Université libre de Bruxelles

Project 1

Global vs. Partitioned DM

**Aldar Saranov, Przemyslaw Gasinski**

[Aldar.Saranov@ulb.ac.be](mailto:Aldar.Saranov@ulb.ac.be)

[Przemyslaw.Gasinski@ulb.ac.be](mailto:Przemyslaw.Gasinski@ulb.ac.be)

INFO-F-404 Real-Time Operating Systems II (M-INFOS/F277)

Joël Goossens, Nikita Veshchikov

November 2016

# Software description

## Generator

### Input/output

Command in form

|  |
| --- |
| ./taskGenerator -u 70.0 -n 8 -o tasks.txt (-a 100.0) |

-u – Total utilisation in percent

-n – Number of tasks

-o – Output filepath/filename

-a – Approximate average WCET (optional argument, the higher it is, the more precisely utilisation will be attained. 100.0 by default if not specified)

Output is passed to the specified file. Every line is corresponds to a task. Order: Offset, Period, Deadline, WCET.

|  |
| --- |
| 56 181 134 61  10 181 150 142  38 181 152 132 |

### Generation algorithm

Generator uses following algorithm:

1. Calculate common period by formula
2. For each task generate offset according to uniform distribution from some (hardcoded) and .
3. Distribute the ordered over each of tasks.
   1. Set .
   2. Set .
   3. If
   4. If
   5. Ensure that the rest tasks will get by 100% utilization each in worst case (extra constraint for
   6. Same (c) and (d) for .
   7. Generate utilization for the next task according uniform distribution from to .
   8. Calculate and decrement .
4. For each task calculate .
5. For each task generate according to uniform distribution from and .

## Simulator

### Input/output

|  |
| --- |
| ./simDM [-g | -p] <tasksFile> <processorsNbr> (-s) |

[-g|-p] – g corresponds to global strategy, p to partitioned.

<taskFile> - input tasks filename/filepath.

<processorNbr> - number of processors available during the simulation.

-s – optional argument, determines if the application will output the simulation steps one after another.

Outputs to console and to result.txt file following:

1. Minimum number of processors required for this input data (besides of the specified number of processors).
2. Flag if the scheduling is feasible for specified number of processors.
3. Simulation interval for every processor (they are equal in global case).
4. Total number of preemptions and per every processor.
5. Total number of idle time units and per every processor.
6. Total utilization and per every processor.

Before that it outputs to console the simulation steps one after another if “-s” was specified.

### Simulation algorithm

We have two possible simulation algorithms which can process several events [1].

1. Future event list/set (similar to GPSS modeling language future event chain algorithm).
2. Delta T.

**Future event chain** has following algorithm (Figure 1). It is based on determining of the next event by comparing their planned time moments and executing the nearest event action with consequent transition of current time to that moment. It is effective at large time scale (which is our case) but slightly hard to implement due to obligation to control the future event list manually (add/remove/execute events).

**Delta T** has following algorithm (Figure 2). It is based on regular adding of a constant delta T timespan to the current simulation moment and analyzing which event actions have to be executed. It is more simple to implement and effective in cases when we have high possibility of obtaining events happening at the same moment. However this algorithm is not effective on large time scale due to regular status analyzing at every moment especially in cases if delta T is much less than average time intervals between expected events.

Eventually future event chain has been implemented due to its effectiveness. Events setup in future event list is illustrated in Table 1. Nonetheless we have modified end of simulation condition. Simulation is over when for every processor simulation time is over (first N events). The task events (start, finish, deadline) are grouped in three-event tuples.

Both in global and partitioned we have agreed on following event types:

* + 1. Processor simulation over.
    2. Job spawn.
    3. Job finish.
    4. Job deadline.

For 0-event type we have to:

1. Unassign processor from a task
2. Erase itself (set this 0-type event to unexpected).

For 1-event type we have to:

1. Check if previous deadline of this job was satisfied (otherwise fire non-feasible).
2. Assign to the job time-left parameter to WCET of its task.
3. Set the next job spawn time to current time + task period.
4. Set the closest deadline to current time + task deadline.

