PI Configuration Implementation Details

Table of Contents

● Add_User(char* NEW_USERNAME)

Creates a new user login on the pi, and creates home directory for the new user. First the existing list of users is queried, so we don’t add a user if it already exists. If the new username doesnt already exists, it is created by executing the shell command:

```
    sudo useradd [USERNAME]
```

Then the user account is added to all the groups on the PI, so the new user account can access all of the PI’s hardware resources. Shell command to add groups to the new user account:

```
    sudo usermod %s -a -G sudo,pi,adm,dialout,cdrom,audio,video,plugdev,
                          users,input,netdev,spi,i2c,gpio
```

● Create_Aliases_File()

Aliases are essentially keywords that represent other commands. They are used to make it simpler to execute otherwise complex or long commands. For instance if you wanted to create an easy way to change the IP Address on the command line, you create an “alias” that represents the longer form of the command.
Example long command:

```
    sudo ifconfig eth0 192.168.250.118        // change IP Address
```

An alias can be created to represent this command. Pick a keyword to represent the above command, say, “setip”.
Example alias for long command:

```
    alias setip=”sudo ifconfig eth0 192.168.250.188”
```

Now entering the command ‘setip’ performs the same function as if you entered the long form of changing eth0 address. Some custom alias commands have been added to make using the pi easier to navigate.

The aliases file where these are defined is location:

```
    /home/pi/.bash_aliases    or    /home/[user]/.bash_aliases
```

Note the file begins with a ‘.’ This file executes when user ‘pi’ logs on. For a different username, the .bash_aliases file needs to be placed in location /home/[user]/.bash_aliases. If you edit this file manually, you wont see the results of changes until reboot. To effect the changes immediately you can enter the command:

```
    source /home/pi/.bash_aliases
```

This file is created by this function. The aliases added are:

`ls`	lists directory contents, but groups directories together and adds color coding
`la`	lists all contents. Same as executing ls -la. Also groups directories and color codes the output.
`dmesg`	returns kernel log information, but presents in human readable format. Timestamps are given as dates and timestamps instead of seconds since epoch
`update`	updates cache of package repository
`upgrade`	updates packages to their latest versiona
`ps`	lists processes currently running, and shows the output in a tree format.
`df`	lists disk space usage stats
`temp`	reports CPU temperature of the PI.

Custom functions/commands:
the "`temp`" command is a custom addition. It reports the cpu temperature of the raspberry pi. It queries the temp by executing:

```
	/opt/vc/bin/vcgencmd measure_temp
```

The output temperature is then parsed, and wrapped with text "CPU temp:", and the temperature component is changed to red in color.

The "`cd`" command has been modified. The new functionality is defined in the `~/.bashrc` file. The standard cd command changes directories. The new cd command changes directories, then lists the contents in the new directory location. Often when changing directories the next command you issue is an "`ls`" command, to list the contents. The new cd function integrates these two steps together and makes navigating the directory tree much easier. The addition to `.bashrc` is:

```
function cd
{
	builtin cd "$@" && ls
}
source /home/pi/.bash_aliases
```

● **char** Exec_Shell_Command(char* command);**

Any command you might normally execute on the command line, you can execute by sending to this function. For instance to list the contents of a file on the command line you might issue:

```
	cat somefile.txt
```

The same functionality can be executed from the C program using Exec_Shell_Command() function. A short code snippet to do the same thing:

```
	char console_command[] = "cat somefile.txt";
	char** console_command_output;
	console_command_output = Exec_Shell_Command( console_command );
	Print_String_Array( console_command_output );
	Free_String_Memory( console_command_output );
```

Notes about usage:
Exec_Shell_Command() returns a list of strings, hence the char** return type. Each string is a line of the output from the command. The list of strings is stored in dynamically allocated memory. As a result, we have to handle the "garbage collection", or letting the OS know when we are done using the memory location. The c function to handle this is free(). Free cannot handle the entire string list though, it can only free one string at a time. It is necessary to loop through and individually free each string. To handle this a helper function was written: Free_String_Memory(), which handles the return of all the dynamic memory allocated. To print any response, Print_String_Array() prints all strings listed in the response of the shell command.

