

# Technische Informatik 3 – Embedded Systems

## Kapitel 1: Ein- und Ausgabe

Prof. Dr. Benjamin Kormann

Fakultät für Elektro- und Informationstechnik

24.03.2023



# Erstes C Programm

Definition einer Funktion mit zwei Parametern

Einbinden des Standard-Headers für Ein-/Ausgabe

```
#include <stdio.h>

// Funktion zur Addition von zwei Ganzzahlen
int add(int a, int b)
{
    return a + b;
}

// Funktion zur Subtraktion von zwei Gleitkommazahlen
double sub(double a, double b)
{
    return a - b;
}
```

Headerinformationen und Funktionsdefinition

Blockdefinition mit {} Klammern

Blockdefinition mit {} Klammern

Hauptprogramm (Programmstart)

```
// Start des Programms (main-Funktion)
int main(int argc, char* argv[])
{
    // Variablen definieren
    int x = 5;
    int y = 7;
    // Funktionsaufruf
    int sum = add(x, y);
    // Ausgabe
    printf("Summe von 5 und 7 ist %d\n", sum);

    // Variablen definieren
    double f = 2.2;
    double g = 4.4;
    // Funktionsaufruf
    double diff = sub(f, g);
    // Ausgabe
    printf("Differenz von 2.2 und 4.4 ist %lf\n", diff);

    // Programm erfolgreich beenden
    return 0;
}
```

Programmparameter (Anzahl: argc, Werte: argv)

Datentyp int für Ganzzahlen

Rückgabewert

Funktionsaufruf

Formatierung der Zahl sum als Ganzzahl (%d) für die Ausgabe

Ausgabe auf der Konsole (stdout)

Datentyp double für Gleitkommazahlen

Formatierung der Zahl diff als Gleitkommazahl (%lf) für die Ausgabe

Beenden des Hauptprogramms

# Der Header `stdio.h`

## Standard Input and Output Library

### Beschreibung des Umfangs der Header Datei

- „This library uses what are called streams to operate with physical devices such as keyboards, printers, terminals or with any other type of files support by the system. Streams are an abstraction to interact with these in an uniform way; All streams have similar properties independently of the individual characteristics of the physical media they are associated with.“ [<https://www.cplusplus.com/reference/cstdio/>]
- Eigenschaften von Streams
  - Zugriff: Auf Streams kann lesend oder schreibend (oder beides) zugegriffen werden
  - Art: Unterscheidung zwischen Textdateien (zeilenweise Kodierung mit `\n` Ende) und Binärdateien (Byte-Sequenz)
  - Puffer: Speicherbereich zur zwischenzeitlichen Datenaufnahme vor dem eigentlichen Lese- oder Schreibvorgang
  - Ausrichtung: Streams sind entweder byte-orientiert (`stdio.h`) oder wide-orientiert (`wchar.h`)
- Indikatoren
  - `error`: Dieser wird gesetzt sobald ein Fehler bei der Stream-Bearbeitung aufgetreten ist
  - `eof`: end-of-file wird gesetzt, nachdem das letzte Byte/Zeichen im Stream gelesen wurde
  - `pos`: Interner Zeiger eines Streams für Lese- und Schreibvorgänge

# Der Header `stdio.h`

## Ein-/Ausgabe mit Hilfe des FILE Pointers

### Streams sind Zeiger auf FILE Objekte

- Ein FILE Objekt repräsentiert einen Stream eindeutig
- C definiert drei Standardobjekte
  - `stdin`: Standardeingabe (Tastatur)
  - `stdout`: Standardausgabe (Konsole)
  - `stderr`: Standardfehlerausgabe (Konsole)

