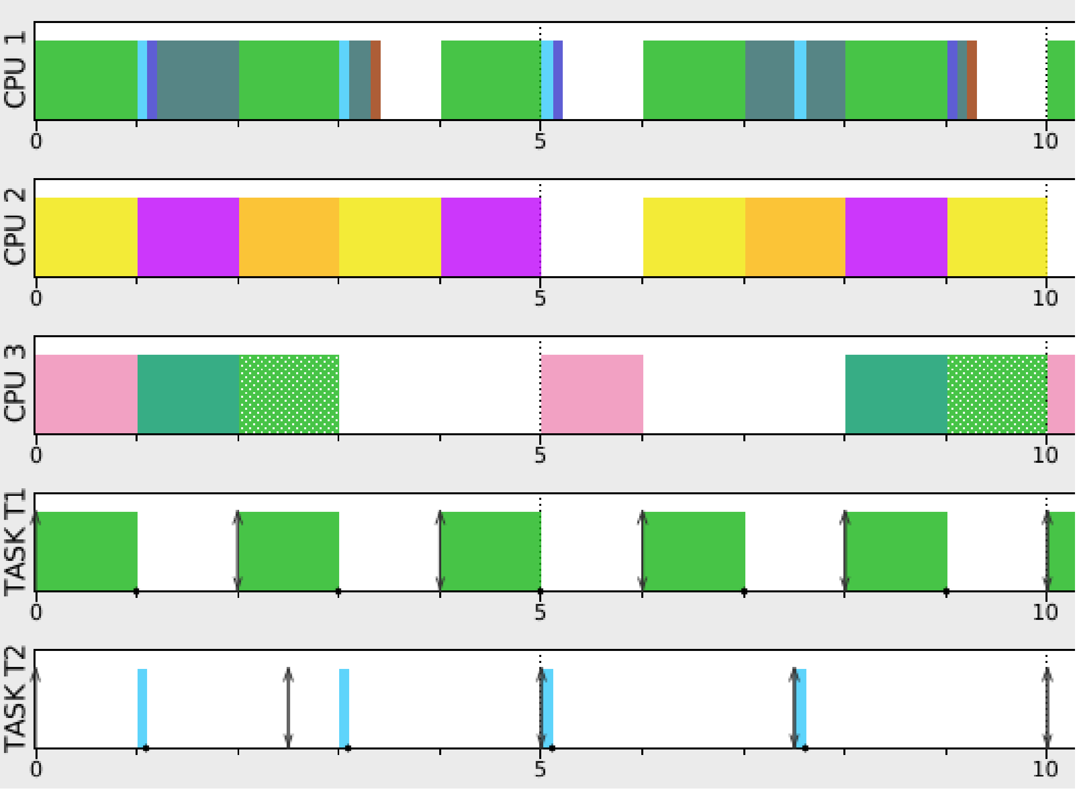
Simulation assignment

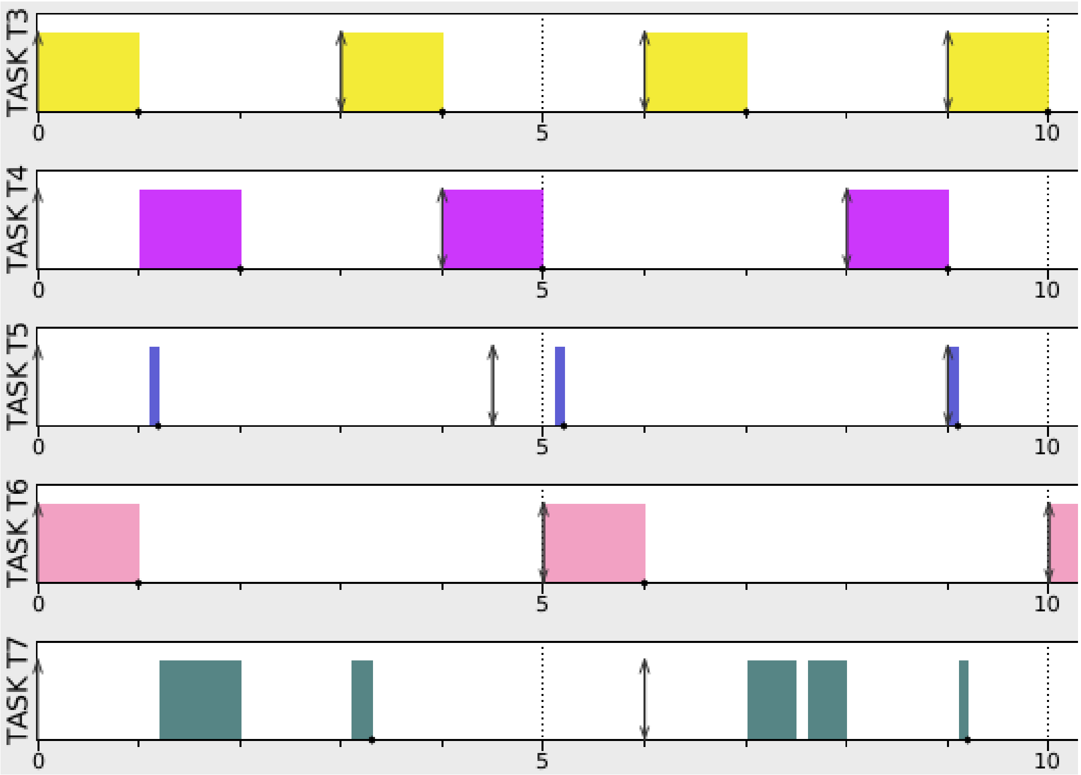
- CT1(2,1) T2(2.5,0.1) T3(3,1) T4(4,1) T5(4.5,0.1) T6(5,1) T7(6,1) T8(7,1) T9(8,1) T10(8.5,0.1) T11(9,1)

