# Perl运行时调试工具

mnstory.net

Perl这个语言没有现成的**运行时调试**工具，网上罗列的办法有：

1. strace只能看系统调用的函数，而且是perl解释器执行的流程。
2. gdb，只能调试perl解释器。
3. gdbperl，bulkdbg，没有用过。
4. perl-stacktrace，inspect-perl-proc两款在我们的虚拟化平台都没能正确运行。

13年的时候，写过一个运行时给Perl注入Enbugger库，使其能调试的脚本，为运行时调试而生。今天想查看某些阻塞进程究竟在做啥，于是加一个新技能，打印运行时的Perl脚本堆栈，是Perl脚本堆栈，不是Perl解释器堆栈。

注意：如果Perl处于阻塞状态，是无法interrupt的，例如下面进程处于等待锁状态，无法插入代码：

host-a0369f033457 /sf/bin # strace -p 8250

Process 8250 attached - interrupt to quit

futex(0x7f7513346e60, FUTEX\_WAIT\_PRIVATE, 2, NULL^C <unfinished ...>

host-a0369f033457 /sf/bin # cat /proc/8250/stack

[<ffffffff810c97ab>] futex\_wait\_queue\_me+0xdb/0x140

[<ffffffff810ca4c6>] futex\_wait+0x186/0x280

[<ffffffff810cc38d>] do\_futex+0x12d/0x570

[<ffffffff810cc868>] SyS\_futex+0x98/0x1a0

[<ffffffff816c9b49>] system\_call\_fastpath+0x16/0x1b

[<ffffffffffffffff>] 0xffffffffffffffff

处于加锁等待，不返回无法注入代码。

## 具体效果

此工具支持两个功能，先看帮助：

host-0cc47a6bd82a /sf # perld

usage:

/sf/bin/perld -l [PORT], waiting for connection on port, default PORT=29219

/sf/bin/perld -c PID/PROCESSNAME [PORT], connect runtime process to debug port

/sf/bin/perld -e PID/PROCESSNAME [CMDS], execute commands in runtime process, print stack info by default

e.g:

/sf/bin/perld -l

/sf/bin/perld -l 29219

/sf/bin/perld -c vtpdaemon 29219

/sf/bin/perld -c 10010 29219

/sf/bin/perld -e vtpdaemon 'use plog; plog::ldebug(caller());'

/sf/bin/perld -e vtpdaemon && tail -f /sf/log/today/sfvt\_vtpdaemon.log

### 功能1：运行时attach Perl进程进行调试

调试时，需要先执行perld –l命令监听一个端口，例如：

host-0cc47a6bd82a /sf # perld -l

perl debug server listen on port 29219

然后执行perl -c 将需要attach的进程attach到监听端口。

host-0cc47a6bd82a / # perld -c vtpdaemon

connect to 127.0.0.1:29219 ok

再切换到刚才的监听界面，已经可以调试了：

host-0cc47a6bd82a /sf # perld -l

perl debug server listen on port 29219

Loading DB routines from perl5db.pl version 1.33

Editor support available.

Enter h or `h h' for help, or `man perldebug' for more help.

IO::Socket::(/usr/lib/perl/5.14/IO/Socket.pm:239):

239: return wantarray ? ($new, $peer)

240: : $new;

DB<1> x $new

0 HTTP::Daemon::ClientConn=GLOB(0x7eb1ff8)

-> \*Symbol::GEN735

FileHandle({\*Symbol::GEN735}) => fileno(12)

这个调试界面和Perl -d类似，唯一的差距是不能识别退格键，上下键。

### 功能2：打印Perl脚本堆栈

这个功能是新加入的，例如执行：

host-0cc47a6bd82a /sf/data/local/w # perld -e vtpdaemon

inject 32305 ok

inject 31247 ok

inject 30577 ok

inject 19031 ok

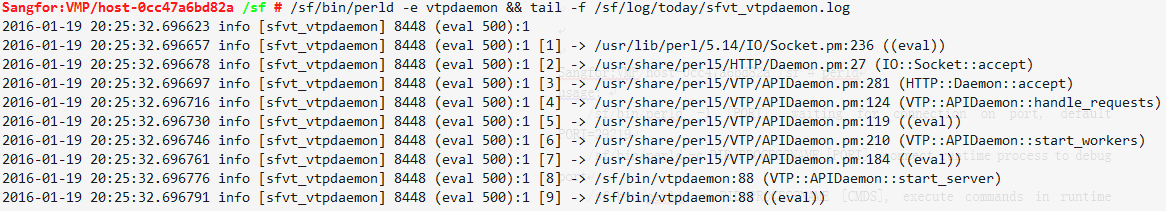
inject 12196 ok

inject 8006 ok

会将所有vtpdaemon进程的堆栈打印到日志文件。

你也可以用PID方式：

/sf/bin/perld -e 8448 && tail -f /sf/log/today/sfvt\_vtpdaemon.log



上图可以看出，vptdaemon在调用accept。

比用strace跟踪好太多，用于定位性能问题非常有效。

## 源码

### perld

#!/bin/bash

#

# runtime perl debug

#

# @author mnstory.net

# @version

#    verion 1.0 - @20131121 - add listen, connect option

#    verion 2.0 - @20160119 - add exec option

arg\_port\_dft=29219

cmd\_tmp\_file=/tmp/perld.cmd

usage()

