Hack a Mobile Phone with Linux and Python

A mobile phone is a cool gadget to play with, especially when I can run my favourite programming language (no prize for guessing what it is!) on it! That was the logic which made me purchase a Nokia Series 60 smartphone, the N-Gage QD. This article describes a few experiments I did with the mobile - like setting up Bluetooth communication links, writing Python/C code and emulating serial ports.

Bluetooth on Linux

Bluetooth is a short distance wireless communication standard. It is commonly used to facilitate data transfer between PC's and cell phones/PDA's without the hassle of `wired' connections. The hardware which provides Bluetooth connectivity on the PC is a small device called a `USB-Bluetooth dongle' which you can plug onto a spare USB port of your machine. I approached the local electronics dealer asking him for such a device and got one which didn't even have the manufacturer's name printed on it. The driver CD which came with it of course contained only Windows software. Deciding to try my luck, I plugged the device on and booted my system running Fedora Core 3 - bluetooth service was started manually by executing:

```
sh /etc/init.d/bluetooth start
```

Here is the output I obtained when the command `hciconfig' ( which is similar to the `ifconfig' command used to configure TCP/IP network interfaces) was executed:

```
hci0:   Type: USB
        BD Address: 00:11:B1:07:A2:B5 ACL MTU: 192:8  SCO MTU: 64:8
        UP RUNNING PSCAN ISCAN
        RX bytes:378 acl:0 sco:0 events:16 errors:0
        TX bytes:309 acl:0 sco:0 commands:16 errors:0
```

My no-name USB-Bluetooth dongle has been detected and configured properly! The number 00:11:B1:07:A2:B5 is the Bluetooth address of the device.

Detecting the mobile

The next step is to check whether Linux is able to sense the proximity of the mobile. If your phone has bluetooth disabled, enable it and run the following command (on the Linux machine):

```
hcitool scan
```

Here is the output obtained on my machine:

```
Scanning ...
        00:0E:6D:9A:57:48       Dijkstra
```

The `BlueZ' protocol stack running on my GNU/Linux box has `discovered' the Nokia N-Gage sitting nearby and printed its Bluetooth address as well the name which was assigned to it, `Dijkstra'.

Pairing the mobile

For security reasons, some interactions with the mobile require that the device is `paired' with the one it is interacting with. First, store a number (4 or more digits) in the file /etc/bluetooth/pin (say 12345). Stop and restart the bluetooth service by doing:

```
sh /etc/init.d/bluetooth stop
sh /etc/init.d/bluetooth start
```

Now initiate a `pairing' action on the mobile (the phone manual will tell you how this is done). The software on the phone will detect the presence of the Bluetooth-enabled Linux machine and ask for a code - you should enter the very same number which you have stored in /etc/bluetooth/pin on the PC - the pairing process will succeed.

Transferring files

Files can be transferred to/from the Linux machine using a high level protocol called OBEX (standing for OBjectEXchange, originally designed for Infrared links). First, you have to find out whether the mobile supports OBEX based message transfer. Try running the following command on the Linux machine (the number is the bluetooth address of the phone):

```
sdptool browse 00:0E:6D:9A:57:48
```

You might get voluminous output - here is part of what I got:

```
Service Description: OBEX Object Push
Service RecHandle: 0x10005
Service Class ID List:
  "OBEX Object Push" (0x1105)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 9
  "OBEX" (0x0008)
```

OBEX is built on top a lower-level protocol called RFCOMM. The `Object Push' service uses RFCOMM `channel' 9. Let's try to upload a file to the phone; run the following command on the Linux machine:

```
obex_push 9 00:0e:6d:9a:57:48 a.txt
```

The phone will respond by asking you whether to accept the message coming over the bluetooth link. The same command, invoked without any option, can be used to receive files sent from the mobile over the bluetooth link (read the corresponding `man' page for more details).

Installing Python

Nokia has recently done a port of Python to the `Series 60' smartphones running the Symbian operating system. The Python interpreter as well as a few important modules are packaged into a single .sis file (somewhat like the Linux RPM file) which can be obtained from http://www.forum.nokia.com/main/0,,034-821,00.html. The file to be installed is named PythonForSeries60_pre_SDK20.SIS. The first step is to transfer this file to the mobile via obex_push. Trying to open the file on the mobile will result in the Nokia installer program running - it will ask you whether to install Python on the limited amount of memory which the phone has or to an additional MMC card (if one is present). Once the installation is over, you will see a not-so-cute Python logo on the main menu of the phone - Figure 1 is a screenshot I took of the main menu.

Figure 2 shows the interactive Python prompt at which you can try typing Python scripts!

Running the Python `Hello, World'

You can write Python scripts on the Linux machine and upload them to the mobile with `obex_push'. If you try to open these scripts (on the mobile), the `applications manager' will ask you whether to install the files as Python scripts or not. Once installed as scripts, you can execute them by following the instructions displayed on the screen when you open the `Python' application from the main menu.

