Georgia Institute of Technology
College of Engineering
School of Electrical and Computer Engineering

ECE 4005
Internetwork Programming

Design Project

Final Report

Fall 2000

Ballard Johnson
Chalon Clemons
Mohammad Braiwish
Naima Nazir

Date: November 30, 2000
7.6  ECONOMICAL ANALYSIS ........................................................................................................ 20

8.  CONCLUSION ...................................................................................................................... 20

9.  DOCUMENTATION AND SOURCES ........................................................................... 20

10. APPENDICES ............................................................................................................... 21

  10.1 SOURCE CODE ........................................................................................................ 21-28
  10.2 USER’S MANUAL FOR SOFTWARE ............................................................................. 29-30
  10.3 USER’S MANUAL FOR GUI ..................................................................................... 30-38
  10.4 MODIFIED BLUETOOTH SOURCE CODE .............................................................. 38-103
  10.5 PRESENTATION SLIDES ....................................................................................... 104
1. INTRODUCTION

Bluetooth is the name given to a new technology using short-range radio links, intended to replace the cable(s) connecting portable and/or fixed electronic devices. It is envisaged that it will allow for the replacement of the many propriety cables that connect one device to another with one universal radio link. Its key features are robustness, low complexity, low power and low cost. Designed to operate in noisy frequency environments, the Bluetooth radio uses a fast acknowledgement and frequency-hopping scheme to make the link robust. Bluetooth radio modules operate in the unlicensed ISM band at 2.4GHz, and avoid interference from other signals by hopping to a new frequency after transmitting or receiving a packet. Compared with other systems in the same frequency band, the Bluetooth radio hops faster and uses shorter packets. Bluetooth is a protocol for the efficient transmission of data between devices, which are Bluetooth compliant. Essentially Bluetooth is a technology, which allows for smooth data transmission between devices without the use of cables, i.e., it is a cable-less technology.

2. OBJECTIVE

We, as a group, are assigned to demonstrate this new accomplished technology. Our group is consisted of four members. The names are Ballard Johnson, Chalon Clemons, Mohammad Braiwish, and Naima Nazir. The prototype that we are trying to achieve is to get two separate PCs to connect wirelessly and be able to exchange data from one another using Bluetooth technology. We will be sending certain files to send from one computer to the other, and make sure that the other computer receives the file.

3. PROJECT DESIGN

3.1 Original Plan

The original project goal was to demonstrate the Bluetooth technology by transferring files from one computer to another computer wirelessly. Certain files would be transferred from one computer to another, and checked to make sure they were received properly. This was planned to be accomplished by getting two PCI cards and installing them into the CPU. These PCI cards would enable the CPUs to talk to each other wirelessly; in another words the CPUs will be Bluetooth enabled. All this would be done in Windows environment. A graphical user interface (GUI) would be written using Visual Basic to make the appearance of the transfer user friendly. Below is the PCI card that was intended to be bought.
3.2 Revised Version

As further research was performed several realizations were made about the setup of the project. The original project involved transferring files between two computers with Windows compatible PCI cards equipped with the Bluetooth technology, as discussed above. Since no companies are currently selling Bluetooth enabled PCI cards, and Bluetooth development kits are so expensive, the project structure had to be reconsidered. After several days of research on Axis Communications’ website, the discovery was made that they sell development boards with their Bluetooth protocol stack available to be installed. The Bluetooth technology can be emulated using two of the developer boards from Axis with a null modem cable connected to both. Since the developer boards are used, the operating system originally planned to be used had to be changed to operate properly on the development boards, i.e. from Windows to Linux. The initial plan was to use Visual Basic to create the Graphical User Interface (GUI) for Windows. Since Linux is being used, Glade will be used to do this particular task. Another change that the project underwent was due to the fact that Bluetooth technology makes the use of Radio Frequency (RF) signals to communicate between two machines. As mentioned before, the hardware to implement Bluetooth technology was not available due to which unfortunately this technology could not be implemented. Hence, the use null cable modem was made. Following figure shows how this developer board looks like.
3.2.1 New Project Components

The Axis Developer Board introduces a way to get acquainted with embedded Linux and Ethernet networking. The Axis developer board is similar to a single board computer. It has a 100 MHz 32-bit RISC CPU with 8 Mbytes DRAM and 2 Mbytes of nonvolatile flash memory. It contains a serial interface including two RS-232 ports to be used to connect two developer boards together using a null cable modem. Axis’ port of Linux, called elinux, is an operating system created specifically for embedded systems with a very small memory footprint. It is based on the 2.038 Linux kernel modified to work with Memory Management Unit (MMU) less processors. These changes made to the kernel are optimal for the developer boards we are using. Using Linux as the operating systems gives access to a wide range of open source applications. Porting existing Linux applications to the developer board is in many cases done by simply recompiling the existing code for the developer board. The software of the developer board is upgradeable over any standard TCP/IP network using FTP. When developing applications, simply compile the applications on the Linux workstation and transfer them to the developer board for the final testing. With an external Bluetooth module supporting the transport layer connected to one of the serial ports on the developer board, it is possible to have our own Bluetooth application up and running.

3.2.2 Project Tradeoffs

Although Axis provides a Bluetooth protocol stack and driver, Bluedrekar, a Bluetooth protocol stack and driver created by IBM for Linux, was also considered for the project. The pros and cons were weighed to decide which one would best suit the project’s needs. Bluedrekar offers sample applications to help people develop their own applications, which would help us to create our applications. Axis’ Bluetooth driver provides you with the source code to make any changes, while Bluedrekar only provides the executable. The Axis’ Bluetooth driver can emulate the Bluetooth technology without actually having Bluetooth enabled hardware, while Bluedrekar does not. Since no Bluetooth hardware is currently available to be used in the project, the Axis’ Bluetooth driver was chosen.
The use of the developer boards was also considered to determine if it would best suit the needs of the project. The developer boards would provide experience in working with the Linux operating system. It would also provide a way to get acquainted with embedded Linux since the boards use the elinux operating system. Since most of the group involved in the project is already familiar with Windows, there would not be much of a learning experience. It would also provide experience with Ethernet Networking, since the software on the boards is transferred using an Ethernet connection to a computer, whereas Windows would not provide that exposure. Although the developer boards do not use RF signals to transfer the files, the Bluetooth technology is still demonstrated.

One question that was considered was “How will these changes affect the rest of the project structure?” The application that will be written to transport files between the two developer boards will not be affected since elinux comes with a C compiler. So far no problems are foreseen with the GLADE software we are using for the Graphical User Interface (GUI), since most applications used on Linux can be recompiled on elinux and used. We have drawn the conclusion that this major change will not affect the rest of our project.

4. SUPPLIES

The supplies needed for this project are as follows:
1. Two axis developer boards.
2. Three null cable modem (female serial ports on both sides).
3. Two personal computers with Linux operating system.
4. Bluetooth software compatible with Linux environment.
5. Elinux for the embedded systems on the developer board.
6. Glade software to create the GUI.

5. PROJECT’S OVERVIEW

5.1 Research

None of the group members were familiar with Bluetooth technology. Hence, we all needed to get a comprehensive overview on this technology. The website that was most useful for getting a good conception and a synopsis on Bluetooth technology is listed below.

http://new.topsitelists.com/topsites.cgi?ID=52&user=bluetooth&area=bestsites

It took us a while to go through each and everyone of the top twenty sites listed in the above hyperlink. By going through these top twenty sites, we learned about how Bluetooth technology is fairly new; we learned about the different applications of Bluetooth; we learned that there are several companies working on this technology; we learned that there is not much out on Bluetooth yet, meaning Bluetooth hardware as well as software are not available widely in the market; last but not least, we learned the structure of Bluetooth technology. Below are few examples of Bluetooth technology’s applications.
5.1.1 Bluetooth Physical System

The Bluetooth system consists of a radio unit, a link control unit and a support unit for link management and host terminal interface functions.

Radio

The Bluetooth air interface is based on a nominal antenna power of 0dBm. Spectrum spreading is accomplished by frequency hopping in 79 hops displaced by 1 MHz, starting at 2.402GHz and finishing at 2.480GHz. The nominal link range is 10cm to 10 m, but can be extended to 100m by increasing the transmit power.

Link Controller

The LC carries out the Base band protocols and other low-level link routines of the Bluetooth system. It uses the base band to establish network connections, define link + packet types and provide error correction.

Link Manager

This software entity carries out link setup, authentication, link configuration and other protocols. It discovers other remote LM’s and communicates with them via the
Link Manager Protocol (LMP). To perform its service provider role, the LM uses the services of the underlying Link Controller (LC).

**Software Functions/Framework**
Different Bluetooth devices have different requirements, and the function of the software functions is to meet these requirements. These are defined in the Bluetooth protocol stack, and range from radio module compliance and air protocols, up to application-level protocols and object exchange protocols. Here is where functions such as cable emulation, audio communication etc. are implemented.

### 5.1.2 Protocol Stack
The Bluetooth protocol stack can be broken into 4 types:
1. **Bluetooth Core Protocols**: Base band, LMP, L2CAP and SDP
2. **Cable Replacement Protocol**: RFCOMM
3. **Telephony Control Protocols**: TCS Binary, AT-commands
4. **Adopted Protocols**: Everything else (excluding Audio)

As can be seen in the figure above. The complete protocol stack comprises of both Bluetooth-specific protocols like LMP and L2CAP, and existing protocols like OBEX and UDP.

### 5.2 Hardware
The elinux operating system which was to be installed on the board was downloaded from the axis website at [http://developer.axis.com/download/elinux/](http://developer.axis.com/download/elinux/). The directions in the README were followed and the OS was successfully installed. The tools needed to install the operating system was downloaded from the axis website at [http://developer.axis.com/download/tools/](http://developer.axis.com/download/tools/).
Once we got our hands on the developer board preparations were made to try and boot up the board. The developer board was logged into by using minicom and a null cable modem connected to a computer running Linux. In order to use minicom to login in to the board some settings had to be changed. Ctrl-A Z to get to the options screen. The option O was selected for configuring minicom. The serial port option was selected. At the “Change which Setting” prompt, e was typed. I, L, V and W for 115200 for speed, 8 Bits Data, W for 1 stop bit. Then Enter, Enter, and Exit were pressed and selected.

The Axis prompt was displayed which allowed me to login to the board. The IP address was then changed by editing the /etc/network/network.conf file using easyedit installed on the board.

After the IP address was changed, attempts were made to boot up the board with elinux. The directions were followed on the sheet that arrived with the boards. The directions did not specify that the board had to be connected to a computer by a null cable modem, although it was inferred. In order to use the boot_elinux program on the developer board, the board had to be connected on the same network as the computer to work. The board had to be connected by null cable modem and Ethernet for the program to boot the board with elinux. An attempt was made to boot the elinux on the board, but afterwards it was not possible to login anymore. No other progress was made with the developer boards, since time did not allow us.

5.3 Software

Red Hat version 6.0 was installed on the two portable hard drives we got. The Bluetooth stack software was found at the axis developer’s website. This website also provided a driver for the stack to enable a user to send data between two computers for the Linux environment. This stack and driver were downloaded from the developer website, the URL being the following.

http://developer.axis.com/download/bluetooth

The Bluetooth stack from Axis is a group of C files used to implement the different stacks of the Bluetooth protocol. The current driver provided with the stack sends a group of random characters as a string to the Bluetooth stack, which then allows for them to be sent to another Bluetooth enabled device. The group decided to change the driver to allow it to send data files across one computer to another computer. In order to accomplish this, the source code for the driver had to be examined.

The first thing that was done was that the stack and driver was downloaded and installed on each hard drive by following the directions in the README file which is placed in the bluetooth directory created by the bluetooth tar file. When the driver was executed by typing ./btd –e 0 –i no_hw for the server and ./btd –e –i no_hw –r client for the client, an error was returned that said “pppd: no such file exists”. It was then investigated to see what the pppd file is. The difference between the server and client is that the server is what is connected to and data sent to.

It was then discovered that pppd was a ppp daemon, which is what the Bluetooth stack runs on top of to work. A search was made to obtain a ppp daemon off of the internet. The pppd was downloaded from http://home.sprynet/~minaret/pppoe and
installed on each hard drive. After reading the README file it was discovered that the kernel had to be rebuilt for the version of pppd that was being used, because the drivers needed to operate it were out of date. The kernel was then rebuilt by following the directions in the README file located in the /usr/src/linux directory.

After rebuilding the kernel, the daemon was then completely installed. The stack was executed again and operated with no errors. The stack was then tested to see if info was actually sent between the two computers. After reading the README file, it was realized that a /proc file was created by the stack to show the user how much data was received or sent from each computer, which was the only way to verify data had actually been transferred. The /proc file was examined after data was sent to the server computer to ensure that data was transferred.

After verification that data was being sent, the code was then reviewed to determine where the data is received on the server side of the computer. After examining the code for the stack, the realization was made that the stack could be operated in two ways. It could be used as a kernel module, in which the entire stack was compiled into a module and inserted inside the kernel. It could also be used as a stack in userspace. Our group was originally using the stack in kernel mode unknowingly. Because the stack was in kernel mode, the data was not printed anywhere on the computer. After realizing this we realized that we would have to use the stack in userstack mode since there are extreme limitations on writing from kernel space to userspace.

The first version of the stack we downloaded and compiled was the 20000814 Version. The userstack mode was set and an attempt was made to use the Bluetooth stack and driver by typing ./btduser –e 0 –i no_hw for the server and ./btduser –e 0 –i no_hw for the client. An error repeatedly occurred stating that the RF layer of the stack could not be connected on both computers. An email was sent to the developers of the stack to identify the problem, but no solution was determined. This development set back the schedule of the project. A newer version of the stack was created on Oct 31, in which the userstack did operate properly.

After downloading and installing the new stack, it was then tested to see if this version worked by typing:

./btduser –e 0 –l no_hw –cmdmode
./btduser –e 0 –l no_hw –r client

Since the stack utilizes the serial port as the physical layer of the stack to transfer the data when an RF Bluetooth module is not available, the focus was set on first creating a script to transfer files via the serial port and then use that code inside the Bluetooth driver. After examination of a program named miniterm the knowledge was obtained on how to write to a serial port.

After this was accomplished, steps were made to integrate the serial port code with the bluetooth driver code. The send function on the driver was changed by now accepting a path to a file instead of the number of bytes and the number of times for it to be sent. A string was sent ahead of each file to indicate the path and size of the file. On the receiving end the code was modified to look for a string starting with special characters to indicate a file was about to be sent. When this string was detected, it then opened up a file with the same name and read all following data until the number of nutes of the file was reached, instead of just sending the data up to the layers and discarding it. The code was changed on both harddrives to allow both computers to send data to each
other. The new code was then tested by sending files of various sizes between the two computers.

5.3.2 Existing Version
This was available on the internet on the developer site. The version which was ultimately used was 20001031bluetooth.tgz.

5.3.2 Modified Version
The modified version is attached in the appendices

5.4 GUI
It was set in the original plan of this project to include a Graphical User Interface (GUI) to control and demonstrate our main project through it. Since our project is based on Linux, we had few GUI builders to choose from. There were three toolkits that would work for our purposes, GTK+, Qt, and Tcl/Tk. However Tcl/Tk is based on the scripting language, Tcl, in which none of our group members is experienced. Consequently, to avoid learning a new programming language, we focused mainly on the C/C++ based solutions, GTK+ and Qt. GTK+ is widely supported in the Linux community and it has an excellent, user friendly development tool kit that is called Glade. Therefore we spent little time to try to look for another GUI builder.

Glade is a program that is used to build UI in a similar way to how Visual Basics functions. It has many built-in GTK widgets that can be used to construct windows with functions that are modified in the source code. In other words, Glade allow the user to manipulate built-in icons, functions, and buttons whose functionality can be modified through callbacks and signal handling. Some of the strengths of Glade are:

- Glade will not overwrite any of your code. Designing the GUI and programming the application is separated. However, this might also be considered as a weakness.
- Glade is pure C/C++. You can customize the indentation style of its output via the 'Source-Writer' class.
- You can change the naming of files/widgets via the 'Naming' class.

