BETRIEBSSYSTEME

Lektion – MPIOSM, mpc555 und Eclipse

Eigenschaften mpc555	Interrupt-gesteuert Maximale Effizienz Ausnützung durch Power-Down Modi Nur ein Benutzer, aber viele verschiedene Prozesse/Tasks "gleichzeitig" Byteweise organisiert			
mapping	**memory mapping**	I/O als Teil des Registers	einfacher	
	I/O mapping	I/O separat	komplizierte Befehle	
Aufbau	mpc555 → MIOS1 (Multiple Input Output System) → I/O-Port MPIOSM mit 16bit			
MPIOSM Address Map (2 Adressen a 16bit)	**MPIOSMDR**	0x306100	Data Register	für Datenverkehr
	MPIOSMDDR	0x306102	Data Direction Register (steuert ob In- oder Output)	0 is Input (Standard) 1 is Output
Entwicklungsumgebung	Host = eclipse mit JAVA Crosscompiler = deep Target = mpc555	Rootklasse angeben:	#rootclasses = "basic/Aufgabe2";	

DDR manipulieren	short sh = US.GET2(MPIOSMDDR)M
	sh |= 0xe0;
	sh &= ~0x800;
	US.PUT2(MPIOSMDDR, sh);
			//hole aus Register (Bits lesen)
			//Register 5-7 als Outputs (1)
			//Register 11 als Input (0)
			//lade zurück ins Register (Bits setzen)	

Memory Zugriff	Über PUT und GET, da Java keine absolute Adressierung erlaubt. `import ch.ntb.inf.deep.unsafe.*;`			
Treiber / Driver	Verbindet ein Programm mit einer Low-level-Funktion und elektronischem Verhalten zum Zielsystem. Vereinfacht oder komplett versteckt die lower-level Details von der Maschine. Wiederverwertbar.			
Layers	Application Program Software / Driver Hardware	Application Programming Interface (API) Register Interface (DDR, DR) Physical Interface (MPIOSM)		
Beispielcode	`US.PUT1(0x306100,0x55)`	schreibt die Zahl 0x55 an die Adresse 0x306100		
	`US.PUT2(0x306100,0xAA55)` `(= US.PUT2(MPIOSMDR,0xAA55))`	schreibt 0xAA an die Adresse 0x306100 und 0x55 an die Adresse 0x306101		

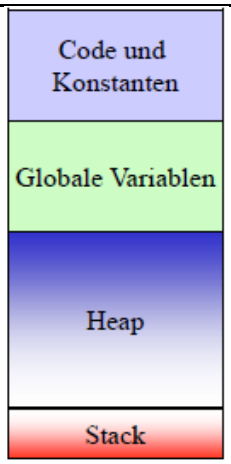
Code

```
import ch.ntb.inf.deep.runtime.mpc555.ntbMpc555HB;
import ch.ntb.inf.deep.unsafe.US;

public class Mpiosm implements ntbMpc555HB {

	public static void init(int channel, boolean out) {//Channel als In-/Output definieren
		short sh = US.GET2(MPIOSMDDR);
		if (out) sh |= (1 << channel); else sh &= ~(1 << channel);
		US.PUT2(MPIOSMDDR, sh);
	}

	public static boolean in(int channel) {		//für Abgrage ob Bit gesetzt (1) ist
		short sh = US.GET2(MPIOSMDR);
		return (sh & (1 << channel)) == (1 << channel);
	}

	public static void out(int channel, boolean val) { //Datenwert setzen (0 oder 1)
		short sh = US.GET2(MPIOSMDR);
		if (val) sh |= (1 << channel); else sh &= ~(1 << channel);
		US.PUT2(MPIOSMDR, sh);
	}
}
```

Internal Memory Map

0	Internal Flash 448kB	0x00 0000 CMF Flash A 256 Kbytes 0x04 0000 CMF Flash B 192 Kbytes 0x06 FFFF
0006'ffff		
		0x07 0000 Reserved for Flash (2.6 Mbytes –16 Kbytes) 0x2F BFFF
	I/O	0x2F C000 USIU & Flash Control 16 Kbytes 0x2F FFFF
		0x30 0000 UIMB Interface & IMB3 Modules (32 Kbytes) 0x30 7FFF
		0x30 8000 Reserved for IMB3 (480 Kbytes) 0x37 FFFF
		0x38 0000 SRAM Control A (8 bytes) 0x38 0008 SRAM Control B (8 bytes) 0x38 0010 Reserved (485.98 Kbytes)
003f'9800	Internal RAM 26kB	0x3F 9800 SRAM A (10 Kbytes) 0x3F C000 SRAM B (16 Kbytes) 0x3F FFFF
003f'ffff		
0080'0000	External RAM 2MB	
009f'ffff		
0100'0000	External Flash 4MB	
013f'ffff		
ffff'ffff		

Lektion – Endianess, Task

Endianess für 0x0055	**Big Endian**	wichtigstes zuerst	10:32:20	US.PUT2(DDR, 0x0055)
	Littel Endian	wichtigstes zuletzt	25.11.1989	US.PUT2(DDR, 0x5500)
Tasking	einfachstes Scheduling kooperatives Multitasking Unterscheiden von (ready tasks / periodische Tasks)			
Task	ist eine Klasse Methode **`action()`** zum starten Methode **`Task.install(task)`** für first use Kann nicht durch einen anderen Task unterbrochen werden (non-preemtive)			
Klassen (Typen)		final ... = 10 ... = 10 Alle Objekte die mit **new** aufgerufen werden Wird von HeapManager verwaltet JAVA hat eine automatische Garbage Collection Methodenaufruf: Variablen, PC, .. werden abgelegt. Methode fertig: obige löschen -> geht immer wieder auf 0.		
fig = new Figure();	1. Aus Konstantenbereich Classdescriptor (=Typdescriptor) holen 2. Grösse des Objekts auslesen 3. Entsprechende ANzahl Bytes allozieren (size +4) 4. Tag setzen			
Typkonversion	((Rectangle)fig).w = 1000;			
Typprüfung	if(fig instanceof Rectangle) ...			

