Time-Sharing Computer System with Audio Integration

Final Report
April 28, 2010

Sponsored by:

ECE 480
Team 4

Manager: David Wilson
Webmaster: Tim Haynie
Document Prep: Andy Bruinsma
Presentation Prep: Bret Charboneau
Lab Coordinator: Billy Mattingly
Executive Summary

Two schools near Arusha, Tanzania currently have multi-seat computer systems that require upgrades. A third school is also scheduled to be added to this project. Michigan State University, along with support from George and Vickie Rock and the Dow Chemical Company, will continue to support this project in May 2010. The two currently operational systems have six sessions running off of a single computer in order to save on both hardware and power costs. These systems require manual input to associate each screen with the proper keyboard and mouse and can be executed incorrectly by students with limited computer experience. The system also periodically crashes when not all of the sessions are in use. It is the responsibility of Design Team 4 to implement a system that assists in associating a keyboard, mouse and monitor to each user station. Team 4 will also bring the third school online when they travel to Arusha in May 2010. In addition, Team 4 will implement audio capabilities to each station to further the educational opportunities for the students.
Acknowledgement

Design Team 4 would like to thank all of those responsible in making this incredible project possible. First and foremost, a special thanks to George and Vickie Rock and the Dow Chemical company for their generous support. We would also like thank Michigan State University and Dr. Erik Goodman in allowing the team to travel to Tanzania and implement the project.

This project was also successful because of previous design team members Dan Newport and Josh Wong. Their insight into installing and maintaining MDM was invaluable to the team’s success.

Also, the team would like to thank Gregg Mulder and Roxanne Peacock of the ECE department for their support and patience in ensuring the team was helped whenever needed.

Last but not least, the team would like to thank Dr. Mukkamala, Design Team 4’s facilitator, for his weekly insight, support, and dedication to the team’s success.
Table of Contents

Chapter 1 ............................................................................................................................. 4
  Introduction ..................................................................................................................... 4
  Background ................................................................................................................... 5
Chapter 2 ............................................................................................................................. 6
  House of Quality ........................................................................................................... 7
  Design Options ............................................................................................................ 9
  Design Decision .......................................................................................................... 10
  Solution Matrices ....................................................................................................... 11
Chapter 3 ........................................................................................................................... 12
  Multiseat Display Manager ....................................................................................... 12
  Sound Implementation .............................................................................................. 16
  USB Devices ................................................................................................................ 17
  Hardware Solution ..................................................................................................... 18
Chapter 4 ........................................................................................................................... 28
  Final Product ............................................................................................................... 28
  Results ........................................................................................................................... 30
Chapter 5 ........................................................................................................................... 31
  Cost Analysis .............................................................................................................. 31
  Gantt Chart ................................................................................................................... 32
  Conclusion .................................................................................................................... 37
Appendix I ........................................................................................................................ 38
  Technical Roles .......................................................................................................... 38
Appendix II ....................................................................................................................... 45
  References .................................................................................................................... 45
Appendix III ...................................................................................................................... 46
  MDM Configuration Files ............................................................................................ 46
  Microcontroller Source Code ......................................................................................... 68
Chapter 1

Introduction

In Tanzania, the availability of educational resources is very scarce which leaves many students limited access to gain knowledge. The students there do not have access to a library and normal computer systems are too expensive for them to be readily available. However, a low-cost solution to provide students with internet access would help solve the problem at hand.

Two teams have successfully implemented a solution that is still in use at two schools near Arusha, Tanzania. At both schools, the current configuration is a multi-seat computer system that is able to produce six user sessions off of just one computer. This semester, Michigan State University will send another team to Tanzania to setup a third school with a six-user system, as well as provide upgrades to the schools already online. Funding of this project is now being sponsored by George and Vickie Rock, as well as the DOW Chemical Company.

By reviewing progress of previous teams, this semester’s team should strive to revise the current setup in order to add the required upgrades. In order to have a successful design, the team’s goals and criteria are as follows:

- Provide a design that is low-cost relative to commercial solutions. If this design is ever widely implemented, it is important that as many schools as possible can gain the opportunity to have a multi-seat system.
- The system should assist students in associating all keyboards and mice to each individual seat.
• Create a design that is capable of having sound output at each seat which will also be properly associated. This is a requested upgrade to improve learning experience of Tanzanian students.

• Ensure that the design is user-friendly for teachers and students.

• Build a system that is of low maintenance due to the infrequent, on-site support of MSU design teams.

**Background**

In the fall semester of 2008, the Lenovo Corporation made the decision to provide an opportunity for ECE 480 students. The students would be able to design a product for underdeveloped Tanzanian schools. The task is to provide low cost, low power usage, and low maintenance solar powered multi-user computers. This system would be designed to overcome the schools’ lack of electricity and telecommunication systems, the harsh climate, and the students’ lack of computer experience.

In previous semesters, two design teams have dedicated their time to provide the requested computers for the Tanzanian schools. The first team successfully designed a system that is powered by solar energy, using solar panels that charge a battery bank. In addition, the team was able to construct a management system that monitors voltages and currents coming from the solar panels and the battery. Also, the system was designed to monitor the temperature in the battery case. If the temperature exceeded the safe temperature, the system will automatically shut down.

The second team had a similar task as they were to design a multi-user computer system that would be implemented in a second Tanzanian school.
Additionally, the team successfully installed modified antennas on two routers, one at each school. The antennas allow a satellite router to be accessed from one school to another, approximately two miles apart. Originally, the satellite router was installed at the first school where there is not access to the power grid. The second team decided to move the satellite to the secondary school located on the power grid in order to relieve power usage from the solar panels at the first school.

Chapter 2

FAST Diagram

In order to maintain focus on the needs of the customer, Design Team 4 used a FAST diagram to aid them throughout the design process. The principle reason for the project’s design is to educate Tanzanian students by providing internet access. From the diagram (figure 1), one can see exactly what the customer needs are: accommodate user by adding sound, reducing cost, removing system bugs, and displaying seat registration instructions.
Team 4 used The House of Quality Matrix (figure 2) to determine the Customer Critical Requirements (CCRs) and implement the System Design Specifications. This matrix was used in the team’s decision making process and to help reduce design uncertainties. The overall goal for the team was to ensure customer satisfaction and that the voice of customer was being considered throughout the design process. Also, Team 4 used the matrix to compare the design performance to the competitor’s design. In the matrix it is noticeable that Team 4’s design is more cost efficient than the competitor. The House of Quality Matrix shows that the customer’s most important need is to have internet access on a multi-user system and minimize seat registration errors.
Figure 2. House of Quality
Design Options

For the design portion of this project, the team had two main options to consider.

1. Commercial Hardware and Software

   Userful – One possible solution to the customer needs comes mainly in the form of a commercially available product from a company called Userful. This company sells and distributes both the software (Userful Desktop) and hardware (Userful Multiplier) necessary for multi-seat capabilities. The Multiplier product allows each user to connect a USB keyboard, a USB mouse, a pair of speakers or headphones, a microphone, and a USB peripheral to his or her seat through the Multiplier. The main disadvantage of Userful is cost. The price ranges from $69-99 per seat according to their website.

2. Modified Open-Source Software and Custom Hardware

   MDM – The other main option is to use a combination of an actual display manager (Gnome Display Manager or GDM) in conjunction with additional software to provide multiple seats. This additional software is called Multiseat Display Manager (MDM) and was specifically designed to be used in a multi-seat configuration. The main advantage of MDM is that it was designed for multi-seat use and assigning keyboards and mice to each session should be relatively simple. MDM should allow for the expansion of adding sound to each session. To do so, a USB soundcard will be added and configured for each session.
**Hardware** – The hardware that would compliment the open-source MDM software would provide an audio interface for each user as well as reduce association errors of keyboards and mice. It was designed not only to be compatible with the MDM software, but also in any multi-user environment. The hardware should also make the system as a whole more user friendly and is advantageous over a commercial solution because it provides a major reduction in cost per seat. The components that comprise this system would include a hub for each user with an audio interface, as well as one common communication link between all user hubs. This communication hub would contain a microcontroller to manage and direct all user inputs from their respective hubs.

**Design Decision**

With cost and quality being the main two concerns, Design Team 4 agreed with previous teams that using the open-source software of GDM and MDM is the best option. In addition to the previously used software, the team also determined their hardware solution was an acceptable method of correcting USB device association problems. While Userful would be an acceptable choice, Design Team 4 made it a goal to match the effectiveness of their product but doing so at a lower cost. Also, because MDM is the current system in place in Tanzania, integration with the currently installed systems should be relatively seamless. In addition to the current installation, however, the team will need to implement sound as well as correct issues experienced by the current schools.
Solution Matrices

Design Team 4 used the following solution matrices (figures 3 and 4) to aid in design decisions. These matrices helped the team focus on the important criterion throughout the design process.

### Feasibility Matrix

<table>
<thead>
<tr>
<th>Engineering Criteria</th>
<th>Possible Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MDM</td>
</tr>
<tr>
<td>Low Cost</td>
<td>X</td>
</tr>
<tr>
<td>User Friendly</td>
<td>X</td>
</tr>
<tr>
<td>Low Power</td>
<td>X</td>
</tr>
<tr>
<td>Easy Replication</td>
<td></td>
</tr>
<tr>
<td>Low Downtime</td>
<td>X</td>
</tr>
<tr>
<td>Flexibility</td>
<td>X</td>
</tr>
<tr>
<td>Support</td>
<td></td>
</tr>
<tr>
<td>Expandability</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3. Feasibility Matrix

### Selection Matrix

<table>
<thead>
<tr>
<th>Engineering Criteria</th>
<th>Importance</th>
<th>Possible Solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MDM</td>
<td>Useful</td>
</tr>
<tr>
<td>Cost</td>
<td>5</td>
<td>o</td>
</tr>
<tr>
<td>User Friendliness</td>
<td>5</td>
<td>o</td>
</tr>
<tr>
<td>Power</td>
<td>1</td>
<td>o</td>
</tr>
<tr>
<td>Replication</td>
<td>4</td>
<td>o</td>
</tr>
<tr>
<td>Downtime</td>
<td>5</td>
<td>o</td>
</tr>
<tr>
<td>Flexibility</td>
<td>2</td>
<td>Δ</td>
</tr>
<tr>
<td>Support</td>
<td>3</td>
<td>o</td>
</tr>
<tr>
<td>Expandability</td>
<td>3</td>
<td>o</td>
</tr>
<tr>
<td>Total</td>
<td>180</td>
<td>164</td>
</tr>
</tbody>
</table>

**Key:**
- **Strong (9)**
- **Moderate (3)**
- **Weak (1)**

Figure 4. Selection Matrix
Chapter 3
Multiseat Display Manager

Section 1 – Implementation decisions

The first step in setting up MDM is deciding what distribution of Linux will be used. Because MDM is not an actual display manager, rather a wrapper to be used in conjunction with many common distributions, an actual display manager needs to be selected. Online documentation shows that KDM, XDM, and others have been chosen, but the easiest and most common distribution to work with is Gnome Display Manager (GDM). Once users are logged into MDM, they use GDM as normal. It’s the login and association upon boot-up that is different. This is handled by MDM, which creates virtual sessions “inside” of GDM. In this way, multiple instances of GDM are provided, one for each user, monitor, and set of keyboard and mouse.