Glade inherits widget code generation functions (in the src/writers dir).

One the other hand, some of the weaknesses of glade are:

- Using different file naming schemes is not supported (without patching glade).
- Source-Writer is not yet runtime customizable (indentation of generated programs).
- Gnome support is not fully developed yet.
- Widgets with duplicated names really confuse Glade.

Glade and all of its necessary components were installed without any troubles. The tutorial provided in the class web page guide you step by step in installation process.
However, some minor changes were made to that tutorial because of the new versions of Glade and some of its components. The modified version of that tutorial is shown in Appendix C. Also included in Appendix C is a tutorial for beginners of how to use glade. Some of the sites from which Glade and some components can be installed are:

- [http://glade.pn.org](http://glade.pn.org)
- [http://www.redhat.com/support/errata/RHEA1999059-01.html](http://www.redhat.com/support/errata/RHEA1999059-01.html)
- [www.seindal.dk/rene/gno](http://www.seindal.dk/rene/gno)
- [http://home.wtal.de/petig/Gtk/abstract](http://home.wtal.de/petig/Gtk/abstract)

**Design of the GUI:**

As mentioned previously, our application consisted mainly of sending files across two Linux machines through a Bluetooth stack. Thus, the main functionality of our GUI is to be able to select files and send them after specifying the destination. Also, the functionality of receiving files and indicating the source should be supported. The main window of our GUI is shown in the following figure.

![GUI Figure](image-url)

As shown in the previous figure, the GUI built for our project is very simple. It allows the user to browse through and choose a file whose name will be displayed in the display bar. The OK button sends the selected file to the selected IP address. The “CANCEL”
button clears the selected file. There are also 2 display-rooms where all the sent and received files are displayed.

The other window supported by the GUI is shown in next figure.

In this window, the user can specify the IP address to which he/she wants to communicate with. It also includes a window that shows all the IP address that have been used recently.

The first problem that we faced in designing and building the GUI was to determine how the GUI should look like. The basic plan was to build a GUI that is as
simple as possible but still do the job. If that is achieved, then we can add some functions to it. The next problem was to learn how to use Glade to build the GUI. Since none of us was familiar with Glade, it took us a while to learn how to operate and use it. Due to the simplicity of Glade, learning it didn’t take too long.

However, the main difficulty was in trying to link the GUI to our Bluetooth code. What made this step harder was the lack of time to do it. Not much documentation was provided about how to link your code with the Glade. The code took a lot of our time while trying to figure out what changes have to be made to apply the code to our application. It didn’t make a lot of sense to start doing the GUI if the main code was not working yet; therefore, we postponed working on the GUI. However, time was running fast and we had to start working on the GUI. By the time we downloaded and installed Glade, and learned how to use it, little time was left to learn how to link the GUI to our source code.

### 5.5 Testing

First the Bluetooth enabled driver was tested after it was downloaded and installed correctly. The way this was planned to take place is by setting up a session between the PCs, one being the server and the other being the client. If this session connects successfully then the driver is functioning correctly. Second test was conducted on the application that will be written in C as an interface between the driver and the development board. This test will took place by sending simple text files from server to the client back and forth. Lastly, running it with the application interface, and making use all the buttons on the window tested the Graphical User Interface (GUI). The way we made sure that there are no bugs in the product was by running it several times with different files.

### 6. LOGISTICS

#### 6.1 Timeline Schedule

<table>
<thead>
<tr>
<th>Week</th>
<th>Projected</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>9/4 –</td>
<td>• Formal project proposal write-up and presentation slides&lt;br&gt;• Research on Bluetooth Technology&lt;br&gt;• Web searching for Bluetooth enabled PCI cards</td>
<td>• Formal project proposal write-up and presentation&lt;br&gt;• Research on Bluetooth Technology&lt;br&gt;• Web searching for Bluetooth enabled PCI cards</td>
</tr>
<tr>
<td>9/11 –</td>
<td>• Research on Bluetooth Technology&lt;br&gt;• Formal project presentation&lt;br&gt;• Call Brainboxes&lt;br&gt;• In class status reports</td>
<td>• Research on Bluetooth Technology&lt;br&gt;• Call Brainboxes&lt;br&gt;• In class status reports</td>
</tr>
<tr>
<td>9/18 –</td>
<td>• Research on Bluetooth Technology&lt;br&gt;• Install PCI cards&lt;br&gt;• In class status reports</td>
<td>• Research on Bluetooth Technology&lt;br&gt;• Web searching for an alternative of Bluetooth enabled PCI cards</td>
</tr>
<tr>
<td>9/4 –</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### 6.2 Project Breakdown

The components and tasks of the project were broken down as follows:

#### 6.2.1 Researching on Bluetooth Technology

This was a very time consuming task. None of the group members were familiar with this fairly new technology. We researched a lot on the technology itself along with its applications, structure, and availability.
6.2.2 Search for Bluetooth hardware and software
The Bluetooth enabled PCI card took us a while to find. We found this hardware on the brainboxes site. The URL is http://www.brainboxes.com/. Upon making a phone call it was found that the PCI card wasn’t available till December. This unavailability made it hard for us to work with Bluetooth technology. Nevertheless after several days of searching the Internet, the developer board was found which supports the RF technology and will help in developing Bluetooth simulations. This board was found on the following site: http://developer.axis.com/hardware/devboard/. This task was not an easy job.

6.2.3 Setting up the environment
This task was fairly easy. We installed Red Hat version 6.0 on two PCs. We also installed the elinux (Linux for embedded systems) on the developer board.

6.2.4 Download and Install the Bluetooth software (driver code)
Downloading and installing software from Internet does not seem to be a challenging task, and it wasn’t. The download of the Bluetooth driver was fairly easy, and so was the installation. The instructions were provided on the same site as the software, which made our job fairly easy.

6.2.5 Running the Bluetooth software (driver code)
It took a lot of efforts to run this code. First, the PPP daemon was missing which had to be installed. The kernel then had to be rebuilt in order for the PPP daemon to work correctly. The first version we downloaded had some complications in running; but as mentioned earlier, when the newer version was downloaded things got much easier. The errors we got with the old version were debugged in the new version, which saved us the trouble of debugging the provided source code.

6.2.6 Modify the Bluetooth software (driver code)
This task was the most challenging of them all. The driver code was modified at several places. Since we chose the option to run the driver with userstack, we had to write a script to put the selected files through the driver code and then write it to the serial ports. The same thing on the other hand, take the selected file off the serial port and put it through the userstack of the driver code.

6.2.7 Boot the developer boards
This task was difficult since the directions were not explicit about how to boot the boards with elinux. The developer boards came with instructions on how to connect them, boot them up, assign IP addresses, and download on them.
6.2.8 Developing GUI

This task was fairly medium when it comes to the difficulty of a task. Glade was downloaded and installed. We found some good tutorials on the internet. The appendices section has the details. The hard part about this task was to actually get it to run with the driver source code. We had to use threads to accomplish this task. None of the group members knew how to use threads. We all researched and asked the Teacher Assistant for assistance.

6.2.9 Testing

Once everything was accomplished, testing was planned. We planned to send files from one computer to another with different text files.

6.3 Order of Completion

See the above section. The order of completion is in the order listed above.

6.4 Milestones

There were many milestones because Bluetooth technology was an unfamiliar area for all of us. We had to struggle through every step of our project. From the basic concept of understand the structure of Bluetooth stack to the modifying the Bluetooth software, we had to do lots of reading and searching of solutions. However, the major milestones (which have already been mentioned in some for above) are:

- Searching for Bluetooth enabled hardware
- Understanding the Bluetooth software’s source code
- Figuring out how to run the Bluetooth driver code without any errors
- booting the developer board and downloading the elinux and Bluetooth software on it
- Compiling the libraries
- Getting the userstack to work correctly

6.5 Individual Group Member Contribution

Ballard Johnson

- Studied the Bluetooth software’s source code
- Downloaded and installed Bluetooth software on one hard drive
- Developed GUI
- Downloaded and compiled all software for developer boards

Chalon Clemens

- Researched the Bluetooth stack (structure)
- Studied the Bluetooth software’s source code
- Modified the Bluetooth’s software source code
- Worked on the proposal presentation slides
- Write up of the critical review report
- Worked with the developer boards
- Downloaded and compiled all software for developer boards
• Tested bluetooth software

Mohammed Braiwish
• Studied the Bluetooth software’s source code
• Developed the GUI
• Worked on the proposal and critical review presentation slides

Naima Nazir
• Researched Bluetooth Technology intensively
• Studied the software’s source code
• Set up the environment on one hard drive
• Report and Presentation Write-Ups for the proposal and the final report
• Worked on the presentation slides for the critical review and the final report

7. MARKETING

In this section, we present the economic plan of this project. Keeping in mind that this is only an estimate, since we are not advertising this project in an actual business entity. If we were to be an actual corporation, all numbers for salary and other costs incurred would be at industry standard.

7.1 Research and Development

Research and development costs consist of salary to each employee on an hourly basis, for the development, testing, and implementation of software. We expect that as the product is developed further when adding further functionalities, the costs incurred will increase at the standard industry rate, which will include the wages paid to each employee for the number of hours worked, bonus, quarterly/yearly salary increase, and increased stock options.

<table>
<thead>
<tr>
<th>Group Member</th>
<th>Average # hours/week</th>
<th>Total # of hours for 15 weeks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ballard Johnson</td>
<td>10</td>
<td>150</td>
</tr>
<tr>
<td>Chalon Clemens</td>
<td>10</td>
<td>150</td>
</tr>
<tr>
<td>Mohammed Braiwish</td>
<td>10</td>
<td>150</td>
</tr>
<tr>
<td>Naima Nazir</td>
<td>10</td>
<td>150</td>
</tr>
</tbody>
</table>

Cost for each employee = $25 per Hour

Total cost = $25 * 600 hours for project = $15,000

7.2 Marketing and Sales

Our Marketing and Sales Division will incur costs primarily from sales employees (with commission), marketing employees, and public relations employees. Our ‘GEE-
WOW!!” factor is extremely high. Bluetooth is a revolutionary technology for the way we do telecommunication these days. This of course will make the job of the Marketing and Sales Division much easier, since our product is in demand.

7.3 Support
Support will be handled in two ways. The first and best way will be a discussion group, where users will be able to submit questions that will be answered by other developers that are utilizing our product. Documentation will also be supplied with the code that gives our email addresses so that people with problems will be able to contact us, the primary source for the code, directly with questions.

7.4 Competitors and Related Risks
The main competitors that we face in this market are other special interest groups that are developing their own applications that utilize the Bluetooth stack. There are a few major companies that are developing products for Bluetooth. Ericsson is currently leading the revolution by developing and releasing functional modules. IBM released a product on 7 November 2000; called BlueDrekar, that is a tool to help Linux developers create applications for Bluetooth. It currently retails at 10,000 licenses for $1000. The market for this type of application is wide open. It is forecasted to be in every kind of electronic device, from cell phones to PDAs. The vision for the Bluetooth is for it to be in every electronic device so that any electronic device can communicate with any other electronic device. With the amount of electronic devices in the world and the amount of Linux users in America, estimated at 30 million, we envision our product as being the educational forefather of a massive amount of new applications that will take advantage of the Bluetooth protocol stack. Below is the brochure that Motorola is planning to put in the market, once they have their Bluetooth devices developed completely.
7.5 Revenue

The application we developed was primarily a research project. Its purpose was to expose the developers to the functions of the bluetooth stack in the Linux environment. Since Linux and the programs that were used in development of the application are licensed under the GNU public domain, they cannot be resold, therefore leaving no way to profit from the application itself.
7.6 Economic Analysis

<table>
<thead>
<tr>
<th>Part</th>
<th>Quantity</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computer</td>
<td>2</td>
<td>Supplied</td>
</tr>
<tr>
<td>Developer Boards</td>
<td>2</td>
<td>$235</td>
</tr>
<tr>
<td>Null modem Cable</td>
<td>3</td>
<td>$30</td>
</tr>
<tr>
<td>Driver</td>
<td>2</td>
<td>Free</td>
</tr>
<tr>
<td>GUI</td>
<td>1</td>
<td>Free</td>
</tr>
<tr>
<td>Testing</td>
<td></td>
<td>Free</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>$1030</td>
</tr>
</tbody>
</table>

8. CONCLUSION

The goal of this project was to implement Bluetooth technology, which was to communicate between two personal computers wirelessly. However due to the lack of the availability of Bluetooth enabled hardware, we had to use null cable modem to connect the two personal computers. The radio frequency (RF) technology of Bluetooth was unfortunately not used. In future when Bluetooth hardware is offered in the market, the radio frequency technology can be implemented within this project. The Bluetooth software that was downloaded from the internet will be enhanced more in future, probably will more features embedded in it. This enhancement can help drastically in improving the functionality of this project.

9. DOCUMENTATION AND SOURCES

- [http://www.bluetooth.com](http://www.bluetooth.com)
- [http://www.infototh.com](http://www.infototh.com)
- [http://motorola.com/bluetooth](http://motorola.com/bluetooth)
- [http://brainboxes.com](http://brainboxes.com)
- [http://developer.axis.com](http://developer.axis.com)
- [http://cambridgesiliconradio.com](http://cambridgesiliconradio.com)
- [http://galde.pn.org](http://galde.pn.org)
- [http://www.redhat.com/support/errata/RHEA1999059-01html](http://www.redhat.com/support/errata/RHEA1999059-01html)
10. APPENDICES

10.1 User Manual for the Bluetooth software

Bluetooth Driver for Linux Kernel v2.0 and v2.2

These files are work in progress. For more information on Bluetooth on Linux see http://developer.axis.com where you can find the newest versions of the software.

What has happened lately

Oct 30 2000

Finally, a new release!

There has been quite a few changes and it is hard to include everything in this text. However, we will try to make a quick introduction of the new functionality.

*** Kernel Stack ***

- We now support the use of several tty:s (currently 7, ttyBT0-ttyBT6) and the control of the stack has been moved to a dedicated tty called ttyBTC(minor 7)
- Improved interface between user mode tty and kernel.
- More ioctls for HCI commands
- Improved SDP which now uses a user mode database for queries.
- Added functionality for choosing different HW (init stuff)
- Added scheduling of cmd queueing for USB
- Lots of bug fixes

*** Comment ***

As of today we have no client functionality in SDP due to changes in the design. However, as server Serial/LAN/DialUP profile is supported. This will be fixed soon.

*** User Mode Applications ***

- New SDP server database application using XML files.
  See /apps/sdp_server/README for details.
- Command line history
- Now pppd works together with the user mode stack (using pty:s)
- Using syslog for most debug (see /var/log/messages)

Please let us know if there are any unclear issues or errors in this README. Comments are appreciated and if there are someone that feels an
urge of making it more complete please contact us or send us a text fragment and we will happily include it!

There are still patches left to include but due to heavy workload we have not had time to add them all. However, in the future if a patch is sent to us reasonably fast (within 2 weeks) after a new release we should be able to merge it in the stack.

Finally, we would also like to thank all contributors on the stack for comments and patches etc. Once we get our open CVS up and running this stack should be very useful for any bluetooth developer!!

Keep up the good work!