### Funktionen auf Standardobjekten (`stdout`, `stdin`)

- `printf()`, `scanf()`, `gets()`

### Funktionen auf beliebigen Streams

- `fprintf()`, `fscanf()`, `fgets()`

### Funktionen auf Zeichenketten

- `sprintf()`, `sscanf()`


```
#include <stdio.h>

int main()
{
    // Syntax:
    // int printf ( const char * format, ... );
    printf("Das ist eine Zeichenkette in einer Zeile\n");

    int value = 17;
    printf("%d * 2 = %d\n", value, value*2);

    // Puffer anlegen für die Eingabe
    char buffer[128];
    // Syntax:
    // int scanf ( const char * format, ... );
    scanf("%s", buffer);
    printf("Eingegebener Wert: %s\n", buffer);

    return 0;
}
```



```
Das ist eine Zeichenkette in einer Zeile
17 * 2 = 34
hallo
Eingegebener Wert: hallo
```

# Der Header `stdio.h`

## Die `printf()` Funktion

```
int printf ( const char * format, ... );
```

- Die Funktion hat mindestens einen (`format`) Parameter
- Die `printf()` Funktion schreibt die übergebenen Werte auf `stdout`
- Der Parameter `format` ist eine Zeichenkette (string)
  - Strings enden mit dem sog. Null-Byte `'\0'`
  - Strings haben in C den Datentyp `char*`
- Eine neue Zeile wird mit `'\n'` in der Zeichenkette codiert

## Die Formatierung der Ausgabe erfolgt über Formatangaben


- Formatangaben werden über das `%` Zeichen festgelegt
- Für jedes `%` Zeichen in dem `format` Parameter
  - Muss ein weiterer Parameter übergeben werden
  - Muss die richtige Codierung gewählt werden

```
#include <stdio.h>

int main()
{
    // Variablendefinitionen
    char c1   = 'A';
    char* c2  = "Hallo";
    short s1  = 73;
    int i1    = 1024;
    long l1   = 8096;
    float f1  = 47.11;
    double d1 = 73.73;

    // Formatierte Ausgaben
    printf("%c\n", c1);
    printf("%s\n", c2);
    printf("%d %08d %ld\n", s1, i1, l1);
    printf("%f %4.4lf\n", f1, d1);

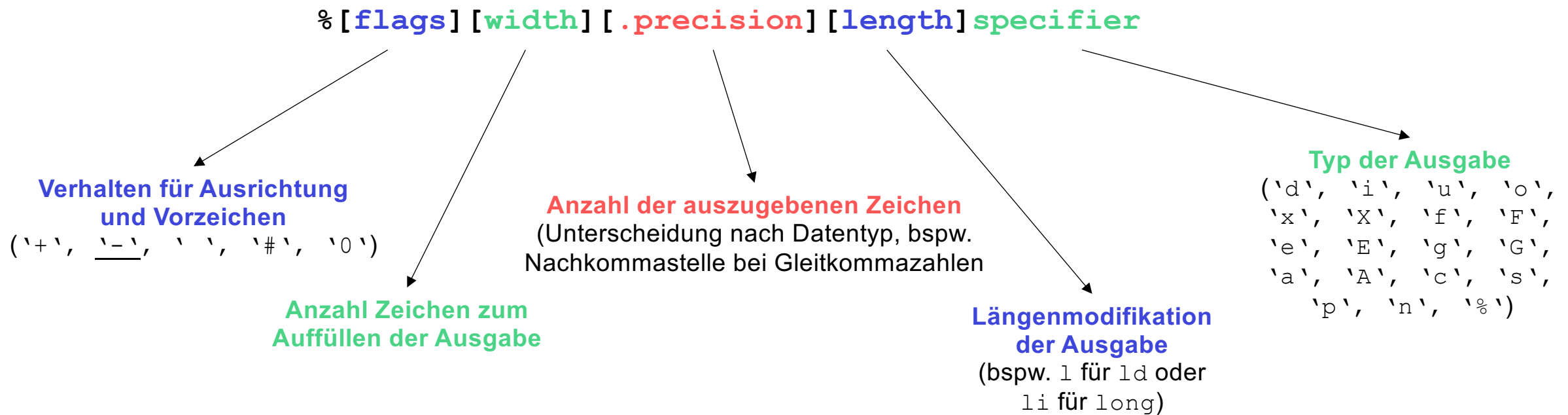
    return 0;
}
```



```
A
Hallo
73 00001024 8096
47.110001 73.7300
```

