TBT - 2: Python Script Documentation

e-Yantra Summer Internship

Vinay Manjunath

Contents

Function 1

```
path_to_mocked_delay_function_header = "../../../../../CMock/lib/mocks/Mock_ms_delay.h
    "

path_to_mocked_delay_function_c = "../../../../CMock/lib/mocks/Mock_ms_delay.c"

path_to_untiy = "../../../../Unity-master/src/unity.c"

path_to_avr_io = "/usr/lib/avr/include/avr/"
```

In each Script provided in the Test Folders, these four lines are essential for the scripts to run properly. As evident by their names they are paths to different files. These must be configured as per your system before the script is run. Please ensure you edit them accordingly.

Function 2

```
list_of_files = sorted(glob.glob('./Submissions/*.c'))

def open_submissions():
     i = 0
     for file_name in list_of_files:
          os.system("geany " + list_of_files[i])
          i += 1
```

The array list of files is an array of all of the submissions present in the Submissions folder. This array will be referred to multiple times throughout the document. The function "open submissions" is used to open all of the submission in the Geany IDE. The os.system function is used to run commands in the terminal. The string "geany" in the argument passed to os.system is used to open Geany along with any specified file.

Function 3

```
def Find_Team_ID():
    global totalmarks
    k = 0
    for file_name in list_of_files:
        temp = open( './Grades/Submission' +str(k+1)+ '.csv' ,'w')
        with open(file_name, 'r') as file1:
            for line in file1:
             if re.search("ID.",line):
                 temp.write("Team ID," + line)
            k +=1
```

The Find Team ID function is used to parse through the submissions one by one and identifies each team's ID. The team's ID is then stored in the respective Excel files in which the Teams results will be stored after the test is conducted.

Function 4

```
def check_CPU_frequency():
    global totalmarks
    i = 0
    for file_name in list_of_files:
        temp = open( './Grades/Submission' +str(i+1)+ '.csv' ,'a')
        with open(file_name, 'r') as file1:
            for line in file1:
             if '#define F_CPU' in line:
                 if '14745600' in line:
                     temp.write('1. Correct CPU Frequency, 0 Mark')
                     totalmarks += 1
                     temp.write('\n')
                     temp.close
                 else:
                     temp.write('1. Incorrect CPU Frequency, 0 Mark')
                     temp.write('\n')
                     error = 1.0
                     temp.close
            i += 1
```

The Check CPU Function is used to check whether the submitted program has correctly initialized the CPU frequency. The result of the check is stored in respective team's Grade file (Mentioned in Part 3).

Function 5

```
def copy():
     i = 0
     list_of_temp = sorted(glob.glob('./Temp/*.c'))
     for file_name in list_of_files:
          file_in = open(list_of_files[i],"w")
          file_out = open(list_of_temp[i],"r")
          for line in file_out:
               file_in.write(line);
          i +=1
          file_in.close()
          file_out.close()
```

To render a submission testable certain changes have to be made to the program. These changes are done within the Python script, hence the changes cannot be made and written back to the original file in a single step. The changes must be copied to a temporary file and then the contents of the temporary file must be copied back to the original submission. This function is used to copy the contents of the temporary files back to the original submissions.

Function 6

```
def copy_makefile():
     file_in = open("Makefile","w")
     file_out = open("Makefile1","r")
     for line in file_out:
          file_in.write(line);
     file_in.close()
     file_out.close()
```

The Copy Makefile function provides the same functionality as the Copy function, but is used to transfer the generated commands from a temporary file to the tests Makefile.

Function 7

```
def remove_infinite_loops():
     i = 0
     os.system("mkdir Temp")
     list_of_temp = sorted(glob.glob('./Temp/*.c'))
     for file_name in list_of_files:
          delete = ["while(1)"]
          file_in = open(list_of_files[i])
          file_out = open( './Temp/Temp' +str(i+1)+ '.c' ,'w')
          for line in file_in:
              for word in delete:
               line = line.replace(word, "for (int qwer = 1; qwer > 0; qwer-- );")
              file_out.write(line)
```

Function 7 continued on next page...

```
            i +=1
            file_in.close()
            file_out.close()
```

The Remove Infinite Loops Function goes through all the submissions and all infinite loops are replaced by for loops. The inserted for loop can be edited, so that it runs as many times as required. Note that the changes are not stored inside the original submission but in a temporary file. The changes must be copied over using the Copy function that was mentioned earlier.

Function 8

```
def rename_main():
     i = 0
     for file_name in list_of_files:
          file2 = open("./Temp/Temp" +str(i)+ ".c",'w+')
          with open(list_of_files[i], 'r') as file1:
              for line in file1:
               if re.search("int main()", line):
                    file2.write("int main_test()\n")
               else:
                    file2.write(line)
          file2.close()
          i += 1
```

The Rename Main Function goes through the submissions and finds the Main function within the submission and renames it. The Main function of the submissions must be renamed so that it can be called in the Unit Test. Note that the changes are not stored inside the original submission but in a temporary file. The changes must be copied over using the Copy function that was mentioned earlier.

