Steps towards the Automatic Derivation of Performance Models from the Mascot Design Method

Ramon Puigjaner
Universitat de les Illes Balears
Departament de Ciències Matemàtiques i Informàtica
Carretera de Valldemossa, Km 7,6
E-07071 PALMA (Spain)
Tel: +34-71-173288
Fax: +34-71-173003
e-mail: dmirpt0@ps.uib.es

Albert Llamosi
Universitat Rovira i Virgili
Departament d'Enginyeria Informàtica
Autovia de Salou s/n
43071 TARRAGONA
Tel: +34-77-559677
Fax: +34-77-559710
e-mail: allamosi@etse.urv.es

ABSTRACT.
An approach for estimating the performance of real-time and concurrent systems during the design is presented. The fact that the estimations can be done before the implementation step allows to tune the design with important savings of time and effort. The notations for design description as well as the model for execution have been taken from the Mascot method. Simple cases of its communication patterns are modelled in detail.

KEYWORDS.
Mascot, Performance modelling, Real-time systems, Concurrent systems, Simulation.

1. INTRODUCTION
Real-time systems are defined as systems having performance constraints either on response time or on throughput or on device utilisation rate. A distributed implementation, so concurrent, is a frequent solution for such kind of systems to take into account its geographical constraints. With conventional techniques for design and implementation the real-time constraints are not considered until the testing phase, when the systems are checked, whereas during the design step, no estimation can be done about the performance constraints. Indeed, in accordance with [SMIT90], important savings of design effort and backtracking could be achieved if performance modelling techniques were included during early phases of the design process in order to have estimations of the performance constraints. A possible solution to include performance oriented information in the design is to create a new design method as it is proposed in [OPDA92, VETL93]. Another approach is to annotate the design with supplementary data concerning the estimations about the elementary times to execute the accesses to the logical and/or physical design entities. This solution may lead to infer the performance model for the full system taking into account the delays generated by the logical lockings and by the conflicts on resource use. Then, the obtained model can be studied by means of either analytical or simulation techniques. So that progressively accurate performance estimations can be obtained as long as the system design is further refined. The technique suggested here developed in the framework of the Esprit project COMPLEMENT\(^1\) intends to be a contribution towards such approach with the trend to automate this process. It takes Mascot [BATE86, JACK86, MASC91] as the framework for systems design and provides the performance models that correspond to simplest cases of its essential basic components. The models are expressed in the modelling language QNAP2. Therefore, a system which is made up by composition of such components can be easily modelled in order to obtain the desired estimations for its real-time constraints. The results obtained from these models should allow the designers to select among several design options taking into account the estimated performance of each one. A previous version of this work has been presented in [JACK92]. Actually, the approach does not fit completely the goal of making design and performance estimations go hand by hand because the derivation of the performance model requires that the system or subsystem to be analysed is designed in terms of basic components (i.e. fully decomposed).

\(^1\) This work has been partially funded by the Commission of the European Communities in the framework of the ESPRIT II project COMPLEMENT (project no. 5409) and also by the Spanish Comisión Interministerial de Ciencia y Tecnología (CICyT) under grant no. TIC-1310-CE.

Epresentatives of the key family of process interactions, namely channels, are modelled in detail. Certainly, a designer can introduce further kinds of interactions and in this case he should make a development analogous to the ones described below in order to model his system. However, the provided models are useful for a large class of systems. Two levels of performance can be explored. On one hand, the logical performance model built from the system logical design, that should lead to estimations of response times and throughputs to be obtained. And on the other hand, the physical performance model
built from the system's physical design (i.e., the implementation of the previous one taking into account the mapping between the logical components and the hardware resources, their geographical distribution and the implementation details of the software components) that should lead to achieve estimations of response times, throughputs and resource utilisation rates.

2. MASCOT

Mascot was developed during the early seventies and has evolved since then. The current official version can be found in MASC91, being BATE86 an excellent tutorial. The basic components of Mascot are processes, intercommunication data areas (IDAs) and paths. A path is a type specification of a set of interaction procedures. When an IDA provides actual procedures that match such a type specification it is said that it offers a window associated to that path. When a process or an IDA require actuals for a certain path, it is said that they have ports. Figure 2.1 describes the graphical representation for those components.