Blinklicht - Ein Pin dauernd ein und ausschalten

```
import ch.ntb.inf.deep.runtime.mpc555.Task;

public class Blinker extends Task {
	static int count;
	int pin;

	public void action() {
		Mpiosm.out(pin, !Mpiosm.in(pin));
	}
	public static int getCount() {
		return count;
	}

	public Blinker(int pin, int time, int period) {
		this.pin = pin;
		this.time = time;
		this.period = period;
		Mpiosm.init(pin, true);
		Task.install(this);
		count++;
	}
	public Blinker(int pin, int period) {
		this(pin, 0, period);
	}
	public Blinker(int pin) {
		this(pin, 0, 1000);
	}
}
```

Buffer - Das Eingangssignal sofort auf das Ausgangssignal weiterleiten

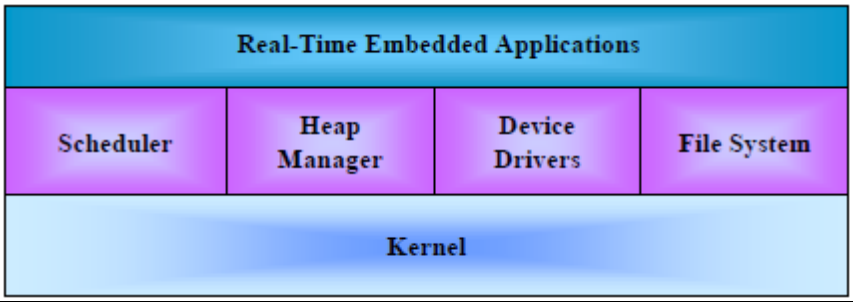
```
import ch.ntb.inf.deep.runtime.mpc555.Task;

public class Buffer extends Task {
	int in;
	int out;

	public void action () {
		Mpiosm.out(this.out, Mpiosm.in(this.in));
	}

	public Buffer (int in, int out) {
		this.in = in;
		this.out = out;
		Mpiosm.init(in, false);
		Mpiosm.init(out, true);
		this.period = 0;
		Task.install(this);
	}
}
```

Taskgeschwindigkeit - Zweiter Task misst Aufrufe des ersten Tasks

```
import ch.ntb.inf.deep.runtime.mpc555.Task;

public class TaskTest extends Task {
	boolean stop;
	public static int count, newCount, oldCount;
	static TaskTest t1, t2;

	public void action() {
		if (this.stop) {
			oldCount = newCount;
			newCount = t1.nofActivations;	//Number of Activations
			count = newCount - oldCount;
		}
	}

	public TaskTest(int period, boolean stop) {
		this.period = period;
		this.stop = stop;
		Task.install(this);
	}

	static {
		t1= new TaskTest(0, false);
		t2= new TaskTest(10000, true);
	}
}
```

Lektion – RTOS (Real Time Operating System)

Aufbau



Master	**Single-Master**	
	Multiple-Master	PPC ist Multimasterfähig

Aufgaben des Kernels

enable internal flash
stack pointer
boot
endless loop
time()
enable/disable interrupts
debug support
FCS (code integrity)

Exceptions vs Interrupts	**Exception**	Change normal program flow	z.B. Reset, System Call, Bus Errors, Debugger
	Interrups	One Type of Exceptions	**extern**: interrupts pint, IMB-Module
		From input pins or devices	**intern**: Internal devices (Timebase, PIT, RTC)

Exception-Beispiele und deren Ursache	Software Emulation Exception	Datenbusprobleme, Fehler im Speicher
	Program Exception	Reference to null-Pointer
	Machine Check	Schreiben oder Lesen von Adresse, ohne Terminierung des Bustransfer
	System Call (sc)	Maschineninstruktion
	Decrementer	Überlauf des Decrementers

Interrupt System

Schwierigkeit: Alle Interrupts springen in n+0x500
-> USIU (Universial System Interface Unit) Interrupt Controller

aktuelle Zeit auslesen

mpc555.Kernel.time();

Counter als Zeitbasis

64bit -> Überlauf nach 584'942 Jahre
da CPU und Datenbus 32bit haben, werden diese auf 2 Registern aufgeteilt.
TBUread(TimeBaseUpper) und TBLread (TimeBaseLower)

Exceptions Classes	**Ordered**	kommt wieder zurück no program state is lost	Alle Ausser:
	Unordered	information can be lost	Reset, Machine check, Non-maskable breakpoints
	Synchrounous	bewirkt durch Instruktionen precise (definierter Zustand)	
	Asynchronous	nicht durch Instruktion bewirkt nicht synchronisiert mit internen Prozessor Events	

Exception Processing	CIA->SRR0	lade diese/**nächste** Instruktion in Register 0
	MSR->SRRI	speichere Machine state in Register 0
	exception Vector->CIA	Exception Vector weiss wo weitergefahren werden muss
	0->MSR	

Register (alle 32bit)	ECR = Exception Cause Register	Register in der die Exception eingetragen wird
	SSR = save/restore registers	Zum speichern und wiederherstellen des Machinenzustands
	MSR = Machine state Register	Maschinenzustand

Code um 32Bit Einheiten zu einer 64Bit Long zusammensetzen (für Counter)

```
public static long time() {
   int high1, high2, low;
   do {
      high1 = US.GETSPR(TBUread);
      low = US.GETSPR(TbLread);
      high2 = US.GETSPR(TBUread);     //falls ein Überlauf von low auf high stattfindet
   } while (high1 != high2);          //dies wird hiermit überprüft
   long time = ((long)high1 << 32) | ((long)low & 0xffffffffL);     //zusammensetzen
   return time;
}
```