Best regards
Bluetooth Team,
Axis Communications

How to build and install the AXIS Bluetooth Stack

# Unpack the archive:
tar xvfz <name of archive>

# Create the bluetooth stack module and all applications (both for use with the kernel version of the stack and the user mode version):
cd bluetooth
source init_env
make

# Install the applications in /usr/local/bin and the SDP configuration file in /etc (must be done as root):
make install

# If the BT device nodes have not been created previously then do (as root):
makes dev

# Insert the module into the kernel (must be done as root):
insmod src/bt.o

# To get debug messages in kernel start another window and issue:
tail -f /var/log/messages
(or whatever file is set in /etc/syslog.conf)

=======================================================================
How to build and install the AXIS Bluetooth Stack
=======================================================================

=
BTD APPLICATION
---------------

Both short and long options may be used whatever suits best.

syntax: ./btd [options]

options:

-u, --physdev <uart device>
sets which uart device that will be used by the stack
default: ttyS2

-b, --btdev <bluetooth device>
sets which bluetooth device that will be used from application
default: ttyBT0

-s, --speed <speed>
sets uart speed
9600, 19200, 38400, 57600, 115200, 230400, 460800
default: 115200 baud

"-r server", --server
sets application role to server

"-r client", --client
sets application role to client)
default: server

-i, --hwinit <HW initialisation>
hw specific initialization
ericsson, digi (DigiAnswer PC card), csr, usb, none
default: ericsson

-m, --cmdmode
enters command line mode
default: skip command line mode

-n, --local-name
prefix used for the local bluetooth device name
default: nothing

-d, --local <local ip>
Sets local ipadress in pppd options

-D, --remote <remote ip>
Sets remote ipaddress in pppd options

-R, --reset
reset bluetooth hardware before use
default: no reset

-e, --modem
Use modem emulation (used when emulate modem in windows dialup). Can also be done from command line mode.
default: on
-S, --unixsock use local unix socket instead of phys port
Used together with hci emulation to test stack locally
(user mode stack only)

-T, --tcpsock <ipaddr:port> use tcp socket instead of
phys port. Used together with hci emulation to test
stack over any TCP/IP based network
Server listens on <:port>, client tries to connect to
<ipaddr:port>. (user mode stack only)

e.g if using ericsson module at 460800 baud and acting as server
./btd --speed 460800 --hwinit ericsson

if using HW that req no init and acting client with command line
interface
./btd -r client --hwinit none

if using non-default devices
./btd --physdev /dev/ttyS2 --btdev /dev/ttyBT3 [options...]

if using the stack in user mode over a local UNIX socket.
Do the following in separate windows
Server: ./btd --cmdmode --unixsock
Client: ./btd --cmdmode --unixsock --client

if using the stack in user mode over a TCP socket.
Do the following on separate computers connected to the
same network
Server: ./btd --cmdmode --tcpsock <:port>
Client: ./btd --cmdmode --tcssocket <ipaddr:port> --client

What do the menu options do?
-----------------------------

con <xx:xx:xx:xx:xx:xx> <line> <srv channel> <profile>
Connect to BT device with BD_ADDR <xx:xx:xx:xx:xx:xx> on line
<line> with RFCOMM server channel <srv channel> <profile (not used)>

Profile is one of:
0 - Serial profile
1 - Lan profile

/* FIXME -- currently there are no client functionality in SDP due to
adoptions to the new SDP server using XML. Thus we do not able to
connect profiles as client. */

disc <line>

Disconnect the connection with line <line>,

send <nbr bytes> <nbr repeats>

Just send chunks of data. <nbr bytes> are sent <nbr repeats>
times. Data is sent using the RFCOMM protocol layer

me <1/0>

Turn modem emulation on/off. Modem emulation is used on the server side to fool a Windows client that it is talking to a modem. The emulation answers OK to all AT\r\n sequences. When it receives ATD\r\n it quits and starts ppp instead.


Set the BT device BD_ADDR to <xx:xx:xx:xx:xx:xx>
(Currently Ericsson specific)

readbd

Get the BT device BD_ADDR

reset

Reset the BT device.

ppp

Quit btd and start ppp to the peer instead.

quit

Just quit.

=======================================================================

How to setup a session between two units
----------------------------------------

Server side:

1. btd --reset --speed 115200 -i ericsson

Client side:

1. btd --reset --speed 115200 -i ericsson -r client
2. Connect to the other unit by doing connect <bd address> 0

=======================================================================

How to use the stack in user mode instead of in the kernel
----------------------------------------------------------

1) Build the applications according to the instructions above.

1) Start the SDP server as 'sdp_user /etc/sdp.xml &'.

3) Then run btduser (as root) with the same options as with the standard btd application (see above).
How to setup a session without any bluetooth hardware

To make the stack work without any hardware HCI_EMULATION must be
switched on in the file btconfig.h and the stack must be recompiled.
You will see a note in the debug messages when starting the stack if
hci emulation really is used or not.

 Note

The HCI emulation as of today simply converts some hci commands to the
corresponding events and simply forwards all acl/sco data.
If you get messages from the stack that something fails when initiating
the stack it is because those hci commands are not supported in the hci
emulator. Anyone that wants are welcome to improve it and send us a
patch. For example, there could be network delays/ packet losses
included and different kinds of hardware can be simulated.

There are three ways depending on how you want to test the stack.
The first one works both when running the stack in kernel and when
running it in user mode. The other two (2 and 3 below) works only in
user mode.

See above for details on command options

1) Using a serial cable (null modem)

Connect 2 Linux PC with a null modem cable, and start the btd
application
as server as one side and btd as client on the other side.

Server side:
1. start btd with the option
   btd --physdev <phys device> --speed <speed> --cmdmode
      --local <local ip> --remote <remote ip>

Client side:
1. start btd with the option
   btd --physdev <phys device> --speed <speed> --client --hwinit none

2. When the menu shows and server also are setup and initialized type:
   (BD address is ignored)
   If connection was successful you will now receive a handle

2) Using a local unix socket on one computer

Server side:
1. btd --cmdmode --hwinit none

Client side:
1. btd --client --hwinit none

2. When the menu shows and server also are setup and initialized type:
   (BD address is ignored)
   If connection was successful you will now receive a handle

3) Using a network tcp socket between two computers
   if using the stack in user mode over a TCP socket.
   Do the following on separate computers connected to a network

Server side:
1. btd --tcpsock <:port>

Client side:
1. Client: ./btd --tcpsocket <ipaddr:port> --client

2. When the menu shows and server also are setup and initialized type:
   (BD address is ignored)

=======================================================================
It does not work, what am I doing wrong?
-------------------------------------------------------------

FIXME -- add FAQ

For debug when running in kernel look in /var/log/messages,
/var/log/debug or
whatever file is specified in /etc/syslog.conf

If you think you have found a bug please switch on all debug in
btdebug.c
(or switch on the define BTD_GLUE_DEBUG in btd.c) and send the logs to
bluetooth-dev@axis.com together with a description of the problem.

=======================================================================

10.2 User Manual for Developer Board

This guide came with the developer board package, and is shown below.

Getting Started Guide to AXIS Developer Board

Serial Number/Ethernet Address

The ethernet address of the developer board is the same as its serial number. The serial number is found on the label on the back of the board.

Setting the IP Address Temporarily

The developer board has a default IP address (192.36.253.80) that most likely needs to be changed to make the board work on your local network. The board does not support automatic IP setting (e.g. DHCP) per default. You need to acquire a unique IP address (ask your network administrator). For the initial setting of the IP address the board needs to be connected to the same network segment as your client.

Use any of the following ways to set the IP address within two minutes after booting the developer board (press the reset button to reboot if you are too late):

ARP and ping from Windows or MS-DOS:

1. Start a DOS prompt window
2. Type the following:
   `arp -s <IP address> <Ethernet address>
   ping <IP address>

Example:
   `arp -s 123.45.67.89 00:40:6c:12:34:56
   ping 123.45.67.89`

ARP and ping from UNIX or GNU/Linux:

1. Start a shell
2. Type the following as superuser (root):
   `arp -s <IP address> <Ethernet address>
   ping <IP address>

Example:
   `arp -s 123.45.67.89 00:40:6c:12:34:56
   ping 123.45.67.89`

This sets the IP address temporarily. You must now put the IP address and subnet mask in a file on the developer board to make it permanent, otherwise the default IP-address (192.36.253.80) will be set again when the board is rebooted.
Making the IP Address Permanent

Once the IP address have been set temporarily it can be set permanently using HTTP, telnet or FTP.

HTTP

1. Start your web browser of choice.
2. Open the URL http://<IP address>/ Example: http://123.45.67.89/.
3. Choose Edit network settings and edit the variables according to your local network. Then click on the Save file button.

telnet

1. Telnet to the IP address of the developer board. Example: Start a command tool (MS-DOS prompt, UNIX shell etc) and type telnet <IP address> (e.g. telnet 123.45.67.89).
2. Edit the file /etc/network/network.conf using the editor /bin/easyedit. Example: easyedit /etc/network/network.conf
3. Change the IP address (and subnet mask).
4. Save the file and exit easyedit by pressing <ESC> a a.
5. Close the telnet connection by typing exit.

FTP

1. Start your FTP client of choice.
2. Open a connection to the IP address of the developer board.
3. Log in as user root with password pass.
4. Download the file /etc/network/network.conf
5. Change the IP address (and subnet mask) in the file.
6. Upload the file to the developer board.
7. Close the connection to the board.

Test your configuration by rebooting the developer board and pinging it. Example: Press and release the reset button on the developer board, wait for approximately 20 seconds (until the developer board is up and running) and type ping <IP Address> (e.g. ping 123.45.67.89).

Development Environment


Subscribe to the ETTRAX developer mailing list by sending an e-mail to majordomo@axis.com with subscribe dev-etrax in the message body.


192.168.2.2
192.168.2.3
10.3 User Manual for GUI

Glade Overview:

This is an overview of how Glade is used to build user interfaces. It includes screenshots of the main Glade windows and also of a couple of applications built with Glade.

This is the main Glade window. It lists all of the windows and dialogs making up your project. Double-clicking on an item in the list shows the corresponding window or dialog.

Menu and toolbar commands enable you to create new projects, load and save them, and build the source code.

The second most important window in Glade is the widget palette. It shows icons representing all of the widgets which are available.

To add new windows and dialogs to your project, simply select the icon in the palette.

To add widgets to a window or dialog, select the widget in the palette, then click on the position you want to add it.

The last major Glade window is the property editor. This allows you to alter the properties of widgets, such as the widget size or the label text.

The signals page also allows you to add signal handlers to widgets, so you can, for example, specify the function to be called when a button is clicked.
If you add a menu bar to your application, you can use this dialog to add the menus and set accelerator keys and signal handlers.

With a bit of practice, it is possible to create complex menu structures in a few minutes.

Building an Interface

Creating interfaces with GTK+ is slightly different to other widget sets. Instead of using fixed positions and sizes for every widget, GTK+ uses containers such as horizontal and vertical boxes and tables to arrange the widgets. This makes it much easier to support resizing of windows, and also helps when interfaces are translated into different languages. However, it does make it slightly more difficult to create the interface.

The window on the left shows the first step in creating a simple text editor application. The window has been divided vertically into four rows, using a GtkVBox widget. A GtkToolbar widget with three items has been added to the second row.

Now we've added a GtkMenuBar and GtkStatusbar to the top and bottom rows, and a GtkText widget in the 3rd row for editing the text. We've also added three buttons to the toolbar for creating, opening, saving files.

That's it! The complete window can be created in a few minutes, including a menu bar with accelerator keys.

The complete application can be found in the examples directory of the Glade distribution.

Create your glade file

Design your GUI up to your favor. Tables are recommend, they look great. For numeric input use spinbuttons, it doesn't cost you more and the user might use the extra functionality.
For textual input you might prefer option menus or combo boxes. You should also take a look at the fancy SearchCombo widget of the gtk--addons I put to cvs.gnome.org. Beware, stock glade does not support it (but gtk--addons come with a patch at hand).

! You will not see the C++ code options in glade unless you select C++ as your language. (Project options)

I recommend putting all your widgets on the heap (use_heap=true, -p option for glade--). Though current glade doesn't default to this behaviour. You might take a text editor and replace each '{@widget>}' by '{@widget}<cxx_use_heap>True</cxx_use_heap>'. Loading and saving this file in glade will reindent it.

Then think about partitioning complex windows into substructures. Toplevel windows always come in an own file (and an own class). Signal callback functions are part of the topmost widget's class. Feel free to break this via the 'separate class' or 'separate_file' switch. I recommend up to ten to twenty callbacks in one class. But on the other hand you lose the ability to easily access widgets outside your class.

Then think whether you need to access a widget in your program. If you need to access it inside your scope (class) declare it as protected. If you need to access it outside your scope, mark it as public. Then include the appropriate header file, glade-- has generated a GMM_* macro for easy access:

```c
GMM_WINDOW1->show();
```

Create your initial program files (glade--)

Look into glade's project options (the sheet) and adjust the pathnames. I recommend to put the glade file near to your program and let glade create fresh files in a subdirectory (source directory option in glade). Then copy the files you need into your current directory.

Simply invoke glade-- by pressing the gear's button in glade or by hand:

```
  glade-- <project file>
```

Some options are only available from the command line:

' -s' gives you some programming examples for signals and widget modification
' -p' puts widgets into the heap. [This is important for large projects. The total size of a structure with virtual functions can not exceed 32k on x86 computers. Since gtk-- is full of virtual function calls putting all widgets into one memory chunk is not advisable.]
' -A' disables autoconf/automake. For small projects these are overkill. Also, gnome automake support is not yet available.

Glade-- will generate the following files:

- myprog.cc: This file contains a sample main function. Simple. Extend it up to your choice.
• autogen.sh: Invoking this shell script configures and compiles the program (standard for gnome projects)

• Makefile.am: automake/autoconf really are your friends! Get used to them

• glademm_support.hh, glademm_support.cc: these files contain the widget map and (if needed) functions for radio group lookup.

• window1_glade.hh: This is the GUI classes declaration file. For the class name the widget name's first letter is capitalized and '_Glade' is appended. Each class contains its child-widgets' declarations as members (widget names converted to lowercase). Do not modify this file!

• window1.hh: These are your program's classes. Signal handlers are declared as member functions, this is the place to add your program's variables and functions (besides creating your own classes of course).

• window1_glade.cc: This file contains the constructors and destructors for the GUI part of your program. Do not edit! You might take a look here to see what the code looks like.

• window1.cc: This file contains your constructors, destructors, signal handlers, custom functions etc. Take them out to other files, it would not matter. Though add these files to the Makefile. Good stuff for a constructor is reading of data (e.g. database interaction) or anything else which has to be done before the user starts interacting with your program. Signal handlers react to user's actions. Writing a program in this manner is not very difficult. Try yourself!

  ! Do not forget to delete glade-- V0.2's 'connect_to_method's from your Constructors if you migrate your programs to V0.5. They went into the window1_glade.cc file!

  You changed the glade file

  Since you should not edit the *_glade.cc and *_glade.hh files glade will overwrite them. Similar for the glademm_support.* files. But what about the rest? Glade-- creates new files:

  • myprog.cc_glade: contains the new main function, most likely new toplevel widgets have been added, insert the variable declaration into your main file.

  • Makefile_glade: New toplevel widgets? Add the corresponding .o files to your Makefile

  • window1.hh_glade: New classes? Add the corresponding class declaration to your header file.

    New signal handlers? Add the handler declaration to the corresponding class. Other changes should not affect this file.
• window1.cc_glade: You divided a class (separate class/separate file)? Move the corresponding Signal callbacks to the right scope! New signal handlers? Add the handler stub and fill it.

Never rename a widget which form a class! (Unless you want to do a lot of search/replace work)
I usually use a
diff window1.hh window1.hh_glade | less
whenever I did huge changes. Did I mention that CVS is a good idea? Use it! Accidental overwrites will not cause you any harm.

The widget map

Glade-- creates a map of public widgets. Access them via the GMM_FOO1 macros. But never access them before they are created or after they have been destroyed.

User types

glade supports custom widgets. Glade-- takes the creation function as include file name (.hh added) and class type. If neccessary you can include and wrap any other class inside the expected header file.

