Constraints at Fitbit

- Memory
- Power
- Time critical tasks - sensor acquisition, algorithms
- Many user facing tasks that need to share the cpu fairly
Tick or tickless?
What is a tick?

• A periodic timer → wakes the system to check for pending events.

• The timer ISR has two jobs:
  • Handle expired timers.
  • Handle the scheduling policy.

• Timer period is a trade-off between precision and power efficiency.
Introducing tickless

• Replace periodic timer ➔ one-shot timer dynamically set to fire on the next event.

• Idle system ➔ timer doesn’t need to trigger at all.

• System can respond to events rapidly when needed and sleep in idle periods.
The ticks in tickless

- Tasks running at 20ms interval

Align expiration with ticks

Allow random expiration

Tick

Running

Idle

Time (ms)

10 20 30 40 50 60
Scheduling Policy
NuttX Scheduling Policies

• Preemptive RTOS ➔ priority is strictly enforced.

• Tasks with equal priority execute FIFO.

• Round robin policy available.
Round Robin Policy

• Each task is assigned a RR_INTERVAL timeslice.

• When the time slice elapses, swap the task with the next task of equal priority.
Round Robin in Action

Tick | 1 | 2 | 3 | 4
--- | --- | --- | --- | ---
T2 | | | | 
T1 | | | | 

RR_INTERVAL = 1
What if we get preempted?

Tick 1 2 3 4

IRQ

T3

T2

T1

P = 50
P = 100
interrupt

Pre-emption

RR_INTERVAL = 1
Limitations

• Tasks are guaranteed to execute at least RR_INTERVAL, but...

• Tasks waiting in line can potentially wait indefinitely to be scheduled.
Can we do better?

- Naive solution ➔ swap task when it gets preempted.

- Unfair to the interrupted task ➔ it doesn’t get to complete its time slice.

- Better solution ➔ carry remaining slice when task resumes.
Can we do better?

• Only restore time slice on:
  • up_blocktask.
  • round-robin swap.

• On task suspend:
  • Subtract elapsed ticks from timeslice.
  • If preempted - save system time in tcb.

• On task resume - check if same tick:
  • If not during preemption tick - decrement timeslice.
  • Execute round-robin swap if timeslice is depleted.
Preemption Behaviour

- **Tick**
  - 1
  - 2
  - 3
  - 4

- **IRQ**
  - P = 50

- **T3**
  - P = 100

- **T2**

- **T1**

- **Interrupt**

- **Pre-emption**

- **RR_INTERVAL = 1**
A small compromise

• Solution requires updating all calls to sched_resume_scheduler in architecture code.

• Move round robin swap from task resume to task suspend.
Pros

• Tasks are guaranteed to execute at least RR_INTERVAL.

• The task that was preempted keeps its time slice if it is able to resume during the same tick.

• The task waiting next in line has to wait no more than RR_INTERVAL + 1 tick until it gets scheduled.
Round Robin and Tickless

• NuttX doesn’t re-evaluate timer on context switch.

• Maximum timer period must always be RR_INTERVAL.

• RR_INTERVAL becomes a faux tick.

• RR preemption handling not possible.
Solution

- Drop the upper bound of the timer interval.
- Reassess alarm after context switches.
Obstacles

• Reassess triggers wdog timer processing.

• Ticks elapsed when setting up the context switch ➔ reassess updates the wrong task.
Pros

• CPU can sleep indefinitely when idle.

• Tasks that do not use RR policy not interrupted every RR_INTERVAL.

• Dynamic round robin time slicing.
Cons

• Reassessing the timer on context switch is a heavy operation.

• Implementation needs SCHED_TICKLESS_ALARM.
Future Improvements

• Move RR slicing to dedicate HW timer to mitigate the penalty of reassessing.

• Start RR slicing only when equal priority tasks are unblocked.
How do we measure scheduler performance?

• Power consumption ➔ -0.45% active duty cycle.

• Round robin swap rate ➔ +95% swap rate.

• UI responsiveness ➔ noticeable improvement.
Conclusion

• Preemptive strict priority scheduling meets real time processing requirements.

• Round robin policy increases fairness of same priority task scheduling.

• Tickless scheduling allows us to meet power requirements.
THANK YOU