![Figure 2.1.](image)

The declaration of a process or an IDA only provides templates, not actual components. These ones must be obtained through instantiation of the templates and it is then that the correspondence between ports and windows is established. In the graphical representation the names of the templates are written inside their figures, whereas the names of the instances are written besides. The link between the port and the window is established drawing an arrow which is labelled with the corresponding path name (figure 2.2.).

![Figure 2.2.](image)

Processes and IDAs, related through paths, make up subsystems, which are described in Mascot by means of round cornered boundaries that can have both ports and windows, as the figure 2.3. shows.

![Figure 2.3.](image)

Subsystems are the fundamental means in Mascot for providing a comprehensive hierarchical organisation of a system either if it is designed top-down or bottom-up. Subsystems are so decomposed or built up as articulations of processes, IDAs and further subsystems (Figure 2.4.). A complete system differs from a subsystem only in that it is closed, i.e. it does not offer nor require external windows or ports. Windows offered by a subsystem are actually implemented by windows of some of its components. Likewise a subsystem requires a port because some of its components require it. It is then said that ports and windows of the subsystem are echoed by ports and windows of the subsystem. The declaration of a subsystem, like it was the case for processes and IDAs, does not lead to an actual component, but to a template. Actual components are obtained through the proper instantiation. In order to improve the semantic richness of Mascot diagrams, some special cases of IDAs and processes have specific shapes. Namely, pools, channels and servers (Figure 2.5.). Pools are those data that are updated and whose consults always get the value of the last update. Channels are those data that are buffered in a first in first out basis. And servers are those design elements which are able to communicate with external hardware devices. Hardware devices are represented as hatched rectangles, joined to their servers by a broken line.
In Mascot's concurrency model, processes can operate in parallel on a single IDA, and it is the programmer's responsibility to introduce explicitly mutual exclusion or whatever more sophisticated synchronisation mechanism. This is achieved by means of variables of the built-in type controlq, which can be defined in access interfaces and behave in a way close to that of the queues in the monitors of concurrent Pascal. The main operations on the type controlq are join and leave, to get and release mutual exclusion on the queue, and wait and stim, to conditionally suspend and resume process execution. A Mascot design can be targeted to any language because a kernel is provided, which supplies the interprocess synchronisation facilities. A possible mapping to Ada is discussed in [JACK86].

3. PERFORMANCE MODEL FOR A SIMPLE CHANNEL

There are several variants for the concept of channel. The simplest one is the bounded buffer where producers await in the case that the buffer is full and resume their operation as soon as their data has been placed in the buffer. This situation is modelled by two semaphores that are associated to the two critical conditions of buffer empty and buffer full plus two servers, CONCONS and CONPROD, for control of consumer and control of producer, respectively, that simulate the possible lock time due to the logical waits plus the delays introduced by the scheduling operations. The model can be simulated by the following code written in QNAP2:

```
/DECLARE/ QUEUE BUF_FULL, MES_PEND,
            CONCONS, CONPROD,
            BUFFER;
     INTEGER BUF_SIZ;
     REAL TPRMENS, TCMENS;
/STATION/ NAME = BUF_FULL;
          TYPE = SEMAPHORE, MULTIPLE(BUF_SIZ);
/STATION/ NAME = MES_PEND;
          TYPE = SEMAPHORE, MULTIPLE(0);
/STATION/ NAME = CONPROD;
          SERVICE = BEGIN
              P(BUF_FULL);
              EXP(TPRMENS);
              TRANSIT(NEW(CUSTOMER), BUFFER)
              V(MES_PEND);
              TRANSIT(CUSTOMER, CUSTOMER.ORIGIN);
          END;
/STATION/ NAME = CONCONS;
          SERVICE = BEGIN
              P(MES_PEND);
              EXP(TCMENS);
              TRANSIT(BUFFER.FIRST, DESTINATION);
              V(BUF_FULL);
              TRANSIT(CUSTOMER.ORIGIN);
          END;
```
The station **CONPROD** simulates the operation of putting a message on a buffer. A **P** operation is performed on the semaphore associated to the condition buffer full, so that a logical lock is simulated in the case that the buffer is full. When there is space for placing the message, a time consumption is simulated by the operation \( \text{EXP (TPRMENS)} \). In this case, it has been assumed that the time consumed is randomly distributed following an exponential law whose mean is **TPRMENS**. If other assumptions were more accurate, for instance constant or hyperexponential time, a proper substitution should replace \( \text{EXP (TPRMENS)} \). Then a copy of the **CUSTOMER** is put in the **BUFFER**. Once the message is considered to be in place, a **V** operation is sent to the semaphore associated to the condition buffer non empty. Finally, control is sent back to the station that had requested the service. It has been assumed that station asking services has been marked as an attribute for the customer. The undefined elements have been represented in italics. A similar description applies to **CONCONS**, the station that simulates the control operations for accessing the buffer by a consumer. In the provided code, it has been assumed that the placing and the removing of a message in the buffer can proceed in parallel. In the case that the implementation that the designer has in mind requires mutual exclusion for the two operations, a further semaphore should be introduced, to be requested immediately before the simulation of the time consumption and to be released immediately after the time consumption by the two stations.

