RIOS: A Lightweight Task Scheduler for Embedded Systems

Bailey Miller*, Frank Vahid*†, Tony Givargis†

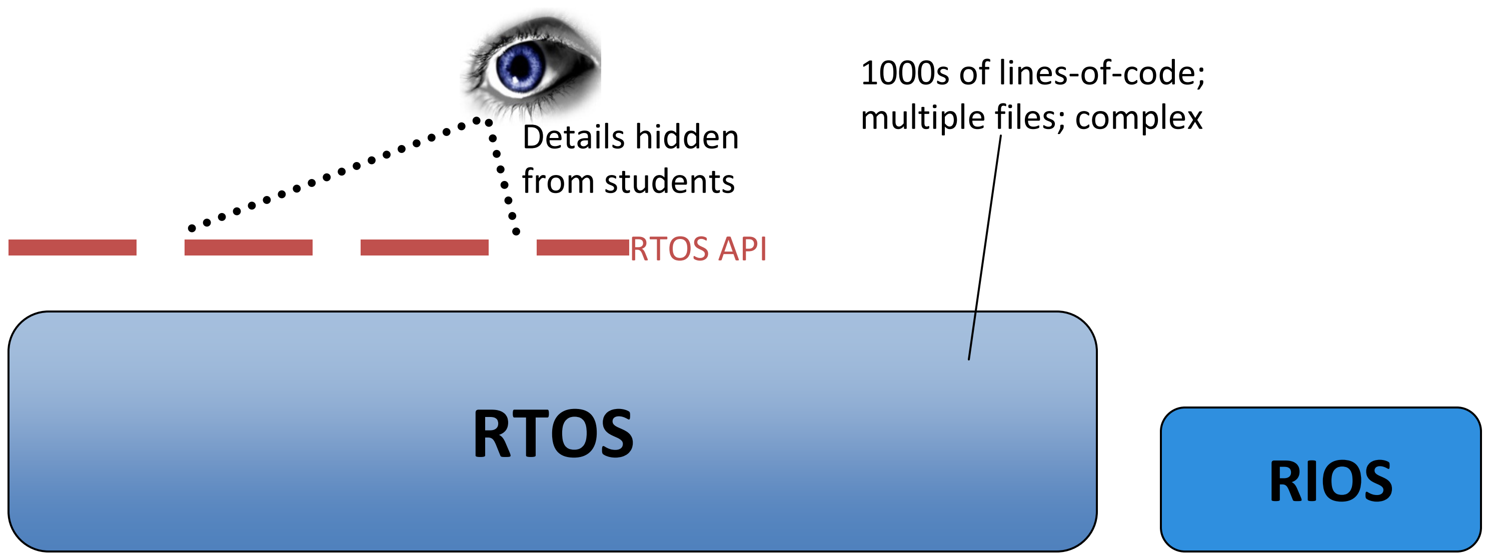
*Dept. Computer Science & Engineering
University of California, Riverside
{chuang,bmiller,vahid}@cs.ucr.edu

†Center for Embedded Computer Systems
University of California, Irvine
givargis@uci.edu

This work was supported in part by a U.S. Department of Education GAANN fellowship.

RIOS: Riverside-Irvine Operating System

- **Task scheduler for embedded systems education**
  - Developed and used at UCR.
  - 10+ classes at UCR, UCI, other universities.
- **Why?**
  - Size
  - Application Programming Interface (API)
  - Complexity

