Conformance Testing of Priority Inheritance Protocols

D. Zöbel and D. Polock

Universität Koblenz–Landau
Universitätsstr. 1
56075 Koblenz, Germany

zoebel@uni–koblenz.de

http://www.uni–koblenz.de/~agrt/

Granada – April, 2003
<table>
<thead>
<tr>
<th>Contents</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>2</td>
</tr>
<tr>
<td>2 Real-time Scheduling</td>
<td>8</td>
</tr>
<tr>
<td>3 The priority inversion problem</td>
<td>11</td>
</tr>
<tr>
<td>4 Formal model for priority inheritance protocols</td>
<td>20</td>
</tr>
<tr>
<td>5 Conformance testing</td>
<td>24</td>
</tr>
<tr>
<td>6 Conclusion</td>
<td>31</td>
</tr>
</tbody>
</table>
1 Introduction

What is a real-time system?

... besides semantical correctness the computation has to satisfy constraints in time ...

Examples:

- ABS in cars
- transmission of audio-data
- balancing a ball
The basic model of a real-time system

A system is said to be deterministic if for each possible state, and each set of inputs, a unique set of outputs and next state of the system can be determined (Laplante 1993)

Criteria:
- timeliness
- predictability
- robustness

Time-critical property of hard real-time systems:

\[ P[r_i + \Delta e_i \leq d_i | B] = 1 \]

\[ r_i \]
\[ \Delta e_i \]
\[ d_i \]
Categorization of computational processes
Matter of abstraction: process $P_i$

- a bound sequence of $P_i$'s individual operations
- a bound state space touched by these operations

Characterization of computational processes

$P$ (type of process): Move a step motor by $a$ positions.

$P_i$ (instance of process $P$): Move the step motor $i$ by $a$ positions.

$P_i^j$ (execution of process $P_i$): The $j$-execution of $P_i$. 
Activation of computational processes

Categorization of the activations of processes:

**periodical processes**: A process becomes ready for execution after a certain period \( \Delta p_i \) of time.

**Example**: The frame grabber provides a bitmap of the ball on the flat board every 40\( ms \).

**aperiodic processes**: A process becomes ready due to some event.

**Example**: What is the frequency of a mouse interrupt?

**sporadic processes**: A process becomes ready due to some event which occurs with an upper bound frequency.

**Example**: The minimal time gap between to subsequent interrupt requests from an Ethernet card is greater than 0.1\( ms \).
Categorization of the execution of processes

The execution of processes is supervised by run-time executives or by operating systems.

- **non-preemptive** execution of processes:
  A process, so it was started, continues to execute until it reaches its final statement.

- **preemptive** execution of processes:
  The execution of processes may be interrupted at any instant of time. Such a process is in a state waiting to resume execution.

  - reason for preemption:
    A process of higher priority becomes ready intermediately

  - technique of preemption:
    A context switch saves the actual state of the interrupted process and allows to resume this process at an appropriate instant of time.

There are completely different theories for scheduling non-preemptive and preemptive processes.
Characteristics of real-time data processing

Data reduction and data expansion

• collect sensor data or external events
  Example: Take a camera image every $40\text{ms}$.

• build an internal image of the real world
  Example: Compute position, velocity, and acceleration of the ball.

• operate on the internal image
  Example: Decide which movements are adequate to keep the ball on the flat board

• sending control data to the actors
  Example: Give the specific instructions for the stepwise movement of the motor to a control latch.
2 Real-time Scheduling

A schedule is a mapping from time to a set \( P = \{P_1, ..., P_n\} \) of processes. Typically real-time is discrete with a granularity of \( \Delta t_G \) per time unit. Hence scheduling is defined as:

\[
s : \mathbb{N} \rightarrow \{P_0, P_1, ..., P_n\}
\]

**Def:** A schedule is **feasible** if all time constraints are satisfied.

**Def:** A scheduling algorithm is **optimal** if it is able to find a feasible schedule for any case that there exists a feasible schedule.

\(^1\) \( P_0 \) is an additional process which runs when no other process in \( \{P_1, ..., P_n\} \) is runnable
Representation of schedules

**explicit**: There exists a complete mapping indicating which process $P_i$ runs on the processor from time $t_1$ to time $t_2$.

In real implementations an (infinite) explicit schedule is represented by one (finite) cycle – a prefix of the schedule – which is executed repeatedly.

