线程调用进程设计文档

变更说明

日期	版本	变更位置	变更说明	作者
2013-08-04	0.0.1		创建初始文档	赖亿

本文档由于工期没有在代码中实现

目录

1 目标

取代原来的设计，dbproxy 给 zabbix agent 发 socket 消息，然后 zabbix agent 再调用脚本的方式。改用线程直接调用脚本的方式。这样简化设计，也简化了运维的负担

事实上，线程调用进程在 pxc（galera）中使用。

2 线程调用进程的风险和预防

2.1 风险

一般 linux 编程准则不提倡线程调用进程，因为很容易导致进程死锁。
参考：http://stackoverflow.com/questions/6078712/is-it-safe-to-fork-from-within-a-thread
http://blog.csdn.net/zhenjing/article/details/4885816 文章中的“准则 3 ：多线程程序里不准使用 fork”
举例如下：

```
void* doit(void*) {
      static pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
      pthread_mutex_lock(&mutex);
      struct timespec ts = {10, 0}; nanosleep(&ts, 0);     //   睡 10 秒
      pthread_mutex_unlock(&mutex);
      return 0;
 }

int main(void) {
 pthread_t t;
 pthread_create(&t, 0, doit, 0);
    if (fork() == 0) {
            // 子进程
          // 在子进程被创建的瞬间, 父的子进程在执行 nanosleep 的场合比较多
           doit(0); return 0;
    }
  pthread_join(t, 0);
 }
```

在这种情况下会死锁

- *线程里的 doit() 先执行.*
- *doit 执行的时候会给互斥体变量 mutex 加锁.*
- *mutex 变量的内容会原样拷贝到 fork 出来的子进程中(在此之前, mutex 变量的内容已经被线程改写成锁定状态).*
- *子进程再次调用 doit 的时候, 在锁定互斥体 mutex 的时候会发现它已经被加锁, 所以就一直等待, 直到拥有该互斥体的进程释放它(实际上没有人拥有这个 mutex 锁).*
- *线程的 doit 执行完成之前会把自己的 mutex 释放, 但这是的 mutex 和子进程里的 mutex 已经是两份内存. 所以即使释放了 mutex 锁也不会对子进程里的 mutex 造成什么影响.*

上面是使用 mutex 会死锁，其实如果用了一些通用函数，比如 malloc，printf 也可能导致死锁。比如 malloc,因为 malloc 自身的实现依赖一个维持自身固有 mutex。

解决方式之一是用 **pthread_atfork（），**这个在函数在 apue 经典的书你讨论了这个函数，但是应该也不靠谱
“使用 pthread_atfork 函数, 在即将 fork 之前调用事先准备的回调函数, 在这个回调函数内, 协商清除进程的内存数据. 但是关于 OS 提供的函数(例:malloc), 在回调函数里没有清除它的方法. 因为 malloc 里使用的数据结构在外部是看不见的. 因此,pthread_atfork 函数几乎是没有什么实用价值的.”

2.2 预防方法

预防的方法很简单，做到下面几点
1. 回调函数（对应上面例子 doit 函数）仅仅在进程中被调用，在线程中不被调用
2. 回调函数中数据局部化：比如不涉及全局变量，mutex，甚至不用 malloc.
3. 线程 fork 的进程的数据通信通过 pipe 方式。

2.3 验证上述结论

通过查看 pxc（galera）源代码来验证上述想法，我看的源代码版本为 Percona-XtraDB-Cluster-5.5.24
pxc 做 sst 是会用到，线程最终调用 posix_spawnp 系统函数来调用脚本，脚本的输出通过 pipe 传递给调用它的线程。它没有考虑脚本超时问题

可以看出 posix_spawnp 系统函数其实满足预防方法上面的条件。

具体的调用堆栈看“pxc 调用命令.c”

3 线程调用进程的方案说明

推荐用方案 1

3.1 方案一

这个方案调用脚本有 timeout 控制，当脚本超时后，脚本会自动被 kill 调。
主要参照 zabbix agent 的代码，稍加改造就可
具体在下载 zabbix 源代码，看 zbxexec 目录下面的 execute.c

线程调用脚本为柱塞式的，格式
zbx_execute(const char *command, char **buffer, char *error, size_t max_error_len, int timeout)

Parameters: command - [IN] command for execution
buffer - [OUT] buffer for output, if NULL - ignored
error - [OUT] error string if function fails
max_error_len - [IN] length of error buffer

Return value: SUCCEED if processed successfully, TIMEOUT_ERROR if timeout occurred or FAIL otherwise

zbx_execute 函数中实现了：
1.设置 alarm 信号，超时时间到了触发 alarm 信号，通过 alarm 信号来杀脚本。
2.fork 子进程，然后通过 execl 系统调用执行脚本，fork 前先建立和主进程的 pipe，这样可以得到脚本的结果。这个 pipe 其实是主进程的所有线程都能看到，但我们实现时只会有其中一个线程读 pipe.

这个 fork 子进程的这部分代码是局部的，不可能被其他线程调用，而且数据也是局部化，所以没有线程调用进程导致死锁的问题

3.主进程的一个线程 wait 子进程结束。有两种结束，一种子进程自然结束，另一种收到 alarm 信号（表示脚本超时）结束

问题：
设置信号是主/子都设置了，为啥？子进程设置了不就行了吗？

3.2 方案二

直接调用脚本得到返回结果，但是没有超时控制。这时候超时控制可以另外写一个程序，程序来实现超时控制，比如用 c 写一个 runcmd 的执行程序，里面完成执行脚本并带超时控制功能。我们 dbproxy 直接调用 runcmd 这个程序

这个方案可参考 Percona-XtraDB-Cluster-5.5.24 源码中的 wsrep_utils.cc，它实现了 运行 os 命令“process()” ，通过 pipe 返回 os 数据“pipe()”， 柱塞式等待脚本完成 “wait ();”
如何使用看我写的“pxc 调用命令.c”