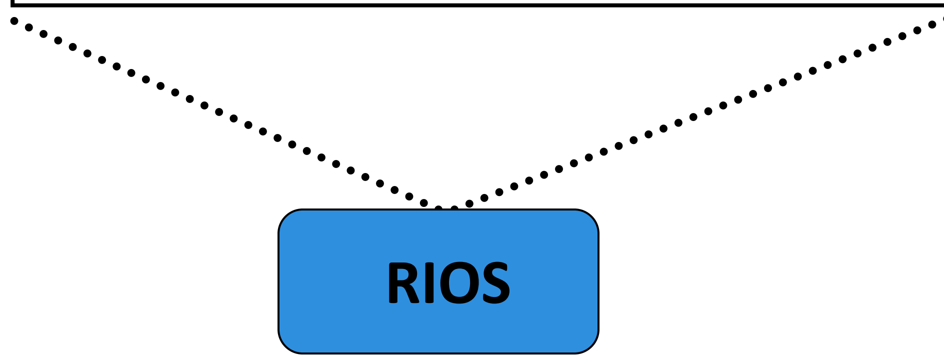


RIOS: Riverside-Irvine Operating System

- **Minimal code**
  - Nonpreemptive scheduler:
    - 12 lines-of-code
  - Preemptive scheduler:
    - ~20 lines-of-code
- **Cooperative tasks**
  - Run-to-completion tasks
  - Supports synch-SM programming model
- **Transparent**
  - Single stack
  - Understandable

RIOS Non-preemptive

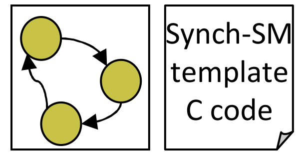
```
void TimerISR() {
  unsigned char i;
  if (processingRdyTasks) {
    printf("Timer ticked before task processing done.\n");
  }
  else { // Heart of the scheduler code
    processingRdyTasks = 1;
    for (i=0; i < tasksNum; ++i) {
      if (tasks[i].elapsedTime >= tasks[i].period) { // Ready
        tasks[i].state = tasks[i].TickFct(tasks[i].state); //execute task tick
        tasks[i].elapsedTime = 0;
      }
      tasks[i].elapsedTime += tasksPeriodGCD;
    }
    processingRdyTasks = 0;
  }
}
```



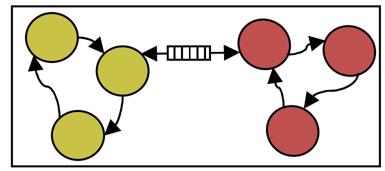
Context for teaching RIOS

- **Progressive methodology**

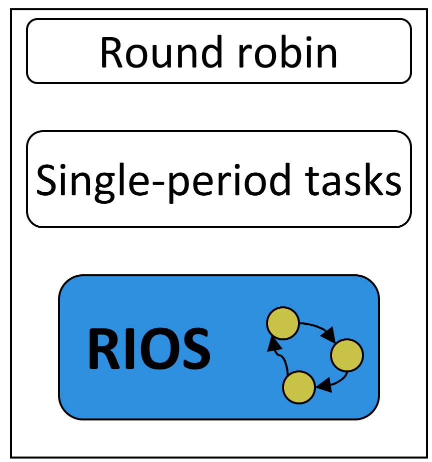
1. **Coding in C review** (1 week)
2. **Capturing behavior with Synch-SMs** (2 weeks)
3. **Synch-SMs and timing with C code** (1 week)
4. **Introduce more complexity (multi synch-SM)** (2 weeks)
5. **Scheduling** (3 weeks)
   - Simple round-robin
   - Priority-based scheduling of tasks with the same period
   - *RIOS – cooperative, non-preemptive priority-based scheduling*



Capturing simple behavior as synch-SMs



Multiple synch-SMs

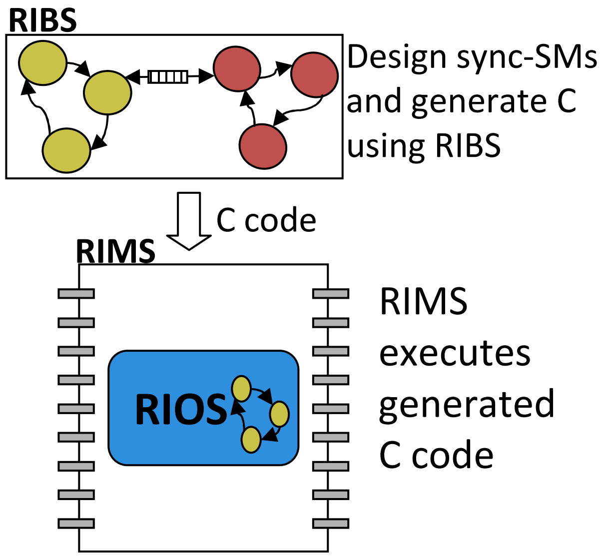


<u>Scheduling</u>

- Simple (Round robin)
- Priority-based scheduling
- RIOS

Context for teaching RIOS

- **Riverside-Irvine Microcontroller Simulator (RIMS)**
- **State machine builder (RIBS)**



http://www.youtube.com/watch?v=M0f3X4OGSSw

RIOS: Non-preemptive priority-based scheduling (simple tasks)

