

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 beantwortete 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 Wechhsel eines Prozesses von “laufend” in “bereit” entzieht das Betriebssystem ihm den virtuellen Prozessor.
- ☐
- ☐ 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 Verzeichnisses

- 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(2)
NAME	NAME
closedir – close a directory	stat, fstat – get file status
SYNOPSIS	SYNOPSIS
int closedir(DIR *dirp);	int stat(const char *path, struct stat *buf);
DESCRIPTION	int fstat(const char *path, struct stat *buf);
The <code>closedir()</code> function closes the directory stream associated with <code>dirp</code> .	DESCRIPTION
RETURN VALUE	These functions return information about a file.
The <code>closedir()</code> function returns 0 on success. On error, <code>-1</code> is returned, and <code>errno</code> is set appropriately.	<code>stat()</code> stats the file pointed to by <code>path</code> and fills in <code>buf</code> .
opendir(3)	
NAME	
opendir, fdopendir – open a directory	
SYNOPSIS	
DIR *opendir(const char *name);	
DESCRIPTION	
The <code>opendir()</code> function opens a directory stream corresponding to the directory <code>name</code> , and returns a pointer to the directory stream.	
RETURN VALUE	
The <code>opendir()</code> function returns a pointer to the directory stream. On error, <code>NULL</code> is returned, and <code>errno</code> is set appropriately.	
readdir(3)	
NAME	
readdir – read a directory	
SYNOPSIS	
#include <dirent.h>	
struct dirent *readdir(DIR *dirp);	
DESCRIPTION	
The <code>readdir()</code> function returns a pointer to a <code>dirent</code> structure representing the next directory entry in the directory pointed to by <code>dirp</code> . It returns <code>NULL</code> on reaching the end of the directory or if an error occurred.	
The <code>dirent</code> 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, <code>readdir()</code> returns a pointer to a <code>dirent</code> structure.	
If the end of the directory is reached, <code>NULL</code> is returned and <code>errno</code> is not changed. If an error occurs, <code>NULL</code> is returned and <code>errno</code> is set appropriately.	

```
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

This image shows a full page of primary-ruled paper. It features ten sets of horizontal lines across the page. Each set consists of three lines: a solid top line, a dashed middle line, and a solid bottom line. The lines are evenly spaced and extend from the left margin to the right edge of the page. There is no handwriting or other markings on the paper.

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

e. _____

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?

[illegible]