Design of Embedded Systems (DES)

Jozef Hooman

http://www.cs.ru.nl/~hooman/DES/

Course 3
17 September 2014

---

Xenomai exercise #3

rt_sem_create has parameter “mode”:
- S_FIFO: pending tasks in FIFO order
- S_PRIO: pending tasks in priority order

rt_sem_broadcast unblocks all tasks waiting on semaphore

rt_sem_p may have time-out value
- after time-out no guarantee about mutual exclusion,
  typically only perform error/recovery actions

---

Tasks with shared resources

- Drop assumption that tasks are independent
- Assume may share resources, typically access is guarded by semaphores
- Hence tasks may block each other, which may lead to
  - deadlock
  - priority inversion

---

Sleep in RT tasks of ex01

```c
void demo(void *arg)
{
    RT_TASK *ctask;
    RT_TASK_INFO ctaskinfo;
    // hello world
    rt_printf("Hello World\n");
    // inquire current task
    ctask=rt_task_self();
    rt_task_inquire(ctask,&ctaskinfo);
    sleep();
    // print task name
    rt_printf("Task Name : %s \n", ctaskinfo.name);
}
rt_printf("end program by CTRL-D\n");
```

See also: http://www.cs.ru.nl/lab/xenomai/tips.html

---

Xenomai exercise #3

```c
rt_sem_v ( RT_SEM * sem )
```

Release a semaphore unit.
If the semaphore is pended, the first waiting task (by queuing order) is immediately unblocked; otherwise, the semaphore value is incremented by one.

---

Priority inversion

Suppose
- low-priority task locks resource;
- it is preempted by long running middle-priority task
  - blocks high-priority task that needs resource

Famous example: Mars Pathfinder mission (1997):
- low-level: gather meteorological data, needs resource
- middle-level task
- high-level: bus management, also needs resource

See: “What really happened on Mars?”
Global architecture on Mars

Global architecture:
- CPU connected by VME bus to radio, camera and interface to 1553 bus
- 1553 bus connects
  - “cruise stage” part (controls thrusters, valves, sun sensor, star scanner) and
  - “lander” part (interface to accelerometers, radar altimeter, and ASI/MET for meteorological science)
- HW interface to 1553 inherited from Cassini;
  software should schedule activity at 8 Hz
RTOS: vxWorks

Tasks on Mars

Tasks (in order of priority, highest prio first):
- \textit{bc\_sched}: set up transactions on 1553 bus
- task controlling entry and landing
- \textit{bc\_dist}: collection and distribution of bus data
- tasks performing spacecraft functions
- science functions (imaging, ASI/MET, …)

\textit{bc\_sched} and \textit{bc\_dist} check each cycle whether other has completed its execution

Task timing on Mars

\texttt{<------------------ .125 seconds ------------------------- ->|}
\texttt{|< - bus ->|}
\texttt{|< - \textit{bc\_dist} ->| \textit{bc\_sched} |<->|}
\texttt{|<-------------------|********|*******|-------------------|}
\texttt{t1 t2 t3 t4 t5 t1+.125}

For most applications, \textit{bc\_dist} delivers data via double buffered shared memory mechanism
Exception: data to ASI/MET delivered via pipe mechanism
Tasks use \textit{select} mechanism to wait for message arrival

Failure on Mars

\textit{bc\_dist} did not complete before start of \textit{bc\_sched}

\begin{itemize}
  \item reset, reinitialize all HW & SW, continue next day
\end{itemize}

Problem:
- ASI/MET calls \textit{select} uses mutex semaphore
call pipeioctl(), which calls selNodeAdd(), which starts to release semaphore
- preemption of ASI/MET by several medium tasks before release of semaphore completed
- \textit{bc\_dist} executes, try to send data to ASI/MET, but blocks on pipeWrite() because of semaphore
- medium tasks continue, until \textit{bc\_sched} task executes and detects problem

Bebugging on Mars

- Debugging features, such as tracing and logging, were used in lab but remained in software by design: “test what you fly and fly what you test”
  - \textit{bc\_sched} already coded such that it could stop trace/log and dump the data
  - Tracing and logging yields more data than can be send back to Earth
    \begin{itemize}
      \item not usable on flight spacecraft
    \end{itemize}
  - But: testing in lab possible without code changes
    In < 18 hours problem could be repeated in the lab

Solution to priority inversion

Priority inheritance:

- A task that holds a resource via a semaphore executes at the priority of highest-priority task that is blocked on that resource
- Included in many RT-OSs as type of semaphore
Correction on Mars

Solution:
- change creation flag in vxWorks for semaphores to enable priority inheritance

Not so obvious that this is OK:
- in vxWorks the change would affect all other semaphores
- how would such a global change affect behaviour of rest of system?
- priority inversion was left out for optimal performance, how will performance degrade?
- would the behaviour of the select mechanism change?

Why not detected before launch?

- They did not expect nor test the "better than we could have ever imagined" case [no stress testing]
  - ASI/MET data rates were higher than estimated and
  - amount of science activities proportionally greater
- Problem was observed before launch, but could not be repeated
  - heavily concentrating on entry and landing, so data collection had lower priority
  - no time to complete lower priority issues
- system was robust and could be reset
  (SW could recover from radiation induced errors in memory or CPU, continue when radio fails, etc.)

