel

所需头文件	#include <pthread.h>
函数原型	int pthread_exit(void *value_ptr)
函数参数	value_ptr：线程退出时返回的值
函数返回值	成功：0
	出错：-1

在不终止整个进程的情况下，单个线程可以有三种方式停止其工作流并退出

- 1.线程从其工作函数中返回，返回值是线程的退出码

所需头文件	#include <pthread.h>
函数原型	int pthread_cancel(pthread_t thread)
函数参数	thread：要取消的线程
函数返回值	成功：0
	出错：-1

- 2.线程可以被同一进程中的其他线程取消
- 3.线程自己调用pthread_exit

实例

编译：gcc test.c –o run -lpthread

- 1. 实现两个线程，每隔1秒，交替打印A、B
  - 物联网实训项目所有资料\3.课程代码讲解实例\3.linux系统编程\4.thread\thread_creat\thread.c
- 2. 测试共享资源：
  - 全局变量、打开的文件

思考

1. 如果我们不想A和B的行交替出现，而是必须打印三行A再三行B？

线程查看

```
peng@ubuntu:~/work/test/signal$ ps -efL
UID          PID    PPID     LWP  C NLWP STIME TTY          TIME CMD
root           1       0       1  0    1 05:18 ?        00:00:01 /sbin/init auto noprompt
root           2       0       2  0    1 05:18 ?        00:00:00 [kthreadd]
root           4       2       4  0    1 05:18 ?        00:00:00 [kworker/0:0H]
root           6       2       6  0    1 05:18 ?        00:00:00 [mm_percpu_wq]
root           7       2       7  0    1 05:18 ?        00:00:00 [ksoftirqd/0]
root           8       2       8  0    1 05:18 ?        00:00:00 [rcu_sched]
root        2221       1    2221  0    3 05:25 ?        00:00:00 /usr/lib/upower/upowerd
root        2221       1    2305  0    3 05:25 ?        00:00:00 /usr/lib/upower/upowerd
root        2221       1    2306  0    3 05:25 ?        00:00:00 /usr/lib/upower/upowerd
root        5302    3757    5302  0    3 07:07 pts/2    00:00:00 ./run
root        5302    3757    5303  0    3 07:07 pts/2    00:00:00 ./run
root        5302    3757    5304  0    3 07:07 pts/2    00:00:00 ./run
```

upowerd进程的线程

- ps –efL
  - UID：用户ID
  - PID：process id 进程id
  - PPID: parent process id 父进程id
  - LWP：表示这是个线程；要么是主线程(进程)，要么是线程
  - NLWP: num of light weight process 轻量级进程数量，即线程数量
  - STIME: start time 启动时间TIME: 占用的CPU总时间
  - TTY：该进程是在哪个终端运行的
  - CMD：进程的启动命令
- ps -ef f
  - 用树形显示进程和线程

03

多线程-互斥锁

线程与进程的同步互斥比较

- 多线程共享同一个进程的地址空间

- 优点：
  - 线程间很容易进行通信
  - 通过全局变量实现数据共享和交换

- 缺点：
  - 多个线程同时访问共享对象时需要引入同步和互斥机制

本项目只用到互斥锁和条件变量

互斥锁（Mutex）

- 互斥锁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，任何时刻最多只能有一个线程能访问该资源
- 使用规则:
  - 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

Mutex 函数-pthread_mutex_init

所需头文件	#include <pthread.h>
函数原型	int pthread_mutex_init(pthread_mutex_t *mutex, pthread_mutexattr_t *attr) // 初始化互斥锁
函数参数	mutex：互斥锁 attr：互斥锁属性 // NULL表示缺省属性
函数返回值	成功：0 出错：-1

Posix Mutex API

PTHREAD_MUTEX_TIMED_NP，这是缺省值，也就是普通锁。*当一个线程加锁以后，其余请求锁的线程将形成一个等待队列，并在解锁后按优先级获得锁*。这种锁策略保证了资源分配的公平性。

PTHREAD_MUTEX_RECURSIVE_NP，嵌套锁，*允许同一个线程对同一个锁成功获得多次，并通过多次unlock解锁*。如果是不同线程请求，则在加锁线程解锁时重新竞争。