Using glade-- generated widgets within another context (components, user defined widgets)

If you want to use components I propose the following route:

• Create another toplevel window (the name determines the source file name), I usually set visible to off and title to " to remember that this is a dummy window. Name the windows child widget to represent your type! Mark it as 'separate class' and 'separate file'

• Pass it through glade--. You might well delete the window's files.

• Use it like any other custom widget (put your widget's name into the 'creation function' part) (kind of abuse)
Main Definitions

autogen.sh: a script which runs automake, autoconf and the related programs in the correct order, making it very easy to build the application. You pass it any arguments you want to be passed to configure. After running it, you can type 'make' to build the application.

configure.in: standard script passed to autoconf to generate the configure script.

Makefile.am: standard make rules passed to automake to generate the Makefile.in, which the configure script turns into Makefile.

acconfig.h: contains a few macros which are set during the configure script and added to the config.h header (which should be the first file #include'd by all of your source files). Most of these macros are needed for gettext support (ENABLE_NLS, HAVE_CATGETS, HAVE_STPCPY), HAVE-LIBSM is needed for Gnome (but it doesn't hurt a GTK+ app), and a few are added by Glade (PACKAGE_LOCALE_DIR, PACKAGE_DATA_DIR, PACKAGE_SOURCE_DIR).

stamp-h.in: used as a timestamp by automake, for rebuilding some generated files.

AUTHORS: These files are all initially empty, but are created to Change Log comply with the GNU conventions.

src/Makefile.am: Standard automake file.

src/main.c: contains the main() function which will create one of each window/dialog for you.

src/interface.h: declarations of functions you can call to create the windows and dialogs which were built in Glade.

src/interface.c: the code to create the windows and dialogs and all the widgets.

src/callbacks.h: declarations of the signal handler and callback functions which you will write.

src/callbacks.c: the signal handler and callback functions.

src/support.h: declarations of some support functions, including
lookup_widget() which you can use to get pointers to widgets.

src/support.c: the support functions.

Code for the GUI

```c
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif
#include <gnome.h>
#include "callbacks.h"
#include "interface.h"
#include "support.h"

extern GtkWidget *transfer_text_box;
extern GtkWidget *file_entry;

void on_send_button_clicked (GtkWidget *button, gpointer user_data)
{
    gchar *file_name;
    file_name = gtk_entry_get_text(GTK_ENTRY(file_entry));
    gtk_text_insert(GTK_TEXT (transfer_text_box), NULL, &transfer_text_box->style->black, NULL, file_name, -1);
    gtk_text_insert(GTK_TEXT (transfer_text_box), NULL, &transfer_text_box->style->black, NULL, "\n", -1);

    /***********************************************************************
    * INSERT SEND CALL HERE
    * ***********************************************************************/
}

void on_cancel_button_clicked (GtkWidget *button, gpointer user_data)
{
    gtk_entry_set_text(GTK_ENTRY(file_entry), "");
```
void on_enter_clicked (GtkWidget *button, gpointer user_data)
{
}

void on_no_clicked (GtkWidget *button, gpointer user_data)
{
}

void on_connect_clicked (GtkWidget *button, gpointer user_data)
{
}
10.4 Modified Bluetooth Source Code

/*
 * btd.c -- Control application for AXIS Bluetooth stack
 * 
 * Copyright (C) 2000  Axis Communications AB
 * 
 * Mattias Agren <mattias.agren@axis.com>
 * 
 * This program is free software; you can redistribute it and/or
 * modify it under the terms of the GNU General Public License
 * as published by the Free Software Foundation; either version 2
 * of the License, or (at your option) any later version.
 * 
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
 * GNU General Public License for more details.
 * 
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
 * 
 * $Id: btd.c,v 1.23 2000/10/30 17:38:49 mattias Exp $
 * 
 */

/*
 * - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
 */

/* syntax:  ./btd [options] 

options:

  -u, --physdev <uart device>
  sets which uart device that will be used by the stack 
  default: ttyS2

  -b, --btdev <bluetooth device>
  sets which bluetooth device that will be used from application 
  default: ttyBT0

  -d, --local <local ip>
  Sets local ipadress in pppd options

  -D, --remote <remote ip>
Sets remote ipaddress in pppd options

-e, --modem
Use modem emulation (used when emulate modem in windows dialup.
Can also be done from command line mode.
default: on

-i, --hwinit <HW initialisation>
hw specific initialization
ericsson, digi (DigiAnswer PC card), csr, usb, none
default: ericsson

-m, --cmdmode
enters command line mode
default: skip command line mode

-n, --local-name
prefix used for the local bluetooth device name
default: nothing

"-r server", --server
sets application role to server
"-r client", --client
sets application role to client)
default: server

-R, --reset
reset bluetooth hardware before use
default: no reset

-s, --speed <speed>
sets uart speed
9600, 19200, 38400, 57600, 115200, 230400, 460800
default: 115200 baud

-S, --unixsock use local unix socket as phys device
Used together with hci emulation to test stack locally
(usermode stack only)

-T, --tcpsock <ipaddr:port> use tcp socket as phys device.
Used together with hci emulation to test stack over any
TCP/IP based network
Server listens on <:port>, client tries to connect to
<iipaddr:port>. (usermode stack only)
e.g. if using Ericsson module at 460800 baud and acting server
./btd --speed 460800 --hwinit ericsson

if using HW that req no init and acting client with command line interface
./btd -r client --hwinit none --e 0
acting as server with a menu
./btd --hwinit none --e 0 --cmdmode
if using non default devices
./btd --physdev /dev/ttyS2 --btdev /dev/ttyBT3 [options...]

if using the stack in usermode over a local UNIX socket.
Do the following in separate windows
Server : ./btd --cmdmode --unixsock
Client : ./btd --cmdmode --unixsock --client

if using the stack in usermode over a TCP socket.
Do the following on separate computers connected to the same network
Server : ./btd --cmdmode --tcpsock <:port>
Client : ./btd --cmdmode --tcpsocket <ipaddr:port> --client

*/

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

#include <sys/time.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/wait.h>
#include <sys/param.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>
#include <stdio.h>
#include <unistd.h>
#include <termios.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <syslog.h>
#include <errno.h>
#include <stdarg.h>
#include <signal.h>
#include <getopt.h>
#include <dirent.h>
/* The following includes are required to be able to determine the local
* address by examining eth0 */
#include <sys/socket.h>
#include <net/if.h>
#include <netinet/in.h>
#include <sys/un.h>
#include <arpa/inet.h>
#include <readline/readline.h>
#include <readline/history.h>

#ifdef __CRIS__
#include <linux/etraxgpio.h>
#include <asm/svinto.h>
#endif

#define RESTART_ENABLED
#define USE_IPASSIGN

/* automatic restart of application after a hw reset */
#ifdef RESTART_ENABLED
#include <setjmp.h>
#endif

#define BTD_GLUE_DBG 1

#if BTD_GLUE_DBG
#define D_GLUE(fmt, args...) printf(fmt, ## args)
#define PPKT(str, data, len) print_data(str, data, len)
#else
#define D_GLUE(fmt, args...) 
#define PPKT(str, data, len) 
#endif

/* USER SPACE STACK TEST */

/* defined in Makefile */
#ifdef BTD_USERSTACK
#define USE_PSEUDOTERMINALS /* remove ? */
#include <pty.h>
#include <pthread.h>
#include "include/btcommon.h"
#include "include/btmem.h"
#include "include/hci.h"
#include "include/l2cap.h"
#include "include/sdp.h"
#include "include/tcs.h"
#include "include/test.h"
#include "rfcomm.h"
#include "btdebug.h"
#include "unplug_test.h"

/*
 * Local functions
 */

static int init_read_thread(void);
static void hci_receive_thread(void);
int open_pty(void);
static int init_pty_thread(void);
static void pty_receive_thread(void);

int bt_write_lower_driver(unsigned char *data, int len);
int bt_write_top(char *buf, int count);
int bt_receive_top(rfcomm_con *rfcomm, unsigned char *data, int len);
void bt_rfcomm_connection_ready(int status);
void bt_sdp_connection_ready(int status);
int bt_initiated(void);
extern int bt_read_proc(char *buf, int len);
bt_stat_struct bt_stat;
static int bt_initdone = 0;
static int bt_current_hw = HW_DEFAULT;
static int ready_for_ppp = 0;
rfcomm_con *test_rfcomm;

#ifdef USE_PSEUDOTERMINALS

static int pty_master_fd;
static int pty_slave_fd;

/* pty device name */
char  ptydev[256];
int Size=0, fsize, Stat=FALSE;
int file=FALSE;

pthread_t pty_thread;

#endif /* USE_PSEUDOTERMINALS */
/* thread used to read data on "physical" device */
pthread_t read_thread;

/*
 * Socket stuff, used to run two instances of the stack on the same computer
 * against each other
 */

/* local UNIX socket */
int use_local_socket=0; /* default */
#define SOCKET_NAME "/tmp/bt_server"

int use_tcp_socket=0; /* default */
char *tcp_address; /* ip address and port string argument */

/* returns fd to socket */
static int open_socket(char *name, int role);
static int open_tcpsocket(char *addrstr, int role);

#else /* standard mode using kernel stack */

static int sdpsrv_pid = 0;
static void start_sdp_server(void);

/* ioctls defines etc */
#include "btd.h"
static int bt_disc = N_BT;
#endif

/*
 * Local functions common to both userstack and standard btd
 */

static int open_device(char* dev, int flags, int role);
static void close_device(int fd);
static void fd_setup(int fd, int speed, int flow);
static int translate_speed(int spd);
static void init_phys(int fd);
static void init_stack(int spd);
static int bt_set_hw(int hw_type);
static void shutdown_stack(void);
static int connect_profile(struct bt_connection * bt);
static int process_cmd(char *buf, int bt_fd);

static char* get_local_addr(void);
static void start_pppd(void);
static void build_pppdopt(int role);
static void btd_killchilds(void);
static void btd_sighandler(int sig);
static void init_sighandler(void);

/* Misc */
static void show_menu();

/* HW specific */
static void reset_ericsson_hw(void);

/* Modem emulator stuff */

#define CONNECT 1
#define NULLCONNECT 2
#define RESTART 3
#define SHUTDOWN 4

static int parse_modem_data(char *databuf, int len, int bt_fd);
static void modem_emulator(int bt_fd, char *data, int len);
#ifdef PALM_FIX
static int search_str(char *data, const char *searchstr);
#endif

/* Some defines */
#define PPPDCMD "pppd"
#define SDPSRV_CMD "sdp_server"
#define BT_CTRL_TTY "/dev/ttyBTC"

/* default settings */
#define CLIENT 0
#define SERVER 1

#ifdef __CRIS__
#define DEFAULT_PHYS_DEV "/dev/ttyS2"
#define DEFAULT_SPEED "460800"
#else
#endif
#define DEFAULT_PHYS_DEV "/dev/ttyS0"
#define DEFAULT_SPEED "115200"
#endif

#define DEFAULT_BT_DEV "/dev/ttyBT0"
#define DEFAULT_HW_INIT "ericsson"
#define DEFAULT_ROLE SERVER

#define USE_NO_FLOW 0
#define USE_FLOW_CTRL 1
#define START_PPP 1
#define QUIT_BTD 2
#define LOCAL_NAME_LENGTH 32
#define HOST_NAME_LENGTH 100
#define DOMAIN_NAME_LENGTH 100
#define BTD_HISTORY_FILE "/tmp/btd_history"

/*
 * Bluetooth discipline define. Should reside in /include/asm/termios.h
 * However, if compiling this standalone simply use the define in bluetooth.h
 */

static int phys_fd;
static int bt_fd;
static int bt_cfd;
static char* physdev = DEFAULT_PHYS_DEV;
static char* btdev = DEFAULT_BT_DEV;
static char* hw = DEFAULT_HW_INIT;
static char* speedstring = DEFAULT_SPEED;
static int hw_init = 0;
static int modem_emulation = 1;
static int modem_connected = 0;
static int enter_cmd_mode = 0;
static int role = DEFAULT_ROLE;
static int do_reset = 0;
static int start_ppp = 1;
static int pppd_pid = 0;
static char *pppd_options[32];
static char local_name[LOCAL_NAME_LENGTH+1];
static int btd_shuttingdown = 0; /* used to know in modem emulator whether to restart or exit */
#ifdef USE_IPASSIGN

/* IP assign stuff */
enum ipa_requests{
    GETCLIENTIP,
    GETDNSSERVERS,
    RETURNCLIENTIP
};

typedef struct ipa_client
{
    struct in_addr ip;  /* Returns IP address for the client */
    struct in_addr dns[2]; /* Returned DNS servers */
    int nbr_of_dns; /* Returns number of found DNS servers. */
    int proxyarp; /* Returns 1 if proxyarp will be used */
    int useradius;
} ipa_client;

typedef struct ipa_request
{
    unsigned short type; /* type of request */
    unsigned char remote_bd[6]; /* Requestors BD address */
    struct ipa_client client; /* return client */
} ipa_request;

#define SRVSOCKET "/tmp/ipa_server"

int ipa_sock = -1;
static char ipa_buf[128];
struct ipa_client *ipa_send(ipa_request *req);
static void ipa_getpars(void);
static struct ipa_request ipa_req;
static struct ipa_client a_client;

#endif

/* Locals used for IP settings to pppd */
char ip_addresses[35]; /* Max length 255.255.255.255:255.255.255.255 */
char *local_address = NULL;
char *remote_address = NULL;
static char *prim_dns = NULL;
static char *sec_dns = NULL;
static int use_proxyarp = 1;
static int use_radius = 0;
static int use_ipassign = 1;

#ifdef RESTART_ENABLED
jmp_buf jmpbuffer;
#endif

static int option_index = 0;

/* long option list */
static struct option long_options[] =
{
    {"physdev", 1, NULL, 'u'},         /* phys device used from stack */
    {"btdev", 1, NULL, 'b'},           /* bluetooth device */
    {"client", 0, &role, CLIENT},
    {"server", 0, &role, SERVER},
    {"speed", 1, NULL, 's'},           /* uart speed towards hw */
    {"hwinit", 1, NULL, 'i'},          /* hw specific initialization */
    {"local-name", 1, NULL, 'n'},      /* set local bluetooth name */
    {"local", 1, 0, 'd'},              /* set local ip address */
    {"remote", 1, 0, 'D'},             /* set remote ip address */
    {"modem", 1, NULL, 'e'},           /* modem emulation */
    {"reset", 0, &do_reset, 1},        /* reset BT HW */
    {"cmdmode", 0, &enter_cmd_mode, 1},/* enter command line mode */
    {"tcpsock", 1, 0, 'T'},            /* use network socket*/
    {"unixsock", 0, 0, 'S'},           /* use local socket */
    {"radius", 0, &use_radius, 1},     /* enable radius */
    {0, 0, 0, 0}
};

char* menu[] =
{
    "Menu",
    "------------------------",
    "inq (inquiry scan)",
    "con <xx:xx:xx:xx:xx:xx> <line> <server channel> <profile>",
    "disc <handle>",
    "send <file name>", /* currently only using handle 0... */
    "me <1/0>", /* modem emulation */
    "setbd <xx:xx:xx:xx:xx:xx>", /* only ericsson HW */
    "readbd", /* read module bd address */
    "reset", /* reset board */
#ifdef BTD_USERSTACK
    "stat", /* similar to reading the /proc/bt */
#endif
};
"ecs_entertest <handle> (hex)",
"ecs_testctrl <par1,par2,...,par9> (hex)",
"ecs_txtest <par1,par2,...,par11> (hex)",
"ecs_testcon <xx:xx:xx:xx:xx:xx> (hex)",
enable_dut /* enable device under test mode */
"ppp",
"quit",
#ifdef BTD_USERSTACK
    