Task struct

Non-preemptive

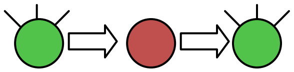
Check if tasks are ready

Execute task tick

Timer initialization

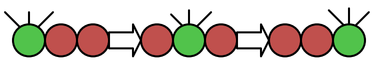
```
typedef struct task {
  unsigned long period; // Rate at which the task should tick
  unsigned long elapsedTime; // Time since task's last tick
  void (*TickFct)(void); // Function to call for task's tick
} task;

int tasksNum = 0;
task tasks[0];

unsigned char tasksPeriodGCD = 0;
unsigned char processingRdyTasks = 0;
void TimerISR() {
   unsigned char i;
   if (processingRdyTasks) {
      printf("Timer ticked before task processing done.\n");
   }
   else { // Heart of the scheduler code
      processingRdyTasks = 1;
      for (i=0; i < tasksNum; ++i) {
         if (tasks[i].elapsedTime >= tasks[i].period) { // Ready
            tasks[i].TickFct(); //execute task tick
            tasks[i].elapsedTime = 0;
         }
         tasks[i].elapsedTime += tasksPeriodGCD;
      }
      processingRdyTasks = 0;
   }
}

void main() {
   TimerSet(tasksPeriodGCD); // TimerISR activates at GCD of tasks
   TimerOn(); // Enable the timer

   while(1) { Sleep(); }
}
```

RIOS: Non-preemptive priority-based scheduling (simple tasks)



```
// Task: Toggle an output
void TickFct_Toggle() {
   static unsigned char init = 1;
   if (init) { // Init behavior
      B0 = 0;
      init = 0;
   }
   else { // Normal behavior
      B0 = !B0;
   }
}
```

Period=1000

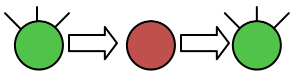


```
// Task: Strobe a 1 across 3 outputs
void TickFct_Sequence() {
   static unsigned char init = 1;
   unsigned char tmp = 0;
   if (init) { // Initialization
behavior
      B2 = 1; B3 = 0; B4 = 0;
      init = 0;
   }
   else { // Normal behavior
      tmp = B4;
      B4 = B3;
      B3 = B2;
      B2 = tmp;
   }
}
```

Period=200

http://www.youtube.com/watch?v=DscITuvL6Kk

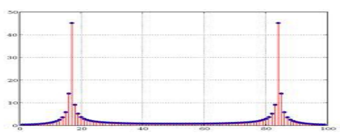
RIOS: preemptive priority-based scheduling

