NuttX RTOS Beginnings

Gregory Nutt
About Me

Math/Physics
Math

Digital Signal Processing

Neuro-science

Computer Science

1970

Academia

1980

Military Avionics

Cooperative Schedulers

1990

LaserJet POSIX RTOS, 1996

Hewlett Packard

Cooperative Schedulers

Commercial Electronics

2000

LaserJet Firmware Architect

About Me

Ridgerun.com

NuttX Release to Open Source, 2007

Costa Rica

NX-Engineering, SA
Hardware of First RTOS

Ruggedized MIP-16 with Optional Control Panel
**Interrupt Driven – OS #1 (Bare Metal)**

- **Single Interrupt Handler**
- **High Priority**
- **Medium Priority**
- **Low Priority**

**Response Latency**

**No OS**: Extensive interrupt processing, prioritized interrupts and, maybe, a *main loop.*

**Deterministic? No**  **Meet Deadlines? Maybe**

Problems: Stacked, Can lose interrupts.
No waiting, all run to completion.
Main Loop – OS #1 (Cont'd)

- Task A
- Task B
- Task C
- Task D
- Task E
- Task F
- Task G

Main, Background Loop

Non-deterministic!

- Round-Robin
- Non-preemptive
- Cooperative Scheduling
- State machines
- Ad hoc strategies

Non-deterministic!
Real Time == Deterministic

Real time does not mean "fast"

Real time systems have **Deadlines**

- Stimulus
- Response Latency
- Response
- Deadline
Main Loop with Priority Queue – OS #2

Still Non-deterministic!

High priority work still has to wait for work in progress.

Interrupt

Brief interrupt processing, only queues work

Lowest Prio

Highest Prio

Main, Background Loop

Task X
Main Loop with Cooperative Scheduler– OS #3

Task X

```c
switch (state) {
    case state A:
        Start event processing;
        state = state B;
        Reschedule;
        Break;
        
        Case state B:
        Continue event processing;
        State = state C;
        Reschedule;
        Break;
        
        Case state C:
        Finish event processing;
        State = state X;
        Break;
        
        Case state X:
        Break;
}
```

- Non-preemptive
- Cooperative Scheduling

- Divide event processing up into pieces
- Manage with a state machine
- Reschedule to allow higher priority tasks
- Other *ad hoc* strategies

Still Non-deterministic!

*High priority work still has to wait for work in progress.*
Foreground / Background Main Loops – OS #4

Partially Deterministic

Context Switch!

Foreground Main Loop

Background Main Loop

Partially Deterministic
Pre-emptive OS – OS #5

The DEC connection

Fully pre-emptible
Context switch:
Think setjmp/longjmp on steroids

Task Control Block (TCB)
States represented by lists of TCBs

Task Start

Pending (NuttX)

Ready-to-run

Blocked

Running

Highest Priority
Ready-to-run task is Running

Wait for signal, semaphore, message queue, page fill, stopped, etc.
RTOS Interrupt Processing

**Stimulus**

- **Interrupt Handler**
- Signals thread via IPC

**RTOS Scheduler**

- Reassess next ready-to-run thread
- Resumes thread if highest priority, ready-to-run

**Task**

- Suspended, Waiting for event
- Awakened, Processes interrupt related event
- Suspended, Waiting for Next event

**Response**
RTOS way:
- Minimal work performed in interrupt handlers
- Interrupt handlers only signal events to tasks
- RTOS scheduler manages real-time behavior
- Prioritized interrupts replaced with prioritized tasks
- No benefit in nesting interrupts (usually)

No OS way: Extensive interrupt processing, prioritized interrupts and, maybe, a main loop.
SMP

- Pending
- Blocked
- Ready-to-run
- Running

Assigned Task List (not shown)

Affinity

n Highest Priority
Ready-to-run tasks are Running

Spinlocks

CPU 0
CPU 1
CPU 2
CPU n
Rate Monotonic Scheduling

Can achieve Real-Time behavior under certain circumstances

- Strict priority scheduling
- Static priorities
- Priorities assigned according to Rate Monotonic conventions

Threads with shorter periods/deadlines are assigned the highest priorities.

And this *unrealistic* assumption:
- No resource sharing
- No waiting for resources
- No semaphores or locks
- No critical sections
- No disabling pre-emption
- No disabling interrupts
Why POSIX?

Why not...
- Versus custom *ad hoc* OS interface
- POSIX device model vs HAL
- Like simpler FreeRTOS, ChibiOS, Zephyr, mbed, RIOT, etc.

At this point POSIX is the NuttX identity
- Portability
- Linux compatibility
- Complex build models: PROTECTED and KERNEL builds
Work Queues

Priority Queue
- Non-preemptive
- Very high priority
- Inappropriate for extended processing

Use with care!

Non-deterministic!
Multiple Work Queues

- Single high priority work queue
- Intended for interrupt “bottom half”
- Should be highest priority

- Multiple low priority work queues
- Support priority inheritance
- Use to implement asynchronous I/O (AIO)

Thread pool