Lab 4
FIFOs, Named Pipes, Serial Communication, Semaphores, Dynamic Threads and RT Tasks in Kernel Space

Objective
In this lab, students will learn more effective ways of performing inter-process synchronization and communication. In order to achieve the objective, students are required to:
- Create a FIFO to communicate between a real time task and a user space program
- Use semaphores as a form of synchronization between threads
- Use named pipes to communicate between programs and threads
- Receive data from serial port and match events with the received data (using a GPS device could be an application of this)

Prelab
Before coming to the lab you should explain the RTAI functions and arguments needed to create and destroy a FIFO, and the functions and arguments needed to read and write to these FIFOs in kernel space: rtf_create, rtf_destroy, rtf_get, rtf_put. Furthermore, explain the semaphore functions in user space: sem_init, sem_wait, sem_post, sem_destroy. Don’t forget to include the flowchart or pseudo code for the assignment.

Scenario
This lab simulates the situation when we want to find the exact location of a real time event. Assume there is a camera mounted on a helicopter, which takes snapshots at different times. A GPS module provides the location data every 250ms. In order to find the exact location of each snapshot, we need to interpolate the two GPS data, one before the events happened and one after.

Lab Procedure
In this lab, a user space program needs to be created that consists of a main loop that receives data from serial port, collects a time-stamp and saves both pieces of information in a buffer. Two other pthreads should also be created. The job of the first pthread is to read data (the time-stamp of the real time event) from a FIFO, and to store the GPS data (provided by the main loop) before and after the real time event. The job of the second pthread is to receive the data (two GPS data and their time-stamps, and the time-stamp of the real time event) through a named pipe and print it out on the screen. In addition to the user space program, a kernel module will
need to be installed. It will create a real time task, check the real time event, collect the time-stamp and send it through a FIFO. The overall flow of the program is as follows:

1. The main program will loop, receiving serial data and collecting time-stamps.
2. When the data is received, it is saved in a common memory buffer by the main program.
3. The real-time task waits for a push button (B0) to be pressed, then passes the time-stamp to the first pthread through a FIFO.
4. The first pthread reads this time-stamp from the FIFO, collects the GPS data and time-stamps right before and after the real time event, and sends these data through a named pipe to the second pthread.
5. The second pthread then prints out all the data and time-stamps.

This lab is broken down into **four parts** (and you'll have **two weeks** to complete them).

**Part 1: Main Program and Serial Communication**

In this part, you need to run a program (will be given in the lab) to receive data from the serial port and make sure the serial communication is working properly. Modify the source code to get the time-stamp of the data received. The data and time-stamps should be placed in a common buffer. A pthread should be created, which will have access to the buffer. Print the data to the screen, just to make sure that you are getting them correctly.

An executable file is provided for students (*/opt/ts7250/students/*), which sends the GPS data (8 bit integer values (char)) every 250 ms through serial port with baud rate of 115200, 8 bit, no parity and 1 stop bit (this will be referred to as the GPS event). A function to receive the data from the serial port on COM2 (/dev/ttyAM1) of the embedded device will be given to you. This function will be a part of your main program.

**Part 2: Kernel Module**

The first purpose of the module is to set up a FIFO that will be used for communication between the user space program and a real time task. The second purpose of the module is to create the real time task that writes to the FIFO. The real time task should go off every 75ms. Remember to check for errors when setting up the task and FIFOs.

The real time task should check the push button on port B0, collect a time-stamp when it is pressed (this will be referred to as the real time or push button event) and write the time-stamp to the FIFO.

**Note 1:** make sure that the *rtai_fifos.o* and *rtai_ksched.o* modules are installed (use *lsmod*). If they are not, you need to locate and install them (using *insmod*).

**Note 2:** the pin, which is connected to the first push button (B0) should be configured as an input in order to check when it is pressed.
Note 3: you might need to do the de-bounce operation when the push button is pressed. Instead of `usleep` command (which is not available in kernel module), you can use `rt_sleep()` command.

**Part 3: Main Pthread and Dynamic Pthreads**

In this part you need to modify the pthread that you created in part 1. The purpose of this main pthread is to receive data from the kernel module through the FIFO and to read the last GPS data and time-stamp from the common buffer, which is being updated by the main program.

In part 4 we will do an interpolation of the GPS data corresponding to the push button event. Therefore, here we need to wait for the next GPS event and save the corresponding data and time-stamp. The final data consists of the real time event’s time-stamp, the previous and the next GPS data and their corresponding time-stamps.

**Question:** What is the problem if the messages are not 1:1? In other words, what happens if there is more than one push button event between two consecutive GPS events? We must be able to handle those multiple push button events. Fix the thread in order to read 1:2, 1:3, ... and 1:n data.

**Part 4: Interpolation and Display**

A printing thread should be created to display the final output for each push button event. The final data (the real time event’s time-stamp, the previous and the next GPS data and their corresponding time-stamps) are to be received by this printing thread through a named pipe. Using the final data, you will interpolate the GPS value corresponding to the push button event. Finally, this thread will print: the time-stamp and value of the GPS event prior to the push button event, the time-stamp and the (estimated) GPS value of the push button event, and the time-stamp and value of the GPS event after the push button event.

**NOTE 1:** Before running your user space program, make sure that your kernel module has been installed.

**NOTE 2:** In Figure 1, a basic flowchart of the program is shown. You might need more pthreads, buffers, semaphores, etc. in order to complete the assignment.

**Post Lab Questions**

1. Imagine that you want to do the interpolation with more GPS points in order to have more accurate result. What would be changed in your program? What kind of buffer would you need? Why?

2. When your user space program reads from the FIFO, is this a blocking operation or non-blocking operation? How do you know?
Figure 1. A basic flowchart of the program