多进程并发在国内气象通信系统的应用

胡英楣　沈文海　宋之光
（国家气象信息中心，北京 100081）

摘　要

该文介绍了多进程并发在国内气象通信系统中传输新增资料的应用，针对国内新增气象资料种类繁多、信息量大、传输时效要求高、传输频率快等特点，传输软件通过使用避免僵尸进程产生的方法以及对多种进程间通信技术的比较，选择了更好满足需求的消息队列通信技术，使气象资料在国内气象通信系统中的处理时间缩短为原来的 43%，提高了各种新增气象资料的传输效率，为气象资料提供了更好的共享平台，更好地发挥各种探测仪器的作用。

关键词：新增资料；多进程并发；僵尸进程；进程间通信

引　言

随着中国气象局实施的大气监测项目和传统气象观测向地球环境监测的过渡，各种探测手段和各类气象服务产品不断涌现，为天气预报、气候预测、气象科学研究和其他气象服务提供了更加丰富的气象资料。为了充分发挥气象资料的重要作用，如何及时、准确、高效传输新增气象资料也就成为一个不容忽视的重要环节。

目前，国内通信系统的传输方式已经由卫星通信改为地面宽带通信，新增资料的传输瓶颈也由传输带宽转换为传输软件。尤其是北京主站，作为全国新增资料的汇聚点，收集资料种类多达 23 类；资料传输频率和时效各不相同，有的资料每 10 min 传输 1 次，有的资料每小时或几小时传输 1 次，每次传输 1 个或多个文件，传输时效也相应从 10 min 至 1 d。如果主站传输软件按照以前的实现方法：单一进程接收、处理、分发所有的资料，那么资料的传输时效将不能满足业务的需求。

针对新增资料传输文件数量多、信息量大、传输频率快、传输时效要求高的特点，为了将收到的新增资料及时传输给各用户，在通信软件开发时，使用了多进程并发技术来提高新增资料的传输时效。

多进程并发[1]是一项成熟的技术，在多道程序设计系统中同时存在着许多进程，在单处理器的情况下，1 个进程的工作没有全部完成之前，另 1 个进程就可开始工作，这些可同时（交替）执行的进程具有并发性，把可同时执行的进程方式称为多进程并发，目前在我国医疗、电信、银行等行业应用很广，但在我国气象通信系统中的应用还是属于首次。

1　多进程并发的应用

1.1　进程概述

进程是一个具有一定功能的程序关于某个数据集合的一次运行活动。进程是操作系统动态执行的基本单元，当运行任何 1 个 UNIX 命令时，shell 至少会建立 1 个进程来运行这个命令，所以可以把任何在 UNIX 系统中运行的程序叫做进程；但是进程并不是程序，进程是动态的，而程序是静态的，并且多个进程可以并发地调用同 1 个程序。

进程在运行期间，会用到很多系统资源，包括最宝贵的 CPU 资源，当某 1 个进程占用 CPU 资源时，别的进程必须等待正在运行的进程空闲出 CPU 后才能运行，如果存在多个进程在等待资源，则操作系统内核通过调度算法来决定将 CPU 分配给哪个进程。

1.2　多进程并发在国内通信系统中的应用

随着地球环境观测、全球气候观测系统、全球大气观测系统等国际性的观测网络的日趋完善，国内参加交换的数据种类、数量、频次也随之猛增，尤其是自动站地面观测、高空探测、多普勒雷达、卫星[2]、闪电

200609-13 收到，2007-04-20 收到再改稿。

定位、GPS、沙尘暴、大气成分、生态、水文原始观测数据以及预报产品等新增的气象数据,给国内通信造成了很大的传输压力。为了提高通信软件的传输效率,使气象数据能及时在国内通信系统中交换,并及时分发给其他的应用系统或用户,在国内通信系统中使用了多进程并发技术,通过多个并发工作的进程来提高气象数据的交换速度,其软件结构如图 1 所示。

