Real-Time Multi-Task Simulation in Forth

Sergey Baranov
SPIIRAS, ITMO University
St.Petersburg, Russia
snbaranov@gmail.com

***Abstract*–Gained experience to rapid developing of software tools for investigating real-time multi-tasking through simulation of the behavior of respective formal models is described. The approach is based on using the interpretative programming language Forth which opens a wide range of options to properly tailor the tool for particular purposes and seems to have a much broader scope if properly used.**

I. INTRODUCTION

Software simulation is an acknowledged method to check feasibility of real-time multi-task applications. This paper describes an experience of constructing such simulator in Forth with the proprietary implementation VFX Forth for Windows [1] as the instrumental platform. A freeware option for the platform is gForth [2]. Forth was selected as the implementations language due to the flexibility it provides for implementing programming solutions. Another advantage is that it allows to use only fixed-point arithmetic in calculations and avoid floating point with all its related issues and trade-offs. The simulator employs a simple model of a multi-task application under study which may use a particular *scheduling mode* with various task priorities for allocation of a multi-core processor computational resource and a particular *access protocol* to access shared informational resources with various kinds of priority inheritance. The simulator helps to study multi-task application behavior and check whether a given combination of the scheduling mode and priority inheritance ensures application feasibility under the given processor performance and system event scenarios. It may also identify the minimal processor performance which still ensures application feasibility under the given conditions.

By now, the nomenclature of scheduling modes and inheritance modes implemented in the simulator consists of two classical scheduling modes – RM (*rate monotonic*) and EDF (*earliest deadline first*) – and three inheritance modes – NI (*no inheritance*), DI (*direct priority inheritance*), and TI (*transitive priority inheritance*). However, it may be further extended to simulate systems with other scheduling modes on a multi-processor and/or multi-core platform and protocols of access to shared informational resources [3].

Total effort for implementing this simulator was 4 staff-month of relatively background work within which 8 successive releases of the simulator with extending functionality on bug fixing were produced. The current version RTMT 8.55 is just 985 LOCs long.

II. SOURCE DATA

Simulation is based on components of four kinds: *resources*, *tasks*, *jobs*, and *events*. Resources and tasks are entities of the application under study; jobs and events are entities created and operated on by the simulator. Resources and tasks are also represented within the simulator with respective entities. The application is assumed to run either on a single-core processor platform with a certain processor performance P in terms of "the number of standard operations per second", or on a multi-core processor platform with $m>1$ identical cores of the same performance P. A scaling parameter determines the actual processor speed. An application under study consists of a number of tasks τ_i. Each application task τ_i is characterized by its timing period T_i – the minimal timing interval between two consecutive activations of τ_i determined by the current scenario of system events, its priority $Prio_i$ – which usually descends with increase of i, its weight W_i – the amount of processor work needed to accomplish this task, its deadline D_i – the maximal time period for the task to be completed, and its phase Ph_i – the offset of the first activation of this task from the simulation starting moment (by default $Ph_i=0$). Like the processor performance P, the task weight W_i is specified in the number of standard operations, and may be converted into seconds: $C_i=W_i/P$. Obviously, $\forall i$ $C_i \le T_i$. The values T_i, D_i, and Ph_i are specified in absolute timing units (e.g., seconds) and do not depend on the processor performance P.

Application tasks may access shared informational resources identified with their unique ID numbers; however, at any moment of time a shared resource may be accessed by only one task. Tasks which do not share any informational resources are considered to be *independent* with respect to each other. To prevent simultaneous access of two or more tasks to a shared resource, *critical intervals* within the task code are established and guarded with *mutexes* – a particular case of Dijkstra semaphores.

With this in mind, the structure of each task τ_i is represented in the simulator as a finite series of *k(i)* segments, each segment performing a certain amount of computational work $w_j>0$ (the segment weight) and terminating with one of the following system events: "*Lock r*", "*Unlock r*", or "*End*", r being the resource ID number. The duration of processing a system event is assumed to be negligibly small. A correct application should neither unlock a resource not locked by this task earlier, nor lock it again

ISSN 2305-7254

without preceding unlocking it, nor leave it locked upon task termination, and each task should terminate with the segment "*End*". Obviously, the task weight W_i equals to the sum of the weights of all its segments: $W_i=\Sigma_{j=1..k(i)} w_j$.

