LAB 01

Trampoline is a Free Software implementation of the OSEK/VDX specification. Trampoline includes an OIL compiler which allows, starting from an OIL description, to generate OS level data structures of the application. In addition to the OIL description, the developer must provide the C sources of tasks and ISRs of the application.

After downloading and correctly installing the Trampoline RTOS[1] we then proceeded the following tasks:

1) Added two tasks in the system: task_0 and task_1.

- task_0 should have priority 1 and its AUTOSTART attribute should be set to FALSE;
- task_1 should have priority 8 and its AUTOSTART attribute should be set to FALSE;
- each task prints its name on a line of the LCD and then terminates.

Why does task_1 execute before task_0 whereas it has been activated after?

The reason is that the priority of task_0 (that is equal to 1) is lower than the priority of task_1 (that is equal to 8).

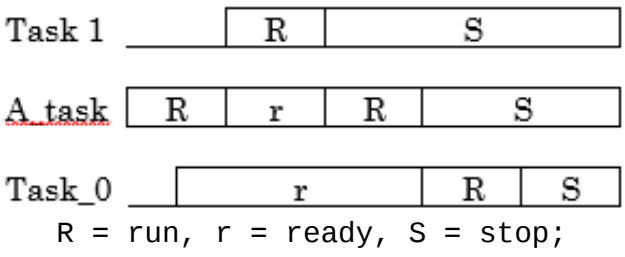
Snippet of .cpp

```
TASK(a_task)
{
    ActivateTask(task_0);
    ActivateTask(task_1);
    TerminateTask();
}
```

Snippet of .oil

```
  TASK task_0 {
        PRIORITY = 1;
        AUTOSTART = FALSE;
        ACTIVATION = 1;
        SCHEDULE = FULL;
    };

    TASK task_1 {
        PRIORITY = 8;
        AUTOSTART = FALSE;
        ACTIVATION = 1;
        SCHEDULE = FULL;
    };
```

The gantt chart

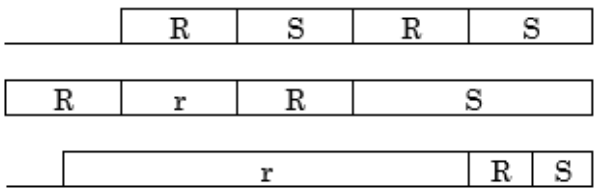


The output on the board

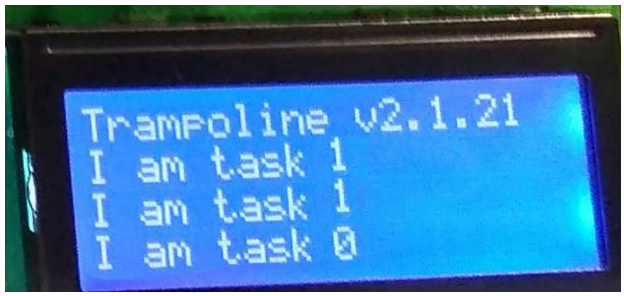
1 Real time operating system

2) Replaced the call to TerminateTask by a ChainTask(task_1) at the end of task a task

Snippet of .cpp	The gun chart

TASK(a_task)
{
ActivateTask(task_0);
ActivateTask(task_1);
ChainTask(task_1);
}
	
	R = run, r = ready, S = stop;

What is happening?



A_task starts,

A_task activates Task_0 (it will be in ready because it has the same priority as A_task)

A_task activates Task_1 (A_task will be preempted, Task_1 runs)

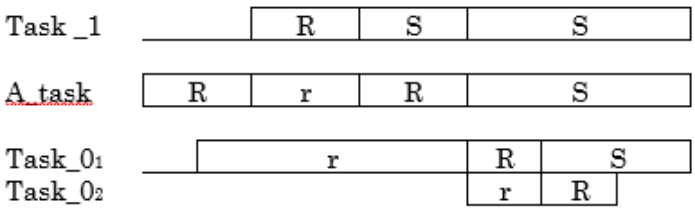
Task_1 terminates, A_task resumes.

