CosmoS

Cosmos-B

RESOURCE SCHEDULING

*Cosmos-B is a managed code operation system designed to run on the Cosmos kit. This document describes the Scheduling system. .*

RESOURCE SCHEDULING

Ben KloosterMan

# 0.OVERVIEW

The most frustrating thing in modern OS is the fact that a task that users wishes to run is performing poorly due to a non CPU resource being swamped such as memory , network or disk. This can easily be fixed by using a more holistic approach to scheduling to include non CPU Resources.

To this end the system implements a global scheduling service.

# 1.CPU SchEduling

System uses a modified Rialto scheduler ( MSR) designed for more Asynchronous applications.

For non real time requirements and lots of Asynchronous STPs we need a scheduler

That

* Prioritizes short running tasks
* Is efficient when doing a task switch and can handle large amounts of waiting or idle threads.
* Tasks waiting on this need to be considered for the result

The Rialto scheduler pre computes slots based on previous performance of calls giving an estimate how long it will take ( best , average and worst) thus giving it real time scheduling capabilities without the overhead of a code path calculation. These statistics can be persisted and have the additional advantage of allowing us to include resource times ( such as HD) in the real time calculations. While not Hard real time it is likely to be more accurate than hard real time systems that don’t include the Hard disk and other resources.

For more fine grained real time we will need either pauseless collectors or add on a potential worst case based on Heap size. (I did say course grained ☺ )

A Rialto scheduler also modifies any scheduling requests to indicate whether it is critical real time or whether a miss is undesirable but not critical such as multi media

We implement no priority levels; priority is based on prioritizing short running tasks Divided by the number of tasks waiting.

Cosmos –B also includes a round robin scheduler for ease of implementation and comparison.

# 1.RESOURCE SchEduling

The system will use Resource Containers see “Resource Containers: A New Facility

for Resource Management in Server Systems” .Resource will use minimum-laxity-first scheduler as it is meaningful to all resources.

Note the interrelation of scheduling eg is a CPU task is waiting on disk scheduling ut

# ISSUES

One issue with a Rialto scheduler is that if a thread is waiting for a non-CPU resource that it can miss its pre-computed CPU scheduling slots. We address this by the fact that most non CPU resources are slow to change and the modified Rialto scheduler can peak at the appropriate resource scheduler to adjust the position, this is greatly assisted by the fact that our STPs have narrow roles and are nearly always waiting for something and hence a dependency graph is easy enough to produce. Also note these STPs due to the narrow role can have strict limits eg keyboard response on screen 30ms.

None Cosmos B runs a lot of Context switches and hence the context switch time needs to be low which it is but we don’t want to add too much scheduler time in their.

# 1.STRUCTURE

CPU efficient scheduler is pointless especially in modern multi core environments it is more important to schedule correctly especially with Multi media and real time requests.

However it is important when a Task switch is required and that the task is waiting on a quick task that it will be cheap.

To achieve this we separate scheduling with a scheduler service. The service determines the next process to run and the actually Task switch which will be a cheap operation that uses this. The Scheduler service runs in its own thread.

# 2. SCHEDULER SERVICE

The scheduler determines the place of all location in a run queue. It is invoked when

* a real-time event past time occurs
* when a process yields or exits
* when a preemption interrupt occurs and there is nothing to schedule

The scheduler service will create a scheduler queue for each CPU it will also mark future pre-emption interrupts to fire events for tasks that are getting urgent or are behind schedule..

All calls relating to scheduling go via the scheduler service.

We use the term task switch to distinguish from the scheduler service the task switch runs simplified structure. Note the task switch calls scheduler service code but not in the context of the scheduler.

# 3.Pre EMPTION TASK SWITCH (Clock Interrupt)

When a pre-emption interrupt occurs the system will suspend the current thread, check for any events to fire (including no task to run) and run the task at the head of the run queue. If there is no task it will idle. Note we only pre-empt on IO interrupts if the scheduling table has changed since the last item was scheduled or if a real time item is on the head of the queue.

# 4. YIELD TASK SWITCH

When a task yields it will run the next program in the task list however there are 2 exceptions.

If the task yielded and still has more than 1/8th of its quanta available

- it will be placed at the head of the queue but behind any tasks of < 1 priority.

- If the task yielded by sending a message the destination is checked and if the destination was waiting on that message it will be immediately scheduled unless there are tasks of higher priority than both the caller and receiver at the head of the queue in that case the receiver is promoted to the highest point at the higher of the caller or receivers privilege.

# MODIFICATION TO REALTO

The key modifications (mainly to do with non real time)

* Time based on previous calls ( Sorted by operation and service)
* A Fair Scheduler approach for non real time e.g. the first to finish is the priority ( based on when the request was made not current)
* Scheduling inheritance where if a thread is waiting on something than all the urgency of the caller is passed to the destination.
* The “priority” is multiplied divided by the number of callers waiting so a 1000 priority would become 200 if 5 processes were waiting on the response.
* The “priority” is doubled when the processor affinity is not set.
* Yield task switch changes
* A task won’t be scheduled unless the resource it is waiting on becomes available.

# 2. DETECTING FAILED PROCESSES

Detecting failed STPs ( ie in a tight loop) is the job of the re-birthing service…

# 4. OTHERS OPTIONS

**Other schedulers**

Unmodified Rialto and BVT (<http://www.cs.ucla.edu/~kohler/class/04f-aos/ref/duda99borrowed.pdf>) while providing good real time performance don’t work well in a completely Async environment where it is better to run tasks others are waiting on even if it is not urgent eg for a disk interrupt you want to schedule a DMA transfer.

**Singularity style resources scheduler**

This was tempting especially isolating the parts but we will try an “ all knowing or at least semi all knowing” scheduler first.

**Exclude Realtime**

Why real time? Mainly multimedia support and its reasonable easy to start with **course time** real time but very difficult to introduce later. We especially want to take this on being a GC OS it will be a presumed issue. In addition we want to get to the point where applications describe what they desire and the OS provides it which may require Real time capable GC scheduling etc.