Lektion - Graphikdisplay an Systembus

Chip-Selects (CS)	Selektieren einer bestimmten Adresse	
Wait States	Für eine gewisse Wartezeit	
Anz. Zyklen	= 2 + Anzahl Wait States	
Register	BR0-BR3	Memory Controller Base Registers
	OR0-OR3	Memory Controller Option Registers
Displays	**Character Display**	2*8Zeichen
	Graphic Display	128*128 Punkte; ansteuern mit Kontroller LC7981
Kontroller LC7981	Register Select Signal (RS)	1=Instruktion / 0=Daten
	Read/Write (R/W')	0=US.PUT / 1=US.GET
	US.PUT1(0x2000000,0x32)	0x32 wird als Datum interpretiert
	data = US.GET1(0x2000000)	Daten werden gelesen
	US.PUT1(0x2000001,0)	0 wird als Instruktion interpretiert
	status = US.GET1(0x2000001)	Busy Flag wird gesetzt

Code - Display ansteuern

```
import ch.ntb.inf.deep.unsafe.US;

public class DisplayV1 {
   // display area
   public static final int bottom = 0;
   public static final int height = 128;
   public static final int left = 0;
   public static final int width = 128;
   static final int top = height - 1;
   static final int right = width - 1;
   static final int Hn  =    16;
   // colors
   public static final int black = 0;
   public static final int white = 15;
   // drawing modes
   public static final int replace = 0;
   public static final int paint = 1;
   public static final int invert = 2;
   // display at 32MB
   static final int Base = 0x2000000;
   static final int BR2 = 0x2FC110;      //Base Register
   static final int OR2 = 0x2FC114;      //Option Register

   static void waitForReady () {// check busy flag after each instruction, not after data write
      byte state;
      do state = US.GET1(Base+1); while ((state & 0x80) != 0); // busy flag
   }
      static void putInstr (byte instr) {
      waitForReady();
      US.PUT1(Base + 1, instr);
   }
      static void putData (byte val) {
      US.PUT1(Base, val);
   }

   static void put (byte instr, byte val) {
      putInstr(instr);
      putData(val);
   }

   public static void initCS () {  //CS = Chip Select
      US.PUT4(BR2, Base / 0x8000 * 0x8000 + 0x403);
      // base address: 2000000H, 8 bit port, RW, internal transfer ack, no bursts

      US.PUT4(OR2, 0xFFFF8611);
      // address mask: FFFF8000, ACS=11, timing relaxed, 5 cycles
   }

   public static void initGraphMode () {
```

```
      put((byte)0, (byte)0x32);    // mode control: display on, master, graphic mode
      put((byte)1, (byte)7); // pitch: Hp = 8, 1Byte = 8dots
      put((byte)2, (byte)(Hn - 1));      // set number of characters : Hn = 16
      put((byte)3, (byte)127);     // set number of time divisions: Nx = 128
      put((byte)8, (byte)0); // set display start low order address: 0
      put((byte)9, (byte)0); // set display start high order address: 0
      put((byte)0xA, (byte)0);     // set cursor address (low order): 0
      put((byte)0xB, (byte)0);     // set cursor address (high order):0
      for (int i = 1; i <= 2048; i++) put((byte)0x0C, (byte)0);      // clear display mem
   }

   public static void setDot (int col, int x, int y, int mode) {
      int addr = (top-y) * Hn + x / 8;
      put((byte)0xA, (byte)(addr));      // set cursor address (low order)
      put((byte)0xB, (byte)(addr >> 8)); // set cursor address (high order)
      switch (mode) {
      case replace:
         if (col == 0) put((byte)0xF, (byte)(x % 8));     // set bit
         else put((byte)0xE, (byte)(x % 8));   // clear bit
         break;
      default:
         break;
      }
   }
```

Code - Display Test

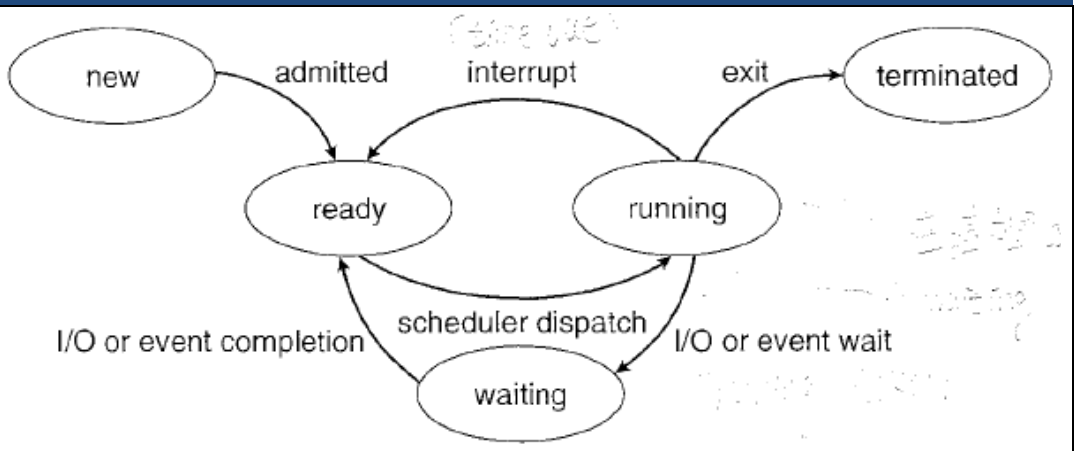
```
import ch.ntb.inf.deep.runtime.mpc555.Task;

public class FernV1 extends Task {
      static final int fac = 0x020000, fac2 = 0x400;
      static final int e =  8, a1 =  0, a2 =  111411, a3 =  26214, a4 =  -19660;
      static final int b1 =  0, b2 =  5242, b3 =  -34078, b4 =  36700;
      static final int c1 =  0, c2 =  -5242, c3 =  30146, c4 =  34078;
      static final int d1 =  20971, d2 =  111411, d3 =  28835, d4 =  31457;
      static final int e1 =  0, e2 =  0,e3 =  0, e4 =  0;
      static final int f1 =  0, f2 =  214748368, f3 =  214748368, f4 =  59055800;
      static final int p1 =  1310, p2 =  111411, p3 =  9175, p4 =  9175;
      static int z, X, Y, x0, y0;

      public void  action() {
            final int a = 33, e = 12;
            int xi, eta, rn, x, y, p1PlusP2, p1PlusP2PlusP3;
            p1PlusP2 = p1 + p2;    p1PlusP2PlusP3 = p1PlusP2 + p3;
            rn = z;
            rn =  (rn + 1) * a % fac;
            if (rn < p1) {x = a1 * X + b1 * Y + e1; y = c1 * X + d1 * Y + f1;}
            else if (rn < p1PlusP2) {x = a2 *X + b2 * Y + e2; y= c2 * X + d2 * Y + f2;}
            else if (rn < p1PlusP2PlusP3) {x = a3 * X + b3 * Y + e3; y = c3 * X + d3 * Y + f3;}
            else {x = a4 * X + b4 * Y + e4; y = c4 * X + d4 * Y + f4;}
            xi = x0 + x / fac2 * e / fac;
            eta = y0 + y / fac2 * e / fac;
            X = x / fac; Y = y / fac;
            DisplayV1.setDot(DisplayV1.black, xi, eta, DisplayV1.replace);
            z= rn;
      }

      static {
            X =  0; Y =  0; x0 =  64; y0 =  0; z = fac;
            DisplayV1.initCS();
            DisplayV1.initGraphMode();
            DisplayV1.line(DisplayV1.black, 17, 120, 33, 120, DisplayV1.replace);
            Task t = new FernV1(); t.period = 10; Task.install(t);
      }
}
```

