Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

Prof. Dr. Frank Bellosa
Lukas Werling, M.Sc.
Dr. Marc Rittinghaus

Musterlösung Hauptklausur

Programmieren

02.03.2020

Alle Punkteangaben ohne Gewähr!

- Bitte tragen Sie zuerst auf dem Deckblatt Ihren Namen, Ihren Vornamen und Ihre Matrikelnummer ein. Tragen Sie dann auf den anderen Blättern (auch auf Konzeptblättern) Ihre Matrikelnummer ein.

  Please fill in your last name, your first name, and your matriculation number on this page and fill in your matriculation number on all other (including draft) pages.

- Die Prüfung besteht aus 15 Blättern: 1 Deckblatt, 14 Aufgabenblättern mit insgesamt 3 Aufgaben und 0 Blättern Man-Pages.

  The examination consists of 15 pages: 1 cover sheet, 14 sheets containing 3 assignments, and 0 sheets for man pages.

- Es sind keinerlei Hilfsmittel erlaubt!

  No additional material is allowed.

- Die Prüfung ist nicht bestanden, wenn Sie aktiv oder passiv betrügen.

  You fail the examination if you try to cheat actively or passively.

- Sie können auch die Rückseite der Aufgabenblätter für Ihre Antworten verwenden. Wenn Sie zusätzliches Konzeptpapier benötigen, verständigen Sie bitte die Klausuraufsicht.

  You can use the back side of the task sheets for your answers. If you need additional draft paper, please notify one of the supervisors.

- Bitte machen Sie eindeutig klar, was Ihre endgültige Lösung zu den jeweiligen Teilaufgaben ist. Teilaufgaben mit widersprüchlichen Lösungen werden mit 0 Punkten bewertet.

  Make sure to clearly mark your final solution to each question. Questions with multiple, contradicting answers are void (0 points).

- Programmieraufgaben sind gemäß der Vorlesung in C zu lösen.

  Programming assignments have to be solved in C.

Die folgende Tabelle wird von uns ausgefüllt!
The following table is completed by us!

Aufgabe	1	2	3	Total
Max. Punkte	15	15	15	45
Erreichte Punkte				
Note				

Aufgabe 1: C Grundlagen

Assignment 1: C Basics

a) Was gibt der unten stehende Code bei der Ausführung der Funktion `print_test()` aus? **1 pt**

What does the code below print when running the function `print_test()`?

```
struct test {
    int a, b, c;
};

void print_test(void) {
    struct test t = {1};
    printf("%d/%d/%d", t.a, t.b, t.c);
}
```

Solution:

1/0/0

b) Geben Sie für alle Felder des unten stehenden `struct mystruct` jeweils die Größe des Feldes und die Größe des Paddings *nach* dem Feld in Byte an. Schreiben Sie "0", falls kein Padding eingefügt wird. Gehen Sie von einem 64-Bit-System aus. **3 pt**

For each field of the `struct mystruct` below, give the field's size and the size of the padding after *the field in Bytes. Write "0" if the compiler does not insert any padding. Assume a 64-bit system.*

Code	Field size [Byte]	Padding size [Byte]
`struct mystruct {`	—	—
`  char a;`	1	3
`  uint32_t b;`	4	0
`  int16_t c;`	2	6
`  int64_t d;`	8	0
`};`	—	—

(1.0 P) for all field sizes, **(0.5 P)** per padding size

c) Definieren Sie ein C-Makro `ARRAY_SIZE`, das die Anzahl der Elemente eines statisch allozierten Arrays berechnet. **1 pt**

Define a C macro `ARRAY_SIZE` that prints the number of elements in a statically allocated array.

Examples:

```
int array[13];
char str[100];
assert(ARRAY_SIZE(array) == 13);
assert(ARRAY_SIZE(str) == 100);
assert(~(~ARRAY_SIZE(array)) == 13);
```

Solution:

```
#define ARRAY_SIZE(a) (sizeof(a) / sizeof((a)[0]))
```

macro syntax (including parentheses) **(0.5 P)**, *sizeof expression* **(0.5 P)**

Begründen Sie, warum das Makro `ARRAY_SIZE` nicht in der unten stehenden Funktion `sum()` eingesetzt werden kann. Welchen Wert gibt `ARRAY_SIZE` in der Funktion `sum()` auf einem 64-Bit-System tatsächlich zurück? Wie kann das Problem behoben werden? **1.5 pt**

