目　录

1 概述

本监控程序用来监控 RedHat Linux 主机的系统状况，包括 CPU 负载、内存使用、网络状况、服务端口、磁盘空间等。本程序基于 C/S 结构，全部用 Perl 实现。基本原理是，Client 运行在各个需要监控的主机上，并起一个 Socket 端口。Server 定期轮询各个 Client，根据配置文件里的选项发送扫描命令，取得客户端的状态，若有异常则发送邮件报警。

这个程序属于偶的业余作品，只花了 1 天半的时间写成。Perl 的优点就是快速开发，且自身有强大的类库，可实现很复杂的功能。这个程序除了 Net::SMTP 模块外，没有使用任何外部类库。

代码仅做参考，可以修改它们作为己用。程序运行稳定，目前已监控了偶们广州公司的上百台 Linux 服务器。但代码至少有如下不足：

1. 没写任何安全控制的代码，包括进程 ID 切换，chroot，Socket 会话加密及认证等。所以客户端默认只在内网 IP 上监听，且不要以 root 运行。
2. 客户端检查系统状态主要使用了外部系统调用，其实大部分系统状态，都可从 POSIX 函数或 Linux 自身的状态表里获取（如/proc 下的文件）。偶不想花时间去研究那些，直接 system call 了。

2 监控 Server

```
#!/usr/bin/perl
use strict;
use IO::Socket;
use POSIX qw(:signal_h WNOHANG setsid);
use Net::SMTP;
use Fcntl qw(:DEFAULT :flock);

# 电子邮件地址，用来接受报警
my @emails = (
    sa@sample-inc.com,
```

