



# ***Betriebssysteme***

## **Tafelübung 1. Prozesse**

<https://ess.cs.tu-dortmund.de/DE/Teaching/SS2019/BS/>

---

**Horst Schirmeier**

[horst.schirmeier@tu-dortmund.de](mailto:horst.schirmeier@tu-dortmund.de)  
<https://ess.cs.tu-dortmund.de/~hsc>





# Agenda

- Besprechung Aufgabe 0
- Fortsetzung Grundlagen C-Programmierung
- Informationen zu Aufgabe 1: Prozesse verwalten
  - Tastatureingaben mit **scanf(3)**
  - Vergleiche von Strings mit **strncmp(3)**
  - Fehlerbehandlung von Standardbibliotheksfunktionen
  - Unix-Prozessmanagement:
    - **fork(2)**
    - **exec1p(3)**
    - **wait(2)**
    - **waitpid(2)**



# Besprechung Aufgabe 0

- Separater Foliensatz Besprechung Aufgabe 0



# Grundlagen C-Programmierung

- Separater Foliensatz C-Einführung (Folie 24-40)



# Standardkonformer Code

- ANSI-C: verschiedene Versionen (C89, C90, C99, C11)
  - C-Compiler versuchen, sich daran zu halten  
→ **portable** Programme möglich
- POSIX (Portable Operating System Interface)
  - Standardisierung der Betriebssystemschnittstelle
  - beinhaltet auch Shell und Hilfsprogramme wie **grep** oder **cc**
  - erlaubt Compilieren unter allen POSIX-konformen Betriebssystemen
    - z.B. Solaris, Darwin (nur „weitgehend“ konform: Linux, OpenBSD)
- GCC-Parameter zur C-Standardkonformität:
  - `-std=c11`
  - `-Wpedantic`
  - siehe **gcc(1)**



# Tastatureingaben mit scanf(3)

- liest Zeichen aus dem Eingabestrom (z.B. von der Tastatur) und konvertiert Teile davon in Variablenwerte
- kehrt zurück, wenn Formatstring abgearbeitet ist, oder wegen eines unpassenden Zeichens abgebrochen wurde
- benötigt **#include <stdio.h>**
- Parameter
  - Formatstring wie bei printf() mit Umwandlungsspezifikatoren
  - Zeiger auf die einzulesenden Variablen, Datentypen entsprechend der Reihenfolge im Formatstring
- Rückgabewert:
  - Anzahl der erfolgreichen Umwandlungen

```
int scanf(Formatstring, Variablenzeiger1, Variablenzeiger2, ...);
```



# scanf – Formatstring

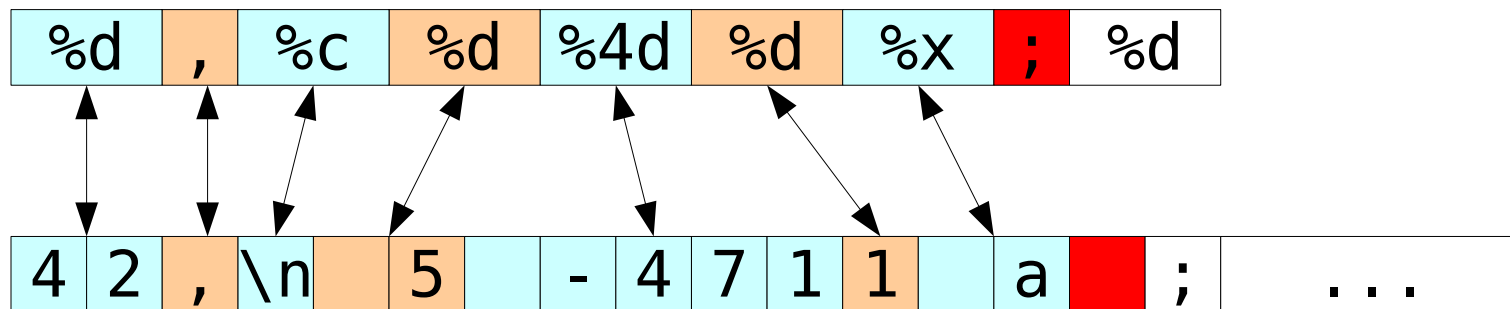
- Formatstring kann **Umwandlungsspezifikatoren**, **Whitespace** und **Zeichen** enthalten
- Die Bearbeitung eines **Umwandlungsspezifikators** endet
  - bei Erreichen unpassender Zeichen (z.B. Buchstabe bei %d)
  - bei Erreichen der maximalen Feldbreite (z.B. %10d)
- **Whitespace** (Newline, Tab, Space)
  - bedeutet, dass an dieser Stelle im Eingabestrom beliebig viel Whitespace auftauchen darf (aber nicht muss)
  - hat nur eine Wirkung vor Zeichen und nichtnumerischen Umwandlungen
- Zeichen müssen genau so im Eingabestrom auftauchen, sonst Abbruch