For 2-event type we have to:

1. Set finish event to unexpected.
2. Set deadline event to unexpected.
3. Unassign task from its processor.

And after 0, 1, 2 type events we must execute reassignTask()

For 3-event type (if program got here) then it means that deadline was trespassed and the system is not feasible.

reassignTask():

1. Assigns tasks to processors and vice versa according to global or partitioned strategy.
2. Tracks preempted tasks.
3. Checks that task does not migrate if it is not preempted in global case.
4. Checks that task does not migrate at all in partitioned case.

### Best fit

Was successfully implemented. Takes tasks and number of processors as arguments. Also returns unfeasible-flag in case if it not possible to distribute them.

## Study

### Input/output

# Difficulties

## Generation fraction wcets

Initially it was proposed to fix the period (for each task) as a constant during task generation. The problem of such approach may be encountered if user sets too large value of or too small . In such case wcets generated at the next step often were fractions between 0 and 1 and therefore were being truncated due to integer type. The actual utilization deviated from the ordered too much.

## Generated task overload

This is the reason why step 3-e (in the algorithm description) was derived. Suppose we have 3 tasks to which we have to assign 270% of utilization. If we neglect 3-e step then for some configuration we can obtain generation range [50%;100%]. In such case if we generate the first task with 60% utilization then we will leave 210% for the rest 2 tasks which cannot distribute these utilization. We must have at least [70%;100%] generation range. should be not less than (if this expression is positive).

## Too large hyper period

For random periods at large scale hyper period (which is LCM of periods) becomes even larger what leads to value overflow for big number of tasks. To resolve this problem the period has been set same (but different for different input) for all the tasks. The hyper period is equal to this value and without any harm caused to the simulation model.

## Too large simulation timeline

Overflow bugs resolved by migrating from unsigned int to long long.

## Calculating the minimal number of processors required

Is calculated (Figure 3) using the number using a loop by regular incrementing the number of allocated processors with consequent simulation. If the number of processors exceeds the number of tasks then it returns non-feasibility value.

# Test planning

# Simulation analysis

## Obtained results

## Conclusion

# Appendix

## Figures

D:\scheduling\report\FEC.png

Figure 1. Future event list algorithm.

D:\scheduling\report\DeltaT.png

Figure 2. Delta T algorithm.

D:\scheduling\report\MinimumProcessor.png

Figure 3. Minimum processor determiner algorithm.

## Tables

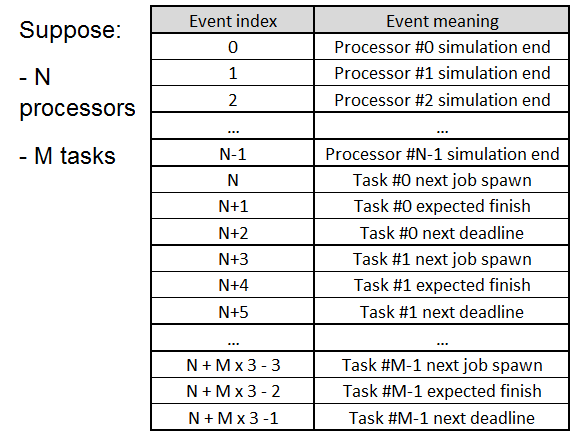


Table 1. Event setup illustration of future event list algorithm.

## References

1. <https://books.google.ru/books?id=XUnPBAAAQBAJ&pg=PA29&lpg=PA29&dq=future+event+list+and+delta+t&source=bl&ots=Ws8uv5r1PN&sig=fpWhjoUvqLrKgaJYm2namMtCUZo&hl=ru&sa=X&ved=0ahUKEwimqcDUycLQAhUCqxoKHZeSDAcQ6AEIJTAB#v=onepage&q=future%20event%20list%20and%20delta%20t&f=false>