dba@sample-inc.com,
其他 email 地址
);

```
# 程序运行的主目录，这里是写死的，需要修改成自己的目录，或写成配置文件
my $rundir = '/home/afoo/monsvr';
# 程序运行的 PID 文件
my $pid_file = $rundir . "/monsvr.pid";
# 配置文件，定义各客户端的 IP 及扫描参数
my $cfg_file = $rundir . "/monsvr.cf";
# 2 个日志文件，错误日志及运行日志
my $err_log = $rundir. "/monsvr.err";
my $log_file = $rundir. "/monsvr.log";
# 客户端的运行端口
my $agent_port = 7780;
# 主循环退出的条件，等于 0 不退出，大于 0 退出
my $DONE = 0;
# 每 10 分钟执行一次扫描
my $scan_inter = 10;
# 最后一次扫描的时间
my $last_scan_time = 0;
# 记录子进程 ID 的状态表
my %status;
# MTA 主机的 IP 及端口，用来发送报警邮件。
# Linux 默认安装了 sendmail，打开它即可。
my $mtahost = '127.0.0.1';
my $mtaport = 25;

# 安装自己的信号处理器
# 子进程退出时，从状态表里删掉子进程 ID
$SIG{CHLD}=sub {while((my $child=waitpid(-1,WNOHANG))>0){delete $status{$child}}};
# SIGTERM 或 SIGINT 信号导致程序退出
$SIG{TERM}=$SIG{INT}=sub {$DONE++};
# 发送 HUP 信号可让程序 reload 自身（kill –HUP `cat monsvr.pid`）
$SIG{HUP}=\&do_hup;
# die 和 warn 调用时，将输出重定向到日志文件
$SIG{__DIE__}=\&log_die;
$SIG{__WARN__}=\&log_warn;

# 获取进程状态，首先从 PID 文件里获取已在运行的进程 ID（如果有的话），如果 kill 0 =>$pid
返回真，则表示进程已在运行，服务拒绝启动。如果 PID 文件不可写，服务也拒绝启动。
```

```
if (-e $pid_file)
{
	open (PIDFILE,$pid_file) or die "[EMERG] $!\n";
	my $pid=<PIDFILE>;
	close PIDFILE;

	die "[EMERG] process is still run\n" if kill 0 => $pid;
	die "[EMERG] can't remove pid file\n" unless -w $pid_file && unlink $pid_file;
}

# 程序进入后台，并将自身的进程 ID 写入 PID 文件。
open (HDW,">",$pid_file) or die "[EMERG] $!\n";
my $pid=daemon();
print HDW $pid;
close HDW;

# 主循环
while (!$DONE)
{
# 如果距上一次扫描的时间间歇大于$scan_inter 定义的分钟，并且没有扫描子进程存在，则启动一个扫描子进程
	if ( (time - $last_scan_time > $scan_inter * 60) && ! %status ) {

# 启动扫描后，将最后一次扫描的时间，更新为当前时间
		$last_scan_time = time;

# 设置信号掩码，屏蔽前面重载的几个信号，防止这几个信号在 fork 的时候进入
		my $signals = POSIX::SigSet->new(SIGHUP,SIGINT,SIGTERM,SIGCHLD);
		sigprocmask(SIG_BLOCK,$signals);

# fork 子进程
		my $child = fork();
		die "can't fork $!" unless defined $child;

# 父进程里取消信号掩码
		if ($child) {
			$status{$child} =1;
			sigprocmask(SIG_UNBLOCK,$signals);

# 子进程里先将 HUP，INT，TERM，CHLD 信号恢复成默认，然后也取消信号掩码。
# 然后子进程调用 do_scan()函数进行扫描和报警，处理完后就写日志并退出
```

```
        } else {

            $SIG{HUP} = $SIG{INT} = $SIG{TERM} = $SIG{CHLD} = 'DEFAULT';
            sigprocmask(SIG_UNBLOCK,$signals);

            my $results = do_scan();
            do_warn($results) if @$results;
            write_log("[$$]",">>>All scan finished<<<");

            exit 0;

        }
    }

# 父进程休眠 10 秒，并继续循环
    sleep 10;
}

#------------------
# 下面定义子函数
#------------------

# 使程序进入后台的函数，原理很简单，就是 fork 一个子进程，父进程 die 掉，子进程调用
setsid()使自己成为进程组的领导。然后重定向 3 个标准 I/O 设备到/dev/null。
sub daemon
{
    my $child = fork();
    die "[EMERG] can't fork\n" unless defined $child;
    exit 0 if $child;
    setsid();

    open (STDIN, "</dev/null");
    open (STDOUT, ">/dev/null");
    open (STDERR,">&STDOUT");

    chdir $rundir;
    umask(022);
    $ENV{PATH}='/bin:/usr/bin:/sbin:/usr/sbin';

    return $$;
}
```

```
# 写日志的函数
sub write_log
{
	my $time=scalar localtime;
	open (HDW,">>",$log_file);
	flock (HDW,LOCK_EX);
	print HDW $time,"    ",join ' ',@_,"\n";
	flock (HDW,LOCK_UN);
	close HDW;
}

# 当调用 die 时，会执行这个函数。也就是先将异常消息写入错误日志，再真正的 die。
sub log_die
{
	my $time=scalar localtime;
	open (HDW,">>",$err_log);
	print HDW $time,"    ",@_;
	close HDW;
	die @_;
}

# 当调用 warn 时，会执行这个函数。
sub log_warn
{
	my $time=scalar localtime;
	open (HDW,">>",$err_log);
	print HDW $time,"    ",@_;
	close HDW;
}

# 扫描函数
sub do_scan
{
# 先读取配置文件，获取要扫描的 IP，以及扫描哪些选项
	my $scan_cfg = get_config();
# 这个数组用来记录扫描结果
	my @results;

# 在 for 循环里逐台扫描
	for my $hid (keys %{$scan_cfg}) {
```

```
# 在 eval 里执行扫描，并设置超时 30 秒。如果 30 秒内客户端未返回结果（客户端所在的主机负载很重时，可能会这样），则记录扫描异常的结果。
        eval {