The next step is to choose the proper video cards for the system being used. While there is a lot of flexibility in this step, the decision also depends on the number of PCI slots available. The computer being used in this project is a Lenovo S20 Workstation and has three PCI express slots available. The three video cards being used are NVidia GeForce 8400GS. Two of the cards have dedicated fans in order to prevent overheating, a problem the team faced with the first cards chosen.

Section 2 – Configuring MDM

Once the display manager and hardware have been installed, the next step is to alter configuration files in order to compliment the system MDM will
be used on. Numerous changes are needed based on the hardware installed and the number of users the system will service. Patches are also needed in order to correct known bugs in MDM.

In order to make effective changes in the MDM configuration files, it is important to become familiar with some of the more important files used by MDM. Unlike a Windows operating system, Linux does not use a registry to alter the configuration setup. Instead, Linux uses a number of configuration files that work collectively to deliver the desired system. Some of the more important configuration files and their purpose are listed below.

- **discover-devices**: used to recognize what hardware devices (keyboards, mice, video cards) are connected to the computer

- **mdm-bin**: manages the creation of the seats needed. This occurs by determining the number of seats in use and calls mdm-start-seat to create each session.

- **mdm-start-seat**: this is the file that prompts each user to press the F1-F6 and click the mouse. Once a keyboard and mouse are connected, a 'lock' is created so that this specific set of keyboard and mouse can only be used in their intended session.

- **mdm-common**: used to start mdm.conf. This file is used by nearly every configuration file.

- **mdm.conf**: the main configuration file that MDM uses. In this file, the type of xephyr sessions can be chosen, as well as some additional options.

- **xephyr-gdm**: this is the most commonly used xephyr setup. Xephyr-gdm creates the virtual or “nested” sessions to allow multiple instances of GDM.

- **xorg.conf.mdm**: when MDM is started, the computer will use xorg.conf.mdm instead of xorg.conf. This file is used to instruct the computer about the physical layout of screens, the types of drivers being used, and many other additional options.

All of these files, as well as many others, are included in the basic installation of MDM found at
http://wiki.c3sl.ufpr.br/multiseat/index.php/Compiling_mdm. While the initial download process is relatively simple, configuring them correctly to work with the hardware installed is quite a challenge.

When a computer is operating in a one-user environment, the user’s session finds a video card and takes control of the entire card’s resources. This means if two monitors are plugged into the video card, the user’s screen would be extended across both monitors. For this implementation, six screens are needed but only three video cards can be used. In order to achieve a six-user setup, Design Team 4 virtualized each screen on each video card. To do so, changes are required in xrandr.info and xorg.conf.mdm.

The first file the team changed was xrandr.info. As shown in figure 5, three screens are setup, but two virtual screens are created inside of each. For xorg.conf.mdm, the screens are associated with their respective video card and monitor. An example of two virtual screens is shown in figure 6.

```
DISPLAY=:0.0
SCREEN_AMOUNT=2
OUTPUT_NAMES=VGA1 VGA2
SCREEN_SIZES="1024x768" "1024x768"
SCREEN_X_ORIGIN=0 1024
-------------- end --------------

DISPLAY=:0.1
SCREEN_AMOUNT=2
OUTPUT_NAMES=VGA1 VGA2
SCREEN_SIZES="1024x768" "1024x768"
SCREEN_X_ORIGIN=0 1024
-------------- end --------------

DISPLAY=:0.2
SCREEN_AMOUNT=2
OUTPUT_NAMES=VGA1 VGA2
SCREEN_SIZES="1024x768" "1024x768"
SCREEN_X_ORIGIN=0 1024
-------------- end --------------
```

Figure 5. xrandr.info file
In addition to the "Screen" field, many other small changes are required in the xorg.conf.mdm. Alterations include the physical layout, keyboard and mouse drivers, video card drivers, and some additional options in the "Server Layout" field. The full xorg.conf.mdm used in this project, as well as all other important configuration files, is located in Appendix III.

After configuring all of the required files for the desired system, one last important change is needed. Design Team 4 has setup the computer to boot to a multi-user environment rather than a normal, one-user login. In this way, the computer should boot directly to six screens waiting for user input to initiate the login sequence. In order to boot directly to MDM, an 'rcconf' command was run and MDM was enabled.

```
Section "Screen"
 Identifier "Screen 1"
 Device       "Video Card 1"
 Monitor      "Monitor 1"
 DefaultDepth 24
 Option "metamodes" "CRT-0: nvidia-auto-select +0+0"
 SubSection "Display"
  Depth 16
  Modes "1024x768"
  Virtual 2048 768
 EndSubSection
 SubSection "Display"
  Depth 24
  Modes "1024x768"
  Virtual 2048 768
 EndSubSection
EndSection
```

Figure 6. xorg.conf.mdm file
Sound Implementation

In order to get sound working on a multi-user system, each USB audio card needs to be defined in the computer’s configuration file. Each audio card has a specific address in which the computer uses to locate its exact position. In order to locate these addresses, the ‘cat /proc/asound/cards’ command must be entered into the terminal. The output should be similar to that of figure 7.

```
user4@ece488:~$ cat /proc/asound/cards
0 [Intel]: HDA-Intel - HDA Intel
  HDA Intel at 0xf0300000 irq 16
1 [default]: USB-Audio - C-Media USB Headphone Set
  C-Media USB Headphone Set at usb-0000:00:1a.0-1.3, full speed
2 [default_1]: USB-Audio - C-Media USB Audio Device
  C-Media USB Audio Device at usb-0000:00:1a.0-2.3, full speed
3 [default_2]: USB-Audio - C-Media USB Headphone Set
  C-Media USB Headphone Set at usb-0000:00:1d.2-1.3, full speed
4 [default_3]: USB-Audio - C-Media USB Headphone Set
  C-Media USB Headphone Set at usb-0000:00:1d.2-2.3, full speed
5 [default_4]: USB-Audio - C-Media USB Headphone Set
  C-Media USB Headphone Set at usb-0000:00:1d.3-1.3, full speed
6 [default_5]: USB-Audio - C-Media USB Audio Device
  C-Media USB Audio Device at usb-0000:00:1d.3-2.3, full speed
```

Figure 7. USB sound devices

Each address should be in a format similar to ‘usb-xxxx:xx:xx.x-x’, which is subject to change depending on whether or not the audio devices are plugged into a hub. Each of these addresses must be entered into the mdm.conf file in order to identify which sound card is assigned to each seat. Figure 8 shows the format of the required information that should be added to the mdm.conf file.

```
SOUND[1]="usb-0000:00:1d.2-1.3"
SOUND[2]="usb-0000:00:1d.2-2.3"
SOUND[3]="usb-0000:00:1a.0-1.3"
SOUND[4]="usb-0000:00:1d.0-2.3"
SOUND[5]="usb-0000:00:1d.3-2.3"
SOUND[6]="usb-0000:00:1d.3-1.3"
```

Figure 8. USB sound card addresses
In addition to editing the mdm.conf file for each user, the computer must be instructed to boot the sound cards at a higher priority than MDM. This is first done by adding the ‘asound_config’ script in Appendix III to the ‘/etc/init.d’ folder and then running a simple command. Running ‘sudo update-rc.d asound_config defaults 15’ will place ‘asound-config’ at priority 15, higher than MDM’s default priority of 20.

Finally, the sound needs to be assigned to each user in the sound preferences inside Ubuntu. Once each user is logged into their MDM session, a test tone is played for each user, one at a time, to determine which sound card is assigned for each user. Once this is completed, the computer is restarted and sound associated properly. Note that sound is being assigned to each user, not to each seat.

**USB Devices**

In addition to Design Team 4’s project description, the team was also able to supply USB flash drive support for the users. This should be useful as teachers will be able to save students’ schoolwork in one useful location. By default, MDM does not allow separate users to access these drives. In order to change these permissions, user settings can be changed by an administrator under ‘Authorizations’ in the ‘Administration’ menu. In the ‘Authorizations’ dialog box, each user must be granted authorization in order for USB drives to accessible for each user session (figure 9).
Hardware Solution

Section 1 – General Overview of System

When the computer is turned on, the Assistive USB Device Registration Interface (AUDRI) system is activated via the power connections from the USB ports on the computer. A personal user hub (figure 10), will sit alongside its respective keyboard and begin flashing its green light. A student will then approach a seat, where they will be prompted on their screen to press the button oriented in the center of the hub.
By pressing this button they are connecting their own keyboard and mouse, while disabling all others who are not yet associated. At this time, all other users will have a solid red light on their hubs indicating that it is not their turn to log into the system. The user that pressed their button first can now associate their devices and will not experience interference from any other user that may accidently click their mouse or keyboard. It is important to note here that the buttons on the other five user hubs will not work until their light turns blinking green again. When they are finished associating, they are then prompted on the screen to press the button again, which in turn will enable the other users to log in. There will now be one hub with no lights blinking (the associated user), and five other hubs with blinking green lights waiting for users to begin. The process then repeats until all users are logged into the system.

**Section 2 – Design and Components**

**User Hub**

The user hub is designed to house a button, red LED, green LED, and two 3.5mm audio input jacks on the front for headphones. There are also three
audio jacks on the back which provide the communication link between the user hub and communication hub (figure 11), which in turn communicates to all other hubs. These three jacks provide an input for audio, an input for LEDs, and an output for a user button. The communication hub will have an output jack for the LEDs and an input for the button which will both be linked by a stereo audio cable.

Figure 11. Communication hub

**Audio Link**

The audio input will come from each respective audio card, and will be connected through a stereo audio cable into the back of the user hub.

**LED Inputs**

Since there are two LEDs that need to be controlled independently, an audio cable was also a convenient source of communication. Stereo left was used as an input for the red LED and stereo right was used as an input for the green LED. Both of the lights are then shorted together in the user hub on their negative side, and soldered to the ground pin on the audio jack. The LEDs then are connected to a 330 ohm resistor which acts as a current limiter to provide
proper light luminosity. This resistor is on the communication hub board (figure 14) and is grounded.

**Switch**

The switch is a SPDT momentary switch, and also uses a stereo audio cable to send a signal to the communication hub. Stereo left is connected to a 5V source on the communication hub and stereo right is connected to an input pin on the microcontroller on the communication hub. It is important here to note that a 10k ohm resistor is connected in parallel to this input port on the communication hub. This is designed to act as a pull-down resistor and is always a good design practice. Essentially this resistor will make the input 0V whenever the 5V switch connection is not present. This will eliminate any stray voltages that may confuse the microcontroller logic. A 10k ohm value was chosen as the resistor in order to allow only a small amount of current to flow, which reduces power consumption.

**Communication Hub**

The communication hub (figures 11 and 14) is the component that provides the actual switching on of the USB devices. For a six-user system there are six keyboards and six mice, bringing the total inputs to 12. There are 12 USB inputs and 12 USB outputs on the hub and this is accomplished via 12 double female USB connectors. Each connector provides an input and an output for each respective device.