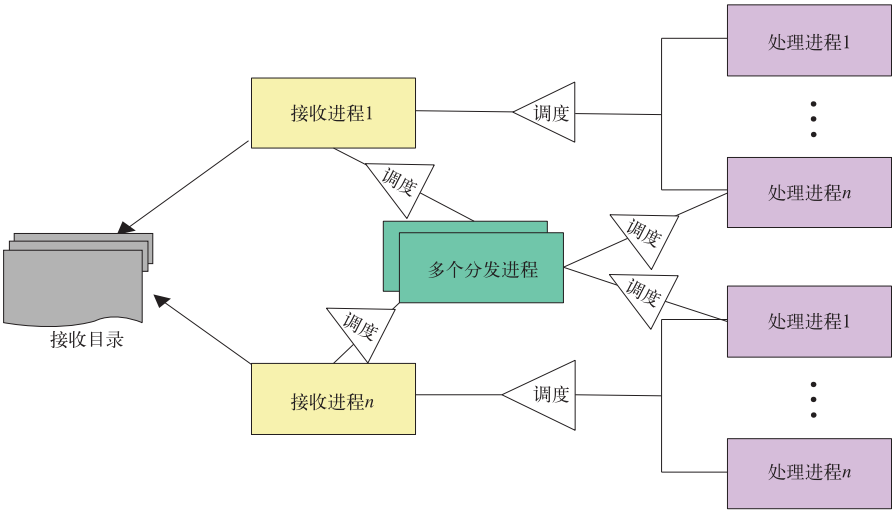

图 1 多进程并发软件结构示意图

Fig. 1 Concurrent multiprocess software structure

通信软件按功能要求可以分为 3 部分:接收、处理和分发。

接收功能:扫描接收目录集中的文件,按节目表将收集的文件进行不同的调度或目录分发。

处理功能:对接收文件进行规范化处理及信息提取,如:文件格式检查、文件打包合并、站号信息抽取等。

分发功能:将接收或处理的文件按分发节目表发送到一个或多个不同的目的地址。

在新增资料的通信软件中,接收进程可以根据要求调度 1 个或多个处理进程和分发进程,处理进程也可以调度 1 个或多个分发进程。因此可以通过多个接收进程来接收相同或不同目录下的气象数据,并同时启动多个处理进程和分发进程来对文件进行处理和分发,以提高通信系统的交换能力(如图 1 所示)。

通过并发接收,提高了气象资料的接收效率;通过并发处理文件,避免了待处理文件的积压,使需要处理的文件能及时被处理;通过多进程并发分发文件,即每个分发目的地址通过一个分发进程来实现,可以在同一时刻向多个目的地址分发文件,避免分发文件的积压,同时也避免了某一目的地址因网络故障而影响其他目的地址的文件分发。

虽然多进程并发可以提高通信软件的传输性能,但进程数目不宜太多,以免影响系统的性能[3-4],必须对进程的个数进行控制。

1.3 多进程并发需要解决的问题

在多进程并发软件开发的过程中,需要解决僵尸进程和进程间通信 2 个关键问题。

① 僵尸进程

“僵尸”进程是一个早已死亡的进程,但在进程表中仍占了一个位置。当通信软件运行时,使用 ps 命令查看进程的执行状态,经常会看到一些进程的状态栏为 defunct[5],这就是所谓的“僵尸”进程。

由于进程表的容量是有限的,所以,defunct 进程不仅占用系统的内存资源,影响系统的性能,而且如果其数目太多,还会导致系统瘫痪。

② 进程间通信

在通信软件运行时,无论进行“接收”、“处理”还是“分发”中的任何一种操作,首先需要获取操作对象——即待操作数据文件的有关信息,获取这些信息的手段通常是对这些文件进行扫描(这在单独完成这些操作时是必不可少的)。但在新增资料通信软件中,由于接收进程可以根据要求调度 1 个或多个处理进程和分发进程,处理进程也可以调度 1 个或多个分发进程,所以被调度进程(即被接收进

程调度的处理进程及分发进程,以及被处理进程调度的分发进程),由于其前后操作对象是一致的,可以通过调度它们的上级进程(或父进程)来获取这些对象的有关信息,而不必再各自另行重复扫描来获取这些信息。

采取这种方法,可大大减少不必要的重复性操作,提高处理进程和分发进程的时效性,并提高有效信息的使用率,从而在整体上提高软件的运行效率。所以,实现调度进程与被调度进程间的信息通信,是大幅度提高软件运行效率,从而实现采用"多进程并发"技术的目的——全面提高通信软件接收、处理、分发时效性的关键所在。