            local $SIG{ALRM} = sub {die "Scan Timeout,$scan_cfg->{$hid}->{IP} is wrong\n"};
            alarm 30;
            my $re = scan_a_host($scan_cfg->{$hid});
            @results = (@results,@$re);
            alarm 0;
        };

        push @results,"Scan Timeout,$scan_cfg->{$hid}->{IP} is wrong" if $@;
    }

    return \@results;
}

# 单独扫描某台机的函数
sub scan_a_host
{
        my $host = shift;       # a hash ref
        my @results;

# 创建到客户端的 socket
        my $sock=IO::Socket::INET->new(PeerAddr => $host->{IP},
                                       PeerPort => $agent_port,
                                       Proto    => 'tcp');
# 如果创建 socket 不成功，则说明客户端的监听端口可能 down 了
        unless (defined $sock) {
            push @results, "$host->{IP}: monitor client seems down";
            warn("[WARN] $host->{IP}: monitor client seems down\n");
            return \@results;
        }

        write_log("[$$]", "prepare to do_scan for $host->{IP}");

# 把配置文件里定义的扫描选项，发送到 client
        for my $key (keys %{$host}) {
            print $sock "$key    $host->{$key}\n"
        }
```

```
# 发送完后关闭写 socket，这步很重要，否则 client 不知道 server 已写完，会阻塞在那里等待，并造成和 server 的交互阻塞。
        $sock->shutdown(1);

# 在 while 循环里读取客户端返回的扫描结果
        while(<$sock>) {
            chomp;
            push @results,$_;
        }

# 关闭 socket 并返回结果给调用者
        $sock->close;
        return \@results;
}

# 该函数用来读取配置文件，并将结果放入一个 Hash。纯文本处理，没有特殊的技巧。请对照配置文件的格式阅读该函数。
sub get_config
{
    my %config;
    open (HDR,$cfg_file) or die "[EMERG] can't open cfg_file: $!\n";

    while(<HDR>)
    {
        next if /^$/;
        next if /^\s*\#/;

        if ( my ($hid) = /\[HOST (\d+)\]/ ) {

            while (<HDR>) {

                next if /^$/;
                next if /^\s*\#/;
                last if /\[\/HOST\]/;
                chomp;

                my ($cfg_key,$cfg_value) = split /=/;
                $cfg_key =~ s/^\s+|\s+$//g;

                $cfg_value =~ s/\#.*$//;
```

```
                $cfg_value =~ s/^\s+|\s+$//g;
                $cfg_value =~ s/^\"|\"$//g;
                $cfg_value =~ s/^\'|\'$//g;

                $config{$hid}->{$cfg_key} = $cfg_value;
            }
        }
    }

    close HDR;
    return \%config;
}

# 告警函数，若扫描有异常，则一方面写入日志，另一方面发送电子邮件报警
sub do_warn
{
    my $warns = shift;

    for (@$warns) {
        write_log($_);
    }
    sendmail($warns);
}

# 发送电子邮件的函数
sub sendmail
{
        my $msg = shift;
        return unless @emails;

        my $smtp = Net::SMTP->new('Host'=>$mtahost,'Port'=>$mtaport);

        if(defined $smtp)
        {
                $smtp->mail("monitor\@sample-inc.com");
                $smtp->recipient(@emails);
                $smtp->data();

                $smtp->datasend("From: Server Monitor <monitor\@sample-inc.com>\n");
                $smtp->datasend("To: SA Team <sa\@sample-inc.com>\n");
                $smtp->datasend("Subject: Server Warnings\n");
```

```
                $smtp->datasend("\n");

                for (@$msg) {
                        $smtp->datasend("$_\n");
                }

                $smtp->dataend();
                $smtp->quit;

        } else {
                warn("[WARN] Can't connect to SMTP host\n");
        }
}

# 发送 SIGTERM 信号，killall 子进程，并 sleep 直到所有子进程都退出
sub kill_children
{
        kill TERM => keys %status;
        sleep while %status;
}

# 在遇到 SIGHUP 信号时，执行本函数 reload 自身
sub relaunch
{
        chdir $rundir;
        unlink $pid_file;
        exec 'perl','monsvr';
}

# 处理 SIGHUP 信号
sub do_hup
{
        warn "[INFO] received SIGHUP,prepare to reload...\n";
        kill_children();

        relaunch();
        die "[EMERG] reload failed\n";
}

__END__
```