A_task terminates and it activate Task_1; Task_1 runs because it has higher priority than Task_0.

Task_1 terminates, Task_0 runs.

3) Chain to task_0 instead of task_1. Draw a schedule of the execution of the system. What is happening?

Snippet of .cpp	The gun chart

TASK(a_task)
{
ActivateTask(task_0);
ActivateTask(task_1);
ChainTask(task_0);
}
	
	R = run, r = ready, S = stop;

A_task starts,

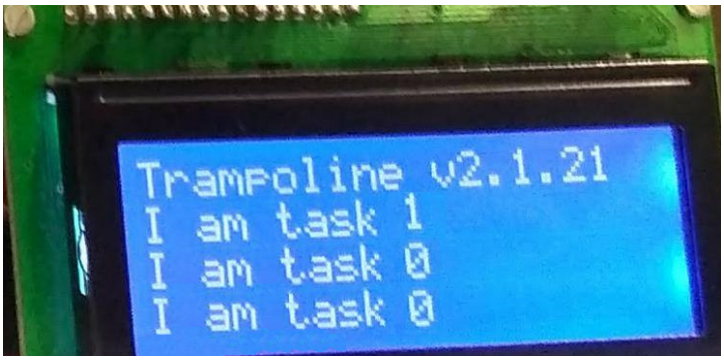
A_task activates $Task\_0_1$ (an instance of Task_0, it will be in ready because it has the same priority as A_task)

A_task activates Task_1 (A_task will be preempted and Task_1 will runs).

Task_1 terminates, A_task resumes.

A_task terminates and it activate Task_0_2 (a new instance of Task_0, it will be in ready state because the there is another task in the ready list that will run first), Task_0_1 runs.

Task_0_1 terminates, Task_0_2 runs.



Question 5

- set priority of task_0 to 8;
- add two events, evt_0 and evt_1:

– evt_0 is set by task a_task to tasktask_0

– evt_1 is set by task a_task to tasktask_1

- modify the body of the tasks:

– task a_task activates task_0 and task_1 then sets evt_0 and evt_1 before to terminates

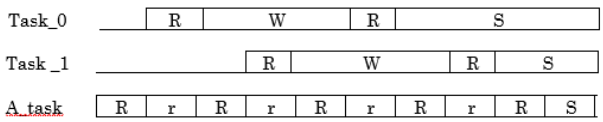
– task task_0 and task_1 wait for their event, clear it, and terminate.

Snippet of .cpp	Snippet of .oil

`DeclareEvent(evt_0);`
`DeclareEvent(evt_1);`
 	
`[…]`
 	
`TASK(task_0)`
`{`
`  lcd.print("a0 ");`
`  WaitEvent(evt_0);`
`  lcd.print("b0 ");`
`  TerminateTask();`
`}`
 	
`TASK(task_1)`
`{`
`  lcd.print("a1 ");`
`  WaitEvent(evt_1);`
`  lcd.print("b1 ");`
`  TerminateTask();`
`}`
 	
`TASK(a_task)`
`{`
`  lcd.print("c ");`
`  ActivateTask(task_0);`
`  lcd.print("d0 ");`
`  ActivateTask(task_1);`
`  lcd.print("d1 ");`
	`  EVENT evt_0 {`
	`  MASK = AUTO;`
	`  };`
	
	`  EVENT evt_1 {`
	`  MASK = AUTO;`
	`  };`
	
	`[…]`
	
	`  TASK task_0 {`
	`  PRIORITY = 8;`
	`  AUTOSTART = FALSE;`
	`  ACTIVATION = 1;`
	`  SCHEDULE = FULL;`
	`  EVENT = evt_0;`
	`  };`
	