# scanf – Formatstring

- Beispiel

Formatstring



Eingabestrom

- Achtung Endlosschleife!

```
printf("Bitte eine Zahl eingeben> ");
while (scanf("%d",&zahl)<1) {
    printf("Nochmal:");
}
```





# scanf mit int – Beispiel

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

```
int main() {
    int eastwood;
    printf("Bitte eine ganze Zahl eingeben> ");

    if (scanf("%d",&eastwood)<1) {
        printf("Fehler bei scanf!\n");
        return 1;
    }

    printf("Die Zahl ist %d.\n",eastwood);
    return 0;
}
```

```
streic00@lithium:~/example$ ./scanf_example.elf
Bitte eine ganze Zahl eingeben> 42
Die Zahl ist 42.
```

```
streic00@lithium:~/example$ ./scanf_example.elf
Bitte eine ganze Zahl eingeben> Pferd
Fehler bei scanf!
```

```
streic00@lithium:~/example$ ./scanf_example.elf
Bitte eine ganze Zahl eingeben> 42Pferd
Die Zahl ist 42.
```



# scanf mit strings – Beispiel

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

```
int main() {  
    char guevara[42];  
    printf("Bitte ein Tier eingeben> ");  
  
    if (scanf("%41s", guevara) < 1) {  
        printf("Fehler bei scanf!\n");  
        return 1;  
    }  
  
    printf("Das Tier ist: %s.\n", guevara);  
    return 0;  
}
```

```
streic00@lithium:~/example$ ./scanf_example.elf  
Bitte ein Tier eingeben> Pferd  
Das Tier ist: Pferd.
```

```
streic00@lithium:~/example$ ./scanf_example.elf  
Bitte ein Tier eingeben> 42  
Das Tier ist: 42
```



# scanf – Sonstiges

- scanf liest nur soviele Eingaben, wie angegeben werden
- Der Rest liegt noch an der Standardeingabe an...
- ... und wird von nachfolgenden scanf-Aufrufen gelesen
- Beispiel:

```
scanf("%s", buffer1); // Eingabe "a b"  
scanf("%s", buffer2); // Keine Eingabe möglich, da "b" noch anliegt  
printf("%s", buffer2); // Ausgabe "b"
```

- Eingabestring leeren durch **getchar(2)**

```
scanf("%s", buffer1); // Eingabe "a b"  
while (getchar() != '\n')  
    ;  
scanf("%s", buffer2); // Eingabe "c"  
printf("%s", buffer2); // Ausgabe "c"
```



# Vergleich von Strings mit strncmp(3)

- strcmp vergleicht zwei Strings
  - bei Gleichheit: Rückgabewert 0, bei Ungleichheit: größer oder kleiner 0
  - Vergleicht nur die ersten *n* Zeichen (üblicherweise Buffergröße)

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

int main() {
    char tier[42];
    printf("Bitte ein Tier eingeben> ");

    if (scanf("%41s", tier) < 1) {
        printf("Fehler bei scanf!\n");
        return 1;
    }

    if ((strcmp("Pferd", tier, 42) != 0)) {
        printf("%s ist kein Pferd!\n", tier);
        return 1;
    }

    return 0;
}
```

```
mm@ios:~/example$ ./strcmp
Bitte ein Tier eingeben> Hund
Hund ist kein Pferd!
```



# Unterschiede: Strings vs. Chars

- **char** verhält sich eher wie **int**
  - Kann man mit ==, <, > usw. vergleichen
  - Nimmt man in scanf mit %c entgegen

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

int main() {
    char tier[42];
    printf("Bitte ein Tier

    if (scanf("%41s", tier) <
        printf("Fehler bei
        return 1;
    }

    if ((strncmp("Pferd", ti
        printf("%s ist kein P
        return 1;
    }

    return 0;
}
```

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

int main() {
    char buchstabe;
    printf("Buchstaben eingeben> ");

    if (scanf("%c", &buchstabe) < 1) {
        printf("Fehler bei scanf!\n");
        return 1;
    }

    if (buchstabe != 'P') {
        printf("%c ist kein P!\n", buchstabe);
        return 1;
    }

    return 0;
}
```



# Fehlerbehandlung

- Insbesondere bei Systemcalls können Fehler auftreten (ähnlich Exceptions)
- Wie geht man in C mit solchen Fehlern um?
  - C kennt keine Exceptions
  - Rückgabewert nutzen?

→ globale Variable: errno

- typisches Schema:

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

void main() {
    while (someSystemCall()==-1) {
        /*Spezialfaelle behandeln*/
        if (errno==EINTR) continue;
        /*allgemeiner Fall*/
        perror("someSystemCall");
        exit(EXIT_FAILURE);
    }
    /* alles ok, weitermachen */

    /* ... */

    return 0;
}
```