Priority inversion

- With priority inversion deadlock still possible:
  \( \tau_1 \) gets R1, \( \tau_2 \) gets R2; next
  \( \tau_1 \) needs R2 and \( \tau_2 \) needs R1

Avoiding priority inversion and deadlock

Priority ceiling (for dynamic priorities):
- For every shared resource, compute ceiling:
  priority of highest priority task that may lock resource

At runtime:
- Task may lock resource only if task priority > ceiling of all currently locked resources
- Task that holds lock inherits highest priority of blocked tasks on that lock until release of lock
  [priority inheritance]

Example Priority Ceiling (1)

Consider 5 tasks and two resources A and B
- Task 1 needs A and B (in this order), prio 6
- Task 2 needs B and A (in this order), prio 5
- Task 3, executes long, priority 3
- Task 4 needs A, priority 2
- Task 5 needs B, priority 1
  (higher number = higher priority)

Example Priority Ceiling (2)

- 1 needs A and B, prio 6
- 2 needs B and A, prio 5
- 3 executes long, prio 3
- 4 needs A, prio 2
- 5 needs B, prio 1
Without any inheritance/ceiling protocol:
- Task 5 arrives, gets B
- Task 4 arrives, preempts 5 and gets A
- Task 2 arrives and gets blocked
- Task 3 arrives, preempts 4, executes long, terminates
- Task 4 resumes, releases resource A
- Task 5 resumes, releases resource B
- Tasks 2 resumes, gets B
- Task 1 arrives, preempts 2 and gets A, but later blocks on B \( \Rightarrow \) deadlock
Example Priority Ceiling (3)

- Task 1 needs A and B, priority 6
- Task 2 needs B and A, priority 5
- Task 3 executes long, priority 3
- Task 4 needs A, priority 2
- Task 5 needs B, priority 1

With priority ceiling:
- A has ceiling 6, B has ceiling 6
- Task 5 gets B
- Task 4 blocks (priority A=2 < ceiling B=6)
- Task 2 arrives, gets blocked, task 5 gets priority 5
- Task 3 arrives, gets blocked
- Task 5 completes and releases B
- Task 2 gets B
- Task 1 arrives, gets blocked when it needs A because its priority is not greater than all locked resources (B has also priority 6)
- Task 2 gets A, completes, and releases A and B
- Task 1 executes and completes
- Task 3 executes long and completes
- Task 4 executes and completes

Comparison

Without priority ceiling:
- Task 5 arrives, gets B
- Task 4 arrives, preempts 5 and gets A
- Task 2 arrives, gets blocked
- Task 3 arrives, preempts 4, executes long, completes
- Task 4 completes, releases A
- Task 5 completes, releases B
- Task 2 resumes, gets B
- Task 1 arrives, preempts 2 and gets A, but later blocks on B due to priority inversion
- Task 2 gets A, completes, and releases A and B
- Task 1 executes and completes
- Task 3 executes long and completes
- Task 4 executes and completes

With priority ceiling:
- Task 5 arrives, gets B
- Task 4 arrives, blocks (priority A=2 < ceiling B=6)
- Task 2 arrives, gets blocked, task 5 gets priority 5
- Task 3 arrives, gets blocked
- Task 5 completes and releases B
- Task 2 gets B
- Task 1 arrives, gets blocked when it needs A because its priority is not greater than all locked resources (B has also priority 6)
- Task 2 gets A, completes, releases A and B
- Task 1 executes and completes
- Task 3 executes long and completes
- Task 4 executes and completes

Avoiding priority inversion and deadlock

There are many more approaches to avoid priority inversion and deadlock. E.g.:
- Combinations with conventional deadlock prevention algorithms
- Dynamic priority ceiling for a combination with EDF
- ... Most RTOSs support priority inheritance

Highest locker versus priority ceiling

- For every shared resource, compute ceiling: priority of highest priority task that may lock resource
- At runtime:
  - Task may lock resource only if task priority > ceiling of all currently locked resources
  - Task that holds resource gets ceiling priority + 1
  - Task that holds lock inherits highest priority of blocked tasks on that lock until release of lock (priority inheritance)

Shared resources

RMA extended

- Suppose tasks may block due to resource access
- Let r1, ..., rn be a list of tasks in the order of priority (highest priority first), with completion times Ci and periods Ti
- Let Bi be the longest blocking time (time it has to wait on a shared resource) of ri in its period

Then the requirement is, for each i, 1 ≤ i ≤ n,

\[ \sum_{j=1}^{i} \left( \frac{C_j}{T_j} \right) + B_i/T_i \leq \left( 2^{1/i} - 1 \right) \]

Again, if extended RMA test fails, system might still be schedulable

RMA with shared resources

<table>
<thead>
<tr>
<th>Task</th>
<th>Priority</th>
<th>Completion time</th>
<th>Period</th>
<th>Resource usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td>3</td>
<td>C1 = 20</td>
<td>T1 = 100</td>
<td>15</td>
</tr>
<tr>
<td>t2</td>
<td>2</td>
<td>C2 = 30</td>
<td>T2 = 150</td>
<td>10</td>
</tr>
<tr>
<td>t3</td>
<td>1</td>
<td>C3 = 50</td>
<td>T3 = 300</td>
<td>18</td>
</tr>
</tbody>
</table>

Assuming priority inheritance:
- For t1: 20/100 + 18/100 = 0.38 < 1
- For t2: 20/100 + 30/150 + 18/150 = 0.52 < 0.8284
- For t3: 20/100 + 30/150 + 50/300 = 0.5667 < 0.7798
So schedulable
Xenomai exercises #6, #7, #8

- #6: change priorities during execution
  [modify existing program]
- #7: round robin scheduling for tasks
  with the same priority
  [modify existing program]
- #8: a: create program with priority inversion
  b: eliminate using mutex
  [mutex = semaphore with priority inheritance]

Assignment for 24 September 2014

- Xenomai exercises #6, #7, and #8
  (we skip #4 and #5)

Mail me before 23 September 18:00:
- Source files + Makefile + short explanation