    --- Test commands ------
    "hotlist bd : 00:01:02:03:04:05",
    "t <testcase>" /* e.g testcase 4.3 't 43 '*/
    "rf_send <bytes> <dlci>",
    "test_conn <xx:xx:xx:xx:xx:xx>" /* l2cap test using PSM 0x1231 */
    "test_disc", /* disconnect only connection registered in test layer */
    "bb_conn <xx:xx:xx:xx:xx:xx>" /* connect only baseband */
    "bb_disc <hci handle>" /* disconnect baseband */
    
#endif

int main(int argc, char **argv)
{
    int opt;
    int spd;

    printf("Bluetooth Control Application\n");
    printf("-----------------------------\n");

    /* now parse options */
    #define OPTIONS_STRING "a:b:d:e:i:mn:Rs:ST:u:X"

    while ((opt = getopt_long(argc, argv, OPTIONS_STRING,
                                 long_options, &option_index)) != -1)
    {
        switch(opt)
        {
            case 'u':
                /* uart device */
physdev = optarg;
break;

case 'b':
    /* bluetooth device */
    btdev = optarg;
    break;

case 'd':
    /* Sets local ipaddress in pppd options */
    local_address = strdup(optarg);
    break;

case 'D':
    /* Sets remote ipaddress in pppd options */
    remote_address = strdup(optarg);
    break;

case 'e':
    /* switches on/off the modem emulation */
    modem_emulation = atoi(optarg);
    break;

case 'i':
    /* HW initialization */
    hw = optarg;
    break;

case 'm':
    /* enter command line mode */
    enter_cmd_mode = 1;
    break;

case 'n':
    strncpy(local_name, optarg, LOCAL_NAME_LENGTH);
    local_name[LOCAL_NAME_LENGTH] = '\0';
    break;

case 'r':
    /* role */
    if (strcmp(optarg, "client") == 0)
        role = CLIENT;
    else if (strncmp(optarg, "server", 6) == 0)
        role = SERVER;
    break;
case 'R':
    /* try to reset the hardware */
    do_reset = 1;
    break;

case 's':
    /* speed */
    speedstring = optarg;
    break;

#ifdef BTD_USERSTACK
    case 'S':
        /* Use local unix socket together with hci emulation */
        use_local_socket = 1;
        /* adjust invalid settings */
        do_reset = 0;
        modem_emulation = 0;
        hw = "none";
        break;

    case 'T':
        /* Use TCP socket together with hci emulation */
        use_tcp_socket = 1;
        tcp_address = optarg; /* ipaddress:port */

        /* adjust invalid settings */
        do_reset = 0;
        hw = "none";
        modem_emulation = 0;
        enter_cmd_mode = 1; /* force */
        break;
#endif

case '?':
    /*      printf("unknown option: %c\n", optopt); */
    break;

}
reset_ericsson_hw();
}
else
{
    printf("ERROR! Do not know how to reset %s hardware.
", hw);
}

#ifdef RESTART_ENABLED
    /* restart point after setting bd address in ericsson hw */
    if (setjmp(jmpbuffer) != 0)
    {
        printf("restart...


");
    }
#endif

if (role == SERVER)
    printf("Running as server\n");
else
{
    printf("Running as client\n");
    enter_cmd_mode = 1; /* force */
}

#ifdef BTD_USERSTACK
    printf("Running stack in user mode\n");

    if (use_local_socket)
    {
        physdev=SOCKET_NAME;
        printf("Using local UNIX socket %s\n", physdev);
    }
    else if (use_tcp_socket)
    {
        physdev=tcp_address;
        printf("Using TCP socket to %s\n", physdev);
    }
    else
    {
        printf("Physdev %s, btdev (not used), speed %s baud\n", 
            physdev, speedstring);
    }
#else
    printf("Physdev %s, btdev %s, speed %s baud\n", 
            physdev, btdev, speedstring);
#endif
#endif

#ifndef USE_IPASSIGN
/* check if there is a server available */
if (!(ipa_sock = open_socket(SRVSOCKET, CLIENT)))
{
    syslog(LOG_INFO, "Could not find IP Assign server\n");
    use_ipassign = 0;
}
else
{
    syslog(LOG_INFO, "IP Assign server OK\n");
    close_device(ipa_sock);
    use_ipassign = 1;
}
#endif

spd = atoi(speedstring);

if ((phys_fd = open_device(physdev, O_RDWR, role)) < 0)
{
    perror("could not open phys dev\n");
    exit(1);
}

init_phys(phys_fd);

#ifndef BTD_USERSTACK
/* Set the current tty to the bluetooth discipline */
if (ioctl(phys_fd, TIOCSETD, &bt_disc) < 0)
{
    perror("Set bt line disc");
    printf("Forgot to insmod bt.o \?\n");
    exit(1);
}

printf("Registered bluetooth line discipline on \%s\n", physdev);
#endif

bt_cfd = open_device(BT_CTRL_TTY, O_RDWR, 0);

init_stack(spd);

/* sleep some to assure that all data was sent before changing baudrate -- FIXME -- */
usleep(10000);
/* Now set phys device speed to whatever HW was set to */
fd_setup(phys_fd, spd, USE_FLOW_CTRL);

tcflush(phys_fd, TCIOFLUSH);

#ifndef BTD_USERSTACK
  start_sdp_server();
#endif

#ifdef BTD_USERSTACK
  open_pty();
  init_pty_thread();
#endif

if (enter_cmd_mode)
{
  /* open btdev if a connection should be made */
  bt_fd = open_device(btdev, O_RDWR, 0);

  read_history(BTD_HISTORY_FILE);

#ifdef BTD_USERSTACK
  sleep(2); /* temp fix - wait for last command response to be returned */
#endif

  printf("Now entering cmd line mode\n");

  show_menu();

  while (1)
  {
    char *line = readline("> ");
    int tmp;
    add_history(line);

    tmp = process_cmd(line,bt_fd);
    if (tmp == START_PPP)
    {
      start_ppp = 1;
      break;
    }
    else if (tmp == QUIT_BTD)
    {
      start_ppp = 0;
      break;
    }
}  
free(line);
}  
write_history(BTD_HISTORY_FILE);

close(bt_fd);

if (!start_ppp) {
  btd_killchilds();

  shutdown_stack();

  close_device(bt_cfd);

  /* now close phys device */
  tcflush(phys_fd, TCIOFLUSH);
  close_device(phys_fd);

  exit(0);
}

while (1) {
  if (modem_emulation) {
    bt_fd = open_device(btdev, O_RDWR, 0);
    syslog(LOG_INFO, "Starting modem_emulator");
    modem_emulator(bt_fd, NULL, 0);

    #ifdef BTD_USERSTACK
      while (!modem_connected)
        sleep(1);
    #endif
    if (btd_shuttingdown)
      exit(0);

    syslog(LOG_INFO, "Modem done.");
    close_device(bt_fd); /* for pppd... */
  }
}

syslog(LOG_INFO, "build_pppdopt");

build_pppdopt(role);
syslog(LOG_INFO, "start pppd");

/* Now start pppd */
start_pppd();

if (modem_emulation)
    modem_connected = 0;

/* pppd done, restart */
syslog(LOG_INFO, "close btdev");
close_device(bt_fd);

    syslog(LOG_INFO, "ppp child died, now restart!


};
}

#endif
static void start_sdp_server()
{
    char *args[] = { SDPSRV_CMD, "/etc/sdp.xml", "/proc/sdp_srv", NULL};

    syslog(LOG_INFO, "Starting SDP server


); if (!(sdpsrv_pid = vfork()))
    { 
        execvp(SDPSRV_CMD, args);

        fprintf(stderr, "%s: no such file or directory\n", SDPSRV_CMD);

        _exit(0);
    }
}
#endif

static void start_pppd()
{

    #ifdef BTD_USERSTACK
    ready_for_ppp = 1;
    #endif

    /* run pppd in a child */
    if (!(pppd_pid = vfork()))
    {
        /* modem emulation resides in stack for now */
if (role == SERVER)
    syslog(LOG_INFO, "Starting ppp server on %s", btdev);
else
    syslog(LOG_INFO, "Starting ppp client on %s", btdev);

execvp(PPPDCMD, pppd_options);

fprintf(stderr, "%s: no such file or directory\n", PPPDCMD);
_exit(0);
}

/* in parent, we let pppd run in the background */
syslog(LOG_INFO, "Spawned pppd[%d] in the background", pppd_pid);

/* Parent, wait for ppp child to die */
wait4(pppd_pid, NULL, 0, NULL);

#ifdef USE_IPASSIGN
if (use_ipassign)
{
    syslog(LOG_INFO, "Returning IP address to IPA");

    /* Return IP address! */
    ipa_req.type = RETURNCLIENTIP;
    //ipa_req.remote_bd
    memcpy(&ipa_req.client, &a_client, sizeof(struct ipa_client));

    if (!ipa_send(&ipa_req))
        syslog(LOG_ERR, "Failed to return IP address to IPA\n");
}
#endif

#ifdef BTD_USERSTACK
    ready_for_ppp = 0;
#endif

}

/* FIXME -- use separate options file 'pppd file <optionfile>' for most common options */

void build_pppdopt(int role)
{
   int i = 0;
#ifdef USE_IPASSIGN
    if (use_ipassign)
        ipa_getpars();
#endif

/* we have no local ip address, get it from eth0 */
if (!local_address)
    local_address = get_local_addr();

/* we have no remote ip address, skip it ! */
if (!remote_address)
    remote_address = strdup(""");

/* create pppd option */
sprintf(ip_addresses,"%s:%s",local_address, remote_address);

/* deallocate strings */
if (local_address)
    free(local_address);
if (remote_address)
    free(remote_address);

local_address = NULL;
remote_address = NULL;
syslog(LOG_INFO, "IP addresses used : %s",ip_addresses);

/* general options */
pppd_options[i++] = (char*)PPPDCMD;

#ifndef USE_PSEUDOTERMINALS
    pppd_options[i++] = btdev;
#else
    pppd_options[i++] = ptydev;
#endif

    pppd_options[i++] = speedstring;
    pppd_options[i++] = "crtscs";
    pppd_options[i++] = "debug";
    pppd_options[i++] = "local";
    pppd_options[i++] = "nodetach";

    if (role==SERVER)
        {
            pppd_options[i++] = "nopersist";
        }
pppd_options[i++] = "silent";
pppd_options[i++] = "passive";

if (use_radius)
{
    pppd_options[i++] = "useradius";
    pppd_options[i++] = "auth";
    pppd_options[i++] = "login";
} else
{
    pppd_options[i++] = "noauth";
}

if (use_proxyarp)
    pppd_options[i++] = "proxyarp";

if (ip_addresses)
    pppd_options[i++] = ip_addresses;

if (prim_dns)
{
    pppd_options[i++] = "ms-dns";
    pppd_options[i++] = prim_dns;
}

if (sec_dns)
{
    pppd_options[i++] = "ms-dns";
    pppd_options[i++] = sec_dns;
}

} else
{
    pppd_options[i++] = strdup("defaultroute");
    pppd_options[i++] = ip_addresses;
}

pppd_options[i] = NULL;

#if 0
/* print pppd_options */
i = 0;
while (pppd_options[i])
    syslog(LOG_INFO, "%s\n", pppd_options[i++]);
#endif

int
translate_speed(int spd)
{
    int speed_c = 0;

    switch(spd)
    {
    case 9600:
        speed_c = B9600;
        break;
    case 19200:
        speed_c = B19200;
        break;
    case 38400:
        speed_c = B38400;
        break;
    case 57600:
        speed_c = B57600;
        break;
    case 115200:
        speed_c = B115200;
        break;
    case 230400:
        speed_c = B230400;
        break;
    case 460800:
        speed_c = B460800;
        break;
    default:
        printf("Bad baudrate %d\n", spd);
        break;
    }
    return speed_c;
}

void
fd_setup(int fd, int speed, int flow)
{
    struct termios t;

#ifndef BTD_USERSTACK
    if (use_local_socket || use_tcp_socket)
return;
#endif

if (fd < 0)
{
    perror("fd_setup");
    exit(1);
}

if (tcgetattr(fd, &t) < 0)
{
    perror("tcgetattr");
    exit(1);
}

cfmakeraw(&t);

t.c_cflag &= ~CBAUD;
    t.c_cflag |= translate_speed(speed) | CS8 | CLOCAL;
    t.c_oflag = 0; /* turn off output processing */
    t.c_lflag = 0; /* no local modes */

if (USE_FLOW_CTRL)
    t.c_cflag |= CRTSCTS;
else
    t.c_cflag &= ~CRTSCTS;

if (tcsetattr(fd, TCSANOW, &t) < 0)
{
    perror("fd_setup : tcsetattr");
    exit(1);
}
return;
}

void
show_menu()
{
    char** option;

    option = menu;
    while (*option)
    {
        printf("%s\n", *option);
        option++;
    }
void
reset_ericsson_hw()
{
#ifdef __CRIS__
/* reset ericsson HW */
    int bitmap;
    int devfd;

    printf("Resetting HW board...
");
    if ((devfd = open_device("/dev/gpioa", O_RDWR, 0)) >= 0)
    {
        /* toggle pin 7 in general port A (reset pin)*/
        bitmap = 1 << 7;
        ioctl(devfd, _IO(ETRAXGPIO_IOCTYPE, IO_CLRBITS), bitmap);
        usleep(1000);
        ioctl(devfd, _IO(ETRAXGPIO_IOCTYPE, IO_SETBITS), bitmap);
        sleep(2);

        close_device(devfd);

        printf("Done.
")
    }
    else
    {
        printf("ERROR! Failed to open gpio port (%d)\n", devfd);
    }
#else
    printf("Please reset HW board within 5 seconds\n");
    sleep(5);
#endif
}

int connect_profile(struct bt_connection * bt)
{
    int con_hdl=-1;

#ifndef BTD_USERSTACK
    if ((con_hdl = ioctl(bt_fd, BTCONNECT, bt)) < 0)
        printf("Connection failed\n");
    else
        printf("Stack returned handle %d\n", con_hdl);
#else
#endif
}
/* User stack */

serport_profile_info p_ser;
int line = 0; /* temp setting */
p_ser.rfcomm_serv_chan = -1;

/* FIXME -- Only one tty supported un usermode, use multiple pty:s */

switch (bt->profile)
{
    case SERPORT_PROFILE:
    {

        /* currently disabled due to new design of SDP, will be fixed ASAP */
        #if 0
            int tmp_hdl;
            int i = 0;

            tmp_hdl = sdp_connect_req(bt->bd, bt->profile, (void*) &p_ser);

        #endif

        printf("connecting SDP");
        /* temp fix - works for now */
        while ((p_ser.rfcomm_serv_chan < 0) && (i < 300))
        {
            usleep(100000); /* 10th of secs */
            i++;
            printf(".");
            fflush(stdout);
        }
        printf("\n");

        if (i==10)
        {
            printf("Failed to get a server channel, exiting...\n");
            return -1;
        }

        printf("got RFCOMM server channel : %d from SDP\n",
                p_ser.rfcomm_serv_chan);
    #endif

        con_hdl = rfcomm_connect_req(bt->bd, 1, line);

    /*FIXME - use semaphores and wait here until RFCOMM is up */
//if (tmp_hdl!=0)
//sdp_disconnect_req(tmp_hdl);
}

break;

case LAN_PROFILE:
    /* will be added shortly after testing */
    break;

default:
    printf("Profile:%d not supported\n",bt->profile);
    break;
}
#endif

return con_hdl;
}

ствие FIXME -- clean this UP !!! */

int
process_cmd(char *buf, int bt_fd)
{
    bt_connection btcon;
    int bd[6];
    int tmp[11];
    int repeat;
    int i, line;
    struct stat Info;
    /* off_t size; */
    int openfile;
    char path[MAXPATHLEN];

