Modeling and Analysis of Message-Queues in Multi-Tasking Systems

Thomas Nolte, Hans Hansson

Mälardalens Högskola/IDT
`thomas.nolte@mdh.se, hans.hansson@mdh.se`

This paper presents work in progress on an analysis method for message queues in a real-time multi-tasking system. This analysis will later be compared with simulation results, in which variations/distributions of execution times, task periods etc will be considered. The intention is to evaluate the level of pessimism in the analytical worst-case analysis, compared to the execution scenarios that actually occur in the real system. We will use these results as a basis for future work in the ARTES project RATAD, which aims at developing schedulability analysis with a wider applicability, by integrating reliability into real-time scheduling theory. What we want is a method for integrating reliability into classical real-time scheduling, so that we can guarantee, up to some level, that our system is working properly. This is to be compared with the classical 0/1 results of the schedulability theory.

1. Introduction

There is hardly any published work on schedulability analysis of message queues. There is however related work on scheduling messages in communication networks, including [2]. These models are however different from those needed for modeling standard multi-tasking operating system message queues (e.g. those proposed for POSIX [3]), in that the networking queues are separately scheduled resources, whereas the handling of the OS queues is performed by a CPU shared by many types of tasks, as well as the OS kernel.
Outline: In Section 2 we present our model and in Section 3 the method used for analysis is described. Finally our conclusion and future work is described in Section 4.

2. Problem formulation and model

We are assuming a producer-consumer scenario with a pre-emptive fixed priority task model. The inter-process communication is performed using message queues. Initially we assume the queues to be FIFO, but later we will extend the model to also cover priority queues.
The consumer has the following behavior

```
while(1){
       ...
```

```
        receive(…); //blocking
        …
}
```

Hence the consumer is blocked if nothing is to be received from the message queue. When the consumer is allowed to execute, it will (in the worst case) execute for C_c time units. This is repeated for every message that it consumes.

The producers are assumed to (in the worst case) behave as periodic tasks that at the end of their execution queue a message.

This paper is restricted to a scenario where all communication and execution is performed locally on a single node. We will for such a system consider a set of message queues, each handled by a single consumer, to which a set of associated producers queue messages as can be seen in Fig 1.

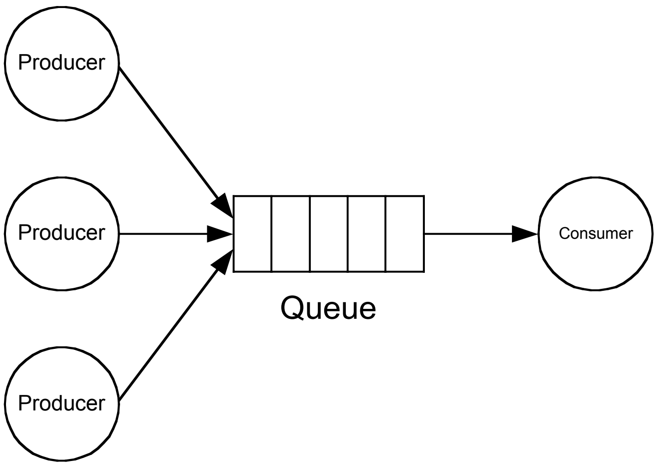

Fig. 1. System containing a set of producers and consumers that are communicating via a message queue.

The problem is here to calculate the worst-case end-to-end delay in a producer – message – consumer transaction. We will additionally calculate an upper bound of the message queue size.

The following terminology is used in this paper:

- T – The period of a task
- C – The worst case execution time of a task
- J – The jitter of a task. $J = r_{max} - r_{min}$, where r is the release time relative to the start of the period associated with each task. If $r_{min} = 0$ then $J = r_{max}$, which is what we will assume in the following.
- P – The priority of a task
- r – The release time
- $\#_p$ – A term associated with the producer
- $\#_c$ – A term associated with the consumer
- $queue_{size}$ – The maximum buffer need for the queue
- R – The worst case end-to-end response time associated with a producer – queue – consumer transaction

- $R_c^{\{0 \cdots n\}}$ – The response time to handle the n first messages in a busy period (a busy period being a time period during which the queue is continuously non-empty).

3. Analysis of message queues

We are about to analyse message queues for inter-task communication on a single node. We start this by a simple scenario, which we then extend to be more general and more complex. We have a producer-consumer scenario, where initially the consumer is blocked, waiting for a message to arrive in the message queue. Following standard fixed-priority schedulability analysis practice [4] we assume the producer to be released at most every T_p time units.

We will now derive some simple schedulability criteria and more detailed analysis for the above type of systems. The formulas presented here may not terminate unless load is less than 100%.

We begin with a single producer per queue and then extend to multiple producers per queue.

3.1. Single producer

First, we assume that we have a producer with a period, T, a certain amount of release jitter, J, execution time, C, and a priority, P. Further on we have a consumer with execution time and a priority as can be seen in Fig 2.

