ELE 437 - File Sharing Application
Bluetooth Project

Laïla Atrmouh

Tuesday 13 December 2012

---

1 Introduction

The purpose of this homework was to develop a bluetooth application running on Gumstix Boards. I decided to develop a file sharing application : the aim of this application is to facilitate the sending of files between two gumstix boards. My main motivation for this application was to develop a user-friendly interface : I wanted to develop a tool easy to use, so that the user could avoid the use of a terminal.

2 Design and plan

There were three mains milestones during the project
- Installing the Software Development Kit (SDK).
- Designing the graphical user interface
- Developing an engine for efficient and secure file transfer

2.1 Installing the SDK

This step was longer than expected. Beside the installation of the Bluetooth libraries, I also had to install the graphical user interface librairies. I first tried to install the Java's Swing libraries, but a lot of the components I was using were not compatible with the Ubuntu Theme ; for example, I wanted to use a FileChooser but it was not managed by the Ubuntu Theme. That's why I decided to switch to the GTK Library. This library does the same job as Swing, but it is written in C.

There are also some material requirements for the project : I had to plug the Gumstix Boards to a monitor in order to have a desktop where the application could be displayed. I just needed a HDMI cable, I used the monitor as a USB hub where I plugged a keyboard and a mouse. Once these two steps were done, I was ready to develop.

2.2 Designing the GUI

I did a lot of sketches in order to think about the ergonomy and the usability of the application. I wanted the "Send File" button to appear in all the views of the application so that the user doesn't have to click too many times before sending a file. Indeed, it is the main functionality of the application, it had to be easily launchable.
I thought that splitting the view for logs and downloads would be better, so that the information are not mixed. This way, the user can clearly understand what's going on in the application.
I also used a lot of popups for the notifications (new file received, or error happened during the sending of a file) in order to make the user aware of all the actions that happened in the runtime of the application.

2.3 The engine for file transfer

The model of client/server is used for this application but the host plays both roles at the same time. It was possible by creating two synchronized processes : the client is charged in the first process (when the user decides to send a file), the server is charged in the second process (when an external host sends a file to the host).

3 Implementation Details

The "Scanning" functionnality uses the "hcitool scan" command. When a user clicks on the "Scan" button, the application launches the "hcitool scan" command, gathers the result and displays them in a popup. All the reachable and discoverable devices are showed in the application.
Two processes are running in the same time. One is suppose to handle the events triggered by the user (when he decides to send a file, to scan the reachable devices, or to see the downloaded files). We needed two processes in order to make the reception and the emission possible in the same time. The two processes are synchronized using a C function : waitpid. The father

waits for its son to finish (and the son ends when the user closes the window of the application).

```
if ( (childProc = fork()) < 0 ) { perror("fork issue"); }
if ( childProc == 0 ) {
        // Child Process - Running the application
        construct_gui(argc, argv);
}
else {
        // Father process - Listening to received files
        while ( waitpid(childProc, &status, WNOHANG) == 0 ) {
                dest = receive_file();
                if ( dest != "" ) {
                        printf("%s received <<<", dest);
                        notifyViewNewFileReceived(dest);
                }
        }
if ( WIFEXITED(status) )
        printf("childProc exited with exit status %d.\n",
            WEXITSTATUS(status));
else
        perror("ChildProc has not terminated correctly.\n");
}
```

Creation of the two synchronized processes

First Process : Listening
The first process listens the L2CAP socket and notifies the host if a new file is received. When a new file is received, the process reads the content of the file in the L2CAP socket and copies the content in a new file that will be recorded in the receiver's file system.

Second process : Sending
The second process enables the user to send a file whenever he clicks on the "Send File" button. It uses the "sendfile" method, that copies the content of the file in the L2CAP socket.

4 Conclusion

The basic functionalities are working in the application. However, the receiving side of the application could have been improved. Indeed, the user receives files whether he wants or not. A popup saying "Do you want to

receive XX's file ?" would be safer. It was a quite challenging project because I didn't know how to work with gumstix boards or GTK librairies before starting the project but it was a good way to learn. I even corrected some wikis while developing the project. It was really interesting to follow the discussions on the Gumstix boards, the Gumstix community is very active.

5 Sources

http ://github.com/leiluspocus/filestix – Source Code of the application

6 Screenshots of the application

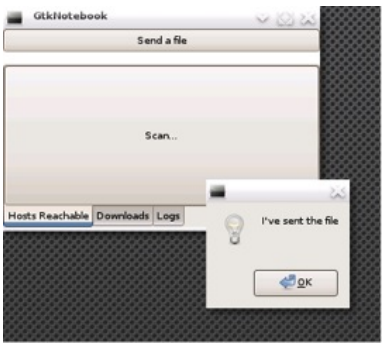

FIGURE 1 – Success Notification after the sending of a file

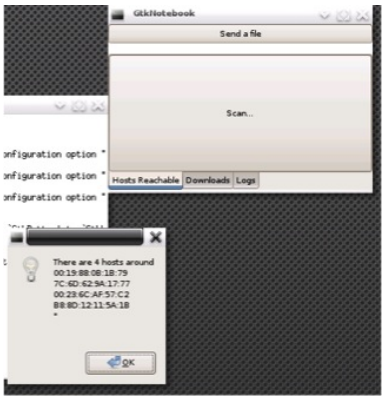

FIGURE 2 – Scanning Devices PopUp

FIGURE 3 – Notification Error - When the user tries to send a file that doesn't exist

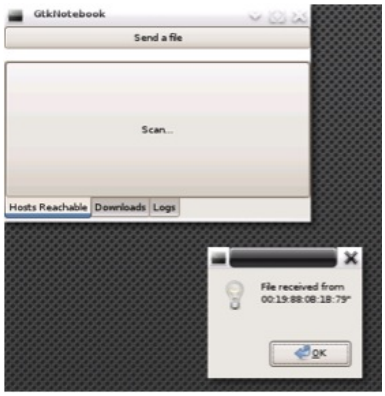

FIGURE 4 – Notification for a file received