

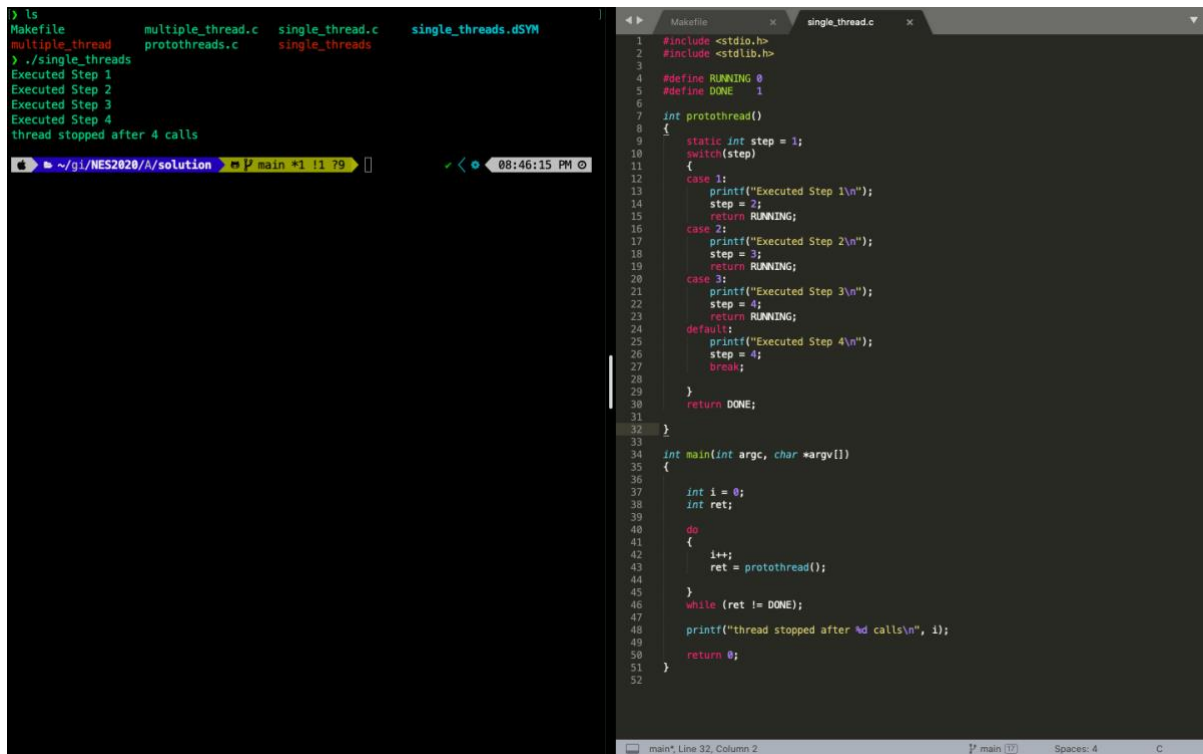
NES Assignment – 1

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:



```
> ls
Makefile      multiple_thread.c  single_thread.c    single_threads.dSYM
multiple_thread  protothreads.c    single_threads

> ./single_threads
Executed Step 1
Executed Step 2
Executed Step 3
Executed Step 4
thread stopped after 4 calls

~/gsl/NES2020//solution main *1 11 79 08:46:15 PM
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #define RUNNING 0
5 #define DONE 1
6
7 int protothread()
8 {
9     static int step = 1;
10    switch(step)
11    {
12    case 1:
13        printf("Executed Step 1\n");
14        step = 2;
15        return RUNNING;
16    case 2:
17        printf("Executed Step 2\n");
18        step = 3;
19        return RUNNING;
20    case 3:
21        printf("Executed Step 3\n");
22        step = 4;
23        return RUNNING;
24    default:
25        printf("Executed Step 4\n");
26        step = 4;
27        break;
28    }
29    return DONE;
30 }
31
32
33
34 int main(int argc, char *argv[])
35 {
36
37     int i = 0;
38     int ret;
39
40     do
41     {
42         i++;
43         ret = protothread();
44     }
45     while (ret != DONE);
46
47     printf("thread stopped after %d calls\n", i);
48
49     return 0;
50 }
51
52
```

