线程变量共享与锁机制

线程相比于进程，创建时由于只需要创建独立的控制流即可，所以在创建开销上要远远小于进程。在若本身某个程序完成工作需要分成多个独立且互有联系的部分去完成的话，创建多线程显然要比多个独立的进程更有效。

当线程创建完成之后，每个进程都会分配独立的tid作为每个线程唯一的标识。为了了解线程就必须先了解线程的共享模型具体是什么样子的。

对比进程，我们知道某个进程通过fork函数创建的两个进程共享所有打开的文件，将共享对象映射到各自的地址空间。由于进程之间互相不能访问对方为地址空间，这就导致进程间通信开销很大（比如建立管道）。然而线程确是运行在某一个进程空间内的多个控制流，最大程度上可以共享变量。

通过下例说明线程共享模型：

#include <stdio.h>

#include <pthread.h>

#define N 2

void \*thread(void\* vargp);

char \*\*ptr;

int main()

{

int i;

pthread\_t tid;

char \*msgs[N] = {

"hello from foo",

"hello from bar"

};

ptr = msgs;

for(i = 0; i < N; ++i)

pthread\_create(&tid, NULL, thread, (void\*)i);

pthread\_exit(NULL);

}

void \*thread(void \*vargp)

{

int myid = (int)vargp;

static int cnt = 0;

printf("[%d]: %s (cnt = %d)\n", myid, ptr[myid], ++cnt);

return NULL;

}

上例包含一个主线程和及其创建的两个对等线程。在程序运行至pthread\_create语句之前和普通的单线程程序没有任何区别，主线程流可以访问在此之前所有定义的变量。创建线程时，i作为参数传递给每个线程，每个线程完成的工作是将字符串输出。

那么那些变量是共享变量呢？首先自然是定义在main函数之外的全局变量，任何线程都可以对其引用。其次是在线程中定义的静态变量，程序中只包含一份存储，两个对等线程共享此变量。至于变量myid，两个线程定义在各自独立的栈中，互相不能访问和共享。所以简单来说多线程模型下，各个线程拥有其独立的栈空间，各线程在栈中定义的变量不共享，但是全局变量、静态变量、指针传递的变量都属于。

但凡涉及变量共享，就必须考虑对共享变量的操作能否得到正确的结果。这里两个对等线程共享静态变量cnt，并且都会改变cnt的值。在现代计算机并发模型中，对两个线程调度并分配处理器的时机是无法准确预测的，所以得到的结果也是无法准确预知的，当我们多次调用上面这个例子的时候会发现最终输出的cnt的值并不是固定的。

这里我们忽略对原因的讨论，因为这会涉及机器指令及CPU执行指令流程的知识。但是我们清楚，这里会改变cnt值的语句就是“++cnt”，我们也将这条语句成为临界区（当然，如果能写出精准标出及其指令，对于临界区定义会更准确）。我们就是要通过锁机制来控制对临界区的访问，以保证最后的结果是我们所预期的值。

对于临界区互斥访问最有名的就是锁机制，我们在操作系统中称其为PV操作，我们设置锁mutex的初值为1（表示一次只有一个线程可以访问）。每个线程进行访问时候首先进行P操作，P操作首先会就将mutex的值减1，如果此时mutex值小于0，表示此时临界区已经被访问，将会把此线程阻塞掉加入等待队列。其中得到临界区访问资格的线程完成对临界区的操作后会执行V操作，如果mutex值小于0则说明等待队列中依然有等待访问临界区的线程，唤醒一个等待线程并且对mutex值加1。

需要注意的是，相比于我们之前加1减1的操作，这里锁中P的加1操作，从加载到运算再到写会都不可以被中断，也就是在执行PV操作时候，其步骤是原子操作，要忽略时钟中断。既然能够忽略时钟中断，自然是系统调用提供给我们的工具。

锁被定义在头文件#include <semaphore.h>中，三个基本操作分别是：

int sem\_init(sem\_t \*sem, 0, unsigned int value);

int sem\_wait(sem\_t \*s); //P操作

int sem\_post(sem\_t \*s); //V操作

所以对于之前那个例子只需要就将锁加在临界区上下就可以了。

sem\_init(&mutex, 0 , 1);

sem\_wait(&mutex);

++cnt;

sem\_post(&mutex);

这样就能保证在多线程数据共享的模型下保证多个进程对于共享数据的正确操作，本质而言是通过锁操作的原子性加上等待队列来实现的。