

Harmonized Scheduling + Performance Monitoring & Optimization

Raj Rajkumar
Lecture #12

Administrivia

- Quiz #3 on Wednesday.

Outline

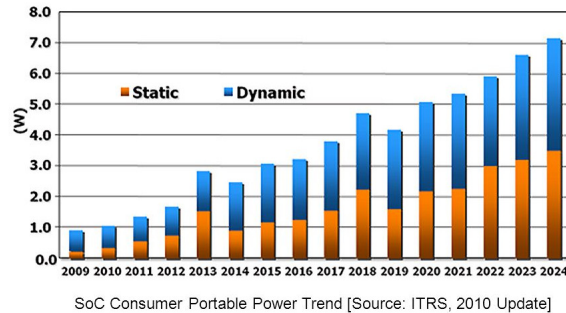
- Power and time constraints of microcontrollers
- Rate-Harmonized Scheduling
- Energy-Saving Rate-Harmonized Scheduling+
- Energy-Saving Rate-Monotonic Scheduling

CMOS Power Dissipation

- $P_{total} = P_{dynamic} + P_{static}$
- **Dynamic Switching Power**
 - $P_{dynamic} = K * C_L * V_{dd}^2 * f$
 - due to charging and discharging of output capacitances
 - reduced using Dynamic Voltage and Frequency Scaling (DVFS)
- **Static Leakage Power**
 - $P_{static} = V_{dd} * I_{leakage}$
 - reduced using low-power sleep states
 - power gating and/or clock gating

Slow Down or Sleep ?

- At fabrication technologies smaller than 65 nm, static leakage power dominates the total power consumption of CMOS-based VLSI circuits.



Slow Down or Sleep ?

- Many low-power microcontrollers do *not* support DVFS but support multiple sleep states.



CPU Power Consumption Parameters

Processor	Frequency (MHz)	Active Power (mW)	Idle Power (mW)	Sleep Power (μ W)	Sleep to Idle (ms)	Idle to Active (μ s)
Atmega 1281 <i>AVR</i>	8	23	6.6	16	12	6
NXP K22 <i>ARM Cortex-M4</i>	120	81.09	31.89	54.3	0.140	5.7

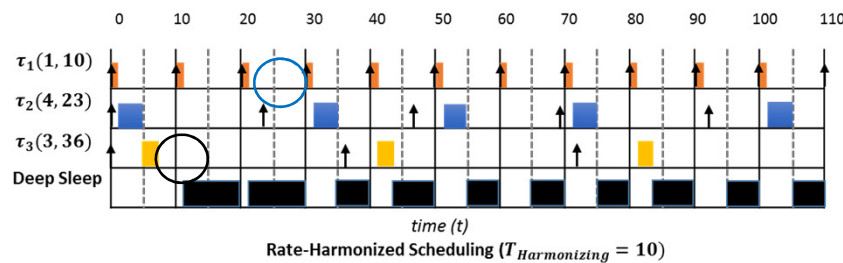
Power Modes of Microcontrollers

Power state	Power (mW)	Upward Transition Time
Active	30 mW	n/a
Idle	6 mW	6 μ s
Sleep	5 μ W	10 ms

- Power Management: maximize the Sleep-time of processors
 - given {Sleep, Idle, Active} modes of operation

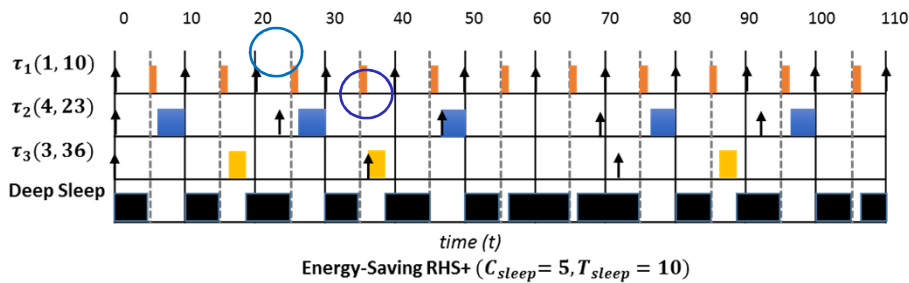
Rate-Harmonized Scheduling (RHS)*

- Pick a **harmonizing period**
 - \leq the **shortest period** in task set
 - same phasing as the **highest priority** task
- Tasks when released become eligible to execute
 - at the **next** harmonizing period boundary OR
 - when **the processor is busy executing a task instance**