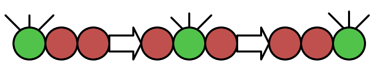


Period=1000

Delay = 500 ms

Low priority





Period=200

Delay = 25 ms

High priority

Scheduled tasks

Platform-specific context switch

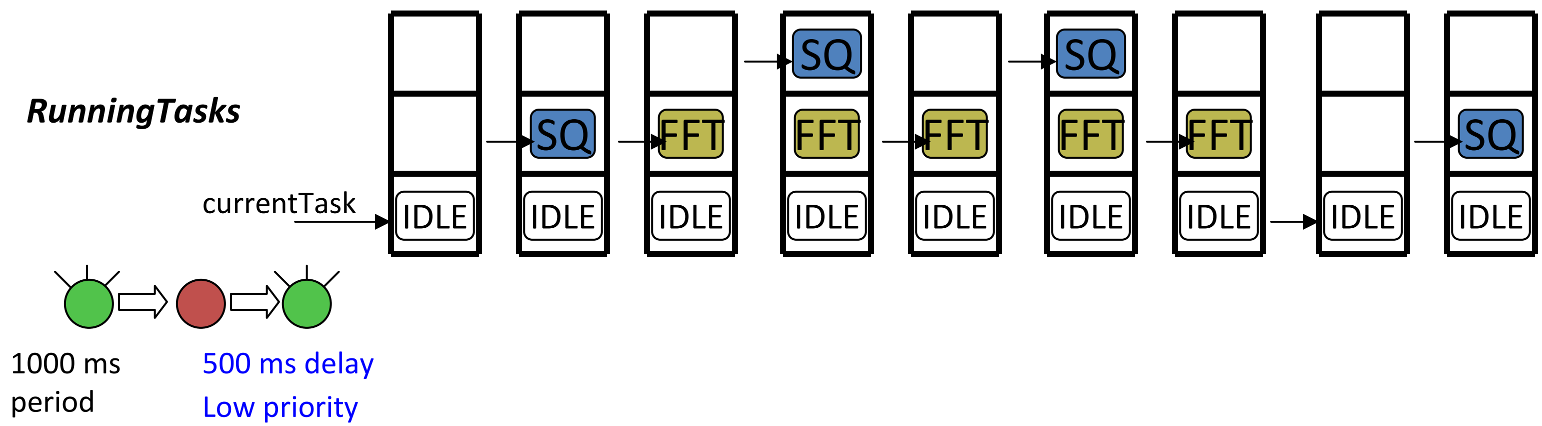
Add task

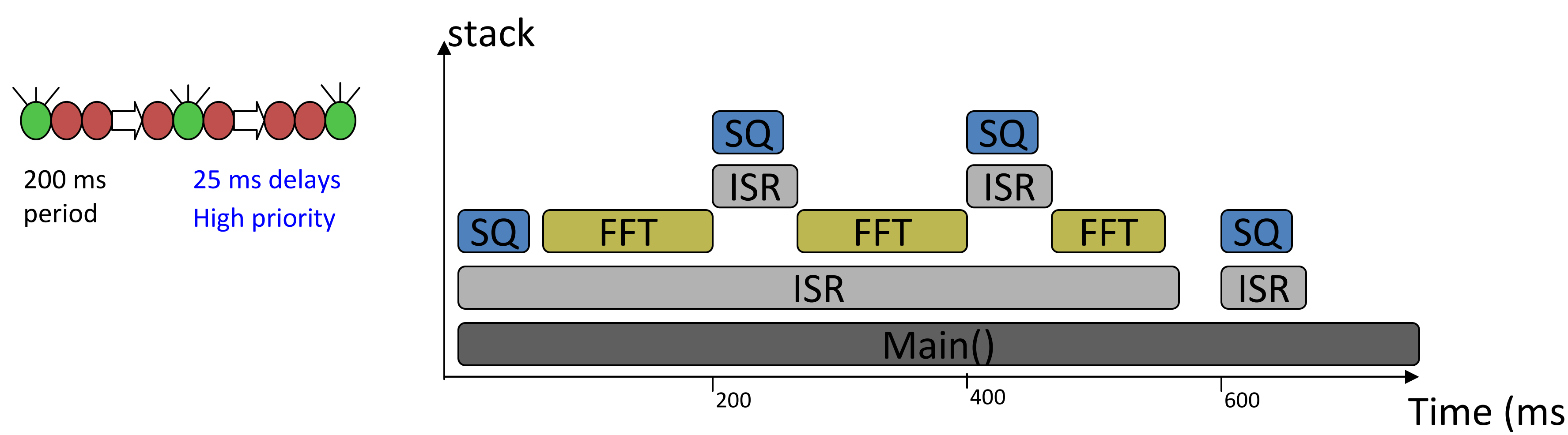
Execute task

Remove task

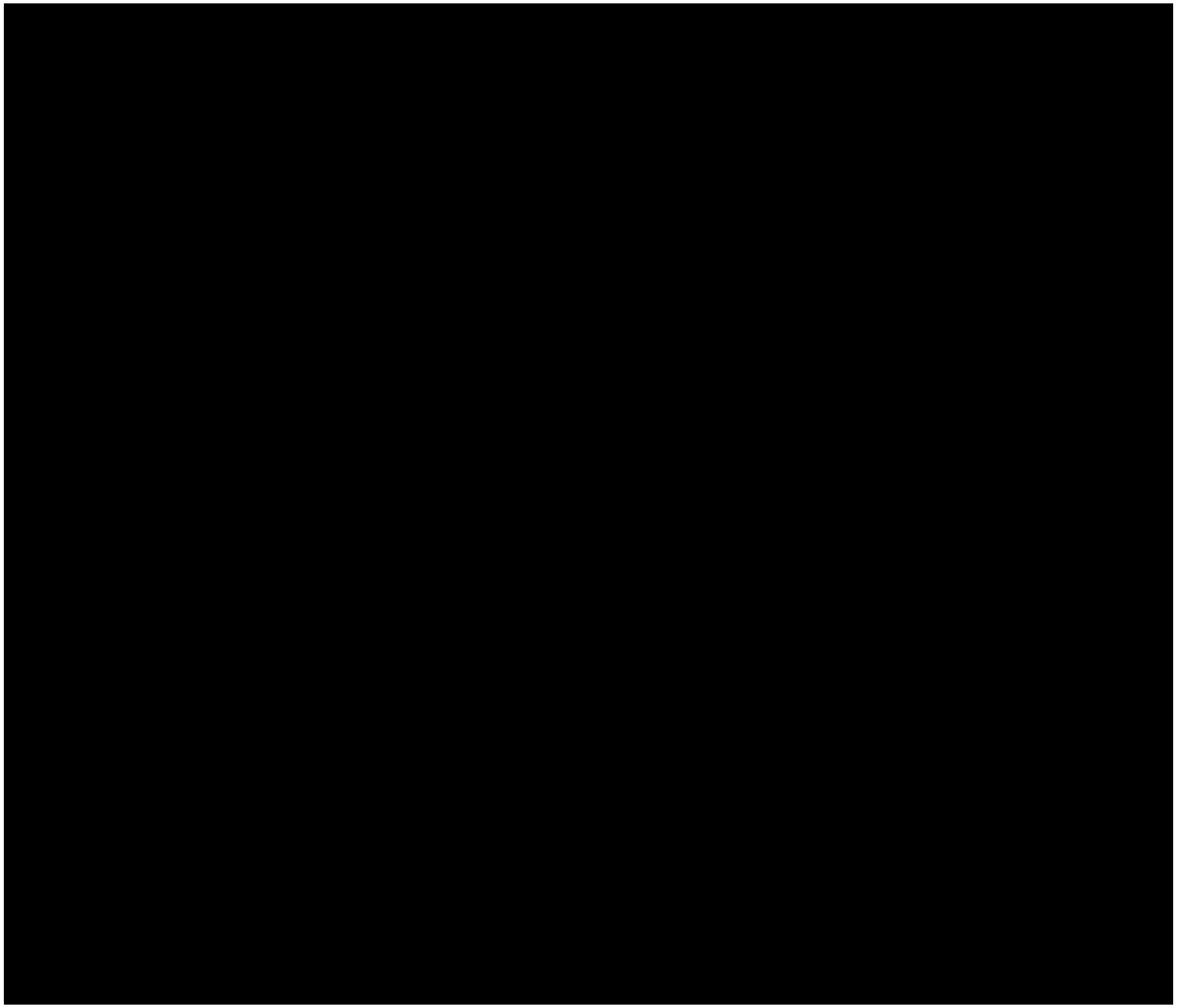
```
unsigned char runningTasks[3] = {255};//Track running tasks-[0] always idleTask
const unsigned long idleTask = 255; // 0 highest priority, 255 lowest
unsigned char currentTask = 0; //Index of highest priority task in runningTasks

void TimerISR() {
   unsigned char i;
   SaveContext(); //save temporary registers, if necessary

   for (i=0; i < tasksNum; ++i) { // Heart of scheduler code
     if ( (tasks[i].elapsedTime >= tasks[i].period) // Task ready
          && (runningTasks[currentTask] > i) // priority > current priority
          && (!tasks[i].running)) { // Task not running (no self-preemption)

         DisableInterrupts();          // Critical section
         tasks[i].elapsedTime = 0;     // Reset time since last tick
         tasks[i].running = 1;         // Mark as running
         currentTask += 1;
         runningTasks[currentTask] = i; // Add to runningTasks
         EnableInterrupts();             // End critical section

         tasks[i].TickFct(); // Execute tick

         DisableInterrupts();        // Critical section
         tasks[i].running = 0;
         runningTasks[currentTask] = idleTask;    // Remove from runningTasks
         currentTask -= 1;
         EnableInterrupts();                      // End critical section
      }
      tasks[i].elapsedTime += tasksPeriodGCD;
   }
   RestoreContext();
}
```

