Programmiertechnik 1

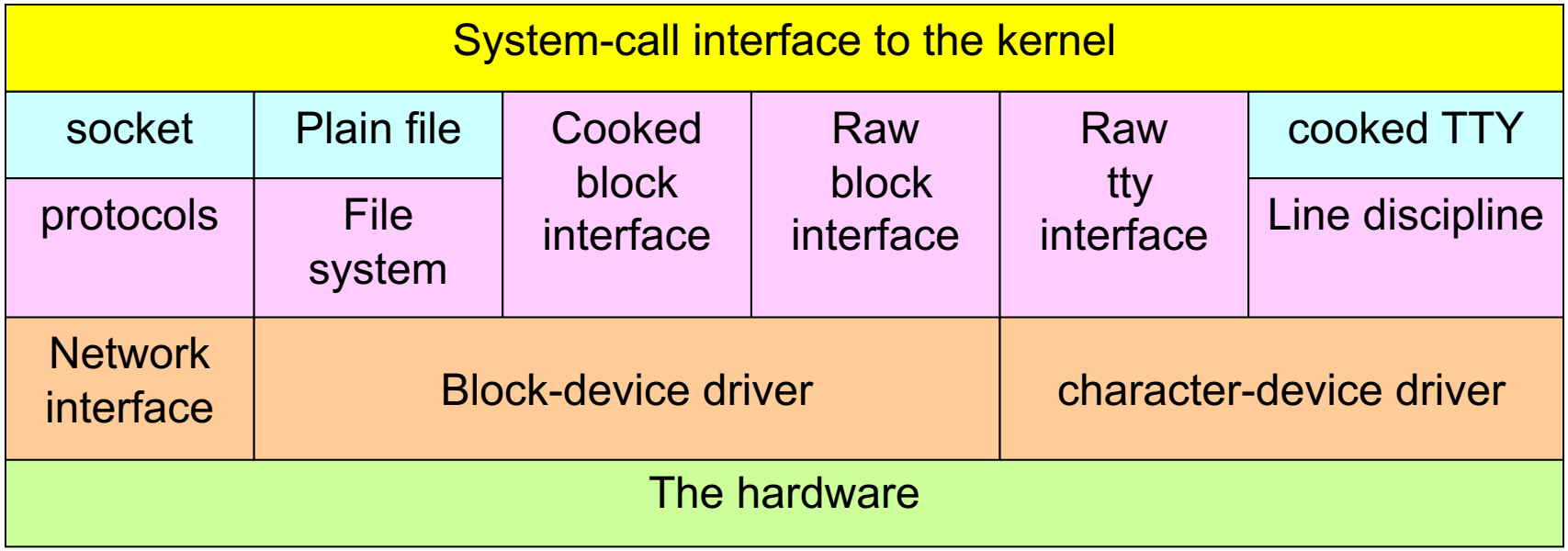
Unit 12: Programmiersprache C - Betriebssystemschnittstellen

Ablauf

- Zugriff auf Betriebssystemdienste
- Dateideskriptoren
- low-level I/O
- read and write
- open, creat, close, unlink
- lseek
- Standard library

Betriebssystemdienste

- Betriebssysteme offerieren Dienste über Systemaufrufe
  - Konzentrieren uns auf UNIX-Systemaufrufe
  - ANSI C-Bibliothek ist nach dem UNIX-Modell strukturiert
- Beispiel:
  - BSD-UNIX I/O-System



Dateideskriptoren

- Alle Ein- und Ausgabe in UNIX erfolgt über Dateien
  - Alle Peripherie-Geräte werden als Dateien dargestellt
  - Tastatur, Bildschirm, Festplatten, Netzwerkendpunkte, etc.
  - Homogene Schnittstelle für Kommunikation zwischen Programmen und Peripherie
- Dateideskriptor
  - Gültig innerhalb eines Prozesses
  - Stellt eine geöffnete Systemressource / Datei dar
- Öffnen einer Datei
  - Überprüfen der Zugriffsrechte und der Existenz einer Datei
  - Resultat ist eine kleine Integer Zahl – der Dateideskriptor
  - Vorherdefiniert – und bereits geöffnet:
    - 0 – Standard-Eingabe, 1 – Standard-Ausgabe, 2 – Standard-Fehlerausgabe
    - Zuordnung auf konkrete Geräte ist von Dateiumlenkungsoperationen der shell abhängig
    - Programm kennt konkrete Quelle und Ziel der I/O-Operationen nicht
  - Geöffnete Dateien werden vom System verwaltet
    - Programme greifen über Dateideskriptoren zu

Low level I/O

- Ein- und Ausgabe: read() und write()
  - int n_read = read(int fd, char * buf, int n);
  - int n_written = write(int fd, char * buf, int n);
  - Jeder Aufruf gibt die Zahl der übertragenes Bytes zurück
  - Rückgabewert 0 bei read → Dateiende
  - Rückgabewert -1 → Fehler
- Blockgrößen
  - Beliebige Zahl von Bytes kann übertragen werden
  - Typisch: 1 – unbuffered
  - 1024 / 4096 – physische Blockgröße der unterliegenden Geräte
  - Große Blockgrößen → weniger Systemaufrufe, höhere Effizienz
    - Anders als bei FILE * findet keine implizite Pufferung statt

copy.c

Abstraktionsebenen

- read() und write() sind Grundlage für getchar(), putchar(), etc.

```
# include <unistd.h>
# define EOF -1

int getchar(void) {
        char c;
        return ( read(0, &c, 1) == 1 ) ? (unsigned char) c : EOF;
}
```

- Details:
  - c muss vom Typ char sein → read() operiert byteweise
  - Typecast nach (unsigned char) eliminiert Probleme mit Vorzeichen (Rückgabewert)
  - Ungepufferte Operation ist ineffizient

getchar.c

Zugriff auf Dateien

- Dateien müssen vor Lese-/Schreibzugriff geöffnet werden
  - int open(char *name, int flags, int permissions);
    - Flags: O_RDONLY, O_WRONLY, O_RDWR, O_CREAT, O_TRUNC...
    - # include <fcntl.h> (<sys/file.h> on BSD Unix)
    - Permissions: 0 für existierende Dateien, 0ugo für O_CREAT
    - Öffnen einer nichtexistenten Datei ist ein Fehler (Rückgabewert -1)
    - Gibt im Erfolgsfall geöffneten Dateideskriptor zurück
  - int creat(char *name, int permissions);
    - Anlegen einer neuen Datei
    - Wenn Datei schon existiert, so wird sie auf Länge 0 abgeschnitten
    - Permissions geben Zugriffsrechte für neue Datei an
      → gefiltert durch umask-Einstellung
- Programm kann nur endlich viele Dateien gleichzeitig öffnen
  - Mindestens 20
    - getdtablesize(); _POSIX_OPEN_MAX aus <limits.h>

mycp.c

Dateideskriptoren

- Nur wenige Deskriptoren können gleichzeitig geöffnet sein
  - Müssen Deskriptoren mehrfach nutzen
  - close(int fd);
    - Schliesst einen Dateideskriptor
    - Keine Puffer-Operationen (wie bei fclose())
  - exit();
    - Schliesst implizit alle geöffneten Dateideskriptoren
- Löschen von Dateien
  - Durch Unix-Dateisystem nicht unterstützt
  - Mehrere Verweise können auf identischen Dateiinhalt zeigen
    - Hard links, ln <name2> <name1>
  - Einfernen eines Verweises
    - int unlink(char * name);
    - Nach Entfernen des letzten Verweises verschwindet Datei
- Löschen temporärer Dateien:
    - Datei öffnen (mit open(name, O_RDWR | O_CREAT, 0666))
    - Datei sofort mit unlink(name) entfernen

