

Hallo Welt!

Übungen zur Vorlesung

Entwicklungsumgebung

Tobias Klaus, Florian Schmaus, Peter Wägemann

Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
Lehrstuhl für Informatik 4 (Verteilte Systeme und Betriebssysteme)
<https://www4.cs.fau.de>

24.10.2016

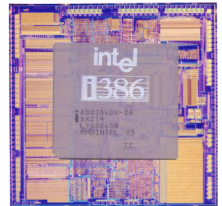


1 Einführung in eCos

2 Entwicklungsumgebung

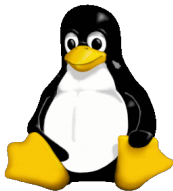


Prozessorvielfalt in der Echtzeitwelt



free **RTOS**

μClinux



eCos is an embedded, highly configurable, open-source, royalty-free, real-time operating system.

- Ursprünglich von der Fa. Cygnus Solutions entwickelt (1997)
- Primäres Entwurfsziel:
 - „deeply embedded systems“
 - „high-volume application“
 - „consumer electronics, telecommunications, automotive, . . .“
- Zusammenarbeit mit Redhat (1999)
- Seit 2002 quelloffen (GPL)



<http://ecos.sourceforge.org>

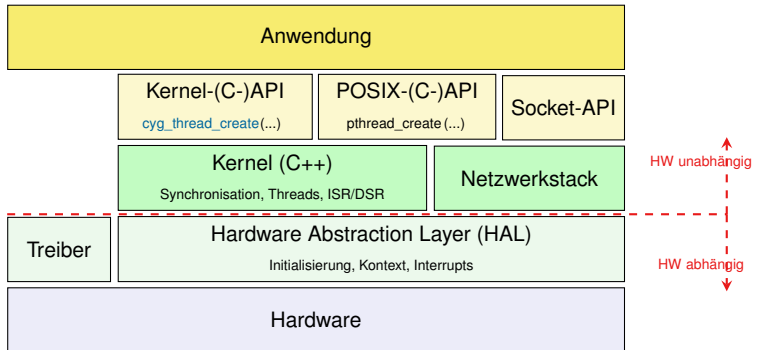


Unterstützte Plattformen

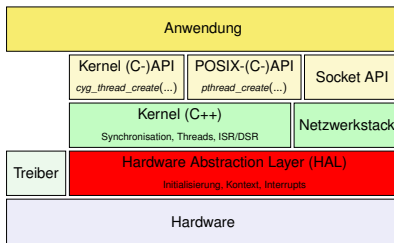
<http://www.ecoscentric.com/ecos/examples.shtml>

- Fujitsu SPARClike
- Matsushita MN10300
- Motorola PowerPC
- Advanced RISC Machines (ARM)
- Toshiba TX39
- Infineon TriCore
- Hitachi SH3
- NEC VR4300
- MB8683X
- *ARM Cortex*
- *Intel x86*
- ...

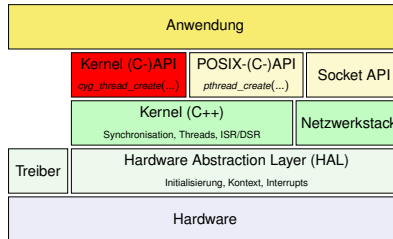




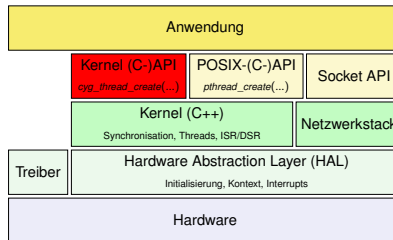
- Abstrahiert CPU- und plattformspezifische Eigenschaften
 - Kontextwechsel
 - Interruptverwaltung
 - CPU-Erkennung, Startup
 - Zeitgeber, I/O-Registerzugriffe



- Implementiert in C++
- Feingranular konfigurierbar
 - Verschiedene Schedulingstrategien (Bitmap/Multilevel Queue)
 - Zeitscheibenbasiert, präemptiv, prioritätenbasiert
- Verschiedene Synchronisationsstrategien
 - Mutexe, Semaphore, Bedingungsvariablen
 - Messages Boxes



- Interrupt Service Routine (ISR)
 - Unverzögliche Ausführung
 - Asynchron
 - Kann DSR anfordern
- Deferred Service Routine (DSR)
 - Verzögerte Ausführung (beim Verlassen des Kernels)
 - Synchron

