3.GUI Programming

Aim :

Learn to write GUI programs using FOSS tools in Linux.

Introduction :

GUI programming on Linux is based on the X Window system. X provides windowing on computer displays and manages keyboard, pointing device control functions and touch-screens. X provides the basic framework, or primitives, for building such GUI environments: drawing and moving windows on the display and interacting with a mouse, keyboard or touchscreen.

X uses GUI toolkits to help the development of GUI programs. **GUI toolkit** is a set of widgets for use in designing applications with graphical user interfaces (GUIs). GUI toolkits can be based on different languages. The most commonly used languages are C/C++. Often the toolkits provide higher level language bindings which allow the user to program GUIs using an easier to use higher level language like Python, Ruby etc.

There are two popular GUI toolkits which are widely used today. They are Qt and GTK+. Qt is written in C++ while GTK+ is written in C. Both toolkits provide bindings in a wide variety of languages.

For users who are familiar with Visual basic, Gambas is a full-featured object language and development environment built on a BASIC interpreter which runs on top of the X window system. It in turn relies on the underlying GUI toolkits (both Qt and GTk+).

This exercise consists of developing GUI programs using the Qt toolkit.

Description :

Students will learn to setup their systems to develop using Qt and write 5 programs and test their results. The programs are of increasing complexity and introduce the students to more and more concepts.

All the programs will be purely code drive, ie. the user interface is developed entirely using C++ code. None of the visual GUI builders will be used for these exercises.

Pre-requisites:

1. In the comandline as root, type

```
> yum install qt-devel qt-demos qt-examples qt-doc
```

The Programs:

A GUI Hello World.

- Create a new directory for the program.

```
> mkdir qthello
```

- Go to the newly created directory.

```
>  cd qthello
```

- create the file in the qthello directory.

```
> gedit qthello.cpp
```

```
//qthello.cpp
//Include file for Qt
#include <QtGui>
int main(int argc, char *argv[])
{
     QApplication app(argc, argv);
     QWidget window;
      //resize window to 320x240
     window.resize(320, 240);
     window.setWindowTitle("Hello World!");
      //Show the window
     window.show();
      //Start the event loop
     return app.exec();
}
```

- Once the file has been saved, use the following commands to compile and execute the program.

```
> qmake-qt4 -project
> qmake-qt4
```

```
> make
> ./qthello
```

- The output will be a window like this:

Window with a button.

- Use the following code to create a window with a button:
- Create a new directory for the program.

```
> mkdir qtbutton
```

- Go to the newly created directory.

```
>  cd qtbutton
```

- create the file in the qtbutton directory.

```
> gedit qtbutton.cpp
```

```
#include <QtGui>

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QWidget window;
```

```
    window.resize(320, 240);
    window.setWindowTitle("Window with a button");
    window.show();

    //Create a button with the text "Press me"
    //as a child of the main window
    QPushButton *button = new QPushButton("Press me", &window);
    //move the button into position
    button->move(100, 100);
    button->show();
    return app.exec();
}
```

- Once the file has been saved, use the following commands to compile and execute the program.

```
> qmake-qt4 -project
> qmake-qt4
> make
> ./qtbutton
```

- The output will be a window like this:

Using Layouts.

- Create a window with two widgets managed by a layout manager.
- Create a new directory for the program.

```
> mkdir qtlayout
```

- Go to the newly created directory.

```
> cd qtlayout
```

- create the file in the qtlayout directory.

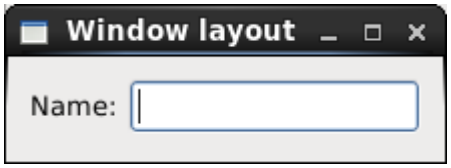
```
> gedit qtlayout.cpp
```

```
//qtlayout.cpp
#include <QtGui>

 int main(int argc, char *argv[])
 {
      QApplication app(argc, argv);
      QWidget window;

      //Create a label an a single line text box
      QLabel *label = new QLabel("Name:");
      QLineEdit *lineEdit = new QLineEdit();

      //Create a layout. Add the label and the lineedit to it.
           QHBoxLayout *layout = new QHBoxLayout();
           layout->addWidget(label);
           layout->addWidget(lineEdit);
      //Apply the layout to the main window.
      //Since the widgets are part of the layout,
      //they are now children of the window.
      window.setLayout(layout);
      window.setWindowTitle("Window layout");
      window.show();
      return app.exec();
 }
```

- Once the file has been saved, use the following commands to compile and execute the program.

```
> qmake-qt4 -project
> qmake-qt4
> make
> ./qtlayout
```

- Compile and execute the program to get the following:



Signals and Slots

- This program demonstrates the use of signals and slots to make two widgets interact with each other.

