LINUX 系统下多线程与多进程性能分析[1]

周丽 焦程波 兰巨龙

(国家数字交换工程技术中心，河南，450002)

摘要：采用多进程处理多个任务，会占用很多系统资源（主要是 CPU 和内存的使用）。在 LINUX 中，则对这种弊端进行了改进，在用户态实现了多线程处理多任务。本文系统论述了多线程间通信技术的使用，通过对单 cpu 系统中多线程和多进程技术的比较和和性能分析，结合线程间通信技术的有关特性提出了应用该项技术所应遵循的原则和思路，并提出了改进思路，同时还讨论了多线程通信中存在的一些问题。

关键词: 多线程 多进程

中图分类号：TP—30　　**文献标志码**：A

The Performance of Multi-Pthreads vs Multi-Processes in Linux Operation System

ZHOU Li　JIAO Cheng-bo　LAN Ju-long

（National Digital Switching System Engineering & Technology Research Center, Henan,450002)

Abstract: Using multi-processes to handle multi-tasks will use a lot of system resources(mainly include cpu and memory).In linux os，this disadvantage is improved by using multi-threads to deal with multi-tasks.In this paper,we discuss the application of multi-threads and multi-processes by comparing their performance.In the end,we give a method which use multi-threads and multi-processes together to improve.Also we mention some problems in multi-threads communication.

Key words :multi-threads；multi-processes

1．关于进程和线程的简述

在 Linux 操作系统中，一个进程(Process)相当于一个任务(Task)，从操作系统核心的角度来看，进程是管理系统资源（cpu，内存，文件等）的基本单位，是为正在运行的程序所提供的运行环境；从用户角度来看，进程是应用程序的一个动态执行过程。进程具有一段可执行的程序、专用的系统堆栈空间、私有的“进程控制块”(即 task_struct 数据结构)和独立的存储空间。

在 linux 操作系统中，内核空间是通过进程模拟线程的，在用户空间用 pthread 创建线程。线程又称为轻量级进程 (LWP)，是程序执行的最小单位。一个进程至少需要有一个线程来执行指令。线程也具有一段可执行的程序、专用的系统堆栈空间、私有的“进程控制块（PCB）”(即 task_struct 数据结构)，但是没有自己的存储空间。

线程与进程的主要区别在于：线程不能够单独执行，它必须运行在处于活动状态的进程中；多个线程共享同一进程除 cpu 以外的所有资源，各线程间允许任务协作和数据交换。创建线程比进程开销小，多线程间通信也比多进程间的通信过程简单。简而言之，线程只不过是进程的一个执行上下文。

通常单独一个程序运行时，缺省的包含一个主线程，主线程以函数地址的形式（如 main 函数）提供程序的启动点，这就是单进程单线程的情况。若在 main 函数中创建多个线程，则该程序运行时，操作系统为每个线程分配不同的 CPU 时间片，并根据线程优先级进行调度。由于每个时间片时间很短，看上去好象各个线程是并发执行的，实际上同一时刻只有一个线程在运行，这就是单进程多线程的情况。若用 fork 函数创建多个进程，而每个进程只采用默认的一个主线程，则程序运行时，由内核调度操作系统，将 cpu 分配给各个进程使用，

[1] 基金项目 国家 863 计划资助项目，可扩展到 T 比特的高性能 IPv4/v6 路由器基础平台及实验系统
项目编号：2003AA103510

这就是多进程的情况。

2．问题描述

在单 cpu 的机器上，采用多进程单线程（每个进程只创建一个线程）和单进程多线程（一个进程创建多个线程）模型来设计程序，后者（多线程）的上下文切换开销就比前者要小的多。虽然多线程提供的优点是单个线程的进程所欠缺的，但是它们也不乏一些缺点：

- 由于线程间共享存储器和进程状态，一个线程的动作可能会对同一个进程中的其他线程产生影响。例如当两个线程同一时刻访问同一个变量时，他们之间就会产生相互干扰。
- 当多个线程试图并发调用同一个库函数时，返回结果是不可预知的。如果多个线程调用某个库函数，线程之间必须互相加以协调，确保某个时刻只有一个线程调用该库函数。
- 缺乏健壮性，在单线程的操作系统中，如果某个存储器出错，操作系统会终止引发故障的进程，但是在多线程操作系统中，如果一个线程出错，操作系统将终止整个进程，从而其他的无关线程也被终止了。这与线程的正常终止是不同的。线程正常终止时可以通过调用 exit 函数终止它所在的整个进程，也可以终止自己而不影响进程内的其他线程。可以在主线程返回时终止该线程，也可以调用 pthreaad_exit 终止该线程。

当在单个进程中创建多线程和采用多个进程完成同样的任务时，对 cpu 和内存的使用是不同的，同时系统的负载也有很大的变化。创建一个进程时，它的第一个线程称为主线程（Primary thread），由系统自动生成。然后可以由这个主线程生成额外的线程，而这些线程，又可以生成更多的线程。创建多线程的目的就是尽可能地利用 CPU 时间。