Energy-Saving Rate-Harmonized Scheduling + (ES-RHS +)

- Create an “Energy Saver” (forced sleep) task with
- **Period** = $T_{\text{sleep}} \leq$ harmonizing period \rightarrow **highest scheduling priority**
- **Budget** = $C_{\text{sleep}} \geq \min-C_{\text{sleep}}$



ES-RHS+ Properties

- **Optimality**

- Clusters all **idle** durations for a schedulable taskset
- → Processor is always either fully asleep or doing useful work

- **Enhances** schedulability

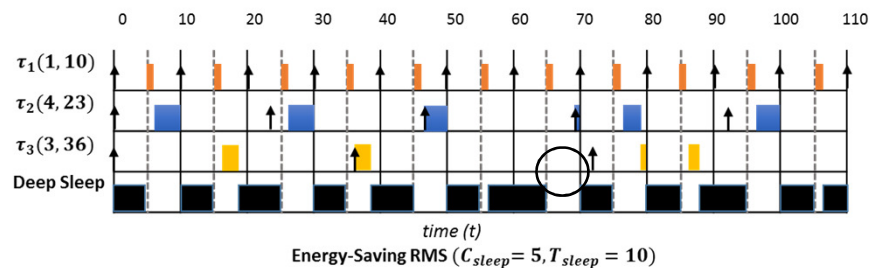
- Response Time recurrence relation for task τ_i in ES-RHS+
 - $W_0 = C_i + T_{sleep} - C_{sleep}$
 - $W_{k+1} = C_i + T_{sleep} - C_{sleep} + \left\lceil \frac{W_k}{T_{sleep}} \right\rceil C_{sleep} + \sum_{j=1}^{i-1} \left\lceil \frac{W_k}{T_j} \right\rceil C_j$

Energy-Saving Rate-Monotonic Scheduling (ES-RMS)

- Extension to RMS

- **Create an Energy Saver task with**

- **period** = $T_{sleep} \leq T_1 \rightarrow$ *highest system priority*
- **budget** = $C_{sleep} \geq C_{sleepMin}$



ES-RMS Properties

- ES-RMS has **better schedulability** than ES-RHS+
 - Response Time recurrence relation for task τ_i in **ES-RMS**
 - $W_0 = C_i$
 - $W_{k+1} = C_i + \left\lceil \frac{W_k}{T_{sleep}} \right\rceil C_{sleep} + \sum_{j=1}^{i-1} \left\lceil \frac{W_k}{T_j} \right\rceil C_j$
 - Response Time recurrence relation for task τ_i in **ES-RHS+**
 - $W_0 = C_i + T_{sleep} - C_{sleep}$
 - $W_{k+1} = C_i + T_{sleep} - C_{sleep} + \left\lceil \frac{W_k}{T_{sleep}} \right\rceil C_{sleep} + \sum_{j=1}^{i-1} \left\lceil \frac{W_k}{T_j} \right\rceil C_j$
- Worst-case blocking **eliminated**
 - Better schedulability
 - Lesser **total deep sleep**, but greater **forced sleep utilization**

Comparative Evaluation

- ES-RHS+ better than RMS
 - up to 33% better energy-savings on some sensor nodes
- In most cases, ES-RMS yields **deep sleep utilization** very **close** to the optimal (on average 1.9% - 6.7% difference)
 - practical for use in many operating systems
- ES-RMS can provide **greater forced sleep utilization**
 - Up to **18%** greater forced sleep duration than ES-RHS+
 - Useful for energy-savings on multi-core scheduling
- Estimated Energy Savings
 - NXP K22 → ES-RHS+: 40.57 mW, ES-RMS: 41.65 mW → (2.59%)
 - AT1281 → ES-RHS+: 11.50 mW, ES-RMS: 11.73 mW → (1.96%)

RHS Conclusions

- Power and time constraints of Microcontrollers
- Rate-Harmonized Scheduling
- Energy-Saving Rate-Harmonized Scheduling+
- Energy-Saving Rate-Monotonic Scheduling