To perform the switching of the devices, an analog multiplexer was used for each device. Specifically, it was TI MUX/DEMUX 8X1 16DIP. A 2x1 package
could have also been implemented, but in order to save costs, the design team
decided to use the 8x1 devices due to their availability. An analog multiplexor is
a bi-directional solid state switch that can be opened or closed from its input
pins. Design Team 4 wired an input from the microcontroller to port A on each
of these devices. When port A receives a 5V signal, it makes the connection
between the input and output of the device on the multiplexor itself. The COM

<table>
<thead>
<tr>
<th>INH</th>
<th>C</th>
<th>B</th>
<th>A</th>
<th>ON CHANNEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>L</td>
<td>Y0</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>L</td>
<td>H</td>
<td>Y1</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>Y2</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>Y3</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>L</td>
<td>L</td>
<td>Y4</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>Y5</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>Y6</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>H</td>
<td>H</td>
<td>Y7</td>
</tr>
<tr>
<td>H</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>None</td>
</tr>
</tbody>
</table>

Figure 12. Multiplexer logic
port is connected to the output USB connector, and the Y1 port is connected to the input USB connector (figure 12). Since the system was designed to have each user's mouse and keyboard turn on at the same time, port A on each multiplexer is wired together.

A USB device has four wires that include a 5V red line, grounded black line, and a green and white line that compose data + and data – respectively (figure 13). The communication hub design only connects and disconnects the white data – line, and all other three lines of the input and outputs are shorted together. This provided sufficient functionality with the system.

<table>
<thead>
<tr>
<th>USB Wire Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Red</td>
</tr>
<tr>
<td>Green</td>
</tr>
<tr>
<td>White</td>
</tr>
<tr>
<td>Black</td>
</tr>
</tbody>
</table>

Figure 13. USB Wire Connections
Figure 14. AUDRI system schematic
Section 3 – Microcontroller

The microcontroller used was a Microchip PIC18F4520. Although the low cost and immediate availability of the chip through Michigan State University were contributing factors in the decision to use the 4520, the chip’s features were ultimately the reason the 4520 was chosen. The first feature of the 4520 that made it appealing to the design was the four 8-pin Input/Output ports and one 4-pin I/O port, for a total of availability of 36 inputs or outputs. Each pin on each port of the 4520 can be individually set as an input or an output within the code. The second feature of interest on the 4520 was the internal oscillator on the chip. Having an internal oscillator saved on overall cost since an external clock did not need to be purchased; however an external oscillator could still be used if desired. These settings, as well as many other features available for the chip, can be configured directly in the MPLAB IDE. Finally, a C compiler (preinstalled in the Microchip MPLAB IDE) and a debugger/programmer device (the MPLAB ICD2) was readily available for use with the 4520. The MPLAB IDE and ICD2 were provided in lab by Michigan State University.

One of the original designs solutions called for a different chip, the Microchip PIC16F1826. This chip featured a cost that was roughly one quarter of the cost of the 4520, had an internal oscillator, and had enough programmable inputs and outputs for a single user, so each station would have its own microcontroller. However, implementing this chip into the design had several drawbacks. First, the chip was not immediately available since it could not be provided by Michigan State University and was on back order with at least one vendor. Furthermore, the solution called for an 1826 chip for each user. Since each computer system accommodates six users, the purchase of
these chips would ultimately cost more than a single 4520 chip. Thirdly, the program logic for the 1826 was much more complicated and difficult to follow and debug as a result of requiring the chips to communicate to each other through a communication bus. Lastly, although C compilers for 1826 do exist, one was not readily available for use in lab. These factors led to a decision that a single chip – the 4520 – would be used instead.

**Programming the PIC18F4520**

This section will cover how to connect the 4520 to an in-circuit debugger (ICD) or programming device and C language statements unique to the PIC18 family used in programming.

Three pins are used for connecting the 4520 to an ICD or programming device:

- Pin 1: V¬¬PP, Programming Voltage
- Pin 39: PGC, Programming Clock
- Pin 40: PGD, Programming Data

In addition, pins 11 and 32 should be connected to VDD and pins 12 and 31 should be connected to Ground. It is not possible to program the chip unless these pins (11, 12, 31, and 32) are wired appropriately unless Low Voltage Programming is used and it was not used during the design process. The ICD or programmer may also require connections to VDD and Ground. No other pins are required to be connected to simply program the chip, although it is practical to build a prototype circuit during the debugging phase for testing purposes.
Several unique statements are used when writing software for the 4520 in the C programming language:

- `#include <p18cxxx.h>`: a header file which will include header files for many microcontrollers in the PIC18 family. All other statements discussed here are defined indirectly by including this file.

- `TRISX = 0b00111100`: This statement sets pins the pins of port ‘X’ as inputs or output. Note that ‘X’ should be substituted with an actual port on the 4520 (A, B, C, D, or E). The ‘0b’ prefix indicates that the number which follows will be a binary representation. A 0 in the binary number will set the corresponding pin as an output and a 1 will set it as an input. Here, pins 0, 1, 6, and 7 of Port ‘X’ are being used as outputs, and ports 2 through 5 are set as inputs. This statement could equivalently be written using a decimal number (`TRISX = 60;`) or a hexadecimal number (`TRISX = 0x3C;` where ‘0x’ is used to denote hexadecimal).

- `PORTXbits.RXY`: This statement can be used to either set an output pin to a high or low (1 or 0) value, and can also be used to read the value of an input pin. Note that ‘X’ should be substituted with an actual port and Y should be substituted with an actual pin number. As an example, to toggle output pin RB7, the statement “`PORTBbits.RB7= !PORTBbits.RB7;`” could be used.

**Logic of the Program**

The program follows the following basic pseudo code:

Main function:

- Declare and initialize required variables
- Set ports as inputs or outputs
Initialize output pins

Loop (forever):

   Blink all unregistered green LEDs
   If unregistered user X holds button for long enough, register user X
   End loop

Register User X function:

   Turn all other users’ red LED on
   Blink User X’s green LED
   Turn the output-enable pin associated with that user’s keyboard and mouse high
   Return when user X holds his button again

Chapter 4

Final Product

Once the design, testing, and debugging were concluded, the team needed a way to package their product. The product needed to be secured in a quality enclosure and be able to withstand external conditions such as dust and temperature. Enclosures were purchased to meet these expectations and the drafting designs of the enclosures are shown below in figures 15 and 16 and 17.
Figure 15. Communication hub

Figure 16. User hub
Results

After each hub was created and the entire system was connected, the final product was put to the test. A box was setup at each user station and MDM was started. Each screen successfully prompted the user for a Function key and mouse click to assign their USB devices (figure 18). Previous to Design Team 4’s product, any user at this time could register their device and login. With the product in place, however, each user, one at a time, had to push their respective start button in order to gain access to assign their devices. Once the association took place, each user was able to log into their session. The full system was in place with over a week to find bugs or errors in the design. Early returns are very promising, as the system continues to operate in stable manner.
Chapter 5

Cost Analysis

The overall cost of the AUDRI system is displayed below in figure 19. In order to achieve a true comparison to the commercial product Userful, however, the costs of only the materials needed to match their product are displayed. Additional purchases, including video cards, computer, monitors, keyboards, and mice would be needed for any system and multiple varieties can be chosen.
Factoring in bulk pricing, Design Team 4 has matched the commercial product for nearly $50 less per seat based on their educational pricing (figure 20).

<table>
<thead>
<tr>
<th>Parts</th>
<th>Quantity/School</th>
<th>Price</th>
<th>Total Cost/Seat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication Hub Enclosure</td>
<td>1</td>
<td>$5.00</td>
<td>$6.00</td>
</tr>
<tr>
<td>User Hub Enclosure</td>
<td>6</td>
<td>$1.50</td>
<td>$9.00</td>
</tr>
<tr>
<td>Female Audio Input Jacks Chassis</td>
<td>42</td>
<td>$0.63</td>
<td>$26.46</td>
</tr>
<tr>
<td>Audio Cables - Male to Male 12 ft Mono</td>
<td>2</td>
<td>$0.88</td>
<td>$15.84</td>
</tr>
<tr>
<td>Switches</td>
<td>6</td>
<td>$0.60</td>
<td>$3.60</td>
</tr>
<tr>
<td>5 mm LED mounting hardware</td>
<td>12</td>
<td>$0.12</td>
<td>$1.44</td>
</tr>
<tr>
<td>[C MLX/DEMUX 6X1 16DIF</td>
<td>12</td>
<td>$0.45</td>
<td>$5.40</td>
</tr>
<tr>
<td>Microchip PIC16F1826-E/P</td>
<td>1</td>
<td>$4.00</td>
<td>$4.00</td>
</tr>
<tr>
<td>Communication hub circuit board</td>
<td>1</td>
<td>$4.00</td>
<td>$4.00</td>
</tr>
<tr>
<td>T-1 3/4 Red LED (5mm)</td>
<td>6</td>
<td>$0.08</td>
<td>$0.48</td>
</tr>
<tr>
<td>T-1 3/4 Green LED (5mm)</td>
<td>6</td>
<td>$0.10</td>
<td>$0.50</td>
</tr>
<tr>
<td>Resistors</td>
<td>12</td>
<td>$0.19</td>
<td>$2.28</td>
</tr>
<tr>
<td>Audio Cards</td>
<td>6</td>
<td>$2.25</td>
<td>$13.50</td>
</tr>
<tr>
<td>USB Female to PCB - Double</td>
<td>12</td>
<td>$0.97</td>
<td>$11.54</td>
</tr>
<tr>
<td>USB Cables MM (user box to hub) - 10 ft</td>
<td>12</td>
<td>$0.69</td>
<td>$8.28</td>
</tr>
</tbody>
</table>

Figure 19. Price per seat

| AUDRI System Total Cost/Seat:      | $18.75 |
| Userful Total Cost/Seat (Corporate): | $97.25 |
| Userful Total Cost/Seat (Education):  | $67.25 |

| Total Savings/Seat (Corporate):     | $78.50 |
| Total Savings/Seat (Education):     | $48.50 |

Figure 20. Comparing AUDRI to Userful

### Gantt Chart

Team 4 used the Gantt chart (figure 21) to monitor the team’s progress in accomplishing a finished design before design day. The team kept track of the critical path and managed any milestones during the design process. During
the first three weeks of design, the time was dedicated to understanding the project definition and doing the necessary research. The fourth week was spent understanding the previous teams’ designs and attempting to recreate the multi-user setup. However, during this time Team 4 ran into issues while installing the display manager used previously.

The main milestone, installing the multi-seat display manager, took longer than expected; approximately 12 weeks to get it running correctly. During this 12 week time span, the team decided to work on the project in parallel. While working to get MDM running on the computer system, the team also worked toward a hardware solution for seat registration. This idea saved the team valuable time at the end when both systems came together in week 13. Week 14 was spent assembling and finding enclosures for the circuitry. Team 4 dedicated week 15 to testing the design, assembling the registration hubs, and the communication box. This included the wiring of all circuits and building all of the enclosures. Week 16 had been dedicated to finalizing any reports and preparing for presentation.
Conclusion

Looking back at this project, Design Team 4 has certainly come a long way and accomplished quite a bit. For many ECE 480 design teams, this project is a way to experiment and test prototypes that may never see the production line. That however, was not the case for Design Team 4. The team was responsible for researching, brainstorming, designing, building, testing, and shortly after the semester concludes, implementing their design. The goal of the semester was to deliver a high quality product to the user at the lowest cost possible. Commercial products continue to be available but designing a competitive product at a much lower cost is an objective Team 4 undoubtedly achieved. The success of the product now lies in the hands of consumer feedback. The AUDRI system, if successful, may very well have an important impact on multi-user systems for many years to come.
Appendix I

Technical Roles

David Wilson

David Wilson handled the organization and collaboration of the team’s design. He also worked with Billy on developing the concept for the AUDRI system. Specifically, he researched the LED design, acquired the specifications for the enclosures, and helped build the user registration hub used in the final prototype. David also worked on developing the communication hub and worked with Billy to develop a prototype for the switching circuit used in the communication hub. He also researched components that could be used as a switch, such as, relays, transistors, and multiplexers. The result is a circuit that uses multiplexers and a micro-controller, to allow switching between the users keyboard and mouse during MDM registration.

