Lab Assignment 5/6
Lab Write Up
akester
CMPE 12L-01
12-06-13

Introduction/Purpose:
In this lab, we were introduced to the C programming language. We utilized C in order to control hardware external to the RasberryPi.

Procedure:
Part 1:
Toggle an LED using a C program from the RasberryPi. Use the wiringPI library to control the led. Use the C program provided.

Part 2:
Toggle two LED using a modified C program. Similar setup from task 1, but adding an additional LED light and reading from the keyboard. Reads from the terminal without waiting for the return key being pressed. One led lights up if the user pushes a vowel, and the other illuminates if a consonant is pressed. When the program finishes, both LED are off. Neither LED should not be on at the same time.

Part 3:
Read from a switch. Connect a single push button switch to either VDD(+) or VSS(-), read, with the 'gpio read' command, either a zero (vss) or a one (vdd) at the command prompt. Connect positive (3.3v) and negative (0v) wires to the breadboard from the ribbon cable attached to the pi. Wire either positive 3.3v or ground to the input of the push button. Connect the output of the button back to a suitable pin on the ribbon cable so that the 'gpio read' command will register either a 0 or a 1 when the button is pressed. Manually read the value of the gpio pin, pressing the button changes the value of the pin (by calling the command at the command line), create a C program that will constantly print out the value of the pin.

Part 4:
Simon says with keyboard (no switches). In this part, connect 4 LED. To control what the simon says, the user can type the characters (a, s, d, e) in the keyboard that correspond to (red, green, blue, white) in the breadboard. Write a C program that starts showing a single color, then the user has to type the correct character. Otherwise all the colors blink and the game starts again. If the user is correct, then it shows that color again and some other random color. The user keeps playing. The simon says game should be able to memory 100 patterns.

Part 5:
Simon says with keyboard and/or switches. Same as task 4 but the game can be controlled either with the keyboard or with the switches. Map the keys in the same order as

the LED are shown in the breadboard (red, green, blue, white). Use small functions. There should be less than 10 statements per function code. This constraint is to force you to structure your code and to have several functions.

Algorithm/Other Data:

Part 1:

```
#include <stdio.h>

#include <wiringPi.h>

int main()

{

printf("Raspberry Pi blinkÄn");

int pin = 7;

wiringPiSetup();

pinMode(pin, OUTPUT) ;

int i;

for (i=0;i<10;i++) {

printf("led onÄn");

digitalWrite(pin, HIGH);

sleep(1);

printf("led offÄn");

digitalWrite(pin, LOW);

sleep(1);

}

return 0;

}
```

Part 2:

```
#include <stdio.h>
```

```
#include <stdlib.h>

#include <termios.h>

struct termios initial_settings,new_settings;

int main(int argc, char **argv) {

tcgetattr(0,&initial_settings);

new_settings = initial_settings;

new_settings.c_lflag &= ~ICANON;

new_settings.c_lflag &= ~ECHO;

new_settings.c_lflag &= ~ISIG;

new_settings.c_cc[VMIN] = 0;

new_settings.c_cc[VTIME] = 0;

tcsetattr(0, TCSANOW, &new_settings);

while(1) {

int n = getchar();

if(n != EOF) {

int key = n;

if(key == 27 II key == 10) {

break;

}

printf("[%c:%d]Än",key,key);

}

}

tcsetattr(0, TCSANOW, &initial_settings);

return 0;

}
```

Part 3:

```
#include <stdio.h>

#include <stdlib.h>

#include <termios.h>

#include <wiringPi.h>


struct termios initial_settings,new_settings;

int main(int argc, char **argv) {


        tcgetattr(0,&initial_settings);

        new_settings = initial_settings;

        new_settings.c_lflag &= ~ICANON;

        new_settings.c_lflag &= ~ECHO;

        new_settings.c_lflag &= ~ISIG;

        new_settings.c_cc[VMIN] = 0;

        new_settings.c_cc[VTIME] = 0;


        tcsetattr(0, TCSANOW, &new_settings);


        int pin = 2;

        wiringPiSetup();

        pinMode(pin, INPUT);

        int button;


        while(1) {
```

```
        int n;

        button = digitalRead(pin);

        printf("%d\n", button);

        sleep(1);

        if((n = getchar()) != EOF) {

            int key = n;

            if(key == 27 || key == 10) {

                break;

            }


            }

    }



    tcsetattr(0, TCSANOW, &initial_settings);

    return 0;

}
```