**implicit**: A priority is assigned to the processes (fixed priority scheduling) and the processes are executed due to the following rule:

The *ready* process with the highest priority is in the state *running* and all other ready processes are in the state *runnable*.
Basic results in real-time scheduling theory

Essential measure: utilization

\[ U = \sum_{i \in P_{\text{per}}} \frac{\Delta e_i}{\Delta p_i} \]

Classical scheduling algorithms:

**EDF** earliest deadline first: at any instant of time out of the set of ready processes the one with the shortest deadline is executed on the processor.

This algorithm is optimal for a utilization:

\[ U(P_{\text{per}}) \leq U_{\text{lub}}(EDF, n) = 1 \]

**RMS** rate monotonic scheduling (for periodic processes): any process \( P_i \) receives a fixed priority corresponding to the shortness of its period \( \Delta p_i \).

RMS is an optimal scheduling algorithm (Liu, Layland 1983).

There is always a feasible schedule for a utilization:

\[ U(P_{\text{per}}) \leq U_{\text{lub}}(RMS, n) = n(\sqrt[n]{2} - 1) \]

with \( \lim_{n \to \infty} U_{\text{lub}}(RMS, n) = \ln(2) \)
3 The priority inversion problem

Typically parallel processes are not independent and have to synchronize due to shared data. Combining priority based scheduling and synchronization (e.g. accessing a semaphore or critical sections) leads to unbound priority inversion (Lampson, Redell 1980).

"A process $L$ with low priority, which holds a resource that is needed by a process $H$ with high priority, might be preempted by an unbound number of processes with medium priority."
Real-time conditions

(a) Real-time conditions of independent Prozessen:

\[ P[r_i + \Delta e_i + \sum_{j=i+1}^{n} \left( \frac{\Delta p_i}{\Delta p_j} \right) \Delta e_j \leq d_i] = 1 \]

(b) Real-time conditions for processes with common critical sections:

\[ P[r_i + \Delta e_i + \sum_{j=i+1}^{n} \left( \frac{\Delta p_i}{\Delta p_j} \right) \Delta e_j + \sum_{j=1}^{i-1} \Delta s_j \leq d_i] = 1 \]

The term \( \Delta s_i \) indicates for process \( i \) how much time this process maximally spend in one of its critical sections.
Priority inversion schema

Priority inversion the time bound given for processes with common critical sections (b).
Established strategies to cope with priority inversion

Principle idea: A process of higher priority being blocked entering a critical section inherits its priority to the process which is responsible for the blocking. Hence, the progress of the blocking process and implicitly the blocked is improved. There are two families of protocols

- priority inheritance protocol (PIP): Critical sections which are not locked can be entered. A process trying to enter a locked critical section inherits its priority to those processes directly or indirectly blocking this process.

- priority ceiling protocol (PCP): Critical section which is not locked can be entered, only if in the sequel the entering process cannot be blocked by a process of lower priority. The critical sections a process may acquire have to be known in advance.
How the PIP works

Inheriting its priority 3 to process 1 process 3 avoids priority inversion by process 2.
Original definition of the PIP

- The most cited paper (Sha, Rajkumar, Lehoczky, 1990) gives a protocol definition in *natural language*, e.g.:
  
  ... When a job \( J \) exits a critical section, the binary semaphore associated with the critical section will be unlocked, and the highest priority job, if any, blocked by \( J \) will be awakened.

- Protocol properties, e.g. maximum blocking time of processes, are proven formally
- Protocol establishes predictability of execution times
- Most RTOS advertise with priority inversion avoidance
## Practical relevance of the PIP- and PCP-protocols

A lot of operation systems or runtime executives advertise with their implementations of PIP- or PCP-protocols.