An example of a typical application description in an XML-type fashion [4] "with the postfix Forth flavor" is provided in Fig. 1 for an application of 4 tasks τ_1, τ_2, τ_3, and τ_4, which share 2 informational resources r_1 and r_2 identified by their numbers 1 or 2. Tasks are enumerated in this description in their natural order from 1 to 4.

```
<application
<task 5 =phase 15 =period />
   1 <lock 1 />  1 <unlock 1 />  <end 1 /> </task>
<task 5 =phase 35 =period />  <end 9 /> </task>
<task 3 =phase 25 =period />
   1 <lock 1 />  2 <lock 2 />  2 <unlock 1 />
   1 <unlock 1 />  <end 1 /> </task>
<task 45 =period />  2 <lock 2 />  2 <unlock 4 />
   <end 1 /> </task>  </application>
```

Fig. 1. Specifying the application structure in XML-like fashion

Due to Forth specifics, all elements of this notation are space separates. Actually, this specification is a Forth-text submitted in a configuration file, definitions of its key words being provided in the simulator realization. Processing this text by the underlying Forth interpreter (e.g., through the Forth word include with the file name as its parameter) results in building the respective internal structure which represents the application after appropriate syntax checks which validate this text and complement it with omitted default values (like =phase in task τ_4).

The code of the highest priority task τ_1 consists of 3 segments of 1 time unit each. Task description starts with the word <task followed by specifications of task parameters (5 =phase) and (15 =period). Parameter enumeration terminates with the word /> and then follows enumeration of task segments. The first segment (1 <lock 1 />) consists of the operation lock for resource number 1 and its duration is 1 timing unit; the next segment specifies unlocking this resource (1 <unlock 1 />) after 1 timing unit of computation, and the third segment terminates the task (<end 1 />). The code of the task τ_2 consists of only one segment of 9 timing units while task τ_3 consists of 5 segments with two critical intervals to access the resources r_1 and r_2, the intervals being embedded in one another. The least priority task τ_4 consists of 3 segments and accesses only the resource r_2.

Task periods T_1, T_2, T_3, and T_4 for task activations are 15, 35, 25, and 45 time units respectively with the phase shifts 5, 5, 3, and 0; deadlines are assumed to be equal to task periods: $D_i=T_i$. Tasks and resources are rendered by objects of the type *task* and *resource* respectively and are created by respective Forth words during simulator initialization when reading an input file with the task descriptions:

```
: CreateTask ( -- task-addr)
: CreateResource ( n -- resource-addr)
```

III. Output data

The aim of the simulator is to calculate certain application characteristics under various combinations of scheduling mode, inheritance mode, the number of processor cores and their performance, obtained as output data from simulation sessions.

For each task τ_i the derivative characteristics are defined: its utility load $u_i=C_i/T_i$ and its hardness $H_i=T_i/D_i$ which characterize tasks execution. If $H_i \le 1$ then the existence intervals of consecutive jobs ${}_j\tau_i$ and ${}_{j+1}\tau_i$ created from two consecutive activations of the task τ_i do not intersect. The reverse condition $H_i>1$ means that they may intersect. An important metric – the density of the whole application: $Dens=max_P(\Sigma_{i=1..n} u_i)$ – may be calculated too, in order to compare different application structures and implementations on their efficiency [5].

The ultimate purpose of simulation is to obtain data on efficiency of various combinations of scheduling modes and inheritance mode of the access protocol in various scenarios of system events on single- and multi-core processors. In particular, the dual problem to calculating the application density – to determine the minimal processor performance which still ensures the feasibility of the application (i.e., that $\forall i\ R_i \le D_i$) under given conditions – may be solved as well.

To calculate the application density, the initial interval $[a,b]$ for selecting the scaling factor $f \in [a,b]$ for the task weights and processor performance is established. Prior to the simulator run, the source values of task segment weights w_j (and therefore, the task weights W_i) in task descriptions and the processor performance P are multiplied by this factor. Obviously, if the inequality $R_i \le D_i$ is violated for some i at the end-values a and b of the interval, it is violated for all intermediate values. However, for $f=a=0$ (which means an infinitely high processor performance) these inequalities do hold for all i. Therefore, the initial values are set to $a=0$ and $b=U=\Sigma_{i=1..n}u_i$ with the standard processor performance $P=10^6$ standard operations per second. Then the first simulation iteration is performed with the scaling factor $f=(b-a)/2$. If no violations of $R_i \le D_i$ occurred, then a is set to f, otherwise b is set to f and simulation is reiterated until the scaling interval shrinks to just one value $[a, a+1]$ in which case the scaling factor equals to this found value a, the application density is calculated accordingly, and the minimal processor performance P which still ensures the application feasibility

is $P=a\times10^6$ operations per second. It usually takes from 5 to 15 simulations to reach the resulting values.