Lektion - Graphikdisplay V2

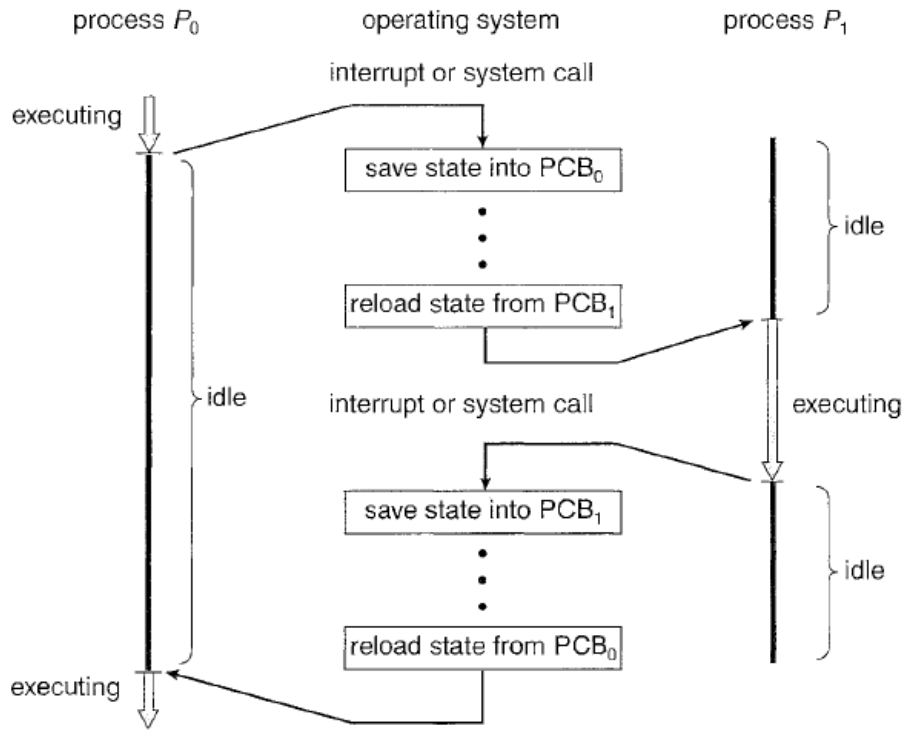
Eigenschaften	Realtime capable Verarbeitung über byte-Array[2048] im RAM (=128*128Bit)
Handler	schreibe 1 Zeile pro Task, der jede ms läuft (=128ms) entspricht 8Hz display frame rate pro Zeile: setze Cursor am Anfang hin und schreibe 16 bytes (automatisches inkrement)
Neu ist in V2	dass die Punkte in den Buffer geschrieben werden, und durch den Update-Task zum Display geschickt.

Lektion - Prozess, Scheduling

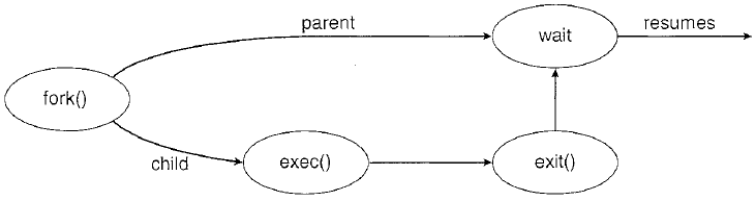
Prozessstatus & -ablauf



Prozesswechsel



fork()
(create new process)



Aufgaben & Lösungen

gemeinsamer Speicher zwischen child und parent	Shared memory segments
3xfork()	= 8 Prozesse erzeugt
child und parent verwenden gleiche Variablen	Übung 3.13
Anzahl Möglichkeiten von Scheduling bei n Prozessen	n!

