Leibniz Universität Hannover

Grundlagen der Betriebssysteme - Testat

	erreichbare Punkte	erhaltene Punkte			
Aufgabe 1	4				
Aufgabe 2	11	**1**	**2**	**3**	
Aufgabe 3	5	**1**	**2**		
Aufgabe 4	6	**1**	**2**	**3**	
Aufgabe 5	3				
Aufgabe 6	16	**H**	**M**		
Summe	45				

(Name) (Vorname) (Matrikel-Nr.)

(Studiengang) (Semester)

Durch meine Unterschrift bestätige ich
- den Empfang der vollständigen Klausur (12 Seiten inklusive Deckblatt),
- den Empfang der Manualseiten (chdir, close, closedir, exec, fork, inode, opendir, readdir, readlink, stat, strcmp und strtok),
- die Kenntnisnahme der Hinweise auf Seite 2.

Hannover, 09.01.2026

(Unterschrift)

Hinweise

Bitte lesen Sie die folgenden Informationen aufmerksam und unterschreiben Sie die Erklärung auf der ersten Seite.

- Die Bearbeitungszeit beträgt 45 Minuten.
- Es sind **keine** eigenen Hilfsmittel zugelassen.
- Die Lösung einer Aufgabe soll auf das Aufgabenblatt in den dafür vorgesehenen Raum geschrieben werden. Beachten Sie bitte, dass der freigelassene Platz großzügig bemessen ist und nicht unbedingt der erwarteten Antwortlänge entspricht.
- Die Lösungen müssen dokumentenecht in blau oder schwarz geschrieben werden. Als falsch Erkanntes muss deutlich durchgestrichen werden. Tintenkiller und andere Korrekturstifte dürfen nicht verwendet werden. Keinen Bleistift verwenden!
- Fragen zu den Prüfungsaufgaben können grundsätzlich **nicht** beantwortet werden.
- Tragen Sie Ihren Namen und Vornamen, Ihre Matrikelnummer, Studiengang und Fachsemesterzahl auf dem Deckblatt der Klausur ein.
- Tragen Sie auf jedem Blatt Ihren Namen ein.
- Die Seiten 9, 10, 11 und 12 (Aufgabenstellungen der Programmieraufgaben und Manpages) dürfen herausgetrennt werden. Bitte schreiben Sie keine Lösungen auf herausgetrennte Seiten. Alle anderen Seiten dürfen **nicht** herausgetrennt werden.
- Bitte legen Sie Ihren gültigen Studierenden- und Lichtbildausweis zur Kontrolle bereit.
- Sie dürfen den Raum nicht verlassen, bevor Ihre Personalien überprüft wurden und Sie die Klausurunterlagen der Aufsicht zurückgegeben haben.
- Um Unruhe und die Störung Ihrer Mitstudierenden zu vermeiden, ist die vorzeitige Abgabe der Klausur ausgeschlossen. Bleiben Sie an Ihrem Platz sitzen, bis am Ende alle Klausurunterlagen eingesammelt sind und die Aufsicht das Zeichen zum Gehen gibt.

Aufgabe 1: (4 Punkte)

1.1: Bitte beantworten Sie, wie häufig Sie die Gruppenübungen wahrgenommen haben. Begründen Sie dies in mindestens einem Satz. Jede nicht leere Antwort ist richtig.

2 Punkte-

- ○ 9
- ○ 8
- ○ 7
- ○ 6
- ○ 5
- ○ 4
- ○ 3
- ○ 2
- ○ 1
- ○ 0

1.2: Bitte beantworten Sie, wie viele Hausaufgaben Sie bearbeitet haben. Begründen Sie dies in mindestens einem Satz. Jede nicht leere Antwort ist richtig.

2 Punkte-

- ○ 9
- ○ 8
- ○ 7
- ○ 6
- ○ 5
- ○ 4
- ○ 3
- ○ 2
- ○ 1
- ○ 0

Aufgabe 2: Unterbrechungen und Ausnahmen (11 Punkte)

2.1: Was bedeutet Teilinterpretation im Kontext eines Betriebssystems? Unter welchen Umständen wird das Betriebssystem dazu aktiv?

3 Punkte-

2.2: Vervollständigen Sie die gegebene Tabelle.

