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

Round-robin: Poll and Serve

```
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)

```
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:

  vHandleDev1 > vHandleDev2 > … vHandleDevn (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!)

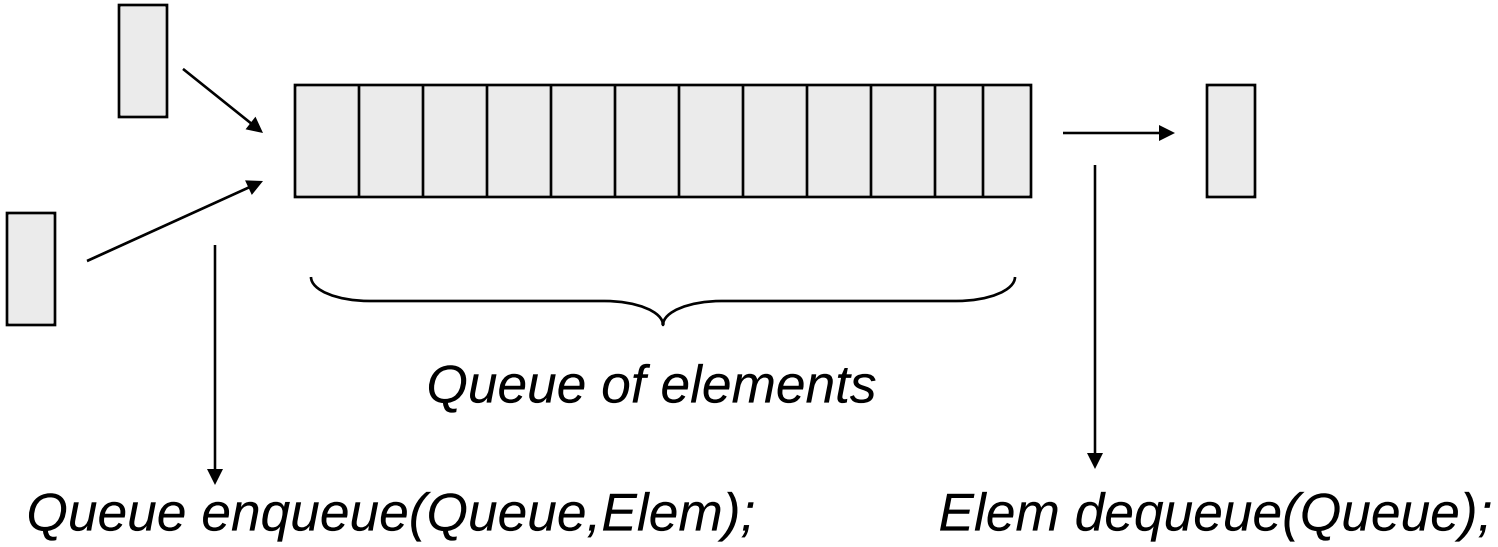
Change the order of flag-polling in main()

Priority through polling order

WCRT: total exec time for all task codes + all ISR-s

Function-Queue-Scheduling

Queue data structure:



First-in-first-out (FIFO) order

Variant: Priority Queue

List of queues ordered according to priority

Function-Queue-Scheduling (1)

```
void interrupt vHandleDev1() {
    // Handle Dev 1
    enqueue(q,fP1);
}
…
void interrupt vHandleDevn() {
    // Handle Dev n
    enqueue(q,fPn);
}
```

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

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
}
…
void interrupt vHandleDevn() {
    // Handle Dev n
    // Send signal #n
}
```

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

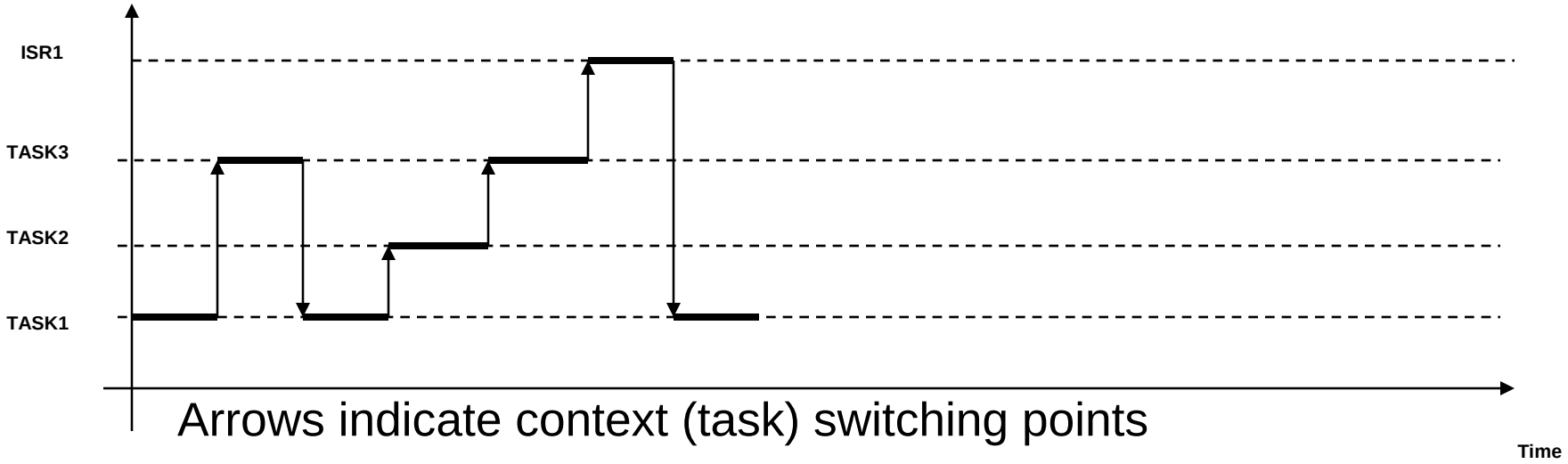
Real-time OS (2)

```
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
}
```

Task: A concurrent activity, with its own thread of control/stack (“context”)

Real-Time OS Architecture

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



Comparison

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