文章编号: 1009－671X(2001)05－0028－03

Linux 下的多线程编程

杨传安, 王国夫, 张海勋
(哈尔滨工程大学 自动化学院, 黑龙江 哈尔滨　150001)

摘　　要: 介绍了 Linux 多线程库、几种线程间互斥和同步方法的实现, 及多线程程序的基本程序结构。①
关　键　词: Linux 线程; 死锁; 互斥; 同步
中图分类号: TP312　文献标识码: A

Multi-thread Programmg Under Linux

YANG Chuan-an, WANG Guo-fu, ZHANG Hai-xun
(Automation College, Harbin Engineering University, Harbin 150001, China)

Abstract: In this paper, LinuxThreads lib and realization of some methods of mutex and synchronization between multi-thread were introduced, and the basic architecture of the program with linuxthreads was summarized.

Key words: LinuxThreads; lockup; mutex; synchronization

0　引　言

线程概念　LinuxThreads 下符合 IEEE POSIX 1003.1c“pthread”接口标准的多线程函数库, POSIX 线程(pthreads)是轻量级进程中一种具有较好移植性的实现。

一个线程就是程序中的单个顺序控制流。利用多线程进行程序设计, 就是将一个程序(进程)的任务划分为执行的多个部分(线程), 每一个线程为一个顺序的单控制流, 而所有线程都是并发执行的, 这样, 多线程程序就可以实现并行计算, 高效利用多处理器, 从而提高程序的性能, 如 I/O 效率、计算速度及反应速度等。

线程、进程比较　传统 UNIX 下, 一个进程让另一个实体处理某个事务是用 fork()系统调用派生子进程的方法来实现的, 子进程执行父进程派发的任务。但用 fork, 它产生的子进程几乎是当前 进程的完全拷贝, 而且在父子进程之间、子进程之间传递信息不需要用到进程间通讯技术(IPC).

使用多线程的编程可以大大增进效率: 和任务关联的各个线程运行在同一地址空间上、共享全局变量和内存, 这样线程之间共享数据很方便, 相互间通信也容易。

当然在同一进程环境下的多个线程之间共享全局数据为这些多个并发的线程通信提供了方便, 但必须对访问共享资源的代码段实现互斥。因为程序中的多个线程并发执行, 不能对它们的相对开始和完成时间, 以及执行的顺序作任何假设, Linux 内核调度程序完成线程的调度。程序员要想影响线程的执行来保证相互作用的任务要求得到满足, 就需要在程序中提供合理的同步机制。此时就必须考虑由于错误使用同步互斥而导致多个线程死锁的问题, 本文随后提供的几种同步机制都可以避免死锁。

1　LinuxPthread 主要库函数

当前大多 Linux 的发行版本中 Pthread 库是 libpthread－0.8.so, 头文件是〈pthread.h〉。这些函数提供线程的创建、结束、同步和互斥等。

①　收稿日期: 2000－12－28
作者简介: 杨传安(1976－), 男, 湖北云梦人, 哈尔滨工程大学自动化学院硕士研究生, 主要研究方向: 计算机控制。

1.1　线程的创建和终止

pthread_create (pthread_t * thread, const pthread_attr_t *attr, void * (start_routine)(void *), void *arg);

每个进程创建时, 系统同时创建一个核心主线程。当要创建新的线程去完成并行任务, 可调用函数 pthread_create, 新线程产生后执行 start_routine, 调用者可通过 arg 给 routine 过程传递参数。

prthread_exit(void *retval);

一个线程可以隐式的退出, 也可以显示的调用 pthread_exit 函数退出。

1.2　线程控制调用

pthread_self(void);

每个线程都有自己的线程 ID, 以便在进程内区分。线程 ID 在 pthread_create 调用时回返给创建线程的调用者; 一个线程也可以在创建后使用 pthread_self()调用获取自己的线程 ID。

pthread_cance(pthread_t thread);

在当前线程中调用此函数, 可以取消由参数 thread 指定的线程。

pthread_join (pthread_t thread, void **thread_return);

这个函数的作用是等待一个线程的结束。调用 pthread_join 的线程将被挂起直到线程 ID 为参数 thread 指定的线程终止。

1.3　线程之间的互斥

pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);

这个函数初始化互斥体变量 mutex。

pthread_mutex_lock(pthread_mutex_t *mutex);

这个函数用来给一个互斥体变量上锁, 如果参数 mutex 指定的互斥体已经被锁住, 则调用线程将被阻塞, 直到拥有 mutex 的线程对 mutex 解锁为止。

pthread_mutex_unlock (pthread_mutex_t * mutex);

这个函数用来对参数 mutex 指定的互斥体解锁。

pthread_mutex_destroy (pthread_mutex_t *mutex);

这个函数用来释放为互斥体分配的资源。

1.4　线程之间的同步

pthread_cond_init(pthread_cond_t * cond, const pthread_condattr_t *attr);

这个函数按参数 attr 指定的属性初始化一个条件变量 cond。

pthread_cond_wait (pthread_cond_t * cond, pthread_mutex_t *mutex);

这个函数的作用是等待一个事件(条件变量)的发生, 发出调用的当前线程自动阻塞, 直到相应的条件变量被置上。等待状态下的线程不占用 CPU 时间。

pthread_cond_broadcast (pthread_cond_t * cond);

这个函数用来对所有等待条件变量 cond 的线程解除阻塞。

pthread_cond_signal(pthread_cond_t *cond);

