Linux多线程编程

Version 1.0

西安電子科技大學

本章重点

- 线程的基本概念
- 线程的创建和标识
- 线程的退出
- 线程资源的回收
- 线程之间的同步和互斥
- 线程属性

西安電子科技大學

线程的基本概念

1. 一个进程包含多个线程
2. 多个线程共享所属进程的资源：
   - 内存空间
   - 文件描述符
   - 安全权限
   - …
3. 为每个线程在共享的内存空间中都开辟了一个独立的栈空间（局部变量、函数形参等）
4. GunLibC中的pthread库实现了多线程API。
5. 因为pthread并非Linux系统的默认库，编译时必须加上-lpthread参数

线程的创建和标识

1. 线程标识：一个pthread_t类型的变量
2. 线程属性：一个pthread_attr_t类型的结构体
3. 创建线程：确定调用该线程函数的入口点

   int pthread_create(pthread_t* tid, pthread_attr_t *attr, void*(*start_routine)(void*), void *arg);
   - tid：输出参数，用于返回所创建现成的标识；
   - attr：用于设定线程属性，大多数情况下传NULL；
   - start_routine：用于指定线程主函数
   - arg：为线程主函数传递的参数

线程的退出

1. 三种退出方式：
   - 线程主函数执行完毕，自动退出；
   - 线程执行过程中调用了pthread_exit();(千万别调exit)
   - 其他线程利用pthread_cancel() 要求该线程强制退出
2. pthread_cancel(pthread_t tid)向目标线程发Cancel信号，但如何处理Cancel信号则由目标线程自己决定，或者忽略、或者立即终止、或者继续运行至Cancelation-point（取消点）
3. int pthread_setcancelstate(int state, int*oldstate);设置本线程对取消信号的反应。(CANCEL_ENABLE/DISABLE)
4. pthread_setcanceltype()函数设置取消类型(立即取消、或运行到下一个取消点)
5. pthread_setcancel()函数设置取消点；

线程的资源的回收

1. 一个线程退出后其部分资源并不能被OS回收，必须等到其他线程（一般是主线程）获得其退出状态并最终回收剩余资源。
2. pthread_join()可以用于将当前线程挂起来等待指定线程的结束。这个函数是一个线程阻塞的函数，调用它的函数将一直等待到被等待的线程结束为止，当函数返回时，被等待线程的资源就被收回。
3. int pthread_join(pthread_t tid, void **retval);
4. 线程也可以利用pthread_detach解除自己与所属进程之间的绑定。分离之后，线程结束，资源被全部回收。
5. 创建线程时也可以设定线程的分离属性。

线程编程基本函数

- **phtread_create()函数语法：**

所需头文件	#include <pthread.h>
函数原型	int pthread_create ((pthread_t *thread, pthread_attr_t *attr, void *(*start_routine)(void *), void *arg))
函数传入值	thread：线程标识符
	attr：线程属性设置（其具体设置参见 6.4.3 小节），通常取为 NULL
	start_routine：线程函数的起始地址，是一个以指向 void 的指针作为参数和返回值的函数指针
	arg：传递给 start_routine 的参数
函数返回值	成功：0
	出错：返回错误码

- **phtread_exit()函数语法：**

所需头文件	#include <pthread.h>
函数原型	void pthread_exit(void *retval)
函数传入值	retval：线程结束时的返回值，可由其他函数如 pthread_join()来获取

线程编程基本函数

- **phtread_join()函数语法：**

所需头文件	#include <pthread.h>
函数原型	int pthread_join ((pthread_t th, void **thread_retum))
函数传入值	th：等待线程的标识符
	thread_retum：用户定义的指针，用来存储被等待线程结束时的返回值（不为 NULL 时）
函数返回值	成功：0
	出错：返回错误码

- **phtread_cancel()函数语法：**

所需头文件	#include <pthread.h>
函数原型	int pthread_cancel((pthread_t th)
函数传入值	th：要取消的线程的标识符
函数返回值	成功：0
	出错：返回错误码

线程间的同步和互斥

互斥锁机制：

- 互斥锁初始化：pthread_mutex_init()
- 互斥锁上锁：pthread_mutex_lock()
- 互斥锁判断上锁：pthread_mutex_trylock()
- 互斥锁解锁：pthread_mutex_unlock()
- 消除互斥锁：pthread_mutex_destroy()

互斥锁可以分为快速互斥锁、递归互斥锁和检错互斥锁。这三种锁的区别主要在于其他未占有互斥锁的线程在希望得到互斥锁时是否需要阻塞等待。快速锁是指调用线程会阻塞直至拥有互斥锁的线程解锁为止。递归互斥锁能够成功地返回，并且增加调用线程在互斥上加锁的次数，而检错互斥锁则为快速互斥锁的非阻塞版本，它会立即返回并返回一个错误信息。默认属性为快速互斥锁。

线程间的同步和互斥

- **互斥锁线程控制**
  - pthread_mutex_init()函数语法：

所需头文件	#include <pthread.h>	
函数原型	int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *mutexattr)	
函数传入值	mutex：互斥锁	
函数传入值	Mutexattr	PTHREAD_MUTEX_INITIALIZER：创建快速互斥锁
		PTHREAD_RECURSIVE_MUTEX_INITIALIZER_NP：创建递归互斥锁
		PTHREAD_ERRORCHECK_MUTEX_INITIALIZER_NP：创建检错互斥锁
函数返回值	成功：0	
	出错：返回错误码	

线程间的同步和互斥

- **互斥锁线程控制**
  - pthread_mutex_lock()函数语法：

所需头文件	#include <pthread.h>
函数原型	int pthread_mutex_lock(pthread_mutex_t *mutex,) int pthread_mutex_trylock(pthread_mutex_t *mutex,) int pthread_mutex_unlock(pthread_mutex_t *mutex,) int pthread_mutex_destroy(pthread_mutex_t *mutex,)
函数传入值	mutex：互斥锁
函数返回值	成功：0
	出错：-1

