Politecnico Di Torino

Department of CONTROL AND COMPUTER ENGINEERING (DAUIN)

Master Degree in Mechatronic Engineering

2024/2025

Operating Systems for Embedded Systems - Prof. Violante

Laboratory 01 – task and alarms

Wednesday, October 15, 2024

Francesca Fornasier – 338688

Exercise 1

OIL file description

```
OIL_VERSION = "2.5";

IMPLEMENTATION trampoline
{
    /* This fix the default STACKSIZE of tasks */
    TASK {
        UINT32 STACKSIZE = 32768 ;
    };

    /* This fix the default STACKSIZE of ISRs */
    ISR {
        UINT32 STACKSIZE = 32768 ;
    };
};

CPU only_one_periodic_task {
    OS config {
        STATUS = EXTENDED;
        TRACE = TRUE {
            FORMAT = json;
            PROC = TRUE;
            RESOURCE = TRUE;
            ALARM = TRUE;
            EVENT = TRUE;
        };
        BUILD = TRUE {
            APP_SRC = "ex1.c";
            TRAMPOLINE_BASE_PATH = "/Users/francescafornasier/github/Operating_Systems_for_Embedded_Systems/trampoline";
            CFLAGS="-ggdb";
            APP_NAME = "ex1_exe";
            LINKER = "gcc";
            SYSTEM = PYTHON;
        };
    };

    APPMODE stdAppmode {};
```

At the beginning of the OIL file there are some directives for the OIL compiler to correctly build the executable from the source code.

```
    ALARM a500msec {
        COUNTER = SystemCounter;
        ACTION = ACTIVATETASK { TASK = TaskA; };
        AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME = 50; CYCLETIME = 50; };
    };

    ALARM a750msec {
        COUNTER = SystemCounter;
        ACTION = ACTIVATETASK { TASK = TaskB; };
        AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME = 150; CYCLETIME = 75; };
    };

    ALARM stopper {
        COUNTER = SystemCounter;
        ACTION = ACTIVATETASK { TASK = stop; };
        AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME = 600; CYCLETIME = 0; };
    };
```

Definition of three alarms

- a500msec: is that expires after 50 ms (ALARMTIME = 50) and then is triggered every 500 ms (CYCLETIME = 50). Its action is to activate the periodic task “TaskA”
- a750msec: it activates TaskB every 750 ms, starting with an initial delay of 1500 ms with respect to the starting of the operating system.
- stopper is a one shot alarm that expires after 6000 ms and activates the task stop.

All three alarms have the property AUTOSTART = TRUE, so they start at the starting of the operating system

```
    TASK TaskA {
        PRIORITY = 2;
        AUTOSTART = TRUE { APPMODE = stdAppmode; };
        ACTIVATION = 1;
        SCHEDULE = FULL;
    };

    TASK TaskB {
        PRIORITY = 1;
        AUTOSTART = FALSE;
        ACTIVATION = 1;
        SCHEDULE = FULL;
    };

    TASK stop {
        PRIORITY = 99;
        AUTOSTART = FALSE;
        ACTIVATION = 1;
        SCHEDULE = FULL;
    };
};

```

TaskA is a periodic task with priority 2, AUTOSTART = TRUE so it is activated for the first time at the starting of execution of the code, it is immediately set as ready

TaskB is periodic with priority 1, property AUTOSTART = FALSE so it is triggered when the a750msec alarm expires (after 1500ms and then every 750ms)

Task stop is a one shot task with priority 99 activated by stopper alarm.

The greater are the values of the property PRIORITY, the higher is the priority of the task.
For all the tasks we have the schedule set to FULL, meaning that they are fully preemptible, and we have the maximum number of concurrent instances that can be activated equal to 1.

C file description

```c
#include <stdio.h>
#include "tpl_os.h"
```

Include standard libraries and file “tpl_os.h” which is a file created by the OIL compiler and contains the data structures we have in our application

```c
int main(void){
    StartOS(OSDEFAULTAPPMODE);
    return 0;
}
```

Inside the main we only start the operating system

```c
DeclareAlarm(a500msec);
DeclareAlarm(a750msec);
```

Declaration of the alarms we defined in the OIL file

```c
int deltaA = 500;
int deltaB = 750;
```

Definition of 2 global variables containing the value of which the counters has to be incremented at every execution of the tasks

```
15  TASK(TaskA){
16      static unsigned int counterA = 0;
17      printf("TaskA: %d\n\r", counterA);
18      counterA+=deltaA;
19      TerminateTask();
20  }
```

Implementation of TaskA: definition of the variable counter, declared as static since it is needed to maintain its value between different executions of the task, initialised at 0 as requested.
Print of the current value of the counter an then increment it of deltaA defined above in the code. Then terminate the task.

```
22  TASK(TaskB){
23      static unsigned int counterB = 1500;
24      printf("TaskB: %d\n\r", counterB);
25      counterB+=deltaB;
26      TerminateTask();
27  }
```

Implementation of TaskB: as for TaskA, definition of the variable counter as static so that the value is maintained, initialised at 1500.
Print of the current value of the counter, then increment it of deltaB defined above in the code and terminate the task.

```
29  TASK(stop){
30      CancelAlarm(a500msec);
31      CancelAlarm(a750msec);
32      printf("Shutdown\n\r");
33      ShutdownOS(E_OK);
34      TerminateTask();
35  }
```

Implementation of stop: this task cancels the alarms set for TaskA and TaskB, prints a message and shuts down the operating system. Then terminate the task.

Output

```
francescafornasier@MacBook-Pro-di-Francesca es01 % ./ex1_exe
TaskA: 0
TaskA: 500
TaskA: 1000
TaskA: 1500
TaskB: 1500
TaskA: 2000
TaskB: 2250
TaskA: 2500
TaskA: 3000
TaskB: 3000
TaskA: 3500
TaskB: 3750
TaskA: 4000
TaskA: 4500
TaskB: 4500
TaskA: 5000
TaskB: 5250
TaskA: 5500
Shutdown
Exiting virtual platform.
francescafornasier@MacBook-Pro-di-Francesca es01 %
```

Exercise 2

OIL file description

Same application but developed to run on ArduinoUno, so most of the code is the same as the previous exercise.

```
OIL_VERSION = "2.5" : "test" ;

CPU only_one_periodic_task {
  OS config {
    STATUS = STANDARD;
    BUILD = TRUE {
      TRAMPOLINE_BASE_PATH = "/Users/francescafornasier/github/Operating_Systems_for_Embedded_Systems/trampoline";
      APP_NAME = "lab01ex2";
      APP_SRC = "ex2.cpp";
      CPPCOMPILER = "avr-g++";
      COMPILER = "avr-gcc";
      LINKER = "avr-gcc";
      ASSEMBLER = "avr-gcc";
      COPIER = "avr-objcopy";
      SYSTEM = PYTHON;
      LIBRARY = serial;
    };
    SYSTEM_CALL = TRUE;
  };

  APPMODE stdAppmode {};
```

LIBRARY = serial is added to the directives for the OIL compiler in order to include the libraries for the communications over the serial port and use the serial monitor

```
  ALARM a500msec {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = TaskA; };
    AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME = 500; CYCLETIME = 500; };
  };

  ALARM a750msec {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = TaskB; };
    AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME = 1500; CYCLETIME = 750; };
  };

  ALARM stopper {
    COUNTER = SystemCounter;
    ACTION = ACTIVATETASK { TASK = stop; };
    AUTOSTART = TRUE { APPMODE = stdAppmode; ALARMTIME = 6000; CYCLETIME = 0; };
  };
```

