Mine Pump

The goal of this project is to provide a full implementation of the "Mine Pump" case study using RT-POSIX[1].

Using code for this lab session

All source code is provided in the archive `minepump.tar.gz`. To open this archive, issue the following command "`tar zxvf minepump.tar.gz`" this will create the directory `minepump` with all source code.

A makefile is provided to compile all source code. Issue "`make help`" for more details.

Note: the code provided for this lab session has been tested for Linux OS only. It relies on mechanisms from Unix and POSIX libraries that are not available on Mac OS X or Windows.

Assignements

You are task to perform all exercises, and write an implementation report. This implementation report is made of two files: a text file describing what you did, and the annotated source code.
Hard deadline: end of this lab session.

The mine pump

A mine pump is a safety mechanism in place in a mine. It monitors various parameters to ensure safety of operators: level of water due to infiltrations, methane gas, etc.

About the minepump

In this project, we consider a safey mechanism used in mines: a mine pump[2].
The study concerns the design of software necessary to manage a simplified pump control system for a mining environment. The system is used to pump mine-water, which collects in a sump at the bottom of the shaft, to the surface. A simple schematic diagram illustrating the system is given in the following figure:
In the following, we will consider a restricted case study, with the following high-level considerations:

1. the system is made of two sensors: a methane detection sensor and a water sensor;
2. the pump relies on a combustion engine. For safety reasons, it cannot operate when the methane level is high. Otherwise, the engine may trigger an explosion;
3. sensors operate as periodic thread, each sensor outputs an integer;

[1]This lab is based on material from Emmanuel Grolleau from ISAE/ENSMA
[2]This case study is adapted from *"HRT-HOOD: A Structured Design Method for Hard Real-time Systems"*, by A. Burns and A. J. Wellings, JRTS 6 (1) 1994

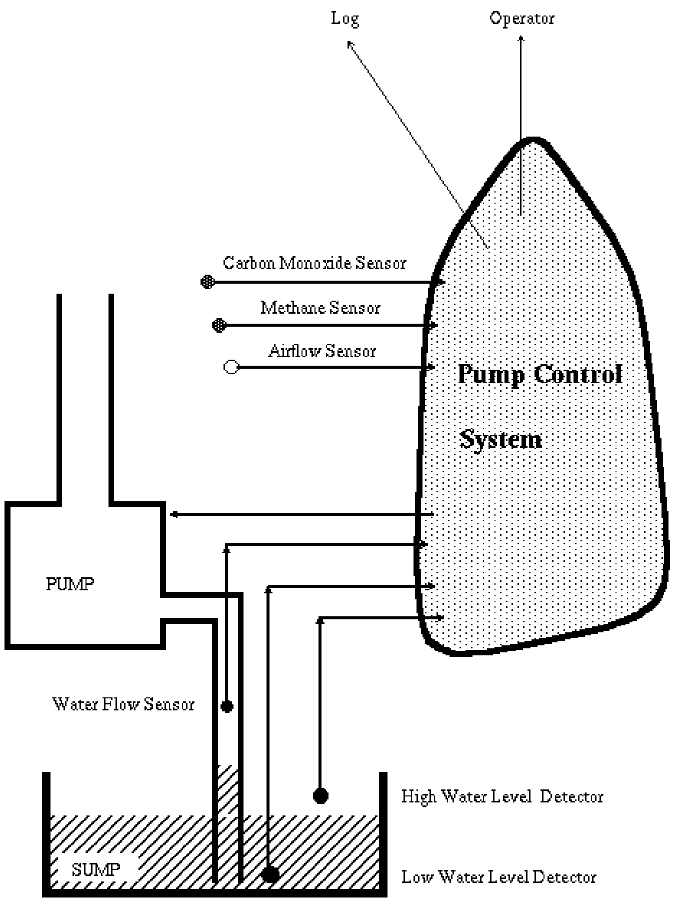

Figure 1: Overview of the mine pump system

4. a sporadic thread operates to control the pump. It is activated by the reception of an event from the methane sensor thread, or the water level sensor. This thread may send an alarm to the alarm controller thread;

5. a sporadic thread controls the alarm signal.

Code base

The code has the following organization:

- `minteger.[ch]` defines a mutex-protected integer;
- `msg_box.[ch]` defines a message box abstraction, with at most one message being stored. A thread may block and wait until a message arrives;
- `periodic_task.[ch]` defines an API to create periodic tasks;
- `simu.[ch]` defines a basic simulator for the Mine Pump system, in interaction with a Tcl/Tk graphical engine;
- `utils.h` defines various helper functions;
- `minepump.c` is the main entry point of this program.

Assignment

In the following exercises, we will build incrementally all required building blocks for the mine pump.

Exercise 1 (`minteger` implementation) *The `minteger.[ch]` unit defines a mutex-protected integer. In thie first exercise, you have to complete its implementation. Review this source code, and complete it to*

- *initialize the `m_integer` structure properly;*
- *implement the read/write operations.*

You may test this unit, simply run "make test_minteger" to build and execute the `test_minteger` program.
Note: you may use the `CHECK_NZ` macro from `utils.h` to check the return value from the POSIX API.

Exercise 2 (`msg_box` implementation) *The `msg_box` unit defines a basic message box abstraction. In this exercise, you have to complete its implementation. This message box follows a basic producer/consummer pattern.*

- *initialize the `msg_box` structure properly;*
- *implement the send/receive operations.*

You may test this unit, simply run "make test_msgbox" to build and execute the `test_msgbox` program.

Exercise 3 (Implementing periodic tasks) *The `periodic_task` unit defines a factory to create periodic taks. Implement the `create_periodic_task` and `periodic_task_body` functions.*
Note: To implement a periodic timer, we will use the `sem_timedwait` system call on a semaphore correctly initialized. Your report shall detail the rationale for the parameters you selected.
You may test this unit, simply run "make test_periodic" to build and execute the `test_periodic` program.

Exercise 4 (Running the minepump) *The `minepump` unit defines the working example. Complete the body of the four tasks following instructions in comments. Also complete the initialization part to setup all required tasks, communication and synchronization elements.*
To compile the program, simply run "make minepump". To run the program, you'll use

- *a TCL/TK GUI, running* `wish minepump.tcl`. *An optional command line argument may be used to indicate a TCP/IP port number;*
- *then the binary you created:* `./minepump`

The GUI provides a display of values sent to/from the program. You may interact with the GUI to check the program behaves as expected.

Exercise 5 (Did you say Real-Time ?) *For now, we did not consider real-time elements. Hence, we rely on defaut OS configuration parameters, running on a multi-core generic Unix-like OS.*

- *Propose additions to the existing code base to support real-time concerns so as to enforce determinism.*
- *How would you analyze the scheduling of the system? Provide a description of the tasks set, indicate your rationale to configure it.*
- *Test them on your workstation, what do you observe ?*

Exercise 6 (Removing the GUI, moving to RTEMS) *We will now move gradually to RTEMS.*
First, compile again your program using the `minepump_nogui` *target. It deactivates the socket communication with the Tcl/Tk simulator. Instead, it implements a basic loopback scenario in* `simu.c`. *Check the behavior of the system.*

The last part implies running on the PRISE platform. Log on this machine, move your files. Then, compile using the `minepump_rtems` *target, and run the system using tsim.*