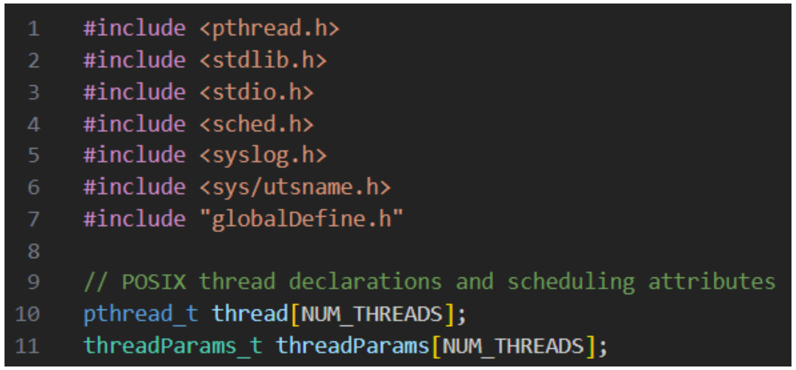
Hello Word Simple Thread Creation

- The code has 2 files:
  pthread.c: logic to implement the threads.
- globalDefines.h: file with all generic definitions.

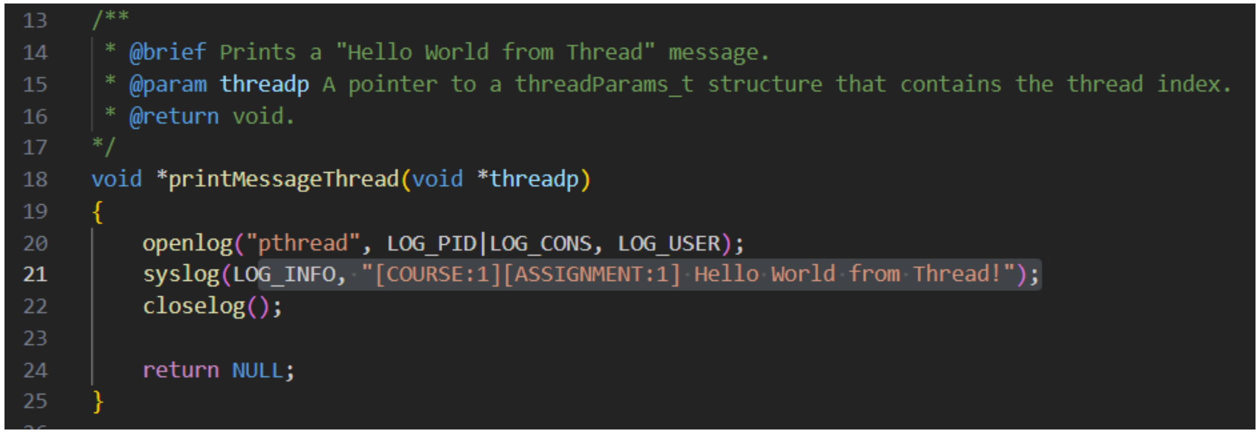
- Pthread file descripction:
- From line 1 to 7 are all file included.
- Line 10 an 11 are the thread and thread parameters



```c
1   #include <pthread.h>
2   #include <stdlib.h>
3   #include <stdio.h>
4   #include <sched.h>
5   #include <syslog.h>
6   #include <sys/utsname.h>
7   #include "globalDefine.h"
8
9   // POSIX thread declarations and scheduling attributes
10  pthread_t thread[NUM_THREADS];
11  threadParams_t threadParams[NUM_THREADS];
```

