国嵌

www.enjoylinux.cn

LINUX
多线程程序设计

Contents

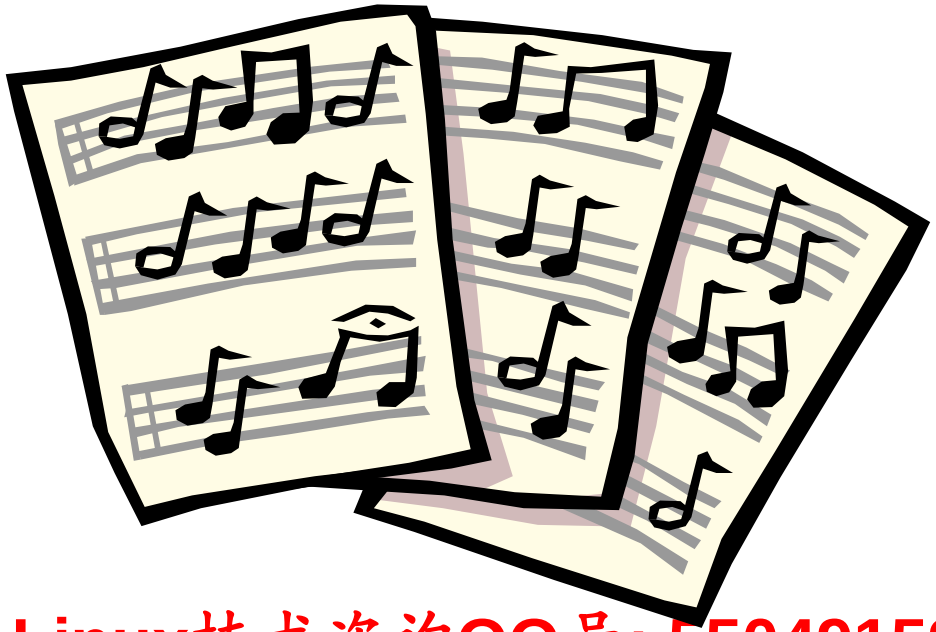
Contents

多线程

线程（thread）技术早在60年代就被提出，但真正应用多线程到操作系统中去，是在80年代中期，solaris是这方面的佼佼者。传统的Unix也支持线程的概念，但是在一个进程（process）中只允许有一个线程，这样多线程就意味着多进程。现在，多线程技术已经被许多操作系统所支持，包括Windows/NT、Linux。

多线程

- 为什么有了进程，还要引入线程呢？
- 使用多线程到底有哪些好处？



优点

使用多线程的理由之一是:

和进程相比，它是一种非常“节俭”的多任务操作方式。在Linux系统下，启动一个新的进程必须分配给它独立的地址空间，建立众多的数据表来维护它的代码段、堆栈段和数据段，这是一种"昂贵"的多任务工作方式。

优点

运行于一个进程中的多个线程，它们之间使用相同的地址空间，而且线程间彼此切换所需的时间也远远小于进程间切换所需要的时间。据统计，一个进程的开销大约是一个线程开销的30倍左右。

优点

使用多线程的理由之二是:

线程间方便的通信机制。对不同进程来说，它们具有独立的数据空间，要进行数据的传递只能通过进程间通信的方式进行，这种方式不仅费时，而且很不方便。线程则不然，由于同一进程下的线程之间共享数据空间，所以一个线程的数据可以直接为其它线程所用，这不仅快捷，而且方便。

优点

除了以上所说的优点外，多线程程序作为一种多任务、并发的工作方式，有如下优点：

- 使多CPU系统更加有效。操作系统会保证当线程数不大于CPU数目时，不同的线程运行于不同的CPU上。
- 改善程序结构。一个既长又复杂的进程可以考虑分为多个线程，成为几个独立或半独立的运行部分，这样的程序会利于理解和修改。

多线程

Linux系统下的多线程遵循POSIX线程接口，称为pthread。编写Linux下的多线程程序，需要使用头文件pthread.h，连接时需要使用库libpthread.a。