IV. Data structures

The simulator uses ordered chained lists whose elements consist of 3 cells: the link to the next list element or NULL, the ordering value and the data specific to the list. Elements in a list are ordered with respect to the ordering value, starting with the smallest one. Lists are defined with the defining word List:

```
: List ( list-element-size, max-list-length -- )
```

and use respective "methods" to add and retrieve elements in lists created by this word:

```
: >List ( new-elem-addr, list-addr -- )
      \ Place a new element into the ordered list
: List@ ( list-addr-- elem-addr)
      \ Get the first (heading) element of the list
: List> ( list-addr-- elem-addr)
      \ Delete the first element from the list
: List>> ( ordering-value, list-addr--)
      \ Find and delete a list element with this value
```

Static objects (tasks and resources) are created at the simulator initialization from the task description file and are modified during simulation.

A resource is rendered with an object of 4 cells: its ID number, priority (reserved for future use), status (either NULL if the resource is currently unlocked, or a reference to the job description, which currently owns this resource and locked it), and a possibly empty ordered list of job descriptions, currently waiting for this resource to become unlocked. Resources are stored in a special pool which allows to easily enumerate them and to add a new one.

Tasks are represented with objects of various length which depends on the number of task segments. It starts with 10 cells followed by a series of 4 cells for each task segment. The initial cells contain: task unique ID number i, task period T_i, task weight in the number of standard operations W_i, task weight in seconds C_i (depends on the scaling factor f), task response time R_i (is calculated during simulation), task deadline D_i, task phase Ph_i, the number of executed task activations, and the number of task segments. Cells for each task segment are: segment type (*Lock*, *Unlock*, or *End*), segment parameter (the resource ID for *Lock*/*Unlock*), segment weight in the number of standard operations S_j, and the segment time in seconds (recalculated while scaling the task data with the scaling factor f).

Dynamic objects (jobs and events) are created during simulation sessions as needed with the words CreateJob and CreateEvent :

```
: CreateJob ( task-addr--job-addr)
: CreateEvent (( resource-addr, job-addr, task-addr,
    event-type, event-time -- event-addr))
```

The job object is represented with 10 cells: the job unique ID, its current priority (it may change with the priority inheritance scheduling mode), current segment number which specifies the segment begin executed, current segment expected termination time, current segment start time, current segment used time, current segment time yet to be used, reference to the respective task, number of references to the job description, and a reference to a resource which this job is waiting for or NULL if the job is not waiting for a resource. Jobs waiting for the processor form a chained list JobList in the order of their current priorities. If the number of elements in this list is L and $L>0$ then the first $l=\min(L,K)$ jobs in this list are current jobs; i.e., they own l cores of the multi-core processor during the given interval if system time, while the remaining $m–l$ cores stay idle (m being the number of processor cores). It's interesting to note that core ordering is irrelevant.

System events are characterized by the time when they occur. Events with the same timing form a group of time-sake events. Four types of system events are considered: *to activate* a task (i.e., to form a job for this task and add it to the list JobList of active jobs waiting for the processor), *to terminate* the current job (and pass the processor to another job in list JobList, if any), *to lock* a resource, or *to unlock* a resource – and these activities are performed with respective Forth words:

```
: TaskActivate ( task-addr--)
: JobTerminate ( job-addr--)
: ResourceLock ( resource-addr, job-addr--)
: ResourceUnlock ( resource-addr, job-addr--)
```

The event object which represents a system event consists of 6 cells: the event unique ID, the scheduled time for this event to occur, the type of the event (*Activate*, *Lock*/*Unlock*, or *End*), a reference to the task object to be activated or NULL, a reference to the job object to be ended or NULL, and a reference to the resource object to be locked/unlocked or NULL. The chained list EventList of system events ordered with respect to their time moments when they scheduled to occur is maintained by the simulator.

