2. Tafelübung

- Schreiben portabler Programme
- Systemcalls:
  - open/read/write/close
  - lseek/chmod/stat
  - umask/utime/truncate

Portable Programme

- 1. Verwenden der standardisierten Programmiersprache ANSI-C
  - gcc-Aufrufoptionen

```
-ansi -pedantic
```

- 2. Verwenden der standardisierten Betriebssystemschnittstelle POSIX
  - gcc-Aufrufoption

```
-D_POSIX_SOURCE
```

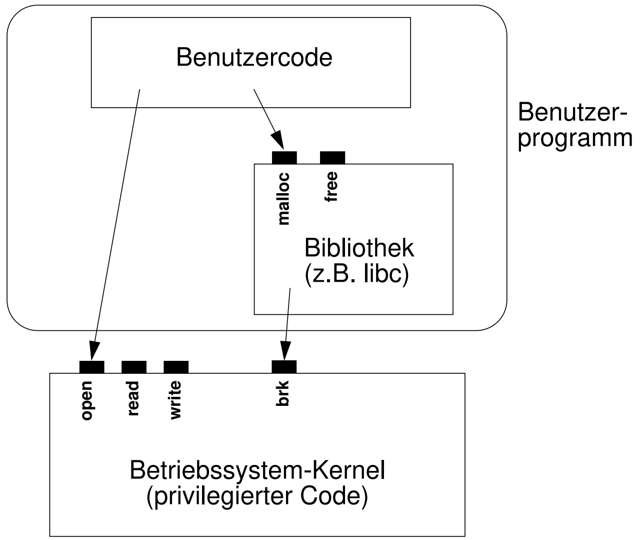
- Programm sollte sich mit folgenden gcc-Aufruf compilieren lassen

```
gcc -ansi -pedantic-errors -D_POSIX_SOURCE -Wall -Werror
```


POSIX

- Standardisierung der Betriebssystemschnittstelle:
  Portable **O**perating **S**ystem **I**nterface (IEEE Standard 10003.1)
- POSIX.1 wird von verschiedenen Betriebssystemen implementiert:
  - SUN Solaris 2.6
  - SGI Irix 6.2/6.4
  - DIGITAL Unix 4.0
  - Linux (größtenteils POSIX, zertifizierte Version von Fa. Unifix)
  - Windows NT (Posix Subsystem)
  - ...

Systemaufrufe vs. Bibliotheksaufrufe



open

- Funktions-Prototyp:

```
#include <fcntl.h>
int open(const char *path, int oflag, ... /* [mode_t mode] */ );
```

- Argumente:
  - Maximallänge von path: **PATH_MAX**
  - **oflag**: Lese/Schreib-Flags, Allgemeine Flags, Synchronisierungs I/O Flags
    - Lese/Schreib-Flags: **O_RDONLY, O_WRONLY, O_RDWR**
    - Allgemeine Flags: **O_APPEND, O_CREAT, O_EXCL, O_LARGEFILE, O_NDELAY, O_NOCTTY, O_NONBLOCK, O_TRUNC**
    - Synchronisierung: **O_DSYNC, O_RSYNC, O_SYNC**
  - **mode**: Zugriffsrechte der erzeugten Datei (nur bei **O_CREAT**) - siehe **chmod**
- Rückgabewert
  - Filedeskriptor oder -1 im Fehlerfall (**errno** wird gesetzt)

open - Flags

- **O_EXCL**: zusammen mit **O_CREAT** - nur *neue* Datei anlegen
- **O_NDELAY, O_NONBLOCK**: Operationen arbeiten nicht-blockierend (bei Pipes, FIFOs und Devices)
  - open kehrt sofort zurück
  - read liefert -1 zurück, wenn keine Daten verfügbar sind
  - wenn genügend Platz ist, schreibt write alle Bytes, sonst schreibt write nichts und kehrt mit -1 zurück
- **O_TRUNC**: Datei wird beim Öffnen auf 0 Bytes gekürzt
- **O_APPEND**: vor jedem Schreiben wird der Dateizeiger auf das Dateiende gesetzt
- **O_NOCTTY**: beim Öffnen von Terminal-Devices wird das Device nicht zum Kontroll-Terminal des Prozesses

open Flags (2)

- Synchronisierung
  - **O_DSYNC**: Schreibaufruf kehrt erst zurück, wenn Daten in Datei geschrieben wurden (Blockbuffer Cache!!)
  - **O_SYNC**: ähnlich **O_DSYNC**, zusätzlich wird gewartet, bis Datei-Attribute wie Zugriffszeit, Modifizierungszeit, auf Disk geschrieben sind
  - **O_RSYNC|O_DSYNC**: Daten die gelesen wurden, stimmen mit Daten auf Disk überein, d.h. vor dem Lesen wird der Buffercache geflushet
  - **O_RSYNC|O_SYNC**: wie **O_RSYNC|O_DSYNC**, zusätzlich Datei-Attribute

read

- Funktions-Prototyp:

```
#include <unistd.h>
ssize_t read(int fildes, void *buf, size_t nbyte);
```

- Argumente
  - **fildes**: Filedeskriptor, z.B. Rückgabe vom open-Aufruf
  - **buf**: Zeiger auf Puffer
  - **nbyte**: Größe des Puffers
- Rückgabewert
  - Anzahl der gelesenen Bytes oder -1 im Fehlerfall

```
char buf[1024];
int fd;
fd = open("/etc/passwd", O_RDONLY);
if (fd == -1) ...
read(fd, buf, 1024);
```

write

- Funktions-Prototyp

```
#include <unistd.h>
ssize_t write(int fildes, const void *buf, size_t nbyte);
```

