文章编号: 1006-2475(2002)07-0007-05

现代操作系统中的多线程技术及其应用

周炎涛, 李立明

(湖南省计算技术研究所, 湖南 长沙 410012, China)

摘要: 多处理器计算机系统的出现、并发多进程并行通信和同步上的困难, 需要多线程来解决。多线程应用在适当的环境中可以大大提高程序的性能。本文对现代操作系统中的多线程技术和线程的私有数据进行了讨论, 并以例程说明了多线程编程及其应用。

关键词: 进程; 线程; 并发; 操作系统

中图分类号: TP316 文献标识码: A

Multithread Techniques and Their Application in Contemporary Operating System

ZHOU Yan-tao, LI Li-ming

(Hunan Computing Technique Institute, Changsha 410012, China)

Abstract: As the establishment of multiprocessor computer, the difficulties of parallel communication and synchronization of concurrent multiprocess, multithread is needed to solve corresponding problems and multithread application can greatly improve the properties of programs in suitable environment. This article discusses multithread techniques and thread-specific data, the examples are showed to introduce multithread programing and their application in contemporary operating system.

Keywords: process; thread; concurrent; operating system

0 引 言

在传统的操作系统中, 进程既是资源分配单位又是程序执行单位, 所谓的并发运行是通过多进程实现的(即让不同进程去执行可以并行执行的程序)。此时进程同时描述资源的占用、和处理机的执行情况[1]。

系统中创建新进程使用系统调用 fork(), 当一个进程需要另一个实体执行某件事情时, 该进程派生(fork())一个子进程, 让子进程进行处理。系统中新的进程与旧的进程使用同一代码段, 而对于数据段和堆栈段, 系统则复制一份给子进程(是逻辑上并非物理上的)。子进程一旦运行, 虽然它继承了父进程的一切数据, 但实际数据也分离、相互之间不再有影响了[2]。因此使用 fork()时: ①子进程是父进程的一个拷贝, 在有大量 fork()子进程时会强加某种形式的额外内核开销, 从而降低系统性能; ②子进程独立后, 相互间需要借助进程间通讯 IPC 来传递消息。

对于有多个执行活动的进程, 现代操作系统中用线程来描述: 一个线程是在同一进程内部按步骤执行的一个明确的序列; 多线程任务中, 一个进程可以有多个线程并发地前进, 线程之间共享内存并相互通信。

与 fork()相比, 使用线程带来的开销小、内核无需单独复制进程的内存空间或文件描述符等, 这样就节省了 CPU 时间、解决了内存不足等问题, 使得线程创建比进程创建好得多。因此只要在程序中有意义, 就应该创建线程。

1 线程的描述

线程类似于进程, 由内核按时间分片管理。在单处理器上执行多线程的进程内部, 处理器在线程间切换执行, 内核使用时间片来模拟线程的并发执行(方式和进程相同); 而在共享内存多处理器上, 每一个线

收稿日期: 2002-03-17

作者简介: 周炎涛(1963-), 男, 湖南汉寿人, 湖南省计算技术研究所高级工程师、副教授, 硕士, 研究方向: 计算机操作系统, 计算机网络与通信。

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

程分别在不同的处理器上进行,线程实现真正的并发执行。

一个线程有自己的栈、寄存器集、程序计数器、线程说明数据、局部线程变量、线程说明信号掩码以及状态信息,但同一进程内的所有线程共用同一地址空间、通用的信号处理机制、虚存、数据与 I/O。进程内所有线程都能访问同一全局变量(采用同步机制)是线程间通信最简单的方法,由于不同进程的地址空间不同,进程间采用这种通信需要操作系统的干预,要切换到核心态。而线程间采用这种通信效率要高。

在系统实现层次上,有两种基本方法或模式创建并管理线程:前者为用户级模式,此时线程运行在操作系统上层并对操作系统透明、操作系统维护一个运行系统以管理线程活动,用户级的线程主要为那些在单 CPU 上运行共享同一进程内部资源而设计的;后者是内核级模型,此时较少依赖系统,对每个指定的线程函数,一个提供直接线程支持的核心包含了系统级代码,内核级线程支持多处理器上多线程的并行运行[3]。