3 监控 Client

```
#!/usr/bin/perl
use strict;
use IO::Socket;
use POSIX qw(:signal_h WNOHANG setsid);
use Fcntl qw(:DEFAULT :flock);

# 如下是扫描系统的外部命令，如前所述，你也可以改成系统内部调用
# 请自行调整外部命令的路径
my $NC = '/usr/bin/nc';
my $CAT = '/bin/cat';
my $PING = '/bin/ping';
my $NETSTAT = '/bin/netstat';
my $UPTIME = '/usr/bin/uptime';
my $IFCONFIG = '/sbin/ifconfig';
my $DF = '/bin/df';
my $FREE = '/usr/bin/free';

# 程序运行目录
my $rundir = '/home/afoo/monagt';
# PID 文件及配置文件
my $pid_file = $rundir . "/monagt.pid";
my $err_log = $rundir. "/monagt.err";
my $log_file = $rundir. "/monagt.log";
# 监听端口
my $agent_port = 7780;
# 主循环退出的条件，跟 server 一样
my $DONE = 0;
# 记录子进程的状态表
my %status;

# 重载信号处理器，跟 server 一样
$SIG{CHLD}=sub {while((my $child=waitpid(-1,WNOHANG))>0){delete $status{$child}}};
$SIG{TERM}=$SIG{INT}=sub {$DONE++};
$SIG{HUP}=\&do_hup;
$SIG{__DIE__}=\&log_die;
$SIG{__WARN__}=\&log_warn;

# 运行前先获取进程状态，跟 server 一样
```

```
if (-e $pid_file)
{
	open (PIDFILE,$pid_file) or die "[EMERG] $!\n";
	my $pid=<PIDFILE>;
	close PIDFILE;

	die "[EMERG] process is still run\n" if kill 0 => $pid;
	die "[EMERG] can't remove pid file\n" unless -w $pid_file && unlink $pid_file;
}

# 程序进入后台
open (HDW,">",$pid_file) or die "[EMERG] $!\n";
my $pid=daemon();
print HDW $pid;
close HDW;

# 获取内外网 IP。如前所述，监控客户端为了安全起见，运行在内网 IP 上。这里假设内网
IP 段是 192.168.xx.xx。如果不是，请修改成自己的 IP 格式。
# $inner_addr 是内网 IP，$exter_addr 是外网 IP。如果没有外网 IP，则都使用内网 IP。
my @ifconfig = `$IFCONFIG`;
my ($inner_line) = grep {/inet addr\:192\.168\./} @ifconfig;
my ($exter_line) = grep {/inet addr\:/ && ! /192\.168\./ && ! /127\.0\.0\./} @ifconfig;
my ($inner_addr) = $inner_line =~ /inet addr\:(\d+\.\d+\.\d+\.\d+)/;
my ($exter_addr) = $exter_line =~ /inet addr\:(\d+\.\d+\.\d+\.\d+)/;
$exter_addr = $inner_addr unless $exter_addr;

# 创建一个基于 TCP 的监听 socket，监听在内网地址的$agent_port 端口
my $listen_socket = IO::Socket::INET->new(
                                          LocalAddr => $inner_addr,
                                          LocalPort => $agent_port,
                                          Listen      => SOMAXCONN,
                                          Proto       => 'tcp',
                                          Reuse       => 1,
                                          Timeout     => 30,
                                         );
die "[EMERG] can't create socket: $@\n" unless defined $listen_socket;

# 主循环
while (!$DONE)
{
# 如果没有 accept 到 server 端发过来的请求，则 next 过去
```

```
    next unless my $sock = $listen_socket->accept;

# 如果有请求，则 fork 子进程处理这个请求。fork 之前先设置信号掩码，跟 server 一样。
    my $signals = POSIX::SigSet->new(SIGHUP,SIGINT,SIGTERM,SIGCHLD);
    sigprocmask(SIG_BLOCK,$signals);

    my $child = fork();
    die "[EMERG] can't fork $!\n" unless defined $child;

# 在父进程里取消信号掩码，并关闭连接 socket
    if ($child) {
        $status{$child} = 1;
        sigprocmask(SIG_UNBLOCK,$signals);

        $sock->close or die "[EMERG] can't close established socket\n";

# 在子进程里先恢复默认信号处理器，再取消信号掩码
    } else {
        $SIG{HUP} = $SIG{INT} = $SIG{TERM} = $SIG{CHLD} = 'DEFAULT';
        sigprocmask(SIG_UNBLOCK,$signals);

# 子进程里关闭监听 socket
        $listen_socket->close or die "[EMERG] can't close listen socket\n";

# 循环读取 socket，获取 server 发过来的扫描选项，并根据这些选项，调用相关函数进行系统扫描。
        my @warnings;
        while (<$sock>) {
            chomp;
            my ($key,$value) = split;

# 扫描端口
            if ($key eq 'PORT') {
                my $re = scan_ports($value);
                my $warn = "$exter_addr: ports dropped: " . join ',',@$re;
                push @warnings,$warn if @$re;

# 检查 CPU 负载
            } elsif ($key eq 'CPU') {
                my $warn = get_cpu_load($value);