PERFORMANCE OPTIMIZATION & TOOLS

Tools

- Optimization in Traditional Environments
- Monitoring and Optimization Requirements in Real-Time Environments
- Performance Monitoring Tools
 - ARM (Advanced Real-time Monitor)
 - WindView from Wind River
 - TimeTrace from TimeSys
 - Linux Trace Toolkit

Traditional Monitoring and Optimization Tools

- Symbolic debuggers
 - Single-step through logic
 - Modify registers, memory and variables
 - Set breakpoints
- Lots of **printf** statements
- Optimizing compilers
 - Intra-procedural optimization
 - Optimize code within a procedure
 - Inter-procedural optimization
 - Optimize code across procedures
 - Optimizing compilers....
 - `gcc -O n` // as n gets larger, more optimizations applied


Intra-Procedural Optimization

- Unwinding of loops

```

for (i = 0; i < 4; i++) {
    x += 4y;
}


```



```

x += 4y;
x += 4y;
x += 4y;
x += 4y;

```



```

z = 4y;
x += z;
x += z;
x += z;
x += z;

```

- Dead-code elimination

```

x = 2;
if (x > 3) {
    ...
}

```



```

x = 2;


```

- Smart arithmetic

```

x = y*17;


```



```

x = (y << 4) + y;

```



```

x = (y << 5) - (y << 2);

```

```

x = y*28;

```

Inter-Procedural Optimization

```

integer b;          // variable "global" to the procedure Silly.

int Silly(int a, int x) {
    if (x < 0) a = x + b;
    else a = -6;
}

main()
{
    int a, x;        // These variables are visible to Silly only as params.

    x = 7; b = 5;
    Silly(a, x); printf("%d", x);
    Silly(x, a); printf("%d", x);
    Silly(b, b); printf("%d", b);
}

```

If arguments passed by value, no effect!

If arguments are passed by reference,

```
x = 7; b = 5;
if (x < 0) a = x + b; else a = -6; print x;    // a is changed.
if (a < 0) x = a + b; else x = -6; print x;    // params are swapped
if (b < 0) then b = b + b else b = -6; print b;
```

- The compiler can follow the constants along the logic and find that the predicates of the *if* statements are constant and so...

```
x = 7; b = 5;
a = -6; printf("7");           // b is not referenced
x = -1; printf("-1");          // b is referenced...
b = -6; printf("-6");          // b is modified as parameter
```

- Since *a*, *b* and *x* deliver nothing to the outside world, there is no point in this code either, and so the result is

```
print 7;
print -1;
print -6;
```

Real-Time Systems

- Many traditional techniques can be applied...
... carefully

- Must be very careful about side effects

```
foo() {
    ...
    x = 5; // may seem like dead code to a traditional
           compiler
           // could actually be writing to an I/O device
}
```

- Such variables are usually tagged with the keyword
“**volatile**”

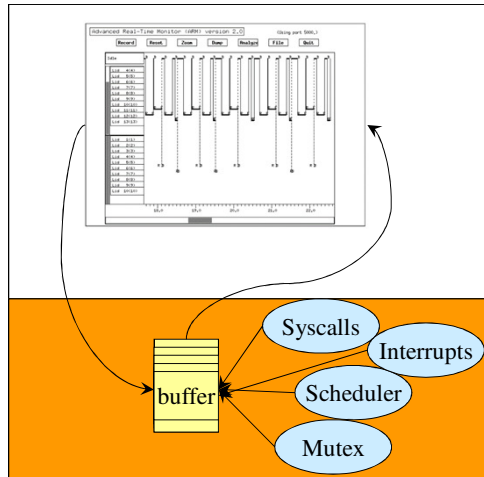
Other Requirements

- Inserting breakpoints that stop program will change timing behavior
- Adding `printf` statements can change timing behavior
- Recursive calls can necessitate big stacks consuming potentially valuable memory.
 - Want to perform memory vs. CPU tradeoffs
- Want to know what is happening when
 - What states tasks are in at different times
 - Which task is communicating with whom
 - What interrupts are happening and when
 - What the execution times are for different tasks
 - What the periods of different tasks are
 - What the budget is for reservations (or aperiodic servers)
 - What the system overheads are
 - Study input jitter and output jitter