In addition, to working with the Billy on the communication and user boxes, David developed the idea to add LEDs to the user box in order to make the product more intuitive for the children using the system. In doing so, David took on the role of making the system user-friendly. He inquired the idea to put more specific instructions on-screen during the seat registration process; you can see this in the team’s final design. David also assisted Tim with developing the code that programs the micro-controller and flashes the appropriate LED. David wrote the pseudo code in order for Tim to program the microcontroller using MPLAB.
David also learned from other team members how to install and operate a Linux based operating system, Ubuntu. He also spent time understanding how to install and configure packages on the system, so that the system could operate multiple monitors. Moreover, the system testing was performed by the entire team including David. He worked with other team members, testing the sound, testing the communication and user boxes, and being sure that the system would be understood by the student in Tanzania. He also assisted in the idea of creating a manual for the design. Therefore, it is useful for future design teams that plan to continue or update Team 4’s design.

**Tim Haynie**
Tim’s technical role in the project was threefold. First, he assumed the role as the primary programmer for the selected microcontroller used in the final product. As such, he was responsible for becoming familiar with the features, proprietary C programming language instructions, and wiring schematic of the microcontroller. This also required him to become familiar with the MPLAB IDE, the software package that is used to develop, program, and debug the microcontroller. The code which he wrote went through two versions in the design process because the microcontroller and design approach which was originally selected for the project turned out to be a less cost-effective approach. However, a second version of the design allowed for improvement to the structure of the code to make it easier for others to follow and understand.

His second role was that of the “Linux expert” of the team since he had the greatest amount of knowledge and background experience using the Linux
operating system. He provided input, support, and feedback on a regular basis to the other team members whose role was to set up and configure the MDM software. However, he was not one of the team members who ultimately were responsible for becoming familiar with MDM or for configuring the system to add audio to the seats.

His final responsibility was to aid in the design of the hardware circuits. In this role he worked cooperatively with other team members to fabricate a hardware solution that could be implemented within the given time period and for reasonably low cost. This involved working with another team member to develop a schematic, determine what voltages needed to be high and low and when they needed to be so, solving the problem of inter-device communication, and giving input for device and component selection criteria.

Andy Bruinsma

The main technical roles assigned to Andy involved software for the MDM system and sound integration. The first several weeks of the semester were spent researching MDM and analyzing different configuration files in order to become more educated in the software being used. One of the first issues to overcome before MDM could be installed was to also become familiar with Linux Ubuntu. Installing and configuring MDM required knowledge in several areas of Ubuntu. Examples of this were learning how to use terminal, ‘gedit’, and ‘vim’.

The most important task Andy worked on was editing and configuring the MDM configuration files. Being able to understand and alter these files when required was the major challenge Andy faced. Several lines of code were
changed and added in order to make the computer system operational. Another issue of focus was learning how permissions of users and files performed in the operating system. In order to alter the required files, permissions needed to be changed to allow alterations to be saved and eventually executed. Once the alterations were in place, Andy worked on getting the system to boot to MDM upon startup. After this step, bugs had to be worked out in the originally unstable MDM system. Through testing and contacting outside support, Andy was able to implement patches and ideas to correct issues experienced by the team.

In addition to working on the multi-user setup, Andy also worked on separating sound output playback for each user. This was done by determining the USB physical port and assigning it to the respective session. Through testing it could then be determined as to which sound device should be registered to each user login.

Once the software portion of the project was complete, Andy switched his focus to the hardware design and implementation. Andy soldered and wired numerous user hubs in order to assemble the required number of seats.

**Bret Charboneau**

Bret played a crucial role in the setup and integration of our multi-seat terminal. This includes getting MDM to work on a six-user system, getting sound to work separately for each user, allowing users to access USB drives, and creating 3-dimensional representations of the AUDRI and computer systems.
In order to install MDM, knowledge of Ubuntu and the terminal are required. There are several different configuration files that must be changed in Ubuntu so that multiple screens work correctly. Bret, along with Andy, helped setup and change these files so MDM could be installed. It took several weeks and a number of installs to get the correct configurations and due to the lack of documentation from previous teams, MDM had to be installed and configured from scratch.

Bret also helped with the installation and configuration of the sound devices. Getting sound to work on a multi-user system required assigning each USB card to each user. This was done by editing and adding scripts to several different files and changing the boot order for those files. Once the scripts were added, the sound cards could then be assigned to each user through Ubuntu's sound settings.

Additionally Bret was able to allow users to access USB devices. Again this was a permissions problem with MDM and could be fixed through Ubuntu's settings. Once all of the users are given permissions, then the computer only mounts one session at a time.

Finally every good design has a prototype, and one that can be viewed 3-dimensionally helps improve the design before it is created. Bret created all of the 3D and 2D drawings for the computer system and AUDRI system which allowed the group to see what the product was going to look like. Creating these drawings is crucial to any design because it allows the group to change their product on the fly to get an even better result.
Billy Mattingly

Billy was responsible for the development, design, testing, and building of all hardware portions of the project. He also performed extensive research on USB devices, namely human interface devices like keyboards and mice. His efforts in the hardware portion of the project led to his design of the AUDRI system, and its integration with the MDM software.

During his research, he found that a USB device could be enabled and disabled by opening and closing the white data line (see appendix x). His findings concluded that a keyboard and mouse could be switched on and off in order to prevent students from associating them incorrectly. He found that since a USB device is a bi-directional form of communication, a simple transistor could not be used as a switch. A relay and analog multiplexer were considered to be used for the switches.

Billy created a working prototype with both, but found that the relay could not be controlled with a microcontroller, as the analog multiplexer could. The analog multiplexer was also inexpensive, and reliable to temperature limits much higher than necessary. With the switch device of an analog multiplexer chosen, he created a working prototype that was integrated with the programmed microcontroller that Tim Haynie was responsible for. This final prototype fully proved the concept of USB switching, and its use of limiting user mistakes.

He created a schematic (see appendix x) for both the User and Communication hubs, which provided the proper technical reference for the hardware design. He made all design decisions pertaining to enclosure choices,
component types and values, as well as communication links between each hub. Billy also designed the PCB, and soldered it entirely.

Finally, he performed extensive testing of the system to further ensure that it would be reliable when installed in Tanzania. The end result was a complete system that is fully functional with MDM and provides a reduction in association error for users.
Appendix II

References

http://wiki.c3sl.ufpr.br/multiseat/index.php/Compiling_mdm
http://wiki.c3sl.ufpr.br/multiseat/index.php/Installing
http://www.ubuntu.com
http://www.egr.msu.edu/classes/ece480/goodman/spring09/group02/
http://www.userful.com
http://focus.ti.com/lit/ds/symlink/sn74lv4051a-q1.pdf
Appendix III

MDM Configuration Files

xorg.conf.mdm:

Section "Files"
  RgbPath    "/etc/X11/rgb"
  ModulePath "/usr/lib/xorg/modules"
  FontPath   "/usr/share/fonts/X11/misc"
  FontPath   "/usr/share/fonts/X11/cyrillic"
  FontPath   "/usr/share/fonts/X11/100dpi/:unscaled"
  FontPath   "/usr/share/fonts/X11/75dpi/:unscaled"
  FontPath   "/usr/share/fonts/X11/Type1"
  FontPath   "/usr/share/fonts/X11/100dpi"
  FontPath   "/usr/share/fonts/X11/75dpi"
  FontPath   "/var/lib/defoma/x-ttcidfont-conf.d/dirs/TrueType"
EndSection

Section "Module"
  Load "record"
  Load "extmod"
  Load "dbe"
  Load "GLcore"
  Load "glx"
  Load "dri"
  Load "xtrap"
EndSection

Section "ServerFlags"
  Option  "DontVTSwitch" "no"
  Option  "DontZoom"   "yes"
  Option  "DontZap"    "no"
  Option  "AllowMouseOpenFail" "yes"
  # Option  "BlankTime"  "0"
  # Option  "StandbyTime" "0"
  # Option  "SuspendTime" "0"
  # Option  "OffTime"    "0"
EndSection

Section "DRI"
  Mode  0666
EndSection

Section "InputDevice"
  Identifier  "Keyboard"
  Driver      "kbd"
  Option      "CoreKeyboard"
  #Option     "XkbRules" ""
  Option      "XkbModel" "kdb"
  Option      "XkbLayout" "us"
Section "InputDevice"
    Identifier "Mouse"
    Driver  "mouse"
        Option      "CorePointer"
        Option      "Protocol" "auto"
        Option      "Device" "/dev/input/mice"
        Option      "ZAxisMapping" "4 5 6 7"
EndSection

Section "Device"
    Identifier "Video Card 1"
    BusID  "PCI:3:0:0"
    Driver  "nv"
        #Option "UseDisplayDevice" "CRT-0"
EndSection

Section "Monitor"
    Identifier "Monitor 1"
    HorizSync 28-49
    VertRefresh 43-72
    Option  "DPMS"
EndSection

Section "Screen"
    Identifier "Screen 1"
    Device  "Video Card 1"
    Monitor  "Monitor 1"
    DefaultDepth 24
    SubSection "Display"
        Depth 16
        Modes "1024x768"
        Virtual 2048 768
    EndSubSection
    SubSection "Display"
        Depth 24
        Modes "1024x768"
        Virtual 2048 768
    EndSubSection
EndSection

Section "Device"
    Identifier "Video Card 2"
    BusID  "PCI:2:0:0"
    Driver  "nv"
        #Option "UseDisplayDevice" "CRT-1"
EndSection

Section "Monitor"
    Identifier "Monitor 2"
    HorizSync 28-49
    VertRefresh 43-72
    Option  "DPMS"
Section "Screen"
  Identifier  "Screen 2"
  Device      "Video Card 2"
  Monitor     "Monitor 2"
  DefaultDepth 24
  #Option "metamodes" "CRT-1: nvidia-auto-select +0+0"
  SubSection "Display"
    Depth 16
    Modes "1024x768"
    Virtual 2048 768
  EndSubSection
  SubSection "Display"
    Depth 24
    Modes "1024x768"
    Virtual 2048 768
  EndSubSection
EndSection

Section "Device"
  Identifier  "Video Card 3"
  BusID       "PCI:1:0:0"
  Driver      "nv"
  #Option "UseDisplayDevice" "CRT-2"
EndSection

Section "Monitor"
  Identifier  "Monitor 3"
  HorizSync 28-49
  VertRefresh 43-72
  Option      "DPMS"
EndSection