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:

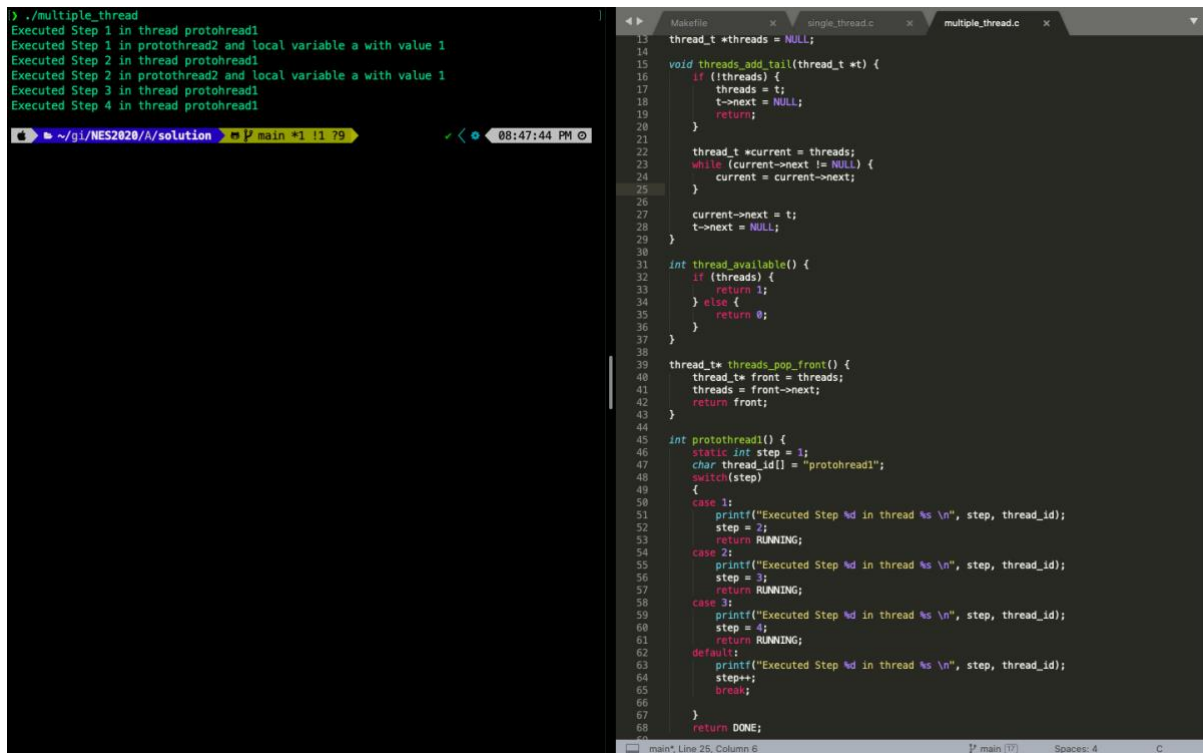
We have implemented a protothread with a local variable and the value of the local variable is changed in each execution step and printed in the output.

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 cannot be used across multiple execution. It can be used to perform simple operations in a single execution.

Output screenshots:



```
./multiple_thread
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
```

```
13 thread_t *threads = NULL;
14
15 void threads_add_tail(thread_t *t) {
16     if (!threads) {
17         threads = t;
18         t->next = NULL;
19         return;
20     }
21
22     thread_t *current = threads;
23     while (current->next != NULL) {
24         current = current->next;
25     }
26
27     current->next = t;
28     t->next = NULL;
29 }
30
31 int thread_available() {
32     if (threads) {
33         return 1;
34     } else {
35         return 0;
36     }
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 }
```

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. The second protothread is automatically started and prints a message/ The third protothread is started by protothread 1 only when the button is clicked and it prints a 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:

