

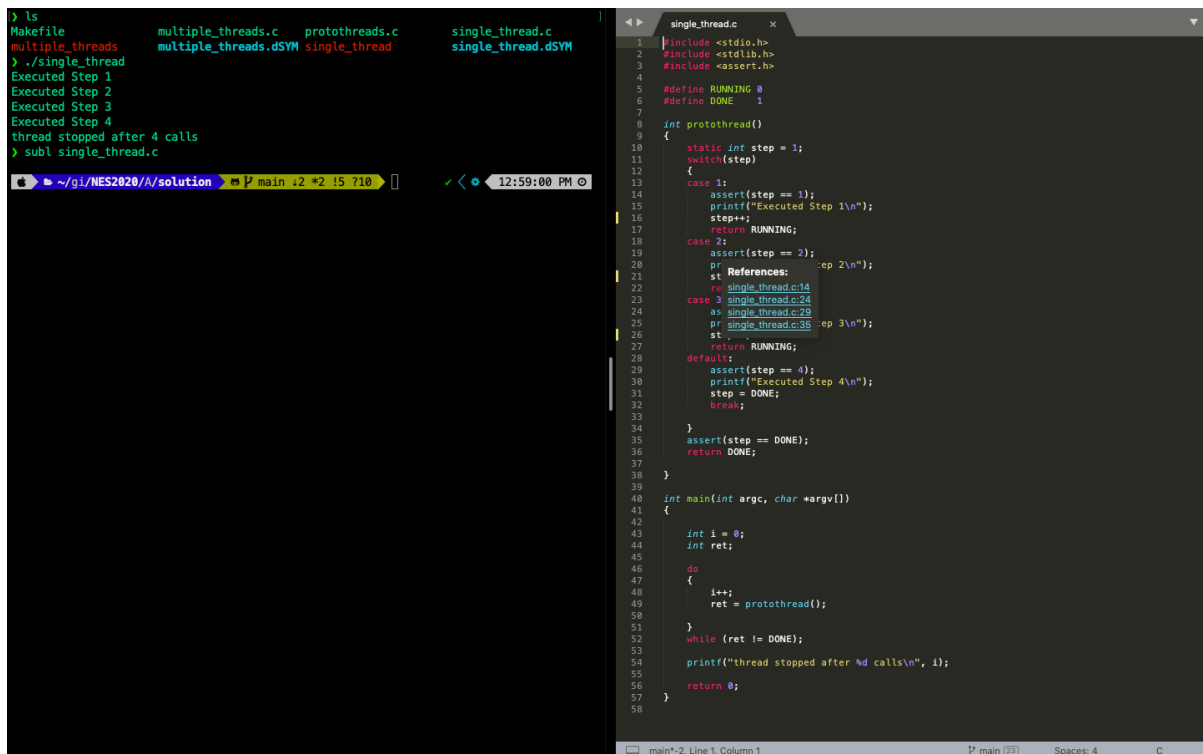
## NES Assignment – 1

### 1. Single\_thread.c

#### Description:

In this program, A single thread is created with a switch case to redirect to different execution step where a message is printed to show the current execution step. The execution is stopped when the thread returns DONE which is a constant 1.

#### Output screenshots:



The screenshot shows a terminal window on the left and a code editor on the right. The terminal displays the output of running the program, showing four execution steps and a final message indicating the thread stopped after 4 calls. The code editor shows the source code of single\_thread.c, which defines a protothread with four cases, each printing a message and incrementing a step counter. The main function calls the protothread and prints the final output.

```
> ls
Makefile      multiple_threads.c  protothreads.c      single_thread.c
multiple_threads.dSYM  single_thread.dSYM

> ./single_thread
Executed Step 1
Executed Step 2
Executed Step 3
Executed Step 4
thread stopped after 4 calls
> subl single_thread.c
```

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <assert.h>
4
5  #define RUNNING 0
6  #define DONE 1
7
8  int protothread()
9  {
10     static int step = 1;
11     switch(step)
12     {
13     case 1:
14         assert(step == 1);
15         printf("Executed Step 1\n");
16         step++;
17         return RUNNING;
18     case 2:
19         assert(step == 2);
20         printf("ep 2\n");
21         References:
22         re single_thread.c:14
23         case 3 single_thread.c:24
24         as single_thread.c:28
25         pr single_thread.c:35 ep 3\n");
26         st ., .
27         return RUNNING;
28     default:
29         assert(step == 4);
30         printf("Executed Step 4\n");
31         step = DONE;
32         break;
33     }
34     assert(step == DONE);
35     return DONE;
36 }
37
38 int main(int argc, char *argv[])
39 {
40     int i = 0;
41     int ret;
42
43     do
44     {
45         i++;
46         ret = protothread();
47     }
48     while (ret != DONE);
49     printf("thread stopped after %d calls\n", i);
50     return 0;
51 }
```

### 2. Multithreading:

#### Description:

We added 2 protothreads here. The first with 4 steps/cases, each printing a message indicating the thread ID and the execution step. and the second with 2 execution steps, printing similar output.

Extend one protothread with a local variable (for example an integer that is incremented in each execution step). Output also the current value of the variable in each step. Reflect on the problems that might occur here. Why is the naive approach not working? Try to pinpoint the core of the problem to understand when normal local variables can be used and when that's not possible.

#### Implementation:

We have implemented a protothread with a local variable and the value of the local variable is changed in each execution step and printed in the output.

## Observation:

The changed value of the local variable is not reflected in the output because the values of the local variables are not preserved but in the case of static variables the value is preserved across multiple executions.

Local variables have block scope so cannot be used across multiple execution. Local variables gets destroyed once block ends. It can be used to perform simple operations in a single execution.

## Output screenshots:

The screenshot shows a terminal window on the left and a code editor on the right. The terminal output shows the execution of a program with protothreads, displaying steps executed in different threads. The code editor shows the implementation of the protothreads, including the main function and the protothread functions.

```
ls
Makefile      multiple_threads.c  protothreads.c      single_thread.c
multiple_threads  multiple_threads.dSYM  single_thread      single_thread.dSYM
> ./multiple_threads
Executed Step 1 in thread protothread1
Executed Step 1 in protothread2 and local variable a with value 1
Executed Step 2 in thread protothread1
Executed Step 2 in protothread2 and local variable a with value 1
Executed Step 3 in thread protothread1
Executed Step 4 in thread protothread1
```

```
multiple_threads.c
34     return ++;
35 } else {
36     return 0;
37 }
38 }
39 thread_t* threads_pop_front() {
40     thread_t* front = threads;
41     threads = front->next;
42     return front;
43 }
44
45 int protothread1() {
46     static int step = 1;
47     char thread_id[] = "protothread1";
48     switch(step)
49     {
50     case 1:
51         printf("Executed Step %d in thread %s \n", step, thread_id);
52         step = 2;
53         return RUNNING;
54     case 2:
55         printf("Executed Step %d in thread %s \n", step, thread_id);
56         step = 3;
57         return RUNNING;
58     case 3:
59         printf("Executed Step %d in thread %s \n", step, thread_id);
60         step = 4;
61         return RUNNING;
62     default:
63         printf("Executed Step %d in thread %s \n", step, thread_id);
64         step++;
65         break;
66     }
67     return DONE;
68 }
69
70 int protothread2() {
71     static int step = 1;
72     int a = 0;
73     char thread_id[] = "protothread2";
74     switch(step)
75     {
76     case 1:
77         a++;
78         printf("Executed Step %d in %s and local variable a with value %d \n", step, thread_id, a);
79         step = 2;
80         return RUNNING;
81     case 2:
82         a++;
83         printf("Executed Step %d in %s and local variable a with value %d \n", step, thread_id, a);
84         step = 3;
85         return RUNNING;
86     default:
87         a++;
88         printf("Executed Step %d in %s and local variable a with value %d \n", step, thread_id, a);
89         step = 3;
90         break;
91     }
92     return DONE;
93 }
94
95 int main(int argc, char *argv[]) {
96     // main function
97 }
```

## 3. Protothreads.c:

### Description:

Here we have implemented 3 protothreads. The first one is automatically started and it calls protothread 3 once a button is clicked from inside protothread 1 and prints the corresponding message. The second protothread is automatically started and prints it's message

You should now be able to answer questions like: • How are protothreads different from, for example, Linux threads?

- Protothreads are stackless threads while Linux threads use stacks. This saves the overhead of large amounts of memory space, as it is anyway not required for Wireless sensor networks.
- Protothreads are independent of Operating system.

- Linux threads need a scheduler/handler for execution of threads but scheduling in case of protothreads have to be taken care by the developer.

### Would it be possible to just reimplement Linux threads for our sensor motes?

It is possible but it again requires large amounts of memory space.

### Protothread macros:

- *PROCESS\_BEGIN()* - The process begins with this macro. The process thread starts from here.
- *PROCESS\_END()* - This indicates the end of the process. The process is then removed from Kernel's list of active processes.
- *PROCESS\_EXIT()* - A process can end in 2 ways. When it reaches *PROCESS\_END()* macro, or when another process calls *process\_exit()* function.
- *PROCESS\_WAIT\_EVENT()* - It waits for an event to occur, it could be any event.
- *PROCESS\_WAIT\_EVENT\_UNTIL()* - It waits for an event to occur, but the occurrence of event is specified with conditions.
- *PROCESS\_YIELD()* - This macro also waits for any event to occur.
- *PROCESS\_WAIT\_UNTIL()* - It only waits for a condition. It might not yield the process.
- *PROCESS\_PAUSE()* - It yields the process temporarily.

### Output Screenshots:

