LAB4, Non-Preemptive Multi-Threading

Goal

Work with multi-threaded programs. Understand non preemption.

Given files

Assuming you copied the lab files according to the lab setup instruction you can get the given files with this command:

```
cp -r /home/TDDI11/lab/skel/lab4 ~/TDDI11/lab4
```

Your modifications goes to `main.c` and perhaps minor adjustments in the other files to make room for a window displaying the timer on the screen. You must reuse your serial implementation from previous assignment. Use `gmake` to compile.

Assignment

Sometimes the work performed by one program can be separated into two or more relatively independent “threads”. The processing of inbound and outbound data in the (previous) serial communications program for example, could be easily separated into two independent threads. In addition, a third thread could be added to continuously update elapsed time on the display. When an application can be partitioned in this way, the software is easier to design, debug, and maintain.

The source code for this program was derived from the previous serial communications program by converting it into a multi-threaded application using the Multi-C non-preemptive (or cooperative) kernel. It is said to be cooperative as all threads must cooperate. Every thread must explicitly give time for other threads to get execution time. The thread programmer (you) must make sure to switch to other thread every now and then.

In its present form, the code contains only two threads: one to process inbound data and one to process outbound data. Your job is to add a third thread to display elapsed time (measured from program start). The pseudo-code for that thread is shown below:

```
void DisplayElapsedTime(void)
{
     thread-specific initialization (if any) goes here;

     for ( ; ; )
     {
          compute hh:mm:ss;
          display hh:mm:ss;
          timer := 1 second;

          while (!timed out)
          {
               yield to other threads;
          }
     }
}
```

Note that the elapsed time thread must release the processor whenever it is idle by explicitly calling the function `MtCYield`.

The main program must be modified to “launch” the new thread. In Multi-C this is accomplished by calling the thread function (with any required parameters) inside a macro called `MtCCoroutine`, as in: `MtCCoroutine(DisplayElapsedTime());`

Deliverables

Code and demo for the lab assistant.