基于分布式环境的子进程监控软件设计与实现*

张 虎,黄海于
(西南交通大学 信息科学与技术学院,四川 成都 610031)

摘 要:针对分布式系统环境下,计算资源代理对其子进程监控的方法单一,且不能准确获取子进程运行状态的问题,提出了一种根据子进程的窗口句柄定时检测子进程运行状态的方法。该方法首先根据子进程的不同类型采用不同的方法获取子进程的窗口句柄,然后根据子进程的窗口句柄定时获取子进程的运行状态,最后将传统的等待子进程退出的方法引入到本应用中。运行结果表明,本方法可以及时准确检测出 Windows 环境下子进程的运行状态,并在子进程异常退出时,可以准确地获取其异常退出码。

关键词:分布式系统;Windows 运行环境;代理;子进程的管理和监控;窗口句柄
中图分类号:TP311 **文献标识码**:A **文章编号**:1674-7720(2013)07-0001-04

Design and realization of the subprocess monitoring software based on the distributed environment

Zhang Hu, Huang Haiyu
(School of Information Science and Technology, Southwest Jiaotong University, Chengdu 610031, China)

Abstract: A method based on the window handle of the subprocess real-time detecting the running state of the subprocess is presented, because the method of the computing resource agent monitoring its subprocess is single and it cannot acquire accurately the question of the running state of the subprocess under the distributed system environment. This method gets the window handle of the subprocess first according to the different types of the subprocess adopting the different ways, then gets regularly the running state of the subprocess according to the window handle of the subprocess, finally the traditional method of waiting for the subprocess to exit is introduced to the application. The running results suggest that this method can detect the running state of the subprocess under the Windows environment in time and exactly, and it can get the abnormal exit code accurately when the subprocess exits abnormally.

Key words: distributed system; Windows running environment; agent; regulation and monitoring of the subprocess; window handle

随着高速列车仿真模拟、物联网应用等领域对计算机计算速度要求的不断提高,单个的计算机已无法满足高计算速度的要求。将一个大的计算任务分解成若干个小的计算任务,并利用分布式系统[1]将各计算任务分散到不同的计算机上,以独立进程的形式进行并行计算是一种比较好的解决方法。而对各独立进程的运行状态的实时监控和管理是实现分布式系统高效运行和管理的基础。但是传统的子进程监控只能对子进程是否正在运行或者退出做出判断,无法判断子进程是否处在挂起状态,也无法及时地获取子进程的退出码。

针对上述需求,本文提出了一种根据子进程的窗口句柄来检测当前子进程运行状态的方法,并结合传统的子进程管理监控方法,设计并实现了一种实时的子进程管理监控软件。该软件用于启动和监控分布式系统中任务调度器分配给本机的任务。目前该软件可以检测子进程的三个状态:正常运行、退出、挂起;可以及时的将子进程的退出码[2]反馈给分布式系统的任务调度器,为任务调度器[3]高效利用计算资源提供了一定的依据;并且为开发人员根据进程的异常退出码对程序进行查错提供了方便。

1 传统子进程监控方法

通常 Windows 系统使用 CreateProcess 函数来新建一个子进程。其中 CreateProcess 函数的最后一个参数的类

*基金项目:"十一五"国家科技支撑计划(2009BAG12A01-A01)

型是 LPPROCESS_INFORMATION 结构体，成功创建子进程后，子进程的基本信息就存储在该结构体中。在 Windows API 的定义中，该结构体包含了子进程的进程句柄、进程 ID、主线程句柄和主线程 ID。创建子进程成功后可以通过子进程的句柄和 Windows 提供的 wait[4]系列函数等待子进程或者子进程组变为 signaled 状态，从而立刻获知该进程或进程组退出。

传统的子进程监控方法的好处是：如果子进程确实正常退出，则该方法能够及时地获知子进程已经退出，并且可以通过 Windows 提供的 API 函数 GetExitCodeProcess 获取子进程的退出码。但是在大量的实际应用过程中，该方法暴露出了其不足之处。例如，创建一个有错的（如除 0 错误）MFC 程序，然后通过以上方法启动这个有错的进程，再用 wait 系列函数等待子进程退出。当子进程运行到错误的语句时会弹出一个错误窗口，如图 1 所示。

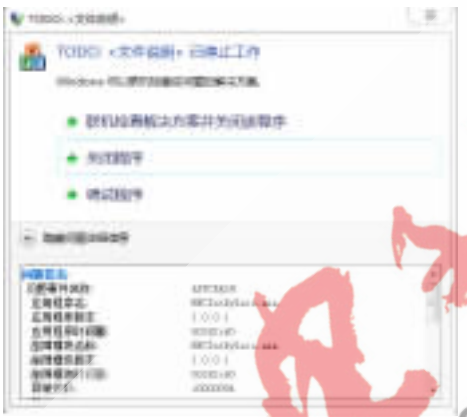

图 1 弹出错误窗口

这时 wait 系列函数没有返回，说明子进程还处在 nonsignaled 状态。用 GetExitCodeProcess 函数获取该子进程的退出码时，得到的退出码为 STILL_ACTIVE，也就是说父进程认为子进程还在正常运行，无法获取子进程的退出码。

