Embedded SW Architectures
Round-robin
Function-queue scheduling
Software Architecture

- How to “do” things – how to arrange code for an embedded system application.
- Four variants:
  1. Round-robin
  2. Round-robin with interrupts
  3. Function-queue-scheduling
  4. Real-time Operating System
void main() {
    while(TRUE) {
        if(// I/O Device #1 needs service) {
            // Service I/O #1
        }
        if(// I/O Device #2 needs service) {
            // Service I/O #2
        }
        ...
        if(// I/O Device #n needs service) {
            // Service I/O #n
        }
    }
}
Round-robin: Poll and Serve

Pro: Very simple, straightforward, no interrupts

Cons:
1. If a device needs faster service than the “cycle-time”, it may not work.
2. If there is a lengthy processing, the system may not react fast enough.
3. Very fragile – hard to extend, reprogram, change.
4. No interrupts!
Round-robin with interrupts (1)

```c
bool fDev1 = FALSE, fDev2 = FALSE, … fDevn = FALSE;
void interrupt vHandleDev1() {
    // Handle Dev 1
    fDev1 = TRUE;
}
…
void interrupt vHandleDevn() {
    // Handle Dev n
    fDevn = TRUE;
}
```

For every device: ISR handles I/O termination and sets flag.
Round-robin with interrupts (2)

void main() {
    while (TRUE) {
        if(fDev1) {
            fDev1 = FALSE;
            // Handle data from Dev 1
        }
    }
    ...
    if(fDevn) {
        fDevn = FALSE;
        // Handle data from Dev n
    }
}

For every device: if device needs attention, main handles data and clears flag.
Round-robin with interrupts

- ISRs: handlers for I/O, main(): “task code”
- ISR-s ensure fast initial reactions to devices
- Priorities:
  \[ v\text{HandleDev}_1 > v\text{HandleDev}_2 > \ldots v\text{HandleDev}_n \text{ (per IT priority)} \]

Problem:
All “tasks” (non-ISR codes) are handled with the same priority

Solution:
Move “task code” into ISR
  May slow down system (longer ISR!)
  Change the order of flag-polling in main()
    Priority through polling order
WCRT: total exec time for all task codes + all ISR-s
Function-Queue-Scheduling

Queue data structure:

Queue enqueue(Queue, Elem);
Elem dequeue(Queue);

First-in-first-out (FIFO) order
Variant: Priority Queue
List of queues ordered according to priority

Queue of elements
void interrupt vHandleDev1() {
    // Handle Dev 1
    enqueue(q,fP1);
}

ISR: Take care of device and enqueue corresponding function pointer.

... 

void interrupt vHandleDevn() {
    // Handle Dev n
    enqueue(q,fPn);
}

Function-Queue-Scheduling (2)

void fP1() {
    // Task code for Dev 1
}
...
void fP2() {
    // Task code for Dev
}
void main() {
    while(TRUE) {
        while(//Queue of function pointers is not empty) {
            fP = dequeue(q);
            // call fP
        }
    }
}

Task functions for each task.

Q: A shared Queue

Dequeue next function pointer and call function.
Function-Queue-Scheduling

WCRT – if priority queue is used:
Longest task code + exec time of ISRs
Tradeoff: Response time for low-priority task code may get worse!
  – “Starvation” because of higher-priority interrupts
Real-Time OS (1)

void interrupt vHandleDev1() {
    // Handle Dev 1
    // Send signal #1
}

ISR: Take care of device and send a unique signal.

... void interrupt vHandleDevn() {
    // Handle Dev n
    // Send signal #n
}
Real-time OS (2)

void task1() {
    // Wait for signal #1
    // Task code for Dev 1
}
...
void taskn() {
    // Wait for signal #n
    // Task code for Dev
}
void main() {
    // Start task1
    ...
    // Start taskn
}
Real-Time OS Architecture

- RTOS provides:
  - Task creation and processor scheduling services
  - Processor is time-sliced across tasks
  - Signaling services (send() and wait())

Arrows indicate context (task) switching points.
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>Priorities</th>
<th>WCRT</th>
<th>Stability at code changes</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round-robin</td>
<td>None</td>
<td>Sum of all task code</td>
<td>Poor</td>
<td>Very simple</td>
</tr>
<tr>
<td>Round-robin with interrupts</td>
<td>ISR in priority order, tasks same priority</td>
<td>Total of all task code + all ISRs</td>
<td>Poor if task code is changed</td>
<td>Shared data problem (ISRs and task code)</td>
</tr>
<tr>
<td>Function Queue Scheduling</td>
<td>ISRs and task code in priority order</td>
<td>Execution time for the longest task code + ISRs</td>
<td>Queue mgmt is critical</td>
<td>Shared data problem and function queue code</td>
</tr>
<tr>
<td>RTOS</td>
<td>ISRs and task code in priority order</td>
<td>$0 + ISR$ execution times</td>
<td>Very good</td>
<td>High (needs kernel)</td>
</tr>
</tbody>
</table>