# Der Header `stdio.h`

## `format specifier`



# Der Header `stdio.h`

## format specifier

<b>specifier</b>	<b>Output</b>	<b>Example</b>
<code>d</code> or <code>i</code>	Signed decimal integer	392
<code>u</code>	Unsigned decimal integer	7235
<code>o</code>	Unsigned octal	610
<code>x</code>	Unsigned hexadecimal integer	7fa
<code>X</code>	Unsigned hexadecimal integer (uppercase)	7FA
<code>f</code>	Decimal floating point, lowercase	392.65
<code>F</code>	Decimal floating point, uppercase	392.65
<code>e</code>	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
<code>E</code>	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
<code>g</code>	Use the shortest representation: <code>%e</code> or <code>%f</code>	392.65
<code>G</code>	Use the shortest representation: <code>%E</code> or <code>%F</code>	392.65
<code>a</code>	Hexadecimal floating point, lowercase	-0xc.90fep-2
<code>A</code>	Hexadecimal floating point, uppercase	-0XC.90FEP-2
<code>c</code>	Character	a
<code>s</code>	String of characters	sample
<code>p</code>	Pointer address	b8000000
<code>n</code>	Nothing printed. The corresponding argument must be a pointer to a <code>signed int</code> . The number of characters written so far is stored in the pointed location.	
<code>%</code>	A <code>%</code> followed by another <code>%</code> character will write a single <code>%</code> to the stream.	%

<https://www.cplusplus.com/reference/cstdio/printf/>

# Der Header `stdio.h`

## Die `scanf()` Funktion

```
int scanf ( const char * format, ... );
```

- Die Funktion hat mindestens einen (`format`) Parameter
- Die `scanf()` Funktion liest Werte von `stdin` in einen Datenbereich
- Der Parameter `format` enthält die Formatanweisung (string)
- **Gefahr der Funktion:** Pufferüberlauf, wenn der Eingabe des Benutzers wird vertraut

### Die Formatierung der Eingabe erfolgt über Formatangaben

- Zeichen werden gemäß `format` string gelesen und interpretiert
- Rückgabewert: Anzahl der erfolgreich gelesenen Werte
- Format: `%[*][width][length]specifier`


```
#include <stdio.h>

int main()
{
    // Datenbereiche anlegen
    char buffer[128];
    int value = 0;

    // Aufforderung und Einlesen der Werte
    printf("Bitte geben Sie Ihren Namen ein: ");
    scanf("%s", buffer);
    printf("Bitte geben Sie Ihr Alter ein: ");
    scanf("%d", &value);

    // Ausgabe
    printf("%s ist %d Jahre alt.\n", buffer, value);

    return 0;
}
```



```
Bitte geben Sie Ihren Namen ein: Manfred
Bitte geben Sie Ihr Alter ein: 61
Manfred ist 61 Jahre alt.
```



# Der Header `stdio.h`

## Die `gets()` Funktion

```
char * gets ( char * str );
```

- Die Funktion hat genau einen (`str`) Parameter
- Die `gets()` Funktion liest eine Zeile von `stdin` in einen Datenbereich
- Die Funktion ist **deprecated**, d.h. sie wird aufgrund der Gefahr eines Pufferüberlaufs nicht mehr empfohlen

```
#include <stdio.h>

int main()
{
    // Datenbereiche anlegen
    char buffer[128];
    printf("Bitte geben Sie Ihren Namen ein: ");
    gets(buffer);

    printf("Herzlich Willkommen: %s\n", buffer);

    return 0;
}
```



```
warning: this program uses gets(), which is unsafe.
Bitte geben Sie Ihren Namen ein: Benjamin Kormann
Herzlich Willkommen: Benjamin Kormann
```