Codebeispiel Prozesse

```
#include <sysltypes.h>
#include <stdio. h>
#include <unistd.h>
int main()
{
pid_t pid pid1;
   /* fork a child process */
   pid = fork();
   if (pid < 0) { /* error occurred */
      fprintf(stderr, "Fork Failed");
      return 1;
   }
   else if (pid == 0) { /* child process */
      pid1 = getpid();
      printf("child: pid = %d",pid); /* A */     0
      printf("child: pid1 = %d",pid1); /* B */  2603
   }
   else { /* parent process */
      pid1 = getpid() ;
      printf("parent: pid = %d",pid); /* C */     2603
      printf("parent: pid1 = %d" ,pid1); /* D */ 2600
      wait(NULL);
   }
   return 0;
}
```

Scheduling

=Welcher Prozess kommt dran

Scheduling-Prinzipien

First In First Serve
Priority Based
Round Robbin (kleine Zeitscheiben) [alle kommen dran, hoher Aufwand]

Linux-Einführung

BASH (born again shell)

Ausführbare Codedatei

Makefile

Name der Datei: "Makefile"
Einrückungen sind obligatorisch
Existiert einmal in jedem Projekt

Opertionen in C

Unär (1 Operand)	Binär (2 Operanden)	Ternär (3 Operanden)
++ -- ~(Einerkomplement) * (Dereferenzierung) & (Adressoperator)	+ - / * > < <= => && (logisches AND) & (bitweises AND)	Bedingung ? Then : Else

Pointer = Zeiger auf Adresse

Java	Referenzen	Operationen werden geprüft
C	Pointer	Keine Prüfung, alles erlaubt

Pointer-Verwendung

*sp += 1;	Variable wird erhöht
sp += 1;	Adresse wird erhöht
&sp;	Adresse vom Zeiger

void-Pointer

falls Datentyp unbekannt
falls universeller Datentyp

Sichtbarkeiten in C

Global	im ganzen Programm, auch in anderen Modulen
Modulglobal	innerhalbt des Moduls(File)
Innerhalb eines Blocks	Codeteil umschlossen von {...}

Namenswahl

mit Präfix stack_element; void stack_add(...)

Schlüsselwort static

nur in diesem Modul sichtbar (private in Java)
Wert nach beenden des Moduls wird behalten

Linux C-Programmierung

Beim Programmstart	Von grossem Memory (Disk) in Speicher (RAM)

Code - Testfile

Test.c

```
//#include "function.c"
int main(void) {
   greet();
   return 0;
}
```

function.c

```
#include <stdio.h>
#include "function.h"
int greet(void) {
   printf("%s!\n", GREETING);
   return 0;
}
```

function.h

```
#define GREETING "Hello World"
```

Makefile

```
# System setup. Stuff like compiler name and system files
CC=gcc
DEPENDFILE = .depend

# Enter the application name here...
EXEC_NAME = test
# Add your sourcefiles here...
SRC =  test.c \
   function.c

# Simple suffix conversion from *.c to *.o
OBJ = $(SRC:%.c=%.o)

# This target creates dependencies in order to make sure the
# modules are rebuilt after modification.
$(DEPENDFILE) : $(SRC)
   @echo Create dependencies...
   $(CC) -MM $(SRC) > $(DEPENDFILE)
# This target creates the executable from all the given object
# files. Please note the dependecies to the dep target and the
# object files. This makes sure, that all the dependencies are
# respected.
all: $(DEPENDFILE) $(OBJ)
   @echo Build target application
   $(CC) $(OBJ) -o $(EXEC_NAME)
# This target removes all generated files from the system.
clean:
   @echo Remove generated files...
   rm -f $(EXEC_NAME)
   rm -f $(OBJ)
   rm -f $(DEPENDFILE)

-include $(DEPENDFILE)
```

Code - Memory löschen - void Pointer

Variante mit char

```
void zero_memory(char *buf, size_t len) {
   int pos = 0;
   for (pos = 0;pos < len;pos++) buf[pos] = 0;
}
```

Variante mit void-Pointer

```
void zero_memory(void *buf, size_t len) {
   char *char_buf = (char *)buf;
   int pos = 0;
   for (pos = 0;pos < len;pos++) char_buf[pos] = 0;
}
```

Process Management

Prozesswurzel

Knoten = "init"; startet alle anderen Prozesse; PID = 1

pSleep10.c
Wartet 10 Sekunden und terminiert dann

```
#include <unistd.h>
int main() {
   sleep(10);
   return 0;
}
```

pExec.c

```
#include <sys/types.h>
#include <wait.h>
#include <unistd.h>
#include <stdio.h>
int main() {
   pid_t pid;
   pid = fork();
   if(pid == 0) {     /*Child Prozess*/
      printf("parent pid = %d\n",getppid());
      printf("child pid = %d\n",getpid());
      execlp("/bin/ps", "ps", "-el", NULL);
      // program can be exchanged only once, this code no
      // longer exists after this line!
   }
   sleep(10);
   return 0;
}
```

pExit.c

```
#include <sys/types.h>
#include <wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
void exitHandler1(){
   printf("exit handler 1 of pid: %i\n", getpid());
}
void exitHandler2(){
   printf("exit handler 2 of pid: %i\n", getpid());
}
int main() {
   pid_t pid;
   atexit(exitHandler1);   //Registriere Exit-Handler 1
   atexit(exitHandler2);   //Registriere Exit-Handler 2
   pid = fork();
   if(pid == 0) {
      printf("parent pid = %d\n",getppid());
      printf("child pid = %d\n",getpid());
   }
   return(0);
   //Danach werden Handler in umgekehrter Reihenfolge aufgerufen
}
```

pSignal.c

```
static void sigHandlerChild(int sigNr){
   printf("child with pid = %i received signal, sigNr =
%i\n",getpid(),sigNr);
   exit(0);
}
int main() {
   pid_t pid;  int state;  printf("SIGINT = %i\n",SIGINT);
   pid = fork();
   if(pid == 0) {
      signal(SIGINT, &sigHandlerChild);
      while(1) {
         printf(".\n");     sleep(1);
      }
   }
   sleep(5);   kill(pid, SIGINT);     sleep(5);
   pid = wait(&state);
   printf("parent with pid = %i has terminated\n",getpid());
   return 0;
}
```

