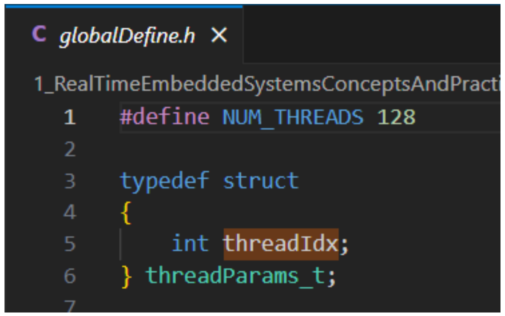
Multiple Threads Fifo

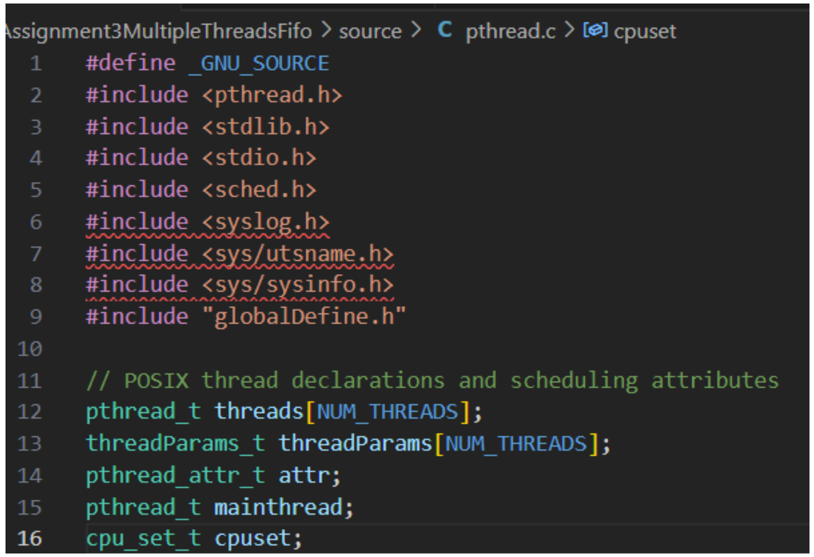
- The code has 2 files:
  pthread.c: logic to implement the threads.
- globalDefines.h: file with all generic definitions.
- Pthread file descripction:
- From line 1 to 9 are all file included.
- Line 11 an 16 are the thread and thread parameters



globalDefine.h ×

1_RealTimeEmbeddedSystemsConceptsAndPracti

```
1  #define NUM_THREADS 128
2
3  typedef struct
4  {
5      int threadIdx;
6  } threadParams_t;
7
```



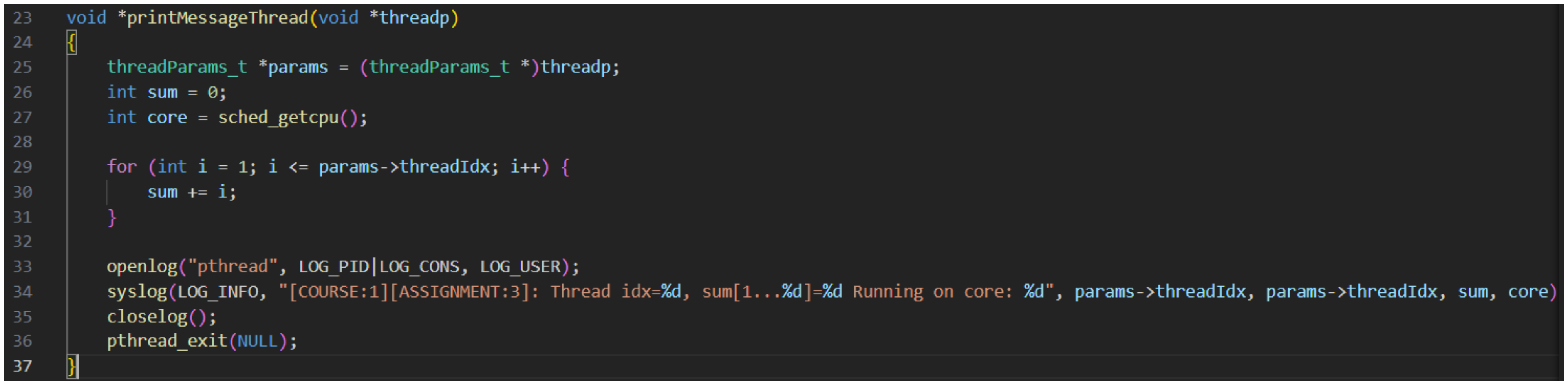
ssignment3MultipleThreadsFifo > source > pthread.c > cpuset

```
1   #define _GNU_SOURCE
2   #include <pthread.h>
3   #include <stdlib.h>
4   #include <stdio.h>
5   #include <sched.h>
6   #include <syslog.h>
7   #include <sys/utsname.h>
8   #include <sys/sysinfo.h>
9   #include "globalDefine.h"
10
11  // POSIX thread declarations and scheduling attributes
12  pthread_t threads[NUM_THREADS];
13  threadParams_t threadParams[NUM_THREADS];
14  pthread_attr_t attr;
15  pthread_t mainthread;
16  cpu_set_t cpuset;
```