# Der Header `stdio.h`

## Die Funktionen auf Standardobjekten und beliebigen Streams

### Funktionen Standardobjekten

- `int printf ( const char * format, ... );`

Formatierte Ausgabe auf `stdout`

- `int scanf ( const char * format, ... );`

Formatierte Eingabe von `stdin`

- `char * gets ( char * str );`

→ **nicht mehr verwenden!**

### Funktionen auf beliebigen Streams

- `int fprintf ( FILE * stream, const char * format, ... );`

Formatierte Ausgabe auf `stream`

- `int fscanf ( FILE * stream, const char * format, ... );`

Formatierte Eingabe von `stream`

- `char * fgets ( char * str, int num, FILE * stream );`

Vorteil, da mit dem Parameter `num` die maximale Anzahl der Zeichen angegeben werden kann

→ kein Pufferüberlauf mehr möglich

- Wenn bei den Funktionen auf beliebigen Streams `stdout` bzw. `stdin` als Wert des Parameters `stream` verwendet wird, so sind diese identisch zu den Funktionen auf Standardobjekten.
- Einziger Unterschied bei `fgets()`, da diese Funktion einen weiteren Parameter (`num`) hat und dadurch Pufferüberläufe im Vergleich zu `gets()` vermeidet.
- Die Funktionen auf beliebigen Streams werden auch für die Dateibearbeitung benötigt.

# Der Header `stdio.h`

## Die `sprintf()` Funktion

```
int sprintf ( char * str, const char * format, ... );
```

- Die Funktion hat mindestens `str` und `format` als Parameter
- Die `sprintf()` Funktion schreibt formatierte Daten in eine Zeichenkette
- Der Parameter `str` ist die Zeichenkette, die erstellt werden soll
- Der Parameter `format` enthält die Formatanweisung (string)

```
Bitte geben Sie Ihren Nachnamen ein: Meier
Bitte geben Sie Ihr Alter ein: 44
Herr/Frau Meier ist 44 Jahre alt.
```



```
#include <stdio.h>

int main()
{
    // Datenbereiche anlegen
    char buffer[128];
    int age = 0;

    // Daten einlesen
    fprintf(stdout, "Bitte geben Sie Ihren Nachnamen ein: ");
    fscanf(stdin, "%s", buffer);
    fprintf(stdout, "Bitte geben Sie Ihr Alter ein: ");
    fscanf(stdin, "%d", &age);

    // Formatieren der eingegebenen Werte
    char message[128];
    sprintf(message, "Herr/Frau %s ist %d Jahre alt.", buffer, age);

    // Ausgabe der formatierten Nachricht
    printf("%s\n", message);

    return 0;
}
```

# Der Header `stdio.h`

## Die `sscanf()` Funktion

```
int sscanf ( const char * s, const char * format, ... );
```

- Die Funktion hat mindestens `s` und `format` als Parameter
- Die `sscanf()` Funktion „zerlegt“ formatierte Daten in einer Zeichenkette in einzelne Daten
- Der Parameter `s` ist die Zeichenkette, die gelesen und ausgewertet werden soll
- Der Parameter `format` enthält die Formatanweisung (string)

Wochentag: Dienstag  
Temperatur: 17



```
#include <stdio.h>
```

```
int main()  
{
```

```
    // Datenbereiche anlegen
```

```
    char day[128];
```

```
    int temperature = 0;
```

```
    // Formatierte Zeichenketten
```

```
    char* text = "Dienstag werden 17 Grad in München erwartet.";
```

```
    // Informationen aus Zeichenkette extrahieren
```

```
    sscanf(text, "%s %*s %d", day, &temperature);
```

```
    // Ausgabe der extrahierten Informationen
```

```
    printf("Wochentag: %s\n", day);
```

```
    printf("Temperatur: %d\n", temperature);
```

```
    return 0;
```

```
}
```

\* bewirkt, dass die Zeichen gelesen, aber für die Auswertung ignoriert werden.