Lektion - Memory Management

Paging	um Daten zu schreiben und lesen vom sekundären Speicher für den Gebrauch im Hauptspeicher
Swapping	moving from/to secondary storage a whole program at a time Durch die Möglichkeit des Swappings wird nun Speicher aus dem RAM auf die Festplatte ausgelagert um das System weiter betreiben zu können. langsamer (da höhere Latzenz und die geringere Datenrate beim Zugriff auf Swap-Bereich)
MMU	Die MMU kann den unveränderlichen Teil der gemeinsam verwendeten Bibliothek in mehreren Prozessen einblenden. Der physikalische Speicherbedarf ist dadurch minimiert. Für die Prozesse sieht es jedoch so aus als hätte jeder seine eigene Kopie der Bibliothek.
externe Fragmentierung	Wenn kein Speicher mehr am Stück verfügbar, wird extern Fragmentiert
interne Fragmentierung	Allozieren von gleichgrossen Blöcken führt zu interner Fragmentierung

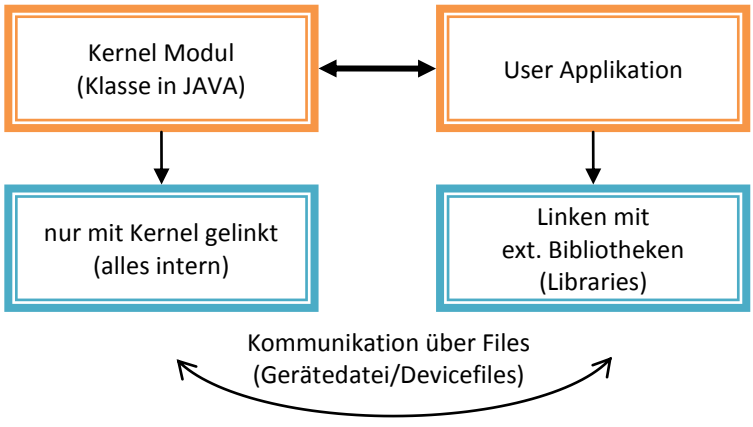
-13. Lektion– Linux Kernel & Treiberentwicklung

Gerätedatei	befinden sich im Verzeichnis /dev/	
Geräteklassen	Zeichengeräte (char devices)	c
	Blockgeräte (block devices)	b
	Netzwerk-Schnittstellen (networking devices)	s
Majornummer	gibt an, in welchem Treiber die notwendige Funktionen für die Ansteuerung eines Geräts implementiert worden sind.	
Minornummer	gibt an, welche Funktionen für das entsprechende Gerät zuständig sind.	

random.c
hole 10 Zufallszahlen
gib sie auf der Console aus

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>

#define MaxCount 1000

int main(void) {
int i;
int buffer;
int dev = open("/dev/random", O_RDONLY);
if(dev != -1){
  for(i = 0; i < MaxCount; i++) {
    read(dev, &buffer, 4);
    printf("%i\n", buffer);
  }
  close(dev);
  return EXIT_SUCCESS;
}
printf("Unable to open device...");
return EXIT_SUCCESS;
}
```

Übersicht



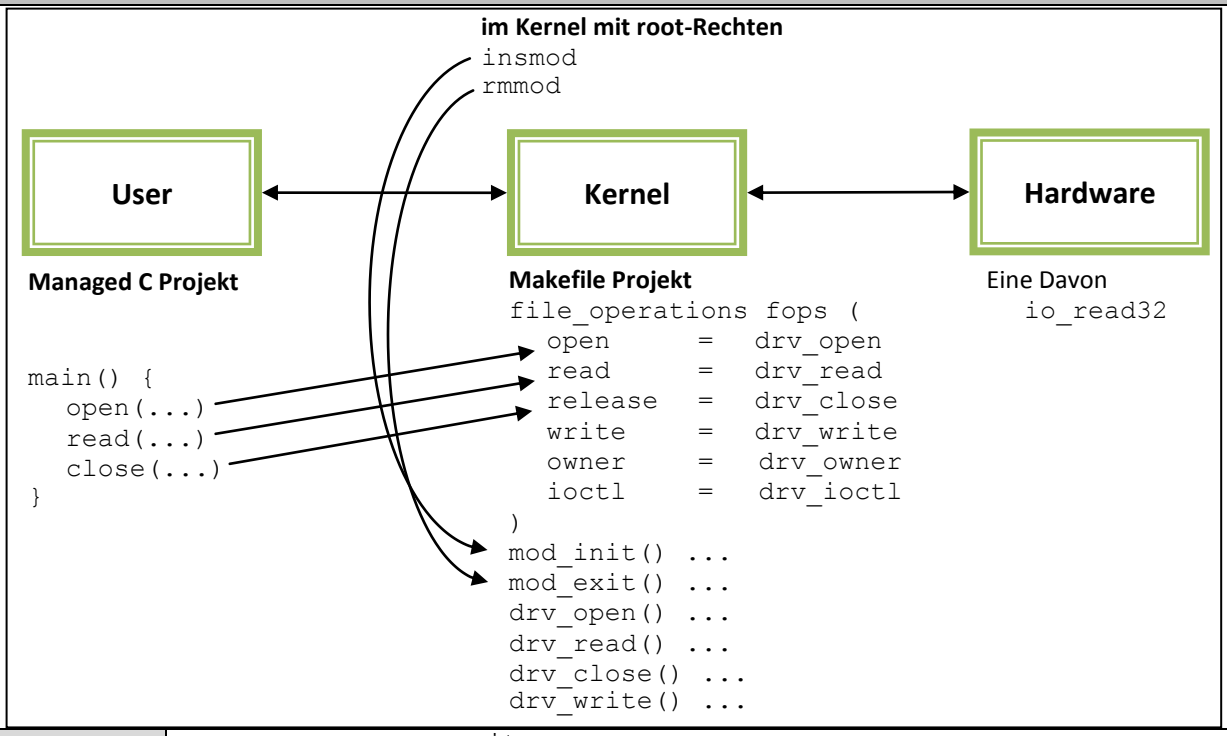
Supervisor Mode = Kernel Mode

Module

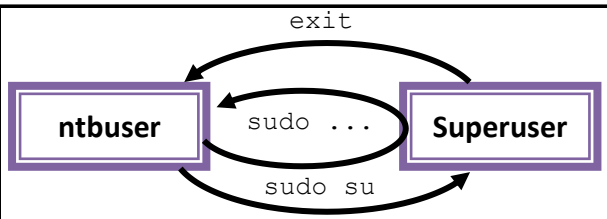
- erweitern Kernel um eine bestimmte Fähigkeit
- jedes Modul kann einzeln übersetzt werden
  example.c → example.ko
- kann zur Laufzeit laden, entladen werden

Kernelmodul

Übersicht



Berechtigungen



hello World.c