Since we are in Arduino, the values of the parameters of the alarms have to be in milliseconds

```
  TASK TaskA {
    PRIORITY = 2;
    AUTOSTART = TRUE { APPMODE = stdAppmode; };
    ACTIVATION = 1;
    SCHEDULE = FULL;
  };

  TASK TaskB {
    PRIORITY = 1;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
    SCHEDULE = FULL;
  };

  TASK stop {
    PRIORITY = 99;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
    SCHEDULE = FULL;
  };
};
```

The tasks are defined in the very same way as in the previous application

CPP file description

```
#include "tpl_os.h"
#include "stdio.h"
#include "stdlib.h"
#include "Arduino.h"
```

In addition to the “tpl_os.h” containing the data structures of our application, also “Arduino.h” file is needed to execute it

```
void setup()
{
    Serial.begin(115200);
    // initialize digital pin 13 as an output.
    pinMode(13, OUTPUT);
    StartOS(OSDEFAULTAPPMODE);
}
```

Inside the setup function we initialise the communication over the serial port and initialise pin 13 as output

```
DeclareAlarm(a500msec);
DeclareAlarm(a750msec);
```

We declare the alarms defined in the OIL file

```
int deltaA = 500;
int deltaB = 750;
```

Set two global variables for the increment values

```
TASK(TaskA)
{
    static unsigned int counterA = 0;
    digitalWrite(13, HIGH);
    Serial.print("TaskA ");
    Serial.println(counterA);
    counterA+=deltaA;
    TerminateTask();
}
```

In the implementation of TaskA we define the static variable counterA (persistent) and initialise it with value 0. We set the output pin 13 to HIGH and print a message to keep track of which task is executing and the actual value of its counter.
Then we increment the counter with the value deltaA and terminate the task.

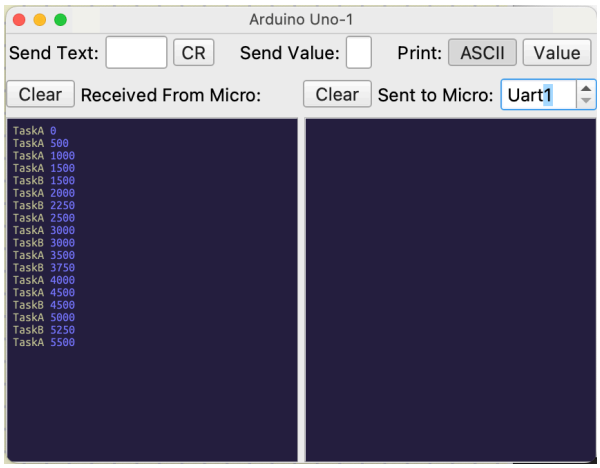
```
TASK(TaskB)
{
    static unsigned int counterB = 1500;
    digitalWrite(13, LOW);
    Serial.print("TaskB ");
    Serial.println(counterB);
    counterB+=deltaB;
    TerminateTask();
}
```

In the implementation of TaskB we define the static variable counterB and initialise it with 1500. We set the output pin 13 to LOW and print a message with “TaskB” and the actual value of the counter, then we increment the counter with the value deltaA and terminate the task.

```
TASK(stop){
    CancelAlarm(a500msec);
    CancelAlarm(a750msec);
    digitalWrite(13, LOW);
    ShutdownOS(E_OK);
    TerminateTask();
}
```

Task stop cancels the alarm that periodically activate TaskA and TaskB, turns off the LED setting pin 13 to LOW then shuts down the operating system and terminates.

Output


Arduino Uno-1
Send Text:
CR
Send Value:
Print:
ASCII
Value
Clear
Received From Micro:
Clear
Sent to Micro:
Uart1
TaskA 0
TaskA 500
TaskA 1000
TaskA 1500
TaskB 1500
TaskA 2000
TaskB 2250
TaskA 2500
TaskA 3000
TaskB 3000
TaskA 3500
TaskB 3750
TaskA 4000
TaskA 4500
TaskB 4500
TaskA 5000
TaskB 5250
TaskA 5500