Overview

Exercise Bussystems

- Motivation
- Overview The ToolChain
- Multitasking in Mikrocontrollers
- Exercise
- Question & Answers
- Homework

Sources

- → http://hlb-labor.de
- → http://thetoolchain.com
- → http://thetoolchain.com/documentation/

Motivation

Chrysler Grand Cherokee: Software-Fehler kann ungewollt Gangschaltung verstellen

Sonntag, 12.05.2013 – 13:18 Uhr

Chicago - Chrysler ruft weltweit fast eine halbe Million Geländewagen wegen notwendiger Software-Aktualisierungen zurück. Etwa 469.000 Autos seien betroffen, teilte die Fiat-Tochter am Samstag mit. Bei einigen Fahrzeugen habe eine fehlerhafte Programmierung dazu geführt, dass die Gangschaltung ungewollt verstellt wurde.

Motivation

- Connecting devices on a bus is simple
- Developing Software for bus is hard
- Software effort often underestimated
- Who writes this software?

The ToolChain

- Collection of Scripts
- Automatically downloads aprox. 700MB
- Sources patched and compiled
- Compiler binaries
- Programmer/ Debugger
- uC Libraries

Features The ToolChain

- Downloaded + installed automatically
- FreeRTOS Multitasking Scheduler
- STM-StdPeripheralsLibrary
- Compiler Suite ARM GCC
- OpenOCD for In Circuit Programming/ Debugging
- additional libraries (math, lcd, lan, …)
- configuration files for different proto boards
- fexible configuration
  → same code can run on different boards
- out of the box running code examples

Multitasking

in

Embedded Systems

Multitasking in ES

- What is Singletasking?
- What is Multitasking?
- Why Multitasking?
- Different approaches
- Realtime Operating Systems
- FreeRTOS
- Synchronization
- Example Project
- Debugging Multitasking Software

What is Singletasking?

- Do only one thing at a time
- Most efficient way to solve problems
- Applicable to every algorithm
- No management overhead
- No internal synchronization required
- Easy to code
- Busy-wait for I/O
- No Interrupts

Singletasking Example

```
while (1) {
    if ( Byte = receiveByte() ) { // blocks!
        Buffer[Index++] = Byte;
        if ( Index >= MessageSize) {
        M_t* M = (M_t*) Buffer;
            switch (M->Type) {
                case mt_CommandA: ...
                case mt_CommandB: …
                default: break;
            }
            Index = 0;
        }
    }
}
```

What is Multitasking?

- Aim for multiple targets
- Switch context often
- Management overhead
- Synchronization required
- Interrupts required
- Hard to code/ debug
- Implicit delays
- Increased memory usage
- Implementation difficult for many algorithms

Multitasking Example

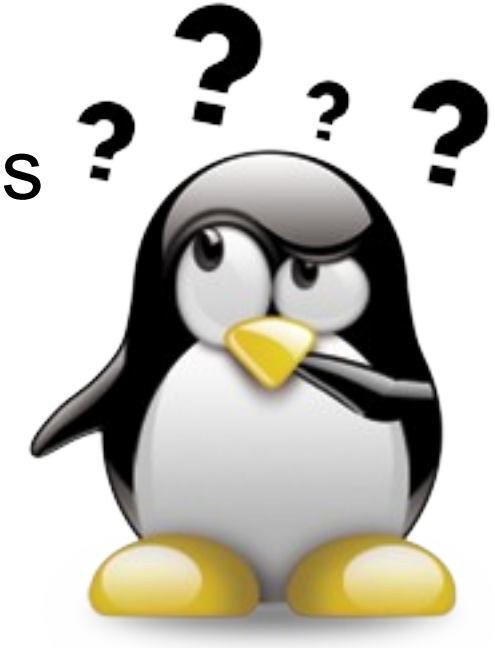
```
main() {
     int Queue = ttc_queue_create(...);
     xTaskCreate(Receive, Queue, …);
     xTaskCreate(Process, Queue, …);
}

void Process(int Q) {
  while (1) {
    M = (M_t*) ttc_queue_pop_front(Q);
    if (M) {
      switch (M->Type) {
        case mt_CmdA: ...
        case mt_CmdB: ...
        default: break;
      } } } }
```

```
void Receive(int Q) {
  char Buffer[10][100];
  int Index = 0;
  while (1) {
    char* Writer = &(Buffer[Index,0]);
    int Remaining = MessageSize;
    while (Remaining > 0) {
      if ( Byte = readByte() ) { // sleeps!
        *Writer++ = Byte; Remaining--;
    }
    ttc_queue_push_back(Q, &(Buffer[Index,0]););
    Index++;
    if (Index > 99)  Index = 0;
  }
}
```

Why Multitasking?