多线程的益处在于:①提高应用程序响应;②使多处理器效率更高;③改善程序结构;④占用较少的系统资源;⑤把线程和远程过程调用 RPC 结合起来;⑥提高了系统性能等。

以 Microsoft Word 为例:在进行文件打印时,就默认选择了后台打印方式,打印处理会在继续编辑文档的同时异步进行。此时 Word 创建了独立的线程来进行打印处理,并将打印线程的优先级设定低于处理用户输入的线程,这在 Windows 2000 与 Windows NT 4.0 中能够得到,它们真正采用抢占式多任务,支持多线程应用。此外,IIS、组件服务和 SQL Server 全部都是多线程的。在实际应用中包含多个独立任务的问题适合用多线程方法解决。

2 多线程编程

当用 fork()创建另一个新进程时,新进程是子进程,原始进程是父进程。这样就创建了非常有用的进程层次关系。而在线程中不存在这样的层次关系,此时主线程可以创建一个新线程,新线程可以创建另一个新线程,它们在等同的层次上。一个程序启动时,会创建一个称作初始线程的单个线程,其他的线程由 pthread_create()函数创建。与进程中 waitpid()函数等待所有子进程终止不同,如果要等待一个线程终止,必须将线程的标识 tid 传送给 pthread_join()系统调用。

多线程要比普通程序设计复杂得多。由于任一时刻都可能有多个线程同时执行,所以,许多的变量、数据都可能会被其他线程所修改。这就是多线程程序中最关键的线程间的同步控制问题。

多线程编程非常灵活,涉及的系统调用多,内容也较复杂。用户级多线程是通过线程库 libthread 来实现的,线程库支持信号,为运行的程序安排队列、并负责同时操作多任务。以下是线程编程简单事例:

```
#include< pthread.h>
#include< stdlib.h>
#include< unistd.h>
#include< stdio.h>
int myvariable;
pthread_mutex_t mymutex= PTHREAD_MUTEX_INITIALIZER;
        void *thread_function(void *arg){
        int i, j;
        for ( i= 0; i< 20; i++ ){
          pthread_mutex_lock(&mymutex);
          j= myvariable;
          j= j+ 1;
          printf(".");
          fflush(stdout);
          sleep(1);
          myvariable= j;
          pthread_mutex_unlock(&mymutex);
        }
        return NULL;
      }
int main(void) {
pthread_t mythread;
int i;
if (pthread_create(&mythread, NULL, thread_function, NULL)){
          printf("error creating thread.");
          abort();
        }
        for ( i= 0; i< 20; i++ ){
          pthread_mutex_lock(&mymutex);
          myvariable= myvariable+ 1;
          pthread_mutex_unlock(&mymutex);
          printf("o");
          fflush(stdout);
          sleep(1);
        }
        if ( pthread_join ( mythread, NULL ) ) {
          printf("error joining thread.");
          abort();
        }
```

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

```
    printf("\nmyvariable equals %d\n", myvariable);
    exit(0);
}
```

虽然我们通过 pthread_create()只创建了一个线程,但主程序也是一个线程,因此两个线程各自独立地对变量 myvariable 递加,直到 pthread_join()将两个线程合并为一个线程。程序运行中,在 pthread_function()结束很久之前,主线程如果就已经调用了 pthread_join(),此时主线程将中断(转向睡眠)然后等待 pthread_function()完成;当 pthread_function()完成后,pthread_join()将返回,这样程序又只有一个主线程。

pthread_mutex_lock()和 pthread_mutex_unlock()系统调用执行了不可缺少的功能(互斥)来保护数据结构。如果没有它们,pthread_function()线程将覆盖主线程对 myvariable 变量的修改,使程序得不到我们设想的正确值。有兴趣的读者可以通过运行程序比较一下。