这个函数是解除一个等待参数 cond 指定的条件变量的线程的阻塞状态。注意: 若有多个挂起的线程等待 cond, 一次调用只唤起一个线程, 被唤起的线程不确定。

2　多线程编程中的互斥和同步控制

当两个线程各自占有一些公共资源, 在没有释放自占有的共享资源的前提下, 又请求别的线程占有的共享资源, 形成无休止的相互等待状态, 在无外力的作用下, 这两个线程永远不能前进。这种僵持的局面就是死锁, 如图 2 示。多线程编程必须避免死锁, Linux Threads 提供的互斥和同步机制可以达到这样的目的。

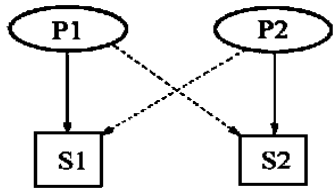

图 1　死锁

2.1　线程互斥

互斥操作, 就是对某段代码或某个变量修改的时候只能有一个线程在执行, 其它线程不能同时进入该段代码或同时修改变量。Linux 下可以通过 pthread_mutex_t 定义互斥体机制完成多线程的互斥操作。该机制的作用是对某个需要互斥的部分, 在进入时先得到互斥体, 如果没有得到互斥体, 表明互斥部分被其它线程拥有, 此时欲获取互斥体的线程阻塞, 直到拥有该互斥体的线程完成互斥部分的操作为止。一个互斥体不可能被两

个线程同时上锁。

下面的代码是对一个共享的全局变量 x 用互斥体变量 mutex 进行互斥保护。

```
int x;//进程中的全局变量
pthread_mutex_t mutex;
pthread_mutex_init( &mutex, NULL);//按缺省的属性初始化互斥体变量 mutex
pthread_mutex_lock( &mutex);//给互斥体变量加锁
/*对变量 x 的操作*/
phtread_mutex_unlock( &mutex);//给互斥体变量解除锁
```

2.2 线程同步

同步就是线程等待某个事件的发生。只有当等待的事件发生线程才继续执行，否则线程挂起并放弃处理器。当多个线程协作时，相互作用的任务必须在一定的条件下同步。Linux 下的 C 语言编程有多种线程同步机制，本文将以多种同步方法实现典型的生产/消费程序。

2.2.1 用互斥体变量

要使用互斥体变量完成生产者线程、消费者线程的同步，可以定义一个全局的开关变量。互斥体变量保护该开关量，通过开关量的约束实现同步。

2.2.2 互斥和条件变量结合

使用互斥体有一个缺点，它只有两种状态：锁定和非锁定。POSIX 的条件变量允许线程阻塞以等待另一个线程的信号。当接收到信号时，阻塞的线程将会被唤起，并试图获得相关的互斥体锁(例子略)。

2.2.3 利用信号量

信号量是在头文件 semaphore.h 中定义的。信号量定义完成了互斥体和条件变量的封装，按照多线程程序设计中访问控制机制，控制对资源的同步访问，提供程序设计人员更方便的调用接口。

```
sem_init(sem_t *sem, int pshard, unsigned int val);
```

这个函数初始化一个信号量 sem 的值为 val；参数 pshared 是共享属性控制，表明是否在进程间共享。

```
sem_wait(sem_t *sem);
```

调用该函数时，若 sem 为无状态，调用线程阻塞，等待信号量 sem 值增加(post)成为有信号状态；若 sem 为有状态，调用线程顺序执行，但信号量的值减一。

```
sem_post(sem_t *sem);
```

调用该函数，信号量 sem 的值增加，可以从无信号状态变为有信号状态。

不要将这里 POSIX1003.1b 的 Semaphores 同 System V 的 Semaphores 混淆。Sytem V 的信号灯是一种 IPC 机制，此处的信号量是 Pthread 的一种同步机制，相当于线程间共享资源计数器，LinuxThreads 库提供这些函数(例子略)。

上述的三种方法均可实现线程间同步，笔者在红旗 Linux1.0 上全部编译通过(gcc -g -O -D_REENTRANT test test.c -l/lib/libpthread-0.8.so)，读者可以根据自己的实际需要修改、扩充。

3 总 结

Linux 下的多线程编程可以在单进程中实现多任务，而不象传统的 UINX 以加大系统负荷的代价通过复制子进程完成多任务，因此提高了系统的效率。

用 LinuxThreads 库的基本程序结构如下。

1)以单进程开始程序运行；

2)定义互斥体变量 pthread_mutex_t lock，并初始化锁 pthread_mutex_init(&lock, NULL)；

3)产生线程 pthread_create(&thread, NULL, f, &arg)，并行代码 f 接受参数 arg 开始运行；

4)运行并行程序代码，在访问共享代码前后调用 pthread_mutex_lock(&mutex), pthread_mutex_unlock(&mutex)加锁、解锁；

5)单进程主体中调用 pthread_join(thread, &retVal)，完成清除处理工作；

6)用-D_REENTRANT 编译，包含头文件 pthread.h，链接时链入 pthread 库，在作者安装的红旗 Linux1.0 中为 libpthread.-0.8.so.

参 考 文 献

[1] K Wall. GNU/Linux 编程指南[M]. 北京：清华大学出版社，2000.

[2] 周巍松. Linux 系统分析与高级编程技术[M]. 北京：机械工业出版社，1999.

[3] 唐靖彪. UNIX 平台下 C 语言高级编程指南[M]. 北京：北京希望电子出版社，2000.

[4] A Kay, Robbins. 实用 UNIX 编程[M]. 北京：机械工业出版社，1999.