- Functions spawnable multiple times
- Eases handling of slow IO
- Benefits from multiple CPU-cores
- Only 1 central Timer required
- Short Interrupt Service Routines
- Less global variables required



No life without Multitasking!

- Every Embedded System needs MT
- MT often implemented via Interrupts
  - Complex Service Routines
  - Data Exchange via Global Variables
  - Difficult to debug
- Typical approach: Super-Loop
  - Periodically starts set of functions
  - Similar to task scheduler

Different Approaches

- Multiprogramming
  - Ancient mechanism for Periheral Access
  - Realized via Terminal Stay Ready (TSR)
- Cooperative Multitasking
  - Central Scheduler manages Processes
  - Each process grants CPU to other processes
  - Single process can block whole system
- Preemptive Multitasking
  - Scheduler interrupts each process periodically
  - Requires central Interrupt-Timer
- Preemptible Multitasking
  - High priority Applications can interrupt others (OS/2, Linux, FreeRTOS)
  - Allows faster response times

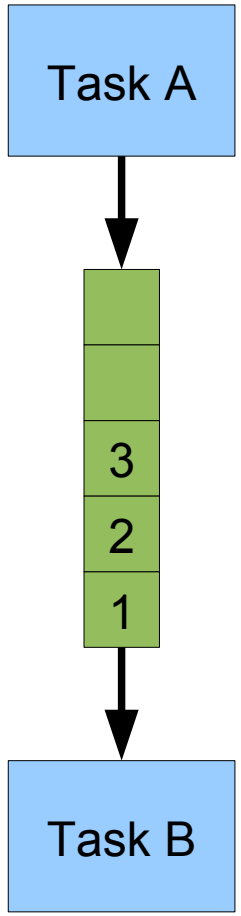
FreeRTOS

- Multitasking Scheduler
  - Premptible Multitasking
  - High triority tasks block low priority ones
- Inter Thread Communication
  - Semaphores
  - Queues
- Developed specially for Embedded Systems
- Open Source
- FreeWare with Commercial Support
- Ported to several µC Architectures

  → http://www.freertos.org/

FreeRTOS – Queues

- Base of inter task communication
- Send message
  - Task → Task
  - Interrupt Service Routine → Task
- Call by value
- Reading from empty queue
  - Function call waits until Queue is filled
  - No CPU time is wasted during Wait



ttc_queue

- Three types of queues in The ToolChain
  - Generic Queues
    ttc_queue_*()
  - Byte Queues
    ttc_queue_byte_*()
  - Pointer Queues
    ttc_queue_pointer_*()

Generic Queues

- Stores elements of any size
- Size of elements defined by create()
- Base functions (→ ttc_queue.c/ .h)
  - ttc_queue_create()
  - ttc_queue_push_back()
  - ttc_queue_pop_front()

Byte Queues

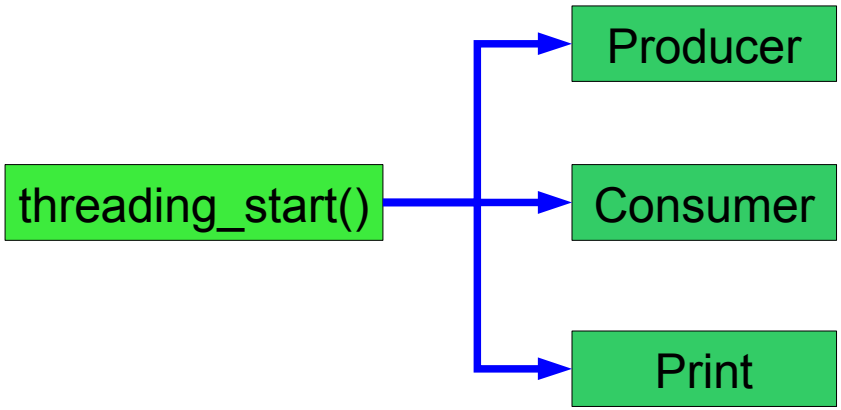
- Stores individual Bytes
- Advantages over Generic Queues
  - Less Memory Overhead
  - Faster PUSH and POP Operations
- Base functions (→ ttc_queue.c/ .h)
  - ttc_queue_byte_create()
  - ttc_queue_byte_push_back()
  - ttc_queue_byte_pop_front()

Pointer Queues

- Every Element stores a Pointer
- Advantages over Generic Queues
  - Less Memory Overhead
  - Faster PUSH and POP Operations
- Base functions (→ ttc_queue.c/ .h)
  - ttc_queue_pointer_create()
  - ttc_queue_pointer_push_back()
  - ttc_queue_pointer_pop_front()