此外,线程的活动也可以使用条件变量、信号量来协调,其功能与上述互斥锁类似[4]。在编写多线程应用程序时,最重要的是控制好线程间的同步资源访问,以确保线程的安全运行。

3　线程的私有数据

有时需要能够维护指定到某个线程中的特殊数据,而这些数据在多个线程执行的代码段中引用,这时通常把具有这样性质的数据命名为线程说明的数据(thread_specific data;简称 TSD),并通过使用线程特定的指针和一个相关的键值(key)引用。

TSD 的读写都通过专门的 Posix Thread 函数进行。这些数据可以被所有的线程访问,但只由每一个线程自己维护,系统用一个惟一的键值来决定一个已知线程中的数据值。以下这个例子说明如何达到存储线程私有数据目的:

```
#include<stdio.h>
#include<pthread.h>
pthread_key_t  key;
void echomsg(int t)
{
        printf("destructor excuted in thread %d, param= %d\n", pthread_self(), t);
}
void *child1(void *arg)
        {
        int tid= pthread_self();
        printf("thread%d enter\n", tid);
        pthread_setspecific(key, (void *)tid);
        sleep(2);
        printf("thread%d returns%d\n", tid, pthread_getspecific(key));
        sleep(5);
}
        void *child2(void *arg)
        {
        int tid= pthread_self();
        printf("thread %d enter\n", tid);
        pthread_setspecific(key, (void *)tid);
        sleep(1);
        printf("thread %d returns %d\n", tid, pthread_getspecific(key));
        sleep(5);
}
int main(void)
{
        int tid1, tid2;
        printf("hello\n");
        pthread_key_create(&key, echomsg);
        pthread_create(&tid1, NULL, child1, NULL);
        pthread_create(&tid2, NULL, child2, NULL);
        sleep(10);
        pthread_key_delete(key);
        printf("main thread exit\n");
        return 0;
}
```

上述程序中创建两个线程分别设置同一个线程私有数据为自己的线程 ID,为了检验其私有性,程序错开了两个线程私有数据的写入和读出的时间,从程序运行结果可以看出,两个线程对 TSD 的修改互不干扰。同时,当线程退出时,清理函数会自动执行,参数为 tid。

4　多线程的应用

开发多线程应用程序可以利用 32 位 Windows 环境提供的 Win32 API 接口函数,也可以利用 VC++ 中提供的 MFC 类库进行开发。用 PB、VB、Delphi 以及 Java 等都能编写多线程应用程序,采用传统的 C 进行多线程的设计,必须控制线程间的同步。利用面向对象的设计方法,如采用 Delphi 进行多线程程序设计,问题就简单多了。而在.NET 和 C#环境中编写一个多线程应用程序非常容易[5],用户可以根据需要选择相应的工具。下面以 VC++ 5.0 中一个简单的基于对话框的 MFC 例程来说明实现多线程任务调度与处理

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net

的方法。

在例程中定义两个用户界面线程，一个显示线程(CDisplayThread)和一个计数线程(CCounterThread)。这两个线程同时操作一个字符串变量 m_strNumber。我们可以分别调整进程、计数线程和显示线程的优先级，同步机制使用 CMutex 和 CSingleLock 来保证两个线程不能同时访问该字符串。

例程的具体实现如下：生成一个名为 Mutexes 基于对话框的应用程序框架，对话框中填加三个组合框、三个复选框和一个列表框；三个组合框分别允许改变进程优先级和两个线程优先级，其 ID 分别设置为：IDC_PRIORITYCLASS、IDC_DSPYTHRDPRIORITY 和 IDC_CNTRTHRDPRIORITY，三个复选框分别对应着同步机制选项、显示计数线程执行选项和暂停选项，其 ID 分别设置为 IDC_SYNCHRONIZE、IDC_SHOWCNTRTHRD 和 IDC_PAUSE，列表框用于显示线程显示程序中两个线程的共同操作对象 m_strNumber，其 ID 设置为 IDC_DATABOX。