Multiple Threads Fifo

From line 23 to 37 is the “printMessageThread” function implementation that has a threadp void pointer, in this implementation de argument is needed to know the thread index.

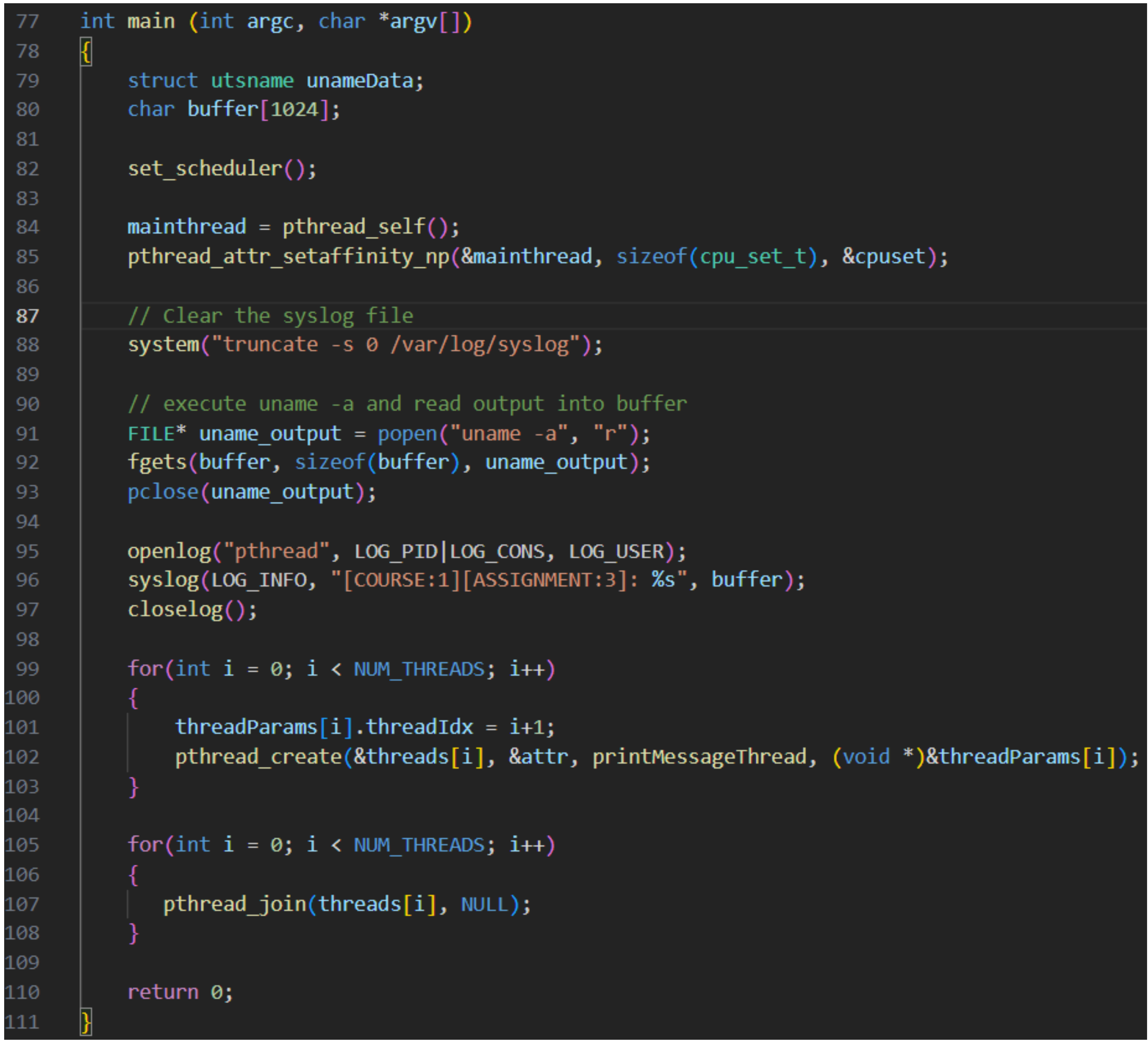
In line 27 we get the core where the pthread is rinning, from line 29 to 31 a sum is done from 1 to the thread index, after that from line 33 to 36 the thread information is written in the syslog file.



```
23  void *printMessageThread(void *threadp)
24  {
25      threadParams_t *params = (threadParams_t *)threadp;
26      int sum = 0;
27      int core = sched_getcpu();
28
29      for (int i = 1; i <= params->threadIdx; i++) {
30          sum += i;
31      }
32
33      openlog("pthread", LOG_PID|LOG_CONS, LOG_USER);
34      syslog(LOG_INFO, "[COURSE:1][ASSIGNMENT:3]: Thread idx=%d, sum[1...%d]=%d Running on core: %d", params->threadIdx, params->threadIdx, sum, core)
35      closelog();
36      pthread_exit(NULL);
37  }
```