Notes about implementation:
The string representation of the command is passed as an argument to Exec_Shell_Command(). To execute the string, popen() is called. Popen() gives you access to either the read stream, or write stream from the process, but not both. To read the command response, we need the read stream, so the POPEN_READ flag is passed as an argument to popen. popen returns a file handle to the IO stream of the executed command. We can read from the file handle using fgets(). The output is read, one line at a time, and the output is copied to a dynamic string array. New memory locations for strings are allocated only as needed, one line at a time. The

end of the string list is indicated by assigning NULL to the end of the list. The pointer to the string list is then returned. It is the callers responsibility to free the memory locations after use.

Enable_Autologin_For_User(char* USERNAME)

This function enables "Autologin" for a user. In practice this means when you boot the pi, you dont have to enter a username and password before getting access to the console or desktop. To enable autologin, two files are edited:

```
      /etc/lightdm/lightdm.conf                     // for desktop GUI
      /etc/systemd/system/autologin@.service              // for console
```

First these two files are examined to see if autologin has already been configured. The line for each file we are looking for is shown below:

file: `/etc/lightdm/lightdm.conf`
line in file: `autologin-user=pi`

file: `/etc/systemd/system/autologin@.service`
line in file: `ExecStart=-/sbin/agetty -autologin pi -noclear %I $TERM`

The entry "pi" will be changed to the username passed as an argument to Enable_Autologin_For_User(). To do the text find and replacement in the files, the linux stream editor sed is used. sed is a command line stream editor, which uses regular expressions to find and replace text. If the new username is FLIR, the sed commands are:

```
sudo sed -i -e '/autologin-user/s/=[a-zA-Z0-9]*/=FLIR/' /etc/lightdm/lightdm.conf

sudo sed -i -e 's/ExecStart=-\/sbin\/agetty
--autologin[][a-zA-Z0-9]*[]/ExecStart=-\/sbin\/agetty --autologin FLIR /'
/etc/systemd/system/autologin@.service
```

**Note: the above three lines are a single line in the file.

The meat of the second sed command is as follows:
`s/ExecStart=-\/sbin\/agetty -autologin[][a-zA-Z0-9]*[]/ExecStart= ….--autologin FLIR /`

s/ perform substitution. The syntax is s/ [find pattern]/ [replacement text]

Examining the first pattern match portion:
`ExecStart=-\/sbin\/agetty -autologin[][a-zA-Z0-9]*[]/`

Breaking the above line into segments for description:
`ExecStart=-\/sbin\/agetty -autologin[ ][a-zA-Z0-9]*[ ]/`

In english it means:
1. find text beginning with pattern `ExecStart=-\/sbin\/agetty -autologin`
2. after the above text, look for a single character enclosed in the first set of brackets [ ], which is a space
3. next look for alpha-numeric characters. Lowercase a to z, uppercase A to Z, or 0 to 9. There may be one or more of these characters, indicated by the '*'. This is the portion containing the username.
4. look for a single character enclosed in the second set of brackets [ ], which is a space
5. '/'marks the end of the pattern

The above pattern matches all text beginning with Exec…, all the way until the end of pi, and a space after pi. The entire line (highlighted in blue) will be replaced by the replacement text (highlighted in green).

*note the '\' characters pre-pended before any string '/' characters. The forward slash is a command flag indicator to sed, so it must be escaped with a backslash.
** If you look at the c code you'll see any '\' characters pre-pended with an additional '\' character. This is due to C interpreting our escape sequence for sed as an escape sequence within a string. In effect, to enter the '/' character as a string character (and not a command flag), it must be pre-pended with '\\'.

