UNIVERSITÉ LIBRE DE BRUXELLES

DÉPARTMENT D'INFORMATIQUE

INFO-F404
REAL-TIME OPERATING SYSTEMS

Project 1 – Audsley

Author:
Hakim BOULAHYA
Youcef BOUHARAOUA

December 21, 2017

Contents

1 Simulator

Implementation choices The simulator of the single FTP simulator is implemented in a discret manner from `start` to `stop`. For each time steps we execute functions that will update the current running job informations, the different requests (arrivals or deadlines) and will store them in datasets. We choose not to print the different actions during the simulation, since the simulation is used for other commands such as the plotter or the audsley algorithm. The output can be produced by a call to the method `FTPSimulation.output()`.

1.1 Periodicity of the requests

There were two possible implementatisn to simulate the scheduler in a interval given as a parameter. The first idea was to simulate the scheduler between the feasability interval $[0, S_n + P)$, but it would be tidious to do it like this, since we would have to find the corresponding interval in $[0, S_n + P)$ of the $[start, stop)$ interval of the simulation.

Arrivals periodicity The important part of the simulation is to detect the arrivals *i.e.* the job requests and the deadlines of those jobs. We know that the job requests for each task are periodic. Let $\tau_i = (O_i, T_i, D_i, C_i)$ a periodic task. We know that each job request of τ_i have a periodicity of T_i with the first request made at time O_i. Ce can detect if a request for τ_i occur at time t using the following formula:

$$[(t - O_i) \geq 0] \;\; and \;\; [(t - O_i) \; mod \; T_i = 0] \tag{1}$$

The left-side formula is the detection of the first request. If $t - O_i$ is bigger than 0, then it means that it is possible that τ_i request a job. The right-side formula is the detection of a request at time t. To detect which job it is we can make an entire division of $t - O_i$ with the period T_i. Figure 1 is a excerpt of the arrival detection.

```python
offset, period = self.tasks[task_id][O], self.tasks[task_id][T]
cond = (t - offset >= 0) and ((t - offset) % period) == 0
job_id = (t - offset) // period  # module == 0 i.e. no decimals
return (job_id, True) if cond else (None, False)
```

Figure 1: Function of the arrival detection for a task at time t

Deadlines periodicity For each job requested job there is a deadline associated *i.e.* the deadline is also periodic. To detect if a deadline occur at time t of a task τ_i we can use the same formula (1), and testing the formula a time $t - D_i$. This method actually detect if a job was requested, and we are a the requested time plus the deadline, the it is the deadline of this job. Figure 2 shows a excerpt of the deadline detection.

```python
deadline = self.tasks[task_id][D]
return self.is_arrival_for(t - deadline, task_id)
```

Figure 2: Function of the deadline detection for a task at time t

1.2 Pending jobs

To be able to process requested jobs, the simulator uses a list of list to handle pending jobs. A sublist per task exists. When a request of a task occurs, the new job is added to the pending jobs sublist of the task. The sublists are ordered by task priority. Therefore the job to process is the first job of the first sublist. If this list is empty, then it process the first job of the second list, etc. When a job finished its computation *i.e.* has been processed C_i time, then it is removed from the pending jobs list. Figure 3 shows the snippet of the function that returns the job to be processed.

```
for sub_jobs in self.pending_jobs:
    for job in sub_jobs:
        return job
return None
```

Figure 3: Job to be processed detection

1.3 Events

For each time steps, the simulator stores relevant informations that can be used to output informations on the standard output or in a plot. Those informations are stored in an object `Event` which is composed of actions that occurs at time t:

- The requests
- The deadlines (and the missed deadlines)
- The computed job

1.4 Hard vs Soft simulation

It is possible to configure the simulation to consider the deadlines as soft or hard. If the soft configuration is chosen, the simulator will not stop at deadline misses. If the hard configuration is chosen, the simulator will stop at the first deadline miss (see section 3 to see how to run each configuration).

1.5 Output

The output production is a post process function. It has been implemented like this to avoid printing while doing the simulation because the simulation is used by other part of the project such as the implementation of the Audsley algorithm or the Schedule plotter. The `output()` method uses the events of the simulator (section 1.3).

The simulator provides two approch to the output. The default mode is the **request mode**. It provide an output as shown in Figure 4b. This mode output the requests, such as arrivals or deadlines, as soon as they occur. So it does output even if a job is processing during a block. This mode has been implemented to provide a more continuous approch to the simulation.

The second approch is the **preemptive mode**. As explained above the process block of a job can be divide in multiple blocks if requests occur during the process. To propose an output that characterize more the preemptive proprety of a FTP simulator, the preemptive mode provide a preemptive resuts as shown in Figure 4c.

Section 3 explained how to run the different modes.

```
0 5 10 5
0 3 20 2
```

(a) Tasks set to output (O_i, T_i, D_i, C_i)

```
0: Arrival of job T1J1
0: Arrival of job T2J1
0-3: T1J1
3: Arrival of job T2J2
3-5: T1J1
```

(b) Requests mode output

```
0: Arrival of job T1J1
0: Arrival of job T2J1
0-5: T1J1
3: Arrival of job T2J2
```

(c) Preemptive mode output

- necessary to run the full simulation to be sure that it works
- Using the hyper period vs calculation with modulo
- explain output (preemptive non-preemptive)
- explain soft and hard simulation
- HowTo : simulation hard/soft, –premptive

2 Plotter

Show scheduler.png of tassk.txt 0 200 and audsley 0 400

3 How to run