Zephyr – Die flexible Open-Source-Lösung für kleine Embedded-Systeme

Mit dem Echtzeit-Betriebssystem (RTOS) Zephyr steht Entwicklern eine vielseitige Open-Source-Alternative zur Verfügung, die für kleine Systeme optimiert ist. Anders als Linux, das für leistungsstarke Systeme gedacht ist, eignet sich Zephyr besonders für Mikrocontroller-basierte Anwendungen, bei denen Speicher und Rechenleistung begrenzt sind. Womit überzeugt Zephyr im Einzelnen?

Linux stößt an Grenzen, wenn es auf kleinen Mikrocontrollern eingesetzt wird, die wenig RAM und Flash-Speicher bieten. Selbst die einfachsten Linux-Implementierungen benötigen mindestens 4 MB RAM, was bei typischen Mikrocontrollern oft nicht verfügbar ist. Genau hier spielt Zephyr seine Stärken aus: Mit einer minimalistischen und anpassbaren Architektur ist es für Systeme mit begrenzten Ressourcen optimiert und bietet die grundlegenden Funktionen eines RTOS – ohne den großen Speicherbedarf von Linux.

Ein Betriebssystem, das nur aus einem einzigen ausführbaren Modul besteht

Zephyr kann auf einer breiten Hardwarebasis, von kleinen Mikrocontrollern bis hin zu komplexen Systemen, betrieben werden. Über ein konfigurierbares Menü lassen sich Treiber, Bibliotheken und Benutzeranwendungen anpassen. Dabei besteht das gesamte Betriebssystem aus einem einzigen ausführbaren Modul, was eine besonders effiziente Nutzung der Systemressourcen erlaubt.

Ein zentraler Vorteil der Architektur ist das flexible Scheduling: Zephyr bietet ein **Priority-Based Preemptive Scheduling**, bei dem Tasks mit höheren Prioritäten jene mit niedrigeren unterbrechen. Ergänzend können Entwickler auf verschiedene Prioritätsbereiche zugreifen, die sich durch zusätzliche Feineinstellungen, wie kooperative Threads und präemptive Prioritätsklassen, noch weiter individualisieren lassen.

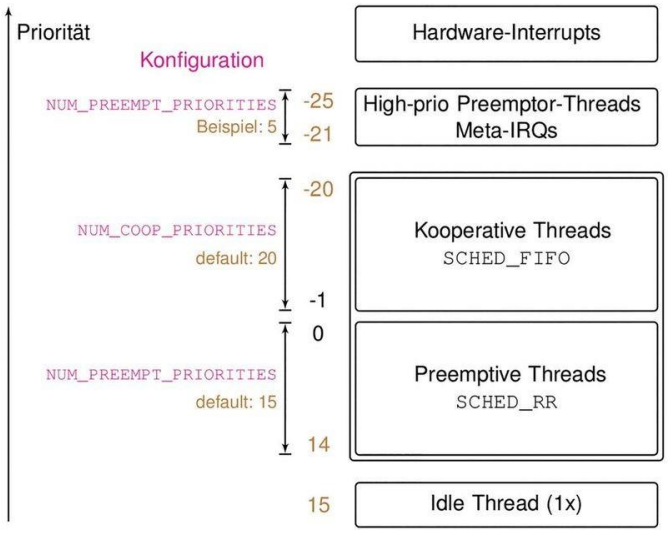

Bild 1: In dem vom Scheduler unterstützten Prioritätsband gibt es konfigurierbare Bereiche für unterschiedliche Threadtypen.

Effiziente Speicherverwaltung verhindert Übergriffe zwischen Threads

In einem System, das mehrere Threads gleichzeitig verarbeitet, ist eine zuverlässige Speicherverwaltung unerlässlich. Zephyr verwendet eine Memory Protection Unit (MPU), die den Threads nur den jeweils benötigten Speicherbereich zuweist und so Übergriffe zwischen Threads verhindert. Diese Architektur bietet Entwicklern Flexibilität und Sicherheit zugleich.