Part 4&5:

```
#include <stdio.h>

#include <stdlib.h>

#include <termios.h>

#include <wiringPi.h>


struct termios initial_settings,new_settings;
```

```
int newGame[100]; //global array of colors

int counter = 0;  //global counter of array items


  int pinRed   = 7;

  int pinGreen = 0;

  int pinBlue  = 4;

  int pinWhite = 5; //hard coded pin values


void blinkLights(int pinRed, int pinGreen, int pinBlue, int pinWhite){

int i;

for(i = 0; i < 4; i++){

  digitalWrite(pinRed, HIGH);

  digitalWrite(pinGreen, HIGH);

  digitalWrite(pinBlue, HIGH);

  digitalWrite(pinWhite, HIGH);

  delay(100);

  digitalWrite(pinRed, LOW);

  digitalWrite(pinGreen, LOW);

  digitalWrite(pinBlue, LOW);

  digitalWrite(pinWhite, LOW);

  delay(100);

 }

}


void startOver(){
  blinkLights(pinRed, pinGreen, pinBlue, pinWhite);
  int i = 0;
  for(i; i<counter; i++){
    newGame[i] = 0;
  }
  counter = 0;
}

void setPins(int pinRed, int pinGreen, int pinBlue, int pinWhite){
  pinMode(pinRed, OUTPUT);
  pinMode(pinGreen, OUTPUT);
  pinMode(pinBlue, OUTPUT);
  pinMode(pinWhite, OUTPUT);
}

int randomNumber(){
  int pin;
  int r = rand()%4;
  if(r == 0) {
   pin = 7;
  }else {
   if(r == 1) {
   pin = 0;
   }else {
   if(r == 2) {
```

```
   pin = 4;
   }else {
   if(r == 3) {
   pin = 5;
   }}}}
  return pin;
}

void showNewColor(){
  int pin = randomNumber();
  digitalWrite(pin, HIGH);
  delay(1000);
  digitalWrite(pin, LOW);
  delay(1000);
  newGame[counter] = pin;
  counter++;
}

void repeatColors(){
  int i = 0;
  for(i; i<counter; i++){
    digitalWrite(newGame[i], HIGH);
    delay(1000);
    digitalWrite(newGame[i], LOW);
    delay(1000);
  }
  showNewColor();
}

void declareTermiosCrap(){
           tcgetattr(0,&initial_settings);
           new_settings = initial_settings;
           new_settings.c_lflag &= ~ICANON;
           new_settings.c_lflag &= ~ECHO;
           new_settings.c_lflag &= ~ISIG;
           new_settings.c_cc[VMIN] = 0;
           new_settings.c_cc[VTIME] = 0;
           tcsetattr(0, TCSANOW, &new_settings);
}

int keyboard(){
  while(1) {
      int n;
      if((n = getchar()) != EOF) {
        int key = n;
        if(key == 27 || key == 10) {
          exit(EXIT_SUCCESS);
                               }
                               if(key == 97 || key == 115 || key == 100 || key == 101){
         return key;
```

```
                                }else{
        printf("Please use 'a' 's' 'd' 'e' as inputs\n");
                                }
                        }
                }
        }

void verify(){
  int guess[100];
  int key;
  int i = 0;
  while(i<counter){
    key = keyboard();
    key = convert(key);
    guess[i] = key;
    i++;
  }
  int j = 0;
  while(j<counter){
    if(newGame[j] == guess[j]){
      j++;
      printf("correct\n");
    }else{
      printf("wrong\n");
      startOver();
      repeatColors();

      verify();

   }

  }

  printf("go again\n");

   repeatColors();

   verify();

}



int convert(int key){

  if(key == 97)

    return 7;

  if(key == 115)

    return 0;

  if(key == 100)

    return 4;

  if(key == 101)

    return 5;

}



int main(int argc, char **argv) {


  declareTermiosCrap();
```

```
 wiringPiSetup();

 setPins(pinRed, pinGreen, pinBlue, pinWhite);

 blinkLights(pinRed, pinGreen, pinBlue, pinWhite);

 repeatColors();

 verify();




        tcsetattr(0, TCSANOW, &initial_settings);

        return 0;

}
```