Section "Screen"
  Identifier  "Screen 3"
  Device      "Video Card 3"
  Monitor     "Monitor 3"
  DefaultDepth 24
  #Option "metamodes" "CRT-2: nvidia-auto-select +0+0"
  SubSection "Display"
    Depth 16
    Modes "1024x768"
    Virtual 2048 768
  EndSubSection
  SubSection "Display"
    Depth 24
    Modes "1024x768"
    Virtual 2048 768
  EndSubSection
EndSection

Section "ServerLayout"
  Identifier  "X Configured"
Screen 0 "Screen 1"
Screen 1 "Screen 2"
Screen 2 "Screen 3"

InputDevice  "Keyboard"
InputDevice  "Mouse"

EndSection

mdm.conf:

#!/bin/bash

# This is the mdm configuration file.

# create_xorg_conf script options:

RECREATE_XORG_CONF='no'

# Screen resolution for all screens:
DEFAULT_MODE='1024x768'

# Mode for specific head, ignores DEFAULT_MODES
MODE[1]='1280x768'

# Monitor frequency:
DEFAULT_HORIZ_SYNC="28-49"
DEFAULT_VERT_REFRESH="43-72"

# Specific heads
HORIZ_SYNC[1]='none'
VERT_REFRESH[1]='none'

# Keyboard options:
DEFAULT_XKB_RULES='evdev'
DEFAULT_XKB_MODEL='evdev'
DEFAULT_XKB_LAYOUT='us'

# Specific heads
XKB_RULES[1]='evdev'
XKB_MODEL[1]='abnt2'
XKB_LAYOUT[1]='br'

# mdm and sub-scripts options:

# Enable this option to disable multiseat if you have multiple video cards
# (xorg.conf will be used instead of xorg.conf.mdm)
# Values: 'yes', 'no'
USE_MONOSEAT='no'

# Reconfigure the input devices every time (re-creating the links)
# Values: 'yes', 'no'.
RECONFIGURE_INPUT='yes'

# Selects how you want to run multiseat
# Values: `ls /usr/share/mdm/modes` (you probably want 'xephyr-gdm')
MULTISEAT_MODE='xephyr-gdm'

########################################
# xephyr-xdmcp options:

# Connection type:
# Values: 'indirect' or 'query'
CONNECTION_TYPE='query'

# Connection target:
# Values: your xdmcp server.
CONNECTION_TARGET='xdmcpserver'

# Sound
SOUND[1]="usb-0000:00:1d.2-1.3"
SOUND[2]="usb-0000:00:1d.2-2.3"
SOUND[3]="usb-0000:00:1a.0-1.3"
SOUND[4]="usb-0000:00:1d.0-2.3"
SOUND[5]="usb-0000:00:1d.3-2.3"
SOUND[6]="usb-0000:00:1d.3-1.3"

discover-devices:

#!/bin/bash

# Copyright (C) 2004-2007 Centro de Computacao Cientifica e Software Livre
# Departamento de Informatica - Universidade Federal do Parana - C3SL/UFPR
#
# 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 the
# 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., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301,
# USA.
# This script discovers evdev mice and keyboards and also discovers video cards
# on the machine.
# For input devices, it uses /proc/bus/input/devices
# For video cards, it uses the "discover" package.

# TODO: find a decent way to do all this.

# This function prints the physical addresses of the mice found
function discover_input () {
    KEYBOARDS=($(hal-find-by-capability --capability input.keyboard))
    MICE=($(hal-find-by-capability --capability input.mouse))
    for i in ${KEYBOARDS[@]}; do
        EVDEV_NODE=$(hal-device $i | grep linux.device_file | cut -d"" -f2)
        echo -e "kbd	${i}"
        echo -e "kevdev	${EVDEV_NODE}"
    done
    for i in ${MICE[@]}; do
        EVDEV_NODE=$(hal-device $i | grep linux.device_file | cut -d"" -f2)
        echo -e "mouse	${i}"
        echo -e "mevdev	${EVDEV_NODE}"
    done
}

# Prints bus address and drivers of the video cards.
function video_cards () {
    IDS_BY_DRIVER=/usr/share/xserver-xorg/pci
    # 'discover' prints in the reverse order of lspci, I think
    BUS_IDS=($(lspci | grep VGA | cut -d' ' -f1 | tac))
    for (( i=0; i < ${#BUS_IDS[@]}; i++ )); do
        PCI_DEVICE=$(lspci -n -s ${BUS_IDS[i]} | cut -d' ' -f3)
        # We might find multiple drivers. Use the first.
        DRIVERS[i]==$(grep -i ${PCI_DEVICE/:/} $IDS_BY_DRIVER/* | cut -d'/' -f6 | cut -d'.' -f1 | head -n 1)
    done
    for (( i=0 ; i < ${#BUS_IDS[@]} ; i++ )); do
        # bus id from lspci is in format 00:00.00
        # below we split in 00 and 00.00
        NUMS=(`echo ${BUS_IDS[$i]} | awk 'BEGIN {FS=":"}{print toupper($1), toupper($2)}'`)
        # now, we split 00.00 in 00 and 00
        SEC_NUMS=(`echo ${NUMS[1]} | awk 'BEGIN {FS="."}{print toupper($1), toupper($2)}'`)
        # now, we convert the numbers from hexa to decimal base
        echo -e "bus	`echo "obase=10;ibase=16;${NUMS[0]};${SEC_NUMS[0]};${SEC_NUMS[1]}" | bc | paste -s -d"::"`"
    done
}
for i in ${DRIVERS[@]}; do
echo -e "driver\t$i"
done

} # video_cards

# *********************** MAIN ***********************

if [[ "$#" = 0 ]]
then
    ARG=all
else
    ARG=$1
fi

case $ARG in
    all)
      CARDS=`video_cards`
      INPUT=`discover_input`
      echo "$INPUT" | grep mouse
      echo "$INPUT" | grep mevdev
      echo "$INPUT" | grep kbd
      echo "$INPUT" | grep kevdev
      echo "$CARDS"
      ;;
    all2)
      CARDS=`video_cards`
      INPUT=`discover_input`
      echo "$INPUT"
      echo "$CARDS"
      ;;
    mouse|mevdev|kbd|kevdev)
      INPUT=`discover_input`
      echo "$INPUT" | grep $ARG
      ;;
    bus|driver)
      CARDS=`video_cards`
      echo "$CARDS" | grep $ARG
      ;;
    cards)
      CARDS=`video_cards`
      echo "$CARDS"
      ;;
    #debug)
    #    echo "$INPUT" | egrep "(PHYS | HANDLER | NAME | --)"
    #    ;;
    *)
      echo "$1: unrecognized argument"
      echo "Valid parameters: "
      echo 'mouse: prints the hal ids of the mice'
      echo 'mevdev: prints the evdev mouse events'
      echo 'kbd: prints the hal ids of the keyboards"
echo "kevdev: prints the evdev keyboard events"
echo "driver: prints the video card drivers"
echo "cards: prints the video card bus IDs"
echo "all: prints everything"
;;
esac

mdm-bin:

#!/bin/bash

# Copyright (C) 2004-2007 Centro de Computacao Cientifica e Software Livre
# Departamento de Informatica - Universidade Federal do Parana - C3SL/UFPR
#
# 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 the
# 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., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301,
# USA.
#
# This is the script that is called by init.d/mdm, it is the one that actually
# starts the "hard work".
#
# Do this first:
MDM_PREFIX=/
MDM_SCRIPTS=${MDM_PREFIX}/usr/sbin
MDM_INCLUDE=${MDM_SCRIPTS}/mdm-common
source $MDM_INCLUDE

MY_LOG=${MDM_LOGS}/mdm.log
MY_PIDFILE=${MDM_PIDS}/mdm.pid
CREATE_XORG_CONF=${MDM_SCRIPTS}/create-xorg-conf
START_SEAT=${MDM_SCRIPTS}/mdm-start-seat
START_MONOSEAT=${MDM_SCRIPTS}/mdm-start-monoseat

function exec_start() {
  log --log-file-only "Multiseat Display Manager version $MDM_VERSION"
  log --log-file-only "Today is $(date)."

  if ([ "$RECREATE_XORG_CONF" = 'yes' ] || [ ! -f "$MDM_XORG_CONF" ]); then
    $CREATE_XORG_CONF
  fi
if [ "$RECONFIGURE_INPUT" = 'yes' ]; then
    # Removing links AND locks
    for i in `ls $MDM_DEVICES`; do
        rm -f $MDM_DEVICES/$i
        rm -f $KEYBOARD
        rm -f $MOUSE
    done
fi

VIDEO_CARDS=$($DISCOVER_DEVICES driver | wc -l)
log --log-file-only "$VIDEO_CARDS video cards detected"

# Other scripts assume we do this:
export DISPLAY=:0

# erase links, locks, logs, close stuff, whatever needed
display_manager_init

# check result
if [ "$?" != "0" ]; then
    log --log-file-only "Error: failed to initialize $MULTISEAT_MODE"
    #exec_stop
    exit 1
fi

# find out how many seats we actually have
OUTPUTS_PER_CARD=$(grep "SCREEN_AMOUNT" $XRANDR_INFO_FILE | cut -d'=' -f2)
SEATS=0
for i in $OUTPUTS_PER_CARD; do
    SEATS=$(($SEATS+$i))
done

log --log-file-only "$SEATS seats detected"

if [ "$SEATS" = "1" ] || [ "$USE_MONOSEAT" = 'yes' ]; then
    log --log-file-only "Not using multiseat"
    rotate_log ${MDM_LOGS}/mdm-monoseat.log
    $START_MONOSEAT &> ${MDM_LOGS}/mdm-monoseat.log &
    PID=$!
    echo $PID > $MDM_PIDS/mdm-start-monoseat.pid
else
    display_manager_start_underneath_xserver >> $MY_LOG 2>&1
    for ((i=0; i < $VIDEO_CARDS; i++)); do
        export DISPLAY=:0.$i
        xrandr_configure_layout
done

    # Finding how many screens on each card
    SEATS_ON_CARD=`(grep "SCREEN_AMOUNT" $XRANDR_INFO_FILE | cut -d='"' -f2)`

    SEAT_DISPLAY=1
for ((i = 0; i < $VIDEO_CARDS; i++)); do
    export DISPLAY=:0.$i

    # START_SEAT has 2 parameters:
    # - the first is the video card number
    # - the second is the seat display
    for (( seat=0; seat < "$ {SEATS_ON_CARD[i]}"; seat++ )); do
        START_SEAT_LOG="${MDM_LOGS}/mdm-start-seat.${SEAT_DISPLAY}.log"
        rotate_log ${START_SEAT_LOG}

        log --log-file-only "Starting seat $((i+1)) ${SEAT_DISPLAY}"
        ${START_SEAT} $((i+1)) $SEAT_DISPLAY &> $START_SEAT_LOG &
        PID=$!
        echo $PID > ${MDM_PIDS}/start-seat.${SEAT_DISPLAY}.pid
        ((SEAT_DISPLAY++))
    done
fi
# hack to get other monitors to come up
#/usr/sbin/mdm-extra-seats > $(MDM_LOGS)/mdm-start-seat-extras.log
}

function exec_stop() {
    log --log-file-only "Stopping mdm..."
    display_manager_stop

    for i in $(ls $MDM_PIDS/*.pid 2> /dev/null | grep -v mdm.pid); do
        rkill $(cat $i) 2> /dev/null
        rm -f $i
    done

    rm -f $MY_PIDFILE
}

function run_service() {
    # /var/run is re-created at every reboot
    mkdir -p ${MDM_RUN}
    echo $$ > $MY_PIDFILE

    trap exec_stop SIGTERM

    exec_start

    # now we sleep until we get a SIGTERM or all our children die
    wait

    log --log-file-only "All children died. Aborting"
    exec_stop

    return 0
}
function print_help() {

    echo "$0 usage:"
    echo "    $0 [options]"
    echo "where options include:"
    echo "    --reconfigure   starts mdm recreating xorg.conf.mdm and"
    echo "                    reconfiguring input, overwriting values"
    echo "    --version       prints mdm's version"
    echo "    --help          prints this message"
}

