REAL TIME SYSTEMS

Ecole Centrale de Nantes

LAB 3 REPORT

REPORT BY

Gopi Krishnan Regulan

Ragesh Ramachandran

Ankit Tripathi

December 2017

Goal:

The lab will show different ways to prevent the wrong behavior (using a bad program that allows to corrupt a shared global variable which is not protected against concurrent writes) by using resources (standard and internal) or other solutions (preemption and priority).

Application Requirements:

The application has 3 tasks and 2 global variables – *bgTask, periodicTask* and *displayTask, val* and *activationCount.*

Question 16:

According to the given conditions, **the variable *val* should be always zero** but the variable *activationCount* incremented every time by 1.

Question 17:

No, the behavior does not correspond to what expected. The value of *val* is increased to 1.

Global variable protection

Question 18:

Find the Fig. 1 and Fig. 2 below for the .cpp file and the .oil file respectively.

Question 19:

The resource is given the priority 4 which is max + 1, where max is the highest priority given to any task. In our case, max priority = 3 of *periodicTask.*

Question 20:

The priority computed by goil and the identifier are as follows:

- *bgTask* - 1
- *periodicTask* - 3
- *displayTask* - 2

Yes, it is the same as defined in the OIL file.

Question 21:

The priority of the resource is 4. Yes, it is consistent.

Question 22:

Yes. The resource is given the priority 4 which is max + 1, where max is the highest priority given to any task. In our case, max priority = 3 of *periodicTask.*

```
TASK(bgTask) {
    while(1) {
        GetResource(resA);
        val++; //increment
        val--; //decrement
        if( i == 0) {
            lcd.clear();
            displayIdAndCurrentPriority();
        }
        ReleaseResource(resA);
        if( i == 0) {
            displayIdAndCurrentPriority();
            i++;
        }
    }
    TerminateTask();
}

TASK(periodicTask) {
    activationCount++;
    GetResource(resA);
    if( j == 0) {
            displayIdAndCurrentPriority();
    }
    if(activationCount%2==1) {
        val++;
    }
    else {
        val--;
    }
    ReleaseResource(resA);
    if( j == 0) {
            displayIdAndCurrentPriority();
            j++;
    }
    TerminateTask();
}

TASK(displayTask) {
    if( k == 0) {
            displayIdAndCurrentPriority();
            k++;
    }
    TerminateTask();
};
```

Fig. 1 Global variable protection .cpp file

```
TASK bgTask {
  PRIORITY = 1;
  AUTOSTART = TRUE { APPMODE = stdMode; };
  ACTIVATION = 1;
  SCHEDULE = FULL;
  RESOURCE = resA;
};

TASK periodicTask {
  PRIORITY = 3;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  RESOURCE = resA;
};

TASK displayTask {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  RESOURCE = resA;
};

ALARM run_periodicTask {
  COUNTER = SystemCounter;
  ACTION = ACTIVATETASK {
    TASK = periodicTask;
  };
  AUTOSTART = TRUE {
    APPMODE = stdMode;
    ALARMTIME = 10;
    CYCLETIME = 10;
  };
};

ALARM run_displayTask {
  COUNTER = SystemCounter;
  ACTION = ACTIVATETASK {
    TASK = displayTask;
  };
  AUTOSTART = TRUE {
    APPMODE = stdMode;
    ALARMTIME = 100;
    CYCLETIME = 100;
  };
};
```

Fig. 2 Global variable protection .oil file

Protection with an internal resource

Question 23:

Now the *GetResource* and *ReleaseResource* are removed from the C file. Due to infinite *while* loop, *bgTask* never preempted because it's having the highest priority. So, nothing is printed on the LCD screen. It keeps on running since the priority is high compared to the other tasks.

```
#define APP_Task_bgTask_START_SEC_CODE

volatile int val = 0;
volatile int activationCount = 0;

DeclareTask(bgTask);
DeclareTask(displayTask);
DeclareTask(periodicTask);
DeclareAlarm(run_displayTask);
DeclareAlarm(run_periodicTask);
DeclareResource(resA);

TASK(bgTask) {
    while(1) {
        val++; //increment
        val--; //decrement
    }
}

TASK(periodicTask) {
    activationCount++;
    if(activationCount%2==1){
        val++; //increment
    } else{
        val--; //decrement
    }
    TerminateTask();
}

TASK(displayTask) {
    lcd.print("val=");
    lcd.println(val);
    lcd.setCursor(0,1);
    lcd.println("activationCount=");
    lcd.print(activationCount);
    TerminateTask();
};
```

Fig.3 Protection with an internal resource – Question 23 .cpp file

```
COUNTER SystemCounter{
 TICKSPERBASE=10;
 MAXALLOWEDVALUE = 50000;
 MINCYCLE =1;

};

RESOURCE resA {
 RESOURCEPROPERTY = INTERNAL;
};

TASK bgTask {
  PRIORITY = 1;
  AUTOSTART = TRUE { APPMODE = stdMode; };
  ACTIVATION = 1;
  SCHEDULE = FULL;
  RESOURCE = resA;
};

TASK periodicTask {
  PRIORITY = 3;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  RESOURCE = resA;
};

TASK displayTask {
  PRIORITY = 2;
  AUTOSTART = FALSE;
  ACTIVATION = 1;
  SCHEDULE = FULL;
  RESOURCE = resA;
};

ALARM run_periodicTask {
  COUNTER = SystemCounter;
  ACTION = ACTIVATETASK {
    TASK = periodicTask;
  };
  AUTOSTART = TRUE {
    APPMODE = stdMode;
    ALARMTIME = 10;
    CYCLETIME = 10;
  };
};

ALARM run_displayTask {
  COUNTER = SystemCounter;
  ACTION = ACTIVATETASK {
    TASK = displayTask;
  };
  AUTOSTART = TRUE {
    APPMODE = stdMode;
    ALARMTIME = 100;
```

Fig.4 Protection with an internal resource – Question 23 .oil file

Question 24:

When infinite while loop is replaced by ChainTask it changes the way priority of tasks been calculated. Now priority of all the tasks is set to 4 because it's the priority of the internal resource because of which all the tasks run as expected and value of variable v always remains 0.

```
DeclareTask(bgTask);
DeclareTask(displayTask);
DeclareTask(periodicTask);
DeclareAlarm(run_displayTask);
DeclareAlarm(run_periodicTask);
DeclareResource(resA);

TASK(bgTask) {
    val++; //increment
    val--; //decrement
    ChainTask(bgTask);
}

TASK(periodicTask) {
    activationCount++;
    if(activationCount%2==1){
        val++; //increment
    } else{
        val--; //decrement
    }
    TerminateTask();
}

TASK(displayTask) {
    lcd.print("val=");
    lcd.println(val);
    lcd.setCursor(0,1);
    lcd.println("activationCount=");
    lcd.print(activationCount);
    TerminateTask();
};
```

Fig.5 Protection with an internal resource – Question 24 .cpp file

Question 25:

Now the priorities of all the tasks are set as 1. So it behaves in the same way as in the ChainTask case. Now priority of all the tasks is set to 1 because it's defined in OIL file and all the tasks run as expected and value of variable v always remains 0.

```
TASK bgTask {
  PRIORITY = 1;
  AUTOSTART = TRUE { APPMODE = stdMode; };
  ACTIVATION = 1;
  SCHEDULE = FULL;
};

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

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

Fig.6 Protection using a single priority level .oil file