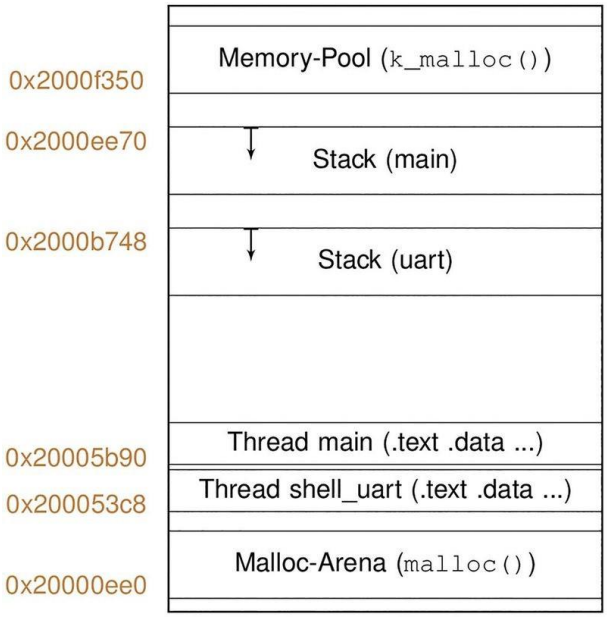

Bild 2: Beispiel für ein Memory-Layout mit zwei Threads, Kernel- sowie Userspace-Heap. Im Vergleich zur Bildschirmausgabe (Bild 3) wurden die restlichen Threads der Übersichtlichkeit wegen weggelassen.

```
uart:~$ kernel stacks
0x200056d0 tracing_thread   (real size 1024): unused  824 usage  200 / 1024 (19 %)
0x200058e0 rx_q[0]          (real size 1536): unused 1044 usage  492 / 1536 (32 %)
0x200057f8 net_mgmt         (real size  768): unused  584 usage  184 /  768 (23 %)
0x200059e0 tcp_work         (real size 1024): unused  872 usage  152 / 1024 (14 %)
0x200006e0 stm_eth          (real size 1504): unused 1040 usage  464 / 1504 (30 %)
0x20005ca0 sysworkq         (real size 1024): unused  792 usage  232 / 1024 (22 %)
0x200053c8 shell_uart       (real size 2048): unused 1088 usage  960 / 2048 (46 %)
0x20004eb8 logging          (real size  768): unused  552 usage  216 /  768 (28 %)
0x20005ac8 idle             (real size  320): unused  140 usage  180 /  320 (56 %)
0x20005b90 main             (real size 4096): unused 3512 usage  584 / 4096 (14 %)
0x2000d4c0 IRQ 00           (real size 2048): unused 1496 usage  552 / 2048 (26 %)
```

Bild 3: Übersicht über die vorhandenen Threads und deren bislang maximale Stack-Nutzung

Zephyr ist ideal für energiearme, langlaufende Anwendungen

Zephyr unterstützt sowohl periodische Timer-Events als auch eine ticklose Architektur, die eine noch feinere Steuerung der zeitlichen Auflösung erlaubt. Ein System mit aktiviertem „Tickless-Kernel“ generiert Ereignisse ereignisgesteuert, was den Overhead durch periodische Ticks minimiert und die Energieeffizienz steigert.

Entwickler können so Zephyr für Systeme mit extrem niedriger Taktfrequenz anpassen – ideal für energiearme, langlaufende Anwendungen.

```
[344/344] Linking C executable zephyr/zephyr.elf
Memory region         Used Size  Region Size  %age Used
           FLASH:      184904 B         1 MB     17.63%
             RAM:      112024 B       128 KB     85.47%
             CCM:          0 GB        64 KB      0.00%
        IDT_LIST:          0 GB         2 KB      0.00%
```

Bild 4: Beispiel aus einem Build-Vorgang

„West“ erleichtert Konfigurations und Debugging

Das Meta-Tool „West“ erleichtert Entwicklern den gesamten Prozess der Konfiguration, Erstellung und Fehleranalyse in Zephyr. Das Tool ermöglicht nicht nur die Verwaltung der Quellcodes, sondern unterstützt auch beim Erstellen komplexer Systemkonfigurationen und beim Debuggen.

Geräteunterstützung und Wiederverwendbarkeit durch Device Trees

Zephyr setzt auf ein durchdachtes Device-Tree-System, das eine hohe Wiederverwendbarkeit von Hardwarebeschreibungen ermöglicht. Entwickler können ihr System so an unterschiedliche Mikrocontroller und Boards anpassen, ohne jedes Mal eine neue Konfiguration schreiben zu müssen.

Fazit: Zuverlässig und für viele Embedded-Projekte skalierbar

Zephyr ist mehr als eine abgespeckte Linux-Variante: Es ist ein modernes, eigenständiges Echtzeitbetriebssystem, das für kleine Embedded-Systeme ausgelegt ist und Entwicklern Flexibilität, Effizienz und Anpassungsfähigkeit bietet. Das RTOS stellt somit eine attraktive Wahl für Entwickler dar, die nach einer Open-Source-Lösung suchen, die zuverlässig und für viele Embedded-Projekte skalierbar ist.

Im MicroConsult Zephyr-Training lernen Sie, wie Sie für den Aufbau eines Zephyr-Targets vorgehen und was Sie dazu benötigen. Der Aufbau und die Funktionsweise eines Zephyr-Systems mit harten Echtzeiteigenschaften stehen im Mittelpunkt. Im Zephyr-Training wird nur frei zugängliche Open-Source-Software eingesetzt

Weiterführende Informationen

1. MicroConsult Training: Zephyr – der kleine Bruder vom Tux
2. MicroConsult Trainings: Embedded- und Echtzeit-Betriebssysteme
3. MicroConsult Fachwissen: Embedded- und Echtzeit-Entwicklung
4. MicroConsult Training & Coaching: Embedded SW-Entwicklung
5. Alle Trainings & Termine auf einen Blick

Der MicroConsult-Newsletter

Wir informieren Sie mehrmals jährlich über Trends und Best Practices im Embedded Systems Engineering.

Erhalten Sie wertvolles Fachwissen und Tipps aus erster Hand von unseren Embedded-Experten!

Jetzt abonnieren!