- Argumente
  - äquivalent zu **read**
- Rückgabewert
  - Anzahl der geschriebenen Bytes oder -1 im Fehlerfall

close

- Funktions-Prototyp:

```
#include <unistd.h>
int close(int fildes);
```

- Argumente:
  - **fildes**: Filedeskriptor der zu schließenden Datei
- Rückgabewert:
  - 0 bei Erfolg, -1 im Fehlerfall

lseek

- Funktions-Prototyp

```
#include <unistd.h>
off_t lseek(int fildes, off_t offset, int whence);
```

- Argumente
  - **fildes**: Filedeskriptor
  - **offset**: neuer Wert des Dateizeigers
  - **whence**: Bedeutung von offset
    - **SEEK_SET**: absolut vom Dateianfang
    - **SEEK_CUR**: Inkrement vom aktuellen Stand des Dateizeigers
    - **SEEK_END**: Inkrement vom Ende der Datei
- Rückgabewert
  - Offset in Bytes vom Beginn der Datei oder -1 im Fehlerfall

chmod

- Funktions-Prototyp:

```
#include <sys/stat.h>
int chmod(const char *path, mode_t mode);
```

- Argumente:
  - **path**: Dateiname
  - **mode**: gewünschter Dateimodus, z.B.
    - **S_IRUSR**: lesbar durch Besitzer
    - **S_IWUSR**: schreibbar durch Benutzer
    - **S_IRGRP**: lesbar durch Gruppe
- Rückgabewert: 0 wenn OK, -1 wenn Fehler
- Beispiel:

```
chmod("/etc/passwd", S_IRUSR | S_IRGRP);
```

stat / lstat

- Funktions-Prototyp:

```
#include <sys/types.h>
#include <sys/stat.h>
int stat(const char *path, struct stat *buf);
int lstat(const char *path, struct stat *buf);
```

- Argumente:
  - **path**: Dateiname
  - **buf**: Puffer für Inode-Informationen
- Rückgabewert: 0 wenn OK, -1 wenn Fehler
- Beispiel:

```
struct stat buf;
stat("/etc/passwd", &buf); /* Fehlerabfrage ... */
printf("Inode-Nummer: %d\n", buf.st_ino);
```


stat / lstat: stat-Struktur

- **dev_t st_dev;** Gerätenummer
- **ino_t st_ino;** Inodenummer
- **ushort st_fstype;** Dateisystem-Typ (siehe **sysfs(2)**)
- **ushort st_mode;** Dateimode, u.a. Zugriffs-Bits (siehe **chmod(1)**)
- **ushort st_nlink;** Anzahl der (Hard-) Links auf den Inode
- **uid_t st_uid;** UID des Besitzers
- **gid_t st_gid;** GID der Dateigruppe
- **dev_t st_rdev;** DeviceID, nur für Character oder Blockdevices
- **off_t st_size;** Dateigröße in Bytes
- **time_t st_atime;** Zeit des letzten Zugriffs (in Sekunden seit 1.1.1970)
- **time_t st_mtime;** Zeit der letzten Veränderung (in Sekunden ...)
- **time_t st_ctime;** Zeit der letzten Änderung der Inode-Information (...)
- **long st_blksize;** Blockgröße des Dateisystems

umask

- Funktions-Prototyp:

```
#include <sys/stat.h>
mode_t umask(mode_t cmask);
```

- Argumente
  - **cmask**: gibt Permission-Bits an, die beim Erzeugen einer Datei ausgeschaltet werden sollen
- Rückgabewert
  - voriger Wert der Maske

utime

- Funktions-Prototyp:

```
#include <utime.h>
int utime(const char *path, const struct utimbuf *times);
```

- Argumente
  - **path**: Dateiname
  - **times**: Zugriffs- und Modifizierungszeit (in Sekunden)
- Rückgabewert: 0 wenn OK, -1 wenn Fehler
- Beispiel: setze atime und mtime um eine Stunde zurück

```
struct utimebuf times;
struct stat buf;
stat("/etc/passwd", &buf); /* Fehlerabfrage */
times.actime = buf.st_atime - 60 * 60;
times.modtime = buf.st_mtime - 60 * 60;
utime("/etc/passwd", &times); /* Fehlerabfrage */
```

truncate

- Funktions-Prototyp:

```
#include <unistd.h>
int truncate(const char *path, off_t length);
```

- Argumente:
  - ◆ **path**: Dateiname
  - ◆ **length**: gewünschte Länge der Datei
- Rückgabewert: 0 wenn OK, -1 wenn Fehler

POSIX I/O vs. Standard-C-I/O

- POSIX Funktionen open/close/read/write/... arbeiten mit Filedescriptoren
- Standard-C Funktionen fopen/fclose/fgets/... arbeiten mit Filepointern
- Konvertierung von Filepointer nach Filedescriptor

```
#include <stdio.h>
int fileno(FILE *stream);
```

- Konvertierung von Filedescriptor nach Filepointer

```
#include <stdio.h>
FILE *fdopen(int fd, const char* type);
```

  - ◆ type kann sein "r", "w", "a", "r+", "w+", "a+"
- Filedescriptoren in **<unistd.h>**:
  STDIN_FILENO, **STDOUT_FILENO**, **STDERR_FILENO**