2　僵尸进程

2.1　僵尸(defunct)进程概述

僵尸进程的出现是由于子进程结束后,父进程没有读取到该子进程的相关信息所致。

如果一个程序设计上有缺陷,就可能导致某个进程的父进程一直处于睡眠状态或是陷入死循环,从而无法读取该子进程的信息,于是当该子进程执行结束后,由于其有关信息迟迟无法被其父进程读取,就变成了 defunct 进程,此时这个 defunct 进程会一直留在系统中。如果这种情况反复出现,则僵尸进程便会越积越多,越来越多地占据进程表的空间,此间系统会表现得越来越迟钝,待僵尸进程将进程表占满后,系统便会瘫痪。

在此期间,即便操作员(或系统维护人员)发现了这些情况,由于调度程序无法选中 defunct 进程,常用的方法(如:使用 kill 命令)无法删除 defunct 进程,因此无法通过正常手段制止或延缓事态的继续恶化。遇此情况,维护人员通常采用的方法(也是较为简单的处理方法)是重新启动系统。

2.2　处理方法

由上面的分析可知,僵尸进程产生的原因在于子进程终止后,父进程无法获取其相应的信息,且这种现象产生的原因多半是由于父进程的程序设计存在缺陷所致。因此根本的解决方法是测试并发现父进程程序设计中的缺陷,并加以改正。

然而如果是操作系统存在缺陷,则编码人员无法通过修改父进程来避免僵尸进程的出现。此时,为达到避免 defunct 进程产生,可采取强制方法,迫使父进程在某个阶段停止自己的工作,专门接收子进程的信息。较为常用的方法是在父进程中调用 wait( )或 waitpid( )。这时,父进程一直处于接收子进程信息的状态,因此在其子进程终止后便会立即得到该子进程返回给它的有关信息(即进程表中的该子进程相应数据),此后系统会因为父进程已收到其子进程的结束信息,而立即在进程表中删除该子进程的进入点,使得该子进程彻底销毁。这样就避免了 defunct 进程的产生。

① wait( )函数工作原理

wait( )的函数原型是:

```
#include < sys/types. h > / * 提供类型 pid_t 的定义 * /
#include < sys/wait. h >
    pid_t wait (int * status)
```

进程一旦调用了 wait( ),就立即阻塞自己,由 wait( )自动分析是否当前进程的某个子进程已经退出,如果让它找到了这样一个已经变成僵尸的子进程,wait( )就会收集这个子进程的信息,并把它彻底销毁后返回;如果没有找到这样的子进程,wait( )就会一直阻塞在这里,直到有一个出现为止。

参数 status 用来保存子进程退出时的一些状态,它是一个指向 int 类型的指针,通过这些信息可以分析该子进程终止的原因等等。由于调用 wait( )的目的是避免僵尸进程的出现,对该子进程是如何终止的并不关心,因此在这里可以设定这个参数为 NULL。如果成功(即出现终止的子进程,并已对其进行处理并将其销毁),wait( )会返回该子进程的进程 ID;如果父进程在调用 wait( )时,已没有子进程存在(无论是活的子进程,或者是僵尸子进程),则调用就会失败,此时 wait( )返回 -1,同时 errno 被置为 ECHILD。

② waitpid( )工作原理

waitpid 的函数原型是:

```
#include < sys/types. h > / * 提供类型 pid_t 的定义 * /
#include < sys/wait. h >
    pid_t waitpid( pid_t pid, int * status, int options)
```

调用 waitpid( )和 wait( )的作用是完全相同的,但 waitpid( )多了两个可由用户控制的参数 pid 和 options,从而为编程提供了另一种更灵活的方式。

参数 pid 的取值范围及相应的含义:pid > 0 时,只等待进程 ID 等于 pid 的子进程,不管其他已经有

多少子进程运行结束退出了,只要指定的子进程还没有结束,waitpid()就会一直等下去;pid = -1 时,等待任何一个子进程退出,没有任何限制,此时 waitpid()和 wait()的作用一模一样;pid = 0 时,等待同一个进程组中的任何子进程,如果子进程已经加入了别的进程组,waitpid()不会对它做任何理睬;pid < -1 时,等待一个指定进程组中的任何子进程,这个进程组的 ID 等于 pid 的绝对值。