Give a reason why the macro `ARRAY_SIZE` does not work in the function `sum()` given below. Which value does `ARRAY_SIZE` actually return in the function `sum()` on a 64-bit system? How can this issue be solved?

```
int32_t sum(int32_t *array) {
    int32_t result = 0;
    for (size_t i = 0; i < ARRAY_SIZE(array); i++)
        result += array[i];
    return result;
}

int main() {
    int32_t array[15];
    /* ... */
    int s = sum(array);
    /* ... */
}
```

Solution:

The macro uses `sizeof` to retrieve the array size from the type information. The type of `array` in `sum()` is `int32_t *` and does not depend on the size of the array. **(0.5 P)**

`sizeof(array)` = `sizeof(int32_t *)` = 8 and
`sizeof(array[0])` = `sizeof(int32_t)` = 4 and thus
`ARRAY_SIZE(array)` = 2. **(0.5 P)**

We can fix `sum()` by adding a second parameter such as `size_t len` that indicates the length of the array. **(0.5 P)**

d) Welches Problem tritt bei der Verwendung der Funktion `init_config()` auf? **1 pt**

Which problem occurs when using the function `init_config()`?

```
struct config {
    int verbose;
    int jobs;
    int dry_run;
};

struct config *init_config(int n) {
    struct config c = {
        .verbose = 0,
        .jobs = n,
        .dry_run = 0
    };
    return &c;
}
```

Solution:

`c` is allocated on the stack **(0.5 P)**, *so pointers to it are only valid during execution of `init_config()`. Consequently, the pointer `&c` that the function returns is immediately invalid (use-after-free)* **(0.5 P)**.

Schreiben Sie eine alternative Funktion `init_config2()`, die die gleiche Funktionalität wie `init_config()` bietet, aber das oben genannte Problem nicht hat. **1.5 pt**

Write an alternative function `init_config2()` that offers the same functionality as `init_config()` but does not have the problem above.

Solution:

```
struct config *init_config2(int n) {
    struct config *c = malloc(sizeof(struct config));

    c->verbose = 0;
    c->jobs = n;
    c->dry_run = 0;

    return c;
}
```

correct malloc **(0.5 P)**, *correct initialization* **(0.5 P)**, *return* **(0.5 P)**

no malloc **(0 P)**

Note: *C99 compound literal syntax requires specifying the type:*

```
*c = (struct config) { .jobs = n };
```

e) Die `memset()`-Funktion der C-Standardbibliothek überschreibt einen Speicherbereich mit einem beliebigen Byte-Wert. In dieser Aufgabe sollen Sie eine Funktion `pattern_memset()` implementieren, die einen Speicherbereich mit einem 8 Bytes langem Muster wie im Beispiel unten überschreiben soll.

The `memset()` function in the C standard library overwrites a memory area with an arbitrary byte value. In this assignment, you will write a function `pattern_memset()` that overwrites a memory area with an 8 bytes long pattern as in the example below.

```
uint64_t pat = 0x123456789ABCDEF0ull;
char buf[14];
pattern_memset(buf, &pat, sizeof(buf));
```

In-memory view of `pat` (little endian):

Byte	0	1	2	3	4	5	6	7
pat	F0	DE	BC	9A	78	56	34	12

Result `buf`:

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13
buf	F0	DE	BC	9A	78	56	34	12	F0	DE	BC	9A	78	56

Implementieren Sie die Funktion `rotr()`, die eine gegebene 64-Bit-Ganzzahl um ein Byte (= 8 Bits) nach rechts rotiert. **1 pt**

Implement the function `rotr()` that rotates a 64-bit integer by one byte (= 8 bits) to the right.

```
uint64_t pat = 0x123456789ABCDEF0ull;
assert(rotr(pat) == 0xF0123456789ABCDEull);
```

In-memory view (little endian):

Byte	0	1	2	3	4	5	6	7
`pat`	F0	DE	BC	9A	78	56	34	12
`rotr(pat)`	DE	BC	9A	78	56	34	12	F0

Solution:

```
uint64_t rotr(uint64_t v) {
    return (v >> 8) | (v << (64 - 8));
}
```

Implementieren Sie die Funktion `pattern_memset()`, die das Muster in `*pat` Byte für Byte in den Puffer `s` der Länge `n` schreibt. Nach dem Aufruf soll `*pat` passend rotiert sein, sodass ein weiterer Aufruf das Muster fortsetzt. **2 pt**

Implement the function `pattern_memset()` that writes the pattern in `*pat` byte by byte to the buffer `s` of length `n`. After the call, `*pat` shall be rotated appropriately so that a subsequent call continues the pattern.

```
uint64_t pat = 0x123456789ABCDEF0ull, pat1 = pat;
char buf[5];
pattern_memset(buf, &pat, 2);
uint64_t pat2 = pat;
assert(pat2 == 0xDEF0123456789ABCull);
pattern_memset(buf + 2, &pat, 3);
uint64_t pat3 = pat;
assert(pat3 == 0x789ABCDEF0123456ull);
```

