Hochschule Niederrhein,
Fachbereich Elektrotechnik und Informatik,
Labor für Echtzeitsysteme

Praktikum Echtzeitsysteme

BI5

Jan Bolle, 1195475, Praktikumsgruppe F

Mateusz Kowalczyk, 1166709, Praktikumsgruppe F

Version 0.0

20.01.2020

Inhaltsverzeichnis

1 Beschreibung der Aufgabenstellung

Im Praktikum Echtzeitsysteme solle eine Software entwickelt werden, deren Aufgabe es ist ein Auto einer Carrerabahn zu steuern. Zu den Hauptanforderungen gehören:

- Erfassen der Fahrzeuggeschwindigkeit und Vermessen der Strecke (Segmenttypen, Länge)
- Vermeidung von Kollisionen an Gefahrenstellen (Kreuzungen), indem die Position des Gegnerfahrzeugs ausgewertet wird
- Funktion unabhängig vom Aufbau der Bahn, der verwendeten Spur oder Fahrzeug

Um die Fahrzeuge steuern zu können sind statt der üblichen Controller zwei Raspberry Pi – einer je Spur – an die Carrerabahn angeschlossen. Auf diesen wird die Anwendung ausgeführt. Dabei kann die Geschwindigkeit des Fahrzeugs gesetzt sowie die Bremse gezogen oder gelöst werden.

In die Carrerabahn sind Lichtschranken eingebaut, deren Daten vom Rechner aufbereitet und an das Programm weitergeleitet werden. Durch diese Lichtschranken lässt sich die Position des Fahrzeugs auf der Strecke sowie dessen Geschwindigkeit bestimmen. Außerdem reichert der Rechner diese Informationen mit *Segmenttypen* an, sodass die Applikation auf die aktuelle Situation reagieren kann.

Des weiteren finden sich über den Kurven Auslenkungssensoren, welche außerdem Daten an die Applikation liefern, falls das Fahrzeug in einer Kurve ausbricht.

2 Bedienung der Applikation

Die Applikation hat den Namen „race“ und wird über die Kommandozeile aufgerufen; hierbei können zwei Argumente übergeben werden:

```
./race [ [Geschwindigkeit] Rundenzahl ]
```

Geschwindigkeit bezeichnet den an den Controller übergebenen initialen Geschwindigkeitswert, *standardmäßig 0x24*.

Rundenzahl bezeichnet die Anzahl der zu fahrenden Runden, *standardmäßig 10*.

3 Generierung und Installation

Die Generierung der Applikation erfolgt auf den Raspberry Pis. Hierzu wird ein Makefile zur Verfügung gestellt, welche die vorliegenden C-Quellcodedateien kompiliert und die erforderlichen Bibliotheken *RT* (Realtime) und *pthread* (POSIX Threads) dazu linkt.

4 Lösung

4.1 Beschreibung

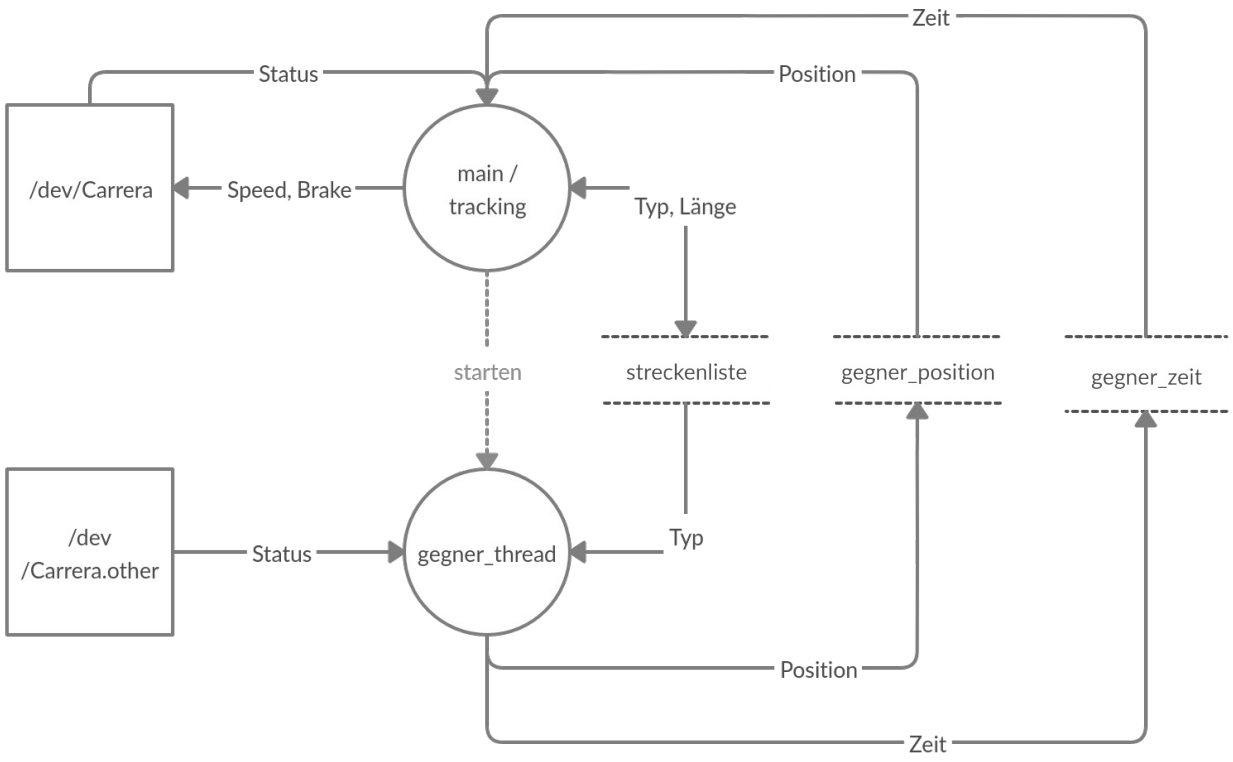
Die Anwendung startet als einzelner Thread: es werden die Devices /dev/Carrera sowie ,/dev/Carrera.other geöffnet, bei Fehler eine Nachricht ausgegeben. Dann werden die beim Programmaufruf übergebenen Argumente ausgewertet: Basisspeed und Rundenzahl.