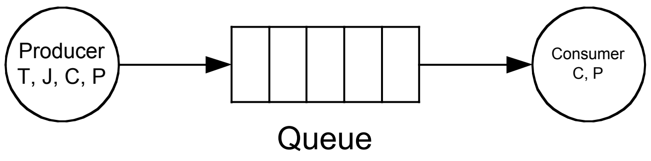

Fig. 2. The properties of the producer and the consumer task used in our model.

What we do now is to find out the worst end-to-end response time for this scenario. In order to find this response time we have to find the maximum response time of all messages produced within a Busy Period, i.e., the time it takes to completely empty the queue in a worst case scenario.

If we use R_n to denote the response time for the consumer to consume the n:th message, then the worst case end-to-end response time, R, is

$$R = \max_{n \in MBP} (R_n) \tag{1}$$

where MBP is the set of messages sent by all producers of a queue within the Busy Period.
What we need to do is to calculate the end-to-end response times for all the messages sent within the busy period. In order for the busy period to terminate, we require a load that is less than 100%. We use an approach similar to the iterative method used for analysis of task response times when deadline exceeds period [5] to iteratively

search for the maximum response time of the messages queued during the busy period. This amounts to initially assume that a single message is queued, and then calculate the response time for this message. If a second message has been queued during this response time we can calculate the response time for this message by analyzing the (in the response time sense) equivalent scenario when both messages are queued initially (i.e. when the consumer needs to be executed twice). If a third message is queued during the response time of the two messages, then we repeat the procedure for three messages. This continues for four, five, etc. messages until we reach a point when no new message is queued during the response time. That such a situation exists follows by the assumption that we have a utilization that is less than 1.

In the worst case scenario we will number consecutive messages in increasing order, starting with message 0.

We define the release time for a message as r. Then we assume that the message is released as soon as the producer is activated. In this way we cover all cases of messages being sent. The release time of message n is given by

$$r = (n * T_p) - J_p \tag{2}$$

Then we calculate the response time for the consumer using traditional exact analysis methods [1].

$$\begin{aligned} C &= (n+1) * C_c \\ w_c &= C + \sum_{j \in hp(tasks)} \left\lceil \frac{w_c + J_j}{T_j} \right\rceil C_j \\ R_c^{\{0 \cdots n\}} &= w_c \end{aligned} \tag{3}$$

When we have the response time for the handling of the n first messages, $R_c^{\{0 \cdots n\}}$, we can calculate the response time associated with message n simply by subtracting the release-time of the current message from the total response time.

$$R_n = R_c^{\{0 \cdots n\}} - r \tag{4}$$

We have now gathered the response times for the n first messages and we must investigate whether or not we have emptied the queue, since this is our termination criterion for the busy period. To do this we investigate whether or not the following condition is met, i.e., if the queue is not empty or not. If the following is true, (2)-(4) is repeated for the next message.

$$(n+1) < \left\lceil \frac{R_c^{\{0 \cdots n\}} + J_p}{T_p} \right\rceil \tag{5}$$

When we have emptied the queue we use (1) to find out the maximum response time present in the Busy Period. When we have the maximum response time we can use it to calculate the buffer need for the queue. The maximum queue size will be

$$queue_{size} = \left\lceil \frac{R + J_p}{T_p} \right\rceil \qquad (6)$$

Intuitively, this captures that the maximum queue size occurs when the worst-case end-to-end response time for a message is present.

3.2. Multiple producers

We have now looked at scenarios where we have a single producer feeding the queue with messages. We now extend this to handle multiple producers to a single queue. We introduce a set M containing all producers associated with each queue, that is all producers belonging to a certain queue. The procedure behaves the same as in the single producer scenario. The only difference is that the queue is receiving messages from multiple producers. If we first try to extract the worst end-to-end response time, we do as in the single producer scenario. What we need to handle is the scenarios where preemption involving several producers occurs, i.e., producers that gets preempted by other producers. If we want to calculate the worst end-to-end response time regardless of producer, all producers that are preempting us will in the worst case always manage to send their messages before us, as illustrated in Fig 3. Here we can see that if all higher priority producers will send their messages before us, we will be consumed as the last one, and therefore we will experience the longest end-to-end response time.

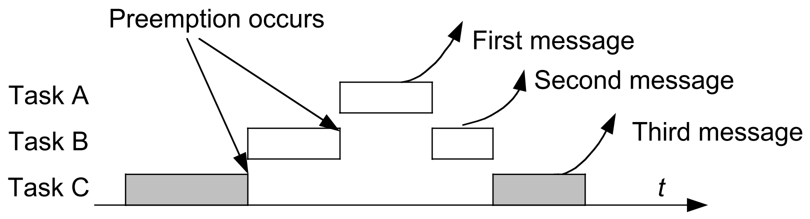

Fig. 3. The worst-case scenario regardless of which producer that is belongs to is always the one created for the producer with the lowest priority.