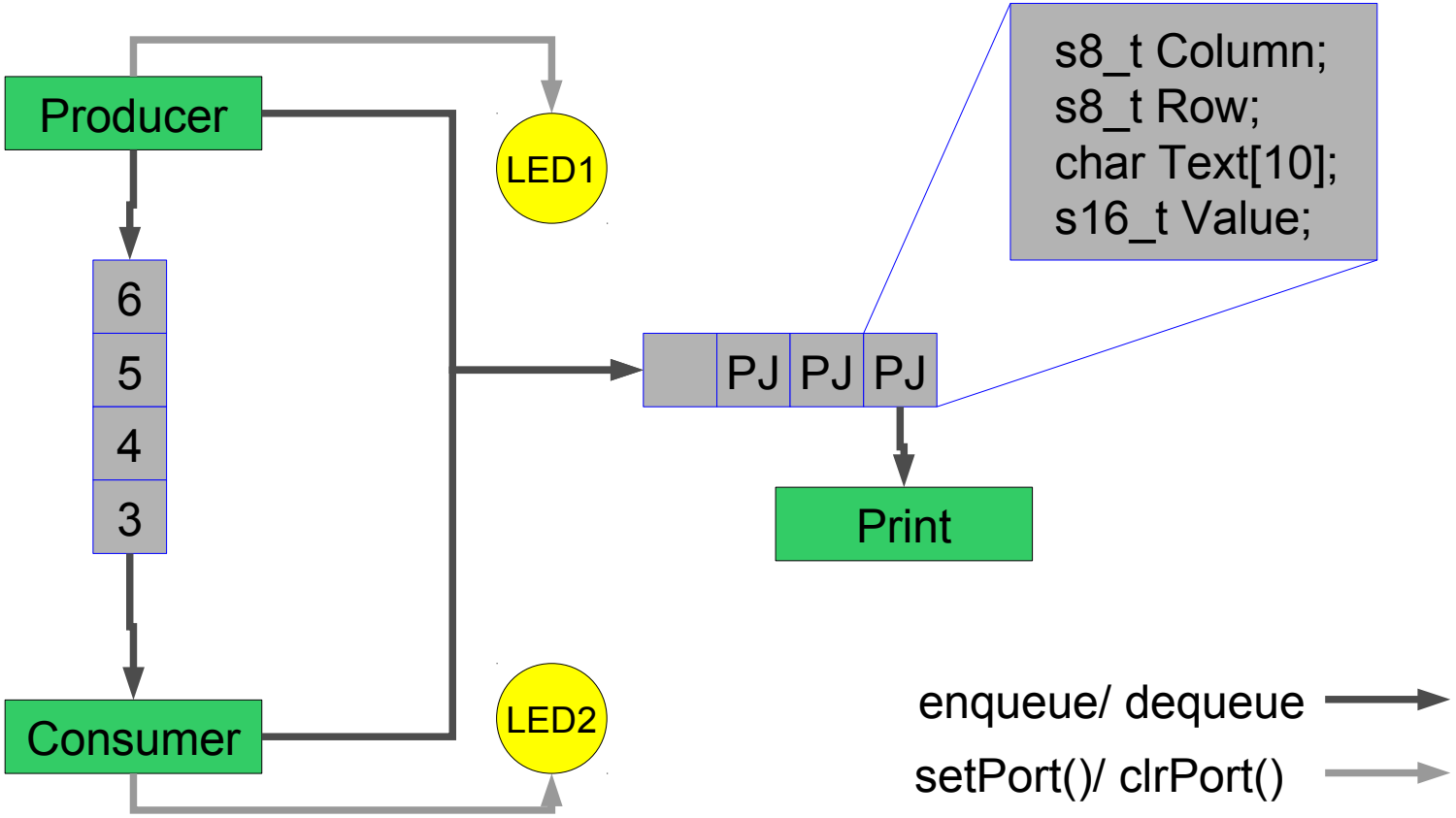
Multithreading with Queues

- Spawning Tasks



Multithreading with Queues

- Communicate via Queues



Multithreading with Queues

- Activate Extensions for Semaphores example on STM32-P107

```
> activate.600_example_threading_queues.sh
```

Homework

- → http://thetoolchain.com/documentation/
- Handwritten Elaboration (1 DIN A4-Page)
  - Difference Project- and ToolChain-Folder?
  - Difference ./compile.sh and ./_/compile.sh?
  - What are Extensions?
  - What are Assert() calls used for?
  - What are the general type of Compilation Errors?
- Until next exercise date (1 week at least)
  → Required to participate next exercise!