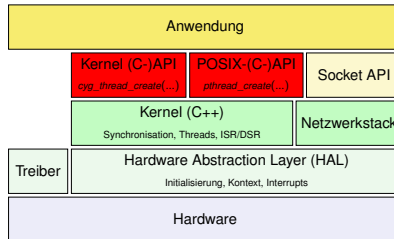


■ Kernel API

- vollständige C-Schnittstelle
- siehe Dokumentation¹

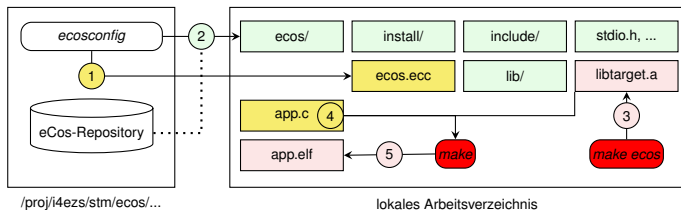
■ (Optionale) POSIX-Kompatibilitätsschicht

- Scheduling-Konfiguration, *pthread_**
- Timer, Semaphore, Message Queues, Signale, ...



¹<http://ecos.sourceware.org/docs-2.0/ref/ecos-ref.html>

1. Erstellen einer Konfiguration (configtool/ecosconfig)
2. Kopieren ausgewählter Komponenten (configtool/ecosconfig)
3. Erstellen einer Betriebssystem**bibliothek**
4. Entwicklung der eigentlichen Anwendung
5. Kompilieren des Gesamtsystems



Wichtig!

Für jede Übung wird eine Konfiguration vorgegeben (Schritte 1–3)



1. vectors.S

- Hardwareinitialisierung
- Globale Konstruktoren

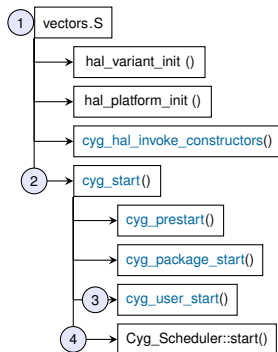
2. cyg_start():

- Hardwareunabhängige Vorbereitungen

3. cyg_user_start():

- Einsprungpunkt für Anwendungscode!
- Erzeugen von Threads

4. Starten des Schedulers

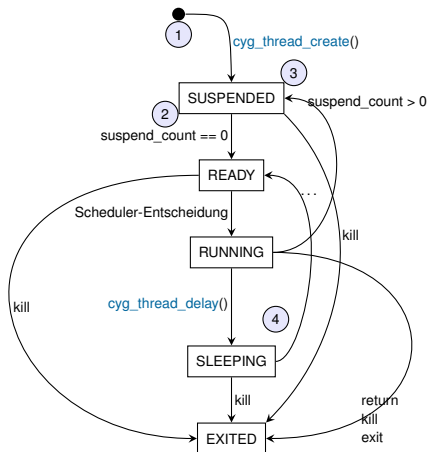


Wichtig!

In allen Übungsaufgaben muss man `cyg_user_start()` implementieren und dort alle Threads anlegen.
Die Funktion muss zurückkehren!



1. Thread wird im Zustand *suspended* erzeugt.
 - `suspend_count = 1`
2. `cyg_thread_resume()` aktiviert
 - `suspend_count--`
3. `cyg_thread_suspend()` suspendiert
 - `suspend_count++`
4. bereit
5. delay, mutex, semaphore wait
6. Threadterminierung



```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info ,
    cyg_thread_entry_t* entry ,
    cyg_addrword_t entry_data ,
    char* name,
    void* stack_base ,
    cyg_ucount32 stack_size ,
    cyg_handle_t* handle ,
    cyg_thread* thread
);
```

- Einbinden der nötigen Headerdatei
- Anlegen eines neuen eCos-Threads

Ausführliche Dokumentation

<http://ecos.sourceware.org/docs-latest/ref/kernel-thread-create.html>



```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info ,
    cyg_thread_entry_t* entry ,
    cyg_addrword_t entry_data ,
    char* name,
    void* stack_base ,
    cyg_ucount32 stack_size ,
    cyg_handle_t* handle ,
    cyg_thread* thread
);
```

- Scheduling-Informationen
- z. B. MLQ-Scheduler
 - Threadpriorität
 - Datentyp `cyg_uint8`

Ausführliche Dokumentation

<http://ecos.sourceware.org/docs-latest/ref/kernel-thread-create.html>




```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info ,
    cyg_thread_entry_t* entry ,
    cyg_addrword_t entry_data ,
    char* name,
    void* stack_base ,
    cyg_ucount32 stack_size ,
    cyg_handle_t* handle ,
    cyg_thread* thread
);
```