Advanced Real-Time Monitor (ARM)

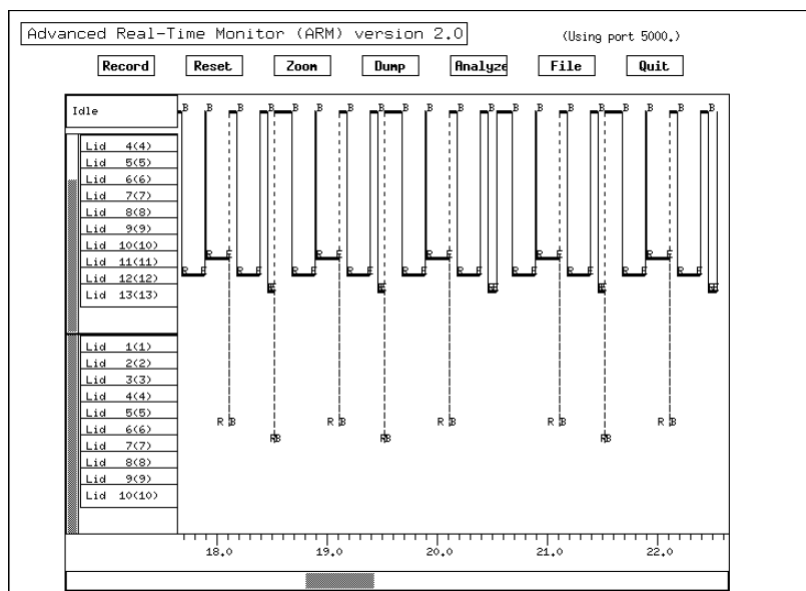
- A performance monitoring tool
- Designed to analyze and visualize the runtime behavior of target nodes in real time.
 - Allows user to reach into a remote target and view the scheduling events
 - Extracted using event taps in RT-Mach.

ARM Architecture



1. OS registers “events of interest”
 - System calls and arguments
 - Scheduler actions
 - Task states
 - Suspensions and Activations each with a high-resolution timestamp
2. Store in kernel buffer
3. Application makes system call periodically to obtain event buffer
4. Application plots data and interacts with user.

ARM Screen-Shot



ARM Messages

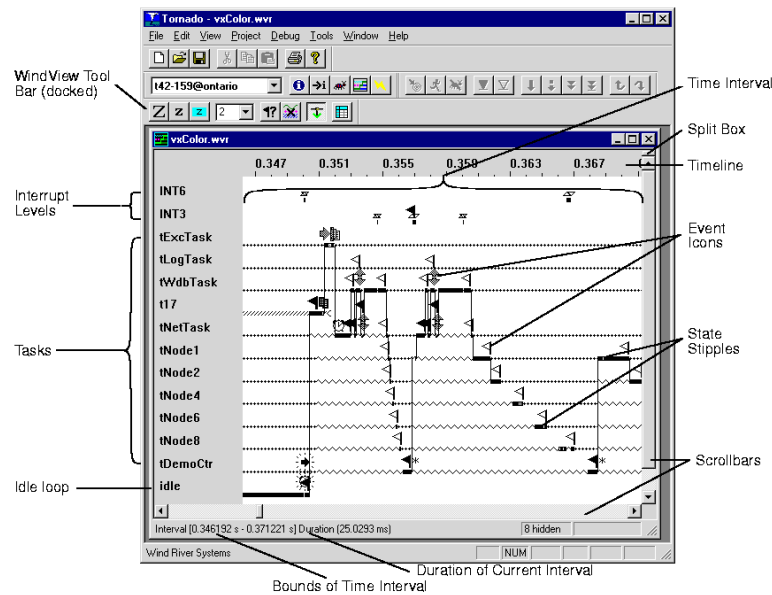
```
Time: 18850 ms.  
Periodic tasks : 1  
Aperiodic tasks : 10  
Total CPU utilization: 0.533 (cyclic tasks (0.444)) (acyclic tasks (0.089))  
Meet deadline : 64  
Missed deadline: 0  
aborted: 0  
Events: 1088 events 57,719 per second
```

Cautionary Notes

- The execution time of one instance of (say a periodic) task depends upon
 - # of instructions executed
 - Branches taken or not taken in code (data dependence)
 - # of loop iterations (more data dependence)
 - Overheads of context switching
 - Caching behavior
 - In both instruction and data caches
 - Demand-paging behavior: usually disabled for real-time tasks
 - Must block paging of code, data, stack *and* heap segments
 - Interference on system bus
 - Conflicts on I/O devices
 - Cycle-stealing on main memory
 - Speculative execution within processor
 - ...