In-memory view (little endian):

Byte	0	1	2	3	4	5	6	7
`pat1`	F0	DE	BC	9A	78	56	34	12
`pat2`	BC	9A	78	56	34	12	F0	DE
`pat3`	56	34	12	F0	DE	BC	9A	78
`buf`	F0	DE	BC	9A	78			

Solution:

```
void pattern_memset(void *s, uint64_t *pat, size_t n) {
    char *b = s;

    for (size_t i = 0; i < n; i++) {
        b[i] = (char)*pat;
        *pat = rotr(*pat);
    }
}
```

casting of buffer **(0.5 P)**, *loop* **(0.5 P)**, *1-byte writing of pattern* **(0.5 P)**, *rotating pattern* **(0.5 P)**

Implementieren Sie die Funktion `pattern_memset_64()`, die das Muster in `*pat` in den Puffer `s` der Länge `n` schreibt und dabei die Zahl der Speicherzugriffe wie folgt reduziert: **3 pt**

- Wenn die Adresse von `s` an 8-Byte ausgerichtet ist (`(addr & 0x7) == 0`), soll `s` mit so vielen 8-Byte-Zugriffen wie möglich befüllt werden.
- Verwenden Sie `pattern_memset()` um restliche Bytes zu schreiben oder wenn die Adresse nicht ausgerichtet ist.
- Nach dem Aufruf soll `*pat` passend rotiert sein, sodass ein weiterer Aufruf das Muster fortsetzt.

Implement the function `pattern_memset_64()` that writes the pattern in `*pat` to the buffer `s` of length `n`. The function shall reduce the number of memory accesses as follows:

- *If the address of `s` is 8-byte aligned (`(addr & 0x7) == 0`), `s` has to be filled with as much 8-byte writes as possible.*
- *Use `pattern_memset()` to write remaining bytes or if the address is not aligned.*
- *After the call, `*pat` shall be rotated appropriately so that a subsequent call continues the pattern.*

Example 1: `s = (void *)0x1008`, `n = 10`

Address	0x1008								0x1010	
s	F0	DE	BC	9A	78	56	34	12	F0	DE
Access size	8								1	1

Example 2: `s = (void *)0x1006`, `n = 10`

Address	0x1006		0x1008							
s	F0	DE	BC	9A	78	56	34	12	F0	DE
Access size	1	1	1	1	1	1	1	1	1	1

Solution:

```
void pattern_memset_64(void *s, uint64_t *pat, size_t n) {
    uint64_t *b = s;

    if (((intptr_t)s & 7) == 0) {
        size_t i = 0;
        for (; i + 8 <= n; i += 8, ++b) {
            *b = *pat;
        }
        n -= i;
    }

    pattern_memset(b, pat, n);
}
```

casting of buffer **(0.5 P)**, *alignment check* **(0.5 P)**, *loop* **(0.5 P)**, *8-byte writing of pattern* **(0.5 P)**, *correctly calling* `pattern_memset()` **(1 P)**

Total: 15.0pt

Aufgabe 2: Log-Rotation

Assignment 2: Log Rotation

Sie sollen Funktionen schreiben, die eine Log-Datei in Textform verwalten. Damit das Log hierbei nicht beliebig viel Speicherplatz verbraucht, soll Ihr Code eine Rotation des Logs implementieren: Sobald die Log-Datei eine bestimmte Größe erreicht hat, soll sie an einen anderen Ort verschoben werden, wobei gegebenenfalls eine andere dort liegende alte Log-Datei überschrieben wird. Weitere Log-Einträge sollen in Folge in eine neue Datei am ursprünglichen Pfad geschrieben werden.

- Binden Sie die in den Teilaufgaben notwendigen C-Header in dem gekennzeichneten Bereich ein.
- Sie müssen in dieser Aufgabe keine Fehlerbehandlung implementieren.
- Geben Sie jegliche in ihrem Code angeforderten Resourcen wieder frei. Lediglich der aktuelle Dateideskriptor `log_fd` darf bei Programmende noch geöffnet sein.

You have to write functions to manage a log file in text form. To prevent the log from using an arbitrary amount of storage space, your code shall implement a rotation of the log: As soon as the log file reaches a certain size, the file shall be moved to another place, potentially overwriting another old log file in that place. New log entries then shall be written into a new file at the original path.