                push @warnings,"$exter_addr: $warn" if $warn;
```

```
# 检查 free 内存的大小
                } elsif ($key eq 'FREEMEM') {
                    my $warn = get_free_mem($value);
                    push @warnings,"$exter_addr: $warn" if $warn;

# 检查并发连接数
                } elsif ($key eq 'CONCONN') {
                    my $warn = get_con_conn($value);
                    push @warnings,"$exter_addr: $warn" if $warn;

# 检查网关
                } elsif ($key eq 'CHECKGW' && $value == 1) {
                    my $warn = check_gw();
                    push @warnings,"$exter_addr: $warn" if $warn;

# 检查可用磁盘空间
                } elsif ($key eq 'FREEDISK') {
                    my $re = check_disk($value);
                    my $warn = "$exter_addr: disk warnings: " . join ';',@$re;
                    push @warnings,$warn if @$re;

# 检查已用交换内存
                } elsif ($key eq 'SWAPUSED') {
                    my $warn = check_swap($value);
                    push @warnings,"$exter_addr: $warn" if $warn;

                }
            }

# client 将扫描的结果，通过 socket 返回给 server，并关闭连接 socket。
            print $sock $_,"\n"    for (@warnings);
            $sock->close or die "[EMERG] can't close established socket\n";

            exit 0;
        }
}

#------------------
# 下面定义子函数
#------------------
```

```
# 使程序进入后台的函数，与 server 同
sub daemon
{
	my $child = fork();
	die "[EMERG] can't fork\n" unless defined $child;
	exit 0 if $child;
	setsid();

	open (STDIN, "</dev/null");
	open (STDOUT, ">/dev/null");
	open (STDERR,">&STDOUT");

	chdir $rundir;
	umask(022);
	$ENV{PATH}='/bin:/usr/bin:/sbin:/usr/sbin';

	return $$;
}

# 写日志的函数，与 server 同
sub write_log
{
	my $time=scalar localtime;
	open (HDW,">>",$log_file);
	flock (HDW,LOCK_EX);
	print HDW $time,"    ",join ' ',@_,"\n";
	flock (HDW,LOCK_UN);
	close HDW;
}

# 重载 die 的函数，与 server 同
sub log_die
{
	my $time=scalar localtime;
	open (HDW,">>",$err_log);
	print HDW $time,"    ",@_;
	close HDW;
	die @_;
}
```

```
# 重载 warn 的函数，与 server 同
sub log_warn
{
	my $time=scalar localtime;
	open (HDW,">>",$err_log);
	print HDW $time,"    ",@_;
	close HDW;
}

# killall 子进程，与 server 同
sub kill_children
{
	kill TERM => keys %status;
	sleep while %status;
}

# reload 自身，与 server 同
sub relaunch
{
	chdir $rundir;
	unlink $pid_file;
	exec 'perl','monagt';
}

# 处理 SIGHUP 的函数，与 server 同
sub do_hup
{
	warn "[INFO] received SIGHUP,prepare to reload...\n";
	kill_children();

	relaunch();
	die "[EMERG] reload failed\n";
}

# 端口扫描的函数，如果指定端口 drop 了，则返回异常报警。
# 调用了 nc 命令，请 man nc
sub scan_ports
{
	my @results;

	for my $port (split/,/,shift) {
```

```
            my $result = `$NC -nzv 127.0.0.1 $port 2>&1`;
            if ($result =~ /Connection refused/) {

                $result = `$NC -nzv $inner_addr $port 2>&1`;
                if ($result =~ /Connection refused/) {

                    $result = `$NC -nzv $exter_addr $port 2>&1`;
                    if ($result =~ /Connection refused/) {

                        push @results,$port;
                    }
                }
            }
        }

        return \@results;
}

# 检查 CPU 负载的函数，如果负载大于指定阀值，则返回异常报警
# 调用了 uptime 命令，请 man uptime
sub get_cpu_load
{
        my $arg = shift;
        my ($sign,$load) = $arg =~ /([+-])([\d\.]+)/;

        my $warn;
        unless (defined $sign && defined $load) {
            $warn = "uncorrect argument in config file";
            return $warn;
        }

        my $result = `$UPTIME`;
        my ($curr_load) = $result =~ /load average: (\d+\.\d+),/;

        if ($sign eq '+' && $curr_load > $load) {