如果在分布式系统中出现这种子进程明明已经出错导致无法继续运行，但是其监控系统认为其还在正常运行的情况，会严重影响分布式系统的负载均衡，不利于分布式系统高效的运行和管理。

2 判断子进程的类型

Windows 支持两种类型的应用程序。一种是基于图形用户界面（GUI）的应用程序，另一种是基于控制台用户界面（CUI）的应用程序[5]。

基于控制台的应用程序属于文本操作的应用程序。它通常不能用于创建窗口和处理消息，并且不需要图形用户界面。虽然基于 CUI 的应用程序包含在屏幕上的窗口中，但是窗口只包含文本。命令外壳程序 CMD.exe 是典型的基于 CUI 的应用程序。基于 GUI 的应用程序有一个图形前端程序，它能创建窗口，拥有菜单，可以通过对话框与用户打交道，并且可以使用所有的标准 Windows 组件。

这两种类型的应用程序之间的界限是非常模糊的，所以 Windows 没有提供 API 用来判断一个程序是基于 GUI 的还是 CUI 的。但是可以通过应用程序在运行时加载的动态库[6]来判断应用程序的类型。

user32.dll 和 comctl32.dll 两个模块[7]是 Windows 用户界面相关应用程序接口，包括窗口消息处理，基本用户界面等特性。结合大量的测试得出，基于 GUI 的应用程序在运行时肯定会加载这两个模块，而单纯的基于 CUI 的应用程序在运行时是不会加载这两个模块的。因此，可以通过检测代理新启动的应用程序是否加载了 user32.dll 和 comctl32.dll 这两个模块来区分应用程序类型。

接下来的工作就是如何检测新启动的进程加载了哪些模块。在 Windows API 中提供了枚举一个进程所加载模块句柄的接口：EnumProcessModules 函数。该函数的原型为：

```
BOOL  WINAPI  EnumProcessModules(
  __in          HANDLE       hProcess,
  __out         HMODULE  *   lphModule,
  __in          DWORD        cb,
  __out         LPDWORD      lpcbNeeded
);
```

该函数接收一个进程的句柄，输出该进程所加载的所有模块的句柄数组，并且通过 lpcbNeeded 参数输出所有模块的句柄所占的字节数。因为在启动子进程的时候，肯定能够得到子进程的进程 ID 和进程句柄，所以通过 EnumProcessModules 函数可以方便地得到某个特定的进程所加载模块的句柄。但是通过模块的句柄，还无法直观的得到进程加载的模块的名称。这种情况下，可以通过 vc 提供的另外一个接口：GetModuleFileNameEx 函数来获取各模块的名称。GetModuleFileNameEx 的函数原型为：

```
DWORD  WINAPI  GetModuleFileNameEx(
  __in          HANDLE     hProcess,
  __in          HMODULE    hModule,
  __out         LPTSTR     lpFilename,
  __in          DWORD      nSize
);
```

该函数接收一个进程的句柄和模块的句柄，通过 lpFilename 参数以字符串的形式输出模块的具体名称。进程的句柄在启动进程时就可以获得，模块的句柄就是之前通过 EnumProcessModules 函数获得模块句柄数组。

计算资源上运行的代理通过这两个 API 的配合使用，可以准确获得启动的子进程所加载的模块具体的名称，也就能够确定子进程是否加载了 user32.dll 和 comctl32.dll 两个模块。这样代理就可以确定子进程是基

于 GUI 的应用程序还是基于 CUI 的应用程序。在确定了应用程序的类型之后，根据各种类型应用程序的不同特点，采用不同的子进程监控方法对其进行监控。

3 基于窗口句柄对子进程监控的方法

在 Windows 系统中不论是 GUI 应用程序还是 CUI 应用程序，在程序启动时都会生成一个窗口。不同的是，GUI 应用程序是根据自己的程序需求生成窗口，CUI 应用程序是系统为其加载的一个文本控制台窗口。系统为每一个窗口生成了唯一的标示，即窗口句柄。而且 Windows 提供了一个通过窗口句柄来检测应用程序是否处于挂起状态的 API 函数，该函数的原型为：BOOL IsHungAppWindow(HWND hWnd)。该函数接收一个窗口句柄作为输入参数，并且判断该窗口所属的进程是否处于挂起状态。当进程处于挂起状态时，函数返回 TRUE；当进程处于非挂起状态时，函数返回 FALSE。只要能获取到进程所对应的窗口句柄，就能够通过定时调用 IsHungAppWindow 函数判断 GUI 应用程序是否处于挂起状态。

