BE minepump

Exercise 1

We initialize the m_integer structure by initializing the mutex. For now, we set NULL as a second argument.

In the function MI_write, we must lock the mutex before writing to avoid two threads writing at the same time. Same when we read with MI_read, so that no other thread use MI_write at the same time.

Exercise 2

We initialize the msg_box structure by initializing the mutex and the condition variable. At the beginning the box is empty so (*mbox).empty = true.

In msg_box_receive, we are waiting to receive a message: we lock the mutex, we check if the box is empty, if it's the case we unlock the mutex and we wait for the conditional variable to send a signal. When we receive a signal we lock the mutex, we check if the box is empty, etc. When the box is not empty, then there is a message and we are the first thread to get it. So we empty the box, we keep the message in memory and we release the mutex.

In order to put a message in the box with the function msg_box_send, it is simpler: we copy the message in the box, we note that it is not empty, we broadcast the conditional variable and we unlock the mutex.

Exercise 3

To create a periodic task with create_periodic_task, we have to create a thread with the good arguments.

What the periodic task does is defined in periodic_task_body. The periodicity is implemented thanks to an infinite for loop and sem_timedwait.

Exercise 4

In WaterLevelMonitoring_Body, we write the content of the periodic task that deals with the level of water : we write in the m_integer LvlWater the level of water (high or low). Same with MethaneMonitoring_Body for the level of methane : we write in LvlMeth the alarm level regarding whether there is methane detected or not.

In PumpCtrl_Body, we write the content of the sporadic task, awaken by a level of methane too high : we turn the alarm on if needed, we switch the pump off if we are in alarm 2. If the alarm is at level 1 or normal, we pump if the water is too high and we don't pump otherwise. In CmdAlarm_Body, we write the content of the sporadic task dealing with the messages in mbox_alarm.

At last, in the main we create the periodic tasks with the requested periods, and the two sporadic tasks.

Exercise 5 :

The mutexes that protect the minteger and the message boxes need to have a protocol to tell in which order to give access to waiting threads, in order to avoid priority inversion.

minteger.c:

pthread_mutex_init needs a pthread_mutexattr_t* to set the attributes of the mutex, in particular the protocol.

Two choices are available : INHERIT and PROTECT,

- INHERIT dynamically sets the priority of the thread in possession of the mutex to the highest priority of the threads that want to access it
- PROTECT statically sets the priority to a level determined by the developper, which should be the highest priority of the treads that could want to access the mutex

Since we have a very simple program we can determine the priorities of every thread, so we choose to set the static protocol.

Following are the steps to create and set those attributes :

```
// create the pointer to the attributes structure
pthread_mutexattr_t *p_attr;
// initialize it
pthread_mutexattr_init(p_attr);
// set the protocol
pthread_mutexattr_setprotocol(p_attr, PTHREAD_PRIO_PROTECT);
// set the priority ceiling
pthread_mutexattr_setprioceiling(p_attr, some_value);
```

We then need to set a real-time scheduling policy and give priorities to threads so as to guarantee real-time properties

periodic_task.c/minepump.c:

We need to choose a scheduling policy and to set threads priorities.

For the scheduling policy two "real-time" choices are available : FIFO and ROUND ROBIN (other non-real-time choices are available which do not have any priority policy).

We choose FIFO for its simplicity of usage and analysis.

As specified in the documentation, "Scheduling parameters are in fact per-thread attributes on Linux" (https://linux.die.net/man/2/sched_setparam).

More explanations from the documentation :

"Conceptually, the scheduler maintains a list of runnable processes for each possible sched_priority value. In order to determine which process runs next, the scheduler looks for the nonempty list with the highest static priority and selects the process at the head of this list. A process's scheduling policy determines where it will be inserted into the list of processes with equal static priority and how it will move inside this list." (https://linux.die.net/man/2/sched_setscheduler)

We then attribute a scheduling policy and a priority to every thread at its creation

Following are the steps to create and set those attributes :

```
// at the beginning of the file
#include <sched.h>
// create the structure of scheduler parameters
sched_param *p_param;
// set a priority into the parameters structure
p_param->sched_priority = some_value;
// set the scheduler policy for the given thread
sched_setscheduler(&tid, SCHED_FIFO, p_param);
```

Note : sched_priority value in the range 1 (low) to 99 (high)

We didn't specify the priorities here, since they have to be determined by an analysis of the system and the real-time requirements.

We can only infer that the methane control flow must be prior to the water level control flow.