UNIX 环境下两种进程守护机制的比较分析①

Comparison and Analysis of Two Monitor Mechanisms for UNIX Processes

王伯天　廖建新　王 纯　朱晓民
（北京邮电大学网络与交换技术国家重点实验室 北京 100876）
（东信北邮信息技术有限公司 北京 100083）

摘要：UNIX 系统上的应用程序有时是一些常驻内存的服务进程。在无人职守的环境下，需要采用某种守护机制，以确保任何进程运行异常时，能尽快恢复到正常的服务运行状态。本文对 UNIX 环境下的守护进程进行了研究，分析和总结了两种常用的进程守护机制：即采用 INETD 进行守护和创建轻量级守护进程实现守护，并对其原理及特点做了深入分析。

关键词：守护进程 超级服务器进程 守护策略

1 引言

守护进程（daemon）又叫精灵进程，是在后台运行的特殊进程，是独立于控制终端并且周期性地执行某种任务或等待处理某些发生的事件的一种进程[1]。大多数 UNIX 系统所支撑的服务器都是用守护进程实现的，比如 WEB 服务器的 HTTPD，INTERNET 服务器的 INETD（Internet Daemon），系统作业控制进程 CROND，系统打印进程 LPD 等等[2,3]，而且，UNIX 系统上的一些应用系统通常都被设计成一些常驻内存的服务进程，也需要采用某种守护机制，以保证在无人职守的环境下遇到任何进程运行时异常都能保证进程能尽快的恢复到正常的服务运行状态。正因为此，守护机制的选择及守护进程的设计在整个软件设计中占有至关重要的地位，特别是对于软件的健壮性和可靠性有着重要的保障。

本文就是在对大量 UNIX 系统上应用程序守护进程的研究基础上，分析和总结出了两种常用的进程守护机制，并对其原理及特点做了深入分析。

2 采用 INETD 进行守护

2.1 INETD 工作原理

INETD 是 UNIX 系统提供的一个超级服务器守护进程，它根据配置监听各种网络请求，当请求到来时，启动实际的服务进程进行处理。称其为超级服务器守护进程，是因为它完成了任何被守护的服务进程都需要做的一些公共的工作，如监听套接口，建立连接，创建对应某个连接的实际服务进程等，从而使得被守护的服务进程只需要集中处理该连接上的数据则可。

INETD 的工作过程如下[2]：

（1）INETD 启动时先从配置文件/etc/inted. conf 中读取一行，并取得相关的服务名及相关信息。并根据服务制定的协议生成相应的套接口。

（2）从文件/etc/services 中查找服务所用的端口号，并将套接口绑定到该端口上。

（3）若使用 TCP 协议，此处需要调用 listen 函数，否则转下一步。

（4）调用 select 函数进行监听，一旦 TCP 类型的套接口上收到连接请求或 UDP 类型的套接口上有数据到达时，select 函数返回。

（5）调用 fork 生成子进程，若服务使用 TCP 协议，其服务标志是"nowait"，则父进程在关闭套接口后回到第 4 步去继续监听；如果服务使用 UDP 协议，则其

① 基金项目：国家杰出青年科学基金（No. 60525110）；国家 973 计划项目（No. 2007CB307100，2007CB307103）；新世纪优秀人才支持计划（No. NCET－04－0111）；电子信息产业发展基金项目（基于 3G 的移动业务应用系统）；电子信息产业发展基金项目（基于内容的综合通信网络计费平台）

服务标志为"wait",父进程需要等待子进程结束后,才能返回。

(6) 子进程调用三次 dup2 函数把套接口复制到文件描述符为 0,1,2 的文件,然后关闭套接口。

(7) 调用 exec 函数调用相应的服务程序取代子进程。

由于网络底层通信的许多基本工作已经由守护进程 INETD 完成了,所以服务进程就可以集中面向于具体服务的处理了。而且,由于 INETD 在启动服务进程之前已经将套接口映射成文件描述符,所以服务进程就可以像操作文件一样去操作套接口了。