            $warn = "CPU Load large than $load\%";

        } elsif ($sign eq '-' && $curr_load < $load) {
            $warn = "CPU Load less than $load\%";

        }
```

```
        return $warn;
}

# 检查 free 内存大小的函数，如果 free 内存小于指定阀值，则返回异常报警
# 调用了 cat 命令，请 man cat
sub get_free_mem
{
        my $arg = shift;
        my ($sign,$mem_free) = $arg =~ /([+-])([\d\.]+)/;

        my $warn;
        unless (defined $sign && defined $mem_free) {
                $warn = "uncorrect argument in config file";
                return $warn;
        }

        my @meminfo = `$CAT /proc/meminfo`;

        my ($total,$free);
        for (@meminfo) {
                if (/^MemTotal/) {
                        $total = (split)[1];
                }elsif (/^MemFree/) {
                        $free = (split)[1];
                }
        }

        my $curr_free = ($free / $total) * 100;

        if ($sign eq '+' && $curr_free > $mem_free) {
                $warn = "Free Mem large than $mem_free\%";

        } elsif ($sign eq '-' && $curr_free < $mem_free) {
                $warn = "Free Mem less than $mem_free\%";

        }

        return $warn;
}
```

```
# 检查并发连接数的函数，如果并发连接数大于指定阀值，则返回异常报警
# 调用了 netstat 命令，请 man netstat
sub get_con_conn
{
	my $arg = shift;
	my ($sign,$conn) = $arg =~ /([+-])(\d+)/;

	my $warn;
	unless (defined $sign && defined $conn) {
		$warn = "uncorrect argument in config file";
		return $warn;
	}

	my @netstat = `$NETSTAT -ts`;
	my ($estab) = grep {/connections established/} @netstat;
	my ($estab_num) = $estab =~ /(\d+)/;

	if ($sign eq '+' && $estab_num > $conn) {
		$warn = "Established Connections large than $conn";

	} elsif ($sign eq '-' && $estab_num < $conn) {
		$warn = "Established Connections less than $conn";

	}

	return $warn;
}

# 检查交换内存使用的函数，如果交换内存使用大于指定阀值，则返回异常报警
# 调用了 free 命令，请 man free
sub check_swap
{
	my $arg = shift;
	my ($sign,$swap) = $arg =~ /([+-])(\d+)/;

	my $warn;
	unless (defined $sign && defined $swap) {
		$warn = "uncorrect argument in config file";
		return $warn;
	}
```

```
        my @meminfo = `$FREE`;
        my ($swap_line) = grep {/^Swap/} @meminfo;
        my ($swap_used) = (split/\s+/,$swap_line)[2];

        if ($sign eq '+' && $swap_used > $swap) {
                $warn = "Used SWAP large than $swap KB";

        } elsif ($sign eq '-' && $swap_used < $swap) {
                $warn = "Used SWAP less than $swap KB";

        }

        return $warn;
}

# 检查网关的函数，如果网关不通，则返回异常报警
# 调用了 ping 命令，请 man ping
sub check_gw
{
        my @routes = `$NETSTAT -nr`;
        my ($gw_line) = grep {/\s+UG\s+/} @routes;
        my $gw = (split/\s+/,$gw_line)[1];

        my $warn;
        `$PING -c 1 $gw`;
        if ($? != 0) {
                $warn = "Network to GW $gw disconnected";
        }

        return $warn;
}

# 检查磁盘空间的函数，如果可用磁盘空间小于指定阀值，则返回异常报警
# 调用了 df 命令，请 man df
sub check_disk
{
        my $arg = shift;
        my ($sign,$disk_free) = $arg =~ /([+-])([\d\.]+)/;

        my @warns;
        unless (defined $sign && defined $disk_free) {
```

```
            push @warns,"uncorrect argument in config file";
            return \@warns;
        }

        my @df = `$DF -h`;
        for (@df) {

            chomp;
            my ($used,$mount) = (split)[4,5];
            $used =~ s/\%//;

            if ( $sign eq '+' && (100 - $used) > $disk_free ) {
                push @warns, "Free Disk on $mount large than $disk_free\%";

            }elsif ( $sign eq '-' && (100 - $used) < $disk_free ) {
                push @warns, "Free Disk on $mount less than $disk_free\%";

            }
        }

        return \@warns;
}

__END__
```

