Practice Using Generic Sequencer To Create Scenario

The code has 2 files:
sequencerGenerator.c: logic to implement the threads.

- globalDefines.h: file with all generic definitions.

sequencerGenerator file descripction:

- From line 1 to 14 are all file included.
- Line 17 and 27 are the thread and thread parameters, semaphoreds, and some couters

globalDefine.h 1

PracticeUsingGenericSequencerToCreate

```c
1  #define NUM_THREADS 3
2
3  #define TRUE (1)
4  #define FALSE (0)
5
6  typedef struct
7  {
8      int threadIdx;
9  } threadParams_t;
```

```c
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 <errno.h>
10 #include <time.h>
11 #include <signal.h>
12 #include <unistd.h>
13 #include <semaphore.h>
14 #include "globalDefine.h"
15
16 // POSIX thread declarations and scheduling attributes
17 pthread_t threads[NUM_THREADS];
18 threadParams_t threadParams[NUM_THREADS];
19 pthread_attr_t attr;
20 pthread_t mainthread;
21 cpu_set_t cpuset;
22 static timer_t timer_1;
23 static struct itimerspec itime = {{1,0}, {1,0}};
24 static struct itimerspec last_itime;
25 static unsigned long long interruptCounter = 0;
26 static unsigned long long totalCounter = 0;
27 sem_t semS1, semS2, semS3;
```

Practice Using Generic Sequencer To Create Scenario

From line 30 to 66 is the “Function Sequencer(): This function is called every 10 milliseconds, using the timer_1 timer. On each call, the function increments the interruptCounter counter. If the value of interruptCounter is 30, it resets it to 0.

Depending on the value of interruptCounter, the Sequencer() function sends signals to the semS1, semS2, and semS3 semaphores. This indicates to the corresponding services that it is time to run.

```
29  // it is called every 10 ms
30  void Sequencer(int id)
31  {
32      int rc, flags=0;
33
34      if(interruptCounter == 30)
35      {
36          interruptCounter = 0;
37      }
38
39      if(interruptCounter % 2 == 0)
40      {
41          //Service_1 start
42          sem_post(&semS1);
43      }
44
45      if(interruptCounter == 1 ||
46      interruptCounter == 11 ||
47      interruptCounter == 21)
48      {
49          //Service_2 start
50          sem_post(&semS2);
51      }
52
53      if(interruptCounter == 3 ||
54      interruptCounter == 6 ||
55      interruptCounter == 5 ||
56      interruptCounter == 15 ||
57      interruptCounter == 17)
58      {
59          //Service_3 start
60          sem_post(&semS3);
61      }
62
63      // received interval timer signal
64      interruptCounter++;
65      totalCounter++;
66  }
```

Practice Using Generic Sequencer To Create Scenario

From line 68 to 108 are Function Service_1(): This function is executed when service 1 receives a signal from the semS1 semaphore. The function simply prints a message to the console and then waits for another signal to be sent to the semaphore.

Function Service_2(): This function is executed when service 2 receives a signal from the semS2 semaphore. The function simply prints a message to the console and then waits for another signal to be sent to the semaphore.

Function Service_3(): This function is executed when service 3 receives a signal from the semS3 semaphore. The function simply prints a message to the console and then waits for another signal to be sent to the semaphore.

```
68  void *Service_1(void *threadp)
69  {
70      while(TRUE)
71      {
72          // wait for service request from the sequencer
73          sem_wait(&semS1);
74          //Do work
75
76          openlog("pthread", LOG_PID|LOG_CONS, LOG_USER);
77          syslog(LOG_INFO, "[Service Generator]: S1 T1=%d ms", ((totalCounter-1)*10L));
78          closelog();
79      }
80  }
81
82  void *Service_2(void *threadp)
83  {
84      while(TRUE)
85      {
86          // wait for service request from the sequencer
87          sem_wait(&semS2);
88          //Do work
89
90          openlog("pthread", LOG_PID|LOG_CONS, LOG_USER);
91          syslog(LOG_INFO, "[Service Generator]: S2 T2=%d ms", ((totalCounter-1)*10L));
92          closelog();
93      }
94  }
95
96  void *Service_3(void *threadp)
97  {
98      while(TRUE)
99      {
100         // wait for service request from the sequencer
101         sem_wait(&semS3);
102         //Do work
103
104         openlog("pthread", LOG_PID|LOG_CONS, LOG_USER);
105         syslog(LOG_INFO, "[Service Generator]: S3 T3=%d ms", ((totalCounter-1)*10L));
106         closelog();
107     }
108 }
```