4. PUTTING PIECES ALTOGETHER

So far, models for the most elementary forms of Mascot IDAs have been presented. But they are meaningful when are combined with activities that interact through them. Therefore, in order to obtain estimations of response times, the system designer should make by himself the QNAP2 models that keep the essential features of his activities, and combine them with those corresponding to the IDAs in order to make up a full system that can be simulated.

It is hard to make a general treatment of the matter, since each system requires a specific model. However, an example can illustrate how to make the modelling work. Assume a simple producer-consumer system that exchange messages through a bounded buffer such as the one described in the diagram shown in figure 5.1 whose queueing network representation appears in figure 5.2. The producer is assumed to spend some time in computing a message, which is randomly distributed following an exponential law of mean **TPRD** and then it sends its data to the buffer, awaiting in the case of finding the buffer full. Similarly, the consumer spends some time in consuming the message, which has been assumed to be randomly distributed following an exponential law of mean **TCNS**. As a first approach, the models for such processes would be very simple:

```
/DECLARE/
QUEUE PROD, CONS;
REAL TPRD, TCNS;
```
/STATION/  NAME = PROD;
  INIT = 1;
  SERVICE = BEGIN
  EXP(TPRD);
  TRANSIT(CONPROD);
  END;

/STATION/  NAME = CONS;
  INIT = 1;
  SERVICE = BEGIN
  EXP(TCNS);
  TRANSIT(CONCONS);
  END;

The only thing that makes them a little bit more sophisticated than it could be naturally expected is the lack of the return concept in QNAP2 for clients with respect to services. In order to resume the producer process once the message has been placed in the buffer, the CONPROD station must know the source station for its customer, and the same applies to the station that controls the consumer. The full system then becomes:

/DECLARE/  QUEUE  BUF_FULL, MES_PEND, CONCONS, CONPROD, BUFFER;
  INTEGER  BUF_SIZ;
  REAL  TPMENS, TCMENS;
  QUEUE  PROD, CONS;
  REAL  TPRD, TCNS;
  CUSTOMER  REF  QUEUE  ORIGIN;

/STATION/  NAME = BUF_FULL;
  TYPE = SEMAPHORE, MULTIPLE(BUF_SIZ);

/STATION/  NAME = MES_PEND;
  TYPE = SEMAPHORE, MULTIPLE(0);

/STATION/  NAME = CONPROD;
  SERVICE = BEGIN
  P(BUF_FULL);
  EXP(TPMENS);
  TRANSIT(NEW(CUSTOMER), BUFFER);
  V(MES_PEND);
  TRANSIT(CUSTOMER.ORIGIN);
  END;

/STATION/  NAME = CONCONS;
  SERVICE = BEGIN
  P(MES_PEND);
  EXP(TCMENS);
  TRANSIT(BUFFER.FIRST, OUT);
  V(BUF_FULL);
  TRANSIT(CUSTOMER.ORIGIN);
  END;

/STATION/  NAME = PROD;
  INIT = 1;
  SERVICE = BEGIN
  EXP(TPRD);
  CUSTOMER.ORIGIN:= QUEUE;
  TRANSIT(CONPROD);
  END;