Other Information:
• What does sleep() do? What units of time does it use?

Sets a delay of one second after digitalWrite, you can also use delay(1000).

• What does EOF mean?

Technically, it means end of file. In this case we are using it as a signal to exit the program.

• What do each of the 3 #include do? Specify what including them allows you to do.

They are libraries. They come with standard functions. The <termios.h> header contains the definitions used by the terminal I/O interfaces, stdlib has a bunch of stuff in that we basically can't live without (standard library definitions) stdio is needed for standard buffered input/output etc..etc...

• Describe what each of the functions do and terms mean in task 2 (i.e: tcsetattr, TCSANOW, etc.)

Input modes, output modes control modes - Change attributes immediately. You can use these functions to do things like turn off input echoing; set serial line characteristics such as line speed and flow control; and change which characters are used for end-of-file, command-line editing, sending signals, and similar control functions.

• What is a struct in C? What is it used for? Give an example using pseudo-code.

It’s like an object.

```
// NodeObj

typedef struct NodeObj{

   char* key;

   char* value;

   struct NodeObj* next;

} NodeObj;
```

• What is \n used for? When would you use it when programming?

End of line. Usually when printing to stdout.

• What is the difference between using brackets <> and quotations “” when #include-ing? Explain the difference in file type that would require brackets or quotations

GCC looks for headers requested with #include "file" first in the directory containing the current file, then in the directories as specified by -iquote options, then in the same places it would have looked for a header requested with angle brackets. For example, if /usr/include/sys/stat.h contains #include "types.h", GCC looks for types.h first in /usr/include/sys, then in its usual search path.

• Why do you return 0 at the end of main()?

Because it’s good practice to let the OS know everything was successful. You don’t have to return 0.

• How many times will the for loop in task 1 run? Give an exact number.

Runs 10 times.

• What different pins could be used as inputs for task 3, besides the one you chose?

Any of them. I set the entire thing up using pins 7, 0, 2, 3, 12, 13, 4, 5. I could still use 14, 15, 16, 1, 6, 10, 11.

• Part 1: Answer a question regarding errors that would occur if you didn’t include the ‘.h’ files.

Because I do this a lot, I can tell you the compiler yells are about making implicit declarations of functions - because it doesn’t know what the hell you’re trying to do.

Conclusion:
This lab was very fun! The tasks built up, starting with the very basic task of using a given c program to light up an LED. Then we were able to modify a given c program to accept user input from the keyboard. Then we got to write our own c program to accept input from the switches. This was very easy to implement using the code samples from the previous two tasks. In tasks 4 and 5 we got to put everything together in a game. My code for tasks 4 and 5 were fairly similar. I refined my code slightly for task 5, but mostly it was just adding a function and call for the switches - which I had already pretty much done in task 2. Setting up the breadboard with LED's and switches was easy given the helpful links that were provided. I just set up the entire circuit for task 5 and then implemented just what was needed for particular tasks. It made is super helpful when testing my functions knowing that all my input/outputs were set up correctly already and tested. This lab took me a decent amount of time, although knowing C, at least on an intermediate level made it very easy. I didn't encounter any difficulties.

Thanks a bunch! Have a fabulous Winter break.