Im Anschluss wird ein Exit-Handler definiert, der bei Empfang des Signals SIGINT aufgerufen wird. Dieser setzt die Geschwindigkeit des Fahrzeugs auf 0, sodass ein definierter Zustand hergestellt wird, wenn das Programm beendet wird.

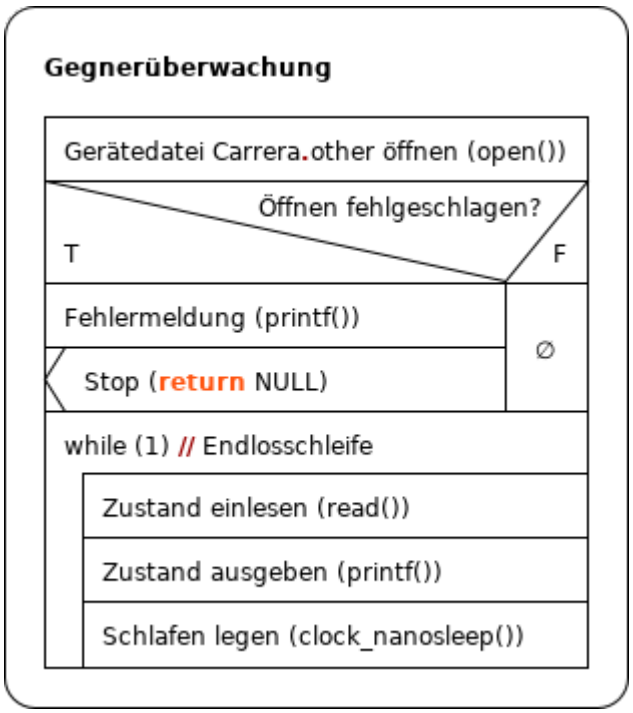
Als letzter Schritt der Vorbereitung wird die Geschwindigkeit des Fahrzeugs auf den übergebenen Basisspeed gesetzt und die Methoden *exploration()* sowie *print_liste()* aufgerufen. Diese lassen das Fahrzeug über die Strecke fahren, bis alle Segmente gefunden und in die globale Liste eingetragen wurden, das Fahrzeug hält dann an der Start/Ziel-Linie. Die Funktion *print_liste()* gibt die soeben erstellte Liste der Segmente aus.

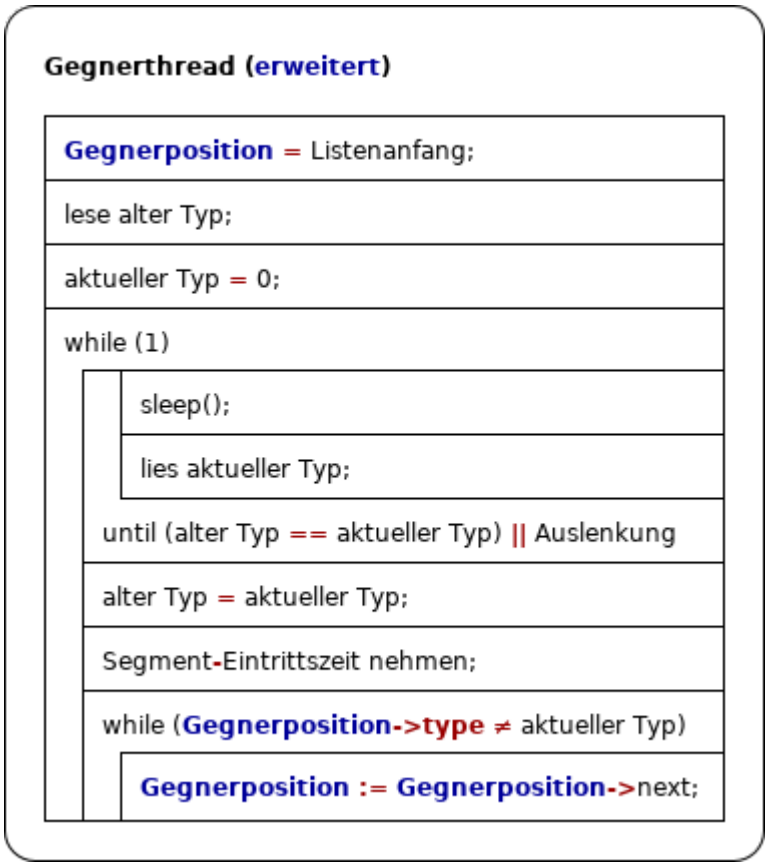
Sind die Vorbereitungen abgeschlossen, gelangen wir zum Hauptteil der Anwendung: für die Überwachung des Gegnerfahrzeugs und zur Kollisionsvermeidung wird ein zweiter Thread erstellt, welcher die Methode *gegner_tracking()* ausführt. Diese liest die aktuelle Position des Gegnerfahrzeugs aus und speichert diese in der globalen Variable *gegner_position*. Der Hauptthread führt die Methode *tracking()* aus, welche das eigene Fahrzeug steuert.