	`  TASK task_1 {`
	`  PRIORITY = 8;`
	`  AUTOSTART = FALSE;`
	`  ACTIVATION = 1;`
	`  SCHEDULE = FULL;`
	`  EVENT = evt_1;`
	`  };`

```
    SetEvent(task_0, evt_0);
    lcd.print("e0 ");
    SetEvent(task_1, evt_1);
    lcd.print("e1 ");

    ChainTask(task_1);
}
```

The gantt chart



R = run, r = ready,
S = stop, W = wait;

The output on the board

Question 6 Program an application conforming to the following requirements:

- it is composed of two tasks: server priority 2, t1 priority 1.
- server is an infinite loop that activates t1 and waits for event evt_1.
- t1 prints “I am t1” and sets evt_1 of server.

Before to run the application, draw a schedule of the execution. Add outputs in the bodies of the task (for instance writes to the LCD or the LEDs) to verify your schedule.

Snippet of .cpp

```
TASK(t1)
{
    digitalWrite(3,HIGH);
    SetEvent(server, ev_1);
    TerminateTask();
}

TASK(server)
{
    while (TRUE) {
    ActivateTask( t1 );
    WaitEvent(ev_1 );
    lcd.print("received ev_1, ");
    digitalWrite(3,LOW);
    digitalWrite(4,HIGH);
    }
```

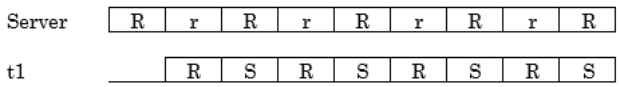
Snippet of .oil

```
EVENT ev_1 {
        MASK = AUTO;
    };

    TASK server {
        PRIORITY = 2;
        AUTOSTART = TRUE
{ APPMODE = stdMode; };
        ACTIVATION = 1;
        SCHEDULE = FULL;
        EVENT = ev_1;
    };

    TASK t1 {
        PRIORITY = 1;
        AUTOSTART = FALSE;
        ACTIVATION = 1;
```

```
    TerminateTask();
}
```

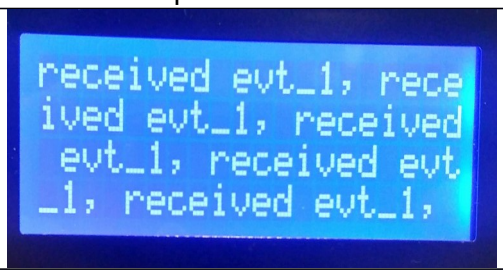
```
        SCHEDULE = FULL;
    };
```

The gantt chart



R = run, r = ready, S = stop;

The output on the board



Question 7 Extend the previous application by adding 2 tasks: t2 and t3 (priority 1 for both) and 2 events evt_2 and evt_3. server activates t1, t2 and t3 and waits for one of the events. When one of the events is set, server activates the corresponding task again.

Snippet of .cpp

```
DeclareEvent(ev_1);
DeclareEvent(ev_2);
DeclareEvent(ev_3);
DeclareTask(server);

[…]

TASK(t1)
{
    lcd.print("s1  ");
    digitalWrite(4,LOW);
    digitalWrite(5,LOW);
    digitalWrite(6,LOW);
    digitalWrite(3,HIGH);
    SetEvent(server, ev_1);
    lcd.print("f1 ");
    TerminateTask();
}

TASK(t2)
{
    lcd.print("s2  ");
    digitalWrite(3,LOW);
    digitalWrite(5,LOW);
    digitalWrite(6,LOW);
    digitalWrite(4,HIGH);
    SetEvent(server, ev_2);
    lcd.print("f2 ");
    TerminateTask();
}
```

