Linux 多线程编程

Andew Huang <bluedrum@163.com>

课程内容

- 线程的概念
  - 线程与进程区别
  - Linux 线程介绍
- Linux 线程基本编程
- Windows多线程编程

线程的概念

1. 线程的优点

- 线程的实现时间远晚于进程.最早实现是solaris 上线程,多线程技术已经被许多操作系统所支持，包括Windows/NT，和Linux
  - 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。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。而运行于一个进程中的多个线程，它们彼此之间使用相同的地址空间，共享大部分数据，.
  - 线程间方便的通信机制 ,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。
  - 多线程的程序会提高响应速度.特别是GUI
  - 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。

2. Linux 对线程的支持

- Linux系统下的多线程遵循POSIX线程接口 ,所以被称为pthread.
- pthread是目前Linux平台上使用最为广泛的线程库，由Xavier Leroy 负责开发完成，并已绑定在GLIBC中发行。
- Linux内核并不支持真正意义上的线程，LinuxThreads是用与普通进程具有同样内核调度视图的轻量级进程来实现线程支持的。这些轻量级进程拥有独立的进程id，在进程调度、信号处理、IO等方面享有与普通进程一样的能力。
- 因此一个多线程程序用ps查看,可以看到每一个线程也占用一行显行
- LinuxThreads 项目最初将多线程的概念引入了 Linux，但是LinuxThreads 并不完全遵守 POSIX 线程标准。一个更新的Native POSIX Thread Library（NPTL）正在应用开来.
- 由于历史原因.有大量应用采用pthread,很多嵌入式平台也只是支持pthread库.本节教材也将采用pthread作为线程应用库

3. 线程与进程区别

- 进程是一个应用程序独立运行单位,而线程不能独立存在,必须由在一个进程创建.
- 在Windows下,线程和进程都是内核内置的机制,而Linux一开始就在内核创建进程机制,直到2.2后才开始加入线程实现,到2.6才才稳定
- Linux的线程库是基于应用库实现,远没有基于内核的进程稳定.在重负衡的关键任务的服

务器,往往采用多进程机制实现.

进程空间

- 每个进程都独立占有4G的独立虚拟内存空间,各自拥有完整的堆,栈,代码段和数据段
- 在调用fork时,子进程完全从父进程4G空间复制过来
  - 这意味,现有堆栈的局域变量,全局变量等的值完全一样
- 在fork后,父子进程各自独立修改自己空间的变量
  - 两者即便是有同名变量,各自值也是独立修改
- 进程间,包括父子进程之间交换数据必须需要进程间通讯机制来实现

线程空间

- 同一个进程创建的所有线程之间共享创建进程的空间,即大家拥有相同2G空间,当一个线程修改一个全局变量后,另一个线程也访问得到这个值
- 因此在多线程并发修改某个值,必须要进行加锁保护,以便能正确修改值,

Linux 线程基本编程

1. Linux 线程的基本函数

- 常用线程函数
  - pthread_create 创建一个线程
  - pthead_exit 线程自行退出
  - pthread_join 其它线程等待某一个线程退出
  - pthread_cancel 其它线程强行杀死某一个线程
- pthread线程库的使用
  - glib库内置了线程库.
  - 在源码中使用头文件 pthread.h
  - 用**gcc**链接时加上 **–lpthread** 选项**,**链接线程库

1)pthread_create 创建一个线程

- 线程的创建
  - **int pthread_create(pthread_t *thread,pthread_attr_t *attr, void *(*start_routine)(void *), void *arg);**
    - pthread_create创建一个线程,thread是用来表明创建线程的ID,attr指出线程创建时候的属性,我们用NULL来表明使用缺省属性.start_routine函数指针是线程创建成功后开始执行的函数,arg是这个函数的唯一一个参数.表明传递给start_routine的参数.
  - 一个进程中的每个线程都由一个线程**ID**（**thread ID**）标识，其数据类型是**pthread_t**（常常是**unsigned int**）。如果新的线程创建成功，其**ID**将通过**thread**指针返回。