对于大多数在 Linux 下编程的程序开发人员来说，多线程与多进程编程是最基本的开发手段之一，但究竟两者在应用中的具体性能差别在哪些呢？这也正是本文的研究重点！

3．多线程和多进程的性能比较

对于软件开发软员来说，CPU，内存(空间)与程序运行时间（时间）是三个最重要资源，本文中也以次三个参数最为衡量程序效率的标准。下面分别用两个程序对实际应用中线程和进程的这三个参数进行了系统的分析和比较。

1． 在第一段代码中调用函数 pthread_create 动态创建 255 个线程，线程执行完任务后再用 pthread_exit 终止线程。之所以将线程的个数定为 255 是因为 linux 操作系统对并发线程数的最大限制；在每个线程中重复打印"hello linux"语句，此时所有的线程就像是在同一时间并发执行一样，如果是在多处理器机器上，系统就可为每个线程分配一个 CPU，多个线程可以并发执行。程序代码如代码 1 所示。

```
#include<pthread.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdio.h>
#define Test_Log "logFIle.log"
FILE *logFile=NULL;
print_hello_linux()
{ int i=0;
 for(i=0;i<a;i++)
  {   printf("hello linux\n");
      fprintf(logFile,"hello linux\n");}
  pthread_exit(0); }
  main()
{ int i=0;
   pthread_t pid[255];
   logFile=fopen(Test_Log,"a+");
   for(i=0;i<255;i++)
   pthread_create(&pid[i],NULL,print_hello_li
nux,NULL);
    for(i=0;i<255;i++)
    pthread_join(pid[i],NULL);
     return 0;}
```

代码 1

2． 在第二段代码的 main 函数中调用 fork 函数创建 255 个进程，这是为了和上面的 255 个线程保持一致，事实上 linux 系统对于最大进程数的限制在 init_fork.c 中采用函数进行限制，实际的数量与进程所占用的资源有关（在 linux 内核 2.4 版本以后采用该函数限制，之前的版本在编译内核时进行了宏定义规定为 512）。进程执行完用 exit 终止进程。在每个进程中完成与多线程相同的任务，重复打印"hello linux"语句。程序代码如代码 2 所示。

```
#include<stdlib.h>
#include<stdio.h>
#define TEST_LOGFILE "logFile.log"
FILE *logFile=NULL;
main()
{ int i=0,j=0;
  logFile=fopen(TEST_LOGFILE,"a+");
  for(i=0;i<255;i++)
  { if(fork()==0)
    { for(j=0;j<10;j++)
      { printf("hello linux\n");
        fprintf(logFile,"hello linux\n");}
exit(0);}}
wait(0);
return 0;}
```

代码2

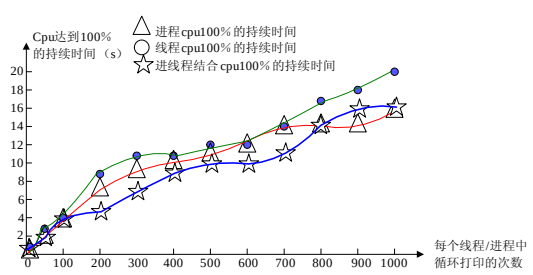
每次打印出来的文本记录在日志文件 logFIle.log 中，a 是重复打印的次数，在不同的测试中分别设为 10 次，50 次，100 次，200 次。。。。900 次。

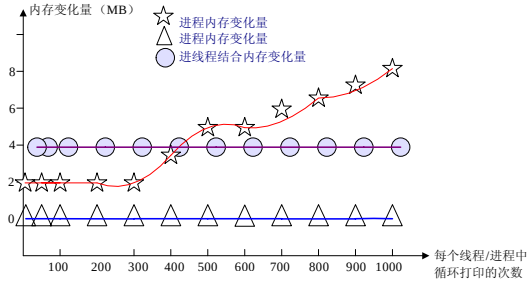
注意：本次试验中使用的单 cpu 机器基本配置为：celeron 2.0 GZ, 256M, Linux 9.2，内核 2.4.8。

为了更好地比较实际应用中多线程和多进程在内存，cpu 占用和时间上的差异，采用进程和线程完成相同的任务，这样可以不考虑 linux 内核对 cpu 的调度问题。

图一中的绿色和红色曲线分别表示采用多进程和多线程完成不同的打印任务时,cpu 的工作时间。不难看出，当采用多线程和多进程分别执行相同的任务，完成相同的任务量时，任务量较大时，多进程比多线程效率高；而完成的任务量较小时，多线程比多进程要快；当任务量为重复打印 600 次时，多进程与多线程所耗费的时间时间相同。