# Fehlerbehandlung

- **int** **errno**    **#include** <errno.h>
  - enthält nach einem gescheiterten Bibliotheksaufruf den Fehlercode
  - kann ansonsten beliebigen Inhalt haben!
  - Fehler wird *meistens* durch den Rückgabewert -1 angezeigt (manpage zum Systemcall lesen!)
- **void** **perror**(**const char** \*s)    **#include** <stdio.h>
  - gibt zum Fehlercode passende Fehlermeldung *auf dem Standard-Fehlerkanal* aus
- **void** **exit**(**int** status)    **#include** <stdlib.h>
  - setzt den Exit-Code und beendet das Programm augenblicklich

```
someSystemCall();  
if (errno) {
```

**FALSCH**



## execvp (3)

- `int execvp(const char *path, const char *arg, ...);`
  - `#include <unistd.h>`
  - Überschreibt die Prozessdaten im Speicher durch Daten aus Datei
  - Prozess wird sofort neu gestartet
  - Identisches Prozessobjekt, PID bleibt gleich
  - Im Fehlerfall: Rückgabewert -1 und `errno` wird gesetzt.

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

int main() {
    char befehl[42];
    if (scanf("%41s", befehl) < 1) /* Begrenze die Puffernutzung. */
        return 1;
    execvp(befehl, befehl, NULL); /* Kehrt niemals zurück. */
    return 1;                    /* Falls doch: Fehler.*/
}
```





## fork (2)

- `pid_t fork(void)`

- `#include <unistd.h>` (`#include <sys/types.h>` für `pid_t`)

- erzeugt eine Kopie des laufenden Prozesses
  - Unterscheidung ob Kind oder Vater anhand des Rückgabewertes (-1 im Fehlerfall)

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h> /* fuer pid_t benoetigt */
int main() {
    pid_t retval;
    retval = fork();
    switch (retval) {
        case -1: perror("fork"); exit(EXIT_FAILURE);
        case 0: printf("I'm the child.\n"); break;
        default: printf("I'm the parent, my child's pid is %d.\n", retval);
    }
    return 0;
}
```



## wait (2)

- `pid_t wait(int *status);`
  - `#include <sys/types.h> #include <sys/wait.h>`
  - legt sich schlafen, wenn Kindprozesse existieren, aber keiner davon Zombie ist
  - Rückgabewert
    - -1 im Fehlerfall, bzw. wenn keine Kinder (mehr) existieren
    - ansonsten pid eines Zombies
  - Parameter
    - Zeiger auf eine Statusvariable, vordefinierte Makros zum Auslesen
  - Fehler
    - ECHILD: Prozess hat keine Kinder
    - EINTR: nichtblockiertes Signal wurde gefangen (sollte bei uns nicht auftreten)



## waitpid (2)

- `pid_t waitpid(pid_t pid, int *status, int options);`  
`#include <sys/types.h> #include <sys/wait.h>`
  - Probleme von `wait()`
    - erlöst irgendeinen Zombie von möglicherweise mehreren
    - evtl. will man sich nicht schlafen legen
  - Parameter:
    - `pid`: PID des gesuchten Zombies (-1, wenn alle Kinder gemeint sind)
    - `status`: Zeiger auf Statusvariable (wie bei `wait`)
    - `options`: 0 (keine) oder mehrere durch bitweises ODER verknüpft
  - Rückgabewert: im Prinzip wie `wait()`, aber...
    - -1, wenn es keine Kinder gibt, die `pid` entsprechen
    - 0, wenn `waitpid` sich schlafen legen würde, aber die Option `WNOHANG` gegeben ist.



# waitpid – Beispiel

- Funktion zum „Einsammeln“ toter Kindprozesse

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/wait.h>

void collect_dead_children() { /*zombies "erloesen"*/
    pid_t res;
    while ((res=waitpid(-1, NULL, WNOHANG)) > 0);
    if (res == -1)
        if (errno != ECHILD) {
            perror("waitpid");
            exit(EXIT_FAILURE);
        }
}
```

da uns der Status  
nicht  
interessiert...

vordefiniertes  
*Präprozessorsymbol*,  
mehr davon in  
man 2 waitpid



# Zusammenfassung

- `man 3 scanf`
- `man 3 strncmp`
- `man 3 errno`
- `man 3 perror`
- `man 3 exit`
- `man 3 execlp`
- `man 2 fork`
- `man 2 waitpid`