Programação gráfica com Qt

Uma das maiores aventuras para os programadores quem vêm do Windows é programar em modo gráfico no Linux. Já temos a esperança do Kylix, que já chegou. Mas além do Kylix, há um bom tempo já temos excelentes bibliotecas como a Qt e a Gtk. Apenas por uma questão de preferência pessoal, escolhi a Qt como minha biblioteca padrão (depois que ela se tornou GPL, é claro). Nesta série de artigos pretendo passar apenas o básico para se montar um pequeno aplicativo com direito a caixas de texto, botões e "eventos" (para o pessoal do VB e Delphi).

Requisitos: Para seguir este tutorial, será necessário um conhecimento (mesmo que bem básico) de programação orientada a objetos e C++ (sabendo C já ajuda).

Importante: Para compilar os programas e seguir os exemplos você deverá ter instalado, pelo menos, os pacotes gcc-g++ e qt2-devel. Pode ser que seja necessário algum outro, mas como cada distribuição empacota os arquivos a sua maneira, não vou poder especificar os pacotes de cada uma.

Parte 1: Hello world!

Nesta primeira parte faremos um simples "hello world". Nas partes seguintes veremos como criar nossos próprios widgets (necessário para termos mais de um componente na mesma tela), como dar funcionalidade ao programa com botões, etc., e como facilitar a vida com o Qt Designer.

Primeiro vamos fazer o tão famoso "Hello world!!!" e procurar entender o que se passa... Crie um programa hello.cpp com o seguinte conteúdo:

```
 1: /*****************************************************
 2:  * Programação gráfica com Qt
 3:  *
 4:  * Programa: hello.cpp
 5:  *****************************************************/
 6:
 7: #include <qapplication.h>
 8: #include <qlabel.h>
 9:
10: int main(int argc, char *argv[])
11: {
12:     QApplication a(argc, argv);
13:
14:     QLabel main_widget("Olá mundo!!!", 0);
15:     a.setMainWidget(&main_widget);
16:     main_widget.show();
17:
18:     a.exec();
19: }
```

Pronto? Então pecisamos compilar isso e gerar um executável (binário). Para isso, execute o seguinte comando:

g++ -I$QTDIR/include -L$QTDIR/lib -lqt -o hello hello.cpp

E, é claro, vamos executá-lo:

./hello

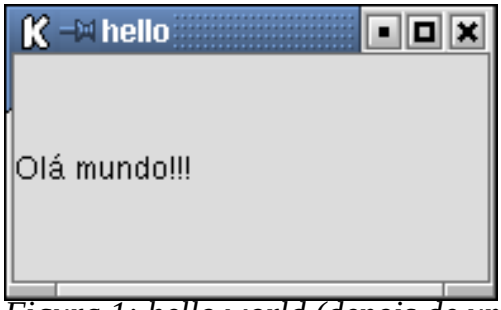

Figura 1: hello world (depois de um resize)

Bom, agora vamos explicar cada passo:

7: #include <qapplication.h>
8: #include <qlabel.h>

Em Qt cada widget (componente) ou classe tem o seu próprio .h. Assim, como vamos trabalhar com o QApplication (a aplicação Qt em si) e o QLabel (o componente usado para mostrar o "Olá mundo!!!"), declaramos isso logo no

começo.

10: int main(int argc, char* argv[])
11: {
12: QApplication a(argc, argv);

Criamos a função main(), que é por onde o programa começa, e passamos os seus argumentos para a aplicação Qt. Note que na linha 12 estamos declarando uma variável chamada a da classe QApplication.

14: QLabel main_widget("Olá mundo!!!", 0);

Aqui criamos um objeto, chamado main_window, da classe QLabel. Esse QLabel terá como texto: "Olá mundo!!!". Quanto ao 0, falaremos dele mais para frente.

15: a.setMainWidget(&main_widget);
16: main_widget.show();

Na linha 15 fazemos com que a, a aplicação, reconheça o nosso main_widget como seu widget (componente) principal. Assim, quando main_widget for fechado, a aplicação será encerrada.

18: a.exec();

Com esse a.exec(), passamos o controle do nosso programa para a Qt. Daqui para frente quem manda é ela e a função main() só será continuada quando o main_widget for fechado.

Curiosidade: Se você quiser que a frase "Olá mundo!!!" fique dentro de um botão, apenas substitua o #include <qlabel.h> por #include <qpushbutton.h> e o QLabel main_widget("Olá mundo!!!", 0) por QPushButton("Olá mundo!!!", 0) .

Colocando mais de um widget na janela

No exemplo anterior montamos apenas uma janelinha com um simples QLabel. Obviamente, qualquer aplicação tem mais de um widget na tela. A maneira para fazer isso é criar um widget que contenha todos os widgets que queremos apresentar...

Apenas um pequeno exemplo, que podemos chamar de hello2.cpp:

```
 1: /*****************************************************
 2:  * Programação gráfica com Qt
 3:  *
 4:  * Programa: hello2.cpp
 5:  *****************************************************/
 6:
 7: #include <qapplication.h>
 8: #include <qwidget.h>
 9: #include <qlabel.h>
10:
11: int main(int argc, char *argv[])
12: {
13:     QApplication a(argc, argv);
14:
15:     QWidget main_widget(0);
16:     QLabel label1("Olá mundo!!!", &main_widget);
17:     QLabel label2("Bom dia!", &main_widget);
18:     label1.move(20, 0);
19:     label2.move(20, 20);
20:     a.setMainWidget(&main_widget);
21:     main_widget.show();
22:
23:     a.exec();
24: }
```

Agora compilamos e executamos da mesma maneira:

g++ -I$QTDIR/include -L$QTDIR/lib -lqt -o hello2 hello2.cpp
./hello2

E veremos uma janela como esta:

Figura 2: Uma janela com mais de um widget

Ok... novamente, vamos às explicações (não serão explicadas as linhas já comentadas no exemplo anterior):

8: #include <qwidget.h>

Como não vamos trabalhar com apenas um widget, criamos um widget básico apenas para "segurar" os outros.

15: QWidget main_widget(0);
16: QLabel label1("Olá mundo!!!", &main_widget);
17: QLabel label2("Bom dia!", &main_widget);

Na linha 15 criamos esse widget básico e o chamamos de main_widget (e novamente um número zero). Já nas linhas 16 e 17 criamos dois QLabel: label1 e label2. Note que no lugar do número zero temos &main_widget, indicando que os nossos QLabel pertencerão ao main_widget. Como o main_widget não pertence a ninguém, coloca-se apenas um 0.

18: label1.move(20, 0);
19: label2.move(20, 20);

Com essas duas linhas, posicionamos os QLabel para que um não sobreponha o outro.

Importante: A forma que usei para demonstrar como colocar mais de um widget na mesma janela não é apropriada para desenvolver uma aplicação. O correto seria

aproveitarmos o poder da programação orientada a objetos e criar uma classe (um novo widget) separada para cada janela. Algo como:

classe minhaJanela (derivada de QWidget)
** QLabel *label1;**
** QLabel *label2;**
** QPushButton pushbutton1;**
fim_classe

Mas veremos uma explicação melhor sobre isso na próxima parte desta série artigo.

Autor: Ricardo Vaz Mannrich
Email:
husk-le@comlinux.com.br