In this way, it is always the producer, which has the earliest release time of its current message that is interesting. What we need to do though, is to investigate whether or not some other producers will preempt this producer. If this is the case, then we always assume that the other producers, the ones that are preempting us, will produce their message before us. That is, we will always produce the worst scenario this way. If we do this, we can use similar methods to the ones used in the single producer scenario.

The situation becomes slightly more complicated when we are to extract the maximum end-to-end response time associated with a certain producer. This will force us to produce different execution scenarios depending on which producer we are interested in, since the actual time that each message is sent will influence the order of messages in the queue. The different scenarios, when having three producers associated with a queue, is the one described in Fig 3 (which is the worst case scenario for Task C) and the ones described in Fig 4 and Fig 5.

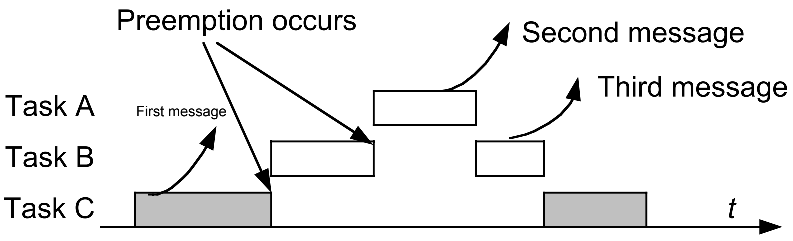

Fig. 4. The worst-case scenario for producer B is that all task with lower priority that we are preempting for the moment, actually has been able to already produce their messages. As usual, all preempting producers with higher priority than us will produce their messages before us.

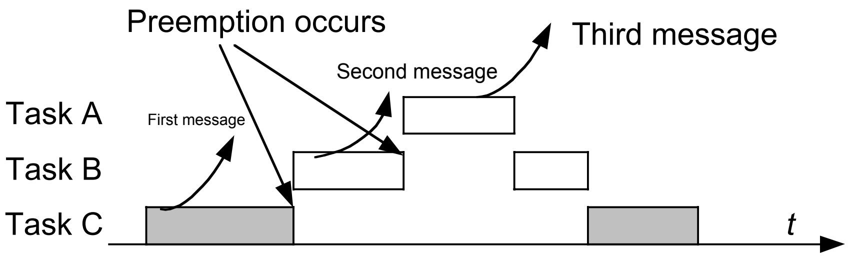

Fig. 5. The last scenario of our task set is when calculating the worst-case for the producer with the highest priority. Here all producers that we have preempted will produce their messages before us.

When we have taken these precautions, we need to modify our formulas used as the termination criteria, i.e., when the queue is empty, and our formula used for calculating the buffer need. What we will do is similar as for the single producer scenario, i.e., we extract all response-times until (5) is not met anymore, but in order to handle multiple producers we need to extend (5) to the following

$$(n+1) < \sum_{j \in M} \left\lceil \frac{R_c^{\{0 \cdots n\}} + J_j}{T_j} \right\rceil \tag{7}$$

When we are done, we can extract R using (1) in order to extract the maximum end-to-end response time so that we can calculate the required queue size. We need to modify (6) to handle multiple producers.

$$queue_{size} = \sum_{j \in M} \left\lceil \frac{R + J_j}{T_j} \right\rceil \tag{8}$$

4. Conclusion and future work

We have presented preliminary results from on-going work on including message queues in schedulability modeling and analysis. Our initial experiments indicate that it is not trivial to construct the worst-case scenario, especially when we are to collect worst-case end-to-end response time associated with each producer. When this is done though, we will extend the model to also cover messages with different priorities. This is essential, since many of the operating systems used in industry today, e.g. those that are POSIX compliant, support priority queues. When we have a good analysis method we will compare the results with simulation results. These results will then be integrated with some probabilistic model making it possible to integrate reliability theory with schedulability analysis.

References

[1] M. Joseph and P. Pandya. "Finding response times in a real-time system", BCS Computer Journal, 29(5): 390-395, 1986

[2] Mikael Sjödin, "Predictable High-Speed Communications for Distributed Real-Time Systems", Ph.D. Thesis, Uppsala university, Information Technology, Department of Computer Systems, May 2000

[3] IEEE Standard for Information Technology - Test Methods Specifications for Measuring Conformance to POSIX/sup R/ - Part 1: System Application Program Interface [API] - Amendment 1: Realtime Extension [C Language] IEEE Std 2003.1b-2000 , 2000

[4] Audsley, N.; Burns, A.; Richardson, M.; Tindell, K.; Wellings, A.J., "Applying new scheduling theory to static priority pre-emptive scheduling", Software Engineering Journal , Volume: 8 Issue: 5 , Sept. 1993, Page(s): 284 –292

[5] Tindell K, Hansson H, "Real Time Systems by Fixed Priority Scheduling", Department of Computer Systems, Uppsala University, 1996