RIOS: preemptive priority-based scheduling (Synch-SMs)

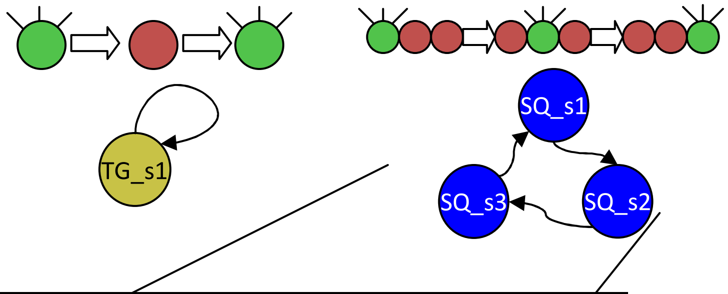




RIOS: preemptive priority-based scheduling (Synch-SMs)



RIOS: Non-preemptive priority-based scheduling (Synch-SMs)




```
enum SQ_States { SQ_s1, SQ_s2, SQ_s3 };
int TickFct_Sequence(int state) {
   switch(state) { // Transitions
      case -1: // Initial transition
         state = SQ_s1; break;
      case SQ_s1:
         state = SQ_s2; break;
      case SQ_s2:
         state = SQ_s3; break;
      case SQ_s3:
         state = SQ_s1; break;
      default:
         state = -1;
   } switch(state) { // State actions
      case SQ_s1:
         B2 = 1; B3 = 0; B4 = 0; break;
      case SQ_s2:
         B2 = 0; B3 = 1; B4 = 0; break;
      case SQ_s3:
         B2 = 0; B3 = 0; B4 = 1; break;
      default:
         break;
   }
   return state;
}
```

Current state

transitions

actions

Next state

```
typedef struct task {
  unsigned long period;// Rate at which the task should tick
  unsigned long elapsedTime; // Time since task's last tick
  int state; // state of task
  void (*TickFct)(void); // Function to call for task's tick
} task;
```

```
void TimerISR() {
   unsigned char i;
   if (processingRdyTasks) {
      printf("Timer ticked before task processing done.\n");
   }
   else { // Heart of the scheduler code
      processingRdyTasks = 1;
      for (i=0; i < tasksNum; ++i) {
        if (tasks[i].elapsedTime >= tasks[i].period) {
          tasks[i].state = tasks[i].TickFct(tasks[i].state);
          tasks[i].elapsedTime = 0;
        }
         tasks[i].elapsedTime += tasksPeriodGCD;
      }
      processingRdyTasks = 0;
   }
}
```

Update task state

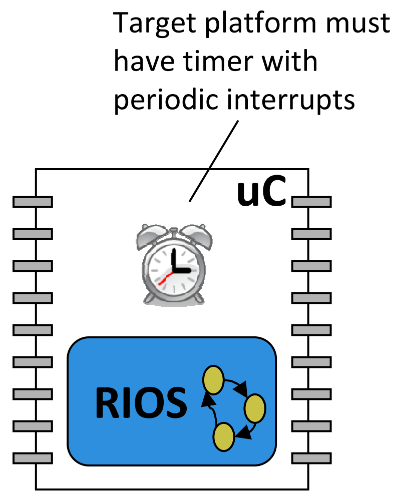
RIOS

```
void main() {
   // Task initialization
   // …
   while(1) { Sleep(); }
}
```


Bailey Miller, WESE'12

RIOS requirements

- **Required functions**
  - void TimerOn()
  - void TimerSet(int ms)
  - void TimerISR()
- **Portable**