参数 options 提供了一些额外的选项来控制 waitpid()。如果在调用 waitpid()函数时,参数 options 设为 WNOHANG,父进程在运行到此处时,会检查此刻有无已退出的子进程信息尚未收集,如有,则对其进行相应的信息收集处理,并最终销毁该子进程,并返回;如没有子进程退出,它也会立即返回,不会像 wait()那样永远等下去。

③ wait()和 waitpid()之间的比较

waitpid()的返回值比 wait()稍微复杂一些,一共有 3 种情况:当正常返回的时候,waitpid()返回收集到的子进程的进程 ID;如果设置了选项 WNOHANG,而调用中 waitpid()发现没有已退出的子进程可收集,则返回 0;如果调用中出错,则返回 -1,这时 errno 会被设置成相应的值以指示错误所在;当 pid 所指示的子进程不存在,或此进程存在,但不是调用进程的子进程,waitpid()就会出错返回,这时 errno 被设置为 ECHILD。

2.3 新增资料通信软件中对僵尸进程的处理方法

① 僵尸进程现象的出现

在新增资料通信软件开发初期,并未对僵尸进程予以充分的重视。然而在通信软件试运行时发现,接收进程每接收一个文件(该接收进程将会根据节目表要求启动相应的处理进程或分发进程),在通信节点机上便会出现若干僵尸进程,而且僵尸进程的个数随着通信软件的运行不断增多,通信节点机性能逐渐下降,只有通信节点机的系统管理员重新启动系统后,僵尸进程消失,节点机性能才能恢复。如果系统在一段时间内没有重新启动,通信节点机的性能会下降得十分严重,以至于接收进程不能正确地启动处理进程和分发进程,从而导致通信软件不能及时地接收气象数据,严重影响了气象资料的及时交换。

② 僵尸进程现象分析

由于操作系统的特点(如分时以及其他因素等),任何进程都存在休眠的可能,接收进程也不例外(虽然很短暂),因此僵尸进程的出现需要编程者特别的关注。

通常在资料较少的情况下,待接收进程结束休眠后,便会因该接收进程(父进程)接收到已结束运行的子进程的相应信息,系统销毁该子进程,从而销毁了该僵尸进程。但在资料大批量突然涌入,导致接收进程被大量触发时,僵尸进程会因系统调度等方面的原因而来不及被一一销毁,一旦僵尸进程销毁的速度低于其产生的速度时,便会陷入恶性循环:僵尸进程数量的增长使得用户进程表空间越来越多地被僵尸进程占据,从而使得用户可启动的进程数量越来越少,导致系统性能的下降;而系统性能的下降更进一步减缓了僵尸进程的销毁速度,从而使得僵尸进程的数量进一步增加,使得系统性能进一步下降,最终导致系统瘫痪。

因此,僵尸进程现象不仅可能与应用程序有关,而且可能与所在系统的特性及配置有关。

③ 处理方法

发现僵尸进程现象后,最初曾对接收进程进行了多次修改,但效果都不好,未能将僵尸进程完全消灭。经过反复探讨,得出了上面的分析结论;在此基础上,经过权衡,最后决定采用强制方法消除僵尸进程。通过多次试验,最终选用了调用 waitpid()函数,以避免僵尸进程出现的方法。其简要实现方法如下:

```
#include < unistd. h >
#include < sys/types. h >
#include < sys/wait. h >

RecvProc( )
{
    pid_t pid;
    int   status;

    ……
    pid = fork( );

    if( pid < 0) //创建子进程失败
    {
    ……
    printf("fork error\n");
    exit(0);
```

其大致的流程是:接收进程 RecvProc()首先通

过 fork( )来创建一个子进程,然后根据需要启动处理或分发进程,最后调用 waitpid( )来收集子进程信息,以避免僵尸进程的产生。

④ 处理效果

接收进程通过调用 waitpid( )函数,完全避免了僵尸进程的产生,达到了预期的效果。同时,通过分析子进程退出时的状态信息,可了解处理和分发进程对文件的处理情况,并根据这些情况制定相应的处理策略,从而确保国内通信系统准确、及时、高效地运行。

3　进程间通信

在 UNIX 操作系统中,每个进程都有自己的地址空间。如果要使多个进程协同工作,就需要进程间通信(interprocess communication)。

3.1　常用的进程间通信技术概述

① 管道(pipe)