创建类 CWinThread 的派生类 CExampleThread。在 CExampleThread 的头文件中填加如下变量：

```
CMutexesDlg *m_pOwner; //指向类 CMutexesDlg 指针
BOOL m_bDone; //用以控制线程执行及函数
void SetOwner(CMutexesDlg * pOwner)
{ m_pOwner= pOwner; }; //取类 CMutexesDlg 的指针
//然后在构造函数当中对成员变量进行初始化
m_bDone= FALSE; //初始化允许线程运行
m_pOwner= NULL; //将该指针置为空
m_bAutoDelete= FALSE; //要求手动删除线程对象
```

创建 CExampleThread 的派生类 CCounterThread 和 CdisplayThread。

首先在对话框的初始化函数中加入代码对对话框进行初始化：初始化进程优先级组合框并置缺省为 NORMAL；初始化显示线程优先级组合框并置缺省为 NORMAL；初始化计数线程优先级组合框并置缺省为 NORMAL；初始化线程挂起复选框为挂起状态。

```
/* 初始化线程 */
m_pDisplayThread= (CDisplayThread *)
  AfxBeginThread(RUNTIME_CLASS(CDisplayThread),
      THREAD_PRIORITY_NORMAL, 0,
      CREATE_SUSPENDED);
  m_pDisplayThread-> SetOwner(this);
  m_pCounterThread= (CCounterThread *)
  AfxBeginThread(RUNTIME_CLASS(CCounterThread),
      THREAD_PRIORITY_NORMAL, 0,
      CREATE_SUSPENDED);
  m_pCounterThread-> SetOwner(this);
……
```

填加成员函数 void AddToListBox(LPCTSTR szBuffer);

```
/* 用于填加列表框显示该函数的实现代码为 */
void CMutexesDlg:: AddToListBox(LPCTSTR szBuffer)
{
  CListBox *pBox= (CListBox *)GetDlgItem(IDC_DATABOX);
    ASSERT(pBox != NULL);
    if (pBox != NULL){
      int x= pBox-> AddString(szBuffer);
      pBox-> SetCurSel(x);
      if (pBox-> GetCount()> 100)
        pBox-> DeleteString(0);
    }
}
```

然后利用 ClassWizard 填加用于调整进程优先级、两个线程优先级以及用于复选线程挂起的函数。

调整进程优先级的代码(略)，复选线程挂起的实现代码如下：

```
void CMutexesDlg:: OnPause()
{
  CButton *pCheck= (CButton *)GetDlgItem(IDC_PAUSE);
  BOOL bPaused= ((pCheck-> GetState()&0x003) != 0);
  if (bPaused) {
    m_pCounterThread-> SuspendThread();
    m_pDisplayThread-> SuspendThread();
  } /* 挂起线程 */
  else {
    m_pCounterThread-> ResumeThread();
    m_pDisplayThread-> ResumeThread();
  } /* 恢复线程运行 */
}
```

此外，多线程的应用在进行数据采集时可以有效地加快程序的反应速度、增加执行的效率。对于数据采集程序，可以用一个单独的线程进行数据采集。这样，能最大限度地保证采集的实时性，而另外的线程同时又能及时地响应用户的操作或进行数据处理[6]。

5 结束语

本文对现代操作系统中的多线程技术进行了讨论，并以例程说明了多线程编程及其应用。线程是多线程编程的主要接口，用户级的线程在用户之间操作，从而避免与内核之间的互相切换。为了控制线程(如更多地控制进程调度策略)，应用程序应当绑定线程。当一个应用程序把线程的所有执行资源绑定后，线程就变成了内核资源。

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved. http://www.cnki.net

不论是单机、C/S 结构、B/S 结构还是分布式结构的程序设计, 多进程、多线程的应用都很广泛, 单线程应用程序重新编写成多线程应用程序的策略和结果不是一成不变[7]。在使用多线程时要谨慎, 因为操作系统要跟踪和确定线程的进度, 因此线程的系统开销会比较大, 由于必须为每个线程分配内存, 太多的线程将会令整个系统的性能受到影响, 这样也不是在应用的任何地方都创建新的线程。

参考文献:

[1] William Stallings. Operating Systems Internals and Principles [M]. Third Edition. Englewood Cliffs, NJ: Prentice Hall, 1998.

