Zusammenfassung Übungsgruppen

UNIX - Shell befehle

<Backspace> letztes Zeichen löschen

<STRG>-U alle Zeichen der Zeile löschen

<STRG>-C Interrupt – Programm abbrechen

<STRG>-Z Stop – Programm wird angehalten

<STRG>-D EOF (End of File)

<STRG>-S/Q Bildschirmausgabe anhalten/fortsetzen

stty(1) Terminal einstellungen vornehmen

chsh Login-Shell ändern

ls Verzeichnis auflisten.

ls -l langes Ausgabeformat

ls -a auch mit . beginnende Datein.

pwd Aktuelles Verzeichnis ausgeben

chmod Zugriffrechte ändern

cp Datei(en) kopieren

mv Datei(en) verschieben

ln Datein linken

ln -s symbloschen Link erzeugen

rm Datei(en) löschen

mkdir Verzeichniserzeugen

rmdir Verzeichnis löschen (muss leer sein!)

id, groups eigen Benuter-ID, Gruppen anzeigen

who Wer ist gerade angemeldet

ps Prozessliste ausgeben

ps -u x Prozesse des Benuters x

ps -ef alle Prozesse(-e), ausführlich (-f)

top Prozessliste, sortiert nach Ressourcenbelegung

kill <pid> Prozess “abschießen” (geordnetes terminieren, kann ignoriert werden)

kill -9 <pid> Töten, jetzt und sofort!

cat Datein ausgeben

more, less Datein bildschirmweise ausgeben

head, tail Anfang/Ende einer Datei ausgeben

pr, lp ,lpr Datei ausdrucken

wc Wordcount (Zeichen, Wörter, Zeichen)

grep, fgrep, egrep nach Mustern o. Wörtern suchen

find Dateibaum durchlaufen

sed Stream-Editor

tr Zeichen aufeinander abbilden oder löschen

awk Pattern-Scanner

cut einzelne Felder aus Zeilen ausschendein

sort sortieren

Der GNU C Compiler

gcc(1) Programmname

-Wall alle (wichtigen?) Warnungen ausgeben

-o <Ziel> Ausgabename (Executable oder Bin)

-Werror Warnungen werden als Fehler gewertet!

-ansi Standart C Code (C-90) annehmen(gcc erweiterungen aus)

-pedantic Auf genaue einhaltung des Standart achten.

-D_POSIX_SOURCE

Operatoren Reihenfolge

Achtung ist nicht Standartisiert!

C-Bibfunktionen

```
int scanf(Formatstring, Var1zeiger, Var2zeiger, ...)
```

Von der Tastatur einlesen.

```
void exit(int status)
```

Beendet das Programm, für Kritische situationen gedacht.

```
unsigned int sleep(unsigned int seconds)
```

Legt den Prozess schlafen.

```
int execlp(const char *path, const char *arg, ...)
```

Ersetzt den aufrufenden Prozess mit den zu Ladenen, PID bleibt gleich.

```
pid_t fork(void)
```

Erzeugt neues Kind durch Kopie des aufrufenden prozess. Beim Vater wird childpid zurück geben, beim Kind 0.

() [] -> .
! ~ ++ – + -(Vorzeichen) *(Zeiger) & sizeof
* / %
+ -
<< >>
< <= >= >
== !=
&
^
|
&&
||
?:
= += -= *= %= &= ^= |= <<= >>=
,

Table 1: Operatoren Vorrangregeln. Oben sind höherwertig. Auf einer Ebene sind alle gleich.

```
pid_t wait(int *status)
```

Legt sich schlafen bis ein Kind terminiert. Returnt pid vom Kind.

```
pid_t waitpid(pid_t pid, int *status, int options)
```

Wie Wait, nur das man einen bestimmen Kind, oder alle(pid=-1) wartet. Kann nichtblockiered sein mit *WNOHANG*(return 0).

C Speicherverwaltung

```
void *malloc(size_t size)
```

Dynamische Reservierung von Speicher.

```
void free(void *ptr)
```

Gibt von malloc belegten Speicher frei.

C Fehlerbehandlungsschema

```
int errno
```

gloable Variable, enthält Fehlercode nach gescheiterten Bibaufruf

```
void perror(const char *s)
```

Gibt zum aktuellen Fehlercode die passende Fehlermeldung auf dem Standart-Fehlerkanal aus.

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

void main() {
  while (someSystemCall()==-1) {
    /* Spezialfälle behandeln */
    if(errno==EINTR) continue;
    /* allgemeiner Fall */
    perror("someSystemCall");
    exit(EXIT_FAILURE);
  }
  /* alles ok, weitermachen */
  return 0;
}
```

Semaphoren unter UNIX

```
int semget(Schlüssel, Anzahl, Optionen)
```

Erzeugt eine Semaphoren**menge**. Diese sind noch uninitialisiert. Returnt *identifier*.