Figure 3 shows the output obtained by installing and running the following script on the mobile:

```
import appuifw # The application UI framework
appuifw.app.title = u'Cool Python'
appuifw.note(u'OK', 'info')
```

Socket programming

Application programs running on both the phone as well as the Linux machine interface with the Bluetooth protocol stack via the socket API. Listing 1
shows a simple client program running on the mobile which connects with a server running on the Linux machine and sends it a message; the server code is shown in Listing 2.

The Python client program running on the mobile opens a Bluetooth socket and connects to the PC whose device address is specified in the variable `ATHLON'. Once the connection is established, it simply sends a string `Hello, world'.

The server program running on the PC opens a Bluetooth stream socket, binds it to RFCOMM channel 4 and calls `accept' - the server is now blocked waiting for a connection request to arrive from the client. Once the request arrives, the server comes out of the accept, returning a `connected' socket calling `recv' on which will result in the server getting the string which the client had transmitted.

The `bacpy' function in the server program is defined as an inline function in one of the header files being included - so you need not link in any extra library to get the executable. But if you are using any of the other Bluetooth utility functions like `ba2str', you have to link /usr/lib/libbluetooth.so to your code.

Using PyBlueZ

There is an interesting Python interface to the Bluetooth library in Linux called `PyBlueZ' available for download from http://org.csail.mit.edu/pybluez. It simplifies the process of writing bluetooth socket programs on the Linux machine. Listing 3

shows the Python implementation of the server program described in the previous section.

Emulating serial links

Programs like `minicom' are used to talk to devices connected over a serial link (say a modem). There is a neat software trick to present a `serial-port-like' view of a bluetooth link so that programs like `minicom' can manipulate the connection effortlessly. Let's try it out.

First, edit /etc/bluetooth/rfcomm.conf so that it looks like the following:

```
rfcomm0 {
        bind no;
        device 00:0e:6d:9a:57:48;
        channel 1;
        comment "Example Bluetooth device";
}
```

After stopping and restarting the bluetooth service, run the following command:

```
rfcomm bind /dev/rfcomm0
```

You should see a file called `rfcomm0' under /dev after executing the above command. Now, you can set up `minicom' by running:

```
minicom -m -s
```

The only thing to do is to set the name of the device to connect to as /dev/rfcomm0. Save the new configuration as the default configuration and invoke:

```
minicom -m
```

Minicom is now ready to talk to your phone! Type in `AT' and the program will respond with an `OK'. Say you wish to make your phone dial a number. Just type:

```
atdt 1234567;
```

There are many other AT commands you can experiment with; try googling for say `mobile phone AT commands' or something of that sort!

After you have finished with your virtual serial port manipulations, you should run:

```
rfcomm release /dev/rfcomm0
```

to `release' the serial-bluetooth link.

Python over a Bluetooth console

Once you get the serial port emulation working, there is another interesting hack to explore. The Nokia Python distribution comes with a program called `btconsole.py'. On one console of your Linux machine, run the command:

```
rfcomm listen /dev/rfcomm0
```

Now run `btconsole.py' on the phone. You will see that after a few seconds, `rfcomm' will respond with a `connected' message. Once you get this message, take another console and run:

```
minicom -m
```

What do you see on the screen? A Python interactive interpreter prompt! You can now type in Python code snippets and execute them on the phone on-the-fly! Isn't that cool?

Parting Thought

I was curious to know how Microsoft's Windows XP operating system, famous for its `ease of use', would compare with Linux when it comes to interacting with my NGage QD. I installed the Windows driver for my no-name usb-bluetooth dongle and tried to get the Nokia PC suite up and running on an XP machine - maybe it's because I am far more experienced in GNU/Linux than on MS operating systems, but I found the XP experience far less `friendly' than MS would care to admit. I believe that most of the `user friendliness' of the Microsoft operating system comes from hardware vendors and application developers tightly integrating their products with the platform rather than any inherent quality of the OS as such.

References

For a general introduction to Bluetooth technology, see http://www.dell.com/downloads/global/vectors/2003_bluetooth.pdf. An interesting paper on Bluetooth security is available at http://www.niksula.cs.hut.fi/~jiitv/bluesec.html.

http://www.holtmann.org/ has plenty of information regarding Bluetooth and Linux; I found the document `Bluetooth Programming for Linux' (http://www.holtmann.org/papers/bluetooth/wtc2003_slides.pdf) very informative.

Lots of information about Python on series 60 mobiles is available at http://www.postneo.com/postwiki/moin.cgi/PythonForSeries60/. ObexFTP seems to be an interesting tool - you can get it from http://triq.net/obex/. There are some documents floating on the net which describe how you can do an NFS mount of your phone's file system - try a google search for more info.

Source code/errata concerning this article will be available at http://pramode.net/lfy-jun/.