/STATION/  NAME = CONS;
  INIT = 1;
  SERVICE = BEGIN
  EXP(TCNS);
  CUSTOMER.ORIGIN:= QUEUE;
  TRANSIT(CONCONS);
  END;
The INIT = 1 sentences tell that actual clients are placed initially in those stations. The queue OUT is predefined and is the sink for the customers sent to it. Now everything is ready to start the simulation, which is achieved after providing values for the parameters of the system. The time constants are obviously assumed to be provided by the model builder all in the same unit (milliseconds, for instance), which is the same in which the results will be provided:

```
/CONTROL/  TMAX = 20000;
          ACCURACY = ALL QUEUE;
/EXEC/ BEGIN
          BUF_SIZE:= 5;
          TPMENS:= 1.0;
          TCMENS:= 0.7;
          TPRD:= 4.0;
          TCNS:= 3.5;
          SIMUL;
END;
```

Results of this example can be found in listing 1 at the end of this work. In the case that the activities require a more complex modelling, conditionals can be introduced, whose branching will usually depend on random number generation, in the style of

```
I:= RINT(1, 4);
IF J<4 THEN ... ELSE...
```

In order to show how several individual models can be connected to create the model of a more complex system let consider the Mascot structure shown in figure 5.3, that can be considered as the connection of two channels by means of a communication line. In this case the reader and the writer that act over the buffer have been replaced by two sources of readers and writers acting in such a way that they leave the system after their actions; this kind of modelling is completely equivalent to the one used in the previous model. The equivalent queueing network scheme appears in figure 5.4. The QNAP2 model is the following:

```
/DECLARE/ QUEUE CONS, PROD, MBUF_FUL, MMES_PEN,
     BUFFER, CONPROD, CONCONS,
     COM, SBUF_FUL, SMES_PEN, SBUFFER, SCONPROD, SCONCONS;
     INTEGER SBUF_SIZ, MBUF_SIZ;
     REAL TPROD, TCNS, TCMENS, TPRMENS, TCOM, STCMENS, STPRMENS;

/STATION/ NAME=SCONPROD;
          SERVICE=BEGIN
                     P(SBUF_FULL);
              EXP(STPRMENS);
              V(SMES_PEND);
        TRANSIT(SBUFFER);
          END;
```

![Diagram of Mascot structure](image)

**Figure 5.3.**
/STATION/ NAME=SCONCONS;
INIT=1;
SERVICE=BEGIN
  P(SMES_PEN);
  EXP(STCMENS);
  V(SBUF_FUL);
  TRANSIT(SBUFFER.FIRST,COM);
  TRANSIT(SCONCONS);
END;

/STATION/ NAME=CONPROD;
SERVICE=BEGIN
  P(MBUF_FUL);
  EXP(TPRMENS);
  V(MMES_PEN);
  TRANSIT(BUFFER);
END;

/STATION/ NAME=CONCONS;
SERVICE=BEGIN
  P(MMES_PEN);
  EXP(TCMENS);
  V(MBUF_FUL);
  TRANSIT(BUFFER.FIRST,OUT);
  TRANSIT(OUT);
END;

/STATION/ NAME=PROD;
TYPE=SOURCE;
SERVICE=BEGIN
  EXP(TPROD);
  TRANSIT(SCONPROD);
END;
Results of this example can be found in listing 2 at the end of this work.

5. MAPPING PROCESSES ON HARDWARE SUPPORT

So far, hardware resources have been considered to be always available, which is not the case if several processes share the same resources such as a common CPU, disk or virtual memory. From a different point of view it will be also interesting to know from the simulation the utilisation rate of the hardware resources under different possible mappings. Or even the mean occupation level of the buffers.