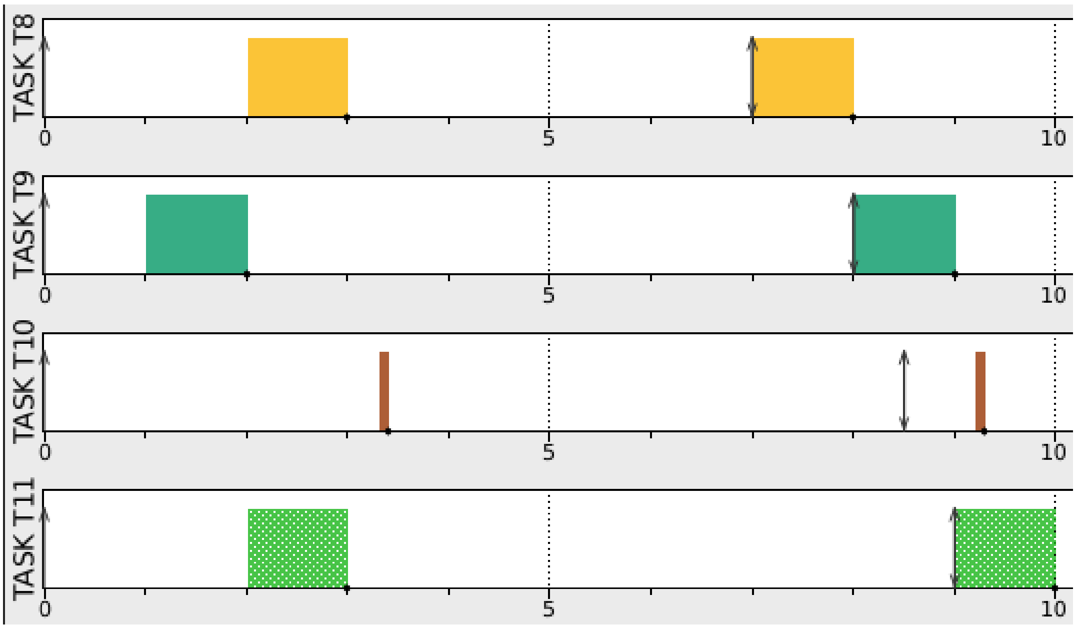
ID	P,e	ui
T1	(2, 1)	0.5
T2	(2.5, 0.1)	0.04
T3	(3, 1)	0.333
T4	(4, 1)	0.25
T5	(4.5, 0.1)	0.0222
T6	(5, 1)	0.2
T7	(6, 1)	0.167
T8	(7, 1)	0.142
T9	(8, 1)	0.125
T10	(8.5, 0.1)	0.01176
T11	(9, 1)	0.111

