Lidando com interfaces Gtk e consultas à Web em Perl

Trabalho de Equipe

O Perl Object Environment (POE) ou Ambiente Orientado a Objetos Perl, propicia uma plataforma para que os scripts possam realizar múltiplas tarefas de forma cooperativa sem precisar de ajuda do sistema operacional. O aplicativo deste mês permite que uma interface GTK realize consultas demoradas via web, sem soluços. POR MICHAEL SCHILLI

Os aplicativos com uma interface gráfica são geralmente baseados em eventos. O programa tem um loop principal, utilizado para esperar por eventos como os cliques de um mouse e entrada de dados através do teclado. É importante que o programa processe estes eventos sem nenhum atraso e retorne rapidamente ao loop principal. Isto impede que o usuário note a indisponibilidade temporária da interface.

No artigo deste mês estaremos analisando um programa, um *stock ticker*, que consulta e exibe o valor de uma determinada ação na Bolsa de Valores. Periodicamente, este programa conecta-se à página de finanças do Yahoo!, atualizando os valores previamente selecionados (veja Figura 1). Dependendo da conexão de rede esta operação pode, incluindo a resolução de nome pelo servidor DNS, levar alguns segundos até se completar. Seria interessante que a interface do aplicativo se mantivesse funcionando durante este período.

Os desenvolvedores já podem usar técnicas como o multiprocessamento ou multithreading para alcançar este objetivo. Entretanto, ambas técnicas fazem com que o programa fique muito mais complexo. Seções críticas precisam ser protegidas contra o acesso paralelo para assegurar a integridade dos dados, evitando desta forma erros de difícil correção. Se você já teve que analisar um arquivo *core* com 200 threads ativas, sabe muito bem do que estou falando.

Há uma alternativa que evita estes inconvenientes – a multitarefa cooperativa com o POE [2], desenvolvido principalmente por Rocco Caputo. O ambiente é implementado como uma máquina de estados que executa exatamente um processo em uma única thread, porém há um "kernel" no espaço de usuário, que permite a realização de múltiplas tarefas quase que simultaneamente.

Acompanhando

Programadores Perl que querem consultar o preço de ações costumam usar o módulo *Yahoo::FinanceQuote*, encontrado na CPAN:

```
use Finance::YahooQuote;
my @quote = ➥
getonequote($symbol);
```

Infelizmente, o módulo trabalha de forma síncrona, o que torna difícil obter o efeito de "scroll" suave que queremos na interface. A função *getonequote (*símbolo*)* envia um pedido HTTP ao servidor Yahoo, espera por uma resposta e então retorna os resultados.

Nós queremos manter o display funcionando enquanto o programa espera por uma resposta – e, de acordo com a lei de Murphy, podemos ter certeza absoluta de que outra janela será arrastada sobre ele neste exato momento. Neste caso, o programa terá que redesenhar a área que foi oculta. Tal processo é conhecido como refreshing.

Infelizmente, devido à demora na resposta à solicitação HTTP, o aplicativo não recebe a ordem para redesenhar sua janela, e conseqüentemente deixa um horroroso "buraco" cinza no desktop – certamente uma visão nada agradável.

A alternativa assíncrona

Seria mais elegante transmitir o pedido à web e retornar para redesenhar a janela, sem ter que esperar pelos resultados. Quando a resposta do servidor Yahoo! chegar, ela deveria causar algum tipo de alerta. Isto significa uma rápida atualização da janela com os valores, e o retorno ao loop principal do programa.

Isto é exatamente o que o POE faz. Fornece um kernel onde cada aplicativo sessões, e máquinas de estado se movem entre estados, trocando mensagens. A atividade de entrada e saída de dados é assíncrona. Ao invés de abrir um arquivo ou uma conexão e esperar pelos dados, você pode simplesmente dizer ao kernel: "Ei! Eu quero ler algumas informações. Você pode me avisar quando elas estiverem disponíveis?"

Figura 1: Nosso aplicativo GTK para consulta de valores de ações contata periodicamente os servidores do Yahoo!.