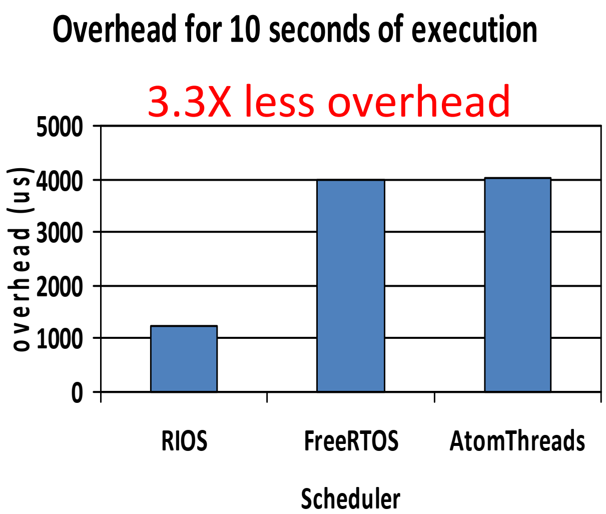


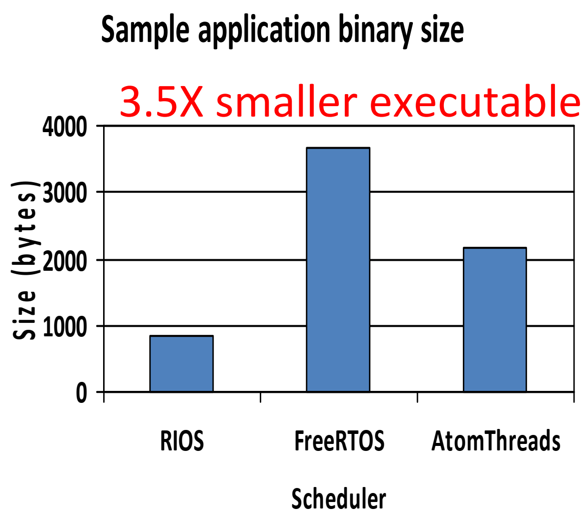
AVR microcontroller

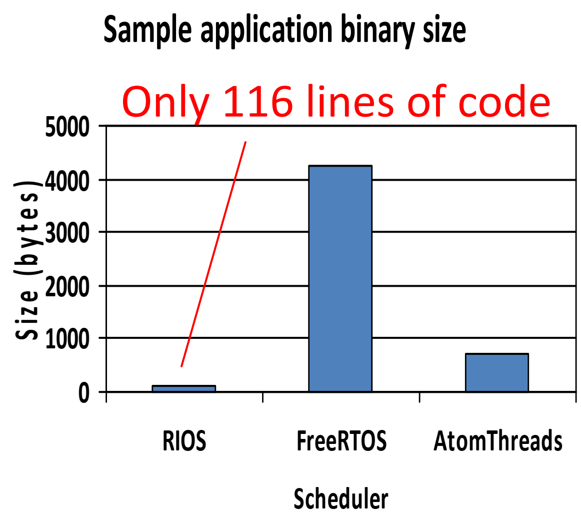
```
void TimerOn() {
  TCCR1B = (1<<WGM12)|(1<<CS12); //Clear timer on compare.
  TIMSK1 = (1<<OCIE1A); //Enables compare match interrupt
  SREG |= 0x80; //Enable global interrupts
}
void TimerSet(int milliseconds) {
  TCNT1 = 0;
  OCR1A = milliseconds*TICKS_PER_MS;
}
void TimerISR() {
  // RIOS scheduler code
}
ISR(TIMER1_COMPA_vect) { // Timer compare match ISR
  TimerISR();
}
```

Comparison to RTOSs

- Sample application of 3 task program
  - All implemented on AVR ATMega324P







Observations

http://www.youtube.com/watch?v=VkCYbA6kXng

- **Condensed summer class (5 weeks total).**
  - 8 days to complete final projects.

- **Average of 5 concurrent tasks, scheduled with RIOS**

Conclusion

- **Minimal** task scheduler suitable for most embedded applications using cooperative tasks
- **Small** code size
- **Understandable** by students
- **Portable**

Code / examples available from http://www.riosscheduler.org

RIOS Non-preemptive

```
void TimerISR() {
  unsigned char i;
  if (processingRdyTasks) {
    printf("Timer ticked before task processing done.\n");
  }
  else { // Heart of the scheduler code
    processingRdyTasks = 1;
    for (i=0; i < tasksNum; ++i) {
      if (tasks[i].elapsedTime >= tasks[i].period) { // Ready
        tasks[i].state = tasks[i].TickFct(tasks[i].state); //execute task tick
        tasks[i].elapsedTime = 0;
      }
      tasks[i].elapsedTime += tasksPeriodGCD;
    }
    processingRdyTasks = 0;
  }
}
```