```c
#include <linux/init.h>
#include <linux/module.h>

MODULE_AUTHOR("martin.zueger@ntb.ch");
MODULE_DESCRIPTION("Hello World module");
MODULE_SUPPORTED_DEVICE("none");
MODULE_LICENSE("GPL");

static int hello_init(void) {
  printk(KERN_ALERT "Hello, world!\n");
  return 0;
}

static void hello_exit(void) {
  printk(KERN_ALERT "Goodbye, cruel world!\n");
}

module_init(hello_init);     //init-Funktion bekanntgeben
module_exit(hello_exit);     //exit-Funktion bekanntgeben

/*
 * Kernelmodul laden, wieder entladen und Ausgaben anzeigen:
 * ntbuser@kdev-vm:~$ sudo insmod helloMod.ko
 * ntbuser@kdev-vm:~$ sudo rmmod helloMod.ko
 * ntbuser@kdev-vm:~$ dmesg | tail
 */
```

Vorgehen

1. Gerätenummern alloziieren (Identifier der Geräte)
2. Zeichengerät registrieren
3. Device Node erstellen (File mit Info's für Kernel vom Gerät)

Treiberentwicklung

__init Funktion aufrufen, gibt bei Erfolg 0 zurück
Ressourcen vorgängig reservieren (Fehlerbehandlung)
3. __exit-Funktionaufrufen, alle reservierten Ressourcen freigeben

Gerätenummern allozieren - statisch oder dynamisch

```
/*************************************************************
 * File:        alloc.c
 * Author:      urs.graf@ntb.ch
 * Date:        2012-11-27
 * License:     GPL
 * Description: Module reserves device numbers
 *************************************************************/

#include <linux/init.h>
#include <linux/module.h>
#include <linux/types.h>
#include <linux/fs.h>
#include <linux/cdev.h>

/* Remove the following line for dynamic allocation: */
#define STATICDEVID

MODULE_AUTHOR("urs.graf@ntb.ch");
MODULE_DESCRIPTION("Dummy module");
MODULE_SUPPORTED_DEVICE("none");
MODULE_LICENSE("GPL");

static dev_t dev;
static unsigned int count = 2;
static char devName[] = "testDevice";
static unsigned int firstMinor = 0;
static unsigned int firstMajor = 240;
static int error;

static int __init init(void) {
  #ifdef STATICDEVID
  //statische Allozierung
  dev = MKDEV(firstMajor, firstMinor);
  error = register_chrdev_region(dev, count, devName);
  #else
  //dynamische Allozierung
  error = alloc_chrdev_region(&dev, firstMinor, count, devName);
  #endif

  if(error) goto devIDRegFailure;

  printk(KERN_ALERT "Driver successfully loaded!\n");
  printk(KERN_ALERT "-> Devicename: %s\n", devName);
  printk(KERN_ALERT "-> Majornumber: %d\n", MAJOR(dev));

  return 0;

  devIDRegFailure: printk(KERN_ALERT "Error while initializing module (%s)!\n",
devName);
  return error;
}

static void __exit cleanUp(void) {
  unregister_chrdev_region(dev, count);
  printk(KERN_ALERT "Driver unloaded!\n");
}

module_init(init);
module_exit(cleanUp);
```

Zeichengerät registrieren + Device Nodes erstellen

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/types.h>
#include <linux/fs.h>
#include <linux/cdev.h>
#include <linux/device.h>

MODULE_AUTHOR("urs.graf@ntb.ch");
MODULE_DESCRIPTION("Dummy module");
MODULE_SUPPORTED_DEVICE("none");
MODULE_LICENSE("GPL");

static dev_t dev;
static struct cdev *testDev;
static char devName[] = "testDevice";
static unsigned int firstMinor = 0;
static int error;
static struct class *my_class;

static struct file_operations testDevFops = {
    .owner = THIS_MODULE
};

static int __init init(void) {
    error = alloc_chrdev_region(&dev, firstMinor, 1, devName); //1. Allozieren
    if(error) goto devIDRegFailure;
    testDev = cdev_alloc();                                    //2. Zeichengerät registrieren
    testDev->owner = THIS_MODULE;                              //2. "
    testDev->ops = &testDevFops;                               //2. "
    error = cdev_add(testDev, dev, 1);                         //2. "
    if(error) goto devRegFailure;
    my_class = class_create(THIS_MODULE, "MyClass");
    device_create(my_class, NULL, dev, NULL, devName);    //3. Device Nodes erstellen


    printk(KERN_ALERT "Driver successfully loaded!\n");
    printk(KERN_ALERT "-> Devicename: %s\n", devName);
    printk(KERN_ALERT "-> Majornumber: %d\n", MAJOR(dev));

    return 0;

    devRegFailure: unregister_chrdev_region(dev, 1);
    devIDRegFailure: printk(KERN_ALERT "Error while initializing module (%s)!\n",
devName);
    return error;
}

static void __exit cleanUp(void) {
    device_destroy(my_class, dev);  // delete sysfs entries - Device Nodes entfernen
    class_destroy(my_class);         // "
    cdev_del(testDev);               // "
    unregister_chrdev_region(dev, 1);
    printk(KERN_ALERT "Driver unloaded!\n");
}

module_init(init);
module_exit(cleanUp);

/*
 * Kernelmodul laden, neue Gerätedatei wird angezeigt mit :
 * ntbuser@kdev-vm:~$ ls -l /dev/te*
 * crw------- 1 root root 249, 0 2012-11-30 10:48 /dev/testDevice
 * wenn rules in /etc/udev/rules.d korrekt gesetzt
 * crw-rw-rw- 1 ntbuser root 250, 0 2012-11-30 17:23 /dev/testDevice
 */
```