The replacement text:
```
ExecStart=-\/sbin\/agetty --autologin FLIR /
```

void Free_String_Array_Memory(char** string_array)

This function frees dynamically allocated memory stored in string lists, the list is passed as an argument to this function. To free the memory, the list is looped through until encountering a NULL value, identifying the last dynamical memory allocation location. After that, the pointer to the list is freed.

char* Get_Environment_Variable_Value(char* env_variable)

There are two variable types that can be accessed: system Environment Variables, which are available system wide, and user Shell Variables, which are user specific shell variables. This function first queries the system environment variables, and if not found, the user shell variables are queried. If a match is found, the value of the variable is printed. To query the system environment variables, the c function:
getenv(env_variable)
which returns a string of the value of the environment variable. If getenv() doesnt find the environment variable, the we try to querying the shell for the value:
echo [environment variable]

The echo command is executed via a call to Exec_Shell_Command().

void Get_Disk_Usage()

Gets disk space usage stats for the micro SD card, and any attached USB flash drives.
Stats are collected using the "df" shell command . "df" returns more stats than
just the USB flash and micro SD card, it also returns info from many locations in the root filesystem, so response needs to be trimmed. To filter, the output from df is piped to grep,
which only returns lines that contain "/dev/root" (microSD),
or "/dev/sd" (USB flash). Any USB drives have a kernel device location of /dev/sdX, where X is a letter a-z. To search for all USB drives, we just search on the path "/dev/sd".
The full shell command is:
```
df -Tha --total | grep -E "/dev/root|/dev/sd"
```

Flags for df:

-T	print filesystem type
-h	print human readable output
--total	produce grand totals

Flags for grep

-E	use regular expressions in search pattern

Example output:

```
/dev/sda1      vfat        970M   297M  664M   32% /home/pi//USB_DRIVE
```

● void Set_Network_Configuration(char* eth0_IP);

Raspbian by default uses a DHCP Daemon (Dynamic Host Control Protocol) to configure the network interfaces. This is a client daemon that accepts IP addresses from a DHCP server. Typically the PI would connect to a router which hosts a DHCP server, and the server would assign the PI an IP address. For our use we want to set a static IP Address on eth0, and leave the wireless interface (wlan0) able to be configured by dhcp. To do this requires editing the dhcpcd.conf file.

*NOTE in previous versions of Raspbian networking was configured via the /etc/network/interfaces file. This no longer is the case. It is imperative to leave the interfaces file alone.

To check if the DHCP daemon is running enter:

```
sudo service dhcpcd status
```

If the service is not started, start and enable the service with:

```
sudo service dhcpcd start
sudo systemctl enable dhcpcd
```

For manual IP address assignment, the file `/etc/dhcpcd.conf` needs to be edited.
Edit the file, un-commenting the section titled “Example Static IP Configuration”. Delete the ‘#’ at the beginning of the line to uncomment. The section should now look like this:

```
interface eth0
static ip_address=192.168.123.11/24
static routers=192.168.123.1
static domain_name_servers=192.168.123.1
```

The static IP is set on the line:

```
      static ip_address=192.168.123.11/24
```

The suffix ‘/24’ describes the subnet mask as 255.255.255.0. (Use first 24 bits of subnet mask. Each “255” corresponds with 8 bits.)

The fields static_routers and static_domain_name_servers should be set to the hub IP, or router if it is present.
To give the PI the new configuration:

```
sudo reboot
```

The eth0 interface will not actually be set to the static address specified. until there is an Ethernet cable connected between the PI and another network card, so the PI can sense an active connection. Then the interface is configured.

Wifi Configuration
Wifi parameters are set in the file /etc/wpa_supplicant/wpa_supplicant.conf. Only a few lines need to be added, which pertain to any wifi networks you want to join. Also it is important to set the country code in this file. The file should look like:

```
country=us
```

```
update_config=1
ctrl_interface=/var/run/wpa_supplicant

network={
scan_ssid=1
ssid="FLIR-Guest"
psk="6thSense+4BEs"
key_mgmt=WPA-PSK
}
```

Get Environment Variables

This function queries the kernel for the values of some kernel environment variables. These specific variables are used in setting up the Pi. The environment variables retrieved are:

$USER
$HOME
$HOSTNAME

Note- these variables change depending on what user is running the command. For instance if user “pi” is logged in but you ask for the environment variables using “sudo”, the environment variables will return values for the user root.