reverse.c

Wahlfreier Zugriff

- Freies Positionieren des Lese-/Schreibezeigers möglich
  - long lseek(int fd, long offset, int origin);
  - Gibt neue Position zurück, -1 im Fehlerfall
  - Positionieren möglich in Bezug auf:
    - Dateianfang: SEEK_SET
    - Aktuelle Position: SEEK_CUR
    - Dateiende: SEEK_END
  - Beispiel: Länge einer Datei ermitteln:
    - long len = lseek(fd, 0L, SEEK_END);
- Lese-/Schreibezeiger kann hinter Dateiende positioniert werden
  - Dateien mit „Löchern“
  - Lesen an einer unbeschriebenen Stelle liefert Null-Bytes (C2 Security)
  - Lesen nach Dateiende liefert Fehler
    - Idee: Datenbank, benutzen key zum Positionieren des Lese-/Schreibezeigers

dbase.c

Einschub: Zugriffsrechte

- Rechte können definiert werden für:
  - (u) user – Owner (creator) einer Datei
  - (g) group – Group
  - (o) other – alle anderen Nutzer des UNIX-Systems

- Beispiel:

```
luna test ( 48 )-% ls -lisa
total 2
421908   1 drwxr-xr-x    2   apolze   1024  Jan  7 15:06 .
116884   1 drwxr-xr-x   13   apolze   2048  Jan  7 15:06 ..
116992   0 -rw-------    1   apolze      0  Jan  7 15:05 Mail.txt
116991   0 -rw-rw-rw-    1   apolze      0  Jan  7 15:05 test.c
```

Zugriffsrechte (contd.)

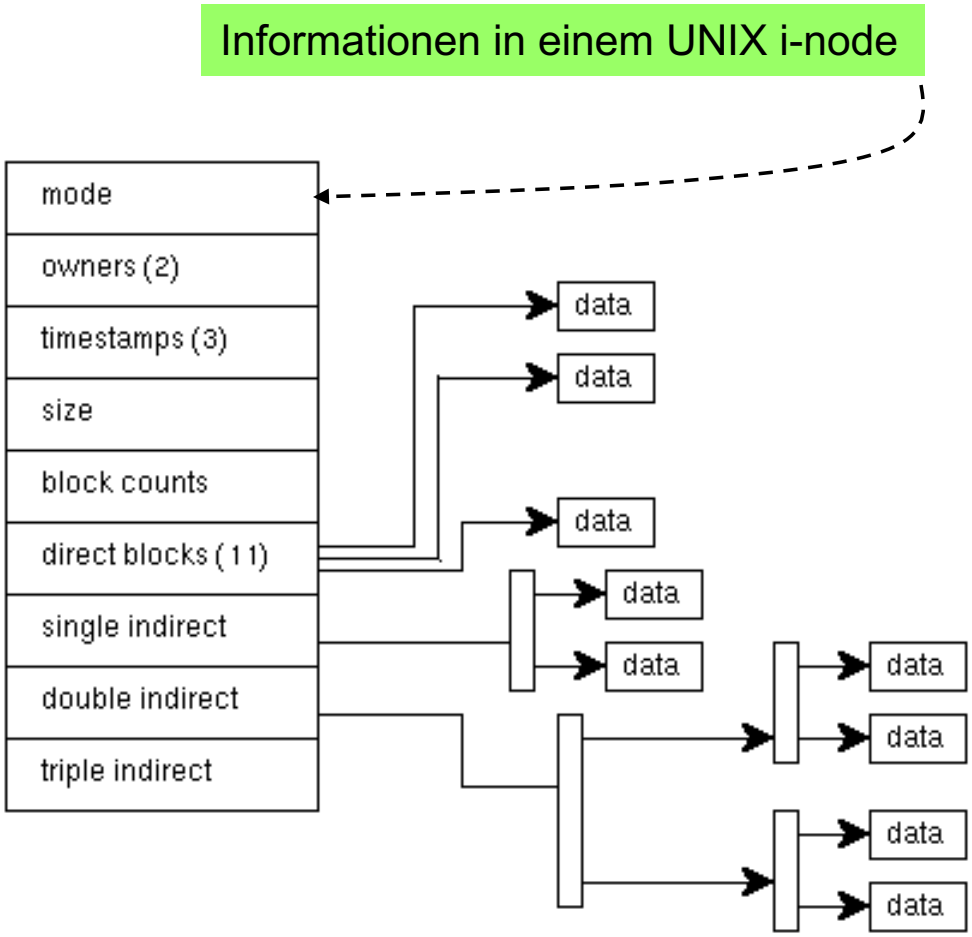
- Zugriffsrechte für eine Datei:
  - (r) Lese-Recht; Recht zu Auflisten von Verzeichnissen
  - (w) Schreibe-Recht; Schließt Recht zum „Verlängern“ und Löschen ein
  - (x) Ausführungs-Recht; Recht, Verzeichnisse zu traversieren
- Binäre Repräsentation:
  - (x): Bit 0 (+1)
  - (w): Bit 1 (+2)
  - (r): Bit 2 (+4)
- Rechte können kombiniert werden
  - Read+Write Zugriffsrecht: 6
  - Read+Execute Zugriffsrecht: 3
  - Read-only: 2

UNIX-Verzeichnisse

- Hierarchisch, Baum-strukturiert
- Verzeichnisse werden als Dateien repräsentiert
  - Problem: Truncation (Verkürzung/Abschneiden)
- Prozesse haben ein aktuelles Arbeitsverzeichnis
  - pwd command
- Jeder Nutzer hat ein Heimatverzeichnis
  - cd; echo $HOME – Information über Heimatverzeichnis erlangen
- Dateisystem hat ein Wurzelverzeichnis
  - cd / - Arbeitsverzeichnis wird auf Wurzel gesetzt
- Spezielle Namen identifizieren Nachbarn im Verzeichnisbaum
  - ./ - das aktuelle Verzeichnis
  - ../ - Eine Verzeichnisebene unter (über) der aktuellen Ebene

Namen und Inhalt verbinden

- UNIX trennt Dateinamen und Dateiinhalt
  - Dateiinhalt kann mehrere, verschiedene Namen haben
  - ln-Kommando assoziiert existierenden Inhalt mit neuem Namen
- Dateiinhalt identifiziert durch:
  - (Device, File system on device, i-node)
  - i-node enthält Verweise auf alle Blöcke einer Datei
  - Für jedes Dateisystem wird eine free-node-Liste gepflegt



Zugriff auf Verzeichnisse

- Spezielle Lese-/Schreibefunktionen für Verzeichnisse
  - Eingeführt mit 4.2BSD Unix
- Implementiert in libc (-lc)
  - # include <dirent.h>
  - # include <sys/types.h>
- Zugriffsfunktionen:
  - DIR *opendir(const char *dirname);
  - int closedir(DIR *dirp);
  - int dirfd(DIR *dirp);
  - struct dirent *readdir(DIR *dirp);
  - void rewinddir(DIR *dirp);
  - void seekdir(DIR *dirp, long loc);

```
struct dirent {
        ino_t     d_ino;
                /* file number of entry */
        u_int64_t d_seekoff;
                /* length of this record */
        u_int16_t d_reclen;
                /* length of this record */
        u_int16_t d_namlen;
                 /* length of string in d_name */
        u_int8_t  d_type;
                /* file type, see below */
        char      d_name[MAXPATHLEN];
                /* name must be no longer than this */
   };
```

Beispiel: Verzeichniseintrag suchen

- Suchen im aktuellen Verzeichnis nach Eintrag „name“

```
len = strlen(name);
dirp = opendir(".");
while ((dp = readdir(dirp)) != NULL)
        if (dp->d_namlen == len && !strcmp(dp->d_name, name)) {
                (void) closedir(dirp);
                return FOUND;
        }
(void) closedir(dirp);
return NOT_FOUND;
```

- Syntax:
  - (void) closedir(dirp) zeigt an, dass Rückgabewert ignoriert werden soll