4 运行方法及配置文件

运行说明：

1. Client 监听在内网端口上，如果你的系统没有内网，那就不要运行它了。否则哪天 OS 被黑了，不要找偶哦。程序分析里已说明，假定内网 IP 地址是 192.168.xx.xx 段，如果不是这个段，那么请改下代码。
2. Client 和 Server 都是 daemon 方式，直接 chmod a+x program_name; ./program_name 运行即可。当然，运行前先需要建立程序里指定的$rundir。
3. 重启 Server 或 Client 只需要发送 HUP 信号即可，如在 Linux shell 里执行 kill –HUP `cat program_name.pid`。修改了 Server 的配置文件，不需要重启 Server。
4. 请运行 ps 和 netstat 命令查看程序有没有起来。如果没有起来，则查看终端输出和错误日志里的输出，多半可查明原因。
5. 默认 10 分钟扫描一次，这个可在 server 的代码里更改（$scan_inter 变量）。
6. 发送报警邮件直接用了 Net::SMTP 模块，MTA 不需要验证，只要把 server 所在机器的 sendmail 打开即可：chkconfig sendmail on;service sendmail start。sendmail 默认只监听在

localhost 的 25 端口，足够安全了。

7. OS 兼容性：目前只在 RedHat Linux 上跑得挺正常的，其他的 OS 请自己改代码和测试吧。

配置文件示例：

只有 Server 才需要一个配置文件，配置文件名在 Server 的代码里定义（$cfg_file 变量）。Client 直接运行，不需要配置文件。配置文件里主要写明要监控主机（Client）的 IP 地址，以及监控项目和参数。

如下展示 1 台要监控主机的配置示例，其他的主机照着抄即可。

```
[HOST 1]   # 每台主机配置请用[HOST X]开头，[/HOST]结尾，X 是个全局唯一整数
IP = 192.168.0.10   # 要监控主机的 IP 地址
PORT = "21,80,873,3306"   # 要监控的端口
CPU = +10%   # CPU 负载报警的阀值，+10%表示大于 10%，注意这个+号不能丢了
FREEMEM = -3%   # free 内存报警的阀值，-3%表示小于 3%，这个监控实际没多少用
CONCONN = +1000   # 并发连接数，如果大于 1000 则进行报警
CHECKGW = 1   # 是否检查网关，1 检查，0 不检查
FREEDISK = -10%   # 检查可用磁盘，如果可用磁盘小于 10%则报警
SWAPUSED = +30000   # 检查已用交换内存，如果大于 30M 则报警，单位是 KB
[/HOST]   # 本节配置结束
```

最后重申：

本程序是偶业余写的，没有任何权威性。大家感兴趣的参考下即可，若正式使用它，偶不承诺任何可靠性。

--兰花仙子

Angel_foo@earthlink.net