Dados em Alta Velocidade

Embora as operações de leitura e escrita não sejam totalmente assíncronas (de uma forma resumida, o POE simplesmente usa as funções *syswrite ()* ou *sysread ()*, executando as chamadas do sistema no modo conhecido como *non-blocking*), qualquer informação é processada em velocidade máxima.

O aspecto cooperativo do POE se baseia no fato de que as sessões competem entre si, mas de forma não egoísta. Se uma tarefa não ocupa totalmente a CPU, a sessão tem que devolver o controle ao kernel. Uma única parte não cooperativa em um programa pode ter impacto no sistema inteiro.

A multitarefa em uma única thread facilita o desenvolvimento do programa – você não precisa se preocupar com travas, não há surpresas com *race conditions*, e mesmo que um erro ocorra, é fácil localizá-lo. POE coopera com o loop principal de vários ambientes gráficos e reconhece automaticamente Perl/TK e gtkperl integrando-os perfeitamente

Isto permite ao kernel designar fatias de tempo aos eventos da interface, da mesma forma que uma sessão explicitamente definida. Esta é a resposta para nosso problema com o refresh.

Alertando o Kernel

O programa na Listagem 2 usa a máquina de estado *POE:: Session* como mostrado na Figura 2. A fase de inicialização, *_start*, gera a interface GTK e registra o apelido *ticker* para nos permitir identificar a seção facilmente mais tarde. A seguir o controle é devolvido ao kernel. A máquina de estados entra no estado *wake_up* a cada 60 segundos (por meio de um alerta) ou quando alguém clica no botão *Update* na interface. Isto lança outra máquina de estados tipo *POE::Component:: Client::HTTP*, e imediatamente devolve o controle do programa ao kernel.

O "PoCoCli::HTTP" é um componente do framework POE, uma máquina de estados que define sua própria seção (chamada *useragent* na Listagem 2, linha 73), aceita pedidos à web no estado *request* e então retorna ao framework POE até receber uma resposta completa via HTTP. Neste ponto, *useragent* pede ao kernel que avise a sessão que a chamou, *ticker*, para entrar em um estado chamado *yhoo_response*, que foi previamente passado à *useragent*.

O kernel diz à seção *ticker* para fazer exatamente isso, e a seção aceita a resposta HTTP, que já estava à espera, e atualiza a janela antes de devolver o controle ao kernel, sem nenhum atraso. Na linha 69, o componente *POE::Component::Client::HTTP* é executado pela função *spawn()*, a qual especifica que o texto *gtkticker/0.01* deve aparecer na string *user agent* e que pedidos devem expirar após 60 segundos.

Manual do Yahoo

As linhas 10 a 12 na Listagem 2 definem o endereço do serviço de cotação de valores de ações do Yahoo! (Yahoo! Finance). A interface deste serviço recebe dois parâmetros:

- Um parâmetro de formato (= f) com os nomes dos campos solicitados: s (symbol), l1 (valor da ação) e c1 (mudança no valor, em percentagem, desde o último dia de operação da bolsa)
- Um parâmetro símbolo (*symbol*) que contém uma lista, separada por vírgulas, dos símbolos das empresas em cujas ações estamos interessados, por exemplo YHOO, MSFT, TWX.

O servidor responde com:

```
"YHOO",45.38,+0.35
"MSFT",27.56,+0.19
"TWX",18.21,+0.75
```

Nosso aplicativo GTKTicker aceita a resposta linha a linha, separa os valores nas vírgulas e envia a informação para a interface gráfica.

Configuração no diretório $HOME

As linhas 13 e 14 da Listagem 2 especificam que o arquivo *.gtkicker* no diretório pessoal do usuário é o arquivo de símbolos a ser usado pelo ticker. As linhas 30 a 37 interpretam este arquivo linha a linha (veja o loop *while* na linha 32), descartando toda linha iniciada por *#*, considerada como comentário (linha 33). O loop *for* implícito no final da linha 35:

```
... for /(\S+)/g;
```

executa a expressão à direita em todas as palavras da linha, e coloca o símbolo da empresa na variável *$_*. Isto permite que uma única linha tenha múltiplos símbolos, separados por espaços. A função *push* (linha 34 da Listagem 2) coloca os símbolos no array @SYMBOLS.