PTHREAD_MUTEX_ERRORCHECK_NP，检错锁，如果*同一个线程请求同一个锁，则返回EDEADLK*，否则与PTHREAD_MUTEX_TIMED_NP类型动作相同。这样保证当不允许多次加锁时不出现最简单情况下的死锁。

PTHREAD_MUTEX_ADAPTIVE_NP，适应锁，动作最简单的锁类型，仅等待解锁后重新竞争。

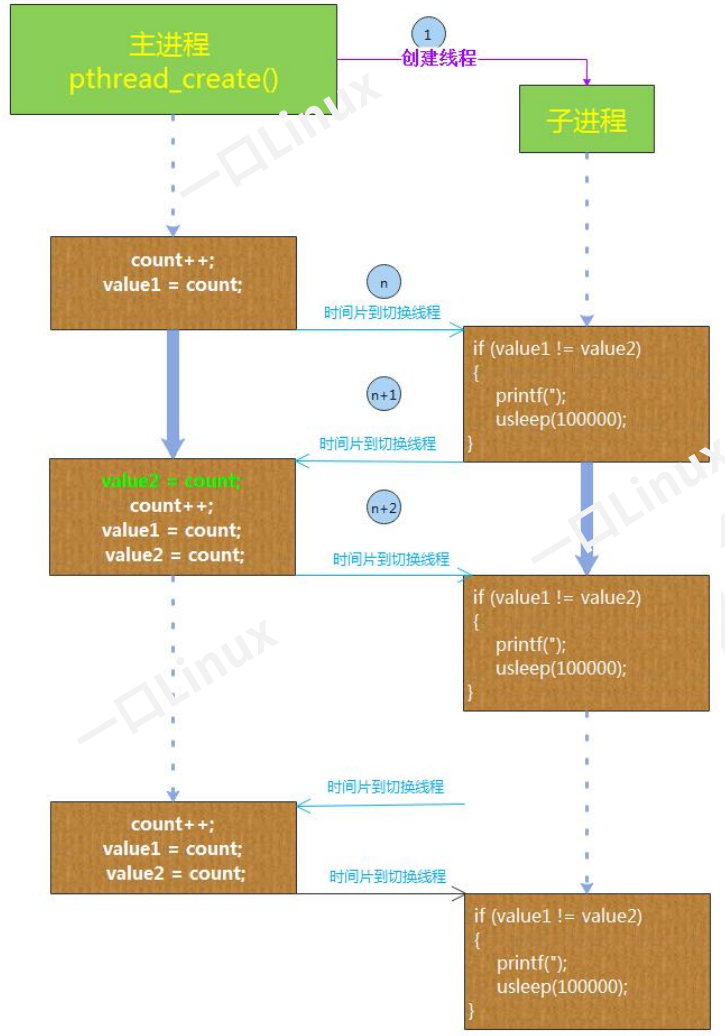
Mutex 函数-pthread_mutex_lock/unlock