但是，在子进程创建的过程中，父进程只能获取到该子进程的进程句柄和该进程的主线程句柄，无法获取到子进程所对应的窗口句柄。所以，如何获取子进程所对应的窗口句柄是基于 GUI 的应用程序监控方法的关键。

获取窗口句柄的方法有很多种，本应用中针对 GUI 应用程序和 CUI 应用程序的不同特点采用了不同的方法获取这两种应用程序的窗口句柄。

3.1 获取 GUI 应用程序的窗口句柄

如果应用程序是一个基于 GUI 的应用程序，则操作系统在启动的过程中不会为应用程序创建控制台窗口，而只是加载应用程序。当基于 GUI 的应用程序启动之后，就根据程序自身的需要生成特定的窗口。这样窗口的进程 ID 即为应用程序的进程 ID。

针对 GUI 应用程序的窗口进程 ID 即为应用程序进程 ID 的特点，获取 GUI 应用程序窗口句柄采用的方法是在创建子进程之后遍历系统中所有窗口，在遍历的过程中根据窗口的句柄来获取窗口所对应的进程的 ID；将获取到的窗口进程 ID 与创建的子进程的 ID 进行匹配。如果匹配成功则该窗口就是子进程创建的窗口，可以通过该窗口的句柄调用 IsHungAppWindow 函数来判断该子进程是否处于挂起的状态。

在遍历窗口句柄时，是通过 GetTopWindow 和 GetNextWindow 这两个 API 函数协同工作完成的；而根据窗口句柄来获取窗口所对应的进程 ID 是通过 GetWindowThreadProcessId 函数实现的。具体的实现代码如下：

```
HWND GetWindowHandleByPid(DWORD dwProcessID)
{
HWND h=GetTopWindow(0);
while(h)
{
    DWORD pid=0;
    DWORD dwThreadID=GetWindowThreadProcessId(h,&pid);
    if(dwThreadID!=0)
    {
        if(pid==dwProcessID)
        {
            return h;
        }
    }
    h=GetNextWindow(h,GW_HWNDNEXT);
}
return NULL;
}
```

该函数的输入参数为需要获取窗口句柄的进程的 ID。如果查找成功则返回进程所对应窗口的窗口句柄；如果不成功则返回 NULL。

3.2 获取 CUI 应用程序的窗口句柄

通常情况下，基于 CUI 的应用程序不会创建窗口和处理消息，并且不需要图形用户界面。但是 Windows 系统会为 CUI 的应用程序自动加载一个文本控制台窗口的外壳程序。CUI 应用程序的标准输入输出都是在这个外壳程序中完成的，所以也可以通过判断该控制台窗口是否处于挂起状态来判断 CUI 应用程序的状态，即通过 IsHungAppWindow 函数来判断应用程序是否挂起。

但是，Windows 对 CUI 应用程序的这种处理方式使得 CUI 程序与 CUI 程序的窗口具有不同的进程 ID。这样就不能通过匹配程序进程 ID 和窗口进程 ID 的方法来确定某一个窗口是否属于某一个应用程序。

针对 CUI 应用程序的以上特点，本应用获取 CUI 应用程序窗口句柄的方法是通过检测窗口的标题来确定该窗口是否属于某一个 CUI 程序。采用这种方法的原因是基于 CUI 的窗口标题肯定为该 CUI 程序的绝对路径，并且在分布式计算的环境下，各个任务的子进程可能同名但是肯定是存在于不同的目录下的。所以通过标题来确定一个窗口是否属于 CUI 程序在本应用环境下完全可行。

基于 CUI 子进程的监控方法是在启动子进程之后，根据子进程的绝对路径调用 FindWindow 系统函数来获取该 CUI 子进程的窗口句柄，这样就可以通过定时调用 IsHungAppWindow 函数来检测应用程序的运行状态。

4 与传统方法相结合的子进程监控方法

以上介绍了通过窗口句柄对各种子进程运行状态监控的可行性。但是，基于窗口句柄检测子进程运行状态的方法是定时检测子进程的运行状态。所以如果定时检测的时间较长时，则缺乏好的实时性；如果定时检测的时间较短时，则增加了计算资源的负载。本应用在综

合考虑了以上问题之后，采用了定时检测子进程的运行状态和开辟新的线程等待子进程退出两种手段相结合的方法来监控子进程的运行状态。这样同时确保了检测子进程运行状态的实时性又弥补了传统子进程检测方法在子进程出现挂起状态时无法检测的问题。