Hello Word Simple Thread Creation

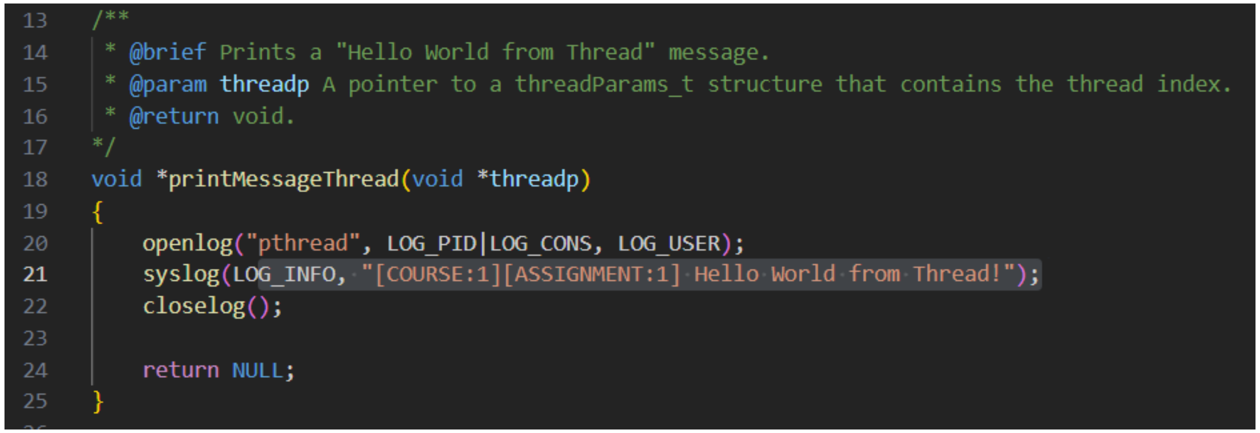
- From line 18 to 25 is the “printMessageThread” function implementation that has a threadp void pointer, in this implementation de argument is not needed.
- From line 20 to 22 the information is written in the syslog file.



```
13  /**
14   * @brief Prints a "Hello World from Thread" message.
15   * @param threadp A pointer to a threadParams_t structure that contains the thread index.
16   * @return void.
17  */
18  void *printMessageThread(void *threadp)
19  {
20      openlog("pthread", LOG_PID|LOG_CONS, LOG_USER);
21      syslog(LOG_INFO, "[COURSE:1][ASSIGNMENT:1] Hello World from Thread!");
22      closelog();
23
24      return NULL;
25  }
```

Hello Word Simple Thread Creation

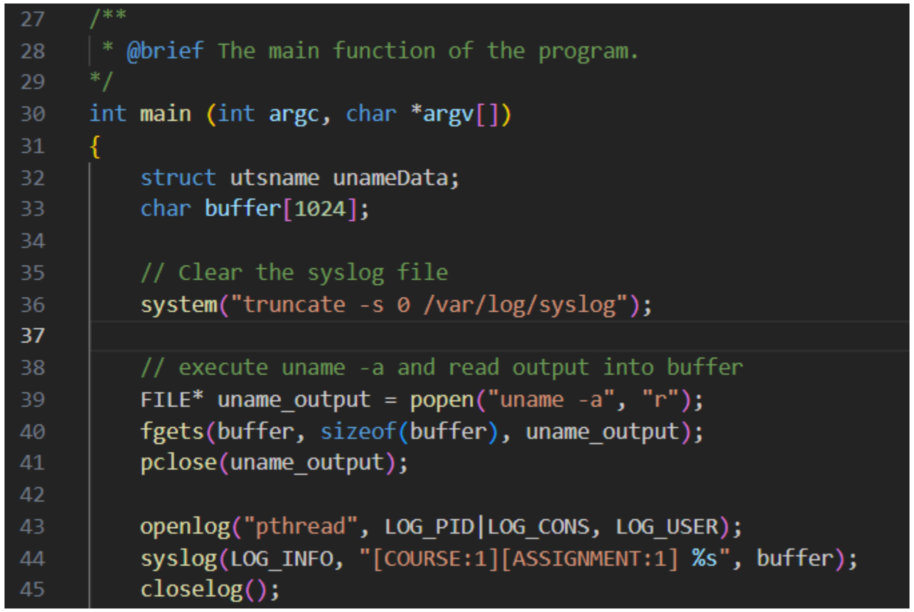
- From line 18 to 25 is the “printMessageThread” function implementation that has a threadp void pointer, in this implementation de argument is not needed.
- From line 20 to 22 the information is written in the syslog file from the threads.



```c
13  /**
14   * @brief Prints a "Hello World from Thread" message.
15   * @param threadp A pointer to a threadParams_t structure that contains the thread index.
16   * @return void.
17  */
18  void *printMessageThread(void *threadp)
19  {
20      openlog("pthread", LOG_PID|LOG_CONS, LOG_USER);
21      syslog(LOG_INFO, "[COURSE:1][ASSIGNMENT:1] Hello World from Thread!");
22      closelog();
23  
24      return NULL;
25  }
```