System-
dependent

WindView from Wind River

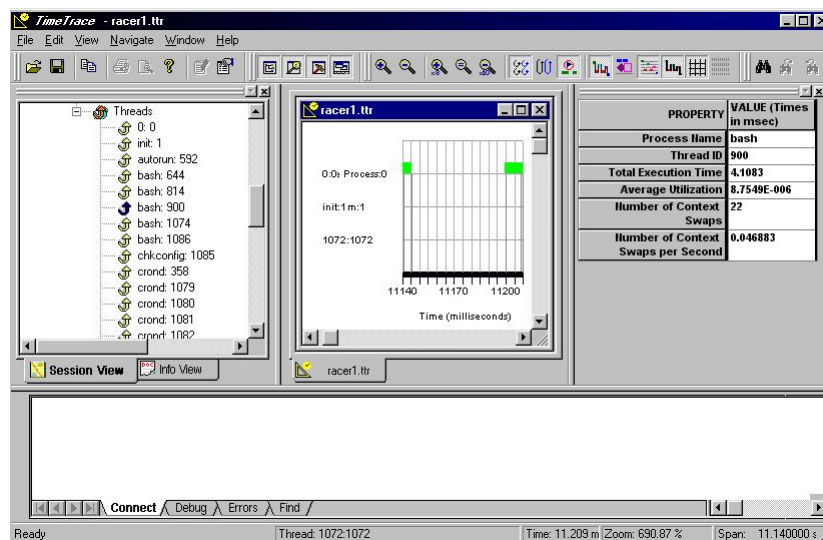


Carnegie Mellon

18-648: Embedded Real-Time Systems

Electrical & Computer
ENGINEERING

TimeTrace from TimeSys

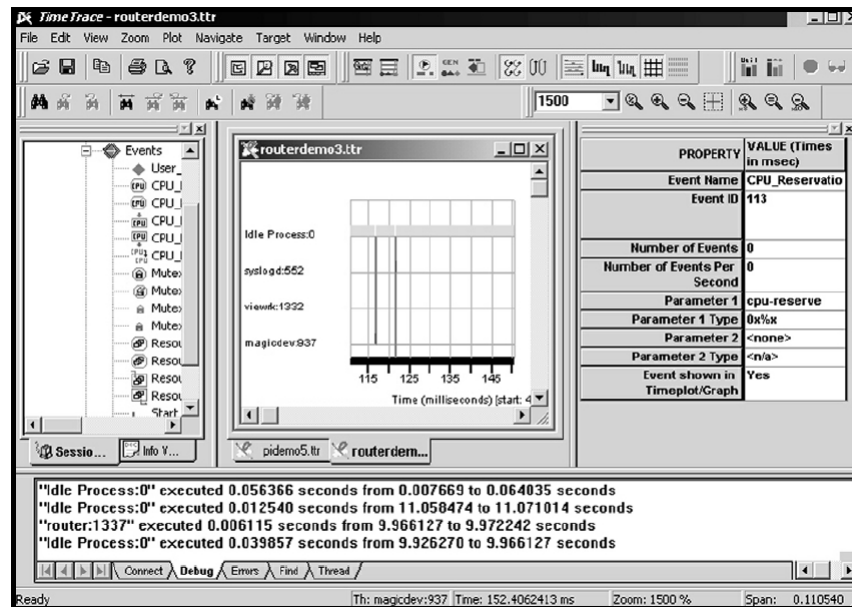


Carnegie Mellon

18-648: Embedded Real-Time Systems

Electrical & Computer
ENGINEERING

Real-Time Profiling Using TimeTrace



Carnegie Mellon

18-648: Embedded Real-Time Systems

Electrical & Computer
ENGINEERING

Some Features of TimeTrace

- Understand periodicity of tasks
 - Track period values
- Real-time data collection and plotting