- Thread Einsprungpunkt
- Funktionszeiger
- Signatur:
`void (*)(cyg_addrword_t)`

Ausführliche Dokumentation

<http://ecos.sourceware.org/docs-latest/ref/kernel-thread-create.html>



```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info ,
    cyg_thread_entry_t* entry ,
    cyg_addrword_t entry_data ,
    char* name,
    void* stack_base ,
    cyg_ucount32 stack_size ,
    cyg_handle_t* handle ,
    cyg_thread* thread
);
```

- Thread Parameter
- Beliebige Übergabeparameter
 - z. B. Zeiger auf threadlokale Daten

Ausführliche Dokumentation

<http://ecos.sourceware.org/docs-latest/ref/kernel-thread-create.html>



```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info ,
    cyg_thread_entry_t* entry ,
    cyg_addrword_t entry_data ,
    char* name,
    void* stack_base ,
    cyg_ucount32 stack_size ,
    cyg_handle_t* handle ,
    cyg_thread* thread
);
```

- Beliebiger Threadname
- Tipp: (gdb) info threads

Ausführliche Dokumentation

<http://ecos.sourceware.org/docs-latest/ref/kernel-thread-create.html>



```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info ,
    cyg_thread_entry_t* entry ,
    cyg_addrword_t entry_data ,
    char* name,
    void* stack_base ,
    cyg_ucount32 stack_size ,
    cyg_handle_t* handle ,
    cyg_thread* thread
);
```

- Basisadresse des Threadstacks
(→ &stack[0])
- `cyg_uint8`-Array
- Global definieren
→ Datensegment!
- *Warum ist die notwendig?*

Ausführliche Dokumentation

<http://ecos.sourceware.org/docs-latest/ref/kernel-thread-create.html>



```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info ,
    cyg_thread_entry_t* entry ,
    cyg_addrword_t entry_data ,
    char* name,
    void* stack_base ,
    cyg_ucount32 stack_size ,
    cyg_handle_t* handle ,
    cyg_thread* thread
);
```

■ Stackgröße in Bytes

Ausführliche Dokumentation

<http://ecos.sourceware.org/docs-latest/ref/kernel-thread-create.html>



```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info ,
    cyg_thread_entry_t* entry ,
    cyg_addrword_t entry_data ,
    char* name,
    void* stack_base ,
    cyg_ucount32 stack_size ,
    cyg_handle_t* handle ,
    cyg_thread* thread
);
```

■ Eindeutiger Identifikator

- zur “Steuerung” z. B.:

`cyg_thread_resume(handle)`

Ausführliche Dokumentation

<http://ecos.sourceware.org/docs-latest/ref/kernel-thread-create.html>



```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info ,
    cyg_thread_entry_t* entry ,
    cyg_addrword_t entry_data ,
    char* name,
    void* stack_base ,
    cyg_ucount32 stack_size ,
    cyg_handle_t* handle ,
    cyg_thread* thread
);
```

- Speicher für interne Fadeninformationen
 - Fadenzustand u. a. suspend_count
- Vermeidung dynamischer Speicherallokation im Kernel

Ausführliche Dokumentation

<http://ecos.sourceware.org/docs-latest/ref/kernel-thread-create.html>



```
#include <cyg/kernel/kapi.h>
void cyg_thread_create
(
    cyg_addrword_t sched_info ,
    cyg_thread_entry_t* entry ,
    cyg_addrword_t entry_data ,
    char* name,
    void* stack_base ,
    cyg_ucount32 stack_size ,
    cyg_handle_t* handle ,
    cyg_thread* thread
);
```

Ausführliche Dokumentation

<http://ecos.sourceware.org/docs-latest/ref/kernel-thread-create.html>



Wichtig!

Zu jeder Übungsaufgabe wird eine eCos-Konfiguration bereitgestellt. Makefiles werden mit „*cmake*“ generiert.

1. Vorgabe herunterladen, entpacken, Verzeichnis betreten
2. **Nötige Umgebungsvariablen setzen:** `source ecosenv.sh`
3. Eigene Quelldateien (nur) in `CMakeLists.txt` eintragen
4. build Verzeichnis betreten → out-of-source build²
5. Makefiles erzeugen: `cmake ..` (*cmake PUNKT PUNKT*)
6. Alles kompilieren: `make`
7. Flashen, ausführen, debuggen: `make flash`
~→ Flasher & Debugger: `make gdb` (später ausführlicher)