2.2 实现方法

从 INETD 的基本原理可知,守护进程 INETD 与被守护进程之间不涉及任何进程间通信问题,进程关系比较独立,而且,INETD 进程完成了套接口的监听和读写等操作,并将套接口复制到标准输入和输出上,因此服务进程不需要实现这些操作,仅仅象操作标准输入输出一样操作套接口则可,所以服务进程的开发显得相对简单了许多。此外,由于 INETD 与各服务进程的守护关系是用一些配置文件来建立的,所以实现由 inetd 守护服务进程就非常容易了。

一般,要实现服务进程由 INETD 守护,需要如下步骤[1]:

(1) 修改/etc/services 文件,增加新的应用程序的服务名和端口号,如下所示:

服务名 端口号/网络类型

ftp 23/tcp

(2) 修改/etc/inetd. conf 增加一行新的配置

#service name socket type protocol wait/nowait userprogram server program

ftp stream tcp nowait cxsvr /home/cxsvr/bin/Ftpd

(关于该文件的配置说明,此处不做详细介绍。)

(3) 修改/etc/environment 文件,增加应用程序所需的环境变量

FTPLOGDIR = /home/cxsvr/log

如果应用程序不需要环境变量,可以略去此步。

至此,已经配置完成了 INETD 与服务进程的守护关系,如果服务进程已经开发完毕,那么就可以启用该守护关系了。启用新增守护关系的方式有两种:一种是先用#stopsrc - s inetd 将 inetd 进程停止,然后用#startsrc - s inetd 启动 INETD 进程,该进程在启动时会重新读取相关配置文件到内存,并建立守护关系。另一种是用#refresh - s inetd 命令通知 INETD 配置文件已经改变,启用新的配置文件。

3 创建轻量级守护进程实现守护

由于 UNIX 服务器上许多应用系统都要求很高的稳定性和可靠性,因此也需要更加完善和灵活的守护机制。通常的做法是应用系统创建一个轻量级精灵进程,各个应用进程与该精灵进程之间采用某种通信策略来实现进程的守护。这种守护进程的实现包含两个方面的问题:一是守护进程基本部分的实现,另一方面是守护进程与各应用进程之间采用何种通信手段去实现守护,也就是守护策略的问题。

3.1 守护进程的编写

不同 UNIX 系统上守护进程的编程实现是不一致的,但都遵循一些基本的编程规则,区别仅仅在于具体的实现细节上。守护进程的编程规则可以总结为如下几个步骤[1,4]:

(1) 在后台运行。调用 fork,然后使父进程 exit,这样做实现了以下几点:第一,如果该守护进程是由 shell 命令启动的,那么使父进程终止使得 shell 认为这条命令已经执行完成。第二,子进程具有一个新的 ID,但继承了父进程的进程组 ID,这就保证了该进程不是进程组的首进程,为下一步做好准备。

(2) 脱离控制终端、登陆会话和进程组。通过调用 setsid 创建一个新的对话期,使该守护进程成为新对话期的首进程,并且已经脱离终端。

(3) 改变工作目录。进程工作是,其工作目录所在的文件系统是不能被卸下的,一般需要将工作目录改变到根目录。通过调用:chdir("/"); 可以简单的实现。

(4) 重新设置文件的创建掩码。进程从创建它的父进程那里继承了文件创建掩码,它可能修改守护进程所创建的文件的存取位,为防止这一点,将文件创建掩码消除:umask(0);

(5) 处理 SIGCHLD 信号。处理 SIGCHLD 信号是很重要的一点,因为如果父进程不等待子进程结束,子进程将成为僵尸,从而占用系统资源。UNIX 系统提供了功能强大的信号处理函数,可以使父进程接受并处理

异步的各种信号。对于 SIGCHLD 信号最简单的处理方式就是忽略它，signal(SIGCHLD,SIG_IGH)[1,5]。

(6) 采用特定的守护策略，实现对应用进程的守护。

3.2 守护策略

由于守护进程基本部分的实现已经有了一些成熟的编程原则可以参照，而且在不同的环境下实现也比较容易。因此，守护进程的功能在很大程度上就体现在守护策略的选择和设计上。下面介绍两种工程应用中常见的守护策略，并进行简单的比较。