Multiple Threads Fifo

- From line 77 to 111 the main function is implemented
- Line 79 and 80 are the variables to write the “uname -a” information.
- In line 84 and 85 we set the main thread in the cpu selected.
- Line 88 is to clean the syslog file.
- From line 91 to 97 we get the uname -a information and after that the information is written in the first syslog line file.

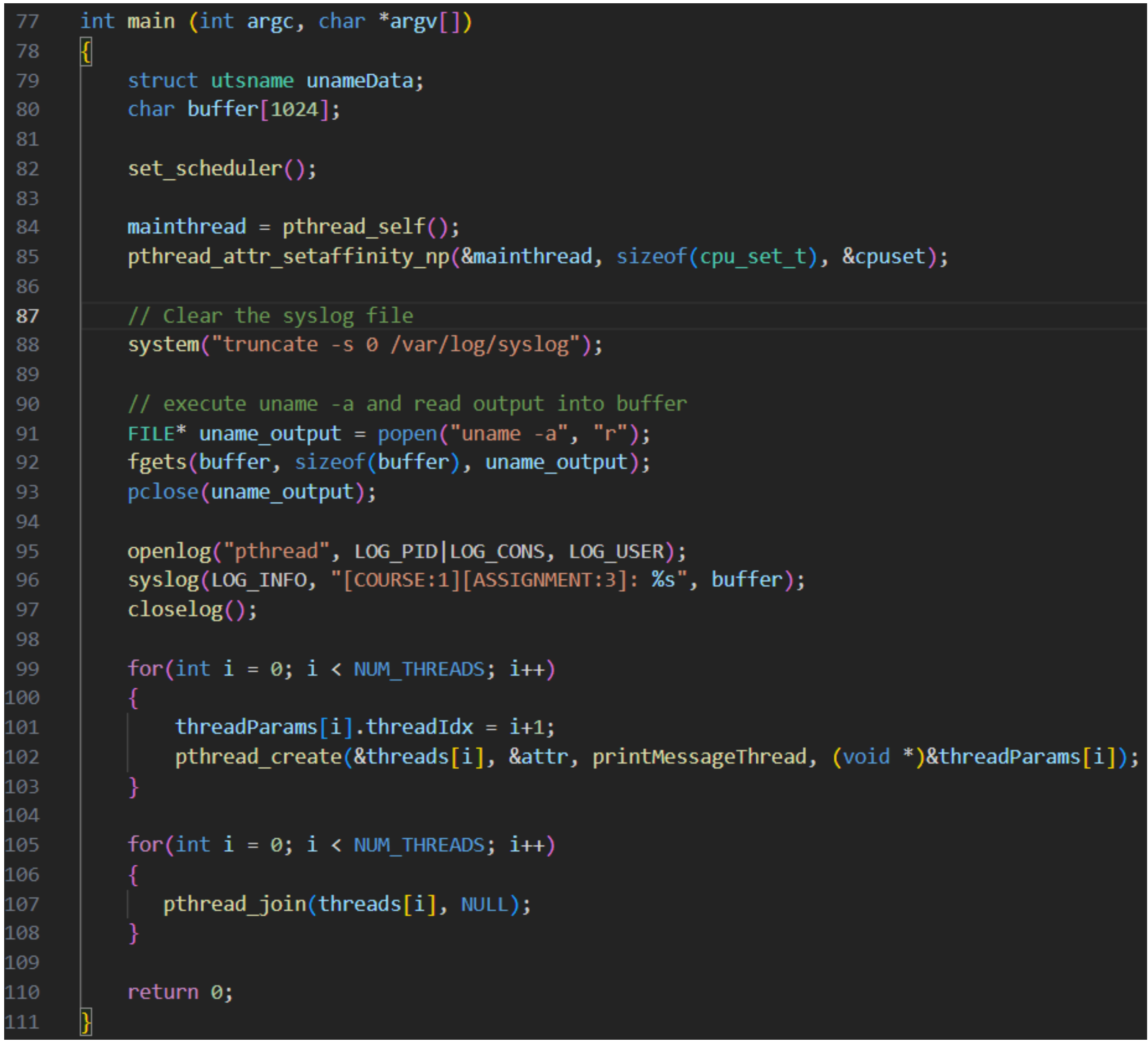


```
77   int main (int argc, char *argv[])
78   {
79       struct utsname unameData;
80       char buffer[1024];
81
82       set_scheduler();
83
84       mainthread = pthread_self();
85       pthread_attr_setaffinity_np(&mainthread, sizeof(cpu_set_t), &cpuset);
86
87       // Clear the syslog file
88       system("truncate -s 0 /var/log/syslog");
89
90       // execute uname -a and read output into buffer
91       FILE* uname_output = popen("uname -a", "r");
92       fgets(buffer, sizeof(buffer), uname_output);
93       pclose(uname_output);
94
95       openlog("pthread", LOG_PID|LOG_CONS, LOG_USER);
96       syslog(LOG_INFO, "[COURSE:1][ASSIGNMENT:3]: %s", buffer);
97       closelog();
98
99       for(int i = 0; i < NUM_THREADS; i++)
100      {
101          threadParams[i].threadIdx = i+1;
102          pthread_create(&threads[i], &attr, printMessageThread, (void *)&threadParams[i]);
103      }
104
105      for(int i = 0; i < NUM_THREADS; i++)
106      {
