Alternative call waiting service with Perl

THE LISTENER

If you can't get through to your home number, a script can check the line status and send a signal to you via the Web to tell you when the line is free. **BY MICHAEL SCHILLI**

You're trying to call home, but somebody is hogging the line. Of course, you could always ask your telco to enable call waiting [2]. The service lets marathon phone users know that a call is waiting by beeping every couple of seconds. On the other hand, the service costs money, and the beep can get on your nerves. This is what prompted me to buy a small phone amplifier (Figure 1 and [3]), and put together a handy application.

The Smart Phone Recorder Control grabs the signal off the phone line and feeds it to the sound card in my Linux computer. Linux can access the signal via the */dev/dsp* device, and the Perl *Audio::DSP* module from CPAN will read it. The script uses a couple of heuristic tricks to determine whether the line is busy, and if so, it keeps on trying until the line is free, before notifying the user via a hidden CGI script on a website.

Remote Sleeper

In its idle state, the *phonewatch* script (Listing 1) on the Linux computer in your home checks the website every 60 seconds to see whether someone has clicked to run the CGI script and enabled the "Check" status (Figure 3). If so, the script wakes up and starts to collect data off the phone line. It displays a "busy" message on the website while the line is busy, refreshing the browser page every 60 seconds.

If you hang up, *phonewatch* detects the status change and switches the CGI script status back to "idle".

The infinite loop that starts in Line 24 detects the status on the website, and sleeps for *$IDLE_POLL* seconds, while *state* is set to *idle*. The *state()* function simply serves to query the current status. However, it can set a new status if you pass a parameter in to it. Both of these tasks are handled by a *get* call from the LWP::Simple module, which references a website by its URL. It filters the script status out of the *<b>* tag in the page content returned in the response.

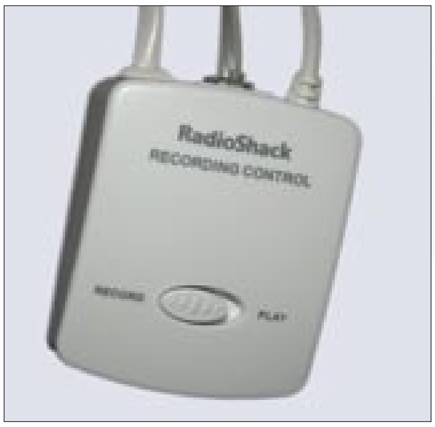

Figure 1: Smart Phone Recorder Control by Radio Shack feeds the signal from the phone line to the sound card in a Linux computer.

Audio Fishing

The *Audio::DSP* class constructor expects four parameters: the length of the data buffer to fill (1024), the number of channels (1 here, for mono), the sample format (unsigned 8bit), and the sampling rate (1024 samples per second).

Digital audio data is written in a numeric format that an ADC samples *n* times per second from the analog sound signal. The sampling rate *n* must be twice the size of the highest audio frequency you want to sample; that is about 40,000 samples per second for HiFi quality (slightly above 20 kHz). As we do not actually need to eavesdrop the conversation, 1024 samples per second are perfectly ok. For more information on “Digital Audio” refer to [2].

Within a second, *phonewatch* fills up the data buffer, and feeds it to the *sample_add()* method of the module shown in the *SoundActivity.pm* script (Listing 2). The method unpacks the 8-bit values (*unpack("C")*) from the data block passed to it, and calculates the standard deviation for the values. The sound card will not be able to pick up a signal from a line that is down; there is only some noise at the sound card’s microphone input.

In idle state, the sampling values, which have a range of between 0 and 255, will all be approximately 127, and will deviate by a maximum of 1 to either side. The standard deviation was typically around 0.5 in my lab.

The *sdev* method that starts in Line 84 of *SoundActivity.pm* calculates the standard deviation for the elements in an array passed in as a pointer to an accuracy of two decimal places. *sdev()* uses the CPAN *Statistics::Basic::StdDev* module for the simple mathematics required here.

Wake Up and Sample!

In active mode, *phonewatch* hangs around in the poll_busy() function. It samples the sound card for a second, waits for 10 seconds, and then ascertains the next sample point. The five recorded values for the standard deviation would look something like this for a phone call: *[0.64, 0.78, 0.73, 0.89, 0.86]*

The maximum number of entries in the standard deviation history is defined by the parameter *max_hist* in *SoundActivity.pm*. *min_hist* defines the mini-

Listing 1: phonewatch

```
#!/usr/bin/perl -w

use strict;
use Audio::DSP;
use Log::Log4perl qw(:easy);
use SoundActivity;
use LWP::Simple;

Log::Log4perl->easy_init(
 {
   file =>
     “/tmp/phonewatch.log”,
   level => $INFO,
 }
);

my $IN_USE_POLL = 10;
my $IDLE_POLL   = 60;
my $STATUS_URL  =
‘https://u:p@_foo.com/
phonewatch.cgi’;
my $SAMPLE_RATE = 1024;

INFO “Starting up”;
while (1) {
  my $state = state();

  if (!defined $state) {
    DEBUG “Fetch failed”;
    sleep $IDLE_POLL;
    next;
  }

  DEBUG
    “web site state: $state”;

  if ($state eq “idle”) {
    DEBUG “Staying idle”;
    sleep $IDLE_POLL;
    next;
  }

  INFO “Monitor requested”;
  state(“busy”);
  poll_busy();
  state(“idle”);
}
################################
sub poll_busy {
################################
  my $dsp = new Audio::DSP(
    buffer   => 1024,
    channels => 1,
    format   => 8,
    rate     => $SAMPLE_RATE,
  );

  $dsp->init()
    or die $dsp->errstr();

  my $act =
    SoundActivity->new();

  while (1) {
    DEBUG “Reading DSP”;
    $dsp->read()
      or die $dsp->errstr();

    $act->sample_add(
      $dsp->data());
    $dsp->clear();

    if (!$act->is_active()) {
      INFO “Hangup detected”;
      $dsp->close();
      return 1;
    }
    sleep $IN_USE_POLL;
  }
}
################################
sub state {
################################
  my ($value) = @_;

  my $url = $STATUS_URL;
  $url .= “?state=$value”
    if $value;
  DEBUG “Fetching $url”;
  my $content = get $url;
  if ($content =~
     m#<b>(.*?)</b>#)
  {
    return $1;
  }
}
```

Listing 2: SoundActivity.pm

```
################################
# Mike Schilli, 2006
# m@perlmeister.com
################################
package SoundActivity;
################################

use strict;
use warnings;
use
  Statistics::Basic::StdDev;
use Log::Log4perl qw(:easy);

################################
sub new {
################################

 my ($class, %options) = @_;

 my $self = {
  min_hist      => 5,
  max_hist      => 5,
  history       => [],
  sdev_threshold => 0.01,
  %options,
 };

 bless $self, $class;
}

################################
sub sample_add {
################################

 my ($self, $data) = @_;

 my $len     = length($data);
 my @samples =
   unpack(“C$len”, $data);

 my $sdev =
   $self->sdev(\@samples);

 my $h = $self->{history};
 push @$h, $sdev;
 shift @$h
   if @$h >
   $self->{max_hist};
 DEBUG “History: [“,
   join(‘, ‘, @$h), “]”;
}

################################
sub is_active {
################################

 my ($self) = @_;

 if (@{ $self->{history} } <
  $self->{min_hist})
 {
  DEBUG
    “Not enough samples yet”;
  return 1;
 }

 my $sdev =
   $self->sdev(
  $self->{history});
 DEBUG “sdev=$sdev”;

 if ($sdev <
  $self->{sdev_threshold})
 {
  DEBUG
    “sdev too low ($sdev)”;
  return 0;
 }

 return 1;
}

################################
sub sdev {
################################

 my ($self, $aref) = @_;

 return sprintf “%.2f”,
   Statistics::Basic::StdDev
   ->new($aref)->query;
}

1;
```

mum number of historic values required by the module to safely guess the line status. When you hang up, the line goes quiet again, and the standard deviation history levels out at a constant value: *[0.51, 0.51, 0.51, 0.51, 0.51]*

To distinguish between these two states, *SoundActivity.pm* simply reevaluates the standard deviation for these five historic values. If it is less than 0.01, the line is assumed to be down, and *is_active()* will return a value of false.

The CGI script on the web server stores its state across multiple calls in *phonewatch.dat*, where it stores a persistent hash called *%store*. New status values are written by the CGI *state* parameter.

All the script has to do then is to retrieve the appropriate state color from the *%states* hash, and call the *process* method from the template toolkit with the HTML template defined in the DATA attachment. This generates an HTML rendering of the page by interpreting the simple but effective template language. At the same time, a reload meta-tag is injected into the page to tell the browser to

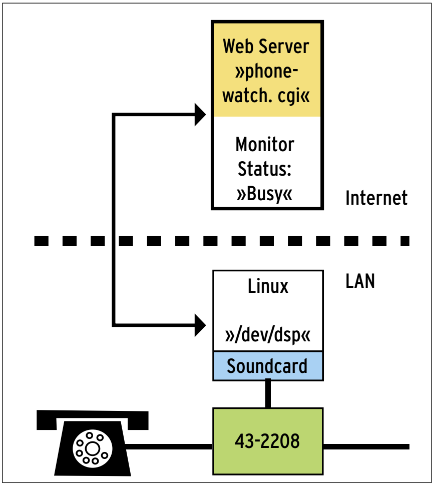

Figure 2: A website enables the script on the Linux machine and outputs the call status.

INFO

[1] Call Waiting: *http://en.wikipedia.org/wiki/Call_waiting*

[2] Ken C. Pohlmann: Principles of Digital Audio, McGraw Hill, 2005

[3] Smart Phone Recorder Control, Radio Shack catalog, *http://www.radioshack.com/search/index.jsp?kw=43-2208*

[4] Listings for this article: *http://www.linux-magazine.com/Magazine/Downloads/74/Perl*

ADVERTISEMENT

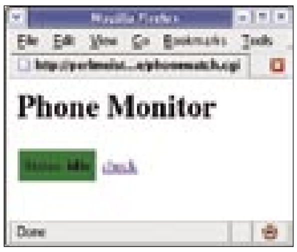

Figure 3: The CGI script is in idle mode; the listener is waiting for action.

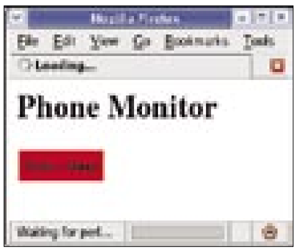

Figure 4: The listener verifies that the line is busy and displays a message.

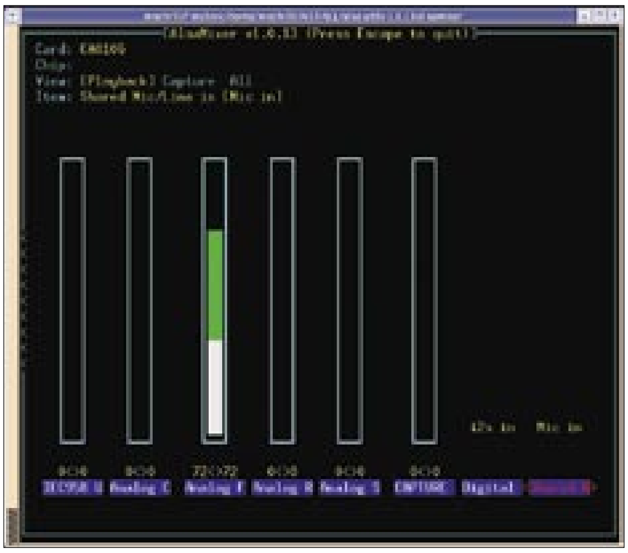
AlsaMixer v1.0.13 (Press Escape to quit)
Card:
Chip:
View: [Playback] Capture All
Item: Shared Mic/Line in [Mic in]

Figure 5: To make sure the microphone input on your sound card works properly, the control on the far right in alsamixer should be set to "Mic" and not to "Line".

refresh the page every 30 seconds with a call to the CGI script. In turn, this ensures that the *state()* change, detected by the Linux machine when the caller hangs up, is passed on to the browser to end the user's wait.

When called without any parameters, the CGI script simply returns the current state as formatted HTML: However, if a new state is passed to the script in the CGI *status* parameter (*idle*, *busy*, or *check*), the script modifies its internal status, and stores it permanently.

It is not a good idea to expose the script on the untrusted Internet; a simple form of protection would be to use a *.htaccess* file that prompts the user for a username and password. When *phonewatch* calls the CGI script, it simply bundles the credentials with the URL, *https://user:pass@URL...*.

Same Old Song

Getting a sound card to run on Linux is not always simple, however, *Audacity* worked perfectly with ALSA 1.0.13, which I used for my experiments. It is important to switch the microphone input from "Line" to "Mic" in the *alsamixer* (Figure 5) to enhance the gain.

The CPAN modules used by the scripts are easily installed using a CPAN shell, as always; you will want to install the additional *SoundActivity* module in the same directory as *phonewatch*, or in a directory in which *phonewatch* will be able to find it.

The logging level for *phonewatch* is set to *$INFO* by default, and this default level only appends a minimal amount of critical status information to the */tmp/phonewatch.log* logfile. You might prefer to set the level to *$DEBUG* for more detail.

Set up an entry in your *inittab* to automatically launch the script. This sends the program into an infinite loop and gives you a friendly helper that you can enable from wherever you are with a web browser. ■

Listing 3: phonewatch.cgi

```
#!/usr/bin/perl -w
use strict;
use CGI qw(:all);
use DB_File;
use Template;

my %states = (
 idle  => 'green',
 check => 'yellow',
 busy  => 'red',
);
tie my %store, "DB_File",
  "data/phonewatch.dat"
  or die $!;
$store{state} = "idle"
  unless
  defined $store{state};
print header();
my $new = param('state');
if ($new
 and exists $states{$new})
{
 $store{state} = $new;
}
my $tpl = Template->new();
$tpl->process(
 \join('', <DATA>),
 {
  bgcolor =>
    $states{ $store{state} },
  state => $store{state},
  self  => url(),
 }
 )
  or die $tpl->error;
################################
__DATA__
<HEAD>
  <META HTTP-EQUIV="Refresh"
        CONTENT="30;
               URL=[% self %]">
</HEAD>
<BODY>
  <H1>Phone Monitor</H1>
  <TABLE CELLPADDING=5>
    <TR>
      <TD BGCOLOR="[% bgcolor
%]">
        Status: <b>[% state
%]</b>
      </TD>
      [% IF state == "idle" %]
      <TD>
        <A HREF="[% self
%]?state=check">
        check</A>
      </TD>
      [% END %]
    </TR>
  </TABLE>
</BODY>
```