2)pthread_exit 退出一个线程

- 线程的退出有两种方式,一种线程函数运行结束,比如到函数结尾或用return退出.线程自然结束.这是最常用的方式.
- 调用pthread_exit退出.
  - **void pthread_exit(void *retval);**
    - 退出当前线程,并且设线程的返回值为retval
    - 返回值可以用**pthread_join**取得

3)pthread_join 等待线程退出

- **int pthread_join(pthread *thread,void **thread_return);**
  - 这个函数类似于waitpid,pthread_join是当前线程在等待另一个线程结束,只不过前者是在等待一个进程退出,后者在等待一个线程编号为thread的线程退出.
  - 当一个线程调用pthread_exit时,如果其它线程或进程使用了pthead_join在等待这个线程结束,那线程ID和退出状态将一直保留到pthread_join执行时.
  - **pthread_exit()**设定的返回值**,**或者**return**的返回值可以被**pthread_join**的**thread_return**捕获

4)pthread_cancel 杀死一个线程

- **pthread_cancel(pthread_t thread) ;**
  - 当前线程将杀死线程ID为thread的线程
  - **pthread_exit()**是当前线程自己退出**,**而**pthread_cancel**是其它线程杀死别的线程

2. 关于线程的生存周期

- 程序的主进程默认为主线程.
- 一个子线程的生存周期由pthread_create 创建后开始,
- 一直到线程函数执行完毕,或者执行到pthread_exit 处.线程的生命到此结束.
- 其它线程可以用pthread_cancel强制杀死另一线程.
- 主线程退出,(比如在主函数里调用exit),它的子线程无论是否执行完毕,都会随主线程退出而一同消失.
  - 所以在一般在主函数里,必须要判断的一下子线程是否真正退出,如是方可以把主线程退出.否则很可能造成程序结果不正确
  - 判断线程是否结束最简单的方法是在主线程的退出用 **pthread_join**来等待子线程退出即可

线程示例

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

#define RUN_TIME 10

/*  定义线程函数*/
void * thread(void * arg)
{
            int i;
            for(i=0;i<RUN_TIME;i++)
            {printf("This is a pthread.\n"); sleep(1);}
}

int main(void)
{
            pthread_t id;
            int i,ret;
           /*  创建一个线程  */
           ret=pthread_create(&id,NULL,(void *) thread,NULL);
           if(ret!=0){
             printf ("Create pthread error!\n");
             exit (1);
             }
```

```
        /*接上一页*/
        for(i=0;i<RUN_TIME;i++)
        {
        printf("This is the main process.\n");
        sleep(1);
       }

       /* 等待线程退出 */
        pthread_join(id,NULL);
        return (0);
}
```

3. 修改线程的属性

- 绝大部分情况下,创建线程使用了默认参数，即将pthread_create函数的第二个参数设为NULL。
- 在特殊情况才需要设置线程属性pthread_attr_t,
  - 属性对象主要包括是否绑定、是否分离、堆栈地址、堆栈大小、优先级。默认的属性为非绑定、非分离、缺省1M的堆栈、与父进程同样级别的优先级。
- 初始化线程属性
  - **pthread_attr_init(&attr);**
  - 这个函数必须在**pthread_create**函数之前调用
- 设置线程绑定特性
  - **pthread_attr_setscope**
  - 设置线程绑定状态的函数为pthread_attr_setscope，它有两个参数，第一个是指向属性结构的指针，第二个是绑定类型，它有两个取值：PTHREAD_SCOPE_SYSTEM（绑定的）和PTHREAD_SCOPE_PROCESS（非绑定的）。下面的代码即创建了一个绑定的线程。
- 设置线程的优先级
  - 它存放在结构**sched_param**
  - 用函数**pthread_attr_getschedparam**和函数**pthread_attr_setschedparam**进行存放，一般说来，我们总是先取优先级，对取得的值修改后再存放回去

修改线程属性实例

