Lab #3
⋆
Shared object access protection

March 22, 2013

Note: All the software and documents are stored at `http://www.irccyn.ec-nantes.fr/~bechenne/trampoline_isis`

1 Goal

To show resources usage, we will use a bad program that allows to corrupt a shared global variable which is not protected against concurrent writes. This has been presented in the course. This lab will show different ways to prevent this wrong behavior by using resources (standard and internal) or other solutions (preemption and priority).

Get the lab3 archive from `http://www.irccyn.ec-nantes.fr/~bechenne/trampoline_isis` and put it into the `trampoline/labs_isis/` directory. The function needed in Question 7 is in `lab3.c`.

2 Application requirements

The application has 3 tasks and 2 **volatile** global variables: `val` and `activationCount` as shown in figure 1:

- a background task called `bgTask`, active at start (`AUTOSTART = TRUE`) that never ends. In an infinite loop this task increments then decrements the global variable `val`. This task has a priority equal to 1.
- a periodic task called `periodicTask`, priority 10, that runs every 100ms[1]. This periodic task increments the global variable `activationCount` which is initialized to 0 at start. Then if `activationCount` is odd, `val` is incremented, otherwise it is decremented.

[1] a counter gets a tick every 1ms

- a periodic task `displayTask`, priority 20, that runs every second and prints on the lcd `val` and `activationCount`.

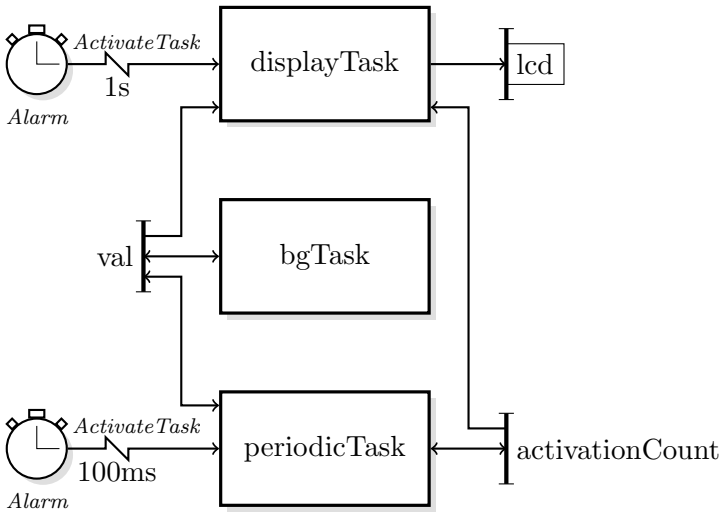

Figure 1: Application diagram

Question 1 *Before programming the application, gives the values that should be displayed for val on the standard output.*

Describe the application in OIL and program it in C.

Question 2 *Does the behavior correspond to what you expect ? Why ?*

3 Global variable protection

Question 3 *Update the OIL file and the C program to protect the access to the global variable* `val`*. Use a resource to do it.*

The resource priority is automatically computed by goil according to the priorities of the tasks which use it.

Question 4 *What priority will be given to the resource ?*

The OIL compiler (goil) generates many files in the directory bearing the same name as the oil file (less the .oil suffix). Among them 3 are interesting:

- `tpl_app_define.h`
- `tpl_app_config.h`
- `tpl_app_config.c`

The file `tpl_app_config.c` contains the tasks' descriptors as long as all other data structures. These structures are commented.

Question 5 *For each task, find the priority computed by goil and the identifier. Is is the same as defined in the OIL file? if not is it aproblem?*

Question 6 *What is the priority of the resource ? Is it compliant with the PCP rule ?*

PCP rule requires the task priority is raised to the resource priority when the resource is taken. We will show this behavior by displaying the priority of the currently running task. Since OSEK does not have a function to do that, we use the following function:

```
void displayIdAndCurrentPriority()
{
    TaskType id;
    GetTaskID(&id);
    if (id >= 0)
    {
        tpl_priority prio = tpl_dyn_proc_table[id]->priority;
        lcd_print_char(':');
        lcd_print_unsigned(id);
        lcd_print_char(',');
        lcd_print_unsigned(prio);
    }
}
```

And you have to add the following line at start of your C file:

```
#include "tpl_os_task_kernel.h"
```

Call the function from your tasks when the resource is not taken and when it is taken.

Question 7 *Is the behavior ok ? Explain.*

4 Protection with an internal resource

An internal resource is automatically taken when the task gets the CPU. Replace the standard resource by an internal resource in the OIL file. Remove the `GetResource` and `ReleaseResource` in the C file.

Question 8 *What happens ? Why ?*

Modify the task `bgTask`: instead of infinite loop, use a `ChainTask` to the `bgTask` (ie: the task chains to itself).

Question 9 *Question 10 What happens ? Explain.*

5 Protection using a single priority level

Keep the version with the `ChainTask` instead of the infinite loop. Modify the OIL file: remove the resource and set the priorities so that no task can be preempted.

Question 10 *What happens ? Why ?*