#ifdef BTD_USERSTACK
    char tmp_bd[6]; /* used for byte swapping */
    int server_channel, profile, dlci;
#endif

if (!strncmp(buf, "quit", 4))
{
    return QUIT_BTD;
}
else if (!strncmp(buf, "ppp", 3))
if (sscanf(buf, "con %x:%x:%x:%x:%x:%x %d %d %d",
    &bd[0], &bd[1], &bd[2], &bd[3], &bd[4], &bd[5], &btcon.line,
    &btcon.server_chn, &btcon.profile) == 9)
{
    for (i = 0; i < 6; i++)
    {
        btcon.bd[i] = (unsigned char)bd[i];
    }

    /* For now SDP is not involved when connecting due to adaptions to
       new SDP design. Will be fixed as soon as possible */

    printf("Connecting to bd: %02X:%02X:%02X:%02X:%02X:%02X\n",
        btcon.bd[0], btcon.bd[1], btcon.bd[2],
        btcon.bd[3], btcon.bd[4], btcon.bd[5]);
    printf("using profile: %d\n", btcon.profile);

    connect_profile(&btcon);
}
else if (sscanf(buf, "disc %d", &line) == 1)
{
    #ifndef BTD_USERSTACK
    if (ioctl(bt_fd, BTDISCONNECT, &line) < 0)
    {
        perror("Disconnect");
        exit(1);
    }
    #else
    rfcomm_disconnect_req(line);
    #endif
    printf("Disconnected!\n");
} else if (strcmp(buf, "inq", 3) == 0)
{
    #ifdef BTD_USERSTACK
    char inq_lap[6];

    printf("Inquiring...\n");

    inq_lap[0] = 0x33;
    inq_lap[1] = 0x8b;
inq_lap[2] = 0x9e;

hci_inquiry(inq_lap, 10, 100);
#endif
inquiry_results *inq_res;

inq_res = (inquiry_results*) malloc(sizeof(inquiry_results) + 10 * 6);
inq_res->nbr_of_units = 10;

if (ioctl(bt_cfd, BTINQUIRY, inq_res) < 0)
{
    perror("Inquiry");
    exit(1);
}
for (i = 0; i < inq_res->nbr_of_units; i++)
{
    printf("BD %d: %02x:%02x:%02x:%02x:%02x:%02x
", i,
    inq_res->bd_addr[0+6*i],inq_res->bd_addr[1+6*i],
    inq_res->bd_addr[2+6*i],inq_res->bd_addr[3+6*i],
    inq_res->bd_addr[4+6*i],inq_res->bd_addr[5+6*i]);
}
#endif

//Modified Code

else if (sscanf(buf, "send %s ", path) == 1) //
{
#define MAXSIZE (4096*2)
    struct timeval start_t, stop_t;
    int bytes_tot;

    int avg_speed = 0; /* bps */
    unsigned int ms;

#if 0
    int next_print = 0;
    int div = 4;
    int bits;
#endif
    int k=0,n=0;
    char tmp[MAXSIZE], new[MAXPATHLEN+9] ;
char size[56];

if (i > MAXSIZE)
{
    i = MAXSIZE;
}
if( lstat(path, &Info) < 0)//
    perror(path); //
else
{
    if(S_ISREG(Info.st_mode))
    {
        if((openfile=open(path, O_RDWR)) < 0)
"File: %s could not be opened!n",path);
else
        {
            strcpy(new, ".......");
            strcat(new, " ");
            strcat(new, path);
            strcat(new, " ");
            printf("New: %s\n", new);
            printf( size, "%d", (int) Info.st_size);
            strcat(new, size);
            k=strlen(new);
            printf("New: %s\n", new);
            ifndef BTD_USERSTACK
                n=write(bt_fd, new,strlen(new) );
            #else
                n=bt_write_top(new, strlen(new));
            #endif
            while((i=read(openfile, tmp, MAXSIZE)) > 0)
        {
            bytes_tot+=i;
            printf("Now sending %d-bytes packet (%d kB)n", 
            i, bytes_tot/1000);
            ifndef BTD_USERSTACK
                n=write(bt_fd, tmp, i);
            #else
                n=bt_write_top(tmp, i);
            #endif
            if (n < 0)
            {
                printf("couldn't write any more data...\n\n");
                bytes_tot+=i;
                break;
            }
        }
    }
}


```c
#endif
printf("%6d kB left to send... \",
(((int)Info.st_size-=i)/1000));
fflush(stdout);

//End of Modification

gettimeofday(&stop_t, NULL);
printf("ndone.\n");

ms = (stop_t.tv_sec-start_t.tv_sec)*1000 +
(stop_t.tv_usec-start_t.tv_usec)/1000;
if (ms)
avg_speed = ((8*bytes_tot)/ms);

printf("Average TX rate : %d kbps (%d kB/s)\n", avg_speed,
avg_speed/8);
```

```c
strcpy(new, " ");
k=0;
#else if (sscanf(buf, "me %d", &i) == 1)
{
/* switches on/off the modem emulation */
modem_emulation = i;
}
else if (sscanf(buf, "setbd %x:%x:%x:%x:%x:%x",
&bd[0], &bd[1], &bd[2], &bd[3], &bd[4], &bd[5]) == 6)
{
unsigned char new_bd[6];

if (hw_init != HW_ERICSSON)
{
printf("setbd currently only works for Ericsson HW\n");
return 0;
}

for (i = 0; i < 6; i++)
{
```
new_bd[i] = (unsigned char)bd[i];
}

#ifndef BTD_USERSTACK
    if (ioctl(bt_cfd, BTWRITEERICSSONBDADDR, new_bd) < 0)
    {
        perror("Set bd addr");
    }
#else
    set_ericsson_bd_addr(new_bd);
#endif

printf("New Bluetooth device address set to: 
%02X:%02X:%02X:%02X:%02X:%02X\n", 
    new_bd[0], new_bd[1], new_bd[2], 
    new_bd[3], new_bd[4], new_bd[5]);

#endif
#endif

#ifdef RESTART_ENABLED
    printf("resetting hw to activate bd change\n");
    close_device(bt_fd);
    close_device(phys_fd);
    reset_ericsson_hw();
    longjmp(jmpbuffer, 1);
#else
    printf("reset HW to activate bd change\n");
#endif

} else if (strncmp(buf, "readbd", 6) == 0)
{
    unsigned char cur_bd[6];

#endif
#endif

    read_bd_addr(cur_bd);
#endif

    printf("Current bd addr : %02X:%02X:%02X:%02X:%02X:%02X\n", 
        cur_bd[0], cur_bd[1], cur_bd[2], 
        cur_bd[3], cur_bd[4], cur_bd[5]);
else if (strncmp(buf, "reset", 5) == 0)
{
    reset_ericsson_hw();
    close_device(bt_fd);
    close_device(phys_fd);
#ifdef RESTART_ENABLED
    /* must set baud rate to default for hw again... */
    longjmp(jmpbuffer, 1);
#endif
}
else if (sscanf(buf, "ecs_testctrl %x,%x,%x,%x,%x,%x,%x,%x,%x",
               &tmp[0], &tmp[1], &tmp[2],
               &tmp[3], &tmp[4], &tmp[5],
               &tmp[6], &tmp[7], &tmp[8]) == 9)
{
    unsigned char ecs_test_msg[11];
    unsigned short hdl = (unsigned short)tmp[0];
    int i;

    if (hw_init != HW_ERICSSON)
    {
        printf("Only supported for ericsson HW\n");
        return 0;
    }

    /* Connection handle */
    ecs_test_msg[0] = (unsigned char)(hdl&0xff);
    ecs_test_msg[1] = (unsigned char)((hdl >> 8)&0xff);

    /* cast ints to to uchars */
    for (i = 2; i < 9; i++)
    {
        ecs_test_msg[i] = (unsigned char)tmp[i-1];
    }

    /* length */
    ecs_test_msg[9] = (unsigned char)(tmp[8] & 0xff);
    ecs_test_msg[10] = (unsigned char)((tmp[8] >> 8)&0xff);

    printf("*** Sending Ericsson_Test_Control with params : ***\n");

    printf("Connection handle : 0x%x\n", hdl);
    printf("Test_Scenario : 0x%x\n", ecs_test_msg[2]);
    printf("Hopping_Mode : 0x%x\n", ecs_test_msg[3]);
printf("TX_Frequency : 0x%x\n", ecs_test_msg[4]);
printf("RX_Frequency : 0x%x\n", ecs_test_msg[5]);
printf("Power_Control_Mode : 0x%x\n", ecs_test_msg[6]);
printf("Poll_Period : 0x%x\n", ecs_test_msg[7]);
printf("Test_Packet_Type : 0x%x\n", ecs_test_msg[8]);
printf("Length_Of_Test_Data : 0x%x\n", tmp[8]);

#ifndef BTD_USERSTACK
    if (ioctl(bt_cfd, BTERICSSONTESTCONTROL, ecs_test_msg) < 0)
    {
        perror("ecs_testctrl");
    }
#else
    hci_ericsson_test_control(hdl, ecs_test_msg + 2);
#endif
#else
    if (sscanf(buf, "ecs_entertest %x", &tmp[0]) == 1)
    {
        unsigned char ecs_test_msg[2];

        ecs_test_msg[0]= (unsigned char)(tmp[7] & 0xff);
        ecs_test_msg[1]= (unsigned char)((tmp[7] >> 8)&0xff);

        if (hw_init != HW_ERICSSON)
        {
            printf("Only supported for ericsson HW\n");
            return 0;
        }

    printf("*** Sending enter test mode (handle : 0x%x)***\n",
            tmp[0]);
#endif

cout("\n");
#else
    if (ioctl(bt_cfd, BTERICSSONENTERTESTMODE, ecs_test_msg) < 0)
    {
        perror("ecs_entertest");
    }
#else
    hci_ericsson_enter_test_mode((unsigned short)tmp[0]);
#endif

else if (sscanf(buf, "ecs_testcon %x:%x:%x:%x:%x:%x",
                &bd[5], &bd[4], &bd[3], &bd[2], &bd[1], &bd[0]) == 6)
{
    int hdl;
for (i = 0; i < 6; i++)
{
    btcon.bd[i] = (unsigned char)bd[i];
}

printf("*** Connecting baseband ***\n");

#ifndef BTD_USERSTACK
    if ((hdl = ioctl(bt_cfd, BTTESTCONNECTREQ, btcon.bd)) < 0)
    {
        perror("ecs_testcon");
    }
#else
    hdl = hci_test_connect_req(btcon.bd);
#endif

printf("Returned handle : %d\n", hdl);

} else if (strncmp(buf, "enable_dut", 10) == 0)
{
    if (hw_init != HWERICSSON)
    {
        printf("Only supported for ericsson HW\n");
        return 0;
    }

    printf("Enable device under test mode\n");
#endif

#ifndef BTD_USERSTACK
    if (ioctl(bt_cfd, BTERICSSONENABLEDUT) < 0)
    {
        perror("enable_dut");
    }
#else
    hci_enable_dut();
#endif

printf("done.\n");
}
else if (sscanf(buf, "ecs_tctest %x,%x,%x,%x,%x,%x,%x,%x,%x,%x",
    &tmp[0], &tmp[1], &tmp[2],
    &tmp[3], &tmp[4], &tmp[5],
    &tmp[6], &tmp[7], &tmp[8],
&tmp[9], &tmp[10]) == 11)
{
    unsigned char ecs_test_msg[12];
    int i;

    if (hw_init != HW_ERICSSON)
    {
        printf("Only supported for ericsson HW\n");
        return 0;
    }

    /* cast ints to to uchars */
    for (i = 0; i < 10; i++)
    {
        ecs_test_msg[i] = (unsigned char)tmp[i];
    }

    /* length of test data */
    ecs_test_msg[10] = (unsigned char)(tmp[10] & 0xff);
    ecs_test_msg[11] = (unsigned char)((tmp[10] >> 8)&0xff);

    printf("*** Sending Ericsson_TX_Test ***\n");

    printf("RX_On_Start   : 0x%x\n", ecs_test_msg[0]);
    printf("Synt_On_Start : 0x%x\n", ecs_test_msg[1]);
    printf("TX_On_Start   : 0x%x\n", ecs_test_msg[2]);
    printf("Phd_Off_Start : 0x%x\n", ecs_test_msg[3]);
    printf("Test_Scenario : 0x%x\n", ecs_test_msg[4]);
    printf("Hopping_Mode : 0x%x\n", ecs_test_msg[5]);
    printf("TX_Frequency : 0x%x\n", ecs_test_msg[6]);
    printf("RX_Frequency : 0x%x\n", ecs_test_msg[7]);
    printf("TX_Test_Interval : 0x%x\n", ecs_test_msg[8]);
    printf("Test_Packet_Type : 0x%x\n", ecs_test_msg[9]);
    printf("Length_Of_Test_Data : 0x%x\n", tmp[10]);

#ifndef BTD_USERSTACK
    if (ioctl(bt_cfd, BTERICSSONTXTEST, ecs_test_msg) < 0)
    {
        perror("ecs_txtest");
    }
#else
    hci_ericsson_tx_test(ecs_test_msg);
#endif
    printf("TX test starting.\n");
}
/*
 * Non usermode stack functions for the stuff
 * below will be added later on
 */

#ifdef BTD_USERSTACK
  else if (strncmp(buf, "stat", 4) == 0)
  {
    #if 0
      char tmp[4096];
      int len;
      len = bt_read_proc(tmp, 0);
      tmp[len] = 0;
      printf("%s", tmp);
    #endif
    }
  else if (sscanf(buf, "ping %x:%x:%x:%x:%x:%x",
                 &bd[0], &bd[1], &bd[2], &bd[3], &bd[4], &bd[5]) == 6)
  {
    for (i = 0; i < 6; i++)
    {
      tmp_bd[i] = (unsigned char)bd[i];
    }
    printf("Pinging bd : %02X:%02X:%02X:%02X:%02X:%02X\n",
            tmp_bd[0], tmp_bd[1], tmp_bd[2],
            tmp_bd[3], tmp_bd[4], tmp_bd[5]);
    l2ca_ping(tmp_bd);
  }
  else if(sscanf(buf, "rf_conn %x:%x:%x:%x:%x:%x %d",
                 &bd[0], &bd[1], &bd[2], &bd[3], &bd[4], &bd[5],
                 &server_channel) == 7)
  {
    for (i = 0; i < 6; i++)
    {
      tmp_bd[i] = (unsigned char)bd[i];
    }
    printf("Connecting RFCOMM to bd :
            %02X:%02X:%02X:%02X:%02X:%02X\n",
            tmp_bd[0], tmp_bd[1], tmp_bd[2],
            tmp_bd[3], tmp_bd[4], tmp_bd[5]);
    printf("RFCOMM server channel is %d\n", server_channel);
  }
#endif
rfcomm_connect_req(tmp_bd, server_channel, 0);
}
else if(sscanf(buf, "rf_send %d %d",&i,&dlci) == 2)
{
    unsigned char tmpbuf[1024];
    int sent;
    int j;

    for (j = 0; j < 1024; j++)
    {
        tmpbuf[j] = (unsigned char) j;
    }
    sent=rfcomm_send_data(tmpbuf, i, 0, (unsigned char) dlci);
    printf("Sent %d bytes on dlci %d\n", sent, dlci);
}
else if (sscanf(buf, "t %d", &i) == 1)
{
    /* unplug test cases, if you don't know what it is, don't use it !!! :) */
    process_test_cmd(i);
}
else if(sscanf(buf, "tcs_conn %x:%x:%x:%x:%x:%x", &bd[0], &bd[1], &bd[2], &bd[3], &bd[4], &bd[5]) == 6)
{
    for (i = 0; i < 6; i++)
    {
        tmp_bd[i] = (unsigned char)bd[i];
    }
    printf("Connecting TCS to bd : %02X:%02X:%02X:%02X:%02X\n",
            tmp_bd[0], tmp_bd[1], tmp_bd[2],
            tmp_bd[3], tmp_bd[4], tmp_bd[5]);
    tcs_connect_req(tmp_bd);
}
else if(sscanf(buf, "sdp_conn %x:%x:%x:%x:%x:%x %d", &bd[0], &bd[1], &bd[2], &bd[3], &bd[4], &bd[5], &profile) ==7)
{
    #if 0
    serport_profile_info ser_nfo;
    for (i = 0; i < 6; i++)
    {
        tmp_bd[i] = (unsigned char)bd[i];
    }
    printf("Connecting SDP to bd : %02X:%02X:%02X:%02X:%02X\n",
            tmp_bd[0], tmp_bd[1], tmp_bd[2],
            tmp_bd[3], tmp_bd[4], tmp_bd[5]);
    #endif
}
tmp_bd[3], tmp_bd[4], tmp_bd[5]);