```
#include <pthread.h>
#include <sched.h>
pthread_attr_t attr;
pthread_t tid;
struct sched_param param;
int newprio=20;

pthread_attr_init(&attr);
pthread_attr_setscope(&attr, PTHREAD_SCOPE_SYSTEM);
pthread_attr_getschedparam(&attr, &param);
param.sched_priority=newprio;   //设置优先级
pthread_attr_setschedparam(&attr, &param);
pthread_create(&tid, &attr, (void *)myfunction, myarg);
```

多线程拷贝文件实例

- 扫描一个目录,把这一个目录下所有文件名加上*.bak 复制一次
- 这一个实例为每一个文件创建一个线程进行拷贝
  - 没有做优化.因此文件较多的情况下,线程可能较多
- 每个程序在内部使用 read,write 来处理文件

Windows线程支持

1. Windows线程

- Windows线程机制非常类似Linux的线程调用.
  - 包含头文件 <process.h>
  - 表示线程数据结构 unsigned long
  - 创建一个线程 _beginthread
    - **unsigned long _beginthread(void(__cdecl **start_address*)(void *), unsigned *stack_size*, void **arglist*);**
    - 第一个参数为线程函数,第二个函数为堆栈空间,取0由系统分配一个缺省尺寸,第三是线程函数参数,当线程创建时,arglist 将传送给线程函数
  - 线程函数执行完毕,将自动销毁线程,或提前用_endthread()进行销毁
  - 链接时时候需要使用**LIBMT.LIB**

```
#include <process.h>
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>


static void thread_func(void * arg)
{
   int i,count = (int)arg;

   fprintf(stderr,"count =%d,pid=%d\n",count,getpid());

   for(i=0; i< count ; i++)
   {
      fprintf(stderr," I am a thread %d\n",count);

      Sleep(1000);
   }

}
int main()
{
    _beginthread(thread_func,0,(void *)10);

     _beginthread(thread_func,0,(void *)20);

    getchar();
}
```

2. CreateThread:创建一个线程

- CreateThread()实现在Windows内核,即kernel.dll的实现,_beginthread实际上是调用CreateThread()来创建一个线程
- 如果想对线程做更复杂的控制,可以使用CreateThread来创建一个线程.
  - CreateThread创建线程数据结构用handle表示
  - 用CreateThread创建可以对线程做更多控制,如挂起,等待…

```
HANDLE CreateThread(
LPSECURITY_ATTRIBUTES lpThreadAttributes, // SD
DWORD dwStackSize, // initial stack size
LPTHREAD_START_ROUTINE lpStartAddress, // thread function
LPVOID lpParameter, // thread argument
DWORD dwCreationFlags, // creation option
LPDWORD lpThreadId // thread identifier
);
```

3. WaitForSingleObject

- WaitForSingleObject从字面意思上可以理解为等待单个对象,在Windows下,线程,互斥量,事件等各个操作系统对象都用使用这个参数.
- 如果是一个线程被等待,相当于是pthread_join的功能,在等待一个线程运行完毕
- WaitForSingleObject的优点之一是可以带超时参数,即到了指定时间未等想要结果,可以结束等待,而pthread_join无此功能.
  - 用宏定义INFINITE 表示无穷大,即无限等待下去
  - WaitForSingleObject(thread, INFINITE);

思考题

- 同样程序你可以用多线程和多进程模型来实现,你是如何权衡采用哪一种体系结构的?
- 在程序设计中,对公共资源(比如缓冲区等)的操作和访问经常需要使用锁来进行保护,但在大并发系统中过多的锁会导致效率很低,通常有那些方法可以尽量避免或减少锁的使用?

课堂练习

- 请设计一个多线程拷贝文件例子
  - 用键盘一个文件名字,然后创建一个线程将其拷贝到/tmp目录下.
  - 拷贝线程和主线程应该是并行运行的.即在输入文件名的同时,原来的文件拷贝线程也是运行
- 将test_mutex例子用Windows来实现
  - 1.线程用_beginThread或CreateThread
  - 2.加锁用WaitSingleObject来lock,用ReleaseMutex来解锁
  - 具体对应表参见上页,调用请参见MSDN
  - 最好定义一组相同接口的宏来封装操作系统细节,
- 要求设计一个双线程结构
  - 输入线程负责接收用户输入,并把结果通过消息队列发往服务器.用read(0…)或getchar均可实现
  - 显示线程接收输入线程发来的字符串,并显示在屏幕上.
  - 消息队列用数组+mutex/链表来实现