F 4. Übung

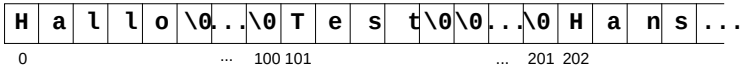
F-1 Überblick

- Aufgabe 2: qsort - Fortsetzung
- Infos zur Aufgabe 4: Verzeichnisse
- Dateisystem: Systemaufrufe

F-2 Aufgabe 2: Sortieren mittels qsort (Fortsetzung)

1 wsort - Datenstrukturen (1. Möglichkeit)

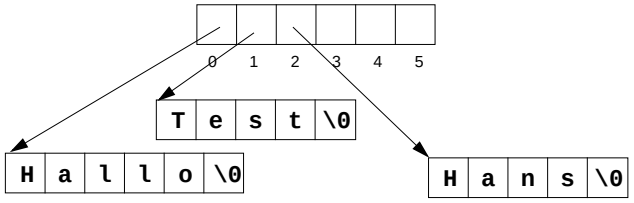
- Array von Zeichenketten



- Vorteile:
  - einfach
- Nachteile:
  - hoher Kopieraufwand
  - Maximale Länge der Worte muss bekannt sein
  - Verschwendung von Speicherplatz

2 wsort - Datenstrukturen (2. Möglichkeit)

- Array von Zeigern auf Zeichenketten



- Vorteile:
  - schnelles Sortieren, da nur Zeiger vertauscht werden müssen
  - Zeichenketten können beliebig lang sein
  - sparsame Speichernutzung

3 Speicherverwaltung

- Berechnung des Array-Speicherbedarfs
  - bei Lösung 1: Anzahl der Wörter * 101 * sizeof(char)
  - bei Lösung 2: Anzahl der Wörter * sizeof(char*)
- realloc:
  - Anzahl der zu lesenden Worte ist unbekannt
  - Array muß vergrößert werden: realloc
  - Bei Vergrößerung sollte man aus Effizienzgründen nicht nur Platz für ein neues Wort (Lösungsvariante 1) bzw. einen neuen Zeiger (Lösungsvariante 2) besorgen, sondern für mehrere.
  - Achtung: realloc kopiert möglicherweise das Array (teuer)
- Speicher sollte wieder freigegeben werden
  - bei Lösung 1: Array freigeben
  - bei Lösung 2: zuerst Wörter freigeben, dann Zeiger-Array freigeben

4 Vergleichsfunktion

- Problem: qsort erwartet folgenden Funktionstyp:

```
int (*compar) (const void *, const void *)
```

- Lösung: "casten"
  - innerhalb der Funktion, z.B. (Feld vom Typ char **):

```
int compare(const void *a, const void *b) {
       return strcmp(*((char **)a), *((char **)b));
}
```

  - beim qsort-Aufruf:

```
int compare(char **a, char **b);
...
qsort(     field, nel, sizeof(char *),
           (int (*)(const void *, const void *))compare);
```


F-3 Aufgabe 4: Verzeichnisse

- opendir(3), readdir(3), closedir(3)
- stat(2), lstat(2)
- readlink(2)
- getpwuid(3), getgrgid(3)

1 opendir / closedir

- Funktions-Prototyp:

```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char *dirname);

int closedir(DIR *dirp);
```

- Argument von opendir
  - **dirname**: Verzeichnisname
- Rückgabewert: Zeiger auf Datenstruktur vom Typ **DIR** oder **NULL**

2 readdir

- Funktions-Prototyp:

```
#include <sys/types.h>
#include <dirent.h>

struct dirent *readdir(DIR *dirp);
```

- Argumente
  - **dirp**: Zeiger auf **DIR**-Datenstruktur
- Rückgabewert: Zeiger auf Datenstruktur vom Typ **struct dirent** oder **NULL** wenn fertig oder Fehler (**errno** vorher auf 0 setzen!)
- Probleme: Der Speicher für **struct dirent** wird von der Bibliothek wieder verwendet!

3 struct dirent

- Definition unter Linux (/usr/include/bits/dirent.h)

```
struct dirent {
    __ino_t d_ino;
    __off_t d_off;
    unsigned short int d_reclen;
    unsigned char d_type;
    char d_name[256];
};
```

- Definition unter Solaris (/usr/include/sys/dirent.h)

```
typedef struct dirent {
    ino_t          d_ino;
    off_t          d_off;
    unsigned short d_reclen;
    char           d_name[1];
} dirent_t;
```


4 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);
```


5 stat / lstat: stat-Struktur

- **dev_t st_dev;** Gerätenummer (des Dateisystems)
- **ino_t st_ino;** Inodenummer
- **mode_t st_mode;** Dateimode, u.a. Zugriffs-Bits (siehe **chmod(1)**)
- **nlink_t 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 (...)
- **unsigned long st_blksize;** Blockgröße des Dateisystems
- **unsigned long st_blocks;** Anzahl der von der Datei belegten Blöcke

6 readlink

- Funktions-Prototyp:

```
#include <unistd.h>

int readlink(const char *path, char *buf, size_t bufsiz);
```

- Argumente
  - **path**: Dateiname
  - **buf**: Puffer für Link-Inhalt
  - **bufsiz**: Größe des Puffers
- Rückgabewert: Anzahl der Bytes oder -1

7 getpwuid

- Funktions-Prototyp:

```
#include <pwd.h>
struct passwd *getpwuid(uid_t uid);
```

- struct passwd:
  - char *pw_name; /* user's login name */
  - uid_t pw_uid; /* user's uid */
  - gid_t pw_gid; /* user's gid */
  - char *pw_gecos; /* typically user's full name */
  - char *pw_dir; /* user's home dir */
  - char *pw_shell; /* user's login shell */

8 getgrgid

- Prototyp:

```
#include <grp.h>
struct group *getgrgid(gid_t gid);
```

- struct group:
  - char *gr_name; /* the name of the group */
  - char *gr_passwd; /* the encrypted group password */
  - gid_t gr_gid; /* the numerical group ID */
  - char **gr_mem; /* vector of pointers to member names */

F-4 Dateisystem Systemcalls

- open(2) / close(2)
- read(2) / write(2)
- lseek(2)
- chmod(2)
- umask(2)
- utime(2)
- truncate(2)

1 open

Hintergrund-Info

- 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)

Hintergrund-Info

1 open - Flags

- **O_EXCL**: zusammen mit **O_CREAT** - nur *neue* Datei anlegen
- **O_TRUNC**: Datei wird beim Öffnen auf 0 Bytes gekürzt
- **O_APPEND**: vor jedem Schreiben wird der Dateizeiger auf das Dateiende gesetzt
- **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_NOCTTY**: beim Öffnen von Terminal-Devices wird das Device nicht zum Kontroll-Terminal des Prozesses

Ü SoS I

Hintergrund-Info

2 close

- Funktions-Prototyp:

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

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

Ü SoS I

Hintergrund-Info

1 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

Ü SoS I

Hintergrund-Info

3 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);
```

Ü SoS I

Hintergrund-Info

4 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

Hintergrund-Info

5 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

6 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);
```


7 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

8 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 */
```


9 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

F-5 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+"
    (**fd** muß entsprechend geöffnet sein!)
- Filedescriptoren in **<unistd.h>**:
  STDIN_FILENO, **STDOUT_FILENO**, **STDERR_FILENO**