- *Include all required C headers in the marked area.*
- *You do not have to implement error handling.*
- *Free all resources allocated in your code. Only the current file descriptor `log_fd` is allowed to remain open at the end of the program.*

```c
/* include statements for the required C headers */
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <pthread.h>
#include <stdio.h>

#define PATH "my_app.log"           /* log file path */
#define ROTATED_PATH "my_app.log.2" /* path to which the log file is moved */
#define MAX_SIZE 1000000            /* maximum log size in bytes */

int log_fd;         /* file descriptor of the log file */
uint64_t log_size;  /* current size of the log file in bytes */

/* mutex for global variables, initialized and ready to be used */
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

a) Die Variable `log_size` hat den Typ `uint64_t`. Welchen Header müssen Sie für diesen Typ inkludieren? **0.5 pt**

The variable `log_size` is of the type `uint64_t`. Which header do you need to include for this type?

Solution:

`stdint.h` (alternative: `inttypes.h`)

b) Vervollständigen Sie die Funktion `log_init()`, die zu Beginn des Programmes aufgerufen wird und die Log-Datei öffnet, ihre aktuelle Größe bestimmt und den Dateideskriptor sowie die Größe in den globalen Variablen speichert. **6.5 pt**

- Falls die Datei noch nicht existiert, soll sie erstellt werden.
- Auf die Log-Datei sollen der Eigentümer lesend und schreibend und alle anderen Benutzer nur lesend zugreifen können.
- Folgende `write()`-Aufrufe sollen die Daten ans Ende der Datei anhängen.

Complete the function `log_init()` which is called at the start of the program and which opens the log file, determines its current size, and stores the file descriptor as well as the size in the global variables.

- *If the file does not exist yet, it shall be created.*
- *The owner shall access the file for reading and writing, whereas other users shall only be able to read the file.*
- *Following `write()` calls shall append the data to the end of the file.*

Solution:

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

void log_init(void) {
    struct stat info;

    log_fd = open(PATH, O_WRONLY | O_APPEND | O_CREAT,
        S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH);

    fstat(log_fd, &info);
    log_size = info.st_size;
}
```

Adding the necessary includes **(2.0 P)**, *`open()` on the correct path* **(1.0 P)** *with the correct flags* **(1.0 P)** *and the correct mode* **(1.0 P)**, *`fstat()` or `stat()` with a correct pointer to a stat buffer* **(1.0 P)**, *storing `st_size` in `log_size`* **(0.5 P)**.

c) Vervollständigen Sie die Funktion `log_write()`, die den übergebenen Text in das Log schreibt und durch den Aufruf von `rotate_log()` eine Rotation des Logs auslöst, wenn durch den Schreibvorgang die durch `MAX_SIZE` festgelegten Größe erreicht oder überschritten wurde. **5.5 pt**

- Die Funktion soll von mehreren Threads parallel aufgerufen werden können. Verwenden Sie den globalen Mutex `mutex`, um Operationen wo nötig zu synchronisieren.
- Sie können davon ausgehen, dass bei einem Schreibvorgang die dem Systemaufruf übergebenen Daten immer vollständig geschrieben werden.

Complete the function `log_write()` *which writes the passed string into the log and which triggers a rotation of the log by calling* `rotate_log()` *if the size specified by* `MAX_SIZE` *was reached or exceeded by the write operation.*

- *It shall be possible to call the function from multiple threads in parallel. Use the global mutex* `mutex` *to synchronize operations where needed.*
- *You can assume that during a write operation the data passed to the system call is always written completely.*

Solution:

```
#include <string.h>
#include <pthread.h>

void log_write(const char *text) {
    pthread_mutex_lock(&mutex);

    write(log_fd, text, strlen(text));

    log_size += strlen(text);
    if (log_size >= MAX_SIZE) {
        rotate_log();
    }

    pthread_mutex_unlock(&mutex);
}
```

Adding the necessary includes **(1.0 P)**, *writing the text into the log* **(1.0 P)**, *incrementing the log size* **(0.5 P)**, *using* `strlen()` *correctly* **(0.5 P)**, *rotating the log* **(0.5 P)** *if the maximum size was exceeded* **(0.5 P)**, *locking and unlocking the mutex at correct locations* **(1.5 P)**.

d) Vervollständigen Sie die Funktion `rotate_log()`, die die Inhalte des aktuellen Logs an den Pfad `ROTATED_LOG` verschiebt und unter Nutzung der Funktion `log_init()` eine neue Log-Datei am ursprünglichen Pfad anlegt. **2.5 pt**

Complete the function `rotate_log()` *which moves the contents of the current log to the path* `ROTATED_LOG` *and which creates a new log file at the original path by using the function* `log_init()`.