Datei-Operationen

```
...

static dev_t dev;
static struct cdev *testDev;
static char devName[] = "testDevice";
static unsigned int firstMinor = 0;
static int error;
static struct class *my_class;

static int open(struct inode *inop, struct file *filp){
    printk(KERN_ALERT "Device opened!\n");
    return 0;
}

static int close(struct inode *inop, struct file *filp){
    printk(KERN_ALERT "Device closed!\n");
    return 0;
}

static ssize_t read(struct file *filp, char __user *data, size_t size, loff_t *offs){
    int i;
    char text[] = "reading from device!\n";
//  for(i = 0; i < 11; i++){
//      if(put_user(text[i], data + i)) return i;
//  }
    i = copy_to_user(data, text, size);
    return i;
}

static ssize_t write(struct file *filp, const char __user *data, size_t size, loff_t
*offs){
    int i;
    char str[size + 1];
//  for(i = 0; i < size; i++){
//      if(get_user(s[i], data + i)) return i;
//  }
    i = copy_from_user(str, data, size);
    str[size] = 0;
    printk(KERN_ALERT "%s", str);
    return i;
}

static struct file_operations testDevFops = {
    .owner = THIS_MODULE,
    .open = open,
    .release = close,
    .read = read,
    .write = write
};

static int __init init(void) {
    ...
}

static void __exit cleanUp(void) {
    ...
}

module_init(init);
module_exit(cleanUp);
```

LINUX - BEFEHLE

Hilfe	man passwd	Hilfeseite (Manpage) des Befehls passwd aufrufen mit "q" quittieren; "/ begriff" zum suchen; "h" für Navigationshilfe
	man 5 passwd	Hilfeseite der Datei passwd
	apropos passwd	Listet von den ähnlichen Befehlen die Mangpage auf
	apropos change passwd	Listet alle Manpages auf die ein Wort enthält
	apropos change passwd -a	Listet alle Manpages auf die beide Worte enthalten
	pstree -help	Optionen anzeigen
Zugriffsrechte	sudo ...	Ein Befehl im Superuser-mode ausführen
	sudo su	gehe in superuser-mode
Verzeichnis & Files	mkdir myDir	Erstellt ein Ordner/Verzeichnis
	ls	Listete Ordner und Dateien auf
	cd myDir	Zu Verzweichnis wechseln
	cd ..	Ein Verzeichnis zurück
	cat > file1.txt Text	Eine neue Datei anlegen Text, beenden mit Ctrl+D
	cat file1.txt	Dateiinhalt anzeigen
	rm file1.txt	Datei löschen
	rmdir myDir	Verzeichnis löschen
	cp fil1 file2	Kopiere Datei
	mv file1 file2	Verschiebe Datei
	mkdir Uebung{0..4}{a..c}	erstellt 15 Verzeichnisse (Uebung0a bis Uebung4c)
	chmod 735 Ueb*	Zugriffsrechte festlegen
	find . -name '*a'	Suche Dateien mit Endung "a"
C-File	gcc test.c -o test ./test	C-File ausführen (-o = Optionen Objektfile test) Makefile ausführen (./ für aktuelles Verzeichnis)
Prozesse	ps ps -e ps -u ntbuser	aktuelle Prozesse auflisten alle Prozesse auflisten alle Prozesse eines Users
	pstree pstree -a pstree -p pstree 1572	Prozessbaum anzeigen bessere Darstellung, da Zeilen nicht abgeschnitten werden mit PID's nur Baum mit PID 1572
	gedit gedit &	Texteditor öffnen (Terminal wartet bis Editor geschlossen wird) Prozess als Child-Prozess starten, um Terminal weiter zu nutzen
	echo $$	Prozess ID der Shell ermitteln
	exit	BASH Terminal beenden
	sleep 5	5 Sekunden warten
	execlp("/bin/ps", "ps", "-el", NULL);	Programm ausführen (unterbricht die Ausführung im C-File)
Swapping	swapoff -a	Swapping ausschalten
	swapon -s swapon -a	Partitionen auflisten, die Swap-Bereiche nutzen Swapping einschalten
	cat /proc/1572/maps	Adressen/Biblitheken des Prozesses 1572 anzeigen r=read, w=write, x=execute, s=shared, p=private(copy on write)
	ps -p 1572 -o pid,comm, size,vsize	Speicherbedarf eines Prozesses size=im Speicher gebraucht; vsize=virtual size
Bibliothek	lsof | grep libc- | wc -l	Anzahl Verwendungen der Bibliothek libc
Kernel	file /dev/log	Typ einer Gerätedatei herausfinden
	ls -l ls –l /dev	Verzeichnisinhalt ausgeben (1 Zeichen = Typ der Gerätedatei) falls noch nicht ins /dev-Verzeichnis gewechselt
	mknod myTestDev c 240 0	Neue Gerätedatei anlegen (240 =Majornummer, 0=Minornummer)
	insmod example.ko	Modul laden (insert)
	rmmod example.ko	Modul entladen (remove)
	lsmod	Module auflisten (list)
	chmod	einem Modul Rechte vergeben
	dmesg	Kernel-log auslesen