V. The simulator

The architecture of the simulator is presented in Fig. 2. Simulator initialization consists in selecting the desired combination of the scheduling mode and inheritance mode of the access protocol, setting the respective simulator constraints, reading the task description file, and forming the respective resource and task objects. Then the initial list of system events EventList is formed which consists in activation of the all tasks at the moments of system time

defined by their phase shifts. Counts for their maximal response times are set to zero and all resources are set to be unlocked.

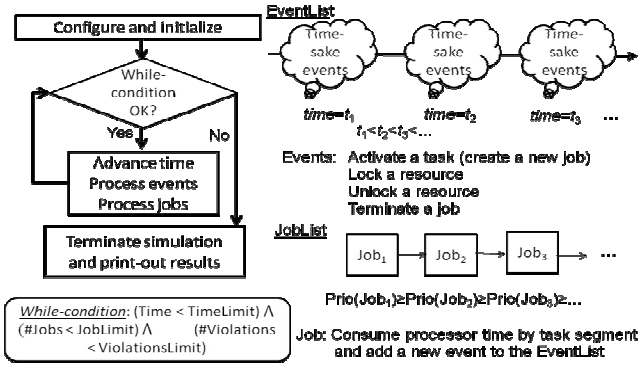


Fig. 2. Simulator architecture

The major simulator loop does the following. The first group of time-sake events in EventList is considered, the simulator system time is set to this time moment and all system events from this first group are processed one-by-one. Processing depends on the event type: activate a task, terminate a job, or lock/unlock a shared resource.

Activating a task. A new job is created from this task referred to by the event with its planned starting time equal to the current system time and is added to JobList with its priority, while a new event is added to EventList – to activated the next copy of this task at the moment of time not less than the current time plus the task period T_i.

Terminating a job. The response time of the task referred to by the respective job object is updated: the difference between the current system time and the moment when this job was created and added to JobList (the response time which consists of the time when the job owned the processor plus the time it waited for it) is calculated and the maximum of this value and the response time already stored in the task referred to is stored as the new value of the task response time. If this exceeds the task deadline D_i, then a violation of the task feasibility is registered. The considered job is deleted from the JobList.

Locking a resource. If the resource is unlocked, then it becomes locked by this task; otherwise, the job is moved from the JobList to the ordered list of jobs waiting for unlocking of this resource.

Unlocking a resource. If the ordered list of jobs waiting for unlocking of this resource is not empty, then the first job form this list is moved from it back to JobList according to its priority and the resource becomes locked by this job; otherwise, the resource becomes unlocked.

Upon completion of the event processing, the considered event is deleted from EventList. After all time-sake events have been processed, JobList, which may have changed as a result of previous event processing, is considered unless it is empty.

If JobList is not empty then the first job from it (which currently owns the processor) is selected and the residue of the processor time not yet consumed by its current segment is considered. This value determines the moment of the segment termination. If this value is greater than the time of the next time-sake group of system events in EventList then this residue is decremented by the remaining time till this event group; otherwise, a new event corresponding to this segment is added to EventList for this moment of segment termination and the next job segment if any becomes its current segment.

Emptiness of JobList means that the processor is idle from this moment till the next time-sake event group in EventList. Upon completion of processing the first job of JobList (if any) the major loop is reiterated. The loop terminates upon exhausting the time limit of the simulation session or when a specified number of created jobs is reached (which of these conditions occurs earlier, if both limits are specified).