printf("SDP profile %d", profile);

sdp_connect_req(tmp_bd, profile, &ser_nfo);
#endif

#else if (sscanf(buf, "test_conn %x:%x:%x:%x:%x:%x",
               &bd[0], &bd[1], &bd[2], &bd[3], &bd[4], &bd[5]) == 6)
{
  for (i = 0; i < 6; i++)
  {
    tmp_bd[i] = (unsigned char)bd[i];
  }

  printf("Connecting TEST layer to bd :
          %02X:%02X:%02X:%02X:%02X:%02X
",
         tmp_bd[0], tmp_bd[1], tmp_bd[2], tmp_bd[3], tmp_bd[4], tmp_bd[5]);

  test_connect_req(tmp_bd);
}
else if (sscanf(buf, "test_case_reject %x:%x:%x:%x:%x:%x",
               &bd[0], &bd[1], &bd[2], &bd[3], &bd[4], &bd[5]) == 6)
{
  u16 fake_psm = 0x4561;
  for (i = 0; i < 6; i++)
  {
    tmp_bd[i] = (unsigned char)bd[i];
  }

  printf("test_case_reject, client tries to connect psm %d\n", fake_psm);
  l2ca_connect_req(tmp_bd, fake_psm);
}
else if (strncmp(buf, "test_disc", 9) == 0)
{
  printf("Disconnecting TEST layer\n");
  test_disconnect_req(testcon); /* extern l2cap_con set in test.c */
}
else if (sscanf(buf, "bb_conn %x:%x:%x:%x:%x:%x",
               &bd[5], &bd[4], &bd[3], &bd[2], &bd[1], &bd[0]) == 6)
{
  for (i = 0; i < 6; i++)
  {
    tmp_bd[i] = (unsigned char)bd[i];
  }

  printf("Connecting baseband to bd : %x:%x:%x:%x:%x:%x\n",
         tmp_bd[0], tmp_bd[1], tmp_bd[2], tmp_bd[3], tmp_bd[4], tmp_bd[5]);

  test_connect_req(tmp_bd);
}
void set_local_name(const char *local_name)
{
    unsigned char domainname[DOMAIN_NAME_LENGTH+1];
    
    
    
    
    
    return local_address;
}
unsigned char buf[LOCAL_NAME_LENGTH+HOST_NAME_LENGTH+DOMAIN_NAME_LENGTH+5];
int len = 0;

*buf = '\0';
*domainname = '\0';

if (*local_name)
{
    while (*local_name && len < LOCAL_NAME_LENGTH)
    {
        if (*local_name >= ' ' && *local_name <= 'z')
        {
            buf[len++] = *local_name++;
        }
    }
}

if (len)
{
    buf[len++] = ' '; 
    buf[len++] = '(';
    buf[len] = '\0';
}

gethostname(&buf[len], HOST_NAME_LENGTH);

getdomainname(domainname, DOMAIN_NAME_LENGTH);
if (*domainname)
{
    strcat(buf, "..");
    strcat(buf, domainname);
}

if (len)
{
    strcat(buf, ")");
}

#ifndef BTD_USERSTACK
if (ioctl(bt_cfd, BTSETLOCALNAME, buf) < 0)
{
    perror("BT set local name");
    exit(1);
}
#else
   hci_change_local_name(buf);
#endif

void init_stack(int spd)
{
   printf("Init stack\n");

#ifdef BTD_USERSTACK
   /* Direct function calls instead of ioctl's... */

   init_read_thread();

   /* Initialise all layers in the bluetooth stack */
   DSYS("Initialising Bluetooth Stack\n");

   /* Set BT hardware type in stack */
   bt_set_hw(hw_init);

   hci_init();
   l2cap_init();
   rfcomm_init();
   sdp_init();
   tcs_init();
   test_init();
   btmem_init();
   unplug_test_init();
   bt_stat.bytes_received = 0;
   bt_stat.bytes_sent = 0;

   switch (hw_init)
   {
      case HWERICSSON:

      /*
         wait for Ericsson module to boot if running this app
         directly at startup
      */

      printf("Setting write_scan_enable in Ericsson module\n");
      write_scan_enable(PAGE_SCAN_ENABLE|INQUIRY_SCAN_ENABLE);

      sleep(1); /* wait for HW... */
      printf("Setting baudrate in Ericsson module\n");
      tcflush(phys_fd, TCIOFLUSH);

      }
set_ericsson_baudrate(spd);

/* wait for HW... must be sure we have sent whole message before changing baudrate on serial port but we must change serial baudrate before the result returns if using P9A modules, why are the P9A ericsson modules not sending the result back on same uart speed ?!!!? */
usleep(10000);
break;

case HW_DIGIANSWER_PC:
    printf("Setting baudrate in Digianswer PC card\n");
    set_digi_baudrate(spd);
    break;

case HW_USBMODULE:
    printf("Setting write_scan_enable in USB module\n");
    write_scan_enable(PAGE_SCAN_ENABLE|INQUIRY_SCAN_ENABLE);
    sleep(1); /* wait for HW */
    break;

case HW_CSR:
    printf("Setting write_scan_enable in CSR module\n");
    write_scan_enable(PAGE_SCAN_ENABLE|INQUIRY_SCAN_ENABLE);
    printf("Setting write_pagescan_activity in CSR module\n");
    write_pagescan_activity(0x50, 0x20); /* more reliable connection process */
    sleep(1); /* wait for HW */
    break;

default:
    printf("No hw init done\n");
    break;
}

bt_initdone = 1;

#else /* Kernel mode stack */

/* FIXME -- add ioctls for all hci functions */

bt_set_hw(hw_init);

if (ioctl(bt_cfd, BTINITSTACK)<0)
{

perror("Init stack");
exit(1);
}

set_local_name(local_name);

switch (hw_init)
{
case HW_ERICSSON:
{
    /*
    wait for Ericsson module to boot if running this app
    directly at startup
    */
    unsigned int wrscan = (PAGE_SCAN_ENABLE | INQUIRY_SCAN.Enable);

    sleep(1);
    printf("Setting write_scan_enable in Ericsson module\n");
    if (ioctl(bt_cfd, BTWRITE_SCAN.Enable, &wrscan) < 0)
    {
        perror("BTWRITE_SCAN.Enable");
        exit(1);
    }

    sleep(1); /* wait for HW...
    printf("Setting baudrate in Ericsson module\n");
    if (ioctl(bt_cfd, BTSETERICsson BAUDRATE, &spd) < 0)
    {
        perror("BTSETERICsson BAUDRATE");
        exit(1);
    }
    
    break;

case HW_DIGIANSWER_PC:
    printf("Setting baudrate Digianswer PC card\n");
    if (ioctl(bt_cfd, BTSETDIGIBAUDRATE, &spd) < 0)
    {
        perror("BTSETDIGIBAUDRATE");
        exit(1);
    }
    
    break;

case HW_CSR:
{
unsigned int wrscan = (PAGE_SCAN_ENABLE | INQUIRY_SCAN_ENABLE);
unsigned int tmp[2];
tmp[0]=0x50;
tmp[1]=0x20;
sleep(1);
printf("Setting write_scan_enable in CSR module!n");

if (ioctl(bt_cfd, BTWRITESCANENABLE, &wrscan) < 0)
{
    perror("BTWRITESCANENABLE");
    exit(1);
}
/* improves reliability when doing a connect */
printf("Setting write_pagescan_activity in CSR module!n");
if (ioctl(bt_cfd, BTWRITEPAGESCANACTIVITY, &tmp) < 0)
{
    perror("BTWRITEPAGESCANENABLE");
    exit(1);
}
break;

/* improves reliability when doing a connect */
case HW_USBMODULE:
{
    unsigned int wrscan = (PAGE_SCAN_ENABLE | INQUIRY_SCAN_ENABLE);
sleep(1);
printf("Setting write_scan_enable in USB module!n");

    if (ioctl(bt_cfd, BTWRITESCANENABLE, &wrscan) < 0)
    {
        perror("BTWRITESCANENABLE");
        exit(1);
    }
    break;
    default:
    printf("No hw init done\n");
    break;
}
#endif
}
void shutdown_stack()
{
    syslog(LOG_INFO, "Shutting down bluetooth stack\n");
#ifndef BTD_USERSTACK
    if (bt_cfd && (ioctl(bt_cfd, BTSHUTDOWN) < 0))
    {
        syslog(LOG_ERR,"Shutting down stack (bt_cfd = %d)", bt_cfd);
        exit(1);
    }
#else
    if (bt_initdone)
    {
        rfcomm_close();
        sdp_shutdown();
        tcs_shutdown();
        l2cap_shutdown();
        btmem_shutdown();
        bt_initdone = 0;
    }
#endif
}

int open_device(char* dev, int flags, int role)
{
    int fd;

#ifndef BTD_USERSTACK
    /* if opening bt dev or control dev simply discard and return fake fd */
    if ((strcmp(dev, btdev)==0) || (strcmp(dev, BT_CTRL_TTY)==0) )
        return 0xb055e;

    if (use_local_socket)
        return open_socket(dev, role);
    else if (use_tcp_socket)
        return open_tcpsocket(dev, role);
#endif

    printf("Opening dev %s\n", dev);
    if ((fd = open(dev, flags)) < 0 )
    {
        perror("open_device");
        exit(1);
    }
tcflush(fd, TCIOFLUSH);
    return fd;
}

void close_device(int fd)
{
    printf("close_device\n");

#ifdef BTD_USERSTACK
    /* if fake fd is used, ignore close since there are no open fd */
    if (fd!=0xb055e)
        close(fd);
#else
    close(fd);
#endif

#endif

//define PALM_FIX

#ifdef PALM_FIX
    int
    search_str(char *data, const char *searchstr)
    {
        int cur = 0;
        int len = strlen(data);

        if (len < 0)
            return 0;

        while (cur < (len-strlen(searchstr)))
        {
            if(!strncasecmp(data+cur, searchstr, strlen(searchstr)))
            {
                /* found searchstr */
                //printf("search_str : found %s!\n", (char*)search_str);
                return 1;
            }
            cur++;
        }
        return 0;
    }
#endif