6 Punkte-

Ausnahme/ Unterbrechung	synchron/ asynchron	explizit/ implizit	behandelt von Prozess (E_3)/ Betriebssystem (E_2)
Timer-Interrupt			
Systemaufruf			
Signal SIGSEGV			
Signal SIGINT			

2.3: Welche Art von Problemen kann auftreten, wenn ein Signalbehandler auf Zustand der Anwendung zugreift? Warum ist das Ausführen von Systemaufrufen aus dem Kontext eines Signalbehandlers davon nicht betroffen?

2 Punkte-

Aufgabe 3: Scheduling (5 Punkte)

3.1: Gegeben sei ein präemptives Betriebssystem, das auf einem System mit einer CPU läuft. Wenn ein Prozess auf diesem Betriebssystem einen kritischen Abschnitt betritt, wird das Scheduling ausgesetzt. Dem Prozess im kritischen Abschnitt wird die CPU also exklusiv zugewiesen und kein anderer Prozess kann sie bekommen.
Hat das Aussetzen des Schedulings einen Einfluss auf das Auftreten von Verklemmungen (Deadlocks)? Begründen und erklären sie.

3 Punkte-

3.2: Wie kann das im vorigen Aufgabenteil beschriebene Schedulingverhalten böswillig ausgenutzt werden? Benennen Sie das Problem und einen Weg es auszunutzen.

2 Punkte-

Aufgabe 4: Konzepte (6 Punkte)

4.1: Welches Konzept muss ein Dateisystem implementieren, damit zwei unterschiedliche Dateien unter dem gleichen Namen zeitgleich existieren können?

2 Punkte-

4.2: Betriebssystemarchitekturen werden konzeptionell in die Kategorien Monolith, Mikrokern, Bibliothek geteilt. Was ist die unterscheidende Eigenschaft? Beschreiben Sie die Umsetzung der Eigenschaft kurz für eine der genannten Kategorien.

2 Punkte-

4.3: Welche fundamentalen Konzepte implementiert ein Betriebssystem? Nennen Sie zwei!

2 Punkte-

Aufgabe 5: Parallelisierung (3 Punkte)

3 Punkte-

Es soll ein Programm geschrieben werden, das ein mathematisches Problem löst. Dabei gelten die folgenden Rahmenbedingungen:

- Das Problem besteht aus vielen parallelisierbaren Teilproblemen.
- Die Teilprobleme sind unabhängig voneinander, es ist aber der Zugriff auf geteilte Daten (im RAM) notwendig.
- Das Programm wird auf einem Mehrprozessorsystem ausgeführt.

Das Programm kann das Problem entweder sequeziell, oder parallel mit Hilfe von Prozessen, Kernel-Level oder User-Level Threads lösen. Welche Variante wählen Sie? Begründen Sie Ihre Entscheidung.

Aufgabe 6: Programmieraufgabe – `irca` (16 Punkte)

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

#define true 1
#define false 0

// Gegeben:

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

// Zu implementieren:

// Eine Datei prüfen
int check_file(char* name);

// Die Liste der Dateien eines Verzeichnisses durchlaufen
int main(int argc, char *argv[]);
```

```
int check_file(char *name) {
```

H:

```
int main(int argc, char *argv[]) {
```


M:

Schreiben Sie ein Programm `irca` (**i**node **r**eferece **c**ounting **a**gent), welches in einem gegebenen Verzeichnis für die Einträge deren Anzahl der Hardlinks und Namen auflistet.

Implementieren Sie die Funktionalität in den Funktionen `check_file()` und `main()`:

Funktion `check_file(char* name)`

- Auslesen der Dateiattribute von `name`
- Rückgabe: Anzahl der Hard-Links
- Im Fehlerfall: Rückgabe `-1`, `errno` enthält den Fehlercode

Funktion `main(int argc, char* argv[])`

- Iterieren über alle Einträge des gegebenen Verzeichnisses
- Mittels `check_file(char* name)` für jeden Eintrag die Hard-Link-Zahl ermitteln.
- Ausgaben:
  - bei Erfolg: „`<Anzahl der Hardlinks in Dezimal> <Dateiname>`“
  - bei Fehler: „`<Dateiname>: <Fehlerbeschreibung>`“

Fehlerbehandlung
Sollte in der `main`-Funktion ein unerwarteter Fehler auftreten, so soll das Programm mitels der Funktion `die(char *msg)` eine Fehlermeldung ausgeben und sich beenden.

Zusicherungen, Annahmen, Bedingungen

- Das Programm wird immer mit genau einem Verzeichnisparameter ausgeführt.
- Ausgaben sind nur in der `main`-Funktion erlaubt.
- Fehlerausgaben müssen auf den Standard-Fehlerausgabekanal geschrieben werden.

Beispielfunktionsweise:

```
fiedler@srademo:$ ls -l /home
drwxr-xr-x  55 root       root        0 Jan  6 14:34 .
drwxr-xr-x  20 root       root     4,0K Jan  2 07:06 ..
drwxr-x---+  3 brehm      srastaff 2,8K Aug 22  2023 brehm
drwxr-x---+  8 fiedler    srastaff 2,5K Jan  6 14:32 fiedler
drwxr-x---+  4 hollmann   srastaff  238 Dez 11 17:23 hollmann
drwxr-x---+ 12 lohmann    srastaff  414 Mär 20  2024 lohmann
drwxr-xr-x   4 maasjost   srastaff 7,3K Nov  6 14:13 maasjost
drwxr-x---+  6 ostapyshyn srastaff  238 Dez 16 15:50 ostapyshyn

fiedler@srademo:~$ ./irca /home
55 .
20 ..
 6 ostapyshyn
 4 hollmann
12 lohmann
 3 brehm
 4 maasjost
 8 fiedler
```

Diese Seite darf herausgetrennt werden.
Diese Seite darf herausgetrennt werden.

Diese Seite darf herausgetrennt werden.

NAME

chdir, fchdir – change working directory

SYNOPSIS

#include <unistd.h>

int chdir(const char ****path*);**
int fchdir(int *fd***);**

DESCRIPTION

chdir() changes the current working directory of the calling process to the directory specified in *path*.
fchdir() is identical to **chdir**(); the only difference is that the directory is given as an open file descriptor.

RETURN VALUE

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

NAME

close – close a file descriptor

SYNOPSIS

#include <unistd.h>

int close(int *fd***);**

DESCRIPTION

close() closes a file descriptor, so that it no longer refers to any file and may be reused.

RETURN VALUE

close() returns zero on success. On error, −1 is returned, and *errno* is set appropriately.

NAME

closedir – close a directory

SYNOPSIS

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

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.

NAME

execl, execlp, execv, execvp – execute a file

SYNOPSIS

#include <unistd.h>

int execl(const char ****pathname*, const char ****arg***, ... , NULL */);**
int execlp(const char ****file*, const char ****arg***, ... , NULL);**
int execv(const char ****pathname*, char *const** *argv*[]**);**
int execvp(const char ****file*, char *const** *argv*[]**);**

DESCRIPTION

The **exec**() family of functions replaces the current process image with a new process image.

The initial argument for these functions is the name of a file that is to be executed.

The functions can be grouped based on the letters following the "exec" prefix.

l - execl(), execlp()
The *const char *arg* and subsequent ellipses can be thought of as *arg0*, *arg1*, ..., *argn*. The list of arguments *must* be terminated by a null pointer.

By contrast with the 'l' functions, the 'v' functions (below) specify the command-line arguments of the executed program as a vector.

v - execv(), execvp()
The *char *const argv[]* argument is an array of pointers to null-terminated strings that represent the argument list available to the new program. The first argument, by convention, should point to the filename associated with the file being executed. The array of pointers *must* be terminated by a null pointer.

p - execlp(), execvp()
These functions duplicate the actions of the shell in searching for an executable file if the specified filename does not contain a slash (/) character.

RETURN VALUE

The **exec**() functions return only if an error has occurred. The return value is −1, and *errno* is set to indicate the error.

NAME

fork – create a child process

SYNOPSIS

#include <sys/types.h>
#include <unistd.h>

pid_t fork(void);

DESCRIPTION

fork() creates a new process by duplicating the calling process. The new process is referred to as the *child* process. The calling process is referred to as the *parent* process.

The child process is an exact duplicate of the parent process except for the following points:

* The child has its own unique process ID.
* The child's parent process ID is the same as the parent's process ID.

RETURN VALUE

On success, the PID of the child process is returned in the parent, and 0 is returned in the child. On failure, −1 is returned in the parent, no child process is created, and *errno* is set appropriately.

Diese Seite darf herausgetrennt werden.

Diese Seite darf herausgetrennt werden.

NAME

inode – file inode information

DESCRIPTION

Each file has an inode containing metadata about the file. An application can retrieve this metadata using **stat**(2) , which returns a *stat* structure.

User ID: *stat.st_uid*
This field records the user ID of the owner of the file. For newly created files, the file user ID is the effective user ID of the creating process.

Group ID: *stat.st_gid*
The inode records the ID of the group owner of the file. For newly created files, the file group ID the group ID of the creating process.

Last access timestamp (atime): *stat.st_atime*
This is the file's last access timestamp. It is changed by file accesses, for example, by **execve**(2), **mknod**(2), **pipe**(2), **utime**(2), and **read**(2) (of more than zero bytes).

Last modification timestamp (mtime): *stat.st_mtime*
This is the file's last modification timestamp. It is changed by file modifications, for example, by **mknod**(2), **truncate**(2), **utime**(2), and **write**(2) (of more than zero bytes). Moreover, the mtime timestamp of a directory is changed by the creation or deletion of files in that directory.

The file type and mode
The *stat.st_mode* field contains the file type and mode.

POSIX refers to the *stat.st_mode* bits corresponding to the mask **S_IFMT** as the *file type*, the 12 bits corresponding to the mask 07777 as the *file mode bits* and the least significant 9 bits (0777) as the *file permission bits*.

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;      /* i-node number */
  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.

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.

NAME

readlink – read value of a symbolic link

SYNOPSIS

#include <unistd.h>

ssize_t readlink(const char * *pathname***, char *** *buf* **, size_t** *bufsiz***);**

DESCRIPTION

readlink() places the contents of the symbolic link *pathname* in the buffer *buf* , which has size *bufsiz*. **readlink**() does not append a null byte to *buf* . It will (silently) truncate the contents (to a length of *bufsiz* characters), in case the buffer is too small to hold all of the contents.

RETURN VALUE

On success, these calls return the number of bytes placed in *buf* . On error, −1 is returned and *errno* is set.

Diese Seite darf herausgetrennt werden.

Diese Seite darf herausgetrennt werden.

NAME

stat, lstat – get file status

SYNOPSIS

#include <sys/types.h> #include <sys/stat.h> #include <unistd.h>

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;      /* File type and mode */
    nlink_t   st_nlink;     /* Number of hard links */
    uid_t     st_uid;       /* User ID of owner */
    gid_t     st_gid;       /* Group ID of owner */
    dev_t     st_rdev;      /* Device ID (if special file) */
    off_t     st_size;      /* Total size, in bytes */
    blksize_t st_blksize;   /* Block size for filesystem I/O */
    blkcnt_t  st_blocks;    /* Number of 512B blocks allocated */

    struct timespec st_atim;  /* Time of last access */
    struct timespec st_mtim;  /* Time of last modification */
    struct timespec st_ctim;  /* Time of last status change */ };
```