²www.cmake.org/Wiki/CMake_FAQ#Out-of-source_build_trees



```
1  #include <cyg/hal/hal_arch.h>
2  #include <cyg/kernel/kapi.h>
3  #include <stdio.h>
4  #define MY_PRIORITY      11
5  #define STACKSIZE        (CYGNUM_HAL_STACK_SIZE_MINIMUM+4096)
6  static cyg_uint8      my_stack[STACKSIZE];
7  static cyg_handle_t   my_handle;
8  static cyg_thread     my_thread;

9  static void my_entry(cyg_addrword_t data) {
10     int message = (int) data;
11     printf("Beginning execution: thread data is %d\n", message);
12     for (;;) {
13         ezs_printf("Hello World!\n"); // \n flushes output
14         ezs_delay_us(1000000); // Delay for 1000000 * 1us = 1 second
15     }
16 }

17 void cyg_user_start(void) {
18     ezs_printf("Entering cyg_user_start() function\n");
19     cyg_thread_create(MY_PRIORITY, &my_entry, 0, "thread 1",
20                      my_stack, STACKSIZE, &my_handle, &my_thread);
21     cyg_thread_resume(my_handle); }
```



1 Einführung in eCos

2 Entwicklungsumgebung



EZS-Wiki

<https://gitlab.cs.fau.de/ezs/ezs16/wikis/home>

- Umstellung auf neues Entwicklungsboard
 - ↪ Mögliche Probleme
- Nicht unterstützte Plattform
- 👉 Dokumentation im Wiki
- Erweiterung durch *alle Teilnehmer an EZS*
 - gitlab account notwendig
- Besondere Aufgaben
 - Anleitung „EZS-Board unter Windows“
 - Anleitung „EZS-Board unter macOS“
- 👉 Gutscheine für I4-Kaffeekarte

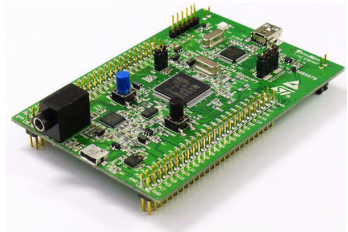


ARM Cortex-M4 Prozessor

- Flash-Speicher: 512 KB
- RAM: 128 KB

reichhaltige Peripherie

- Serielle Kommunikation
- Timer
- GPIOs
- ADCs
- 3-Achsen Gyroskop
- Beschleunigungssensor
- Audio Sensor
- *integrierte Debugging-Schnittstelle*
- ...





- Mini-USB-Kabel anschließen
- Nach Verbinden des USB-Kabels zwei serielle Ports
 - `/dev/ttyACM0`: Debugger
 - `/dev/ttyACM1`: serielle Kommunikation (Ausgabe z.B. mit cutecom)
- Ausleihbare Boards sind mit Blackmagic Firmware³ bereits ausgestattet

³<https://github.com/blackspere/blackmagic>

- Bashprompt %, gdb-Prompt: >

```
% cd <ezs-aufgabe1>
```

```
% source ecosenv.sh
```

```
% cd build
```

```
% cmake ..
```

```
% make
```

```
% arm-none-eabi-gdb -nh app.elf #Starten des Debuggers
```

```
> target extended-remote /dev/ttyACM0 #gdb ueber ttyACM0
```

```
> monitor swd #Verwendung von Serial Wire Debugging (SWD)
```

```
> attach 1 #Erstes Interface verwenden
```

```
> load #Laden des Systems in Flash (Flashen)
```

```
> continue #Starten der Ausfuehrung
```

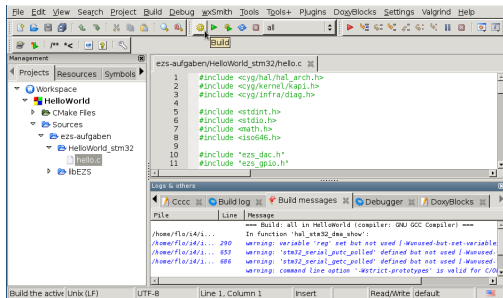
- gdb -nh: Verhindert Ausführung der Befehle aus ~/.gdbinit

- gdb-Befehle in Datei flash.gdb zusammenfassen und starten mittels:

```
arm-none-eabi-gdb --batch -x flash.gdb -nh app.elf
```



Editieren mit kate und codeblocks



- Anleitungen zum Editieren mit kate und codeblocks im Wiki
- Generieren der Projektdatenien
 - `cmake -G"Kate - Unix Makefiles" ..`
 - `cmake -G"CodeBlocks - Unix Makefiles" ..`
- Weitere Makefile-Targets
 - `make flash`, `make gdb`, `make edit`, ...
 - Mehr dazu $\leadsto$ Rechnerübung