TimeLimit=25 JobLimit=0 ViolationLimit=1 SchedulingMode=RM Inheritance=NI Configuration file name: c:\MPE\App_4t2r.txt Time=0 Proc=0 for 0 A 4.1 Time=2 Proc=4.1 for 2 L 4.1 of 2 Time=3 Proc=4.1 for 1 A 3.2 Time=4 Proc=3.2 for 1 L 3.2 of 1 Time=5 Proc=3.2 for 1 A 1.3 A 2.4 Time=6 Proc=1.3 for 1 W 1.3 of 1 Time=15 Proc=2.4 for 9 E 2.4 Time=16 Proc=3.2 for 1 W 3.2 of 2 Time=19 Proc=4.1 for 3 U 4.1 of 2 L 3.2 of 2 Time=20 Proc=3.2 for 1 U 3.2 of 2 Time=21 Proc=3.2 for 1 U 3.2 of 1 L 1.3 of 1 Time=22 Proc=1.3 for 1 U 1.3 of 1 Time=23 Proc=1.3 for 1 E 1.3 Time=24 Proc=3.2 for 1 E 3.2 Time=25 Proc=4.1 for 1 E 4.1 Time=25 Hardness=1,0000 1/Hardness=1,0000 Density=0,6056 ScalingFactor=1,0000 **ERROR: Deadline violation in Task 1** ok	TimeLimit=25 JobLimit=0 ViolationLimit=1 SchedulingMode=RM Inheritance=DI Configuration file name: c:\MPE\App_4t2r.txt Time=0 Proc=0 for 0 A 4.1 Time=2 Proc=4.1 for 2 L 4.1 of 2 Time=3 Proc=4.1 for 1 A 3.2 Time=4 Proc=3.2 for 1 L 3.2 of 1 Time=5 Proc=3.2 for 1 A 1.3 A 2.4 Time=6 Proc=1.3 for 1 W 1.3 of 1 Time=7 Proc=3.2 for 1 W 3.2 of 2 Time=10 Proc=4.1 for 3 U 4.1 of 2 L 3.2 of 2 Time=11 Proc=3.2 for 1 U 3.2 of 2 Time=12 Proc=3.2 for 1 U 3.2 of 1 L 1.3 of 1 Time=13 Proc=1.3 for 1 U 1.3 of 1 Time=14 Proc=1.3 for 1 E 1.3 Time=23 Proc=2.4 for 9 E 2.4 Time=24 Proc=3.2 for 1 E 3.2 Time=25 Proc=4.1 for 1 E 4.1 Time=25 Hardness=1,0000 1/Hardness=1,0000 Density=0,6056 ScalingFactor=1,0000 ok

Fig. 3. Two simulation sessions with different priority inheritance

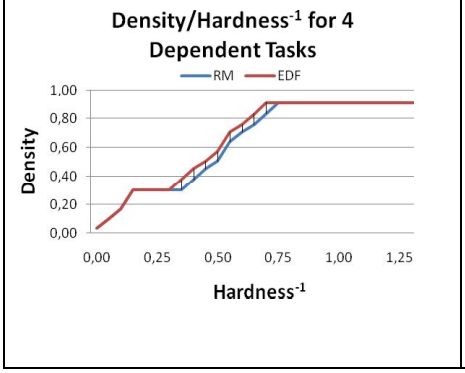
The results of simulation – task maximal response time, number of deadline violations, the application density, and other statistics data are displayed. A simulation log may also be displayed. When any system event is processed, the respective time and other accompanying data are printed-out. All these data may be easily copied into MS Excel for

a graphical representation of the obtained results and execution log.

There are the two logs of simulator runs in Fig. 3 – for two different protocols of access to shared resources: NI (no inheritance) and BI (basic priority inheritance) as they are recorded by the simulator. The number after "Time=" is the time of an occurring system event denoted by one of the letters: A – activate, E – end, L – lock, U – unlock, or W – wait to lock an already locked resource, followed by the event parameter. The job ID is displayed as two numbers (the task number and the unique job number separated with a period). The section "of" is followed by the resource number to be locked or unlocked, while a number after "for" is the activity duration terminated with this event.

This application, when simulated twice with different access protocols, demonstrates two different behaviors: a violation of the specified deadline 15 for the highest priority task τ_1 under the protocol NI – Fig. 3 (left side), and correct work with no violations under the protocol DI – Fig. 3 (right side).

Fig. 4 demonstrates impact of two scheduling modes on the application density for the same application of 4 tasks and 2 shared resources defined in Fig. 1 when running on a single-core processor. The output simulation data were copied into an MS Excel file to obtain these charts. Data for application hardness and respective density values for the two scheduling modes are in the right-hand columns of the chart. As one can see, there's no big difference in the application density between the two scheduling modes RM and EDF for this particular application. Density as a function of $hardness^{-1}$ grows nearly linearly with two plateaus and then the growth stops after $hardness^{-1}$=0.75. As one can see, this application cannot reach 100% density – its maximum is 0.9083 with the application $hardness$=1/0.75=1.33 and it does not change with further decrease of hardness (i.e., increase of $hardness^{-1}$), which means that the processor would be inevitably idle for at least ≈10% of time while executing this application.