(1) 文件加锁。UNIX 系统中通过 fcntl 等调用可以对文件实施加锁操作，系统提供两种类型的锁：共享读锁(F_RDLCK)和独占性写锁(F_WRLCK)[1,5]。按照加锁操作的规则，多个进程可以给一个文件加多个共享读锁，但是如果有了独占写锁后，其它进程不可加任何锁。此外，按照锁的隐含继承和释放规则[1] – 当一个进程终止时，它所占有的所有锁全部被释放。因此，我们可以巧妙的利用文件锁与进程的这种关系，实现进程的守护。

被守护进程在初次启动时在指定目录下创建一个文件，一般都为隐藏文件且文件名一般都定义为进程名，往往还需要将进程 PID 写入该文件。之后，被守护进程对该文件加独占锁，此时，我们称该文件为该进程的锁文件。根据锁与进程的关系可知，锁文件会被对应进程独占，直到进程退出后自动释放该锁。守护进程在运行期内周期性的去读取该文件并通过尝试加锁的方式来判断被守护进程是否存活，如果文件存在并加锁失败，则说明被守护进程在运行中；反之，说明被守护进程已经不在线，此时，守护进程立刻启动该被守护进程，并由被守护进程重新生成锁文件。

该种守护策略的实现非常简单，首先守护进程需要在主循环中周期性的调用 fcntl 尝试对各个被守护进程所对应的锁文件实施加锁操作，如果加锁成功，需要守护进程重启该进程；反之，被守护进程正常运行中，守护进程进入下一个轮询周期。

(2) 进程间通信—心跳信息。心跳信息被定义为一种内容简单、且具有固定发送频度的信息。UNIX 系统上常采用这种进程间传递心跳信息的机制来实现服务的可用性测试以及进程的状态判断等。采用该种方式实现进程守护的方法如下：

首先，选择守护进程与被守护进程通信的方式，协商心跳消息格式，确定心跳最大周期。通常选择 FIFO 作为进程间通信方式，其主要的优势在于实现简单。心跳信息仅仅由进程可执行程序名组成，心跳周期可以灵活配置，如 10 秒。

其次，守护进程在启动时建立通信渠道(FIFO)，并初始化被守护进程队列。FIFO 的命名及存放位置需要守护进程和被守护进程协商，通常 FIFO 命名采用"[守护进程名]_[被守护进程名].fifo"的方式，存放路径是应用系统主目录下的 fifo 目录。守护进程还需要将被守护进程的信息记录下来，可以采用如下结构：

```
typedef struct TAppNode{
char   path[50];   //程序运行的路径
int    tagOfExist; //程序存在的标志 0:不存在;1:存在
int    interval;   //报活时间间隔
string fifoName;   //与该进程对应的管道名
pid_t  pid;        //进程号,如果该进程没有运行,则为 0
time_t timeStamp;  //最近一次的活动时间
struct TAppNode * pNext;
}
```

如果被守护进程有多个，则构成一个队列，在此定义该队列为 pAppList。

然后，守护进程进入主循环。守护进程在主循环中需要完成两件事：其一，监听各个 fifo 描述符是否有心跳消息送达。如果有消息送达，则读取该消息，并从 pAppList 中查找对应的被守护进程节点，找到后将该节点的 timeStamp 更新为当前时间。其二，遍历 pAppList，判断当前时间减去上次活动时间(timestamp)是否大于进程心跳周期(interval)，如果大于，说明心跳超时，此时需要守护进程重启该被守护进程。

采用该种策略实现守护时，需要在被守护进程中增加与守护进程通信的功能，也就是被守护进程主循环中每隔一定周期向 FIFO 写心跳信息，也正因为此，可以及时的检测出被守护进程在运行中长时间阻塞到某个操作上的情况，因此该种守护策略被广泛的应用在工程实践中。