# Linux Prozesse - Allgemein

## Übersicht der wichtigen Prozessinformationen mit `htop`

pi@raspberrypi: ~/work

Datei Bearbeiten Reiter Hilfe

CPU[| 0.7%] Tasks: 68, 74 thr; 1 running  
Mem[||||| 218M/1.97G] Load average: 0.08 0.11 0.05  
Swp[| 0K/2.10G] Uptime: 04:58:32

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
6612	pi	20	0	88644	30548	24728	S	0.7	1.5	0:01.90	lxterminal
912	root	20	0	134M	57192	28948	S	0.7	2.8	2:00.72	/usr/lib/xorg/Xorg :0 -seat seat0 -auth /var
8028	pi	20	0	9236	3692	3048	R	0.0	0.2	0:01.17	htop
1024	pi	20	0	21980	2036	1712	S	0.0	0.1	0:24.41	/usr/bin/VBoxClient --draganddrop
1031	pi	20	0	21980	2036	1712	S	0.0	0.1	0:24.40	/usr/bin/VBoxClient --draganddrop
426	root	20	0	13444	4652	4252	S	0.0	0.2	0:00.06	/sbin/wpa_supplicant -u -s -O /run/wpa_suppl
390	nobody	20	0	4724	2556	2388	S	0.0	0.1	0:03.46	/usr/sbin/thd --triggers /etc/triggerhappy/t
904	root	20	0	32964	2548	2200	S	0.0	0.1	0:03.42	/usr/sbin/VBoxService --pidfile /var/run/vbo
947	root	20	0	134M	57192	28948	S	0.0	2.8	0:11.76	/usr/lib/xorg/Xorg :0 -seat seat0 -auth /var
1056	pi	20	0	161M	36632	28680	S	0.0	1.8	0:07.65	lxpanel --profile LXDE-pi
1050	pi	20	0	69296	18844	14544	S	0.0	0.9	0:02.05	openbox --config-file /home/pi/.config/openb
542	ntp	20	0	9552	3076	2672	S	0.0	0.1	0:01.19	/usr/sbin/ntpd -p /var/run/ntpd.pid -g -u 10
907	root	20	0	32964	2548	2200	S	0.0	0.1	0:02.00	/usr/sbin/VBoxService --pidfile /var/run/vbo
1121	rtkit	RT	1	23600	2680	2500	S	0.0	0.1	0:00.05	/usr/lib/rtkit/rtkit-daemon
1059	pi	20	0	93472	28140	21188	S	0.0	1.4	0:06.65	pcmanfm --desktop --profile LXDE-pi
909	root	20	0	32964	2548	2200	S	0.0	0.1	0:00.14	/usr/sbin/VBoxService --pidfile /var/run/vbo
376	root	20	0	8184	4692	1596	S	0.0	0.2	0:01.66	/usr/sbin/haveged --Foreground --verbose=1 -
957	pi	20	0	56496	12872	11604	S	0.0	0.6	0:00.98	/usr/bin/lxsession -s LXDE-pi -e LXDE
8069	pi	20	0	9320	4144	3324	S	0.0	0.2	0:00.01	bash
410	messagebu	20	0	7072	4204	3456	S	0.0	0.2	0:01.14	/usr/bin/dbus-daemon --system --address=syst
8030	pi	20	0	9604	4400	3336	S	0.0	0.2	0:00.02	bash
1116	rtkit	21	1	23600	2680	2500	S	0.0	0.1	0:00.12	/usr/lib/rtkit/rtkit-daemon
1890	root	20	0	56488	8468	7600	S	0.0	0.4	0:00.25	/usr/lib/upower/upowerd

F1 Help F2 Setup F3 Search F4 Filter F5 Tree F6 SortBy F7 Nice F8 Nice + F9 Kill F10 Quit