图二中的红色和蓝色曲线分别表示采用多进程和多线程完成不同的打印任务时，内存的变化量。不能看出，当采用多进程和多线程分别执行相同的任务，完成相同的任务量时，多线程的建立不产生额外的内存占用，因为多线程创建时采用的是内存共享机制；而多进程则需要占用 2 到 8M 不等的内存，可见多进程以较快的速度完成任务是以内存的占用为前提的，也就是“以空间换时间”。





与标准 fork() 相比，线程的创建非常快捷带来的开销很小，因为内核无需单独复制进程的内存空间或文件描述符等等，从而节省了大量的 CPU 时间，使得线程创建比新进程创建快上十到一百倍。但是这并不意味着可以大量使用线程而无需太过于担心带来的 CPU 或内存不足，事实上在任务量繁重时，建立多个线程并不一定能够获得更高的效率，相反会加大 cpu 的任务量，即使是在多 cpu 操作系统中线程数也不应该超过 cpu 的个数。使用 fork() 时会导致的大量 CPU 占用，但是占用的时间是短暂的。可见当待完成的任务量较大而系统设备性能较好时，可以通过创建多个进程在最短的时间内完成任务；当系统的内存较小而

对实时性的要求不高时，倾向于创建多线程。具体应该按照设计要求和系统地性能来选择。

4．对多进程和多线程技术的改进及应用

鉴于上面对于多线程和多进程的研究分析和性能比较，两者有利有弊：多线程耗时但节约空间，多进程省时但是占用内存。因此最好将两者结合起来，我们尝试着将多线程与多进程结合起来，重复 3 中的过程（函数代码如代码 3 所示）测试 cpu100％的时间和内存的占用变化量，图一中的蓝色曲线记录了 CPU 的工作时间，图二中的紫色曲线记录了内存的占用变化量。可见，将这两者结合起来应用可以在较短的时间内高效的完成任务，同时对内存的要求也比多进程要低，这样的高效是多进程和多线程所无法达到的。

```
#include<pthread.h>
#include<unistd.h>
#include<stdlib.h>
#include<stdio.h>
#define Test_Log "logFIle.log"
FILE *logFile=NULL;
print_hello_linux()
{ for(int i=0;i<a;i++)
   { printf("hello linux\n");
     fprintf(logFile,"hello linux\n");}
   pthread_exit(0); }
main()
{ int i,j;
   pthread_t pid[15];
   logFile=fopen(Test_Log,"a+");
   for(i=0;i<16;i++)
       if(fork()==0){
   for(j=0;j<16;j++)
   pthread_create(&pid[j],NULL,print_hello_li
nux,NULL);
   for(j=0;j<16;j++)
   pthread_join(pid[j],NULL);
   exit(0);}
   wait(0);
   return 0;}
```

代码3

但是我们也必须看到，在测试中我们所用的程序只是非常简单的打印程序，在实际的应用中，对实时性有着非常高的要求。通常采用临界区，互斥量，信号灯或者事件来实现线程和进程的同步。但是当两者结合起来使用时，涉及到同步和资源共享的问题,不象多进程或多线程那样单单用信号量或者加锁就可以解决，还有待进一步的研究。

5．结束语

线程和进程在计算机技术快速发展的今天越来越受到程序员的重视。除了 DOS 操作系统以外，几乎所有的操作系统都支持多线程和多进程，在 window 操作系统和 linux 操作系统中更是广泛地应用了多线程/多进程技术。本文主要对单 cpu 系统中多线程和多进程技术的性能进行了实际应用中的比较，分析了其利与弊，并根据试验结果提出了新的应用。软件开发人员可以根据具体情况(包括软件使用场合，具体服务器配置等)参考本文，完成更高效的程序代码。

参考文献:

1　Edward Bradford，linux 和 windows 上的高性能编程技术， 2001 年 10 月；
2　郭玉东，《linux 操作系统结构分析》，西安电子科技大学出版社，2002 年 1 月 1 日；
3　Mark G.Sobell[美]，《A Practical Guide to Red Hat Linux 8》电子工业出版社，2004 年 2 月；
5．李善平，郑扣根编著。Linux 操作系统及试验教程，北京机械工业出版社，1999 年；

[作者简介]：

1. 周丽 女，1981 年 6 月出生，汉族，在读研究生，通信与信息系统专业，主要从事高性能 IPV6 路由器方面的研究。通信地址：河南郑州 1001 信箱研究所研究生队 邮编：450002

E-mail：zhouli@mail.ndsc.com.cn

ZHOU LI female，born in July，1981 ，Han，Master,major in communication and information system,now study high performance router applying IPV6.

2. 兰巨龙 男，1962 年 3 月出生，汉族，博士生导师，主要从事高速宽带信息网络技术方面的研究。

LAN Julong male ,born in March,1962,Han,Dr. ,major in Technology of High Speed and Broad Band Information Network.