4.2 Datenflussdiagramm



4.3 Struktogramme





fahre_segment(type, laenge)

schlafzeit := laenge * 0.8;

type->bahn == innen

T

F

type

Gerade	Kurve	Gefahr	default
speed += 20;	speed -= 10;	speed -= 15;	
set_speed();	set_speed();	set_speed();	∅
schlafen(**schlafzeit**);	schlafen(**schlafzeit**);	schlafen(**schlafzeit**);	∅
speed -= 20;	speed += 10;	speed += 15;	
set_speed();	set_speed();	set_speed();	

type->bahn == aussen

T

F

type

Gerade	Kurve	Gefahr	default
speed += 20;	speed -= 10;	speed -= 15;	
set_speed();	set_speed();	set_speed();	∅
schlafen(**schlafzeit**);	schlafen(**schlafzeit**);	schlafen(**schlafzeit**);	∅
speed -= 20;	speed += 10;	speed += 15;	
set_speed();	set_speed();	set_speed();	

5 Praktikumstermine

5.1 Praktikum 1

Im ersten Praktikumstermin haben wir neben dem Kennenlernen der Hardware und der Entwicklungsumgebung, ein Programm zur Geschwindigkeitsmessung des Fahrzeugs und der Längenmessung der Strecke auf Basis eines vorhandenen Quellcodes weiterentwickelt, sowie einen Thread implementiert mit dem die Position des Gegnerfahrzeugs überwacht werden kann.

5.1.1 Vorbereitung

Zur Vorbereitung des ersten Praktikums galt es ein vorgegebenes Struktogramm zu implementieren, um das für den Praktikumstermin benötigte Passwort zu erhalten. Das Programm öffnet das Device */dev/Carrera* und setzt das Fahrzeug per *set_speed()* in Bewegung. Im Fehlerfall wird jeweils eine Meldung ausgegeben und das Programm beendet. In einer Endlosschleife wird dann das Fahrzeug über die Bahn fahren gelassen, der Status der Sensoren wird dabei jeweils eingelesen und ausgegeben. Die ersten Halbbytes der Ausgabe, von oben nach unten gelesen ergaben das gesuchte individuelle Passwort.

5.1.2 Nachbereitung

Ziel des ersten Termins war es, sich mit der Strecke vertraut zu machen. Dabei galt es die Länge dieser zu bestimmen. Hierfür wurden die Zeitpunkte der eingelesenen Statusworte in eine Tabellenkalkulation übertragen. Da jeder Sensor zwei Mal auslöst, ein Mal beim Einfahren, ein Mal beim Ausfahren, ließ sich die Geschwindigkeit des Fahrzeugs am Sensor errechnen und kominiert mit dem zeitlichen Abstand zwischen den einzelnen Sensoren auch die Länge l der Strecke.

$$v = \frac{s}{t} = \frac{22mm}{t}$$

$$l = t_{Segment} * v_{Segment}$$

ZEIT	SEGMENT	LÄNGE	GESCHW.
1131 [ms]:	Start/Ziel		
23 [ms]:	Start/Ziel	22	0,956521739
1007 [ms]:	Kurve	963,2173913	0,956521739
19 [ms]:	Kurve	22	1,157894737
854 [ms]:	Kurve	988,8421053	1,157894737
22 [ms]:	Kurve	22	1
826 [ms]:	Brückenanfang	826	1
22 [ms]:	Brückenanfang	22	1
1331 [ms]:	Brückenende	1331	1
16 [ms]:	Brückenende	22	1,375
210 [ms]:	Kurve	288,75	1,375
22 [ms]:	Kurve	22	1
1140 [ms]:	Kurve	1140	1
31 [ms]:	Kurve	22	0,709677419
	Bahnlänge [mm]	5669,809497	

Unseren Messungen zufolge beträgt die Streckenlänge der Brückenbahn ca. 5,67 Meter.

Des Weiteren lag uns das bereits fertig geschriebene Programm *race* vor, welches das Fahrzeug, entsprechend der Parameter in der Datei *parametersatz* über die Strecke fahren ließ. Die folgenden Werte erwiesen sich als besonders effektiv: das Fahrzeug war schnell genug, geriet jedoch nicht ins Schlingern und geriet auch nicht aus der Spur:

```
# Folgende Parameter sind einstellbar:
#
# Verfahren:
# Es gibt eine Basisspeed. Man kann parametrieren, um wieviel sich
# die Basisspeed ändert, bei bestimmten Auslenkungen:
#       Wert des Auslenkungssensors: keine schwach normal stark
# Falls "keine" parametriert ist, wird die Basisspeed bei Durchfahren
# der Start/Ziellinie angepasst.
#
# Ausserdem kann fuer jeden Sensortyp der Offsetwert eingegeben werden,
# der auf die Basisspeed addiert wird. (Eventuell auch die Zeitdauer, wie lange
# dieser Offsetwert aktiv ist)
#
# Typ | Innen/Aussen | Offset (+/-) | Dauer [ms] |
#
# Dauer==0: Bis zum nächsten Sensor (Default)
#
# Basisspeedanpassung
# Auslenkung     Innen    Aussen
0                +3      +3
1                +2      +2
2                +1      +1
3                +1      +1
4                +1      +1
5                -1      -1
6                -2      -2
7                -3      -3
#
#OFFSETS
#
# Typ         Innen   Aussen    Zeit
START            +30      +30      0
GEFAHRENSTELLE   -40      -40      0
ANFANG_BRUECKE   +50      +40      0
ENDE_BRUECKE     -20      -20      0
KURVE            -30      -30      0
```

