**Aufgabe 1: Ankreuzfragen (10 Punkte)**

In dieser Aufgabe sind jeweils $m$ Aussagen angegeben. Davon sind $n$ ($0 \leq n \leq m$) Aussagen richtig. Kreuzen Sie jeweils an, ob die entsprechende Aussage richtig oder falsch ist.
Jede korrekte Antwort gibt einen halben Punkt, jede falsche Antwort einen halben Minuspunkt. Nicht beantwortetete Aussagen gehen neutral in die Bewertung ein. Eine Teilaufgabe wird minimal mit 0 Punkten gewertet, falsche Antworten wirken sich nicht auf andere Teilaufgaben aus.

Wollen Sie eine falsch angekreuzte Antwort korrigieren, streichen Sie bitte das Kreuz mit drei waagrechten Strichen durch (☒).

Lesen Sie die Frage genau, bevor Sie antworten.

a) Bereich: Betriebssystemdefinitionen

| Richtig | Falsch | |
|---|---|---|
| ☐ | ☐ | Ein Betriebssystem ist ein Superprogramm, dass zwischen Nutzer und Hardware vermittelt. |
| ☐ | ☐ | Ein Betriebssystem bietet eine virtuelle Maschine zur einfacheren Verwendung der Hardware. |
| ☐ | ☐ | Ein Betriebssystem verteilt Betriebsmittel an sich bewerbend Nutzer. |
| ☐ | ☐ | Ein Betriebssystem hat keine eindeutige Definition. Es kommt auf den Blickwinkel an. |

b) Bereich: POSIX-Systemaufrufe

| Richtig | Falsch | |
|---|---|---|
| ☐ | ☐ | Das Programm mit dem an `exec()` übergebene Programmpfad wird durch einen neuen Prozess ausgeführt. |
| ☐ | ☐ | Durch Ausführen eines Programms als Administrator (root) gelangt man in den privilegierten Modus des Prozessors. |

c) Bereich: Adressräume

| Richtig | Falsch | |
|---|---|---|
| ☐ | ☐ | Bei Speicher mit wahlfreiem Zugriff hat die Zugriffsreihenfolge Einfluss auf das Programmverhalten |
| ☐ | ☐ | Federgewichtige Prozesse (user threads) erlauben es dem Betriebssystem besonders günstig zwischen diesen zu wechseln. |





d) Bereich: Dateisysteme

| Richtig | Falsch | |
|---|---|---|
| ☐ | ☐ | Verzeichnisse sind spezielle Dateien des Dateisystems, die Namen an Dateiobjekte binden. |
| ☐ | ☐ | In einem UNIX-Dateisystem sind Dateiobjekte stets in einer Baumstruktur angeordnet. |
| ☐ | ☐ | Verzeichnisse definieren den Kontext für die (hierarchische) Namensauflösung. |
| ☐ | ☐ | Ein Dateideskriptor repräsentiert eine prozesslokale Zugriffsbefähigung auf eine Datei. |

e) Bereich: UNIX-Prozesse

| Richtig | Falsch | |
|---|---|---|
| ☐ | ☐ | Ein Prozess kann sich selber vom Zustand “blockiert in “bereit überführen. |
| ☐ | ☐ | Die Instruktionen der Ebene $E_3$ sind immer eine Obermenge der Ebene $E_2$. |
| ☐ | ☐ | Beim Wehchsel eines Prozesses von “laufend” in “bereit” entzieht das Betriebssystem ihm den virtuellen Prozesor. |
| ☐ | ☐ | Ein Prozess wird durch seine Prozess-ID identifiziert. |

f) Scheduling

| Richtig | Falsch | |
|---|---|---|
| ☐ | ☐ | Ein Prozess, der sich in einem kritischen Abschnitt befindet, kann unterbrochen werden. |
| ☐ | ☐ | Präemptives Scheduling und Mehrprogrammbetrieb schließen sich gegenseitig aus. |
| ☐ | ☐ | Leichtgewichtige Prozesse (kernel-threads) können die Multiprozessorfähigkeit des Betriebssystems ausnutzen. |
| ☐ | ☐ | Aktives Warten verhindert Verklemmungen. |

**Aufgabe 2: Programmieraufgabe – Linkcheck (14 Punkte)**

Sie haben ein altes Speichermedium erhalten, dass sie aufräumen sollen. Dieses enthält unzählige Dateien, Verzeichnisse und Links. Ihre Aufgabe ist es ein Program zu schreiben, welches alle verwaisten symbolischen Links im aktuellen Verzeichnis findet.

Das von Ihnen zu entwickelnde Programm `lc` soll das aktuelle Verzeichnis durchsuchen und alle enthaltenen symbolischen Links auf der Standardausgabe ausgeben, deren Ziel nicht existiert.

Sollte ein unerwarteter Fehler auftreten, so soll sich das Programm mit der Funktion `die(char *msg)` mit einer Fehlermeldung beenden.

`handle_link(char *filename)`

`filename` – Name der zu prüfenden Datei

- Prüfen, ob verwaist. Wenn ja, dann Namen ausgeben

`recurse(char *dirname)`

`dirname` – Name des zu durchlaufenden Verzeichnises

- Versteckte Dateien werden ignoriert.
- Alle Dateien außer Verzeichnissen und Links werden ignoriert.
- Es wird nur in Verzeichnisse (nicht Links) hinabgestiegen.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <dirent.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>

#define true 1
#define false 0

// Gegeben:

// Beenden mit Fehlerausgabe
void die(char *msg);

// Zu implementieren

// einen Link prüfen und behandeln
void handle_link(char *filename);

// das aktuelle Verzeichnis durchlaufen
int main(int argc, char *argv[]);
```

---

closedir() closedir()

**NAME**
closedir – close a directory

**SYNOPSIS**
**int closedir(DIR *_dirp_);**

**DESCRIPTION**
The **closedir**() function closes the directory stream associated with *dirp*.

**RETURN VALUE**
The **closedir**() function returns 0 on success. On error, −1 is returned, and *errno* is set appropriately.

---

opendir(3) opendir(3)

**NAME**
opendir, fdopendir – open a directory

**SYNOPSIS**
**DIR *opendir(const char *_name_);**

**DESCRIPTION**
The **opendir**() function opens a directory stream corresponding to the directory *name*, and returns a pointer to the directory stream.

**RETURN VALUE**
The **opendir**() function returns a pointer to the directory stream. On error, NULL is returned, and *errno* is set appropriately.

---

readdir(3) readdir(3)

**NAME**
readdir – read a directory

**SYNOPSIS**
**#include <dirent.h>**

**struct dirent *readdir(DIR *_dirp_);**

**DESCRIPTION**
The **readdir**() function returns a pointer to a *dirent* structure representing the next directory entry in the directory pointed to by *dirp*. It returns NULL on reaching the end of the directory or if an error occurred.

The *dirent* structure is defined as follows:

```
struct dirent {
    ino_t          d_ino;       /* Inode number */
    off_t          d_off;       /* Not an offset; see below */
    unsigned short d_reclen;    /* Length of this record */
    unsigned char  d_type;      /* Type of file */
    char           d_name[256]; /* Null-terminated filename */ };
```

*d_type* This field indicates the file type:

| | |
|---|---|
| **DT_DIR** | This is a directory. |
| **DT_FIFO** | This is a named pipe (FIFO). |
| **DT_LNK** | This is a symbolic link. |
| **DT_REG** | This is a regular file. |
| **DT_SOCK** | This is a UNIX domain socket. |

**RETURN VALUE**
On success, **readdir**() returns a pointer to a *dirent* structure.

If the end of the directory is reached, NULL is returned and *errno* is not changed. If an error occurs, NULL is returned and *errno* is set appropriately.

---

stat(2) stat(2)

**NAME**
stat, fstat – get file status

**SYNOPSIS**
**int stat(const char *_path_, struct stat *_buf_);**
**int lstat(const char *_path_, struct stat *_buf_);**

**DESCRIPTION**
These functions return information about a file.

**stat**() stats the file pointed to by *path* and fills in *buf*.

**lstat**() is identical to **stat**(), except that if *path* is a symbolic link, then the link itself is stat-ed, not the file that it refers to.

All of these system calls return a *stat* structure, which contains the following fields:

```
struct stat {
    dev_t     st_dev;     /* ID of device containing file */
    ino_t     st_ino;     /* inode number */
    mode_t    st_mode;    /* protection */
    nlink_t   st_nlink;   /* number of hard links */
    off_t     st_size;    /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for file system I/O */
};
```

**RETURN VALUE**
On success, zero is returned. On error, −1 is returned, and *errno* is set appropriately.

**ERRORS**

**EACCES**
Search permission is denied for one of the directories in the path prefix of *path*. (See also **path_resolution**(7).)

**...**

**ENAMETOOLONG**
File name too long.

**ENOENT**
A component of the path *path* does not exist, or the path is an empty string.

---

```
void handle_link(char *filename) {
```

**H:**





**M:**

**Aufgabe 3: Textfragen (21 Punkte)**

a) Erklären Sie folgende Begriffe **stichwortartig** und setzen Sie diese in Beziehung zueinander: Prozess, Faden, Prozessor

b) Sie haben zwei Prozesse, die auf einem Rechner mit einer CPU aufeinander warten. Begründen Sie **stichwortartig**, ob aktives oder passives Warten besser geeignet ist.

c) Warum sind Hardlinks auf Verzeichnisse üblicherweise nicht erlaubt? Beschreiben Sie **stichwortartig** das Problem.

d) Warum sind Verzeichnisse ein sinnvolles Konzept in einem Dateisystem? Begründen Sie **stichwortartig**.





e) Klassifizieren Sie die folgenden Betriebsmittel

| | |
|---|---|
| `Speicher` | |
| `Signal` | |
| `Platz in einer Queue` | |
| `Ausgabekanal` | |

f) Ein Prozess führt eine nicht ausführbare Instruktion (z. B. 0-Division) aus. Beschreiben Sie **stichwortartig** die darauf folgenden Schritte. Ist der Ablauf unterdrückbar?