In order to incorporate this aspect into the modelling, the resources (CPU, for example) shall be considered as stations. Again, the lack of a return primitive in QNAP2 requires the introduction of a somewhat cumbersome mechanism consisting on numbered flags, which are raised when the resource operation becomes completed. Also, the customer incorporates the expected consumption in the cases where it can be variable. The modelling of the CPU therefore becomes:

/DECLARE/ QUEUE CPU;
FLAG CPUFL(#call_points);
REF CUSTOMER C;
CUSTOMER INTEGER CALL_POINT;
CUSTOMER REAL CONSUMPTION;

/STATION/ NAME = CPU;
SERVICE = BEGIN
CST(CUSTOMER.CONSUMPTION);
SET(CPUFL(CUSTOMER.CALL_POINT));
TRANSIT(OUT);
END;

And the calling point will require the steps:
C := NEW(CUSTOMER);
C.CALL_POINT := px;
C.CONSUMPTION := EXP(tx);
UNSET(CPUFL(px));
TRANSIT(C, CPU);
WAIT(CPUFL(px));

Each call point (px) has a flag associated plus an expected consumption, which has been assumed in this case to be randomly distributed by an exponential law whose mean is tx. Upon call, a new copy of the current customer is created, and its attributes for the calling point placement and expected CPU time consumption updated. Then the flag associated to the calling point is unset and the customer's copy sent to the station emulating the CPU. Then a waiting operation is executed in order to await for completion of the CPU execution. It is left as an exercise for the reader to extend to the case with just one CPU the former naïve producer-consumer system. The results of this case with CPUs working with FIFO and time sharing policies can be found in listing 3 and 4 at the end of this work.

6. FURTHER SOPHISTICATION OF THE MODELS

Real systems are usually more complex than the examples shown above in order to illustrate the approach. On one hand, the management of channels is usually far more complex than the simple bounded buffer and the readers/writers policy on a common data area. On the other hand, the scheduling policy can influence meaningfully the behaviour of the system, and therefore must be introduced in the model.

By default, in QNAP2 customers execute services on stations following a non preemptive FIFO policy. This is usually a correct assumption when the services simulate software modules, but may fail when the services simulate actual shared resources such as CPU or disk. In those cases QNAP2 must effectively simulate the scheduling policy that will be applied in the implementation of the system. This can be introduced by the SCHED clause:

/STATION/     NAME = CPU;
               SCHED = ...;
               SERVICE = as before;

The scheduling can be prescribed to be
- either preemptive or not (if not, a customer is allowed to end the service with no interruption once it has started),
- either time sharing or not (if yes the customer will be interrupted each quantum of time, that can be specified, and then the scheduler will select the next customer to be served),
- either FIFO or LIFO,
- such that it can consider the customer's priorities.

Less mechanical to be solved is the modelling of complex access procedures to IDAs. There is not a general rule for building a model from a given set of procedures. The designer should make the model for each special case and the solutions proposed above should serve only as a pattern to follow and provide step by step a library of models representing the timing behaviour of different IDAs. In the references [PUIG91, PUIG92A, PUIG92B, PUIG92C] the reader can find the detailed modelling of more complex cases such as a buffer with priorities, pools of several kinds, buffers and pools that involve internal network communication.

7. INTERPRETATION OF THE RESULTS

The simulation process supplies key data for system design. The set of data to be obtained, as well as their accuracy are established in QNAP2 by the control clause. Standard output provides, for all stations, the mean time of each execution, the percentage of time they have been busy, the mean number of customers awaiting the service and the mean response time, as well as the number of services executed. The values are provided together with their confidence intervals, as the listings appended at the end of this writing show. They all correspond to the producer-consumer example. In the first case the processes are assumed to be run on independent CPUs, in the second case they are run on a single CPU under a non-preemptive FIFO scheduling policy and in the third case they are also run on a single CPU but the policy has been time sharing with a quantum of 0.1 units.

8. CONCLUDING REMARKS

The ability to estimate the performance of real-time and embedded systems has evident consequences in savings of implementation effort. The approach suggested above allows to achieve performance data for systems which have been designed at the detail level of IDAs and activities. In the case that IDAs are simple bounded buffer and readers/writers accessed data, the models for those components can be directly
taken from those shown in sections 2 and 3. In the case of designing a more complex component, the
designer should extend the proposed models accordingly to the designed behaviour.
However, trying a similar approach for subsystems, in order to make the estimations as long as the system
is refined, is a horse of another color, because a subsystem can reveal some combined active behaviour that
can be hardly modeled, from the performance point of view, unless the different elementary components
are known. Another critical aspect is the estimation of the expected time consumption constants that are
required to feed the model. Only empirical data coming from similar modules run on comparable hardware
support can provide reliable values.