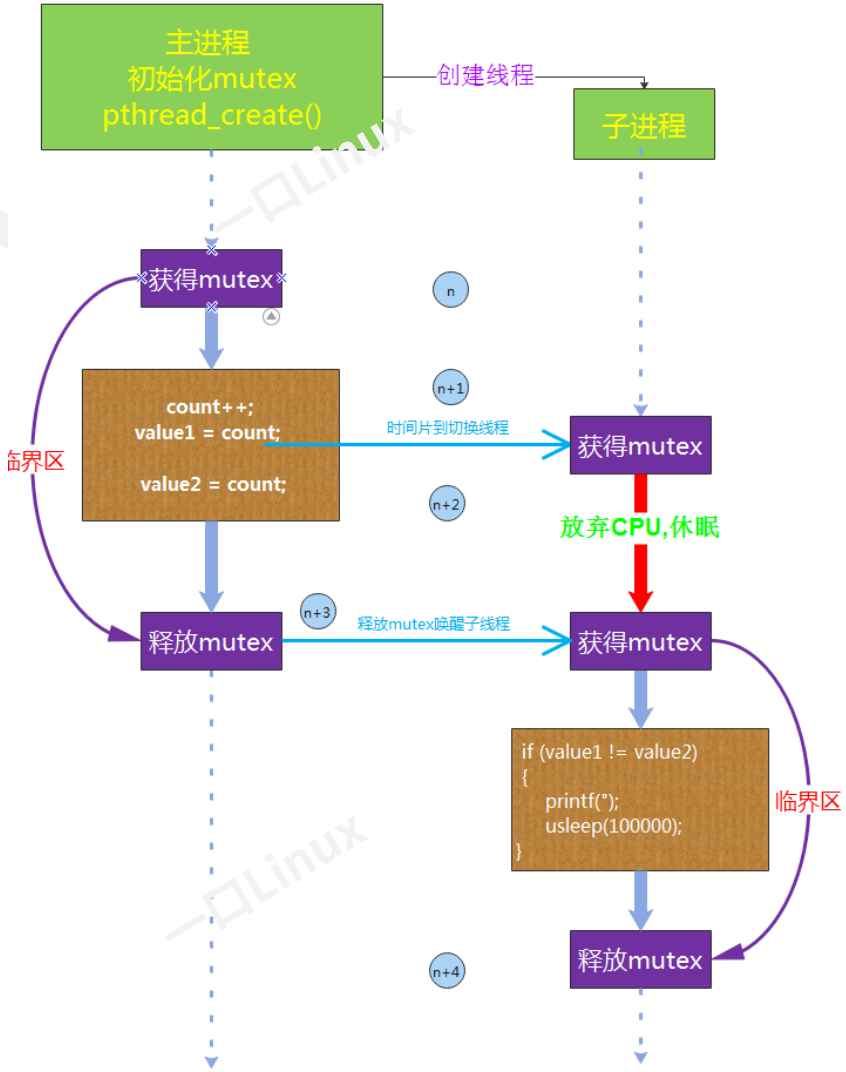
所需头文件	#include <pthread.h>
函数原型	int pthread_mutex_lock(pthread_mutex_t *mutex) // 申请互斥锁
函数参数	mutex：互斥锁
函数返回值	成功：0
	出错：-1

所需头文件	#include <pthread.h>
函数原型	int pthread_mutex_unlock(pthread_mutex_t *mutex) // 释放互斥锁
函数参数	mutex：互斥锁
函数返回值	成功：0
	出错：-1