<table>
<thead>
<tr>
<th>Operating systems</th>
<th>PIP</th>
<th>PCP</th>
<th>others</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIX</td>
<td>iRMX</td>
<td>ERCOS</td>
<td>pSOS</td>
</tr>
<tr>
<td>OS-9</td>
<td>LynxOS</td>
<td>OS/Open</td>
<td>Nucleus Plus</td>
</tr>
<tr>
<td>RMOS</td>
<td>OS/Open</td>
<td>Virtuoso</td>
<td>:</td>
</tr>
<tr>
<td>WindowsCE</td>
<td>QNX</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td></td>
<td>Solaris</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td></td>
<td>Tornado</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td></td>
<td>VxWorks</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td></td>
<td>VRTX</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>posix</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>1003.4</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>RTOS-UH</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>Pearl</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>Java virtual</td>
<td>RTKernel</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>machine</td>
<td>PERC's Java</td>
<td>Ada 95</td>
<td>C-Executive</td>
</tr>
</tbody>
</table>

---

A closer look at the protocol definition shows:

- Disinheritance rule is incorrectly stated
- Reassignment of a critical section is underspecified

A closer look at some protocol implementations reveals:

- Restrictions on inheritance
- Restrictions on disinheritance

☞ Need for a formal specification
☞ Need for consistent proofs
☞ Check conformance of operating system implementations
Flaw in the original PIP

In (Sha, Rajkumar, Lehoczky 1990) we read:

When a process exits a critical section, it resumes the priority it had at the point of entry into the critical section.
4 Formal model for priority inheritance protocols

Structural idea:

- class diagrams
- architectural model
- functional model
- statecharts
- ease of implementation
- possibility of automated validation
- coherence of protocol verification

Z-specifications
The system class model
Statechart of a user process

- Transition: `sigReady` from `blocked` to `ready`
- Transition: `sigBlock` from `ready` to `blocked`
- Transition: `sigUnblock` from `blocked` to `ready`
- Transition: `sigPreempt` from `running` to `ready`
- Transition: `sigRun` from `ready` to `running`
- Transition: `calculation complete` from `sleeping` to `ready`
- Transition: `period complete` from `sleeping` to `ready`
- Transition: `enter critical section j` from `ready` to `sleeping`
- Transition: `leave critical section j` from `sleeping` to `ready`
Statechart of a priority inheritance scheduler

- **Idle State**
  - Transition to `sigEnterCS (p, c)` with condition `[isEmpty(c)]`
  - Transition to `enterCS (p, c)` with condition `[isNotEmpty(c)]`

- **Block On CS (p, c)**
  - Transition to `blockOnCS (p, c)`

- **Enter CS (p, c)**
  - Transition to `n = next ()`

- **Sig Complete (p)**
  - Transition to `leaveCS (p, c)` with condition `[someProcessBlocked(c)]`
  - Transition to `makeReady ()`

- **Leave CS (p, c)**
  - Transition to `sigLeaveCS (p, c)`

- **Timer Tick**
  - Transition to `r = runningP ()`

- **Preempt**
  - Transition to `^r.preempt`

- **Make Ready**
  - Transition to `^p.sigBlock`
  - Transition to `^h.sigUnblock`

- **Wake Up**
  - Transition to `h = wakeUp (c)` with condition `[noProcessBlocked(c)]`

- **Process Ready**
  - Transition to `^r.preempt`

- **Sig Run**
  - Transition to `^n.sigRun`
5 Conformance testing

Current state of our project:

- (Semi-)Automatically performed test case generation

- Manually performed operating system mapping
  - QNX Software Systems: QNX Neutrino / QNX Real-Time Platform
  - Sun Microsystems: Solaris 8 (Intel)
Checking the scheduling algorithm

The test suite assumes, that processes are scheduled by priority.

QNX:

Solaris:

```c
void desc_p1() {
    tgPriority (1);
    tgReadyAt (1);
    tgCalculate (6);
}
```
Priority Inheritance

Basic priority inheritance: P1 inherits priority of P3

**QNX:**

P3:

P2:

P1:

**Solaris:**

P3:

P2:

P1:
Conformance Testing of Priority Inheritance Protocols

Retrieving the inherited priority

actprio = 8

print actprio

D. Zöbel  Working Group Real–Time Systems

26
Transitive blocking (1)

QNX:

Solaris:
Conformance Testing of Priority Inheritance Protocols

Transitive blocking (2)

QNX:

Solaris:
Leaving a critical section which blocks another process

QNX:

Solaris:
6 Conclusion

Status quo:

- Even actual textbooks copy the original flaws of the protocol
- Practical implementations in real-time operating systems have defects
- Application programmer are in doubt about the calculation of worst case execution times based

Results achieved so far:

- Formal definition of the aimed results: time constraints
- Structured overall design
- Solid basis for the correct implementation of protocols
- Mathematically consistency and correctness
- Usable as a basis for generating test cases