{

local help=`cat <<- HELP

usage:

    $0 -l [PORT], waiting for connection on port, default PORT=$arg\_port\_dft

    $0 -c PID/PROCESSNAME [PORT], connect runtime process to debug port

    $0 -e PID/PROCESSNAME [CMDS], execute commands in runtime process, print stack info by default

    e.g:

        $0 -l

        $0 -l $arg\_port\_dft

        $0 -c vtpdaemon $arg\_port\_dft

        $0 -c 10010 $arg\_port\_dft

        $0 -e vtpdaemon 'use plog; plog::ldebug(caller());'

        $0 -e vtpdaemon && tail -f /sf/log/today/sfvt\_vtpdaemon.log

HELP`

    echo "$help" >&2

    return 1

}

inject()

{

    local arg\_pid="$1"

    shift

    local arg\_cmd="$@"

    if [ -z "$arg\_cmd" ]; then

        arg\_cmd="print 'inject test!'";

    fi

    echo > $cmd\_tmp\_file

    echo "call (void\*)Perl\_eval\_pv((void\*)Perl\_get\_context(),\"$arg\_cmd\",0)" >> $cmd\_tmp\_file

    echo detach >> $cmd\_tmp\_file

    echo q >> $cmd\_tmp\_file

    rtinject "$arg\_pid" $cmd\_tmp\_file

    lasterr=$?

    if [ $lasterr -ne 0 ]; then

        echo "rtinject $arg\_pid $cmd\_tmp\_file failed($lasterr)" >&2

        return $lasterr

    fi

}

onListen()

{

    if [ -z "$1" ]; then

        arg\_port=$arg\_port\_dft

    else

        arg\_port=$1

    fi

    echo perl debug server listen on port $arg\_port

    nc -l -p $arg\_port

}

onConnect()

{

    arg\_pid=$1

    if [ -z "$arg\_pid" ]; then

        usage

        return 1

    fi

    if [ -z "$2" ]; then

        arg\_port=$arg\_port\_dft

    else

        arg\_port=$2

    fi

    netstat -l --numeric-ports | grep -P ":$arg\_port\s.\*?LISTEN" >/dev/null 2>&1

    if [ $? -ne 0 ]; then

        echo "you must run: '$0 -l $arg\_port' first"

        return 1;

    fi

    inject $arg\_pid "eval{require Enbugger;warn q(stopping);\$ENV{PERLDB\_OPTS}='RemotePort=127.0.0.1:$arg\_port';Enbugger->stop;};print STDERR \$@;"

    if [ $? -eq 0 ]; then

        echo "connect to 127.0.0.1:$arg\_port ok"

    fi

}

onExec()

{

    arg\_pid=$1

    if [ -z "$arg\_pid" ]; then

        usage

        return 1

    fi

    if [ -z "$2" ]; then

        arg\_cmd='eval{ use plog qw(linfo); linfo(\"\"); foreach $i (1..30) { ($p,$f,$l,$s)=caller($i); if(!$f) {last}; linfo(\"[$i] -> $f:$l ($s)\"); } };print STDERR $@;'

    else

        arg\_cmd="$2"

    fi

    expr "$arg\_pid" "+" 10 >/dev/null 2>&1

    if [ $? -ne 0 ];then

        arg\_pid=$(pidof "$arg\_pid")

        if [ -z "$arg\_pid" ]; then

            echo "can't find runtime process $1" >&2

            return 2

        fi

    fi

    for pid in $arg\_pid; do

        inject $pid "$arg\_cmd"

        echo "inject $pid ok"

    done

}

main()

{

    arg\_cmd=$1

    shift

    if [ -z "$arg\_cmd" ]; then

        usage

        return 1

    fi

    if [ "$arg\_cmd" = "-l" ]; then

        onListen "$@"

    elif [ "$arg\_cmd" = "-c" ]; then

        onConnect "$@"

    elif [ "$arg\_cmd" = "-e" ]; then

        onExec "$@"

    else

        usage

        return 1

    fi

}

main "$@"