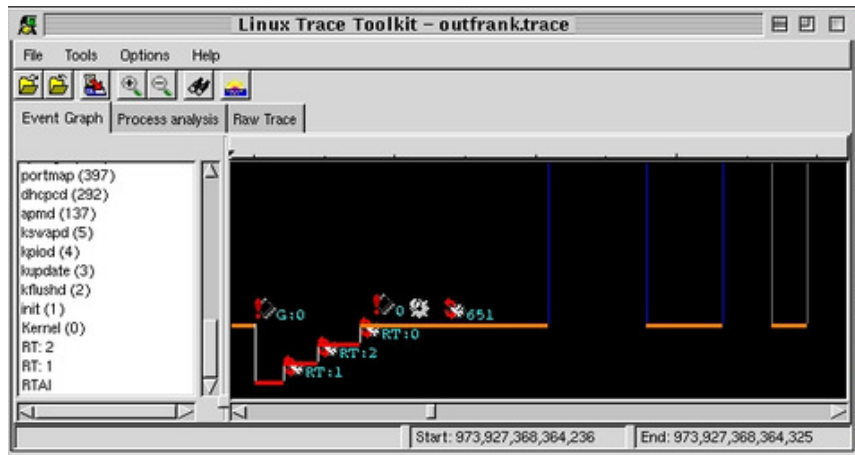
Carnegie Mellon

18-648: Embedded Real-Time Systems

Electrical & Computer
ENGINEERING

Task Interactions with Linux Trace Toolkit

- Open source version available for Linux



Carnegie Mellon

18-648: Embedded Real-Time Systems

Electrical & Computer
ENGINEERING

Raw Data from Linux Trace Toolkit

The screenshot shows the Linux Trace Toolkit interface with the 'outfrank.trace' file loaded. The 'Raw Trace' tab is active, displaying a table of raw trace data. The table has columns: CPU-ID, Event, Time, PID, Entry Length, and Event Description. The data shows various real-time events, including RT-Global IRQ entry, RT-Timer, RT-Switch to RT, RT-Sched change, RT-FIFO, RT-Task, and RT-Switch to Linux. The timeline at the bottom shows 'Start: 973,927,368,364,237' and 'End: 973,927,368,364,325'.

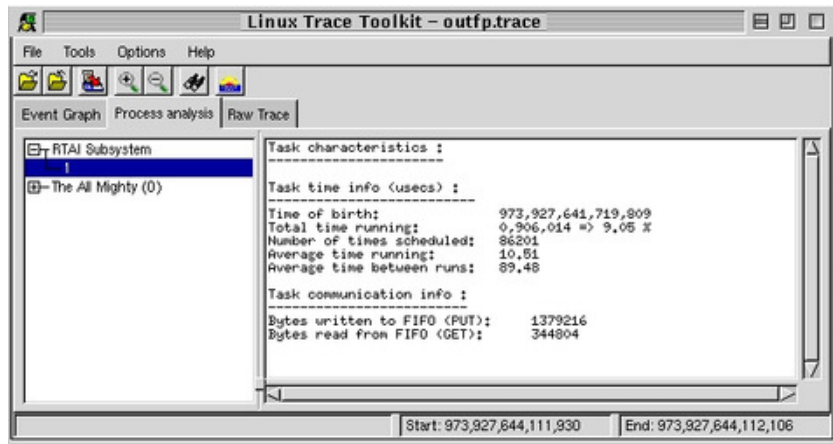
CPU-ID	Event	Time	PID	Entry Length	Event Description
0	RT-Global IRQ entry	973,927,368,364,240	RT:0	7	IRQ : 0, IN-KERNEL
0	RT-Timer	973,927,368,364,242	RT:0	18	TIMER EXPIRY
0	RT-Switch to RT	973,927,368,364,243	RT:0	7	CPU-ID : 0
0	RT-Sched change	973,927,368,364,244	RT:1	18	IN : 1; OUT : 0; STATE : 1
0	RT-FIFO	973,927,368,364,246	RT:1	18	GET => MINOR NB : 3; COUNT
0	RT-FIFO	973,927,368,364,247	RT:1	18	PUT => MINOR NB : 0; COUNT
0	RT-Task	973,927,368,364,248	RT:1	30	WAIT PERIOD
0	RT-Sched change	973,927,368,364,249	RT:2	18	IN : 2; OUT : 1; STATE : 5
0	RT-FIFO	973,927,368,364,251	RT:2	18	GET => MINOR NB : 4; COUNT
0	RT-FIFO	973,927,368,364,252	RT:2	18	PUT => MINOR NB : 1; COUNT
0	RT-Task	973,927,368,364,253	RT:2	30	WAIT PERIOD
0	RT-Switch to Linux	973,927,368,364,254	RT:2	7	CPU-ID : 0
0	RT-Sched change	973,927,368,364,255	RT:0	18	IN : 0; OUT : 0; STATE : 0