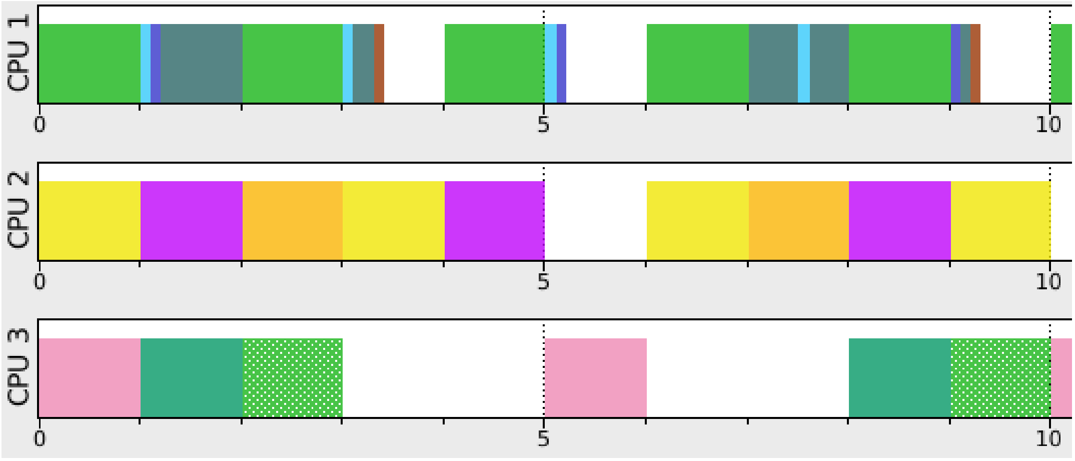
P1: T1 T2 T5 T7 T10
P2: T3 T4 T8
P3: T6 T9 T11

Urm(2) = 2(2^1/2 – 1) = 0.8284
Urm(3) = 3(2^1/3 – 1) = 0.7797
Urm(4) = 4(2^1/4 – 1) = 0.7568
Urm(5) = 5(2^1/5 – 1) = 0.7435
Urm(6) = 6(2^1/6 – 1) = 0.7347