Es galt dennoch einen kleinen Fehler in der Datei race.c zu beheben. Wurde das Fahrzeug aus der Bahn gehoben und woanders wieder eingesetzt, so zeigte der Pointer für die aktuelle Position auf das falsche Listenelement, die Meldung *wrong position* wurde ausgegeben. Hierzu haben wir eine While-Schleife eingefügt, welche so lange die Liste durchläuft, bis wieder das richtige Element erreicht ist. So erschien die Fehlermeldung nur ein Mal.

```
while ( state_act != position→type)
        position = position->next;
```

Zu guter Letzt galt es das Programm zur Gegnerbeobachtung zu erweitern:

```
void *gegner( void *args ){


        int ret;
        int fd;
        int status;
        int error;
```

```
        struct timespec sleeptime;
        sleeptime.tv_sec = 0;
        sleeptime.tv_nsec = 250000000;

        fd = open ("/dev/Carrera.other", O_RDONLY);
        if ( fd < 0){
                printf("Fehler beim Öffnen /dev/Carrera.other");
                return NULL;
        }

        while(1){
                ret = read(fd, &status, sizeof(status));
                printf("Gengner Status: %4.4x\n", status);
                if ((error=clock_nanosleep(CLOCK_MONOTONIC,0,&sleeptime,NULL))!=0){
                        printf("clock_nanosleep reporting error %d\n",error);
                }
        }

        close(fd);
        return NULL;
}
```

5.2 Praktikum 2

Im zweiten Praktikumstermin haben wir das Gegnerbeobachtungsthread weiterentwickelt und die Methode „tracking“ um die Funktionalität erweitert, die die Kollisionsgefahr an den Gefahrenstellen erkennt und entsprechend mit Bremsvorgang reagiert.

5.2.1 Vorbereitung

Zur Praktikumsvorbereitung war auf dem Vorbereitungsserver die Funktion *unsigned long timespec_to_ulong_microseconds(struct timespec *ts)* in der Datei *collision.c* zu implementieren. Diese liefert den Wert des übergebenen Zeitstempels *ts* als unsigned long in Mikrosekunden zurück.

Anhand eines vorliegenden Struktogramms wurde die Funktion *int collision_check(int fd)* implementiert: befährt das eigene Fahrzeug die Gefahrenstelle und befindet sich das Gegnerfahrzeug ebenfalls, seit weniger als 11 Sekunden, darin, so wird das eigene Fahrzeug abgebremst und zum Stillstand gebracht, um eine Kollision zu vermeiden. Der Thread wird dann schlafen gelegt, beim aufwachen wird die Bremse des eigenen Fahrzeugs gelöst und die vorgesehene Geschwindigkeit gesetzt.

5.2.2 Nachbereitung

Der vorbereitete Code zur Beobachtung der Gegnerspur wurde während des Praktikums in das Programm eingebaut. Es nutzt die beiden globalen Variablen *gegner_position* und *gegner_zeit*, welche das aktuelle Segment und den Einfahrtzeitpunkt des Gegners enthalten.

```
void *gegner_tracking( void *args ){
        gegner_position = root;
        int error;
        unsigned long time_act;
        __u16 state_old = read_with_time(gegner_fdc, &time_act);
        state_old ^= 0x0800;
        __u16 state_act = 0;
```

```
        struct timespec gegner_timestamp;

        struct timespec gegner_sleeptime;
        gegner_sleeptime.tv_sec = 0;
        gegner_sleeptime.tv_nsec = 5000000;

        while (1) {
                do {
                        if ((error=clock_nanosleep(CLOCK_MONOTONIC,0,&gegner_sleeptime,NULL))!=0){
                                printf("clock_nanosleep reporting error %d\n",error);
                        }
                        state_act = read_with_time(gegner_fdc, &time_act);
                        state_act ^= 0x0800;
                } while (state_old == state_act || is_sling(state_act));

                state_old = state_act;

                clock_gettime(CLOCK_MONOTONIC,&gegner_timestamp);
                gegner_time = (gegner_timestamp.tv_sec * 1000000)+(gegner_timestamp.tv_nsec/1000);
                //gegner_time = time_act;

                while (gegner_position->type != state_act)
                        gegner_position = gegner_position->next;

        }
}
```