# Check if we're root
if [ ! "$(id -u)" -eq "0" ]; then
    log "ERROR: not running as root user"
    exit 1
fi

RUNNING_AS_DAEMON=0

while (( $# > 0 )); do
    case "$1" in
        --daemon)
            RUNNING_AS_DAEMON=1
            shift 1
        ;;
        --reconfigure)
            # overwrite what's written in mdm.conf:
            RECREATE_XORG_CONF='yes'
            RECONFIGURE_INPUT='yes'
            exec_stop
            exec_start
            shift 1
        ;;
        --version)
            echo "$MDM_VERSION"
            exit 0
        ;;
        --help)
            print_help
            exit 0
        ;;
        *)
            echo "$0: unrecognized option: "$1"
            print_help
            exit 1
        ;;
    esac

    done

# This script needs to run as a daemon, so whenever someone call us (probably
# from init.d), we’ll fork to become a real daemon. Since there is no decent
# "fork" in bash, we just do "$0 --daemon $@
if [ "$RUNNING_AS_DAEMON" = "1" ]; then
  cd /
  umask 0

  log --log-file-only "mdm daemon started"

  run_service
  exit $?
else
  # Don't rotate log after becoming a daemon
  rotate_log $MY_LOG

  log --log-file-only "Starting mdm daemon..."

  # Forking, dettaching, closing outputs, etc
  setsid nohup $0 --daemon $@ &> /dev/null & # Run setsid or nohup first?
  exit $?
fi

mdm-common:

#!/bin/bash

# Copyright (C) 2004-2007 Centro de Computacao Cientifica e Software Livre
# Departamento de Informatica - Universidade Federal do Parana - C3SL/UFPR
#
# 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 the
# 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., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301,
# USA.
#
# This is the file that is included by everyone. It should contain all the
# variables that are common to every function.

MDM_VERSION='0.0.3'

MDM_PREFIX=/

MDM_LOGS=$MDM_PREFIX/var/log/mdm/
MDM_RUN=$MDM_PREFIX/var/run/mdm/
MDM_ETC=$MDM_PREFIX/etc/mdm/
MDM_SHARE=$MDM_PREFIX/usr/share/mdm/
MDM_SCRIPTS=$MDM_PREFIX/usr/sbin/
MDM_LOCALE=$MDM_PREFIX/usr/share/locale/

CONFIG_FILE=$MDM_ETC/mdm.conf

MDM_DEVICES=$MDM_ETC/devices/
MDM_MODES=$MDM_SHARE/modes/
MDM_PIDS=$MDM_RUN/

MDM_XORG_CONF=$MDM_ETC/xorg.conf.mdm
DISCOVER_DEVICES=$MDM_SCRIPTS/discover-devices
XRANDR_FUNCTIONS=$MDM_SCRIPTS/xrandr-functions

source $XRANDR_FUNCTIONS

function log() {
    # Whoever calls this must set the $MY_LOG variable!
    ME=`basename $0`

    #echo "MY_LOG = $MY_LOG $1 $2"
    if [ "$1" == `'-log-file-only'` ]; then
        echo "$ME: $2" >> $MY_LOG
        echo "$(date)" >> $MY_LOG
    else
        echo "$ME: $1" >> $MY_LOG
        echo "$ME: $1"
    fi
}

function debug() {
    if [ [ "$DEBUG" -eq "1" ] ]; then
        echo "$1"
    fi
}

function load_config() {
    # TODO: we should check for the file integrity and for missing variables
    source $CONFIG_FILE
}

function rotate_log () {
    # $1 is the name of the log file
    if [ -f $1 ]; then
        mv $1 $1.old
    fi
}

function select_display_manager () {
    source $MDM_MODES/$MULTISEAT_MODE
}

#function rkill()
#{
#   # PID=$1
}
CHILDREN=$(ps --ppid $PID | awk 'NR>1{print $1}')

kill $PID

for CHILD in $CHILDREN; do
    rkill $CHILD
done

load_config
select_display_manager

xephyr-gdm:

#!/bin/bash

# Copyright (C) 2004-2007 Centro de Computacao Cientifica e Software Livre
# Departamento de Informatica - Universidade Federal do Parana - C3SL/UFPR
# 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 the
# 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., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301,
# USA.
#
# This configuration runs Xephyr on gdm, using gdmdynamic

XEPHYR_WRAPPER=${MDM_SCRIPTS}/xephyr-wrapper
RETRIES=2
SLEEP=1
GDMDYNAMIC="gdmdynamic -t $RETRIES -s $SLEEP"
GDM_SOCKET=/var/run/gdm_socket'

export XAUTHORITY='/var/lib/gdm/:0.Xauth'

function display_manager_init () {
    # This function should stop running processes, remove logs, links, locks, or
    # anything your display manager might need.
    # Consider calling display_manager_stop here if you don’t need anything
    # else.

    # No problem if it’s already running...
display_manager_stop

# /etc/init.d/gdm start

if [ ! -e $GDM_SOCKET ]; then
    log --log-file-only "Error: gdm is not running"
    return 1
fi

return 0
}

function display_manager_start_monoseat () {
    # This function is called when the computer has only 1 video card.
    # You should open only 1 xserver and that's it: normal usage, not multiseat
    # Do NOT configure devices or stuff like that
    # We assume $DISPLAY is already set here.

    $GDMDYNAMIC -a 0=XHandled
    $GDMDYNAMIC -r 0

    echo "Exiting (gdm will handle the X server for us)."
    return 0
}

function display_manager_start_underneath_xserver () {
    # In case we're using nested xservers (like Xephyr), we'll need to start an
    # xserver to run the nested servers on top of it
    # Don't put login screen on this server!
    # We assume $DISPLAY is already set here.

    $GDMDYNAMIC -a 0=XNotHandled
    $GDMDYNAMIC -r 0

    # XXX: find a better way to wait for X to start
    sleep 5

    return 0
}

function display_manager_start_seat () {
    # This is where we start the xserver that will have a login screen
    # In case of multiseat with Xephyr, this function starts the Xephyrs and
    # puts the login screens on them.
    # We assume $DISPLAY is already set here.
    # This function is called by mdm-start-seat, which gives you variables:
    # - $KEYBOARD and $MOUSE, which point to evdev device nodes.
    # - $SEAT_DISPLAY variable corresponds to the number that the user will
    #   have to press on the keyboard. Might be useful here.
    # - $MY_SCREEN_SIZE for the size of screen (needed when running nested
    #   xservers)
    # - $MY_XKB_LAYOUT and $MY_XKB_MODEL, for the keyboard associated with the
    #   seat

    echo "Xephyr :${SEATDISPLAY} -br"
    echo "       -parent $SEAT_WINDOW_ID"
}
echo "       -mouse evdev,,device=$MOUSE"
echo "       -keybd"
echo -n "        evdev,,device=$KEYBOARD,xkbmodel=$MY_XKB_MODEL,"
echo             "xkblayout=$MY_XKB_LAYOUT"
echo "       -${CONNECTION_TYPE} ${CONNECTION_TARGET}"
echo "       -dpi $DPI_VALUE 2>&1 &"

$GDMDYNAMIC "-a $SEAT_DISPLAY=${XEPHYR_WRAPPER} -br $SEAT_DISPLAY
"-display $DISPLAY
-xauthority $XAUTHORITY
-parent $SEAT_WINDOW_ID
-keybd
-evdev,,device=$MOUSE
-"xkblayout=$MY_XKB_LAYOUT"
-dpi $DPI_VALUE"
$GDMDYNAMIC -r $SEAT_DISPLAY

# XXX: We're letting gdm do everything: it will restart Xephyr whenever it
# dies, it will do a lot of stuff. This way, we won't be able to reconfigure
# input. We need to find a way to do this, maybe through gdm itself.
echo "Exiting (gdm will handle the Xephyrs for us)."
exit 0

#function display_manager_stop () {
#   if [-e "$GDM_SOCKET" ]; then
#     SERVERS=($(gdmdynamic -l | tr ';' '
'| cut -d':' -f2 | cut -d',' -f1))
#     for i in ${SERVERS[@]}; do
#       $GDMDYNAMIC -d $i
#     done
#   fi
#}

mdm-start-seat:

#!/bin/bash

# Copyright (C) 2004-2007 Centro de Computacao Cientifica e Software Livre
# Departamento de Informatica - Universidade Federal do Parana - C3SL/UFPR
# 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 the


This script starts a seat.
Its arguments are:
- The number of the "video card". Video card N has display 0.(N-1) on the
  underneath xserver.
- The number of the display that the seat will have to open. This should be
  unique for all seats.

We assume $DISPLAY is already set here
key_press)
    $WRITE_MESSAGE $SEAT_WINDOW_ID "$Hold the START button and then press F$2" 1>&2
button_press)
    $WRITE_MESSAGE $SEAT_WINDOW_ID "$Click the mouse left button" 1>&2
reconfigure)
    $WRITE_MESSAGE $SEAT_WINDOW_ID "$Press ESC to reconfigure" 1>&2
wait)
    $WRITE_MESSAGE $SEAT_WINDOW_ID "$Please wait" 1>&2

done)
    $WRITE_MESSAGE $SEAT_WINDOW_ID "$Press the DONE button" 1>&2
esac

function parse_config_file () {
    if [ ! -z ${XKB_MODEL[VIDEO_CARD_NUMBER]} ]; then
        MY_XKB_MODEL=${XKB_MODEL[VIDEO_CARD_NUMBER]}
    else
        MY_XKB_MODEL=$DEFAULT_XKB_MODEL
    fi
    if [ ! -z ${XKB_LAYOUT[VIDEO_CARD_NUMBER]} ]; then
        MY_XKB_LAYOUT=${XKB_LAYOUT[VIDEO_CARD_NUMBER]}
    else
        MY_XKB_LAYOUT=$DEFAULT_XKB_LAYOUT
    fi
    if [ ! -z ${MODE[SEATDISPLAY]} ]; then
        MY_SCREEN_SIZE=${MODE[SEATDISPLAY]}
    else
        MY_SCREEN_SIZE=$DEFAULT_MODE
    fi
}

function select_keyboard () {
    if -e "${MDM_DEVICES}/keyboard${SEAT_DISPLAY}" ]; then
        KEYBOARD="${MDM_DEVICES}/keyboard${SEAT_DISPLAY}"
    return
    fi