Snippet of .oil

```
EVENT ev_1 {
        MASK = AUTO;
    };

    EVENT ev_2 {
        MASK = AUTO;
    };

    EVENT ev_3 {
       MASK = AUTO;
    };

    TASK server {
        PRIORITY = 2;
        AUTOSTART = TRUE
{ APPMODE = stdMode; };
        ACTIVATION = 1;
        SCHEDULE = FULL;
        EVENT = ev_1;
        EVENT = ev_2;
        EVENT = ev_3;
    };


    TASK t1 {
        PRIORITY = 1;
        AUTOSTART = FALSE;
        ACTIVATION = 1;
        SCHEDULE = FULL;
    };

    TASK t2 {
```

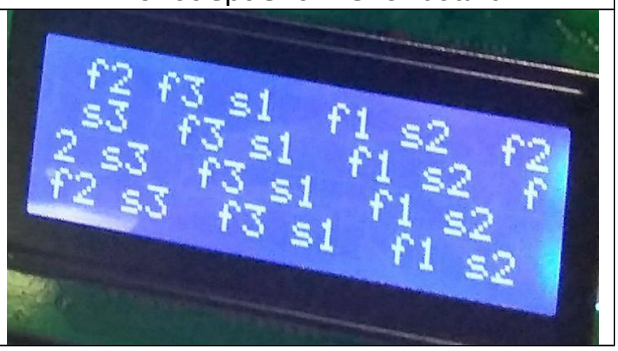
```
TASK(t3)
{
    lcd.print("s3  ");
    digitalWrite(4,LOW);
    digitalWrite(3,LOW);
    digitalWrite(6,LOW);
    digitalWrite(5,HIGH);
    SetEvent(server, ev_3);
    lcd.print("f3 ");
    TerminateTask();
}

TASK(server)
{
    while (TRUE) {

        ActivateTask( t1 );
        ActivateTask( t2 );
        ActivateTask( t3 );
        digitalWrite(6,HIGH);

        EventMaskType event_got, x;

     WaitEvent(ev_1 | ev_2 | ev_3);
     GetEvent(server, &event_got);
        x = event_got;
        ClearEvent(x);
        if (event_got & ev_1) {
            ActivateTask( t1 );
        }
        if (event_got & ev_2) {
            ActivateTask( t2 );
        }
        if (event_got & ev_3) {
            ActivateTask( t3 );
        }
      }
    TerminateTask();
}
```

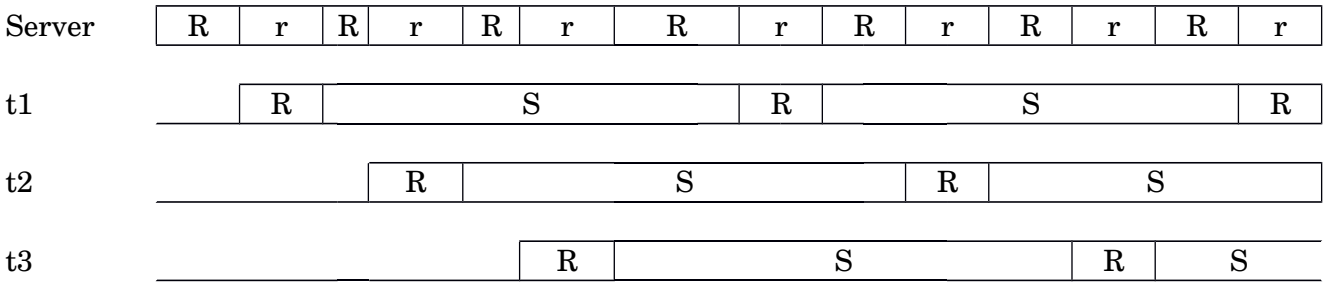
```
    PRIORITY = 1;
    AUTOSTART = FALSE;
    ACTIVATION = 1;
    SCHEDULE = FULL;
    };

    TASK t3 {
        PRIORITY = 1;
        AUTOSTART = FALSE;
        ACTIVATION = 1;
        SCHEDULE = FULL;
    };
```

The gantt chart

R = run, r = ready, S = stop;

The output on the board




Server
R r R r R r R r R r R r R r
t1
R S R S R
t2
R S R S
t3
R S R S