Der folgende, auf der Vorbereitung beruhende, Code zur Kollisionsvermeidung wurde in die Funktion *tracking()* eingebaut:

```
                else if ((state_act & 0xf000) == 0x2000){  // Kreuzungsstelle
                        sleeptime.tv_sec = 0;
                        sleeptime.tv_nsec = 100000000;     //100 ms
                        diff_zeit = time_act - gegner_time;
                        while (( position == gegner_position ) && ( diff_zeit < 2500000 )) {
                                //Gegner im gleichen Segment und diff kleiner als 2500ms
                                set_speed(fdc, 0x0b);
                                //write(fdc, &brake, sizeof(brake));
                                printf("Gegner auch in Gefahrenstelle seit : %ld\n ms", diff_zeit);
                                //Schlafe
                                clock_nanosleep(CLOCK_MONOTONIC, 0, &sleeptime, NULL);
                                //Messe die Zeitunterschied neu
                                clock_gettime(CLOCK_MONOTONIC,&timestamp);
                                time_act = (timestamp.tv_sec * 1000000)+(timestamp.tv_nsec/1000);
                                diff_zeit = time_act - gegner_time;
                        }
                        //Clear brake
                        set_speed(fdc, 0x0c);
                        //write(fdc, &clear_brake, sizeof(clear_brake));
                }
```

5.3 Praktikum 3

Der dritte Praktikumstermin stand im Zeichen der Basisspeedanpassung. Wie das Programm follow im ersten Praktikum sollen hier die verschiedenen Segmente der Strecke mit unterschiedlichen Basisgeschwindigkeiten durchfahren werden.

5.3.1 Vorbereitung

Wie in der Aufgabenstellung gefordert berücksichtigt unser Entwurf neben dem Segemnttyp und der Auslenkung des Fahrzeugs die Innen- und Außenbahn der Strecke. Da sich der Kurvenradius der einzelnen Spuren unterscheidet sollte hier auch eine andere Geschwindigkeit gefahren werden.

Zudem galt es die Frage zu beantworten, warum es nicht ausreicht die Geschwindigkeit des Fahrzeugs nur nach den Sensoren zu setzen? Warum müssen wir die Streckenlänge mit berücksichtigen?

Es reicht nicht die Geschwindigkeit anzupassen, wenn der Sensor durchfahren wird, denn dann ist es möglicherweise schon zu spät. Berücksichtigt man die Länge der einzelnen Segmente ist es möglich auf einer Geraden stärker zu beschleunigen und schon früher für die kommende Kurve abzubremsen. Mit der gleichen Technik ist es möglich nach der Hälfte der Kurve die Geschwindigkeit zu erhöhen und aus dieser hinauszubeschleunigen.

Als grobe Idee für eine solche Basisgeschwindigkeitsanpassung haben wir folgenden Pseudocode entworfen:

```
fahre_segment(unsigned int type, unsigned int laenge) {
    schlafzeit = laenge * 0.8;

    if(innen) {
        switch(type) {
            case GERADE:
                speed += 10;
                set_speed();
                schlafen();
                speed -= 10;
                set_speed();
                break;
            case GEFAHR:
                speed -= 10;
                set_speed();
                schlafen();
                speed += 10;
                set_speed();
                break;
            case KURVE:
                speed -= 10;
                set_speed();
                schlafen();
                speed += 10;
                set_speed();
                break;
        }
    } else {
        switch(type) {
            case GERADE:
```

```
                speed += 10;
                set_speed();
                schlafen();
                speed -= 10;
                set_speed();
                break;
            case GEFAHR:
                speed -= 10;
                set_speed();
                schlafen();
                speed += 10;
                set_speed();
                break;
            case KURVE:
                speed -= 10;
                set_speed();
                schlafen();
                speed += 10;
                set_speed();
                break;
        }
    }
}
```

5.3.2 Nachbereitung

Nachdem wir den in der Vorbereitung erdachten Code in unsere Anwendung eingebaut hatten, haben wir die Anwendung mit verschiedenen Fahrzeugen getestet, um die entsprechend optimale anzugebende Basisgeschwindigkeit beim Aufruf angeben zu können.

Während des abschließenden Rennens stellte sich jedoch heraus, dass der Code im Bereich der Auslenkungserkennung zu lange in der Schleife verweilte, sodass die Geschwindigkeit des Fahrzeugs zum Ende des Rennens hin gegen Null ging, das Rennen letztendlich sogar von uns abgebrochen wurde.