```
int semctl(SemID, SemNr, Kommando, ...)
```

Initialisieren und Abfrage von Semaphoren. Kommandos:

SETVAL/GETVAL Setzen/Abfragen des Semaphor SemNr

SETALL/GETALL Setzen/Anfragen aller Semaphoren in der Menge

IPC_RMID Freigeben der Semaphorenmenge

```
int semop(SemID, *sembuf, AnzahlOPs
```

Führt die Operationen atomr auf den Semaphoren aus. Struktur *sembuf* wichtig:

```
struct sembuf{
  unsigned short sem_num;
  short sem_op;
  short sem_flg; }
```

sem_op>0 Address Wert auf die Semaphore (sem_op=1⇒**v-Operation**)

sem_op=0 Warten bis Semaphore 0 wird.

sem_op<0 Wartet wenn betrag zu groß(|sem_op|<sem), oder zieht den Betrag ab und macht weiter (sem_op=-1⇒**p-Operation**).

SharedMemory unter UNIX

SharedMemory objekte werden mit einem Key angsprochen. Sie müssen an Prozesse erst *attached* werden.

```
int shmget(key_t key, size_t size, int shmflg)
```

Erzeugt ein SharedMemory und gibt ein *identifyer* zurück.

```
int shmctl(int shmid, int cmd, shmid_ds *buf)
```

Verwalten, Zerstören, Informationen abfragen eines Shared Memory.

```
void *shmat(int shmid, const void *shmaddr, int shmflg)
```

SharedMemory an den Prozess anhängen. Gibt Pointer auf den SharedMemory-Speicher zurück.

```
int shmdt(const void *shmaddr)
```

Entfernt ein SharedMemory aus dem Speicher des Prozesses.

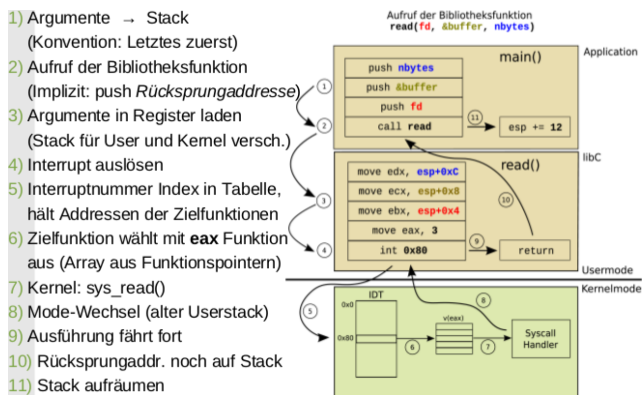
Signalhänder in UNIX

```
int sigaction(int signum, sigaction *act, sigaction *oldact)
```

Überschreibt den Standartsignalhändler. *act* ist der neue signalhändler, in *oldact* wird der alte abgelegt.

```
struct sigaction{
  //Zeiger auf Handler-Funktion
  void (*sa_handler)(int);
  //Zu ignorierende Signale
  sigset_t sa_mask;
  //Optionen
  int sa_flags;
  //Veraltet, ignorieren.
  void (*sa_restorer)(void); }
```

Systemcalls in UNIX



Dateioperationen in UNIX

Low-Level

Arbeitet mit Filedeskriptoren (einem INT). Methode ist POSIX standart.

```
int open(const char *path, int flags)
```

Gibt Dateideskriptor zurück. -1 Bei Fehler (errno)

```
ssize_t read(int fd, void *buf, size_t count)
```

Liesst aus der Datei und Anzahl der gelesen Bytes zurück.

```
ssize_t write(int fd, void *buf, size_t count)
```

Schreibt in die Datei und gibt Anzahl geschriebener Bytes zurück.

```
int close(int fd)
```

Schließt den Filedeskriptor

High-Level

Benutzt Streams als eine Abstraktion. FILE ist ein Struct. Dies ist C-standart.

```
FILE *fopen(const char *path, const char *mode)
```

Gibt FILE Stream zurück. NULL bei fehler

```
size_t fread(void *buf, size_t size, size_t count, FILE *stream)
```

Liesst aus dem Dateistrom und gibt Anzahl der gelesen Bytes zurück.

```
size_t fwrite(void *buf, size_t size, size_t count, FILE *stream)
```

Schreibt in den Dateistrom und gibt Anzahl geschriebener Bytes zurück.

```
int fclose(FILE *stream)
```

Schließt den Dateistrom