Patricia Gavelek, Anders Dahl EECE2160	Embedded Design: Enabling Robotics Lab 9

Assignment 1:

The first line of the shell script shortens ‘/sys/class/gpio’ to ‘FILEPATH,’ for convenience’s sake. This is the virtual file in which pins are reserved. ‘10 > $FILEPATH/unexport 2>/dev/null’ clears if there is any file at the given location. The 11 > $FILEPATH/export lines allocate GPIO ports, reserving the pins of the values given, and creating a new virtual directory called ‘gpio#’ that has all the files to configure the port. ‘out > $FILEPATH/gpio13/direction’ designates it as an output port.

Assignment 2 (relevant code):

```
GCC = g++
CFLAGS = -g -Wall
OBJS = GPIO.o main.o
EXE = GPIO

$(EXE): $(OBJS)
	$(GCC) $(OBJS) -o $(EXE)

GPIO.o: GPIO.cc GPIO.h
	$(GCC) $(CFLAGS) -c GPIO.cc

main.o: main.cc GPIO.h
	$(GCC) $(CFLAGS) -c main.cc

clean:
	rm $(OBJS) $(EXE)
```

Assignment 3 (relevant code):

```
GCC = g++
CFLAGS = -g -Wall
OBJS = GPIO.o ServoPosition.o
EXE = GPIO

$(EXE): $(OBJS)
	$(GCC) $(OBJS) -o $(EXE)

GPIO.o: GPIO.cc GPIO.h
	$(GCC) $(CFLAGS) -c GPIO.cc

ServoPosition.o: ServoPosition.cc GPIO.h
	$(GCC) $(CFLAGS) -c ServoPosition.cc

clean:
	rm $(OBJS) $(EXE)
```

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

#include "GPIO.h"

/**
 * Print the menu and get a selection from the user.
 *
 * @return Number of selection.
 */
int PrintMenu()
{
    int sel;

    printf("Main menu:\n\n" );
    printf("1. Move the base\n" );
    printf("2. Move the bicep\n" );
    printf("3. Move the elbow\n" );
    printf("4. Move the wrist\n" );
    printf("5. Move the gripper\n" );
    printf("6. Exit \n" );
    printf("Select an option: " );

    scanf("%d", &sel); // Scan a digit from the user

    return sel; // Return the chosen digit
}
```

```
/**
 * Calculate the on period of a PWM signal for controlling a servomotor. The C+
 * function receives the servo position (0 to 180 degrees) and returns the time
 * in micro seconds that PWM signal should be on during each period so that the
 * RC servo moves to the specified servo position.
 *
 * @param position, an integer representing the servo position
 * @return integer, time in micro seconds
 */
int degreeToOnDelay(int position) {
    if (position > 160 || position < 20) {
        printf("Position not inclusively between 20 and 160.\n");
        exit(0);
    }

    return position * 10 + 600;

}
```

```
/**
 * Run the navigational loop
 *
 * @return Number on exit. 0 for no errors.
 */
int Run()
{
    int sel;
    int angle;

    // While true
    while(true) {

        // Print the menu and get a selection
        sel = PrintMenu();

    printf("Enter an angle between 20 and 160 degrees: ");
    scanf("%d", &angle); // Scan a digit from the user

    // Open device file 13 on Linux file system
    GPIO gpio_base(13);

    // Open device file 10 on Linux file system
    GPIO gpio_bicep(10);

    // Open device file 11 on Linux file system
    GPIO gpio_elbow(11);

    // Open device file 12 on Linux file system
    GPIO gpio_wrist(12);

    // Open device file 0 on Linux file system
    GPIO gpio_gripper(0);
```

```
        // Next step depends on the selection made
        switch(sel) {

            // User chose 1
            case 1:
                printf("You selected \"Move the base\"\n\n");

        // Generate 400 periods, this will take 20ms * 400 iterations = 8s
        gpio_base.GeneratePWM(20000, degreeToOnDelay(angle), 400);

                break;

            // User chose 2
            case 2:
                printf("You selected \"Move the bicep\"\n\n");

        // Generate PWM signal with 20ms period and 1.5ms on time.
        gpio_bicep.GeneratePWM(20000, degreeToOnDelay(angle), 400);

                break;

            // User chose 3
            case 3:
                printf("You selected \"Move the elbow\"\n\n");

        // Generate PWM signal with 20ms period and 1.5ms on time.
        gpio_elbow.GeneratePWM(20000, degreeToOnDelay(angle), 400);

                break;