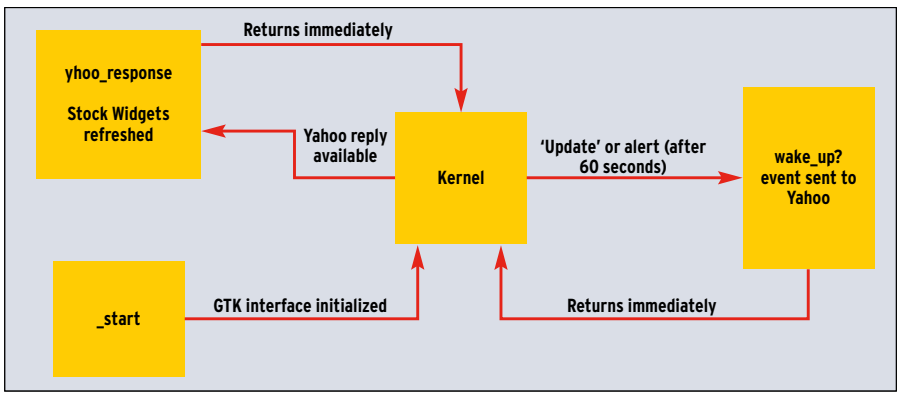

Figura 2: A máquina de estados do GTKTicker. Após o estado de inicialização, *_start*, o controle é devolvido ao kernel. A cada 60 segundos a máquina entra no estado *wake_up*, e chama uma outra máquina de estados responsável por realizar a solicitação HTTP e devolver o controle ao kernel

Apesar de estar usando o framework POE, o GTKTicker emprega funções síncronas comuns de entrada/saída para ler o arquivo de configuração, já que o arquivo é curto e o kernel POE não está em execução neste momento.

Que começe a dança!

A linha 39 da Listagem 2 define a máquina de estados do ticker. A função *inline_states* mapeia funções aos estados. O kernel irá executar estas funções quando a máquina entrar no estado correspondente. A linha 53 declara o estado *wake_up* da sessão *ticker* ao kernel:

```
$poe_kernel->post("ticker", ➥
"wake_up");
```

através da variável $poe_kernel exportada pelo POE. A linha 55 inicia o loop principal do kernel,

```
$poe_kernel->run();
```

o qual permanece em execução até que seja desativado. É isto!

A construção dos objetos POE::Session mostrada anteriormente tem um efeito colateral. Ela executou a rotina *start()* definida na linha 58, que é mapeada ao estado *_start*. Este, por sua vez, registra o apelido da sessão como *ticker* e então pula para *my_gtk_init()*. Esta função, que começa na linha 98, constrói a interface gráfica em GTK.

Interfaces gráficas com GTK

O módulo Gtk encontrado na CPAN foi escrito por Marc Lehmann. Na verdade este módulo foi substituído pelo Gtk2, mas esta nova versão ainda tem alguns pequenos problemas com o POE. Mas não se preocupe, pois o "obsoleto" módulo baseado no Gtk1 ainda funciona, e muito bem.