Hier hätten wir gerne geforscht, wie dieser Fehler zustande kam, aufgrund der Zeitbeschränkung des Praktikums war dies aber leider nicht möglich.

```
/*
Quellcode zum Praktikum "Echtzeitsysteme" im Fachbereich Elektrotechnik
und Informatik der Hochschule Niederrhein.

Fuer die Generierung wird die Realzeitbibliothek "rt" benoetigt.
Wenn das folgende Makefile verwendet wird, muss zur Generierung nur
noch "make" auf der Kommandozeile eingegeben werden:
==================================
CFLAGS=-g -Wall -m32
LDLIBS=-lrt

all: race

clean:
        rm -f race *.o
==================================
```

```
Sa 27. Sep 17:57:59 CEST 2014
*/
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <malloc.h>
#include <time.h>
#include <sys/time.h>
#include <asm/types.h>
#include <pthread.h>

#define make_seconds( a ) (a/1000000)
#define make_mseconds( a ) ((a-((a/1000000)*1000000))/1000)

static void set_speed( int fd, int Speed );

static unsigned char basis_speed_in=0x24, basis_speed_out=0x24;
static unsigned long last_time;
static int fdc=0;
static int gegner_fdc=0;
struct _liste *gegner_position = NULL;
static unsigned long gegner_time = 0;
static unsigned char brake = 0x0b, clear_brake = 0x0c;

int auslenkung_innen = -1;
int auslenkung_aussen = -1;

typedef struct _liste {
        int type;
        int length;
        struct _liste *next;
} streckenliste;

static streckenliste *root = NULL;

static struct _liste *add_to_liste( int type, int length )
{
        struct _liste *ptr, *lptr;

        lptr = malloc( sizeof(struct _liste) );
        if( lptr==NULL ) {
                printf("out of memory\n");
                return NULL;
        }
        lptr->type = type;
        lptr->length = length;
        lptr->next = root;
        if( root == NULL ) {
```

```
                root = lptr;
                lptr->next = lptr;
                return root;
        }
        for( ptr=root; ptr->next!=root; ptr=ptr->next )
                ;
        ptr->next = lptr;
        return lptr->next;
}

static void print_liste(void)
{
        struct _liste *lptr = root;
        int length_sum=0;

        do {
                printf("0x%04x - %d [mm]\n", lptr->type, lptr->length );
                length_sum += lptr->length;
                lptr = lptr->next;
        } while( lptr!=root );
        printf("sum: %d\n", length_sum );
}

static void exithandler( int signr )
{
        if( fdc )
                set_speed( fdc, 0x0 );
        exit( 0 );
}

static void set_speed( int fd, int speed )
{
        printf("new speed: 0x%x\n", speed );
        if( fd>0 )
                write( fd, &speed, sizeof(speed) );
}

static __u16 read_with_time( int fd, unsigned long *time1 )
{
        struct timespec timestamp;
        __u16 state;
        ssize_t ret;

        ret=read( fd, &state, sizeof(state) );
        if (ret<0) {
                perror( "read" );
                return -1;
        }
        clock_gettime(CLOCK_MONOTONIC,&timestamp);
        *time1 = (timestamp.tv_sec * 1000000)+(timestamp.tv_nsec/1000);
        return state;
```

```
}

static inline int is_sling( __u16 state )
{
         int auslenkung = 0;
        if ( (state&0xf000)==0x0000 ){
                // Es liegt eine Auslenkung vor...
                 auslenkung = (int)(state&0x000f);

                // Die Auslenkung wird abhaengig von der Spur
                // in auslenkung_innen bzw. auslenkung_aussen abgespeichert
                if ( (state&0x0800)==0x0000){
                        // auslenkung innen
                        auslenkung_innen = auslenkung;
                }
                if ( (state&0x0800)==0x0800){
                        // auslenkung aussen
                        auslenkung_aussen = auslenkung;
                }
                printf("Aussen: %d, Innen: %d\n", auslenkung_aussen, auslenkung_innen);
                return 1;
        }
        printf("Aussen: %d, Innen: %d\n", auslenkung_aussen, auslenkung_innen);
        return 0; // keine Auslenkung

}

int change_speed()
{
        switch(auslenkung_innen)
        {
                case -1: basis_speed_in += 6;
                        break;
                case 0: basis_speed_in += 0;
                        break;
                case 1: basis_speed_in += 5;
                        break;
                case 2: basis_speed_in += 3;
                        break;
                case 3: basis_speed_in += 3;
                        break;
                case 4: basis_speed_in -= 2;
                        break;
                case 5: basis_speed_in -= 1;
                        break;
                case 6: basis_speed_in -= 4;
                        break;
                case 7: basis_speed_in -= 5;
                        break;
                default: break;
        }
```

```
        switch(auslenkung_aussen)
        {
                case -1: basis_speed_out += 7;
                        break;
                case 0: basis_speed_out += 4;
                        break;
                case 1: basis_speed_out += 4;
                        break;
                case 2: basis_speed_out += 1;
                        break;
                case 3: basis_speed_out += 3;
                        break;
                case 4: basis_speed_out -= 5;
                        break;
                case 5: basis_speed_out -= 1;
                        break;
                case 6: basis_speed_out -= 0;
                        break;
                case 7: basis_speed_out -= 4;
                        break;
                default: break;
        }
         if(basis_speed_out>0xc0) {
                  basis_speed_out=0xc0;
         }
         if(basis_speed_in>0xc0) {
                  basis_speed_in=0xc0;
         }
        return 1;
}

void schlafen(unsigned long msec) {
         struct timespec ts;
         ts.tv_sec=msec/1000;
         ts.tv_nsec=(msec%1000)*1000000;
         clock_nanosleep(CLOCK_MONOTONIC, 0, &ts, NULL);
}


static char *decode( __u16 status )
{
         if( (status&0xf000)==0x0000 )
                  return "Auslenkungssensor";
         if( (status&0xf000)==0x1000 )
                  return "Start/Ziel";
         if( (status&0xf000)==0x2000 )
                  return "Gefahrenstelle";
         if( (status&0xf000)==0x3000 )
                  return "NICHT KODIERT";
         if( (status&0xf000)==0x4000 )
                  return "NICHT KODIERT";
```

```
        if( (status&0xf000)==0x5000 )
                return "NICHT KODIERT";
        if( (status&0xf000)==0x6000 ) {