display_message key_press $SEAT_DISPLAY

CREATED=0
while ( [ ! CREATED ] ); do
    # Who are the keyboards?
    KEYBOARDS="${DISCOVER_DEVICES kevdev | cut -f2}"
    #echo " Keyboards = $KEYBOARDS" 1>&2
    #echo "$KEYBOARD" 1>&2
    break
    key_press)
    $WRITE_MESSAGE $SEAT_WINDOW_ID "$Hold the START button and then press F$2" 1>&2
button_press)
    $WRITE_MESSAGE $SEAT_WINDOW_ID "$Click the mouse left button" 1>&2
reconfigure)
    $WRITE_MESSAGE $SEAT_WINDOW_ID "$Press ESC to reconfigure" 1>&2
wait)
    $WRITE_MESSAGE $SEAT_WINDOW_ID "$Please wait" 1>&2

done)
    $WRITE_MESSAGE $SEAT_WINDOW_ID "$Press the DONE button" 1>&2
esac

function parse_config_file () {
    if [ ! -z ${XKB_MODEL[VIDEO_CARD_NUMBER]} ]; then
        MY_XKB_MODEL=${XKB_MODEL[VIDEO_CARD_NUMBER]}
    else
        MY_XKB_MODEL=$DEFAULT_XKB_MODEL
    fi
    if [ ! -z ${XKB_LAYOUT[VIDEO_CARD_NUMBER]} ]; then
        MY_XKB_LAYOUT=${XKB_LAYOUT[VIDEO_CARD_NUMBER]}
    else
        MY_XKB_LAYOUT=$DEFAULT_XKB_LAYOUT
    fi
    if [ ! -z ${MODE[SEATDISPLAY]} ]; then
        MY_SCREEN_SIZE=${MODE[SEATDISPLAY]}
    else
        MY_SCREEN_SIZE=$DEFAULT_MODE
    fi
}

function select_keyboard () {
    if -e "${MDM_DEVICES}/keyboard${SEAT_DISPLAY}" ]; then
        KEYBOARD="${MDM_DEVICES}/keyboard${SEAT_DISPLAY}"
    return
    fi

display_message key_press $SEAT_DISPLAY

CREATED=0
while ( [ ! CREATED ] ); do
    # Who are the keyboards?
    KEYBOARDS="${DISCOVER_DEVICES kevdev | cut -f2}"
    #echo " Keyboards = $KEYBOARDS" 1>&2
    #echo "$KEYBOARD" 1>&2
    break

