# Beispiele

## Hallo Welt reloaded

```
section .data
     msg db 'Displaying 9 stars',0xa ;Message, Aneinanderreihung von "byte"
(db) Bereichen
     len equ $ - msg  ;Laenge der Message
     s2  times 9 db '*' ; 9 x ein byte (db) mit dem Inhalt "*"


 section .text
    global _start     ;must be declared for linker

 _start:          ;startadresse
    mov  edx,len  ;Laenge nach edx
    mov  ecx,msg  ;Message nach ecx
    mov  ebx,1    ;file descriptor (stdout)
    mov  eax,4    ;system call number (sys_write)
    int  0x80     ;call kernel

    mov  edx,9    ;message length
    mov  ecx,s2   ;message to write
    mov  ebx,1    ;file descriptor (stdout)
    mov  eax,4    ;system call number (sys_write)
    int  0x80     ;call kernel

    mov  eax,1    ;system call number (sys_exit)
    int  0x80     ;call kernel
```

Hier kommt wieder der **Linux-Systemaufruf** 4 zum Einsatz. Linux-Systemaufrufe funktionieren grob folgendermaßen:

- Lege die Nummer des Systemaufrufs in das EAX-Register
- Speichere die Argumente für den Systemaufruf in den Registern EBX, ECX, usw.
- Rufe den Interrupt (80h) auf
- Das Ergebnis des Systemaufrufs wird normalerweise im EAX-Register zurückgegeben.

Um das erfolgreich zum Einsatz zu bringen, muss man wissen, was ein Systemaufruf in welchem Register erwartet, damit er funktioniert, beim System-Call 4 (Write) ist es folgendermaßen:

| Name | %eax | %ebx | %ecx | %edx | %esx | %edi |
|---|---|---|---|---|---|---|
| Write | 4 | unsigned int (Output Stream) | const char * (Inhalte) | size_t (Länge) | - | - |

Das kann man sich ein wenig wie eine Funktion/Methode mit Argumenten vorstellen: Man befüllt zunächst die Register mit den Argumenten und ruft dann den Systemaufruf auf.

### Weitere System-Calls:

| Name | %eax | %ebx | %ecx | %edx | %esx | %edi |
|---|---|---|---|---|---|---|
| Exit | 1 | int (Exit Code) | - | - | - | - |
| Fork | 2 | struct pt_regs | - | - | - | - |
| Read | 3 | unsigned int | char * | size_t | - | - |
| Write | 4 | unsigned int (Output Stream) | const char * (Inhalte) | size_t (Länge) | - | - |
| Open | 5 | const char * | int | int | - | - |
| Close | 6 | unsigned int | - | - | - | - |

## Benutzereingabe

Mit dem Sys-Call 3 lässt sich also eine Eingabe realisieren - "Read".

Damit das funktioniert muss man mehrere Dinge beachten:

- Man benötigt einen reservierten, beschreibbaren Speicherbereich im .bss-Abschnitt des Programms
- Ins Register eax muss der Wert 3 geschrieben werden
- Ins Register ecx muss die Adresse des in .bss reservierten Bereichs

From:
https://info-bw.de/ -

Permanent link:
**https://info-bw.de/faecher:informatik:oberstufe:techinf:assembler:beispiele1:start?rev=1631552335**

Last update: **13.09.2021 16:58**