管道[6-7]是许多应用程序的基本构建模块。管道是半双工的,数据只能向一个方向流动,如需要双向通信时,需建立两个管道。管道对于管道两端的进程而言,单独构成一个文件系统。它们的数据是一个字节流,类似于 TCP 连接。在某一进程往管道中写入数据前,要求另外某一进程在该管道上等待数据的到达,如果读出者不存在,则先有写入者是没有意义的。

② 消息队列(message queue)

消息队列可以认为是一个消息链表。有足够写权限的进程可往队列中写入消息,有足够读权限的进程可从队列中读出消息。每个消息是一个记录,它由发送者赋予一个优先级。一个进程可以往某一队列中写入消息后终止,让另一个进程在以后某个时刻读出这些消息。这跟管道是相反的。

③ 信号灯(semaphore)

信号灯是一种用于提供不同进程间或一个给定进程的不同线程间同步手段的原语。信号灯与其他进程间通信方式不同,它主要提供对进程共享资源访问控制机制。相当于内存中的标志,进程可以根据它判定是否能够访问某些共享资源,同时,进程也可以修改该标志。

④ 共享内存(shared memory)

共享内存是最快的 IPC(InterProcess Communication 形式。它通过将同一块物理内存映射到各进程的地址空间来实现共享内存中数据的更新。由于多个进程共享同一块内存区域,必然需要某种同步机制。

⑤ socket 会话编程(socket session programming)

socket 会话编程[8-9]主要用于远程进程间通信,使用该技术,用户编程的灵活性较大。TCP/IP 的 socket 提供了 3 种类型套接字:流式套接字(stream sockets)、数据报套接字(datagram sockets)、原始套接字(raw sockets)。

3.2　几种常用进程间通信技术的比较分析

在上述 5 种进程间通信技术中,每种技术都有自己的特点和使用范围,管道普遍用于 shell 中,不过也可以从程序中使用,往往是从子进程向父进程回传信息。

消息队列与管道相比,具有更大的灵活性,首先,它提供有格式字节流,有利于减少开发人员的工作量;其次,消息具有类型,在实际应用中,可作为优先级使用。这两点是管道所不能比的。同样,消息队列可以在几个进程间复用,而不管这几个进程是否具有亲缘关系,但消息队列是随内核持续的,与管道相比,生命力更强,应用空间更大[10-11]。

信号灯与其他进程间通信方式不大相同,它主要提供对进程间共享资源访问控制机制[12]。相当于内存中的标志,进程可以根据它判定是否能够访问某些共享资源,同时,进程也可以修改该标志。信号灯在使用时受到一些限制:每个 semop 系统调用可以执行的信号操作的数量 SEMOPM、系统允许的信号量值的最大数 SEMVMX、信号集的最大数目 SEMMNI、用户可访问的最大信号量 SEMMNS、每个信号集的最大信号数量 SEMMSL。

共享内存虽然是最快的进程间通信技术,因为进程可以直接读写内存,而不需要任何数据的拷贝,但是由于多个进程共享同一块内存区域,必然需要采用某种同步机制。

3.3　进程间通信技术在新增资料通信软件中的应用

① 新增资料通信软件的特点

国内通信系统传输新增资料时具有 6 个特点:传输信息量增长速度快;传输文件大小相差较大;传输时段高度集中;传输时效要求严格;传输内容要求准确;资料处理方法各异。

为了满足业务的需求,结合新增资料的 6 个传

输特点,通信软件在开发时必须考虑满足资料处理的及时性、处理优先级,以及软件的可扩充性。

选择采用多进程并发技术来提高资料传输的及时性,并选择采用进程间通信技术来进一步提高软件的运行时效。

② 进程间通信技术的选择

本文通过多进程并发来提高资料传输的及时性。由 1.3 节 ② 的叙述可知,此地的进程间通信主要是指接收进程与被其调度的处理进程和分发进程之间的父进程与子进程之间的通信。

由 3.1 节及 3.2 节得知,如果采用管道、信号灯、共享内存、socket 会话这 4 种进程间通信技术,通信软件必须要开发单独的程序代码来控制对不同数据的处理优先级,这样既增加了应用软件的开发难度,也增多了发生故障的环节。