            // User chose 4
            case 4:
                printf("You selected \"Move the wrist\"\n\n");

        // Generate 400 periods, this will take 20ms * 400 iterations = 8s
        gpio_wrist.GeneratePWM(20000, degreeToOnDelay(angle), 400);

                break;
```

```
            // User chose 5
            case 5:
                printf("You selected \"Move the gripper\"\n\n");

    //  Generate 400 periods, this will take 20ms * 400 iterations = 8s
    //  gpio_gripper.GeneratePWM(20000, degreeToOnDelay(angle), 400);

                break;


            // User chose 6
            case 6:
                printf("You selected \"Exit\"\n\n");

                // Return here, with no erros, to exit the function.
                // Clean up will be next
                return 0;

            // User chose soomething not on the menu
            default:
                printf("Please enter a valid number from the menu!\n\n");
                break;
        }

    printf("--------------------\n");

    }
}
```

```
int main()
{
    Run();
}
```

Assignment 4 (relevant code):

```
void GPIO::GenerateVariablePWM(int period, int first_pulse, int last_pulse, int num_periods)
{

    for (int i = 0; i < 50; i++) {
        write(fd, "1", 1);
        usleep(first_pulse);
        write(fd, "0", 1);
        usleep(period - first_pulse);
    }

    num_periods -= 100;
    int pulse = first_pulse;
    int difference = abs(first_pulse - last_pulse) / num_periods;


    for (int i = 0; i < num_periods; i++) {
        write(fd, "1", 1);
        usleep(pulse);
        write(fd, "0", 1);
        usleep(period - pulse);
        if (first_pulse < last_pulse)
            pulse += difference;
        else
            pulse -= difference;
    }


    for (int i = 0; i < 50; i++) {
        write(fd, "1", 1);
        usleep(last_pulse);
        write(fd, "0", 1);
        usleep(period - last_pulse);
    }
}
```

```
GCC = g++
CFLAGS = -g -Wall
OBJS = GPIO.o ServoSpeed.o
EXE = GPIO

$(EXE): $(OBJS)
	$(GCC) $(OBJS) -o $(EXE)

GPIO.o: GPIO.cc GPIO.h
	$(GCC) $(CFLAGS) -c GPIO.cc

ServoSpeed.o: ServoSpeed.cc GPIO.h
	$(GCC) $(CFLAGS) -c ServoSpeed.cc

clean:
	rm $(OBJS) $(EXE)
```

```
/**
 * Caculate the number of periods for more consistent, slow movement.
 *
 * @param first_angle, an integer representing the first angle
 * @param second_angle, an integer representing the second angle
 * @param speed, an integer representing the speed
 * @return integer, the number of periods
 */
int calculateNumPeriods (int first_angle,
    int second_angle,
    int speed) {

    return 100 + (abs(first_angle - second_angle) / speed) * 50;
}
```

```
    int sel;
    int first_angle;
    int second_angle;
    int speed;

    // While true
    while(true) {

        // Print the menu and get a selection
        sel = PrintMenu();

        printf("Enter the first angle between 20 and 160 degrees: ");
        scanf("%d", &first_angle); // Scan a digit from the user

        printf("Enter the second angle between 20 and 160 degrees: ");
        scanf("%d", &second_angle); // Scan a digit from the user

        printf("Enter the speed: ");
        scanf("%d", &speed); // Scan a digit from the user
```

```
// User chose 1
case 1:
    printf("You selected \"Move the base\"\n\n");

    // Generate 400 periods, this will take 20ms * 400 iterations = 8s
    gpio_base.GenerateVariablePWM(20000,
            degreeToOnDelay(first_angle),
            degreeToOnDelay(second_angle),
            calculateNumPeriods(first_angle, second_angle, speed));


    break;

    // User chose 2
case 2:
    printf("You selected \"Move the bicep\"\n\n");

    // Generate PWM signal with 20ms period and 1.5ms on time.
    gpio_bicep.GenerateVariablePWM(20000,
            degreeToOnDelay(first_angle),
            degreeToOnDelay(second_angle),
            calculateNumPeriods(first_angle, second_angle, speed));

    break;

    // User chose 3
case 3:
    printf("You selected \"Move the elbow\"\n\n");

    // Generate PWM signal with 20ms period and 1.5ms on time.
    gpio_elbow.GenerateVariablePWM(20000,
            degreeToOnDelay(first_angle),
            degreeToOnDelay(second_angle),
            calculateNumPeriods(first_angle, second_angle, speed));

    break;
```

```
    // User chose 4
case 4:
    printf("You selected \"Move the wrist\"\n\n");