线程间的同步和互斥

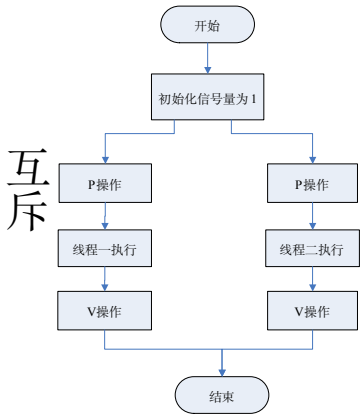
信号量机制：

- sem_init()用于创建一个信号量，并初始化它的值。
- sem_wait()和sem_trywait()都相当于P操作，在信号量大于零时它们都能将信号量的值减一，两者的区别在于若信号量小于零时，sem_wait()将会阻塞进程，而sem_trywait()则会立即返回。
- sem_post()相当于V操作，它将信号量的值加一同时发出信号来唤醒等待的进程。
- sem_getvalue()用于得到信号量的值。
- sem_destroy()用于删除信号量。

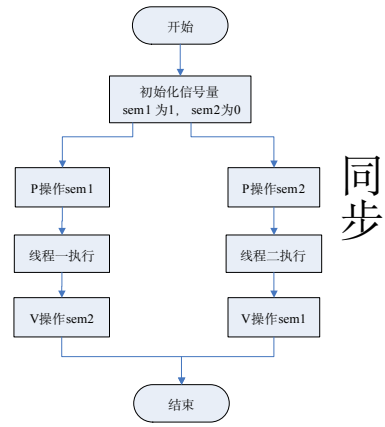
线程间的同步和互斥

- 信号量线程控制
  - PV原子操作主要用于进程或线程间的同步和互斥这两种典型情况。

互斥



同步



线程的属性

线程常用属性包括：

1. 绑定属性
2. 分离属性
3. 堆栈地址和大小
4. 运行优先级
5. 系统默认的属性为非绑定、非分离、缺省1M的堆栈以及与父进程同样级别的优先级。

如何设定线程的属性?

如何获得特定线程的属性?

线程的属性

- **pthread_create()函数的第二个参数（pthread_attr_t *attr）表示线程的属性。如果该值设为NULL，就是采用默认属性**
- **绑定属性**
  - 绑定属性就是指一个用户线程固定地分配给一个内核线程，因为CPU时间片的调度是面向内核线程（也就是轻量级进程）的，因此具有绑定属性的线程可以保证在需要的时候总有一个内核线程与之对应。而与之对应的非绑定属性就是指用户线程和内核线程的关系不是始终固定的，而是由系统来控制分配的。
- **分离属性**
  - 分离属性是用来决定一个线程以什么样的方式来终止自己。

线程的属性

- **pthread_attr_init()**
  - pthread_attr_init()函数对属性进行初始化
  - pthread_attr_init()函数语法：

所需头文件	#include <pthread.h>
函数原型	int pthread_attr_init(pthread_attr_t *attr)
函数传入值	attr：线程属性结构指针
函数返回值	成功：0
	出错：返回错误码

线程的属性

- **pthread_attr_setscope()**
  - pthread_attr_setscope()函数设置线程绑定属性
    pthread_attr_setscope()函数语法：

所需头文件	#include <pthread.h>	
函数原型	int pthread_attr_setscope(pthread_attr_t *attr, int scope)	
函数传入值	attr：线程属性结构指针	
	scope	PTHREAD_SCOPE_SYSTEM：绑定
		PTHREAD_SCOPE_PROCESS：非绑定
函数返回值	成功：0	
	出错：-1	

线程的属性

- **pthread_attr_setdetachstate()**
  - pthread_attr_setdetachstate()函数设置线程分离属性
  - pthread_attr_setdetachstate()函数语法：

所需头文件	#include <pthread.h>	
函数原型	int pthread_attr_setscope(pthread_attr_t *attr, int detachstate)	
函数传入值	attr：线程属性	
	detachstate	PTHREAD_CREATE_DETACHED：分离
		PTHREAD _CREATE_JOINABLE：非分离
函数返回值	成功：0	
	出错：返回错误码	

线程的属性

- **pthread_attr_setschedparam()**
  - pthread_attr_setschedparam()函数设置线程优先级
  - pthread_attr_setschedparam()函数语法：

所需头文件	#include <pthread.h>
函数原型	int pthread_attr_setschedparam (pthread_attr_t *attr, struct sched_param *param)
函数传入值	attr：线程属性结构指针
	param：线程优先级
函数返回值	成功：0
	出错：返回错误码

线程的属性

- **pthread_attr_getschedparam()**
  - pthread_attr_getschedparam()函数获得线程优先级
  - pthread_attr_getschedparam()函数语法：

所需头文件	#include <pthread.h>
函数原型	int pthread_attr_getschedparam (pthread_attr_t *attr, struct sched_param *param)
函数传入值	attr：线程属性结构指针
	param：线程优先级
函数返回值	成功：0
	出错：返回错误码

线程的属性

pthread_attr_init(), pthread_attr_destroy()

pthread_attr_setaffinity_np(),

pthread_attr_setdetachstate(),

pthread_attr_setguardsize(),

pthread_attr_setinheritsched(),

pthread_attr_setschedparam(),

pthread_attr_setschedpolicy(),

pthread_attr_setscope(), pthread_attr_setstack(),

pthread_attr_setstackaddr(), pthread_attr_setstacksize(),

pthread_getattr_np()

例子 src/ch6/pthread/pthd_attr/thread_attr.c