[2] 尹俊文, 等. 分布式操作系统[M]. 长沙: 国防科技大学出版社, 2000.

[3] John Shapley Gray. Interprocess Communications in Unix[M]. Second Edition. Englewood Cliffs, NJ: Prentice Hall, 1998.

[4] Daniel Robbins. Common theads: POSIX threads explained, Part2 The little things called mutexes[DB/OL]. http://www-106.ibm.com/developerworks/library/posix2/index.html, 2000-08-02.

[5] Microsoft Corporation. Microsoft Visual C#.NET: Features Overview[DB/OL]. http://msdn.microsoft.com/vcsharp/productinfo/features.asp, 2002-01-12.

[6] 查礼, 等. 一种多线程计算程序的机群移植方法[J]. 计算机学报, 2002, 25(3): 306~312.

[7] Microsoft Corporation. Multithreading with C and Win32[DB/OL]. http://msdn.microsoft.com/library/en-us/vccore98/HTML/_core_multithreading_with_c_and_win32.asp, 2002-01-12.

(上接第 3 页)

但当传感器噪声很大时, 则会降低所要计算的如分割标志和边界特征等的品质。但是, 应用遗传算法可以消除上述不利因素, 达到良好的图像处理效果。

6 结束语

尽管遗传算法并未发展成为一种完全成熟的算法, 如对于交叉和变异概率的选择仍没有一个确定性的规定, 种群的个体数目也需要试算来决定, 但它已经在很多的领域中得到了应用, 尤其是在一些传统的算法不能得到很好的应用的问题上更加具有优越性。笔者正拟将遗传算法应用于机动车的变速齿轮箱的故障诊断中, 以此来解决变速箱故障集与征兆集之间的复杂因果关系。

参考文献:

[1] 张文修, 梁怡. 遗传算法的数学基础[M]. 成都: 西南交通大学出版社, 2000. 5.

[2] 陈国良, 王煦法, 庄镇泉, 王东生. 遗传算法及应用[M]. 北京: 人民邮电出版社, 2001, 2.

[3] 徐章遂, 房立清, 王希武, 左宪章. 故障信息诊断原理及应用[M]. 北京: 国防工业出版社, 2001, 7.

(上接第 6 页)

本文提出了用梯形来剖分多边形的通用算法, 多边形可以是非单调的, 可以包含孔, 孔也可以梯形化。算法分 3 个步骤: 初始化、梯形化及优化(后处理)。在最坏的情况下, 算法的时间复杂度为 $O(n^2 \log_2 n)$。每个多边形的局部分类规则和包含性检查确保算法对累积误差不敏感, 运算稳定。

参考文献:

[1] 王玉兰, 沈越江. 多边形的三角剖分及应用[J]. 成都理工学院学报, 1997(1): 108~111.

[2] 徐春蕾, 李思昆. 一种适用任意平面多边形的三角剖分算法[J]. 国防科技大学学报, 2000(2): 82~85

[3] 巩丹超, 戴晨光, 张永生. 三维模型重建中的凹多边形三角剖分[J]. 解放军测绘学院学报, 1999(3): 194~196.

[4] Seidel R. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons[J]. Computational Geometry: Theory and Applications, 1991(1): 151~154.

[5] Narkhede A, Manocha D. Fast polygon triangulation based on Seidel's algorithm[M]. Boston: Academic Press, 1995.

[6] Preparata FP, Shamos Mi. Computational Geometry: An Introduction[M]. Berlin: Springer, 1985.

[7] Manber U. Introduction to algorithms: a creative approach [M]. MA: Addison-Wesley, 1989.

[8] Mortenson ME. Geometric modeling[M]. New York: Wiley, 1985.

[9] O'Rourke J. Computational Geometry in C[M]. Cambridge: Cambridge University Press, 1993.

?1994-2015 China Academic Journal Electronic Publishing House. All rights reserved.　http://www.cnki.net