The entire application is divided into 3 files:

1. communicate.h
2. communicate.cpp
3. main.cpp

- Create a new directory for the program.

```
> mkdir qtsignals
```

- Go to the newly created directory.

```
>  cd qtsignals
```

- create the file in the qtsignals directory.

```
> gedit communicate.h
```

```
//communicate.h
#include <QWidget>
#include <QApplication>
#include <QPushButton>
#include <QLabel>

class Communicate : public QWidget
{
//The Q_OBJECT macro causes the moc tool to initialise
//code for signals and slots, run time type information
//and dynamic property system
  Q_OBJECT

  public:
     Communicate(QWidget *parent = 0);
```

```
//add a lot which allows widget communications
  private slots:
      void add();

  private:
      QLabel *label;

};
```

- create the file in the qtsignals directory.

```
> gedit communicate.cpp

//communicate.cpp
#include "communicate.h"
#include <QDesktopWidget>

Communicate::Communicate(QWidget *parent)
      : QWidget(parent)
{
  resize(180, 140);

  QPushButton *plus = new QPushButton("+", this);
  plus->setGeometry(50, 40, 50, 30);

  label = new QLabel("0", this);
  label->setGeometry(120, 40, 20, 30);

//Connect the clicked event of the button to
//the add method of the class
  connect(plus, SIGNAL(clicked()), this, SLOT(add()));
}

void Communicate::add()
{
//Change the text displayed in the label
  int val = label->text().toInt();
  val++;
  label->setText(QString::number(val));
}
```

- create the file in the qtsignals directory.

```
> gedit main.cpp
```

```
//main.cpp
#include "communicate.h"

int main(int argc, char *argv[])
{
  QApplication app(argc, argv);

  Communicate window;

  window.setWindowTitle("Communicate");
  window.show();

  return app.exec();
}
```

- Once the file has been saved, use the following commands to compile and execute the program.

```
> qmake-qt4 -project
> qmake-qt4
> make
> ./qtsignals
```

On compiling and running the program, the following window is displayed. The number can be incremented by pressing the plus button.

Menus and Toolbars.

- This program will display a menu which can be used to close the program. Once again we have 3 files as part of the program:
    1. mymenu.h

2. mymenu.cpp
3. main.cpp

- Create a new directory for the program.

```
> mkdir qtmenu
```

- Go to the newly created directory.

```
>  cd qtmenu
```

- create the file in the qtmenu directory.

```
> gedit mymenu.h
```

```
//mymenu.h
#include <QMainWindow>

class MyMenu : public QMainWindow
{
  public:
      MyMenu(QWidget *parent = 0);
};
```

- create the file in the qtmenu directory.

```
> gedit mymenu.cpp
```

```
//mymenu.cpp
#include "mymenu.h"
#include <QMenu>
#include <QMenuBar>
#include <QApplication>

MyMenu::MyMenu(QWidget *parent)
      : QMainWindow(parent)
{
  //create the quit action object
  QAction *quit = new QAction("&Quit", this);

  //create the file menu
  QMenu *file;
  file = menuBar()->addMenu("&File");
```

```
  //add the quit action to the new menu
  file->addAction(quit);

  //connect the triggered signal from the quit action menu
  //to the global quit method which closes the application
  connect(quit, SIGNAL(triggered()), qApp, SLOT(quit()));

}
```

- create the file in the qtmenu directory.

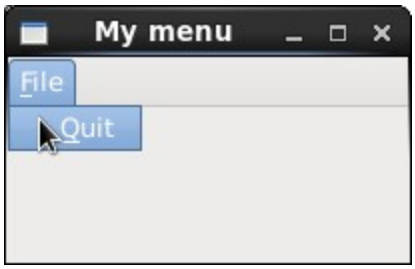
```
> gedit main.cpp
```

```
//main.cpp
#include "mymenu.h"
#include <QDesktopWidget>
#include <QApplication>


int main(int argc, char *argv[])
{
  QApplication app(argc, argv);

  MyMenu window;

  window.setWindowTitle("My menu");
  window.show();

  return app.exec();
}
```

- Once the file has been saved, use the following commands to compile and execute the program.

```
> qmake-qt4 -project
> qmake-qt4
> make
> ./qtmenu
```

- On running the program we get a window with a menu which allows us to close the program.



References:

1. http://doc.qt.nokia.com/ -- Qt documentation site.
2. http://zetcode.com/tutorials/qt4tutorial/ -- A Qt4 tutorial