使用消息队列则可以利用消息类型来作为数据的处理优先级,这样简化了通信软件的开发难度,也减少了发生故障的环节。此外,如果增加新的气象资料,只要开发出相应的处理代码,根据情况增加或利用原有的消息类型,即可轻松实现;这使得通信软件的可扩充性得以满足业务的需求。

③ 进程间通信技术实现

通信软件采用进程间通信的工作原理如下:

接收进程通过目录扫描获取文件后,用消息队列将文件信息传输给处理进程和分发进程,处理进程和分发进程采用阻塞[13]的方式来读取消息,一旦接收进程发送一个消息,处理进程和分发进程立刻就能接收到该消息,及时对文件进行处理和分发,减少了扫描目录的次数;处理进程和分发进程从消息队列中读取消息时,利用消息队列的读取机制:用户可以定义读取消息的类型 msgtype,如果 msgtype = 0,读取消息队列中的第 1 个消息;如果 msgtype 为正整数,则读取该类型的第 1 个消息;如果 msgtype 为负整数,则在其类型值小于或等于 msgtype 绝对值的所有消息中,选择类型最低的第 1 消息读取。通信软件充分利用消息类型的这一特点,将其作为气象资料的优先级,优先读取时效性强的资料。

通信软件利用消息对列的第 2 个优点是用阻塞的方式读取消息,如果消息队列中有消息时,按优先级读取消息;如果没有消息时,读取进程则一直阻塞,直至有新的消息到达。因此处理进程和分发进程不用定时休眠,提高了处理和分发气象资料的时效。综上所述,使用消息队列进程间通信技术大大加强了国内通信系统对时效要求较高的气象资料的及时交换能力,从而提高了通信系统的数据交换效率。

实现方法可以分成以下几部分:

- 确定消息订单的结构

```
typedef struct {
        int TaskNum;   /* 接收、处理、分发 */
        char Directory[64];      /* 目录 */
        char FileName[18];       /* 文件名 */
        int SendMode;   /* 处理或分发模式 */
        char Reserve[6];      /* 预留成员 */
}ORDER;

typedef struct{
        long MsgType;      /*  message type  */
        ORDER order;       /*  order  */
}MESSAGE;
```

- 消息队列的创建

消息报文在发送和接收前,必须先创建一个能够被唯一识别的消息队列和数据结构,这个唯一的标识符称为消息队列描述符。使用 msgget(long key, int msgflg)函数可以创建消息队列,参数 key 可以由用户自己定义,参数 msgflg 是消息队列操作权和控制命令的组合。

- 消息发送

消息队列创建后,就可以用 msgsnd(int msgid, void *msgp, size_t msgsz, int msgflg)函数来发送消息了。其中各参数意义如下:

msgid:经 msgget 创建的消息队列描述符。

msgp:指向消息段的指针,该指针指向上面的结构 MESSAGE。

msgsz:msgp 参数指向的数据结构中订单 ORDER 的长度。

msgflg:当消息队列满时,系统要采取的行动。

- 消息接收

使用 msgrcv(int msgid, void *msgp, size_t msgsz, long msgtype, int msgflg)函数可以从 msgid 消息队列中读取一条消息并将其放入消息段指针 msgp 指向的结构。其中 msgid, msgp, msgsz, msgflg 参数的意义同上,msgtype 参数用来指定要求的消息类型:msgtype = 0 时,接收消息队列中的第 1 个消息;msgtype > 0 时,接收消息队列中类型为 msgtype 的第 1 个消息;msgtype < 0 时,接收消息队列中小于

等于 msgtype 绝对值的最低类型的第 1 个消息。

④ 应用效果

表 1 为采用进程间通信和未采用进程间通信两种方法的处理时效对比。

表 1　新增资料在国内通信系统的处理时间

Table 1　New data process time in domestic Meteorological Telecommunication System

方法序号	方法描述	处理时间/s
1	用消息队列来实现进程间通信	58
2	无进程间通信	135

在表 1 中,方法 1 表示通信软件实现时,接收进程、处理进程以及分发进程之间采用消息队列来实现进程间通信,方法 2 表示通信软件的接收进程、处理进程以及分发进程之间不使用进程间通信,各进程获取气象资料的方法均为目录扫描。处理时间表示为气象资料从各观测站发送到卫星主站开始,经通信软件处理,直至分发给用户需要的时间。从表 1 可以看出,采用消息队列实现进程间通信,能将新增资料在国内通信系统中的处理时间缩短为原来的 43%,大大提高了国内新增气象资料的传输效率。

