

# Echtzeitbetriebssysteme — Übung

Oliver Jack

Ernst-Abbe-Hochschule Jena  
Fachbereich Elektrotechnik und Informationstechnik

Sommersemester 2025



**Ernst-Abbe-Hochschule Jena**  
University of Applied Sciences

# Übung 5: Signal handler

- VxWorks unterstützt Signale. Wir betrachten in diesem Praktikum POSIX-Signale.
- Signale ändern asynchron den Steuerfluss eines Prozesses.
- Der signalisierte Task unterbricht unmittelbar seine Ausführung und die spezifizierte **signal handler-Routine** wird ausgeführt, sobald der Task das nächste mal den Prozessor besitzt.
- Der signal handler wird auch ausgeführt, wenn der Task blockiert ist.
- Der signal handler ist eine Benutzer-Routine, die an ein bestimmtes Signal gebunden ist.
- Signale sind eher für Fehler- und Ausnahme-Behandlung angemessen, als für allgemeine Task-Kommunikation.

- VxWorks unterstützt 31 verschiedene Signale (s. VxWorks Manual).
- Ein Signal kann mit der Funktion `kill()` gesetzt werden, analog zu einem Interrupt oder einer Hardware-Ausnahme.
- Das Signal wird mit der Funktion `sigaction()` an einen signal handler gebunden.
- Während der signal handler ausgeführt wird, sind andere Signale blockiert.
- Tasks können das Auftreten bestimmter Signale mit der Funktion `sigprocmask()` blockieren.
- Falls Signale so blockiert sind, wird deren signal handler ausgeführt, sobald das Signal deblockiert ist.

# Signal handler

## Definition

```
void sigHandlerFunction(int signalNumber)
```

```
{  
    ..... /* signal handler code */  
    .....  
    .....  
}
```

signalNumber ist das Signal, für das sigHandlerFunction aufgerufen wird.

# Signal handler (Forts.)

Die Funktion `sigaction` installiert den signal handler für einen Task.

```
int sigaction(int signo, const struct sigaction *pAct,  
             struct sigaction *pOact);
```

- `struct sigaction` enthält die handler Information.
- Parameter 1: `signo` ist die gebundene Signalnummer
- Parameter 2: `pAct` ist Zeiger auf die neue Handler Struktur
- Parameter 3: `pOact` ist Zeiger auf die alte Handler Struktur
- Falls die alte Handler Struktur nicht benötigt wird, kann `NULL` übergeben werden.
- Die Zuweisung eines Signals zu einem Task erfolgt mit der Funktion `kill(int, int)`, wobei der erste Parameter die Task-Id der zugewiesenen Task und der zweite Parameter das Signal ist.

# Experiment

- Im Beispielprogramm `VxWorks_signal.c` erzeugt die Funktion `sigGenerator()` das SIGINT oder Ctrl-C Signal und sendet es an die `sigCatcher` Task. Wenn `sigCatcher` das Signal empfängt, unterbricht dieser Task die normale Abarbeitung und “springt” zu dem installierten signal handler (Funktion `catchSIGINT()`).

- Aufgabe:** Aufgabe: bringen Sie das Beispielprogramm `VxWorks_signal.c` zum laufen. Das Programm sollte eine Ausgabe erzeugen, wie auf der letzten Folie dieses Dokuments zu sehen ist.
- Aufgabe:** Modifizieren Sie das Programm, so das kein signal handler für `sigCatcher` installiert ist. Was passiert jetzt, wenn das Signal SIGINT an `sigCatcher` gesendet wird?
- Aufgabe:** Modifizieren Sie das Programm, so dass das Signal SIGINT blockiert ist, d. h. `sigCatcher` kann das Signal SIGINT nicht empfangen.

# VxWorks\_signal.c

```
/* includes */
#include "vxWorks.h"
#include "sigLib.h"
#include "taskLib.h"
#include "stdio.h"

/* function prototypes */
void catchSIGINT(int);
void sigCatcher(void);

/* globals */
#define NO_OPTIONS 0
#define ITER1 100
#define LONG_TIME 1000000
#define HIGHPRIORITY 100
#define LOWPRIORITY 101
int ownId;
```



```
void sigGenerator(void) /* task to generate the SIGINT signal */
{
    int i, j, taskId;
    STATUS taskAlive;

    if((taskId = taskSpawn("signal",100,0x100,20000,(FUNCPTR)sigCatcher
        ,0,0,0,0,0,0,0,0,
        0,0,0)) == ERROR)
        printf("taskSpawn_sigCatcher_failed\n");

    ownId = taskIdSelf(); /* get sigGenerator's task id */

    taskDelay(30); /* allow time to get sigCatcher to run */
}
```

```
for (i=0; i < ITER1; i++)
{
    if ((taskAlive = taskIdVerify(taskId)) == OK)
    {
        printf("++++++SIGINT_signal_
              generated\n");
        kill(taskId, SIGINT); /* generate signal */
        /* lower sigGenerator priority to allow sigCatcher to run
           */
        taskPrioritySet(ownId, LOWPRIORITY);
    }
    else /* sigCatcher is dead */
        break;
}

printf("\n*****sigGenerator_Exited*****\n");
}
```

# VxWorks\_signal.c

```
void sigCatcher(void) /* task to handle the SIGINT signal */
{
    struct sigaction newAction;
    int i, j;

    newAction.sa_handler = catchSIGINT; /* set the new handler */
    sigemptyset(&newAction.sa_mask); /* no other signals blocked */
    newAction.sa_flags = NO_OPTIONS; /* no special options */

    if(sigaction(SIGINT, &newAction, NULL) == -1)
        printf("Could not install signal handler\n");

    for (i=0; i < ITER1; i++)
    {
        for (j=0; j < LONG_TIME; j++);
        printf("Normal processing in sigCatcher\n");
    }

    printf("\n+++++++sigCatcher_Exited+++++++\n");
}
```

# VxWorks\_signal.c - Ausgabe

```
Normal processing in sigCatcher
Normal processing in sigCatcher
+++++SIGINT signal generated
-----SIGINT signal caught
+++++SIGINT signal generated
-----SIGINT signal caught
// Insgesamt 100 mal
+++++SIGINT signal generated
-----SIGINT signal caught
+++++SIGINT signal generated
-----SIGINT signal caught

*****sigGenerator Exited*****
Normal processing in sigCatcher
Normal processing in sigCatcher
// Insgesamt 100 mal
Normal processing in sigCatcher
Normal processing in sigCatcher

+++++sigCatcher Exited+++++
```