在分布式计算环境下，计算资源上的代理启动子进程的时间不确定，并且可能同时管理多个子进程。所以为了方便管理，在本应用中建立了带头结点的子进程信息链表用于同时管理多个子进程。链表的结构体定义如下：

```
struct ProcessInfoList
{
  short int ProjectID;                    //进程工程号
  short int ConditionID;                  //进程工况号
  short int ModuleID;                     //进程模块号
  short int flag;          //进程运行状态。-1 为未响应;
                           //0 为已退出;1 为正在运行
  HWND hProcessWND;              //进程对应的窗口句柄
  int ProcessType;     //进程的类型。1 为 GUI 应用程序;
                                  //0 为 CUI 应用程序
  int SendFlag;
          //用于标示是否已经提示调度器该进程已挂起
  PROCESS_INFORMATION  ProcessInfo;
              //进程的信息，包括进程句柄和进程 ID 等
  DWORD ExitCode;                 //进程退出时的退出码
  TaskHistoryList *next;
};
```

其中 ProjectID、ConditionID、ModuleID 在整个分布式计算系统中确定唯一一个子进程，并且子进程的可执行文件根据这三个值的不同而存放在不同的目录下。SendFlag 变量是用于提示调度器该进程是否已挂起，当该值为 1 时，说明已经提示调度器该进程挂起，不用重复提醒；当该值为 0 时，说明还未提示调度器该进程已挂起。

4.1 定时检测子进程的运行状态

代理启动时会创建子进程信息链表头结点，并会创建定时器 T，定时遍历子进程信息链表。当子进程信息链表中除头结点外没有其他的结点时，则等待下一次定时的到来；当子进程信息链表中除头结点外还有其他进程的信息结点，则首先关闭定时器 T；然后遍历链表各个结点中的窗口句柄信息，根据进程的窗口句柄判断进程是否处于挂起的状态，如果进程处于挂起的状态，则及时将该进程的信息及进程的当前运行状态发送给调度器，如果进程没有处于挂起状态，则继续遍历下一个结点，最后遍历完成之后重新创建定时器 T，定时遍历子进程信息链表。经过大量的实验，最终将定时器的定时时间设为 3 s，即每 3 s 检查所有子进程是否处于挂起状态。

4.2 创建新的线程等待子进程退出

当有任务提交给代理时，代理首先启动相应的应用程序，然后创建新的线程等待子进程的退出，最后判断子进程的类型，获取进程的窗口句柄，为新启动的应用程序创建子进程链表节点。其中等待子进程退出线程中所做的工作有：(1) 根据新启动进程的进程句柄调用 WaitForSingleObject 函数，等待子进程的退出；(2) 当子进程退出后获取进程的退出码，并存放在相应的子进程信息链表结点中；(3) 获取退出码之后，将该子进程的基本信息、子进程的当前状态、子进程的退出码发送给调度器，线程结束。

以上介绍了在 Windows 系统环境下对基于 GUI 和基于 CUI 子进程监控的实现方法。该方法主要是通过定时检测子进程对应的窗口是否挂起以及开辟新的线程等待子进程退出两种手段相结合的方式实现对子进程的监控。虽然通过定时检测窗口是否挂起的方法存在缺乏实时性的问题，但是通过缩短定时时间也可以将时间控制在毫秒级，在绝大多数的分布式计算应用系统中是可以接受的。

该方法解决了传统的子进程检测方法无法检测子进程挂起状态的问题。对子进程运行状态的检测更准确，提高了分布式计算环境下的资源利用率。

参考文献

[1] 葛澈.分布式计算技术概述[J].微电子学与计算机，2012(5)：201-204.

[2] DAVE T. Understanding exit codes[J]. Linux Journal，2010(197)：24-25.

[3] Xie Tao，Qin Xiao. A Security-qriented task scheduler for heterogeneous distributed systems [J]. Lecture Notes in Computer Science，2006(4297)：35-46.

[4] STRVENS W R，RAGO S A. UNIX 环境高级编程(第 2 版)[M].尤晋元，张亚英，戚正伟译.北京：人民邮电出版社，2005：179-182.

[5] RICHTER J，NASARRE C. Windows 核心编程 (第 5 版)[M].葛子昂，周靖，廖敏译.北京：清华大学出版社，2008：69-72.

[6] 高连生，盛柏林.动态链接库在组态软件中的应用[J].工业控制计算机，2010(6)：21-22.

[7] 周超.Windows 和 Linux 动态链接库研究及应用 [D].上海：华东理工大学，2007.

[8] Microsoft. QIsHungAppWindow function (Windows) [OL]. [2012 -11 -28]. http://msdn.microsoft.com/ZH -CN/library/windows/desktop/ms633526(v=vs.85).aspx

(收稿日期：2013-01-03)

作者简介：

张虎，男，1987 年生，硕士研究生，主要研究方向：分布式计算，高性能技术。

黄海于，男，1970 年生，副教授，主要研究方向：分布式计算，云计算。