4　小　结

国内气象通信系统新增资料传输软件通过多进程并发来传输新增资料,避免了文件排队等待处理和分发的问题,提高了资料传输的效率,同时通过严格控制并发进程的数目以保证系统的性能。各进程之间使用消息队列来进行通信,缩短了对接收文件的处理和分发时间,避免了因为重复扫描目录造成不必要的耗时;利用消息类型来实现时效要求严格的气象资料的优先级传输和处理,进一步提高了重要资料的传输时效。该软件从 2005 年汛期开始投入业务使用以来,运行稳定,保障了全国新增资料的实时接收和共享,取得了良好的效果。由此可见,在解决了僵尸进程及进程间通信等关键技术问题的情况下,多进程并发技术是一个可在现有环境下大幅度提高资料传输效率十分有效的技术手段。

参 考 文 献

[1] Stevens W R. UNIX Network Programming Volume Interprocess Communications. 2000: 50-100.

[2] 冉茂农,瞿建华,沙利,等. 基于 DVB-S 数据共享平台的 NOAA/ATOVS 资料获取、处理与显示系统. 应用气象学报, 2006,17(4):502-508.

[3] Yves Lepage Paul larrera. UNIX 系统管理员大全. 段剑波,译. 北京:清华大学出版社,2000:156-157.

[4] Robbins K A, Robbins Steve. UNIX Systems Programming: Communication, Concurrency and Threads. 2005:20-24.

[5] 周明德. UNIX/Linux 核心. 北京:清华大学出版社,2004:70-72.

[6] 蔡传俊. UNIX/TCP/IP/NFS 网络编程与应用开发. 北京:海洋出版社,1993:22-24.

[7] 周明天,汪文勇. TCP/IP 网络原理与技术. 北京:清华大学出版社, 2001:186-187.

[8] Stevens W R. UNIX 网络编程(第 1 卷). 北京:机械工业出版社,2004:56-58.

[9] Douglas E Comer. 用 TCP/IP 进行国际互联. 赵刚,译. 北京:电子工业出版社,2001.

[10] Robbins K A, Robbins Steve. UNIX Systems Programming: Communication, Concurrency and Threads. 2005,5:20-24.

[11] 郑彦兴. Linux 环境进程间通信. http://www-128. ibm. com/developerworks/cn/linux. 2004.

[12] 段晓航,刘京志,王凌. UNIX 系统高级程序设计. 北京:中国铁道出版社,2001:40.

[13] 张炯. Unix 网路编程实用技术与实例分析. 北京:清华大学出版社,2002:120-126.

Concurrent Multiprocess Application to Domestic Telecommunication System

Hu Yingmei Shen Wenhai Song Zhiguang

(*National Meteorological Information Center*, *Beijing* 100081)

Abstract

As the national meteorological telecommunication center, Beijing Meteorological Telecommunication System (BMTS) is mainly responsible for the collection and dissemination of national meteorological data. With all kinds of means of exploration and products of weather service teeming, domestic meteorological telecommunication software cannot satisfy the meteorological data real time transmission needs, so meteorological data can not be used in time. Communication software is imperative to improve the ability to exchange information to meet the ever-increasing demand for the exchange of real-time weather information. Along with the computer technology development, the multi-advancements technology also obtained the widespread application and development. By analyzing the causes and treatment of the zombie process, function waitpid( ) is used to avoid the zombie process. Furthermore, corresponding strategy is formulated by analyzing the quit status information and it is made sure that meteorological data exchange accurately, timely and efficiently in BMTS. On the other hand, by comparing many interprocess communication technologies and combining domestic communications features and the actual business, message queue is adopted. Now the processing time of newly data on BMTS decreases to 43% After solving the problems above, multiprocess technology is applied on BMTS, by which the transmission efficiency of domestic meteorological data is improved. This software has been running reliably and stably in the system since 2005 and real-time exchanging and sharing of the nationwide newly meteorological data are ensured.

Key words: newly meteorological data; concurrent multiprocess; zombie process; interprocess communication