### rtinject

perld里面调用了rtinject，这个脚本是很早以前写的，用于运行时注入代码。

#!/bin/bash

#

# 1. runtime commands inject

# @author mnstory.net@20131024

lerror()

{

    echo "$@" >&2

}

ldebug()

{

    echo "$@" >&2

}

usage()

{

    local help=`cat <<- HELP

usage:

    $0 PID/PROCESSNAME COMMANDS/FILE [TRIPLEVEL]

    TRIPLEVEL - trip stdout/stderr, choose value from 'all' 'trip' 'none', default is 'trip'

e.g:

    $0 a.out /tmp/commands.txt

    $0 29219 'thread apply all bt' none

    $0 bash 'bt'

HELP`

    echo "$help" >&2

}

inject\_pid()

{

    local arg\_pid=$1

    local arg\_cmd=$2

    if [ ! -d "/proc/$arg\_pid" ]; then

        lerror "proc $arg\_pid not exist, no inject!"

        return 1

    fi

    local cmd\_args="-n -q /proc/$arg\_pid/exe $arg\_pid"

    if test -f "$arg\_cmd" 2>/dev/null; then

#        ldebug gdb $cmd\_args -x $arg\_cmd

        gdb $cmd\_args -x $arg\_cmd 2>&1

    else

#        ldebug gdb $cmd\_args "$arg\_cmd"

gdb $cmd\_args 2>&1 <<EOF

$arg\_cmd

EOF

    fi

}

inject\_cmd()

{

    local arg\_pid="$1"

    local arg\_cmd="$2"

    expr "$arg\_pid" "+" 10 >/dev/null 2>&1

    if [ $? -ne 0 ];then

        # just peek first pid

        arg\_pid=$(pidof "$arg\_pid" | awk '{print $1}')

        if [ -z "$arg\_pid" ]; then

            lerror "can't find runtime process $1"

            return 2

        fi

    fi

    inject\_pid "$arg\_pid" "$arg\_cmd"

}

main()

{

    local arg\_pid="$1"

    local arg\_cmd="$2"

    local arg\_trip="$3"

    if [ "$arg\_pid" = "" -o "$arg\_cmd" = "" ]; then

        usage

        return 1

    fi

    if [ "$arg\_trip" = "all" ]; then

        inject\_cmd "$arg\_pid" "$arg\_cmd"

    elif [ "$arg\_trip" = "none" ]; then

        inject\_cmd "$arg\_pid" "$arg\_cmd" >/dev/null

    else

        inject\_cmd "$arg\_pid" "$arg\_cmd" | /bin/sed -n -e 's/^(gdb) //' -e '/^#/p' -e '/^Thread/p'

    fi

}

main "$1" "$2" "$3"

rtinject也可以单独运行，我之前写的clog日志库，就提供了运行时调整level的功能，主要就是用这个原理：

host-0cc47a6bd82a /sf/bin # rtinject

usage:

    /sf/bin/rtinject PID/PROCESSNAME COMMANDS/FILE [TRIPLEVEL]

    TRIPLEVEL - trip stdout/stderr, choose value from 'all' 'trip' 'none', default is 'trip'

e.g:

    /sf/bin/rtinject a.out /tmp/commands.txt

    /sf/bin/rtinject 29219 'thread apply all bt' none

    /sf/bin/rtinject bash 'bt'

host-0cc47a6bd82a /sf/bin # rtinject kvm 'thread apply all bt'

Thread 32 (Thread 0x7f1e5ecbb700 (LWP 31848)):

#0  0x00007f1e65e7d64b in pthread\_cond\_timedwait@@GLIBC\_2.3.2 ()

#1  0x00007f1e6b2ff323 in syncenv\_task (proc=proc@entry=0x7f1e6fb0bb30)

#2  0x00007f1e6b2ff7a0 in syncenv\_processor (thdata=0x7f1e6fb0bb30)

#3  0x00007f1e65e78b50 in start\_thread ()

#4  0x00007f1e65bc2a7d in clone () from /lib/x86\_64-linux-gnu/libc.so.6

#5  0x0000000000000000 in ?? ()

Thread 31 (Thread 0x7f1e5d8ba700 (LWP 31849)):

#0  0x00007f1e65e7d64b in pthread\_cond\_timedwait@@GLIBC\_2.3.2 ()

#1  0x00007f1e6b2ff323 in syncenv\_task (proc=proc@entry=0x7f1e6fb0bef0)

#2  0x00007f1e6b2ff7a0 in syncenv\_processor (thdata=0x7f1e6fb0bef0)

#3  0x00007f1e65e78b50 in start\_thread ()

#4  0x00007f1e65bc2a7d in clone () from /lib/x86\_64-linux-gnu/libc.so.6

#5  0x0000000000000000 in ?? ()

2016/1/21