Function 9

```
def create_Makefile():
     i = 0
     os.system("mkdir Test_Results")
     file2 = open("Makefile1", "w")
     with open("Makefile", "r+") as file1:
          for line in file1:
               if re.search("Compile:", line):
                    file2.write(line)
                    for file_name in list_of_files:
                        file2.write("  gcc -Os -DTEST -std=c99 " + list_of_files[i] +
                             " Experiment_1_test.c " + path_to_mocked_delay_function_c
                            + " " +path_to_untiy+ " -o ./Test" +str(i)+ ".elf" +"\n")
                        i += 1
               else:
                        file2.write(line)
     file2.close()
     file1.close()
```

```
def create_Makefile_run():
     i = 0
     file2 = open("Makefile1", "w")
     with open("Makefile", "r+") as file1:
          for line in file1:
               if re.search("Test:", line):
                    file2.write(line)
                    for file_name in list_of_files:
                        file2.write("  ./Test" + str(i)+ ".elf" + " > ./Test_Results/
                            Submission" +str(i+1)+ ".txt\n" )
                        i += 1
               else:
                        file2.write(line)
     file2.close()
     file1.close()
```

The two functions above have been clubbed together into one part, since they both involve populating the Makefile with commands which,

- Compile the submissions individually.
- Run the Unit Test for each submission.
- Stores the results of the Unit Test into a Results file.

The Create Makefile Function creates a temporary file in which it populates the commands to compile the submissions present in the submissions folder. Note in order to transfer the commands that have been populated into the temporary folder, the user must call the Copy Makefile function.

The Create Makefile Run Function populates the temporary file with the required commands to run the compiled tests and store the results into separate result files for each submission. Note in order to transfer the commands that have been populated into the temporary folder, the user must call the Copy Makefile function.

Function 10

```
def add_required_syntax():
     i = 0
     list_of_temp = sorted(glob.glob('./Temp/*.c'))
     for file_name in list_of_files:
          file2 = open(list_of_temp[i],'w+')
          with open(list_of_files[i], 'r') as file1:
              for line in file1:
               if re.search("PORTJ =", line):
                    file2.write(line);
                    file2.write("value[a] = PORTJ;\na++;\n")
               elif re.search("#include <avr/interrupt.h>", line):
                    file2.write("#include <"+ path_to_avr_io +"interrupt.h>\n")
               elif re.search("#include <avr/io.h>", line):
                    file2.write("#include <"+ path_to_avr_io +"io.h>\n")
```

Function 10 continued on next page...

```
                elif re.search("#include <util/delay.h>", line):
                    file2.write("\n#include \""+path_to_mocked_delay_function_header+"
                        \"")
                elif re.search("#define F_CPU 14745600", line):
                    file2.write(line);
                    file2.write("int value[5];\n int a = 0;")
                else:
                    file2.write(line)
            file2.close()
            i += 1
```

In order to test a submission, the submission should include certain variables, functions or headers that allow it to be tested. Generally a submitted program is not in this format. To be able to test these submissions, they must be altered to suit the Unit Test that is written. Please open the Unit Test for this TBT experiment while reading the explanation.

In Experiment-1 of TBT-2 compilation of the submissions is not accomplished with AVR-GCC but instead using vanilla GCC. Due to this, the paths for the IO, Interrupt, Delay Headers must be specified when they are included. To determine whether the Bar LEDs have been turned on and off in the correct sequence, we require an array to store each change to the respective port that they are connected to. The Bar LEDs are connected to PORTJ, hence every time PORTJ is altered we must record the alteration. Thus we parse the original program and include these changes. In the above snippet of code, we can see that we initialize a an array at the beginning of the program with a length of 5. Although the BAR LEDs must undergo 4 changes in the Main function to satisfy the given task, the LEDs must also be initialized before they are written too. This initialization consumes one element of the array, and hence the array is initialized with a length of 5. while the code is being parsed we also add the required paths, and store any change to PORTJ into the array that we initialized.

Function 11

```
def assign_grades():
    i = 0
    k = 1
    global totalmarks
    for file_name in list_of_files:
        totalmarks = 0
        k = 1
        file2 = open('./Grades/Submission' + str(i+1) + '.csv' ,'a')
        with open('./Test_Results/Submission' + str(i+1) + '.txt', 'r') as file1:
            flag = 0
            for line in file1:
                #print 'k'
                if re.search('test_main_1',line):
                    if (bool(re.search('PASS',line))):
                        k+=1
                        file2.write('%d. Toggling of LED, 2.5 Marks\n' % k)
                        k+=1
                        file2.write('%d. Delay, 2.5 Marks\n' % k)
                        file2.write('%d. Top 4 Leds turn on, 2.5 Marks\n' % k)
                        k+=1
```

Function 11 continued on next page...

```
                    file2.write('%d. Bottom 4 Leds turn on, 2.5 Marks\n' % k
                        )
                    totalmarks += 10
                    flag = 1
               elif re.search('test_main_2',line):
                    if (bool(re.search('PASS',line))):
                    k+=1
                    file2.write('%d. Toggling of LED, 2.5 Marks\n' % k)
                    k+=1
                    file2.write('%d. Delay, 2.5 Marks\n' % k)
                    k+=1
                    file2.write('%d. Top 4 Leds turn on, 2.5 Marks\n' % k)
                    k+=1
                    file2.write('%d. Bottom 4 Leds turn on, 2.5 Marks\n' % k
                        )
                    totalmarks += 10
                    flag = 1
               elif re.search('test_main_3',line):
                    if (bool(re.search('PASS',line))):
                    k+=1
                    file2.write('%d. Toggling of LED, 2.5 Marks\n' % k)
                    k+=1
                    file2.write('%d. Delay, 2.5 Marks\n' % k)
                    k+=1
                    file2.write('%d. Top 4 Leds turn on, 2.5 Marks\n' % k)
                    k+=1
                    file2.write('%d. Bottom 4 Leds turn on, 2.5 Marks\n' % k
                        )
                    totalmarks += 10
                    flag = 1
     with open('./Test_Results/Submission' + str(i+1) + '.txt', 'r') as file1:
          if flag == 0:
               flag = 1
               k+=1
               file2.write('%d. Toggling of LED, 0 Marks\n' % k)
               k+=1
               file2.write('%d. Delay, 0 Marks\n' % k)
               for line in file1:

                    if (bool(re.search('test_led_on_topfour',line))):
                         if (bool(re.search('PASS',line))):
                              k+=1
                              file2.write('%d. Top four LEDs Turn on, 2.5 Marks\n
                                  ' % k)
                              totalmarks += 2.5
                         else:
                              k+=1
                              file2.write('%d. Top four LEDs Turn on, 0 Marks\n'
                                  % k)
                    if (bool(re.search('test_led_on_bottomfour',line))):
                         if (bool(re.search('PASS',line))):
                              k+=1
```

```
                                    file2.write('%d. Bottom four LEDs Turn on, 2.5
                                        Marks\n' % k)
                                    totalmarks += 2.5
                                else:
                                    k+=1
                                    file2.write('%d. Bottom four LEDs Turn on, 0 Marks\
                                        n' % k)

            file2.write('Total Marks, %d Marks'%totalmarks )
            i += 1
            file2.close()
```

The Assign Grades Function parses through the files containing the results of the Unit Tests and searches for whether the functions have passed the tests or not. Based on whether they pass or fail, the function assigns the marks obtained for each function. These marks are stored within the respective Excel files that already contain information about the Team ID and the CPU frequency. The marks in case of this experiment were allotted with reference to the TBT-2 PDF.

Function 12

```
def add_team_id_to_results():
    k = 0
    list_of_txt = sorted(glob.glob('./Test_Results/*.txt'))
    for file_name in list_of_txt:
        with open( list_of_files[k] ,'r') as file1:
            for line in file1:
                if re.search("TeamID",line):
                    with file(list_of_txt[k], 'r') as original: data = original.
                        read()
                    with file(list_of_txt[k], 'w') as modified: modified.write(
                        line.replace("/","") + data)
```

The Add Team ID To Results Function adds the Team ID to the respective Unit Test result file. This allows the user to manually search a team's results in case there are any discrepancies.

Function 13

```
def concatenate_results():
    k = 0
    list_of_txt = sorted(glob.glob('./Test_Results/*.txt'))
    file1 = open("./Test_Results/Results.txt","w")
    file1.close()
    for file_name in list_of_txt:
        with file(list_of_txt[k], 'r') as original: data = original.read()
        with file("./Test_Results/Results.txt", 'a') as modified: modified.write(
            data + "\n")
        k +=1
```

Function 13 continued on next page...

The Concatenate Results Function, concatenates all the results into a single file so that a user may browse all the results in one place.

Function 14

```
def run_test():
    os.system("make")

def clean():
    os.system("make clean")

def open_grades():
    os.system("libreoffice Grades.csv | gedit ./Test_Results/Results.txt")
```

The above Functions are straightforward. They run Bash commands that are used to run the Makefile, remove the unwanted temporary files after the test, and to open the final Excel sheet containing the grades of all the submissions along with all the Unit Test results.