Contents

创建线程


```
#include <pthread.h>
```

```
int pthread_create(pthread_t * tidp,const pthread_attr_t *attr,
void *(*start_rtn)(void),void *arg)
```

tidp: 线程id

attr: 线程属性(通常为空)

start_rtn: 线程要执行的函数

arg: start_rtn的参数

编译

因为pthread的库不是linux系统的库，所以在进行编译的时候要加上

-lpthread

 # gcc filename -lpthread

实例分析（演示）

thread_create.c

thread_int.c

thread_string.c

thread_struct.c

thread_share.c

终止线程

如果进程中任何一个线程中调用exit或_exit，那么整个进程都会终止。线程的正常退出方式有：

(1) 线程从启动例程中返回

(2) 线程可以被另一个进程终止

(3) 线程自己调用pthread_exit函数

线程退出


```
#include <pthread.h>
        void pthread_exit(void * rval_ptr)
```

功能：终止调用线程

Rval_ptr:线程退出返回值的指针。

实例分析（演示）

thread_exit.c

线程等待


```
#include <pthread.h>
int pthread_join(pthread_t tid,void **rval_ptr)
```

功能：阻塞调用线程，直到指定的线程终止。

Tid :等待退出的线程id

Rval_ptr: 线程退出的返回值的指针

实例分析（演示）

thread_join.c

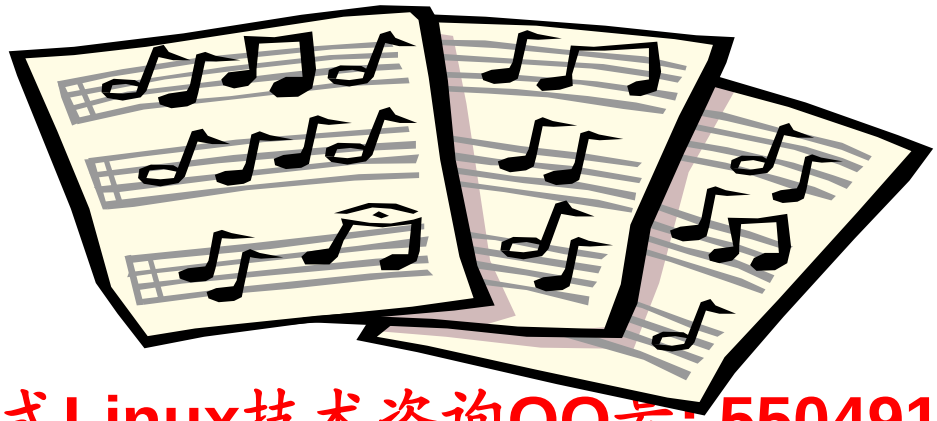
线程标识


```
#include <pthread.h>
```

```
pthread_t pthread_self(void)
```

功能:

获取调用线程的 thread identifier



实例分析（演示）

thread_id.c

清除

线程终止有两种情况：正常终止和非正常终止。线程主动调用pthread_exit或者从线程函数中return都将使线程正常退出，这是可预见的退出方式；非正常终止是线程在其他线程的干预下，或者由于自身运行出错（比如访问非法地址）而退出，这种退出方式是不可预见的。

清除

不论是可预见的线程终止还是异常终止，都会存在资源释放的问题，如何保证线程终止时能顺利的释放掉自己所占用的资源，是一个必须考虑解决的问题。

清除

从pthread_cleanup_push的调用点到pthread_cleanup_pop之间的程序段中的终止动作（包括调用pthread_exit()和异常终止，不包括return）都将执行pthread_cleanup_push()所指定的清理函数。

清除


```
#include <pthread.h>
void pthread_cleanup_push(void (*rtn)(void *),void *arg)
```

功能:

将清除函数压入清除栈

Rtn:清除函数

Arg:清除函数的参数

清除


```
#include <pthread.h>
```

void pthread_cleanup_pop(int execute)

功能:

将清除函数弹出清除栈

参数:

Execute执行到pthread_cleanup_pop()时是否在弹出清理函数的同时执行该函数，非0:执行; 0:不执行

实例分析（演示）

thread_clean.c

Contents

线程同步

进行多线程编程，因为无法知道哪个线程会在哪个时候对共享资源进行操作，因此让如何保护共享资源变得复杂，通过下面这些技术的使用，可以解决线程之间对资源的竞争:

1 互斥量Mutex

2 信号灯Semaphore

3 条件变量Conditions

互斥量

为什么需要互斥量:

```
Item * p =queue_list;
Queue_list=queue_list->next;
process_job(p);
free(p);
```

当线程1处理完Item *p=queue_list后，系统停止线程1的运行，改而运行线程2。线程2照样取出头节点，然后进行处理，最后释放了该节点。过了段时间，线程1重新得到运行。而这个时候，p所指向的节点已经被线程2释放掉，而线程1对此毫无知晓。他会接着运行process_job(p)。而这将导致无法预料的后果！

互斥量

对于这种情况，系统给我们提供了互斥量。线程在取出头节点前必须要等待互斥量，如果此时有其他线程已经获得该互斥量，那么该线程将会阻塞在这里。只有等到其他线程释放掉该互斥量后，该线程才有可能得到该互斥量。互斥量从本质上说就是一把锁，提供对共享资源的保护访问。

创建

在Linux中，互斥量使用类型pthread_mutex_t表示。在使用前，要对它进行初始化：

- 对于静态分配的互斥量，可以把它设置为默认的mutex对象PTHREAD_MUTEX_INITIALIZER

- 对于动态分配的互斥量，在申请内存(malloc)之后，通过pthread_mutex_init进行初始化，并且在释放内存(free)前需要调用pthread_mutex_destroy。

创建


```
#include <pthread.h>
```

- int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr)
- int pthread_mutex_destroy(pthread_mutex_t *mutex)

加锁

对共享资源的访问，要使用互斥量进行加锁，如果互斥量已经上了锁，调用线程会阻塞，直到互斥量被解锁。

- int pthread_mutex_lock(pthread_mutex_t *mutex)
- int pthread_mutex_trylock(pthread_mutex_t *mutex)

返回值：成功则返回0，出错则返回错误编号。

trylock是非阻塞调用模式，如果互斥量没被锁住，trylock函数将对互斥量加锁，并获得对共享资源的访问权限；如果互斥量被锁住了，trylock函数将不会阻塞等待而直接返回EBUSY，表示共享资源处于忙状态。

解锁

在操作完成后，必须给互斥量解锁，也就是前面所说的释放。这样其他等待该锁的线程才有机会获得该锁，否则其他线程将会永远阻塞。

int pthread_mutex_unlock(pthread_mutex_t *mutex)

互斥量PK信号量

- Mutex是一把钥匙，一个人拿了就可进入一个房间，出来的时候把钥匙交给队列的第一个。
- Semaphore是一件可以容纳N人的房间，如果人不满就可以进去，如果人满了，就要等待有人出来。对于N=1的情况，称为binary semaphore。
- Binary semaphore与Mutex的差异:

  1. mutex要由获得锁的线程来释放（谁获得，谁释放）。而semaphore可以由其它线程释放

  2. 初始状态可能不一样： mutex的初始值是1 ,而semaphore的初始值可能是0（或者为1）。

厕所理论

Mutex:

Is a key to a toilet. One person can have the key - occupy the toilet - at the time. When finished, the person gives (frees) the key to the next person in the queue.Officially: “Mutexes are typically used to serialise access to a section of re-entrant code that cannot be executed concurrently by more than one thread. A mutex object only allows one thread into a controlled section, forcing other threads which attempt to gain access to that section to wait until the first thread has exited from that section.”Ref: Symbian Developer Library(A mutex is really a semaphore with value 1.)

厕所理论

Semaphore:

Is the number of free identical toilet keys. Example, say we have four toilets with identical locks and keys. The semaphore count - the count of keys - is set to 4 at beginning (all four toilets are free), then the count value is decremented as people are coming in. If all toilets are full, ie. there are no free keys left, the semaphore count is 0. Now, when eq. one person leaves the toilet, semaphore is increased to 1 (one free key), and given to the next person in the queue.

厕所理论

Officially: “A semaphore restricts the number of simultaneous users of a shared resource up to a maximum number. Threads can request access to the resource (decrementing the semaphore), and can signal that they have finished using the resource (incrementing the semaphore).” Ref: Symbian Developer Library