Carnegie Mellon

18-648: Embedded Real-Time Systems

Electrical & Computer
ENGINEERING

Execution Statistics Linux Trace Toolkit

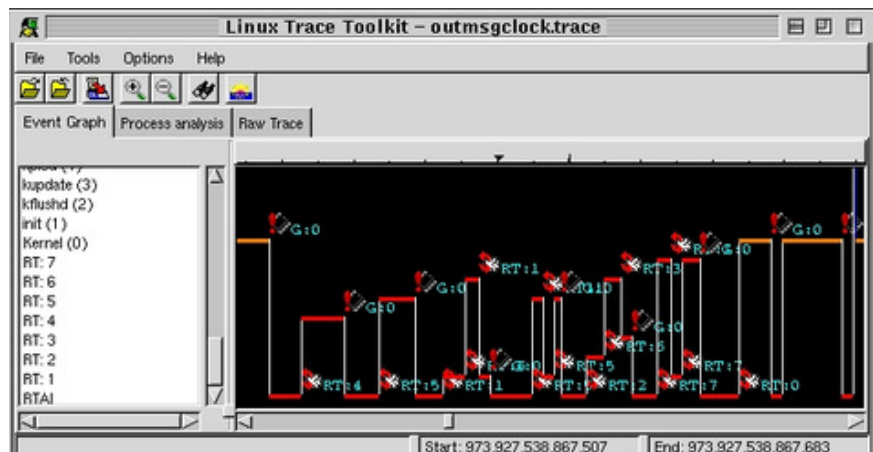


Carnegie Mellon

18-648: Embedded Real-Time Systems

Electrical & Computer
ENGINEERING

More Interactions in Linux Trace Toolkit



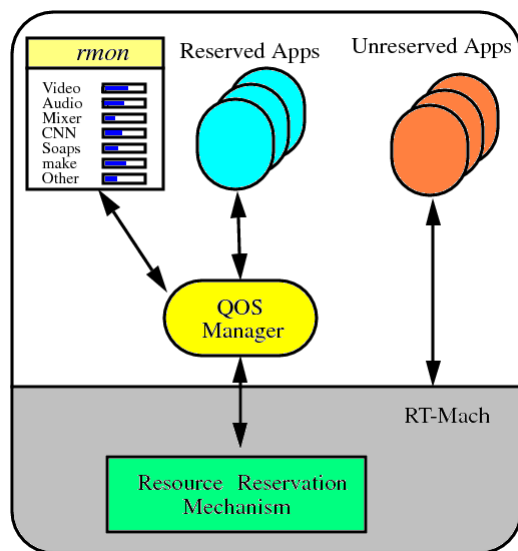
Carnegie Mellon

18-648: Embedded Real-Time Systems

Electrical & Computer
ENGINEERING

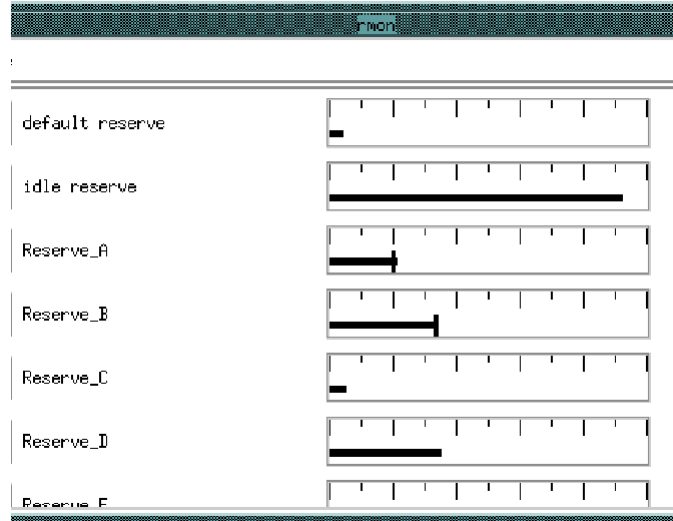
Working With Reservations

Reservation Monitoring and Adaptation



Reservation Monitoring

- **rmon** from Carnegie Mellon

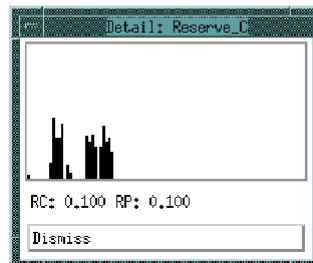


Carnegie Mellon