# Linux Prozesse - Allgemein

## Übersicht der wichtigen Prozessinformationen mit `htop`

Prozess-ID der vorhandenen Prozesse

Benutzer, der den Prozess ausführt

Prozentuale CPU-Nutzung

Prozentuale Speichernutzung

Kommando des Prozesses

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
6612	pi	20	0	88644	30548	24728	S	0.7	1.5	0:01.90	lxterminal
912	root	20	0	134M	57192	28948	S	0.7	2.8	2:00.72	/usr/lib/xorg/Xorg :0 -seat seat0 -auth /var
8028	pi	20	0	9236	3692	3048	R	0.0	0.2	0:01.17	htop
1024	pi	20	0	21980	2036	1712	S	0.0	0.1	0:24.41	/usr/bin/VBoxClient --draganddrop
1031	pi	20	0	21980	2036	1712	S	0.0	0.1	0:24.40	/usr/bin/VBoxClient --draganddrop
426	root	20	0	13444	4652	4252	S	0.0	0.2	0:00.06	/sbin/wpa_supplicant -u -s -O /run/wpa_suppl
390	nobody	20	0	4724	2556	2388	S	0.0	0.1	0:03.46	/usr/sbin/thd --triggers /etc/triggerhappy/t
904	root	20	0	32964	2548	2200	S	0.0	0.1	0:03.42	/usr/sbin/VBoxService --pidfile /var/run/vbo
947	root	20	0	134M	57192	28948	S	0.0	2.8	0:11.76	/usr/lib/xorg/Xorg :0 -seat seat0 -auth /var
1056	pi	20	0	161M	36632	28680	S	0.0	1.8	0:07.65	lxpanel --profile LXDE-pi
1050	pi	20	0	69296	18844	14544	S	0.0	0.9	0:02.05	openbox --config-file /home/pi/.config/openb
542	ntp	20	0	9552	3076	2672	S	0.0	0.1	0:01.19	/usr/sbin/ntpd -p /var/run/ntpd.pid -g -u 10
907	root	20	0	32964	2548	2200	S	0.0	0.1	0:02.00	/usr/sbin/VBoxService --pidfile /var/run/vbo
1121	rtkit	RT	1	23600	2680	2500	S	0.0	0.1	0:00.05	/usr/lib/rtkit/rtkit-daemon
1059	pi	20	0	93472	28140	21188	S	0.0	1.4	0:06.65	pcmanfm --desktop --profile LXDE-pi
909	root	20	0	32964	2548	2200	S	0.0	0.1	0:00.14	/usr/sbin/VBoxService --pidfile /var/run/vbo
376	root	20	0	8184	4692	1596	S	0.0	0.2	0:01.66	/usr/sbin/haveged --Foreground --verbose=1 -
957	pi	20	0	56496	12872	11604	S	0.0	0.6	0:00.98	/usr/bin/lxsession -s LXDE-pi -e LXDE
8069	pi	20	0	9320	4144	3324	S	0.0	0.2	0:00.01	bash
410	messagebu	20	0	7072	4204	3456	S	0.0	0.2	0:01.14	/usr/bin/dbus-daemon --system --address=syst
8030	pi	20	0	9604	4400	3336	S	0.0	0.2	0:00.02	bash
1116	rtkit	21	1	23600	2680	2500	S	0.0	0.1	0:00.12	/usr/lib/rtkit/rtkit-daemon
1890	root	20	0	56488	8468	7600	S	0.0	0.4	0:00.25	/usr/lib/upower/upowerd

F1 Help F2 Setup F3 Search F4 Filter F5 Tree F6 SortBy F7 Nice F8 Nice + F9 Kill F10 Quit

# Linux Prozesse - Prozessinformationen

## Abfragen der Prozess-IDs

```
#include <stdio.h>
#include <unistd.h> // Header für getpid(), getppid()

int main()
{
    // Abfrage der Prozess Identifikation
    pid_t pid = getpid();
    printf("PID: %d\n", pid);

    // Abfrage der Eltern Prozess Identifikation
    pid_t ppid = getppid();
    printf("PPID: %d\n", ppid);

    // Programmende
    return 0;
}
```



Ausgabe

```
PID: 64373
PPID: 64088
```

Prozess-ID der  
Konsole (Shell)

## Beschreibung der Funktionen

- `pid_t getpid()`
  - Gibt die Prozess-ID des aufrufenden Prozesses zurück
  - Häufige Verwendung für Eindeutigkeit in einem Dateinamen
- `pid_t getppid()`
  - Gibt die Prozess-ID des übergeordneten (Eltern) Prozesses des aufrufenden Prozesses zurück
  - Meist in Verwendung mit `fork()`