9. REFERENCES
BATE86 Bate, G., "Mascot3: an informal introductory tutorial", *Software Engineering Journal*, (May
JACK92 Jackson, K., Llamosf, A. and Puigjaner, R. "Specification of Performance Constraints in
(Florida, USA), November 1992, pp. 243-251.
MASC91 *The Official Handbook of MASCOT*, version 3.1. British Standards Institution, BSI DD
PUIG91 Puigjaner, R., *Paradigms on CPE Modeling*, Complement Project, doc. n. UIB9-2.0,
December 1991.
Project, doc. n. UIB61-1.0, June 1992.

Listing 1: producer-consumer running each on a different CPU

```
* NAME   * SERVICE   * BUSY PCT  * CUST NB  * RESPONSE  * SERV NB  *
*         *           *           *           *           *           *
* BUF_FULL* 0.0000E+00* 0.0000E+00* 0.3511E-01* 3.745*       * 187*
* +/--     * 0.0000E+00* 0.0000E+00* 0.1093E-01* 0.4278*  * *
*          *           *           *           *           *           *
* MES_PEND* 0.0000E+00* 0.0000E+00* 0.1925*     * 3.873*       * 994*
* +/--     * 0.0000E+00* 0.0000E+00* 0.2427E-01* 0.2778*  * *
*          *           *           *           *           *           *
* CONCONS  * 1.696*    * 0.3270*    * 0.3270*    * 1.696*    * 3855*
* +/--     * 0.1423*    * 0.2192E-01* 0.2192E-01* 0.1423*  * *
*          *           *           *           *           *           *
* CONPROD  * 1.197*    * 0.2310*    * 0.2310*    * 1.197*    * 3860*
* +/--     * 0.5625E-01* 0.1211E-01* 0.1211E-01* 0.5625E-01*  * *
* BUFFER   * 0.0000E+00* 0.0000E+00* 1.570*    * 8.124*    * 3855*
* +/--     * 0.0000E+00* 0.0000E+00* 0.1761*    * 0.9314*  * *
* PROD     * 3.983*    * 0.7690*    * 0.7690*    * 3.983*    * 3861*
* +/--     * 0.1621*    * 0.1211E-01* 0.1211E-01* 0.1621*  * *
* CONS     * 3.491*    * 0.6730*    * 0.6730*    * 3.491*    * 3856*
* +/--     * 0.1212*    * 0.2192E-01* 0.2192E-01* 0.1212*  * *
```
<table>
<thead>
<tr>
<th>NAME</th>
<th>SERVICE</th>
<th>BUSY PCT</th>
<th>CUST NB</th>
<th>RESPONSE</th>
<th>SERV NB</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONS</td>
<td>3.028</td>
<td>1.000</td>
<td>1.000</td>
<td>3.028</td>
<td>33023</td>
</tr>
<tr>
<td></td>
<td><em>0.3567E-01</em>0.00000E+00<em>0.00000E+00</em>0.3567E-01*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PROD</td>
<td>3.051</td>
<td>1.000</td>
<td>1.000</td>
<td>3.051</td>
<td>32775</td>
</tr>
<tr>
<td></td>
<td><em>0.3342E-01</em>0.00000E+00<em>0.00000E+00</em>0.3342E-01*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MBUF_FUL</td>
<td><em>0.0000E+00</em>0.0000E+00*0.1735</td>
<td>3.057</td>
<td>5676</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>0.0000E+00</em>0.0000E+00*0.1064</td>
<td><em>0.7921E-01</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MMES_PEN</td>
<td><em>0.0000E+00</em>0.0000E+00*0.4915</td>
<td>3.060</td>
<td>16064</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>0.0000E+00</em>0.0000E+00*0.1300</td>
<td><em>0.4432E-01</em></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BUFFER</td>
<td><em>0.0000E+00</em>0.0000E+00*5.495</td>
<td>16.77</td>
<td>32774</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>0.0000E+00</em>0.0000E+00*3.664</td>
<td>11.36</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CONPROD</td>
<td>1.547</td>
<td>*0.5069</td>
<td>15.63</td>
<td>47.69</td>
<td>32774</td>
</tr>
<tr>
<td></td>
<td>*0.4858</td>
<td>*0.1333</td>
<td>12.55</td>
<td>37.19</td>
<td></td>
</tr>
<tr>
<td>CONCONS</td>
<td>2.506</td>
<td>*0.8214</td>
<td>104.0</td>
<td>314.1</td>
<td>32774</td>
</tr>
<tr>
<td></td>
<td>*0.3450</td>
<td>*0.1161</td>
<td>39.11</td>
<td>122.5</td>
<td></td>
</tr>
<tr>
<td>COM</td>
<td>1.012</td>
<td>*0.3318</td>
<td>*0.4986</td>
<td>1.521</td>
<td>32774</td>
</tr>
<tr>
<td></td>
<td><em>0.1415E-01</em>0.6777E-02<em>0.1464E-01</em>0.3404E-01*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBUF_FUL</td>
<td><em>0.0000E+00</em>0.0000E+00<em>0.6266E-02</em> 1.071</td>
<td>585</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>0.0000E+00</em>0.0000E+00<em>0.1126E-02</em>0.1921</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMES_PEN</td>
<td><em>0.0000E+00</em>0.0000E+00*0.6689</td>
<td>3.049</td>
<td>21936</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>0.0000E+00</em>0.0000E+00<em>0.5601E-02</em>0.3751E-01*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBUF</td>
<td><em>0.0000E+00</em>0.0000E+00*0.4883</td>
<td>1.490</td>
<td>32774</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>0.0000E+00</em>0.0000E+00<em>0.1263E-01</em>0.3114E-01*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCONPROD</td>
<td>1.026</td>
<td>*0.3362</td>
<td>*0.5085</td>
<td>1.551</td>
<td>32774</td>
</tr>
<tr>
<td></td>
<td><em>0.1324E-01</em>0.6327E-02<em>0.1670E-01</em>0.4046E-01*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCONCONS</td>
<td>3.051</td>
<td>1.000</td>
<td>1.000</td>
<td>3.051</td>
<td>32774</td>
</tr>
<tr>
<td></td>
<td><em>0.3442E-01</em>0.0000E+00<em>0.0000E+00</em>0.3442E-01*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Listing 3: producer-consumer running on a single CPU under non-preemptive FIFO scheduling policy