实例

物联网实训项目所有资料\3.课程代码讲解实例\3.linux系统编程\4.thread\mutex\mutex.c





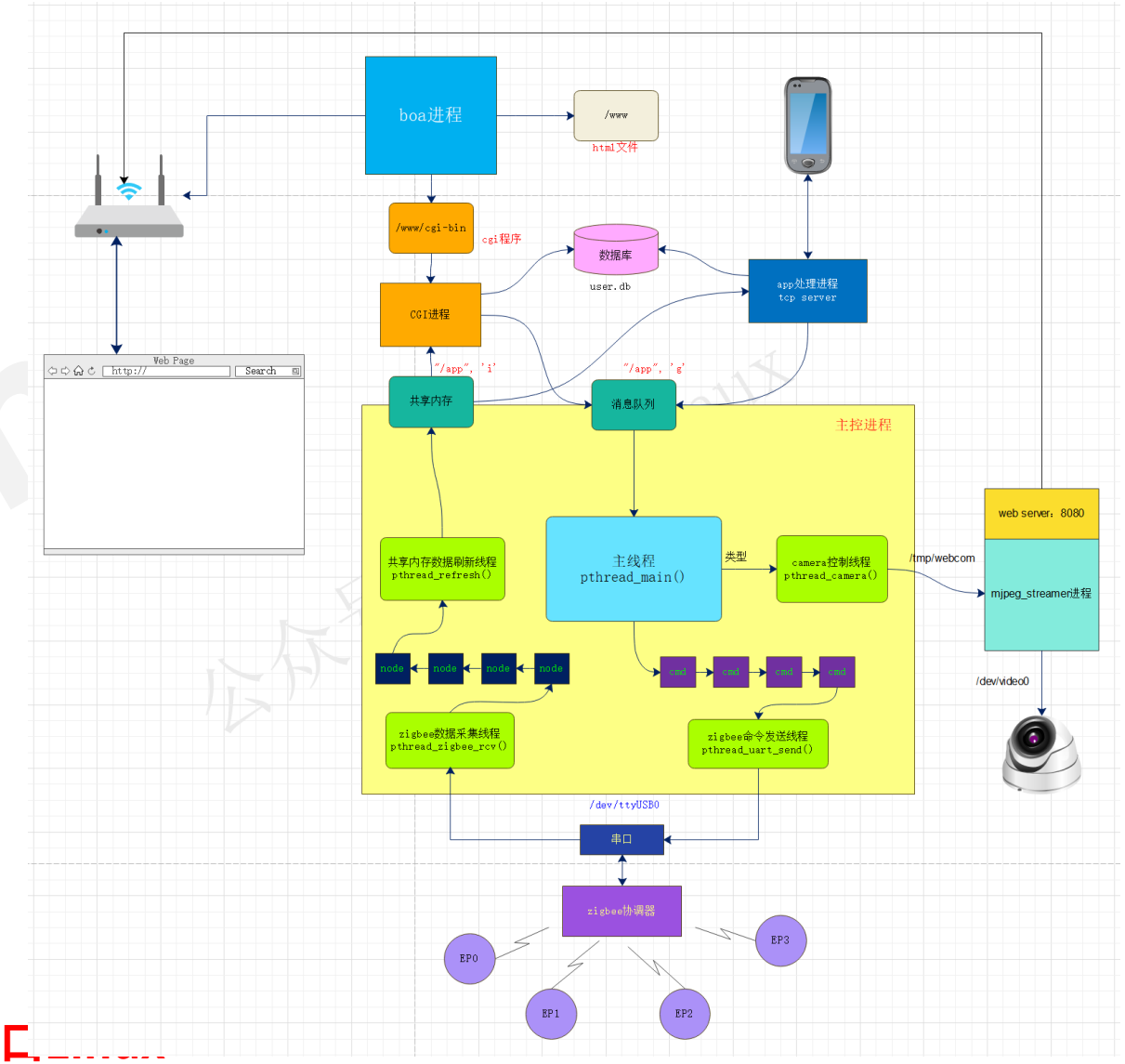
04

多线程-条件变量

本项目使用场景

- 主控进程的主线程通过阻塞方式从消息队列中提取出指令后需要发送给camera控制线程和zigbee命令发送线程，后面2者必须阻塞直到主线程发送数据

- 共享内存数据刷新线程也必须阻塞等待Zigbee数据采集线程通过串口读取到的数据



条件变量

- 条件变量是利用线程间共享的全局变量进行同步的一种机制。

- 主要包括两个动作：
  - 一个线程等待“条件变量的条件成立”而挂起；
  - 另一个线程使“条件成立”（给出条件成立信号）并唤醒挂起线程。

- 为了防止竞争，条件变量的使用总是和一个**互斥锁**结合在一起。对条件的测试是在互斥锁（互斥）的保护下进行的。

条件变量创建和注销

所需头文件	#include <pthread.h>
函数原型	int pthread_cond_init(pthread_cond_t *cond, pthread_condattr_t *cond_attr)
函数参数	cond：条件变量 attr ：通常为NULL。默认值是 PTHREAD_ PROCESS_PRIVATE，即此条件变量被同一进程内的各个线程使用。
函数返回值	成功：0
	出错：返回错误码

所需头文件	#include <pthread.h>
函数原型	int pthread_cond_destroy(pthread_cond_t *cond)
函数参数	cond：条件变量
函数返回值	成功：0
	出错：返回错误码

条件变量等待

所需头文件	#include <pthread.h>
函数原型	int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex)
函数参数	cond：条件变量 mutex ： 互斥锁
函数返回值	成功：0
	出错：返回错误码