# Linux Prozesse - Prozessinformationen

## Verwenden der Prozess-ID zum Beenden eines Programms

```
#include <stdio.h>
#include <unistd.h>

int main()
{
    // Abfrage der Prozess Identifikation
    pid_t pid = getpid();
    printf("PID: %d\n", pid);


    // Endlosschleife
    while(1)
    {
        ;
    }

    // Programmende
    return 0;
}
```

Konsole

```
$ gcc pid-example.c -o pid_example
$ ./pid_example
PID: 8409
Beendet
```

### Ausschnitt aus htop



PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
8409	pi	20	0	2392	496	452	R	100.0	0.0	0:10.29	./pid_example
1031	pi	20	0	21980	2036	1712	S	0.7	0.1	0:27.36	/usr/bin/VBoxClient --draganddrop
8404	pi	20	0	9124	3444	2944	R	0.0	0.2	0:00.38	htop
1890	root	20	0	56488	8468	7600	S	0.0	0.4	0:00.30	/usr/lib/upower/upowerd
912	root	20	0	136M	58480	28900	S	0.0	2.8	2:03.36	/usr/lib/xorg/Xorg :0 -seat seat0 -auth /var
1024	pi	20	0	21980	2036	1712	S	0.0	0.1	0:27.35	/usr/bin/VBoxClient --draganddrop
6612	pi	20	0	88644	30676	24792	S	0.0	1.5	0:09.91	lxterminal
911	root	20	0	32964	2548	2200	S	0.0	0.1	0:00.97	/usr/sbin/VBoxService --pidfile /var/run/vbo
907	root	20	0	32964	2548	2200	S	0.0	0.1	0:02.65	/usr/sbin/VBoxService --pidfile /var/run/vbo
947	root	20	0	136M	58480	28900	S	0.0	2.8	0:11.87	/usr/lib/xorg/Xorg :0 -seat seat0 -auth /var
390	nobody	20	0	4724	2556	2388	S	0.0	0.1	0:03.55	/usr/sbin/thd --triggers /etc/triggerhappy/t
376	root	20	0	8184	4692	1596	S	0.0	0.2	0:01.74	/usr/sbin/haveged --Foreground --verbose=1 -
1056	pi	20	0	161M	36632	28680	S	0.0	1.8	0:08.84	lxpanel --profile LXDE-pi
904	root	20	0	32964	2548	2200	S	0.0	0.1	0:04.50	/usr/sbin/VBoxService --pidfile /var/run/vbo
1050	pi	20	0	69296	18844	14544	S	0.0	0.9	0:02.08	openbox --config-file /home/pi/.config/openb
1059	pi	20	0	93472	28116	21164	S	0.0	1.4	0:06.92	pcmanfm --desktop --profile LXDE-pi
410	messagebu	20	0	7072	4204	3456	S	0.0	0.2	0:01.41	/usr/bin/dbus-daemon --system --address=syst
957	pi	20	0	56496	12872	11604	S	0.0	0.6	0:01.03	/usr/bin/lxsession -s LXDE-pi -e LXDE
542	ntp	20	0	9552	3076	2672	S	0.0	0.1	0:01.45	/usr/sbin/ntpd -p /var/run/ntpd.pid -g -u 10

Konsole

Signal  
SIGTERM

```
$ kill 8409
$
```

Beendet den Prozess  
pid\_example (PI: 8409)



# Zusammenfassung

## Definition der Ein-/Ausgabe in C

- Zentrale Funktionen sind in der Header Datei `stdio.h` implementiert
- Header wird über `#include <stdio.h>` eingebunden
- Ein-/Ausgabe erfolgt über sog. `streams`

## Der FILE Pointer

- Ein-/Ausgabe in C wird über den `FILE` Pointer organisiert
- Standardobjekte zur Ein-/Ausgabe sind `stdin`, `stdout`, `stderr`
- Für die Ein-/Ausgabe stehen verschiedene Funktionen zur Verfügung

## Wichtig bei der Bearbeitung von Ein-/Ausgabedaten

- Durch die Verwendung unsicherer Funktionen können Sicherheitslücken entstehen
- Alle Funktionen für `streams` können für die Dateibearbeitung als auch für die Standardobjekte genutzt werden