<table>
<thead>
<tr>
<th>NAME</th>
<th>SERVICE</th>
<th>BUSY PCT</th>
<th>CUST NB</th>
<th>RESPONSE</th>
<th>SERV NB</th>
</tr>
</thead>
<tbody>
<tr>
<td>MES_PEND</td>
<td>0.0000e+00</td>
<td>0.0000e+00</td>
<td>0.1366e-03</td>
<td>2.732</td>
<td>1*</td>
</tr>
<tr>
<td></td>
<td>-1.000</td>
<td>0.0000e+00</td>
<td>0.3169e-03</td>
<td>-1.000</td>
<td>*</td>
</tr>
<tr>
<td>CONCONS</td>
<td>4.605</td>
<td>0.5024</td>
<td>0.5024</td>
<td>4.605</td>
<td>2182*</td>
</tr>
<tr>
<td></td>
<td>0.1666</td>
<td>0.1105e-01</td>
<td>0.1105e-01</td>
<td>0.1666</td>
<td>*</td>
</tr>
<tr>
<td>CONPROD</td>
<td>1.682</td>
<td>0.1836</td>
<td>0.1836</td>
<td>1.682</td>
<td>2183*</td>
</tr>
<tr>
<td></td>
<td>0.6749e-01</td>
<td>0.6535e-02</td>
<td>0.6535e-02</td>
<td>0.6749e-01</td>
<td>*</td>
</tr>
<tr>
<td>BUFFER</td>
<td>0.0000e+00</td>
<td>0.0000e+00</td>
<td>0.8916</td>
<td>8.170</td>
<td>2182*</td>
</tr>
<tr>
<td></td>
<td>0.0000e+00</td>
<td>0.0000e+00</td>
<td>0.4612e-02</td>
<td>0.2039</td>
<td>*</td>
</tr>
<tr>
<td>CPU</td>
<td>2.290</td>
<td>0.1000</td>
<td>0.1000</td>
<td>2.290</td>
<td>8730*</td>
</tr>
<tr>
<td></td>
<td>0.5070e-01</td>
<td>0.3169e-03</td>
<td>0.3169e-03</td>
<td>0.9996e-01</td>
<td>*</td>
</tr>
<tr>
<td>PROD</td>
<td>7.478</td>
<td>0.8164</td>
<td>0.8164</td>
<td>7.478</td>
<td>2183*</td>
</tr>
<tr>
<td></td>
<td>0.1954</td>
<td>0.6535e-02</td>
<td>0.6535e-02</td>
<td>0.1954</td>
<td>*</td>
</tr>
<tr>
<td>CONS</td>
<td>4.558</td>
<td>0.4976</td>
<td>0.4976</td>
<td>4.558</td>
<td>2182*</td>
</tr>
<tr>
<td></td>
<td>0.1415</td>
<td>0.1105e-01</td>
<td>0.1105e-01</td>
<td>0.1415</td>
<td>*</td>
</tr>
</tbody>
</table>