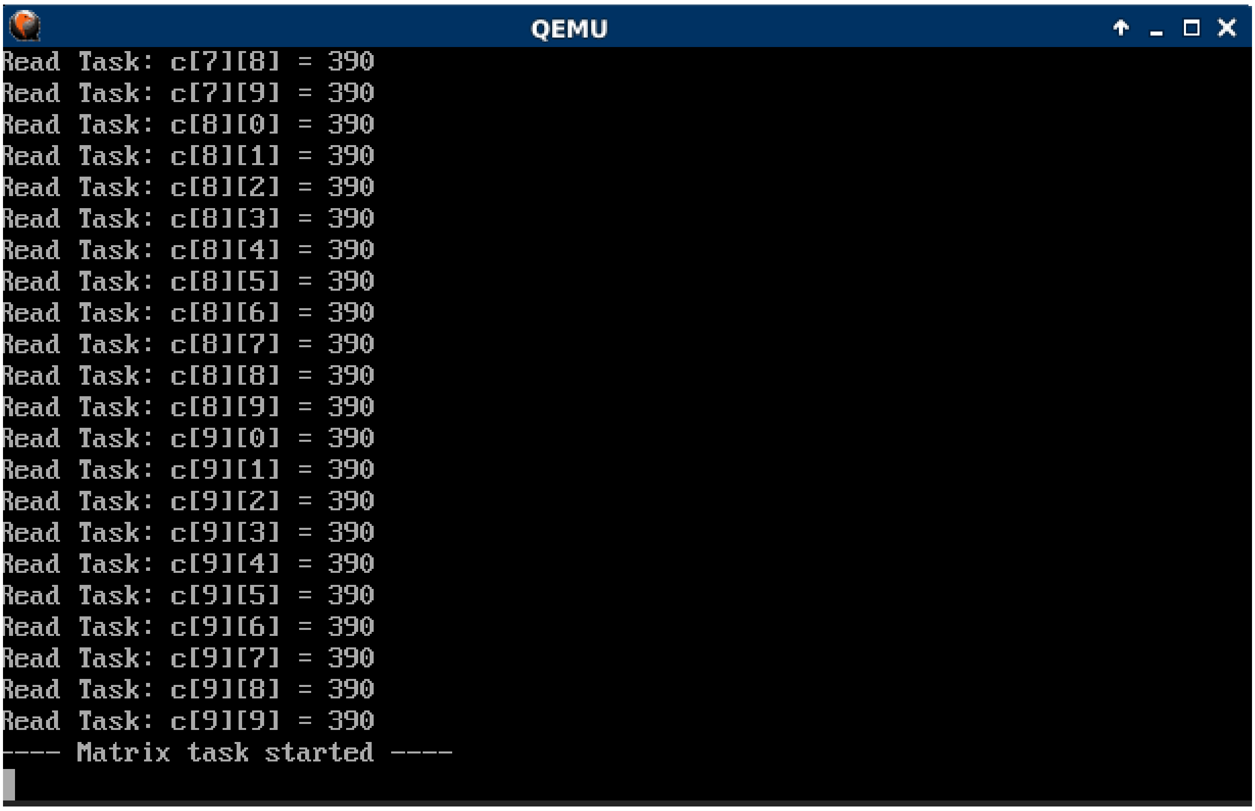

TASK T3
TASK T4
TASK T5
TASK T6
TASK T7
0
5
10


TASK T8
TASK T9
TASK T10
TASK T11
0
5
10


CPU 1
CPU 2
CPU 3
0
5
10

```
"""
Partitionned EDF using PartitionedScheduler.
"""
from simso.core.Scheduler import SchedulerInfo
from simso.utils import PartitionedScheduler
from simso.schedulers import scheduler
import math

@scheduler("simso.schedulers.P_RM")
class P_RM(PartitionedScheduler):
    def init(self):
        PartitionedScheduler.init(
            self, SchedulerInfo("simso.schedulers.RM_mono"))

    def packer(self):
        # First Fit
        cpus = [[cpu, 0] for cpu in self.processors]
        cpu_list = {}
        cpu_list[0] = 0
        cpu_list[1] = 0
        cpu_list[2] = 0
        for task in self.task_list:
            m = cpus[0][1]
            j = 0
            print task.period
            # Find the processor with the lowest load.
            for i, c in enumerate(cpus):
                util = c[1] + float(task.wcet) / task.period
                task_num = cpu_list[i] + 1
                urm = task_num*(math.pow(2, 1.0/task_num) - 1)
                #print i, util, task_num, urm
                if util < urm:
                    #print task.period
                    self.affect_task_to_processor(task, cpus[i][0])
                    cpus[i][1] += float(task.wcet) / task.period
                    cpu_list[i] += 1
                    break
                else:
                    continue
                #if c[1] < m:
                    #m = c[1]
                    #j = i

            # Affect it to the task.
            #self.affect_task_to_processor(task, cpus[j][0])

            # Update utilization.
            #cpus[j][1] += float(task.wcet) / task.period
        return True
```

Programming assignment



```
Read Task: c[7][8] = 390
Read Task: c[7][9] = 390
Read Task: c[8][0] = 390
Read Task: c[8][1] = 390
Read Task: c[8][2] = 390
Read Task: c[8][3] = 390
Read Task: c[8][4] = 390
Read Task: c[8][5] = 390
Read Task: c[8][6] = 390
Read Task: c[8][7] = 390
Read Task: c[8][8] = 390
Read Task: c[8][9] = 390
Read Task: c[9][0] = 390
Read Task: c[9][1] = 390
Read Task: c[9][2] = 390
Read Task: c[9][3] = 390
Read Task: c[9][4] = 390
Read Task: c[9][5] = 390
Read Task: c[9][6] = 390
Read Task: c[9][7] = 390
Read Task: c[9][8] = 390
Read Task: c[9][9] = 390
---- Matrix task started ----
```