                if( (status&0x0400)==0x0400 )
                        return "Brueckenende";
                else
                        return "Brueckenanfang";
        }
        if( (status&0xf000)==0x7000 )
                return "NICHT KODIERT";
        if( (status&0xf000)==0x8000 )
                return "Kurve";
        if( (status&0xf000)==0x9000 )
                return "NICHT KODIERT";
        if( (status&0xf000)==0xa000 )
                return "NICHT KODIERT";
        if( (status&0xf000)==0xb000 )
                return "NICHT KODIERT";
        if( (status&0xf000)==0xc000 )
                return "NICHT KODIERT";
        if( (status&0xf000)==0xd000 )
                return "NICHT KODIERT";
        if( (status&0xf000)==0xe000 )
                return "NICHT KODIERT";
        if( (status&0xf000)==0xf000 )
                return "NICHT KODIERT";
        return "TYP UNBEKANNT";
}

void fahre_segment(unsigned int type, unsigned int laenge) {
        unsigned long schlafzeit = laenge * 0.8;

        if ( type==0x0000){ // innen
                switch(type&0xf000) {
                        case 0x2000:
                                /*basis_speed_in -= 10;
                                set_speed(fdc, basis_speed_in);
                                printf("Gefahrenstelle: -10");
                                schlafen(schlafzeit*0.4);
                                basis_speed_in += 10;
                                printf("Gefahrenstelle: +10");
                                set_speed(fdc, basis_speed_in);*/
                        break;
                        case 0x8000:
                                /*basis_speed_in -= 15;
                                set_speed(fdc, basis_speed_in);
                                printf("Kurve: -15");
                                schlafen(schlafzeit*0.4);
                                basis_speed_in += 15;
                                printf("Kurve: +15");
                                set_speed(fdc, basis_speed_in);*/
```

```
                        break;
                        default:
                                basis_speed_in += 30;
                                set_speed(fdc, basis_speed_in);
                                schlafen(schlafzeit*0.5);
                                basis_speed_in -= 30;
                                set_speed(fdc, basis_speed_in);
                        break;
                }
        }
        if ( type==0x0800){
                // aussen

                switch(type&0xf000) {
                        case 0x2000:
                                /*basis_speed_in -= 10;
                                set_speed(fdc, basis_speed_in);
                                printf("Gefahrenstelle: -10");
                                schlafen(schlafzeit*0.4);
                                basis_speed_in += 10;
                                printf("Gefahrenstelle: +10");
                                set_speed(fdc, basis_speed_in);*/
                        break;
                        case 0x8000:
                                /*basis_speed_out -= 15;
                                set_speed(fdc, basis_speed_out);
                                printf("Kurve: -15");
                                schlafen(schlafzeit*0.4);
                                basis_speed_out += 15;
                                printf("Kurve: +15");
                                set_speed(fdc, basis_speed_out);*/
                        break;
                        default:
                                basis_speed_out += 30;
                                set_speed(fdc, basis_speed_out);
                                schlafen(schlafzeit*0.5);
                                basis_speed_out -= 30;
                                set_speed(fdc, basis_speed_out);
                        break;
                }
        }
}


static void exploration( int fdc )
{
        __u16 state_old, state_act;
        unsigned long time_act, time_old;
        int rounds=0, length, length_sum=0, v;

        do {
```

```
                state_old = read_with_time( fdc, &time_old );
        } while( (state_old&0xf000)!=0x1000 ); // fahre bis Start
        while( rounds<2 ) {
                do {
                        state_act=read_with_time( fdc, &time_act );
                } while( is_sling(state_act) );
                printf("old/act: 0x%x/0x%x\t%6.6ld [ms] - ",
                        state_old, state_act,
                        (time_act-time_old)/1000 );
                if( state_act == state_old ) {
                        v = 22000000/(time_act-time_old);
                        if( (state_act&0xf000)==0x1000 ) {
                                rounds++;
                                last_time = time_act;
                                printf("Round: %d - %d [mm]\n",
                                        rounds, length_sum );
                                length_sum = 0;
                        }
                        printf("v=0,%d\n", v );
                } else {
                        length = v * (time_act - time_old)/1000000;
                        length_sum += length;
                        printf("%s: length=%d\n", decode(state_act),length);
                        add_to_liste( state_act, length );
                }
                state_old = state_act;
                time_old = time_act;
        }
}

static void tracking( int rounds_to_go )
{
        struct _liste *position = root;
        int rounds=0;
        __u16 state_act;
        unsigned long time_act, besttime=(-1), meantime=0;
        struct timespec sleeptime;
       unsigned long  diff_zeit;
        struct timespec timestamp;

        do {
                do {
                        state_act=read_with_time( fdc, &time_act );
                } while( is_sling(state_act) );

                state_act=read_with_time( fdc, &time_act );
                printf("0x%04x (expected 0x%04x)\n",state_act,position->type);
                if( state_act != position->type){
                        printf("wrong position 0x%04x (0x%04x)\n",
                                state_act, position->type);
                        while ( state_act != position->type)
```

```
                                position = position->next;
                }

                fahre_segment(position->type, position->length);

                if( (state_act&0xf000)==0x1000 ) { // Start/Ziel
                        rounds++;