    // Generate 400 periods, this will take 20ms * 400 iterations = 8s
    gpio_wrist.GenerateVariablePWM(20000,
            degreeToOnDelay(first_angle),
            degreeToOnDelay(second_angle),
            calculateNumPeriods(first_angle, second_angle, speed));

    break;

    // User chose 5
case 5:
    printf("You selected \"Move the gripper\"\n\n");

    // Generate 400 periods, this will take 20ms * 400 iterations = 8s
    gpio_gripper.GenerateVariablePWM(20000,
            degreeToOnDelay(first_angle),
            degreeToOnDelay(second_angle),
            calculateNumPeriods(first_angle, second_angle, speed));

    break;


    // User chose 6
case 6:
    printf("You selected \"Exit\"\n\n");

    // Return here, with no erros, to exit the function.
    // Clean up will be next
    return 0;

    // User chose soomething not on the menu
default:
    printf("Please enter a valid number from the menu!\n\n");
    break;
```

Extra (relevant code):

```
GCC = g++
CFLAGS = -g -Wall
OBJS = GPIO.o MultiSpeed.o
EXE = GPIO

$(EXE): $(OBJS)
	$(GCC) $(OBJS) -o $(EXE) -pthread -std=c++0x

GPIO.o: GPIO.cc GPIO.h
	$(GCC) $(CFLAGS) -c GPIO.cc

MultiSpeed.o: MultiSpeed.cc GPIO.h
	$(GCC) $(CFLAGS) -c MultiSpeed.cc -pthread -std=c++0x

clean:
	rm $(OBJS) $(EXE)
```

Note the changes here in order to use threads.

```
void runSelection(int sel, int first_angle, int second_angle, int speed,
	GPIO * gpio_base, GPIO * gpio_bicep, GPIO * gpio_elbow,
	GPIO * gpio_wrist, GPIO * gpio_gripper) {

		// Next step depends on the selection made
		switch(sel) {
```

Put the switch statement into a function.

```
/**
 * Run the navigational loop
 *
 * @return Number on exit. 0 for no errors.
 */
void Run()
{
    int sel1;
    int sel2;

    int first_angle_first_move;
    int second_angle_first_move;
    int speed_first_move;

    int first_angle_second_move;
    int second_angle_second_move;
    int speed_second_move;

    // While true
    while(true) {

        // Print the menu and get a selection
        sel1 = PrintMenu();

        printf("Enter the first angle between 20 and 160 degrees for the first move: ");
        scanf("%d", &first_angle_first_move); // Scan a digit from the user
        printf("Enter the second angle between 20 and 160 degrees for the first move: ");
        scanf("%d", &second_angle_first_move); // Scan a digit from the user
        printf("Enter the speed for the first move: ");
        scanf("%d", &speed_first_move); // Scan a digit from the user
```

```
    // Print the menu and get a selection
    sel2 = PrintMenu();

    printf("Enter the first angle between 20 and 160 degrees for the second move: ");
    scanf("%d", &first_angle_second_move); // Scan a digit from the user
    printf("Enter the second angle between 20 and 160 degrees for the second move: ");
    scanf("%d", &second_angle_second_move); // Scan a digit from the user
    printf("Enter the speed for the second move: ");
    scanf("%d", &speed_second_move); // Scan a digit from the user

    // Open device file 13 on Linux file system
    GPIO gpio_base(13);

    // Open device file 10 on Linux file system
    GPIO gpio_bicep(10);

    // Open device file 11 on Linux file system
    GPIO gpio_elbow(11);

    // Open device file 12 on Linux file system
    GPIO gpio_wrist(12);

    // Open device file 0 on Linux file system
    GPIO gpio_gripper(0);

    if (sel1 == sel2) {
        printf("Cannot run the same servo at the same time!.\n");
        exit(0);
    }
```

```
    std::thread first (runSelection, sel1, first_angle_first_move, second_angle_first_move, speed_first_move,
            &gpio_base, &gpio_bicep, &gpio_elbow,
            &gpio_wrist, &gpio_gripper);

    std::thread second (runSelection, sel2, first_angle_second_move, second_angle_second_move, speed_second_move,
            &gpio_base, &gpio_bicep, &gpio_elbow,
            &gpio_wrist, &gpio_gripper);

    // synchronize threads:
    first.join();               // pauses until first finishes
    second.join();              // pauses until second finishes

    printf("--------------------\n");

  }
}
```

And finally the threads.