# Remove used keyboards

```bash
REMOVE=$(stat -c %N $MDM_DEVICES/keyboard* 2> /dev/null |
    cut -d"" -f3 | cut -d"" -f1)
for i in $REMOVE; do
    KEYBOARDS=$(sed "s#$i\b##g" <<< $KEYBOARDS)
done
```n

```bash
# See if someone presses the key:
PRESSED=$(ps -e $MDM_DEVICES $SEAT_DISPLAY $KEYBOARDS | grep '^detect'
    | cut -d'|' -f2)
if [ -z "$PRESSED" ]; then
    # if $READ_DEVICES gets killed the script won't do bad stuff
    continue
else
    if [ "$PRESSED" = 'timeout' ]; then
        continue
    fi
    # Ok, someone pressed the key
    ln -sf $PRESSED $MDM_DEVICES/keyboard$SEAT_DISPLAY
    CREATED=1
    # Verify is there is already another link in $MDM_DEVICES/keyboard* that
    # points to the device. If there is, erase the one I created and
    # continue looping
    for i in `ls $MDM_DEVICES | grep \"<keyboard\``; do
        if [ "$i" != "keyboard${SEAT_DISPLAY}" ]; then
            AUX=$(stat -c %N $MDM_DEVICES/$i | cut -d"" -f3 | cut -d"" -f1)
            if [ "$AUX" = "$PRESSED" ]; then
                # Keyboard link already exists...
                rm -f $MDM_DEVICES/keyboard$SEAT_DISPLAY
                CREATED=0
            fi
        fi
done
```n

```bash
KEYBOARD="$MDM_DEVICES/keyboard$SEAT_DISPLAY"
```n

```bash
function select_mouse () {
    if [ -e "${MDM_DEVICES}/mouse${SEAT_DISPLAY}" ]; then
        MOUSE="${MDM_DEVICES}/mouse${SEAT_DISPLAY}"
        return
    fi
}
CREATED=0
while (( ! CREATED )); do
  # Who are the mice?
  MICE=$($DISCOVER_DEVICES mevdev | cut -f2)

  REMOVE=$(stat -c %N $MDM_DEVICES/mouse* /dev/null |
          cut -d"" -f3 | cut -d"" -f1)
  for i in ${REMOVE}; do
    MICE=$(sed "s#$i\b##g" <<< $MICE)
  done

  if [ -z "$MICE" ]; then
    # No mice. Hopefully someone will connect one.
    sleep 1 # Don’t use 100% CPU
    continue
  fi

  # Create the lock
  LOCK_EXISTS=1
  while (( LOCK_EXISTS )); do
    touch $MDM_DEVICES/lock$SEAT_DISPLAY
    LOCK_EXISTS=0
    for i in `ls $MDM_DEVICES | grep "<lock"`; do
      if [ "$i" != "lock$SEAT_DISPLAY" ]; then
        LOCK_EXISTS=1
      fi
    done
  done

  if ( ( LOCK_EXISTS )); then
    # Yes, we’ll call this lots of times...
    display_message wait
    rm -f $MDM_DEVICES/lock$SEAT_DISPLAY
    sleep 1;
  fi
done

# Now we have the lock!
display_message button_press

# See if someone presses the button:
PRESSED=$($READ_DEVICES 13 $MICE | grep '^detect' | cut -d'|' -f2)

if [ -z "$PRESSED" ]; then
  # If $READ_DEVICES gets killed, don’t do unwanted stuff
  rm -f $MDM_DEVICES/lock$SEAT_DISPLAY
  continue
fi

if [ "$PRESSED" = 'timeout' ]; then
  # Wait 5 seconds to give other machines the opportunity to enter the
  # lock
  display_message wait
  rm -f $MDM_DEVICES/lock$SEAT_DISPLAY
  sleep 5
  continue
fi
# Ok, someone pressed the key
ln -sf $PRESSED ${MDM_DEVICES}/mouse${SEAT_DISPLAY}
CREATED=1
display_message done
sleep 3

# Verify is there is already another link in $MDM_DEVICES/mouse* that
# points to the device. If there is, erase the one I created and
# continue looping
for i in `ls $MDM_DEVICES | grep "<mouse"`; do
  if [ "$i" != "mouse${SEAT_DISPLAY}" ]; then
    AUX=$(stat -c %N ${MDM_DEVICES}/$i| cut -d'`' -f3|cut -d"" -f1)
    if [ "$AUX" = "$PRESSED" ]; then
      # Mouse link already exists...
      rm -f ${MDM_DEVICES}/mouse${SEAT_DISPLAY}
      CREATED=0
    fi
  fi
done
rm -f ${MDM_DEVICES}/lock${SEAT_DISPLAY}
done

MOUSE="${MDM_DEVICES}/mouse${SEAT_DISPLAY}"

parse_config_file

KEYBOARD=
MOUSE=
SEAT_WINDOW_ID=
SCREEN_X_ORIGIN=($(grep -A4 $DISPLAY $XRANDR_INFO_FILE | tail -1 | cut -d'=' -f2))

# XXX: In case $SCREEN_X_ORIGIN has only one value, and SEAT_DISPLAY is odd,
# the open_seat_parent_window given parameter would be empty
# ($SCREEN_X_ORIGIN[1] for example).

open_seat_parent_window ${SCREEN_X_ORIGIN[$((SEAT_DISPLAY % 2))]:=0}

while (( 1 )); do
  #log --log-file-only "Configuring devices...."
  echo -e "\n--"
  echo "Hello, now is "$(date)."
  echo "Configuring seat:"
  echo " selecting keyboard"
  select_keyboard
  echo " selecting mouse"
  select_mouse
  echo " starting seat"
  display_manager_start_seat
  display_message reconfigure
PRESSED=$(READ_DEVICES 14 KEYBOARD | grep '^detect' | cut -d'|' -f2)
if [ "$PRESSED" = 'esc' ]; then
    rm -f KEYBOARD
    rm -f MOUSE
fi
done

asound_config:

#!/bin/sh
config_file="/etc/mdm/mdm.conf"
[[ -f $config_file ]] || exit 1
. $config_file
ncards=${#SOUND[@]}
index=1
echo "#automatically created sound file DO NOT EDIT" > /etc/asound_card.conf
OIFS=$IFS
IFS=';'
for card in `awk '{i1=$0; getline ; i2=$0 ; x= i1 i2 ; print x ";"} /proc/asound/cards` ; do
    IFS=$OIFS
    index=1
    while [ "$index" -le "$ncards" ] ; do
        if echo $card | grep ${SOUND[$index]} -qs ; then
            echo SOUND$index=`echo $card | awk '{print $1}'`>> /etc/asound_card.conf
        fi
        let "index = $index + 1"
    done
done
IFS=';'
Microcontroller Source Code

#include <p18cxxx.h> //The microcontroller headers

/*
This section defines individual pins as specific names. This not only makes the code
easier to read but also simplifies any necessary pin reconfigurations. The only other
place a modification might be necessary is in the TRIS statements used for defining
ports and pins as inputs or outputs.
*/

#define BUTTON1 PORTBbits.RB1
#define BUTTON2 PORTBbits.RB2
#define BUTTON3 PORTBbits.RB3
#define BUTTON4 PORTBbits.RB4
#define BUTTON5 PORTBbits.RB5
#define BUTTON6 PORTBbits.RB6

#define REDLED1 PORTBbits.RB0
#define REDLED2 PORTBbits.RB7
#define REDLED3 PORTCbits.RC0
#define REDLED4 PORTCbits.RC1
#define REDLED5 PORTCbits.RC2
#define REDLED6 PORTCbits.RC3

#define GREENLED1 PORTCbits.RC4
#define GREENLED2 PORTCbits.RC5
#define GREENLED3 PORTCbits.RC6
#define GREENLED4 PORTCbits.RC7
#define GREENLED5 PORTDbits.RD0
#define GREENLED6 PORTDbits.RD1

#define MUXOUT1 PORTEbits.RE0
#define MUXOUT2 PORTDbits.RD3
#define MUXOUT3 PORTDbits.RD4
#define MUXOUT4 PORTDbits.RD5
#define MUXOUT5 PORTDbits.RD6
#define MUXOUT6 PORTDbits.RD7

/*
These variables are used to track whether a user is already registered.
*/
int user1registered;
int user2registered;
int user3registered;
int user4registered;
int user5registered;
int user6registered;

/*
These variables are used to track how long a user has held their associated button.
*/
int buttontimer1;
int buttontimer2;
int buttontimer3;
int buttontimer4;
int buttontimer5;
int buttontimer6;

// This variable is used as a timer to turn LEDs on or off.
unsigned int LEDtimer;

/*
These functions are used to register each individual user. They are defined below.
*/
void RegisterUser1();
void RegisterUser2();
void RegisterUser3();
void RegisterUser4();
void RegisterUser5();
void RegisterUser6();

// The main function
void main()
{
    // initially set all users as unregistered
    user1registered = 0;
    user2registered = 0;
    user3registered = 0;
    user4registered = 0;
    user5registered = 0;
    user6registered = 0;

    // start the timer at 0
    LEDtimer = 0;

    // set the input and output pins
    TRISE = 0b00000000;  // all pins on PORTE are outputs
    TRISB = 0b01111110;  // pins 33 and 40 are output, 34-39 are input. This is so
    that RB1-RB6 correspond to button inputs for simplicity
    TRISC = 0x00;  // pins 15-19, 23-26 are output
    TRISD = 0x00;  // pins 19-22, 27-30 are output

    /*
    As a precaution, explicitly set ALL pins as low. These lines were added when it
    was discovered that occasionally a stray voltage on a pin at startup could cause an
    input to accidentally go high.
    */
    PORTBbits.RB0 = 0;
    PORTBbits.RB1 = 0;
    PORTBbits.RB2 = 0;
    PORTBbits.RB3 = 0;
    PORTBbits.RB4 = 0;
    PORTBbits.RB5 = 0;
    PORTBbits.RB6 = 0;
    PORTBbits.RB7 = 0;
    PORTCbits.RC0 = 0;
    PORTCbits.RC1 = 0;
PORTCbits.RC2 = 0;
PORTCbits.RC3 = 0;
PORTCbits.RC4 = 0;
PORTCbits.RC5 = 0;
PORTCbits.RC6 = 0;
PORTCbits.RC7 = 0;
PORTDbits.RD0 = 0;
PORTDbits.RD1 = 0;
PORTDbits.RD2 = 0;
PORTDbits.RD3 = 0;
PORTDbits.RD4 = 0;
PORTDbits.RD5 = 0;
PORTDbits.RD6 = 0;
PORTDbits.RD7 = 0;

//implicitly declare all outputs as off
REDLED1 = 0;
REDLED2 = 0;
REDLED3 = 0;
REDLED4 = 0;
REDLED5 = 0;
REDLED6 = 0;
GREENLED1 = 0;
GREENLED2 = 0;
GREENLED3 = 0;
GREENLED4 = 0;
GREENLED5 = 0;
GREENLED6 = 0;
MUXOUT1 = 0;
MUXOUT2 = 0;
MUXOUT3 = 0;
MUXOUT4 = 0;
MUXOUT5 = 0;
MUXOUT6 = 0;

while (1) //loops forever
{
    if (BUTTON1 && !user1registered) /*if user 1’s button is being pressed
    AND user 1 is not registered*/
    {
        buttontimer1++; //increment user 1’s timer
        if (buttontimer1 > 5000) RegisterUser1(); /*if the timer has
incremented 5000 times, begin the registration process for user 1*/
    }
    else buttontimer1 = 0; /*otherwise, set the timer to 0. This forces the
user to hold the button for a set period of time as a countermeasure to accidental
pushes*/

    //Likewise for user’s 2 through 6*/
    if (BUTTON2 && !user2registered)
    {
        buttontimer2++;
    }
if (buttontimer2 > 5000) RegisterUser2();
else buttontimer2 = 0;

if (BUTTON3 && !user3registered)
{
    buttontimer3++;
    if (buttontimer3 > 5000) RegisterUser3();
}
else buttontimer3 = 0;

if (BUTTON4 && !user4registered)
{
    buttontimer4++;
    if (buttontimer4 > 5000) RegisterUser4();
}
else buttontimer4 = 0;

if (BUTTON5 && !user5registered)
{
    buttontimer5++;
    if (buttontimer5 > 5000) RegisterUser5();
}
else buttontimer5 = 0;

if (BUTTON6 && !user6registered)
{
    buttontimer6++;
    if (buttontimer6 > 5000) RegisterUser6();
}
else buttontimer6 = 0;

LEDtimer++;//increment the LED timer
if (LEDtimer >= 5000) //if the LED timer has incremented 5000 times */
{
    LEDtimer = 0; //reset the timer
    /*toggle the green LED's if that user is not registered. This will cause the LEDs to blink if not registered*/
    if (!user1registered) GREENLED1 = !GREENLED1;
    if (!user2registered) GREENLED2 = !GREENLED2;
    if (!user3registered) GREENLED3 = !GREENLED3;
    if (!user4registered) GREENLED4 = !GREENLED4;
    if (!user5registered) GREENLED5 = !GREENLED5;
    if (!user6registered) GREENLED6 = !GREENLED6;
}
} //end while (1)
} //end void main()

/*
These are the function definitions that register each individual user. Only user 1 will have code comments, but the methodology used for user 1 carries over to all other users.
*/
void RegisterUser1()
{
    // turn LED1 solid green, all other green LEDs off
    GREENLED1 = 1;
    GREENLED2 = 0;
    GREENLED3 = 0;
    GREENLED4 = 0;
    GREENLED5 = 0;
    GREENLED6 = 0;

    // turn on everyone else's red LED
    REDLED2 = 1;
    REDLED3 = 1;
    REDLED4 = 1;
    REDLED5 = 1;
    REDLED6 = 1;

    /*turn on the output pin which will allow the keyboard and mouse to register*/
    MUXOUT1 = 1;

    LEDtimer = 0; // reset the LED timer
    buttontimer1 = 0; // reset the timer for button 1
    while(BUTTON1); /* wait for them to release the button. The green LED will
    remain solid until the button is released*/

    while (buttontimer1 < 5000) /* wait until button 1's timer is incremented 5000
    times consecutively without interruption to counter accidental presses*/
    {
        if (BUTTON1) buttontimer1++;
        else buttontimer1 = 0;
        LEDtimer++;
        if (LEDtimer > 5000)
        {
            LEDtimer = 0;
            GREENLED1 = !GREENLED1; /* toggle user 1's Green LED*/
        }
    }

    user1registered = 1; // set user 1 as registered
    LEDtimer = 0; // reset the timer
    GREENLED1 = 0; // turn off user 1's green LED

    // turn off everyone else's red LED
    REDLED2 = 0;
    REDLED3 = 0;
    REDLED4 = 0;
    REDLED5 = 0;
    REDLED6 = 0;
}

/* RegisterUser2 through RegisterUser6 follow the same methodology as
RegisterUser1*/
void RegisterUser2()
{
GREENLED2 = 1;
GREENLED1 = 0;
GREENLED3 = 0;
GREENLED4 = 0;
GREENLED5 = 0;
GREENLED6 = 0;
REDLED1 = 1;
REDLED3 = 1;
REDLED4 = 1;
REDLED5 = 1;
REDLED6 = 1;
MUXOUT2 = 1;

LEDtimer = 0;
buttontimer2 = 0;
while(BUTTON2); //wait for them to release the button
while (buttontimer2 < 5000)
{
    if (BUTTON2) buttontimer2++;
    else buttontimer2 = 0;
    LEDtimer++;
    if (LEDtimer > 5000)
    {
        LEDtimer = 0;
        GREENLED2 = !GREENLED2;
    }
}
user2registered = 1;
LEDtimer = 0;
GREENLED2 = 0;
REDLED1 = 0;
REDLED3 = 0;
REDLED4 = 0;
REDLED5 = 0;
REDLED6 = 0;

void RegisterUser3()
{
    GREENLED3 = 1;
    GREENLED2 = 0;
    GREENLED1 = 0;
    GREENLED4 = 0;
    GREENLED5 = 0;
    GREENLED6 = 0;
    REDLED2 = 1;
    REDLED1 = 1;
    REDLED4 = 1;
    REDLED5 = 1;
    REDLED6 = 1;
MUXOUT3 = 1;
LEDtimer = 0;
buttontimer3 = 0;
while (BUTTON3); //wait for them to release the button
while (buttontimer3 < 5000)
{
    if (BUTTON3) buttontimer3++;
    else buttontimer3 = 0;
    LEDtimer++;
    if (LEDtimer > 5000)
    {
        LEDtimer = 0;
        GREENLED3 = !GREENLED3;
    }
}
user3registered = 1;
LEDtimer = 0;
GREENLED3 = 0;
REDLED2 = 0;
REDLED1 = 0;
REDLED4 = 0;
REDLED5 = 0;
REDLED6 = 0;
}
void RegisterUser4()
{
    GREENLED4 = 1;
    GREENLED2 = 0;
    GREENLED3 = 0;
    GREENLED1 = 0;
    GREENLED5 = 0;
    GREENLED6 = 0;
    REDLED2 = 1;
    REDLED3 = 1;
    REDLED1 = 1;
    REDLED5 = 1;
    REDLED6 = 1;
    MUXOUT4 = 1;
    LEDtimer = 0;
    buttontimer4 = 0;
    while (BUTTON4); //wait for them to release the button
    while (buttontimer4 < 5000)
    {
        if (BUTTON4) buttontimer4++;
        else buttontimer4 = 0;
        LEDtimer++;
        if (LEDtimer > 5000)
        {
            LEDtimer = 0;
        }
GREENLED4 = !GREENLED4;

user4registered = 1;
LEDtimer = 0;
GREENLED4 = 0;
REDLED2 = 0;
REDLED3 = 0;
REDLED1 = 0;
REDLED5 = 0;
REDLED6 = 0;

void RegisterUser5()
{
    GREENLED5 = 1;
    GREENLED2 = 0;
    GREENLED3 = 0;
    GREENLED4 = 0;
    GREENLED1 = 0;
    GREENLED6 = 0;
    REDLED2 = 1;
    REDLED3 = 1;
    REDLED4 = 1;
    REDLED1 = 1;
    REDLED6 = 1;
    MUXOUT5 = 1;
    LEDtimer = 0;
    buttontimer5 = 0;
    while(BUTTON5); //wait for them to release the button
    while (buttontimer5 < 5000)
    {
        if (BUTTON5) buttontimer5++;
        else buttontimer5 = 0;
        LEDtimer++;
        if (LEDtimer > 5000)
        {
            LEDtimer = 0;
            GREENLED5 = !GREENLED5;
        }
    }
    user5registered = 1;
    LEDtimer = 0;
    GREENLED5 = 0;
    REDLED2 = 0;
    REDLED3 = 0;
    REDLED4 = 0;
    REDLED1 = 0;
    REDLED6 = 0;
void RegisterUser6()
{
    GREENLED6 = 1;
    GREENLED2 = 0;
    GREENLED3 = 0;
    GREENLED4 = 0;
    GREENLED5 = 0;
    GREENLED1 = 0;
    REDLED2 = 1;
    REDLED3 = 1;
    REDLED4 = 1;
    REDLED5 = 1;
    REDLED1 = 1;

   MUXOUT6 = 1;

    LEDtimer = 0;
    buttontimer6 = 0;
    while(BUTTON6); //wait for them to release the button
    while (buttontimer6 < 5000)
    {
        if (BUTTON6) buttontimer6++;
        else buttontimer6 = 0;
        LEDtimer++;
        if (LEDtimer > 5000)
        {
            LEDtimer = 0;
            GREENLED6 = !GREENLED6;
        }
    }

    user6registered = 1;
    LEDtimer = 0;

    GREENLED6 = 0;
    REDLED2 = 0;
    REDLED3 = 0;
    REDLED4 = 0;
    REDLED5 = 0;
    REDLED1 = 0;
}