



6. System Calls in Assembler und C

Laden Sie von der Kurswebseite das Quellcode-Archiv `syscalls.tgz` herunter und entpacken Sie es:

```
$ wget http://swf.hgesser.de/b1-ss2020/prakt/syscalls.tgz
$ tar xzf syscalls.tgz
```

Es entsteht ein neuer Unterordner `syscalls/`, in den Sie mit `cd syscalls` hinein wechseln.

In Assembler-Programmen rufen Sie System Calls über den Software-Interrupt `int 0x80` auf: ¹

Linux	FreeBSD
<pre>section .text global _start ; fuer den Linker (ld) _start: ; fuer Linker (wo gehts los) mov edx,len ; Nachrichtenlaenge mov ecx,msg ; Adresse der Nachricht mov ebx,1 ; file descriptor (1=stdout) mov eax,4 ; Syscall-Nr. (sys_write) int 0x80 ; Syscall ausfuehren mov eax,1 ; Syscall-Nr. (sys_exit) int 0x80 ; Syscall ausfuehren section .data msg db 'Hallo Welt!',0xa ; Text len equ \$ - msg ; Laenge</pre>	<pre>section .text global _start ; fuer den Linker (ld) _syscall: int 0x80 ; system call ret _start: ; fuer Linker (wo gehts los) push dword len ; Nachrichtenlaenge push dword msg ; Adresse der Nachricht push dword 1 ; file descriptor (1=stdout) mov eax,0x4 ; Syscall-Nr. (sys_write) call _syscall ; Syscall ausfuehren add esp,12 ; Stack aufraeumen ; (3 Argumente, Laenge 4) push dword 0 ; exit code mov eax,0x1 ; Syscall-Nr. (sys_exit) call _syscall ; Syscall ausfuehren ; nach exit nicht aufr. section .data msg db "Hallo Welt!",0xa ;Text len equ \$ - msg ;Laenge</pre>

Die allgemein übliche Unix-Variante (auf Intel-CPU's) ist die von FreeBSD: Argumente in umgekehrter Reihenfolge auf den Stack pushen, dann die Syscall-Nummer in Register EAX schreiben und einen Syscall-Interrupt auslösen (hier: `int 0x80`). Linux verwendet stattdessen die Register EBX, ECX, EDX, ESI, EDI und EBP für bis zu sechs Argumente (und ebenfalls den Software-Interrupt `0x80`). Der Rückgabewert des Syscalls steht in EAX.

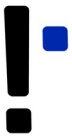
Die Linux-Variante können Sie auch in C-Programme übernehmen und definieren dafür folgende Inline-Assembler-Funktion:

```
int syscall (int eax, int ebx, int ecx, int edx) {
    int result;
    asm (
        "int $0x80"
        : "=a" (result)
        : "a" (eax), "b" (ebx), "c" (ecx), "d" (edx) );
    return result;
}
```

Diese Funktion `syscall()` erwartet dann als erstes Argument die Syscall-Nummer (wie Sie sie in der Datei `/usr/include/asm/unistd_32.h` finden, eine Kopie der Datei liegt im Archiv).

Anstelle von `exit(0)`; können Sie mit obiger Definition also auch `syscall(1,0,0,0)`; schreiben, um den aktuellen Prozess zu beenden.

¹ Quelle dieser Listings: <http://asm.sourceforge.net/intro/hello.html> (übersetzt) - Syntax für Assembler nasm geeignet



a) Betrachten Sie das folgende Programm (fork+write.c im Aufgabenarchiv):

```
int main() {
    char vater[]="Ich bin der Vater.\n";
    int vlen=sizeof(vater);
    char sohn[]="Ich bin der Sohn.\n";
    int slen=sizeof(sohn);

    int pid=fork();

    if (pid) {
        write(1,&vater,vlen);
    }
    else {
        write(1,&sohn,slen);
    }
    return 0;
}
```

Es verwendet die Systemaufrufe `fork()` und `write()`. Die Bedeutung der Argumente in `write()` entnehmen Sie der Manpage (`man 2 write`); das Argument 1 ist der File-Deskriptor (fd) für die Standardausgabe `stdout` (0: Standardeingabe `stdin`, 2: Standardfehlerausgabe `stderr`).

Ersetzen Sie im Programm die drei Aufrufe (`fork`, `write`, `write`) durch Aufrufe von `syscall()`. Die benötigten Syscall-Nummern finden Sie in der Datei `unistd_32.h`. Schreiben Sie nur auf, wodurch Sie die drei rot markierten Zeilen ersetzt haben. Prüfen Sie, ob Ihr verändertes Programm funktioniert. (Die Datei enthält bereits die Definition von `syscall()`.) Überprüfen Sie, dass Ihr Programm nach der Änderung noch genauso funktioniert wie vorher.

(Hinweis: Das funktioniert so nur unter Linux; wenn Sie das Programm z. B. unter macOS testen, führen Aufrufe von `syscall()` zur Fehlermeldung „illegal instruction“.)

b) Schreiben Sie ein C-Programm, das

- mit `creat()` eine neue Datei (mit im Programm vorgegebenen Namen) erzeugt und öffnet,
- mit `write()` den String "Hallo\n" in diese Datei schreibt,
- mit `close()` die neue Datei schließt.

Verwenden Sie dafür zunächst die angegebenen Systemaufrufe und ersetzen Sie diese anschließend durch Aufrufe von `syscall()`. Welche Parameter Sie `creat()` übergeben müssen, verrät wieder die Manpage (`man 2 creat`). Tipp: `syscall()` erwartet immer genau vier Argumente. Benötigt Ihr Syscall weniger Argumente, dann „füllen Sie mit Nullen auf“.

Falls Sie diese Aufgabe nur „auf Papier“ bearbeiten, schreiben Sie nur die wesentlichen Aufrufe auf (keine Funktionsdefinition für `main()` etc.).