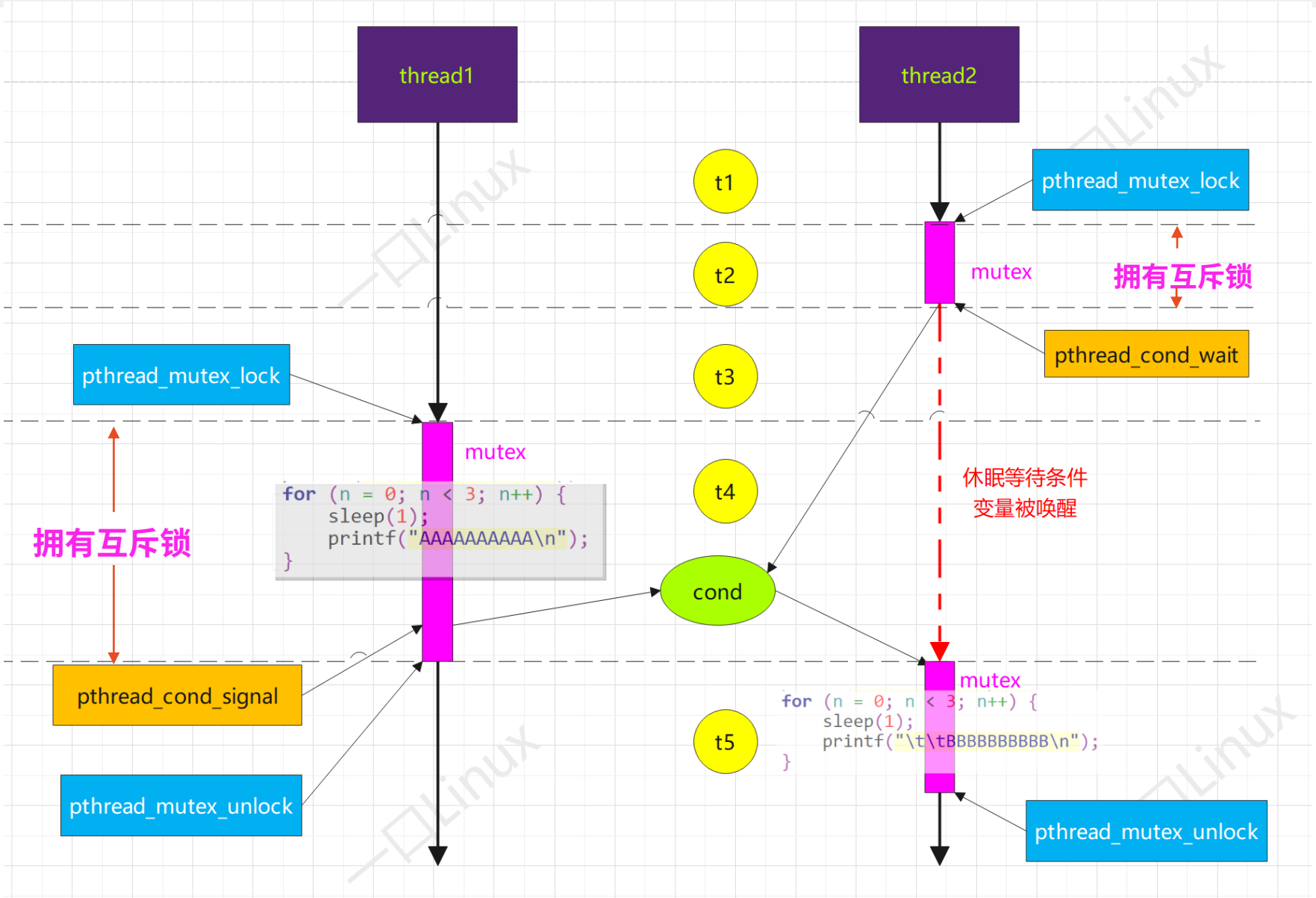
条件变量唤醒

所需头文件	#include <pthread.h>
函数原型	int pthread_cond_signal(pthread_cond_t *cptr) 按入队顺序唤醒其中一个 int pthread_cond_broadcast(pthread_cond_t *cptr) 唤醒所有等待线程
函数参数	cond：条件变量
函数返回值	成功：0
	出错：返回错误码

实例

- 修改
  - 物联网实训项目所有资料\3.课程代码讲解实例\3.linux系统编程\4.thread\thread_creat\thread.c
- 实现必须先打印3行A，再打印3行B?

代码执行流程


thread1
thread2
t1
t2
t3
t4
t5
pthread_mutex_lock
mutex
拥有互斥锁
pthread_cond_wait
休眠等待条件
变量被唤醒
cond
for (n = 0; n < 3; n++) {
sleep(1);
printf("AAAAAAAAAA\n");
}
pthread_cond_signal
pthread_mutex_unlock
for (n = 0; n < 3; n++) {
sleep(1);
printf("\t\tBBBBBBBBBB\n");
}

更多嵌入式Linux知识
请关注一口君的公众号：一口Linux