                        rounds_to_go--;
                        if( last_time ) {
                                if( (time_act-last_time)<besttime )
                                        besttime = time_act-last_time;
                                meantime += time_act-last_time;
                                printf("\n---> Runde: %d - Zeit: %ld.%03lds "
                                        "(Beste: %ld.%03lds, "
                                        "Mean: %ld.%03lds)\n",
                                        rounds,
                                make_seconds((time_act-last_time)),
                                make_mseconds((time_act-last_time)),
                                make_seconds(besttime),
                                make_mseconds(besttime),
                                make_seconds(meantime/rounds),
                                make_mseconds(meantime/rounds));
                        }
                        last_time = time_act;
                }

                else if ((state_act & 0xf000) == 0x2000){  // Kreuzungsstelle
                        sleeptime.tv_sec = 0;
                sleeptime.tv_nsec = 100000000;     //100 ms

                        diff_zeit = time_act - gegner_time;

                        printf("Eigenfahrzeug %4.4x - Gegnerfahrzeug  %4.4x, diff_zeit %lu\n", position->type,
gegner_position->type,diff_zeit);

                        while (( position == gegner_position ) && ( diff_zeit < 2500000 )) {
                                //Gegner im gleichen Segment und diff kleiner als 2500ms
                                set_speed(fdc, 0x0b);
                                //write(fdc, &brake, sizeof(brake));
                                printf("Gegner auch in Gefahrenstelle seit : %ld\n ms", diff_zeit);
                                //Schlafe
                                clock_nanosleep(CLOCK_MONOTONIC, 0, &sleeptime, NULL);
                                //Messe die Zeitunterschied neu
                                clock_gettime(CLOCK_MONOTONIC,&timestamp);
                                time_act = (timestamp.tv_sec * 1000000)+(timestamp.tv_nsec/1000);
                                diff_zeit = time_act - gegner_time;
                        }
                        //Clear brake
                        set_speed(fdc, 0x0c);
                        //write(fdc, &clear_brake, sizeof(clear_brake));
                }
                position = position->next;
```

```
                change_speed();
                if ( (state_act&0x0800)==0x0000){
                        // innen
                        set_speed(fdc, basis_speed_in);
                }
                if ( (state_act&0x0800)==0x0800){
                        // aussen
                        set_speed(fdc, basis_speed_out);
                }
        } while( rounds_to_go );
}

//void *gegner( void *args ){
//
//      int ret;
//      int fd;
//      int status;
//      int error;
//
//      struct timespec sleeptime;
//      sleeptime.tv_sec = 0;
//      sleeptime.tv_nsec = 250000000;
//
//      fd = open ("/dev/Carrera.other", O_RDONLY);
//      if ( fd < 0){
//              printf("Fehler beim Öffnen /dev/Carrera.other");
//              return NULL;
//      }
//
//      while(1){
//      if (open ("/dev/Carrera.other", O_RDONLY) < 0){
//              ret = read(fd, &status, sizeof(status));
//              printf("Gengner Status: %4.4x\n", status);
//              if ((error=clock_nanosleep(CLOCK_MONOTONIC,0,&sleeptime,NULL))!=0){
//                      printf("clock_nanosleep reporting error %d\n",error);
//              }
//
//              }
//      }
//
//      close(fd);
//      return NULL;
//}

void *gegner_tracking( void *args ){

        gegner_position = root;
        int error;
        unsigned long time_act;
        __u16 state_old = read_with_time(gegner_fdc, &time_act);
        state_old ^= 0x0800;
```

```
        __u16 state_act = 0;
        struct timespec gegner_timestamp;

        struct timespec gegner_sleeptime;
        gegner_sleeptime.tv_sec = 0;
        gegner_sleeptime.tv_nsec = 5000000;

        while (1) {
                do {
                        if ((error=clock_nanosleep(CLOCK_MONOTONIC,0,&gegner_sleeptime,NULL))!=0){
                                printf("clock_nanosleep reporting error %d\n",error);
                        }
                        state_act = read_with_time(gegner_fdc, &time_act);
                        state_act ^= 0x0800;
                } while (state_old == state_act || is_sling(state_act));

                state_old = state_act;

                clock_gettime(CLOCK_MONOTONIC,&gegner_timestamp);
                gegner_time = (gegner_timestamp.tv_sec * 1000000)+(gegner_timestamp.tv_nsec/1000);
                //gegner_time = time_act;

                while (gegner_position->type != state_act)
                        gegner_position = gegner_position->next;

        }
}

int main( int argc, char **argv )
{
        int rounds_to_go=10;
        struct sigaction new_action;

        fdc = open( "/dev/Carrera", O_RDWR );
        if( fdc<0 ) {
                perror( "/dev/Carrera" );
                return -1;
        }


        gegner_fdc = open ("/dev/Carrera.other", O_RDONLY);
        if ( gegner_fdc < 0){
                perror( "/dev/Carrera.other" );
                return -1;
        }

        if( argc > 1 ) {
                basis_speed_in=basis_speed_out=
                        (unsigned char)strtoul(argv[1],NULL,0);
                if( argc > 2 ) {
                        rounds_to_go = (unsigned int)strtoul(argv[2],NULL,0);
```

```
                }
        }
        new_action.sa_handler = exithandler;
        sigemptyset( &new_action.sa_mask );
        new_action.sa_flags = 0;
        sigaction( SIGINT, &new_action, NULL );

        set_speed( fdc, basis_speed_in );
        exploration( fdc );
        print_liste();

        set_speed( fdc, brake );
        printf("Exploration fertig.\n");
        printf("Enter drücken um Rennen zu starten!");
        getchar();
        set_speed( fdc, clear_brake );

        // Gegner-thread

        pthread_t gegner_thread;
        if( pthread_create (&gegner_thread, NULL, gegner_tracking, NULL)!=0){
                fprintf(stderr, "Creation of gegner_thread failed");
                return -1;
        }

        tracking( rounds_to_go );

        pthread_join(gegner_thread, NULL); //auf thread beendigung warten

        set_speed( fdc, 0x0 );
        return 0;
}
```