Hello Word Simple Thread Creation

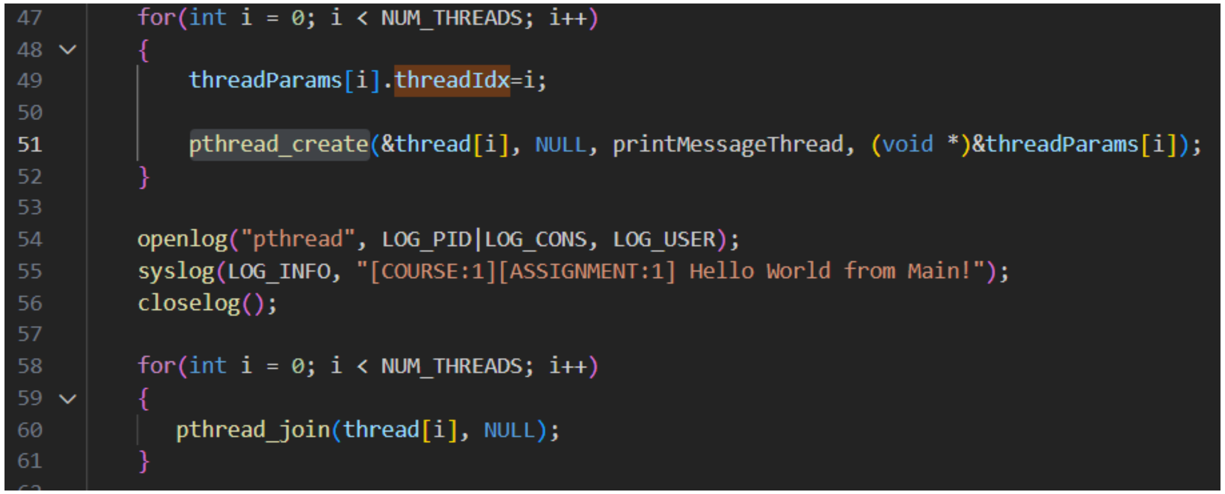
- From line 27 to 64 the main function is implemented
- Line 32 and 33 are the variables to write the “uname -a” information.
- Line 36 is to clean the syslog file
- From line 39 to 45 the uname -a information is written in the first syslog line file.



```c
27  /**
28   * @brief The main function of the program.
29   */
30  int main (int argc, char *argv[])
31  {
32      struct utsname unameData;
33      char buffer[1024];
34
35      // Clear the syslog file
36      system("truncate -s 0 /var/log/syslog");
37
38      // execute uname -a and read output into buffer
39      FILE* uname_output = popen("uname -a", "r");
40      fgets(buffer, sizeof(buffer), uname_output);
41      pclose(uname_output);
42
43      openlog("pthread", LOG_PID|LOG_CONS, LOG_USER);
44      syslog(LOG_INFO, "[COURSE:1][ASSIGNMENT:1] %s", buffer);
45      closelog();
```

Hello Word Simple Thread Creation

- From line 47 to 52 a for is implemented to crate the threads, to do that we use pthread_create.
- From line 54 to 56 thw syslog file is writen from main function.
- From line 58 to 61 the execute waits until all theads finish.



```
47      for(int i = 0; i < NUM_THREADS; i++)
48      {
49          threadParams[i].threadIdx=i;
50
51          pthread_create(&thread[i], NULL, printMessageThread, (void *)&threadParams[i]);
52      }
53
54      openlog("pthread", LOG_PID|LOG_CONS, LOG_USER);
55      syslog(LOG_INFO, "[COURSE:1][ASSIGNMENT:1] Hello World from Main!");
56      closelog();
57
58      for(int i = 0; i < NUM_THREADS; i++)
59      {
60         pthread_join(thread[i], NULL);
61      }
```