Listagem 2: Gtkticker

```
001 #!/usr/bin/perl
002 ##################################
003 # gtkticker
004 # Mike Schilli, 2004
005 # (m@perlmeister.com)
006 ##################################
007 use warnings;
008 use strict;
009
010 my $YHOO_URL =
011  "http://quote.yahoo.com/d?".
012  "f=sl1c1&s=";
013 my $RCFILE   =
014  "$ENV{HOME}/.gtkticker";
015 my @LABELS       = ();
016 my $UPD_INTERVAL = 60;
017 my @SYMBOLS;
018
019 use Gtk;
020 use POE qw(
021     Component::Client::HTTP);
022 use HTTP::Request;
023 use Log::Log4perl qw(:easy);
024 use Data::Dumper;
025
026 Log::Log4perl->easy_init(
027                       $DEBUG);
028
029     # Read config file
030 open FILE, "<$RCFILE" or
031    die "Cannot open $RCFILE";
032 while(<FILE>) {
033     next if /^\s*#/;
034     push @SYMBOLS, $_
035                 for /(\S+)/g;
036 }
037 close FILE;
038
039 POE::Session->create(
040  inline_states => {
041    _start   => \&start,
042    _stop    => sub {
043              INFO "Shutdown" },
044    yhoo_response =>
045                \&resp_handler,
046    wake_up  =>
047              \&wake_up_handler,
048    }
049 );
050
051 my $STATUS;
052
053 $poe_kernel->post(
054         "ticker", "wake_up");
055 $poe_kernel->run();
056
057 ##################################
058 sub start {
059 ##################################
060
061   DEBUG "Starting up";
062
063   $poe_kernel->alias_set(
064                       'ticker');
065   my_gtk_init();
066
067   $STATUS->set("Startup");
068
069  POE::Component::Client::HTTP
070   ->spawn(
071     Agent   =>
072           'gtkticker/0.01',
073     Alias   => 'useragent',
074     Timeout => 60,
075   );
076 }
077
078 ##################################
079 sub upd_quotes {
080 ##################################
081
082   my $request =
083     HTTP::Request->new(
084       GET => $YHOO_URL .
085          join ",", @SYMBOLS);
086
087   $STATUS->set(
088           "Fetching quotes");
089
090   $poe_kernel->post(
091     'useragent',
092     'request',
093     'yhoo_response',
094     $request);
095 }
096
097 ##################################
098 sub my_gtk_init {
099 ##################################
100
101   my $w = Gtk::Window->new();
102   $w->set_default_size(
103                       150,200);
104
105       # Create Menu
106   my $accel =
107       Gtk::AccelGroup->new();
108   $accel->attach($w);
109   my $factory =
110     Gtk::ItemFactory->new(
111       'Gtk::MenuBar',
112       "<main>", $accel);
113
114   $factory->create_items(
115     { path =>  '/_File',
116       type =>  '<Branch>',
117     },
118     { path        =>
119               '/_File/_Quit',
120       accelerator =>
121                 '<control>Q',
122       callback    =>
123       [sub { Gtk->exit(0) }],
```

Um objeto da classe *Gtk::Window* representa a janela principal do aplicativo que está sendo construído. No topo desta janela é possível encontrar um menu principal, que nos dá acesso a outros menus. Em especial, há neste menu uma opção denominada Quit. Esta opção, que tem a finalidade de encerrar o programa, utiliza uma rotina conhecida como *Gtk->exit(0)*. Há também o objeto *Gtk::AccelGroup*, que permite que o usuário pressione as teclas *[Ctrl]+[Q]* para finalizar o programa. Além disso, o objeto adiciona atalhos de teclado para os itens nos menus.

Construindo Menus

O menu é criado pela classe *Gtk::ItemFactory*, que a princípio é usada para criar a barra de menus *Gtk::MenuBar*. Cada entrada no menu, e seus sumenus correspondentes, é criada pelo método *create_items()*. O parâmetro path especifica a posição de um item no menu, por exemplo, */_File/_Quit* define que o item *Quit* estará abaixo de *File* na barra de menu. Os sublinhados presentes antes de *File* e *Quit* têm a função de ressaltar a primeira letra destes dois nomes, permitindo que o usuário possa acessar itens do menu através de atalhos, como *[Alt]+[F]* ou *[Alt]+[Q]*. O parâmetro *callback* especifica qual função o Gtk irá chamar sempre que o usuário selecionar um item ou pressionar as teclas de atalho correspondentes.

Gerenciador de layout

Há dois métodos diferentes para ordenar os widgets: *Gtk::VBox* e *Gtk::Table*. O *Gtk::VBox* é uma caixa responsável por alinhar verticalmente os itens nele contidos. A função *pack_start()* posiciona os elementos de cima para baixo enquanto a função *pack_end()* os coloca de baixo para cima. Veja o exemplo a seguir:

Listagem 2: Gtkticker

```
    });

my $vb = Gtk::VBox->new(
                        0,0);
my $upd = Gtk::Button->new(
                  'Update');

$vb->pack_start(
  $factory->get_widget(
       '<main>'), 0, 0, 0);

    # Button at bottom
$vb->pack_end($upd,
              0, 0, 0);

  # Status line on top
  # of buttons
$STATUS= Gtk::Label->new();
$STATUS->set_alignment(
                  0.5, 0.5);
$vb->pack_end($STATUS,
                   0, 0, 0);
my $table =
  Gtk::Table->new(
    scalar @SYMBOLS, 3);

$vb->pack_start($table,
                1, 1, 0);

for my $row (0..
              @SYMBOLS-1) {
  for my $col (0..2) {
    my $label =
         Gtk::Label->new();
    $label->set_alignment(
                  0.0, 0.5);
    push @{$LABELS[$row]},
         $label;

   $table->attach_defaults(
     $label, $col, $col+1,
       $row, $row+1);
   }
  }

  $w->add($vb);

     # Destroying window
  $w->signal_connect(
    'destroy', sub {
              Gtk->exit(0)});

   # Pressing update button
  $upd->signal_connect(
    'clicked', sub {
      DEBUG "Sending wakeup";
      $poe_kernel->post(
        'ticker', 'wake_up')}
  );
  $w->show_all();
}

##################################
sub resp_handler {
##################################
  my ($req, $resp) =
    map { $_->[0] }
              @_[ARG0, ARG1];

  if($resp->is_error()) {
      ERROR $resp->message();
      $STATUS->set(
           $resp->message());
      return 1;
  }

  DEBUG "Response: ",
        $resp->content();

  my $count = 0;

  for(split /\n/,
          $resp->content()) {

    my($symbol, $price,
       $change) =
               split /,/, $_;

      chop $change;
      $change = "" if
        $change =~ /^0/;

      $symbol =~ s/"//g;
      $LABELS[$count][0]->
                set($symbol);
      $LABELS[$count][1]->
                 set($price);
      $LABELS[$count][2]->
                set($change);
      $count++;
  }

  $STATUS->set("");

  1;
}

##################################
sub wake_up_handler {
##################################
  DEBUG("waking up");

      # Initiate update
  upd_quotes();

      # Re-enable timer
  $poe_kernel->delay(
  'wake_up', $UPD_INTERVAL);
}
```

```
$vb->pack_start($menu_bar, ➥
$expand, $fill, $padding);
```

Este exemplo posiciona uma barra de menus no topo da VBox. Na linha 132, o GTKTicker usa a função *$factory->get_widget('<main>')* para acessar o objeto da barra pelo seu nome. O parâmetro *$expand*, usado na função *pack_start()*, especifica se a área ocupada pelo widget deverá crescer caso o mouse seja usado para redimensionar a janela. Nesse caso, fill especifica se os widgets também deve crescer, o que permitirá que botões e controles aumentem de tamanho de forma estrondosa. Finalmente, *$padding* determina o menor número de pixels que deverá existir para separar verticalmente o widget de seus vizinhos. Nosso GTKTicker exibe mensagens de status no widget *Gtk:: Label* logo acima do botão *Update*. A função *set_alignmet()* usa a seguinte sintaxe

```
$STATUS->set_alignment ➥
(0.5, 0.5);
```

para alinhar o texto horizontal e verticalmente. Caso queira experimentar, utilize na horizontal o valor 0.0 para alinhar o texto à esquerda, e o valor 1.0 para alinhar o texto à direita.

Ao contrário do *Gtk::VBox*, o *Gtk:: Table* fornece aos programadores uma ferramenta para convenientemente dispor widgets em uma tabela. A função *attach_defaults()* recebe cinco parâmetros: o widget que será alinhado e duas coordenadas de linha e coluna, entre as quais o widget será colocado. Veja o exemplo abaixo:

```
$table->attach_defaults
($label, 0, 1, 1, 2);
```

$label será colocado entre a primeira linha ("entre 0 e 1") e a segunda coluna ("entre 1 e 2") da tabela chamada *$table*

Ação!

Você pode designar ações para os widgets do tipo *Gtk::Button*, e o Gtk executará a ação quando um usuário pressionar o botão. A função *signal_connect()*, na linha 177 da Listagem 2, especifica que Gtk deve enviar o evento *wake_up* ao kernel quando o usuário clicar no botão *Update*.

A janela principal também tem uma ação determinada – os usuários podem clicar no botão com o "*X*" na parte superior direita da janela para finalizar a aplicação. O código seria:

```
$w->signal_connect('destroy',
  sub {Gtk->exit(0)});
```

A sub-rotina finaliza o programa. Após definir os widgets, a função *show()* (linha 183) os exibe na janela principal.

O kernel contra-ataca

No estado *yhoo_response*, o kernel salta para função na linha 187, *resp_handler*. Por definição, quando isto acontece o *POE::Component::Client::HTTP* armazena os pacotes com a solicitação e a resposta em ARG0 e ARG1. O POE usa este modo ligeiramente estranho de passar parâmetros após a introdução de novas funções que representam constantes numéricas, tais como KERNEL, HEAP, ARG0, ARG1. Os responsáveis pelo POE esperam que os programadores as usem para indexar o array de parâmetros das funções, como @_. Por exemplo, $_[KERNEL] sempre retornará o objeto kernel, o que ajuda a manter o índice para o qual KERNEL aponta transparente.

Os pacotes de solicitação e resposta mencionado acima são apenas referências a dois arrays, cujos elementos iniciais contém objetos *HTTP::Request* ou *HTTP::Response*. O comando map na linha 190 os extrai para *$req* e *$resp*.

Se ocorre um erro de HTTP, a linha 195 gera uma mensagem apropriada no widget de status e a função retorna. Se não, o array bidimensional global de widgets *label* é atualizado. Os widgets exibem o símbolo da empresa, o valor atual de cada ação e a variação como uma percentagem. No caso da variação ser zero, ela é simplesmente ignorada.

Acorde!

Um evento *wake_up* no kernel chama a função *wake_up_handler()* definida a partir da linha 232 (veja a Listagem 2). Ela chama a função *upd_quotes()*, implementada a partir da linha 79. Esta função define um objeto *HTTP::Request* e usa um evento para enviá-lo ao componente *POE::Component::Client::HTTP*. O estado alvo da sessão ticker é então designado como *yhoo_response*.

Após completar estes passos preparatórios, *wake_up_handler()* usa a função *delay()* do kernel para enviar um alerta. Isso faz com que um evento *wake_up* ocorra na sessão *ticker* quando o número de segundos definido em *$UPD_INTERNAL* (60 segundos neste caso) tiver passado. A partir daí, o ticker irá automaticamente atualizar os valores a cada 60 segundos, sem que seja necessário pressionar o botão *Update*.

É fascinante ver o quão suave é a atualização da tela. Mesmo mexendo nos menus enquanto o aplicativo executa uma atualização automática através de uma conexão de rede lenta, a interface permanece intacta.

Problemas na instalação

É melhor usar o CPAN para instalar o POE e os módulos necessários, como *POE::Component::Client::HTTP*. Se o módulo *POE::Component::Client:DNS* for instalado, as consultas ao servidor DNS serão realizadas de forma assíncrona. Caso contrário, a função *gethostbyname()* pode causar um pequeno atraso. A instalação do Gtk a partir do CPAN causou alguns probleminhas em meu sistema. Mas basta executar:

```
touch ./Gtk/build/➥
perl-gtk-ref.pod
perl Makefile.PL ➥
--without-guessing
```

seguido de *make install* no diretório de instalação para resolvê-los. ■

INFORMAÇÕES

[1] Código deste artigo: *http://www.linux-magazine.com/Magazine/Downloads/42/Perl/*

[2] POE: *http://poe.perl.org*

[3] Jeffrey Goff, "A Beginner's Introduction to POE", 2001: *http://www.perl.com/pub/a/2001/01/poe.html*

[4] Matt Sergeant, "Programming POE", talk at TPC 2002: *http://axkit.org/docs/presentations/tpc2002*

[5] Gtkperl: *http://gtkperl.org*

[6] Tutorial de Gtkperl: *http://personal.riverusers.com/~swilhelm/gtkperl-tutorial/*

[7] Eric Harlow, "Developing Linux Applications with GTK+ and GDK": New Riders,1999,ISBN 0735700214