107         pthread_join(threads[i], NULL);
108      }
109
110      return 0;
111  }
```

Multiple Threads Fifo

From line 99 to 103 a for is implemented to crate the threads, to do that we use pthread_create with printMessageThread function in the argument to write the phreat information, sum index and core where is running.

From line 105 to 108 the execute waits until all theads finish.



```
77  int main (int argc, char *argv[])
78  {
79      struct utsname unameData;
80      char buffer[1024];
81
82      set_scheduler();
83
84      mainthread = pthread_self();
85      pthread_attr_setaffinity_np(&mainthread, sizeof(cpu_set_t), &cpuset);
86
87      // Clear the syslog file
88      system("truncate -s 0 /var/log/syslog");
89
90      // execute uname -a and read output into buffer
91      FILE* uname_output = popen("uname -a", "r");
92      fgets(buffer, sizeof(buffer), uname_output);
93      pclose(uname_output);
94
95      openlog("pthread", LOG_PID|LOG_CONS, LOG_USER);
96      syslog(LOG_INFO, "[COURSE:1][ASSIGNMENT:3]: %s", buffer);
97      closelog();
98
99      for(int i = 0; i < NUM_THREADS; i++)
100     {
101         threadParams[i].threadIdx = i+1;
102         pthread_create(&threads[i], &attr, printMessageThread, (void *)&threadParams[i]);
103     }
104
105     for(int i = 0; i < NUM_THREADS; i++)
106     {
107        pthread_join(threads[i], NULL);
108     }
109
110     return 0;
111 }
```