Listing 4: producer-consumer running on a single CPU under time-sharing (quantum of 0.1 units)

<table>
<thead>
<tr>
<th>NAME</th>
<th>SERVICE</th>
<th>BUSY PCT</th>
<th>CUST NB</th>
<th>RESPONSE</th>
<th>SERV NB</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUF_FULL</td>
<td>0.0000e+00</td>
<td>0.0000e+00</td>
<td>0.1912e-01</td>
<td>3.643</td>
<td>105*</td>
</tr>
<tr>
<td></td>
<td>-1.000</td>
<td>0.0000e+00</td>
<td>0.7883e-02</td>
<td>-1.000</td>
<td>*</td>
</tr>
<tr>
<td>MES_PEND</td>
<td>0.0000e+00</td>
<td>0.0000e+00</td>
<td>0.1069</td>
<td>3.859</td>
<td>554*</td>
</tr>
<tr>
<td></td>
<td>0.0000e+00</td>
<td>0.0000e+00</td>
<td>0.1920e-01</td>
<td>0.2281</td>
<td>*</td>
</tr>
<tr>
<td>CONCONS</td>
<td>2.372</td>
<td>0.2595</td>
<td>0.2595</td>
<td>2.372</td>
<td>2187*</td>
</tr>
<tr>
<td></td>
<td>0.1758</td>
<td>0.2044e-01</td>
<td>0.2044e-01</td>
<td>0.1758</td>
<td>*</td>
</tr>
<tr>
<td>CONPROD</td>
<td>1.935</td>
<td>0.2120</td>
<td>0.2120</td>
<td>1.935</td>
<td>2191*</td>
</tr>
<tr>
<td></td>
<td>0.9771e-01</td>
<td>0.1216e-01</td>
<td>0.1216e-01</td>
<td>0.9771e-01</td>
<td>*</td>
</tr>
<tr>
<td>BUFFER</td>
<td>0.0000e+00</td>
<td>0.0000e+00</td>
<td>1.666</td>
<td>15.18</td>
<td>2187*</td>
</tr>
<tr>
<td></td>
<td>0.0000e+00</td>
<td>0.0000e+00</td>
<td>0.2149</td>
<td>2.004</td>
<td>2187*</td>
</tr>
<tr>
<td>CPU</td>
<td>2.283</td>
<td>0.1000</td>
<td>0.1874</td>
<td>4.279</td>
<td>8758*</td>
</tr>
<tr>
<td></td>
<td>0.5638e-01</td>
<td>0.2595e-01</td>
<td>0.2595e-01</td>
<td>0.1209</td>
<td>*</td>
</tr>
<tr>
<td>PROD</td>
<td>7.190</td>
<td>0.7880</td>
<td>0.7880</td>
<td>7.190</td>
<td>2192*</td>
</tr>
<tr>
<td></td>
<td>0.4235</td>
<td>0.1216e-01</td>
<td>0.1216e-01</td>
<td>0.4235</td>
<td>*</td>
</tr>
<tr>
<td>CONS</td>
<td>6.769</td>
<td>0.7405</td>
<td>0.7405</td>
<td>6.769</td>
<td>2188*</td>
</tr>
<tr>
<td></td>
<td>0.3246</td>
<td>0.2044e-01</td>
<td>0.2044e-01</td>
<td>0.3246</td>
<td>*</td>
</tr>
</tbody>
</table>