Get_Environment_Variable_Value

A string of the environment variable name is passed to this function. It then queries the kernel for the variable using the “getenv()” system call. If getenv() cannot find the environment variable name, next the shell is queried. The variable may be a shell variable instead of a system environment variable. The shell is queried via the command:

echo $variable

The command is executed via the EXEC_SHELL() function. The response from the shell is read using the fgets() function.

Verify_SSH_Enabled

To enable SSH and other hardware settings, typically the rapi-config GUI is used. Though not documented, raspi-config also exposes a command line interface. The command line interface consists of a “get” and “set” option for each parameter. To query for SSH status the command is:

```
sudo raspi-config nonint get_ssh
```

The response is:

0 = SSH enabled
1 = SSH disabled.

The raspi-config command is executed in the shell via popen(), and the shell response is read via the fgets() command. The response is parsed to an integer with the strtol() function. (string to long)
Verify_SSH_Enabled() inverts this logic (1= enabled) and returns the value

- ## Set_SSH_Enabled

This also functions via command line interface to raspi-config. To enable SSH the command is:

```
sudo raspi-config nonint do_ssh 0               // 0 enables, 1 disables
```

- ## Verify_Boot_File_Config

The boot file is located here: /boot/config.txt . Don't manually edit this file unless you are knowledgeable about what you're doing. Most of the device enables are in this file. The gpio pins on the PI can be configured for different functions, it is done in this file. Each line configures a different setting. To verify the PI is setup how we want, this function searches the file for the configuration lines we want. Each option we desire is searched for.

- ## List_Users()

To list all users on a linux OS use the command:

```
getent passwd
```

getent is the shell command for "get entries", and is used for mining databases, more than just the password file. Here we are asking for all entries in the password file. There are many users, here is the full output of the command:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/usr/sbin/nologin
systemd-network:x:101:103:systemd Network Management,,,:/run/systemd:/usr/sbin/nologin
systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd:/usr/sbin/nologin
_apt:x:103:65534::/nonexistent:/usr/sbin/nologin
pi:x:1000:1000:,,,:/home/pi:/bin/bash
messagebus:x:104:110::/nonexistent:/usr/sbin/nologin
_rpc:x:105:65534::/run/rpcbind:/usr/sbin/nologin
statd:x:106:65534::/var/lib/nfs:/usr/sbin/nologin
sshd:x:107:65534::/run/sshd:/usr/sbin/nologin
avahi:x:108:113:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/usr/sbin/nologin
lightdm:x:109:114:Light Display Manager:/var/lib/lightdm:/bin/false
```

```
systemd-coredump:x:996:996:systemd Core Dumper:/:/usr/sbin/nologin
```

There are many extraneous users we aren't interested in. To get only the "regular" users, we need to filter this list.

There are two fields we care about for each line entry:

```
    pi:x:1000:1000:,,,:/home/pi:/bin/bash
```

The first highlighted field is the username, the second field is the User ID, or UID.

In the file /etc/login.defs there are parameters that dictate the range of numbers assigned to "regular" user UIDs:

UID_MIN
UID_MAX

The first step in filtering the large list of users is to read the min and max UID parameters. This done by searching the /etc/login.defs file for these two parameters. This done with the command:

```
    grep -E '^UID_MIN|^UID_MAX' /etc/login.defs
```

The command is executed in the shell via Exec_Shell_Command. The return from the shell command is two strings of the form:

```
UID_MIN                  1000
UID_MAX                 60000
```

These two strings are then then parsed to the integer values: 1000 & 60000. These integer values are the filters for parsing the output from the command getent passwd. To parse the long form of the user list, the helper function parseUsers() is invoked.