int
parse_modem_data(char *databuf, int len, int bt_fd)
```c
{  
  syslog(LOG_INFO, "parse_modem_data : %d chars", len);
  databuf[len] = 0;
  syslog(LOG_INFO, "data : %s", databuf);
  if (len <= 0)
  {
    if (btd_shuttingdown)
      return SHUTDOWN;
    else
      return RESTART;
  }
  if(!strncasecmp(databuf, "ATD", 3)) /* windows standard modem */
  {
    syslog(LOG_INFO, "got ATD");
    return CONNECT;
  }
  else if(!strncasecmp(databuf, "CLIENT", 6)) /* windows null modem */
  {
    syslog(LOG_INFO, "got CLIENT");
    return NULLCONNECT;
  }
#ifdef PALM_FIX
  else if (search_str(databuf, "DT") /* palm */) 
  {
    syslog(LOG_INFO, "found DT");
    return CONNECT;
  }
#endif
  else
  
    return 0;
}

void modem_emulator(int bt_fd, char *data, int len)
{
  int done = 0;
  /* FIXME -- adjust this to real speed */
  char *connect = "CONNECT 115200\r\n";
  char *ok = "OK\r\n";
  
#ifdef BTD_USERSTACK
  if ((bt_fd==0xb055e) && (len > 0))
  {
    data[len] = 0;
    done = parse_modem_data(data, len, bt_fd);
  }
#endif
}
if (done == CONNECT)
{
#ifdef PALM_FIX
    sleep(2); /* wait for client */
#endif
    syslog(LOG_INFO, "Write : %s", ok);
    bt_write_top(ok, strlen(ok));
    syslog(LOG_INFO, "Write : %s", connect);
    bt_write_top(connect, strlen(connect));
    syslog(LOG_INFO, "Modem connected !!");
    modem_connected = 1;
}
else if (done == NULLCONNECT)
{
    bt_write_top("CLIENTSERVER\r\n", 9);
    syslog(LOG_INFO, "Nullmodem connected !!");
    modem_connected = 1;
}
else if (done == RESTART)
{
    /* If we got a SIGHUP from kernel reopen btdev */
    syslog(LOG_INFO, "Restart modem emulator...");
    close_device(bt_fd);
    bt_fd = open_device(btdev, O_RDWR, 0);
    done = 0;
}
else
{
    syslog(LOG_INFO, "Modem emulator replies %s", ok);
    bt_write_top(ok, strlen(ok));
}
/* return and possibly wait for new modem data
   (is fed from bt_receive_top) */
return;
#else /* BTD_USERSTACK */

    fd_set rfd;
    char databuf[128];

    while (!done)
    {
        FD_ZERO(&rfd);
        FD_SET(bt_fd, &rfd);

select(FD_SETSIZE, &rfd, NULL, NULL, NULL);

if (FD_ISSET(bt_fd, &rfd))
{
    int len = read(bt_fd, &databuf, 128);

databuf[len] = 0;
done = parse_modem_data(databuf, len, bt_fd);

    if (done == CONNECT)
    {
        #ifdef PALM_FIX
            sleep(2); /* wait for client */
        #endif
        syslog(LOG_INFO, "Write : %s", ok);
        write(bt_fd, ok, strlen(ok));
        syslog(LOG_INFO, "Write : %s", connect);
        write(bt_fd, connect, strlen(connect));
        syslog(LOG_INFO, "Modem connected !");
        modem_connected = 1;
    }
    else if (done == NULLCONNECT)
    {
        write(bt_fd, "CLIENTSERVER\r\n", 9);
        syslog(LOG_INFO, "Nullmodem connected !");
        modem_connected = 1;
    }
    else if (done == RESTART)
    {
        /* If we got a SIGHUP from kernel reopen btdev */
        syslog(LOG_INFO, "Restart modem emulator...");
        close_device(bt_fd);
        bt_fd = open_device(btdev, O_RDWR, 0);
        done = 0;
    }
    else if (done == SHUTDOWN)
    {
        syslog(LOG_INFO, "Shutting down modem emulator\n");
        return;
    }
    else
    {
        syslog(LOG_INFO, "Modem emulator replies OK");
        write(bt_fd, ok, strlen(ok));
    }
}
/*
* HW initialization
*/

char *get_hw_name(int hw_type)
{
    switch (hw_type)
    {
    case HW_NOINIT:
        return "HW no init";
        break;

    case HW_ERICSSON:
        return "Ericsson";
        break;

    case HW_DIGIANSWER_PC:
        return "DigiAnswer PC card";
        break;

    case HW_DIGIANSWER_RS232:
        return "DigiAnswer RS232 dongle";
        break;

    case HW_CSR:
        return "CSR";
        break;

    case HW_XIRCOMM:
        return "XIRCOMM";
        break;

    case HW_USBMODULE:
        return "USB";
        break;

    default:
        return "Unknown";
        break;
    }
}
/ * Initializes phys device to whatever HW requires 
*/

void init_phys(int fd)
{
    ifdef BTD_USERSTACK
    if (use_local_socket || use_tcp_socket)
        return;
    endif

    printf("Initiating physical driver\n");

    if (strncmp(hw, "ericsson", 8) == 0)
    {
        printf("Initializing Ericsson HW\n");
        /* Setup HW default phys device speed */
        fd_setup(fd, 57600, USE_FLOW_CTRL);
        hw_init = HW_ERICSSON;
    }
    else if (strncmp(hw, "digi", 4) == 0)
    {
        printf("Initializing DigiAnswer HW\n");
        /* Setup HW default phys device speed */
        fd_setup(fd, 9600, USE_FLOW_CTRL);
        hw_init = HW_DIGIANSWER_PC;
    }
    else if (strncmp(hw, "usb", 3) == 0)
    {
        printf("Initializing USB HW\n");
        hw_init = HW_USBMODULE;
    }
    else if (strncmp(hw, "csr", 3) == 0)
    {
        printf("Initializing CSR HW\n");
        fd_setup(fd, 115200, USE_FLOW_CTRL); /* default speed */
        hw_init = HW_CSR;
    }
    else
    {
        /* No HW initialization */
        printf("No HW initialization\n");
        hw_init = HW_NOINIT;
    }
}
static int bt_set_hw(int hw_type)
{
    syslog(LOG_INFO, "Current HW set to %s\n", get_hw_name(hw_type));

#ifdef BTD_USERSTACK
    if (!bt_initialized())
        bt_current_hw=hw_type;
    else
        D_WARN("Cannot change HW when stack is active\n");
#else
    if (ioctl(bt_cfd, BTSETHARDWARE, &hw_type) < 0)
    {
        perror("Set BT Hardware in stack");
        exit(1);
    }
#endif

    /* Fixme - This should do all HW specific settings */

    switch (hw_type)
    {
    case HW_NOINIT:
        break;
    case HWERICSSON:
        break;
    case HW_DIGIANSWER_PC:
        break;
    case HW_DIGIANSWER_RS232:
        break;
    case HW_CSR:
        break;
    case HW_XIRCOMM:
        break;
    case HW_USBMODULE:
        break;
    default:
        break;
    }
    return 0;
}

static void
btd_killchilds()
{
#ifndef BTD_USERSTACK
    {
        #ifdef BTD_USERSTACK
        if (!bt_initialized())
            bt_current_hw=hw_type;
        else
            D_WARN("Cannot change HW when stack is active\n");
#else
    if (ioctl(bt_cfd, BTSETHARDWARE, &hw_type) < 0)
    {
        perror("Set BT Hardware in stack");
        exit(1);
    }
#endif

    /* Fixme - This should do all HW specific settings */

    switch (hw_type)
    {
    case HW_NOINIT:
        break;
    case HWERICSSON:
        break;
    case HW_DIGIANSWER_PC:
        break;
    case HW_DIGIANSWER_RS232:
        break;
    case HW_CSR:
        break;
    case HW_XIRCOMM:
        break;
    case HW_USBMODULE:
        break;
    default:
        break;
    }
    return 0;
}
if (sdpsrv_pid > 0)
{
    syslog(LOG_INFO, "Killing SDP server\n");
killsdpsrv_pid, SIGTERM);
    if (waitpid(sdpudp_pid, NULL, 0) < 0)
        perror("waitpid sdp server");

    sdpsrv_pid = 0;
}
#endif

if (pppd_pid > 0)
{
    syslog(LOG_INFO, "Killing pppd\n");
killspppd_pid, SIGTERM);
    if (waitpppd_pid, NULL, 0) < 0)
        perror("waitpid pppd");
    ppppd_pid = 0;
}

const char*
signame(int sig)
{
    switch (sig)
    {
    case SIGHUP:
        return "SIGHUP";
        break;

    case SIGINT:
        return "SIGINT";
        break;

    case SIGCHLD:
        return "SIGCHLD";
        break;

    case SIGQUIT:
        return "SIGQUIT";
        break;

    case SIGTERM:
        return "SIGTERM";
        break;
}
default:
    return "Unknown signal";
}
}

static void
btd_sighandler(int sig)
{
    if (sig == SIGCHLD)
        return;
    else if (btd_shuttingdown)
    {
        syslog(LOG_INFO, "Why are we getting more than one signal here \n");
        return;
    }

    btd_shuttingdown = 1;

    btd_killchilds();

    shutdown_stack();

    close_device(bt_cfd);

    /*now close phys device */
tcflush(phys_fd, TIOFLUSH);
close_device(phys_fd);
exit(0);
}

static void
init_sighandler(void)
{
    struct sigaction act;
syslog(LOG_INFO, "Initiating signal handler\n");
act.sa_handler = btd_sighandler;
sigemptyset(&act.sa_mask);
act.sa_flags = 0;
sigaction(SIGHUP, &act, 0);
sigaction(SIGINT, &act, 0);
sigaction(SIGCHLD, &act, 0);
sigaction(SIGQUIT, &act, 0);
sigaction(SIGTERM, &act, 0);
}
#ifdef BTD_USERSTACK

/*
 * USER STACK FUNCTIONS (instead of bluetooth.c)
 */

int init_read_thread()
{
    printf("Initiating read thread\n");
    if (pthread_create(&read_thread, NULL,
                        (void*)hci_receive_thread, NULL)!==0)
        perror("pthread_create");
    sleep(1); /* wait for thread to start */
    return 0;
}

void hci_receive_thread()
{
    fd_set rfd;
    char databuf[4096];

    while (1)
    {
        FD_ZERO(&rfd);
        FD_SET(phys_fd, &rfd);

        select(phys_fd+1, &rfd, NULL, NULL, NULL);

        if (FD_ISSET(phys_fd, &rfd))
        {
            int len = read(phys_fd, &databuf, 4096);

            if (len > 0)
            {
                hci_receive_data(databuf, len);
            }
        }
    }
}

/*
 * Allocate a pty master-slave pair.
 */
int open_pty()
{
    printf("Open pty.\n");
}
return openpty(&pty_master_fd, &pty_slave_fd, ptydev, NULL, NULL);
}

/* Create a thread that reads data from the pty and
   passes it to the rfcomm layer */
int init_pty_thread()
{
    printf("Initiating pty thread\n");
    if (pthread_create(&pty_thread, NULL,
                        (void*)pty_receive_thread, NULL)!=0)
        perror("pthread_create");
    sleep(1); /* wait for thread to start */
    return 0;
}

void pty_receive_thread()
{
    fd_set rfd;
    char databuf[4096];

    while (1)
    {
        FD_ZERO(&rfd);
        FD_SET(pty_master_fd, &rfd);

        select(pty_master_fd+1, &rfd, NULL, NULL, NULL);

        if (FD_ISSET(pty_master_fd, &rfd))
        {
            int len = read(pty_master_fd, &databuf, 4096);

            if (len > 0)
            {
                bt_write_top(databuf, len);
            }
        }
    }
}

int bt_write_lower_driver(unsigned char *data, int len)
{
    int i;
    D_GLUE("bt_write_lower_driver : %d bytes\n", len);
    //PPKT("bt_write_lower_driver ", data, len);
    i = write(phys_fd, data, len);
return i;
}

int
bt_write_top(char *buf, int count)
{
    int retval;
    int line;
    int bytes_sent = 0;
    line = 0;

    D_GLUE("bt_write_top %d bytes on line %d\n", count, line);

    while (bytes_sent!=count)
    {
        retval = rfcomm_send_data(buf+bytes_sent, count-bytes_sent, count-bytes_sent, line, 2);

        if (retval > 0)
            bytes_sent+=retval;
        else if (retval==0)
            usleep(1000); /* wait some ... */
        else
        {
            printf("error\n");
            return retval;
        }

        D_GLUE("sent %d out of %d bytes\n", bytes_sent, count);
        usleep(1000); /* wait some time ... */
    }

    bt_stat.bytes_sent+=bytes_sent;

    return bytes_sent;
}

int bt_receive_top(rfcomm_con *rfcomm, unsigned char *data, int len)
{
    int n;
    int r=0, new_file,i;
    char name[MAXPATHLEN], new_data;
    struct stat Info, R_Info;
    D_GLUE("bt_receive_top : %d bytes ...\n", len);

    }*/
//Modified code
if(file == TRUE)
{
    if(Stat == TRUE)
    {
        if( (new_file=open(name, O_RDWR | O_CREAT ) < 0)
            fprintf(stderr, "File: %s could not be opened!\n", name );
            file=FALSE;
        }
        Stat=FALSE;
    }
    if(Size < fsize)
    {   if( (r=write(new_file, data,len )) < 0)
            fprintf(stderr,"Write is not writing\n");

            Size+=r;
    }
    else
    {    close(new_file);

        printf("File: %s has been received\n", name);
            file=FALSE;
            Stat=FALSE;
            Size=0;
    }
}
if(strncmp(data, ".", 2 ) == 0)
{
    sscanf(data,"....... %s %d", name, &fsize);
    i=strlen(name);
    name[i]=\'\0\';
    file=TRUE;
    Stat=TRUE;
}

//End of modification

 ifdef BTD_USERSTACK
.strict_
ifdef BTD_USERSTACK

    /* FIXME --  why is data echoed back if no application is running on top
       of PTY ?? */

    */
if (!ready_for_ppp)
    return len;
#endif

/* feed this to PTY connected to pppd or whatever application
   that may be running there... */

if ((n = write(pty_master_fd, data, len)) != len)
{
    D_GLUE("bt_receive_top: tried to write %d bytes, got %d\n",
           len, n);
}

bt_stat.bytes_received+=n;

return n;
}
void bt_rfcomm_connection_ready(int status)
{
    printf("rfcomm_create_connection ready.\n");
}

void bt_sdp_connection_ready(int status)
{
    printf("sdp_create_connection ready.\n");
}

int bt_register_rfcomm(rfcomm_con *rfcomm)
{
    test_rfcomm = rfcomm;
    return 0;
}

int bt_initiated()
{
    return bt_initdone;
}

/* TCP socket */
static int open_tcpsocket(char *addrstr, int role)
{
    int server_sockfd;
    struct sockaddr_in server_address;
    int client_sockfd;
    struct sockaddr_in client_address;
int server_len;
int client_len;
int port;
char ipstr[16];
char *pos;
/*parse address string */
pos = strchr(addrstr, ':');
/* copy ip address */
memcpy(ipstr, addrstr, pos-addrstr);
ipstr[pos-addrstr]=0; /* null term */
/* extract port number */
port = atoi(pos+1);
if (role==SERVER)
{
    /* open socket */
    server_sockfd = socket(AF_INET, SOCK_STREAM, 0);
    server_address.sin_family = AF_INET;

    server_address.sin_addr.s_addr = htonl(INADDR_ANY);
    server_address.sin_port = htons(port);

    server_len = sizeof(server_address);
    if (bind(server_sockfd, (struct sockaddr *)&server_address, server_len))
    {
        perror("bind");
        exit(1);
    }

    if (listen(server_sockfd, 5))
    {
        perror("listen");
        exit(1);
    }

    /* request for a new connection */
    client_sockfd = accept(server_sockfd, (struct sockaddr *)&(client_address), &client_len);
return client_sockfd;
}
else
{
/* Client */

printf("Connecting to TCP server socket [%s:%d]\n", ipstr, port);

/* open socket */
client_sockfd=socket(AF_INET, SOCK_STREAM, 0);

/* destination address */
server_address.sin_family=AF_INET;
server_address.sin_addr.s_addr = inet_addr(ipstr);
server_address.sin_port = htons(port); /* for example... */
server_len = sizeof(server_address);

if (connect(client_sockfd,
              (struct sockaddr *)&server_address,
              server_len) < 0) {
    perror("open_tcpsocket");
    exit(1);
}

return client_sockfd;
}
}

#endif /* BTD_USERSTACK */

/* Local UNIX socket stuff */
static int open_socket(char *name, int role)
{
    int server_sockfd;
    struct sockaddr_un server_address;
    int client_sockfd;
    struct sockaddr_un client_address;
    int server_len;
    int client_len;

    syslog(LOG_INFO, "Opening socket %s ", name);
    if (role==SERVER)
    {
        /* remove any old socket */
        unlink(name);
/* open socket */
server_sockfd=socket(AF_UNIX, SOCK_STREAM, 0);
server_address.sun_family=AF_UNIX;
strcpy(server_address.sun_path, name);
server_len = sizeof(server_address);
bind(server_sockfd, (struct sockaddr *)&server_address, server_len);
listen(server_sockfd, 5);

client_sockfd=accept(server_sockfd,
    (struct sockaddr *)&(client_address),
    &client_len);

return client_sockfd;
} else {
    /* Client */

    client_sockfd=socket(AF_UNIX, SOCK_STREAM, 0);

    /* 'destination' socket */
    server_address.sun_family=AF_UNIX;
    strcpy(server_address.sun_path, name);
    server_len = sizeof(server_address);

    if (connect(client_sockfd,
        (struct sockaddr *)&server_address,
        server_len) < 0) {
        syslog(LOG_ERR, "open_socket %s failed", name);
        return 0;
    }

    syslog(LOG_INFO, "Socket connected to %s\n", server_address.sun_path);
    return client_sockfd;
}

#endif USE_IPASSIGN

struct ipa_client* ipa_send(ipa_request *req)
{
    int len;
    struct ipa_client *rsp = NULL;

    printf("Opening socket to IP Assigner\n");

if (!(ipa_sock = open_socket(SRVSOCKET, CLIENT)))
{
    perror("open_socket");
    return NULL;
}

printf("Sending ipa request\n");

write(ipa_sock, req, sizeof(struct ipa_request));

syslog(LOG_INFO, "IP Assigner listening for incoming responses");

len = read(ipa_sock, &ipa_buf, 128);

if(!strncasecmp((char *)&ipa_buf, "OK", 2))
{
    syslog(LOG_INFO, "ipa_send : request succeeded");
    return (struct ipa_client *)ipa_buf; /* FIXME !*/
}
else if(!strncasecmp((char *)&ipa_buf, "ERROR", 5))
{
    syslog(LOG_INFO, "ipa_send : request failed");
    return (struct ipa_client *)NULL;
}
else if (len == sizeof(struct ipa_client))
{
    syslog(LOG_INFO, "ipa_send : request succeeded");
    rsp = (struct ipa_client*)ipa_buf;
}

printf("Closing ipa socket\n");
    close(ipa_sock);
    return rsp;
}

void ipa_getpars()
{
    char *ms_dns = NULL;
    char *ms_dns2 = NULL;
    struct ipa_client *client;

    /*
    * Get options from IPAssigner
    */
printf("Reading IP assign configuration.\n");
ipa_req.type = GETCLIENTIP;
/* memcpy(&ipa_req.remote_bd, remote_bd, 6); */

/* send request */
client = ipa_send(&ipa_req);

if (client == NULL)
{
    syslog(LOG_INFO, "Failed to get client settings");

    /* Use noip if not set from btd option */
    if (!remote_address)
        remote_address = strdup("");

    use_radius = 0;
    use_proxyarp = 1;
}
else
{
    a_client = *client;

    /* Set local values */
    use_proxyarp = a_client.proxyarp;
    use_radius = a_client.useradius;

    if (!remote_address)
        remote_address = strdup(inet_ntoa(a_client.ip));
    else
        syslog(LOG_INFO, "remote_address already set from btd option!");

    syslog(LOG_INFO, "Client IP : %s", remote_address);

    /* Since most clients should be able to receive the DNS configuration the 
     * MS style we send them with ms-dns (if we got any) */

    if(a_client.nbr_of_dns > 0)
    {
        ms_dns = "ms-dns";
        prim_dns = strdup(inet_ntoa(a_client.dns[0]));
        if(a_client.nbr_of_dns > 1)
        {
            ms_dns2 = "ms-dns";
            sec_dns = strdup(inet_ntoa(a_client.dns[1]));
        }
printf("DNS used : prim: %s sec: %s\n", prim_dns?prim_dns:"none", sec_dns?sec_dns:"none");
"
#endif /* USE_IPASSIGN */
10.5 Presentation Slides