1/H	RM	EDF
0.00	0.0336	0.0336
0.05	0.1009	0.1009
0.10	0.1682	0.1682
0.15	0.3028	0.3028
0.20	0.3028	0.3028
0.25	0.3028	0.3028
0.30	0.3028	0.3028
0.35	0.3028	0.3784
0.40	0.3784	0.4541
0.45	0.4541	0.5046
0.50	0.5046	0.5719
0.55	0.6392	0.7065
0.60	0.7065	0.7568
0.65	0.7568	0.8326
0.70	0.8326	0.9083
0.75	0.9083	0.9083

Fig. 4. RM vs. EDF for same application of 4 tasks with 2 resources

Another example of using MS Excel for better graphics is presented in Fig. 5 for a study of dependency between application density and the number of processor cores. As one can see, a substantial difference turned out to be between single-core (the lower line) and two-or-more-core processors (all upper lines), while increasing the number of cores did not impact the density of an application with the given task structure.

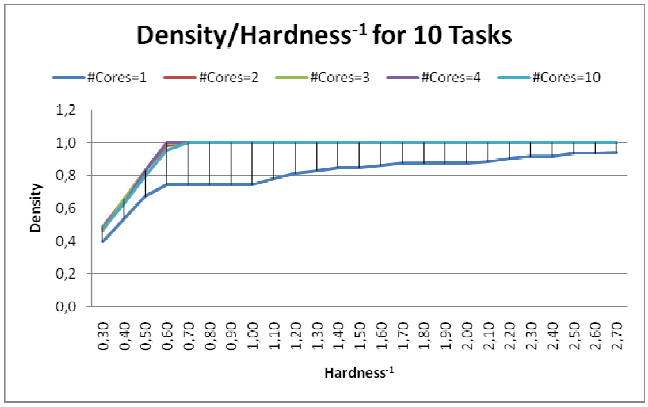


Fig. 5. Application density and the number of cores

These examples demonstrate the remarkable flexibility of Forth for interoperation with other powerful tools for computer-aided analysis of data.

VI. Four Dining Philosophers

This classical puzzle, first proposed by E.Dijkstra as "Five Dining Philosophers" [6], demonstrates the situation of mutual blocking under certain scenarios of dependent task behavior with more than one respective processes. Let's consider 4 iterative processes, each with two alternate activities called "think" and "eat", the latter assuming simultaneous access to 2 of 4 shared resources (called the left and the right fork for this philosopher) for a certain period of time. Access to the resources is performed via critical intervals guarded with respective mutexes.

The puzzle works for any number of philosophers greater than 1. Let's take 4 (a bow to Forth); with the proposed technique this may represented as 4 tasks τ_1, τ_2, τ_3, and τ_4 (the philosophers), which share 4 informational resources r_1, r_2, r_3, and r_4 (the forks). Task phases are 10, 7, 4, and 1 respectively; in 2 units after its start the task τ_1 locks the resource r_1 and after 4 units more it locks the resource r_2. Then after 20 time units it unlocks r_1 and in 68 units more it unlocks r_2. After 1000 time units or more since its start, the task τ_1 reiterates. Other tasks behave similarly with 73, 79, and 85 time units rather than 68 for unlocking their second resource (left fork). In the formalism of Fig. 1 the behavior of task τ_1 may be specified as (others are similar):

```
<task 10 =phase 1000 =period />
    1 <lock 2 />  2 <lock 4 />
    1 <unlock 20 />  2 <unlock 68 />
 <end 2 />  </task>
```

With the specified phases and timings for locking/unlocking resources, a clinch occurs at time=25, as Fig. 6 displays this with the log obtained by the simulator.

TimeLimit=1000000 JobLimit=0 ViolationLimit=0	**Interpretation/Comments**
SchedulingMode=RM	Rate Monotonic with
AccessProtocol=PI	Priority Inheritance
Configuration file name: c:\MPE\App_4PhD.txt	
Time=1 Proc=0 for 1 A 4.1	Task 4 (job 4.1) activates
Time=3 Proc=4.1 for 2 L 4.1 of 4	Task 4 (job 4.1) locks res.4
Time=4 Proc=4.1 for 1 A 3.2	Task 3 (job 3.2) activates
Time=6 Proc=3.2 for 2 L 3.2 of 3	Task 3 (job 3.2) locks res.3
Time=7 Proc=3.2 for 1 A 2.3	Task 2 (job 2.3) activates
Time=9 Proc=2.3 for 2 L 2.3 of 2	Task 2 (job 2.3) locks res.2
Time=10 Proc=2.3 for 1 A 1.4	Task 1 (job 1.4) activates
Time=12 Proc=1.4 for 2 L 1.4 of 1	Task 1 (job 1.4 locks res.1
Time=16 Proc=1.4 for 4 W 1.4 of 2	Task 1 (job 1.4) waits res.2
Time=19 Proc=2.3 for 3 W 2.3 of 3	Task 2 (job 2.3) waits res.3
Time=22 Proc=3.2 for 3 W 3.2 of 4	Task 3 (job 3.2) waits res.4
Time=25 Proc=4.1 for 3 **Mutual clinch for job 4.1 on resource 1** ok	Clinch detected for task 4 (job 4.1) when it tried to lock resource 1 at time=25

Fig. 6. System log for the 4 philosophers puzzle

The resource status displayed by the word .resources confirms this clinch. As one can see there's a vicious circle of locked resources with mutually waiting jobs:

```
Resource_1 Prio=0 Status=Job 1.4 JobsWaiting=NULL
Resource_2 Prio=0 Status=Job 2.3 JobsWaiting=Job 1.4
Resource_3 Prio=0 Status=Job 3.2 JobsWaiting=Job 2.3
Resource_4 Prio=0 Status=Job 4.1 JobsWaiting=Job 3.2
```

This example demonstrates the necessity of certain means for dynamic detecting of clinches in multi-task applications and feasibility of such preventive measures which increase the robustness of the overall system and allow to use more efficient data access protocols and scheduling modes.

VII. Conclusion

The simulator was written in ANS-Forth 200x [7] with VFX Forth for Windows, version 4.70, provided to the author at the courtesy of MPE [8], and is just 985 lines of code under the respective coding standards. It uses only fixed-point arithmetic and works remarkably fast on a PC.

To avoid memory overflow, the simulator uses its own simple subsystem for memory allocation and reuse for chained list elements, jobs and events. Further work will be focused on improving the user interface, extending the nomenclature of scheduling modes and access protocols of this simulator, and transition to simulation of multi-core and multiprocessor platforms, as well as running more experiments with models of real-time multi-task applications.

The described programming solution based on data structures EventList and JobList which control the simulation process turned out to be both effective and efficient, so it is worth for reuse in other applications or subject domains. The described simple system log allows for relatively easy detecting violations and errors in the simulation process and helps in debugging the simulator and its input data.

This Forth-based simulator was developed in parallel with another one developed on C# and results of these two dissimilar implementations were compared on a number of benchmarks. One of the differences between the two implementations was that the Forth-based one used only fixed-point arithmetic while the that in C# used floating-point. The results turned out to be remarkably close taking into account the difference in rounding rules.

Though "production quality" of the developed software may be argued, the total number of real defects found in fixed (4 major and 5 minor) after intensive testing suggests high enough quality of this tool.

This work was partially financially supported by Government of the Russian Federation, Grant 074-U01.

References

[1] VFX Forth for Windows. User manual. Manual revision 4.70, 19 August 2014. – Southampton: MicroProcessor Engineering Limited, 2014. – 429 p.

[2] gForth. Free Software Foundation, Inc. – [computer resource] https://www.gnu.org/software/gforth/ .

[3] Andersson B., Baruah S., Jonsson J. "Static-Priority Scheduling on Multiprocessors", *Proc. of 22nd IEEE Real-Time Systems Symposium*. – London, 2001. – P.193-202.

[4] Nikiforov V.V., Shkirtil V.I. "Specification of interfaces in real-time software applications by XML forms", *SPIIRAS Proceedings*, 2009, issue 11. – P. 159-175. (In Russian.)

[5] Baranov S.N., Nikiforov V.V. "Density of Multi-Task Real-Time Applications", *Proceedings of the 17th Conference of Open Innovations Association FRUCT*, Yaroslavl, Russia, 20-24 April 2015. – P.9-15.

[6] Dijkstra E.W. "Hierarchical ordering of sequential processes", *Acta Informatica* 1(2), 1971. – P.115-138.

[7] Forth extension proposal RfDs and CfVs. – [computer resource] http://www.forth200x.org/ .

[8] MicroProcessor Engineering Limited. – [computer resource] http://www.mpeforth.com .