18-648: Embedded Real-Time Systems

Electrical & Computer
ENGINEERING

More rmon Features



Carnegie Mellon

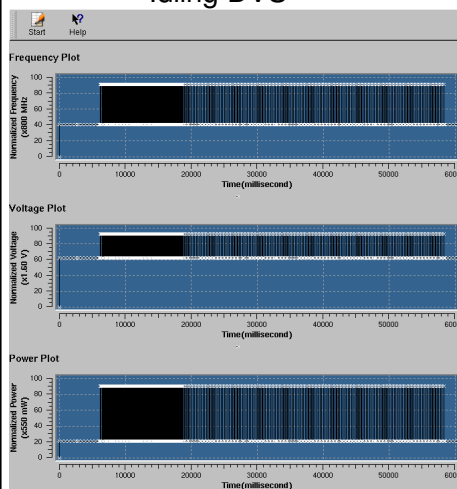
18-648: Embedded Real-Time Systems

Electrical & Computer
ENGINEERING

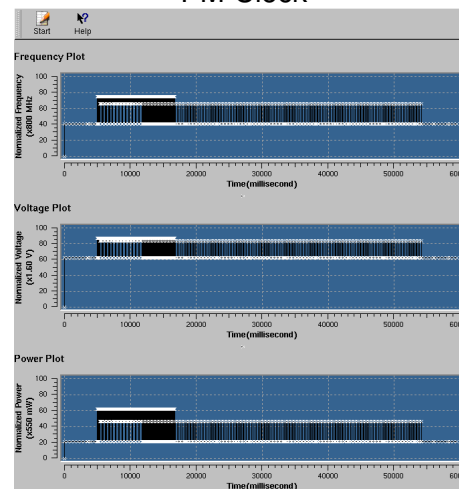
Working with DVS and DFS

PowerMon to monitor Energy & Clock Frequency

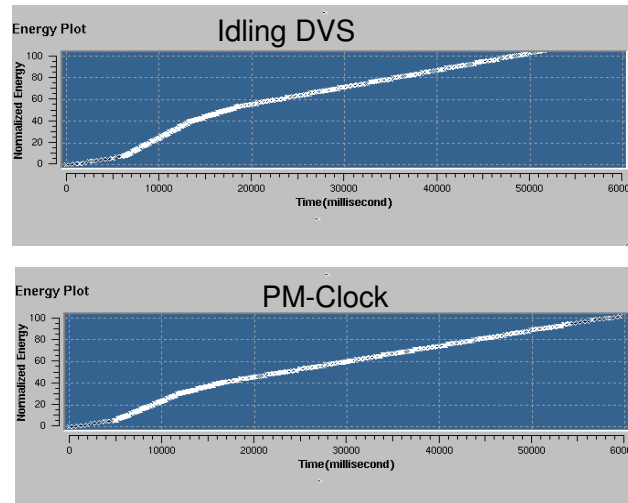
Idling DVS



PM-Clock



Energy Monitoring with PowerMon



Conclusions

- Optimizing compilers try to squeeze more performance out
 - Often, there is a tradeoff between optimizing memory space and execution time
- Real-time systems can have very different needs
- A host of performance monitoring tools is available