Practice Using Generic Sequencer To Create Scenario

The Function

set_scheduler(): This function sets the scheduling policy and CPU affinity for the threads. The scheduling policy is set to FIFO, which guarantees that the threads will run in order of priority. The CPU affinity is set to a single core.

```c
void set_scheduler(void)
{
    int cpu_set_t;
    int cpuIndex;

    // zero out the set of CPU cores.
    CPU_ZERO(&cpuset);

    //Here we assign the threads to run ONLY on core 2.
    cpuIndex=(1);
    CPU_SET(cpuIndex, &cpuset);

    // Set scheduling policy to FIFO
    struct sched_param schedParam;
    schedParam.sched_priority = sched_get_priority_max(SCHED_FIFO); //99
    sched_setscheduler(0, SCHED_FIFO, &schedParam);

    // Set thread attributes to use FIFO scheduling policy
    pthread_attr_init(&attr);
    pthread_attr_setinheritsched(&attr, PTHREAD_EXPLICIT_SCHED);
    pthread_attr_setschedpolicy(&attr, SCHED_FIFO);
    pthread_attr_setaffinity_np(&attr, sizeof(cpu_set_t), &cpuset);
    pthread_attr_setschedparam(&attr, &schedParam);
}
```

Practice Using Generic Sequencer To Create Scenario

Function main(): This function is the main function of the program. The function creates the three threads (service 1, service 2, and service 3) and then starts the timer_1 timer. Once the timer has started, the main() function waits for all threads to finish executing.

```c
int main (int argc, char *argv[])
{
    int flags=0;
    struct utsname unameData;
    char buffer[1024];

    set_scheduler();

    mainthread = pthread_self();
    pthread_attr_setaffinity_np(&mainthread, sizeof(cpu_set_t), &cpuset);

    // Clear the syslog file
    system("truncate -s 0 /var/log/syslog");

    // execute uname -a and read output into buffer
    FILE* uname_output = popen("uname -a", "r");
    fgets(buffer, sizeof(buffer), uname_output);
    pclose(uname_output);

    openlog("pthread", LOG_PID|LOG_CONS, LOG_USER);
    syslog(LOG_INFO, "[Service Generator]: %s", buffer);
    closelog();

    sem_init(&semS1, 0, 0);
    sem_init(&semS2, 0, 0);
    sem_init(&semS3, 0, 0);

    pthread_create(&threads[0], &attr, Service_1, (void *)&threadParams[0]);
    pthread_create(&threads[1], &attr, Service_2, (void *)&threadParams[0]);
    pthread_create(&threads[2], &attr, Service_3, (void *)&threadParams[0]);

    // Sequencer = RT_MAX   @ 100 Hz
    //
    /* set up to signal SIGALRM if timer expires */
    timer_create(CLOCK_REALTIME, NULL, &timer_1);
    signal(SIGALRM, (void(*)()) Sequencer);
    itime.it_interval.tv_sec = 0;
    itime.it_interval.tv_nsec = 10000000; //10 ms
    itime.it_value.tv_sec = 0;
    itime.it_value.tv_nsec = 10000000;
    timer_settime(timer_1, flags, &itime, &last_itime);



    for(int i = 0; i < NUM_THREADS; i++)
    {
        pthread_join(threads[i], NULL);
    }

    return 0;
}
```