For further information on the above fields, see **inode**(7).

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 is a dangling symblic link, or the path is an empty string.

NAME

strcmp, strncmp – compare two strings

SYNOPSIS

#include <string.h>

int strcmp(char ****s1*, char ****s2***);**
int strncmp(char *s1***[.***n***], char** *s2***[.***n***], size_t** *n***);**

DESCRIPTION

The **strcmp**() function compares the two strings *s1* and *s2*.

strcmp() returns an integer indicating the result of the comparison, as follows:

- 0, if the *s1* and *s2* are equal;
- a negative value if *s1* is less than *s2*;
- a positive value if *s1* is greater than *s2*.

The **strncmp**() function is similar, except it compares only the first (at most) *n* bytes of *s1* and *s2*.

RETURN VALUE

The **strcmp**() and **strncmp**() functions return an integer less than, equal to, or greater than zero if *s1* (or the first *n* bytes thereof) is found, respectively, to be less than, to match, or be greater than *s2*.

NAME

strtok – extract tokens from strings

SYNOPSIS

#include <string.h>

char *strtok(char ****str*, const char ****delim***);**

DESCRIPTION

The **strtok**() function breaks a string into a sequence of zero or more nonempty tokens. On the first call to **strtok**(), the string to be parsed should be specified in *str*. In each subsequent call that should parse the same string, *str* must be NULL.

The *delim* argument specifies a set of bytes that delimit the tokens in the parsed string. The caller may specify different strings in *delim* in successive calls that parse the same string.

RETURN VALUE

strtok() returns a pointer to the next token, or NULL if there are no more tokens.

Diese Seite darf herausgetrennt werden.