(3) 两种守护策略的比较。以上两种守护策略是工程应用中比较常见的，由于其采用的处理机制不同，

因此体现出了不同的优点和缺点。首先由于文件加锁的策略本质上基于文件锁与进程的依赖关系，也就是只有在进程终止后文件锁才被释放，因此对于那些功能复杂的服务进程来说，一旦阻塞在某个环节，该进程就无法被及时恢复。而采用心跳的方式，就可以很好的避免此问题，灵活及时的实现守护。其次，从守护策略的实现难度来说，文件加锁的策略只需遍历各个被守护进程的锁文件并尝试加锁则可，其中仅仅涉及到对 fcntl 的系统调用。采用心跳通信的守护策略，需要建立通信管道，而且在处理周期需要不断的访问通信管道，涉及到多个系统调用，而且从工程应用来看，这些有名管道往往会由于系统或人为的原因被毁坏，因此在实现细节上需要许多异常的判断和处理，实现难度较高。同时，从被守护进程的编程来看，文件加锁策略中被守护进程只需要在程序入口生成锁文件则可，其它地方不需做任何特殊处理，而采用心跳通信的策略，需要读写通信管道，并处理各种异常，实现上难度较高。最后，由于两种策略执行的行为都是简单的遍历，因此执行的性能和效率并无多大差别，所以对于那些功能简单的服务进程来说，采用任意一种策略都可以满足需要。

4 两种进程守护机制的比较分析

应用程序服务进程采用 INETD 进行守护，有许多的优点：首先可以充分的利用 INETD 进程提供的底层通信支持，使得应用程序不需要考虑套接口处理的复杂问题，仅仅用普通的文件 I/O 调用就可以实现套接口的读写操作。其次，由于该种守护方式下，套接口的监听是由 INETD 来完成的，服务进程由 INETD 来启动而不会常驻内存，节省了系统资源。最后，由于 INETD 是由操作系统提供的一个守护进程，运行稳定、可靠，因此采用该种方式建立的守护关系也就有无与伦比的稳定性和可靠性。

当然，由于该种方式的简单性也就决定了其功能必然会受到限制。该种机制比较适合于一些非实时运行的服务进程的守护，而且不能对套接口做特殊的处理设置。对于那些套接口操作复杂，且需要实时处理应用需求的服务进程来说，该种方式就难以胜任了。所以在一些大型的应用系统中一般很少采用该种方式。

通过创建轻量级守护进程实现守护，就等于是为被守护进程量身定制了守护机制，可以依据被守护进程的特点来灵活的选择守护策略，功能之强大自然是无与伦比的。而且，这种守护机制下被守护进程可以独立开发，独立运行，具有很好的灵活性。但是，该种机制也有其劣势：首先守护进程的编程比较复杂，特别是对不同守护策略的实现和信号的处理有较高的要求，因此也就注定了这种守护机制的稳定性和健壮性相对来说差一些。其次，守护进程会作为一个独立进程运行在系统中，必然会占用系统资源，包括 CPU、内存、文件描述符等，这在对系统资源要求比较高的系统内是绝对不允许的。

5 结束语

从实际的应用来看，采用 INETD 进行守护的方式较多的应用在一些系统级的服务上，如 FTP，TIME，DNS 等，大多数应用程序，如电信级的各种应用系统较多的采用第二种守护机制，通过在系统中创建一个轻量级的守护进程，确定好完善的守护策略，实现系统各个应用进程的守护和服务保障，而且，可以通过添加一些操作维护功能，实现守护关系的动态生成和管理。这种方式的守护，已经在北邮国家重点实验室移动智能网业务平台和彩铃业务平台等多个系统中得到广泛应用和证实。

参考文献

1 W. RICHARD STEVENS 著，尤晋元等译，UNIX 环境高级编程，北京：机械工业出版社，2002. 2：61－62，239－244.

2 张哲、王艳平，利用 inetd 管理守护进程，中国金融电脑，2001 年第 12 期：46－49.

3 （美）Robin Burk 等著，前导工作室译，UNIX 技术大全，2000. 3 499－503.

4 谢敬东，守护进程浅析，电脑信息技术，2003 年第 2 期：61－63.

5 （美）W. RICHARD STEVENS 著，杨继张译，UNIX 网络编程（第二版）第 2 卷：进程间通信，北京：清华大学出版社，2000. 3：45－47.