Solution:

```
#include <stdio.h>

void rotate_log(void) {
    close(log_fd);
    rename(PATH, ROTATED_PATH);
    log_init();
}
```

Adding the necessary includes **(0.5 P)**, *closing the file descriptor* **(0.5 P)**, *renaming the file to the new location* **(1.0 P)**, *creating a new log file* **(0.5 P)**. **Total: 15.0pt**

Aufgabe 3: Dateisystemimplementierung

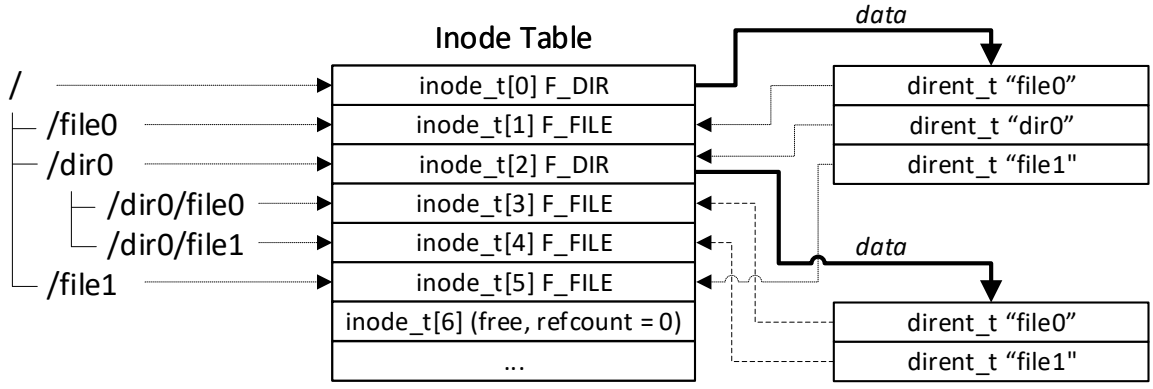
Assignment 3: File System Implementation

Im Folgenden sollen verschiedene Funktionen eines einfachen Dateisystems implementiert werden. Dateien und Ordner werden als Inodes (`inode_t`) repräsentiert, die in einer statischen Inode-Tabelle gespeichert werden. Ordner unterscheiden sich von regulären Dateien, indem der entsprechender Typ (`F_DIR`) im Inode hinterlegt ist und die referenzierten Daten aus einer Liste von Verzeichniseinträgen (`dirent_t`) bestehen.

- Sofern nicht anders genannt, gehen Sie davon aus, dass alle Parameter valide Werte enthalten und das Dateisystem nicht beschädigt ist.

In the following you have to implement various functions of a simple file system. Files and directories are represented by inodes (`inode_t`), which are stored in a static inode table. Directories differ from regular files in that the corresponding type (`F_DIR`) is assigned to the inode and the referenced data is a list of directory entries (`dirent_t`).

- *Unless otherwise specified, assume that all parameters contain valid values and that the file system is not damaged.*




```c
typedef enum {
    F_FILE = 0,         /* the inode is a regular file */
    F_DIR  = 1          /* the inode is a directory */
} file_type;

typedef struct {
    uint32_t refcount;  /* number of references to inode, 0 = free */
    off_t length;       /* length of the file in bytes */

    file_type type;     /* type of file */

    /* ... */
} inode_t;

#define ROOT_DIR 0      /* inode number of root directory */
#define MAX_INODES 64
typedef struct {
    inode_t inodes[MAX_INODES];  /* static inode table */
} filesystem;
```

```
#define MAX_NAME 15     /* max length of file/directory name */
typedef struct {
    int inode;          /* inode number referenced by directory entry */

    char name[MAX_NAME + 1];     /* MAX_NAME chars + zero byte */
} dirent_t;
```

a) Vervollständigen Sie die Funktion `fs_alloc_inode()`, die einen freien Inode aus der Inode-Tabelle alloziert und dessen Nummer zurückgibt. **3 pt**

- Setzen Sie den Inode-Referenzzähler auf 1, die Länge auf 0 und konfigurieren Sie den Dateityp.
- Geben Sie -1 zurück, falls kein freier Inode existiert.

Complete the function `fs_alloc_inode()`, which allocates a free inode from the inode table and returns its number.

- *Set the inode's reference counter to 1, the length to 0, and configure the file type.*
- *Return -1 if no free inode exists.*

Solution:

```
int fs_alloc_inode(filesystem* fs, file_type type) {
    for (int n = 0; n < MAX_INODES; ++n) {
        inode_t* node = &fs->inodes[n];

        if (node->refcount == 0) {
            node->refcount = 1;
            node->length = 0;
            node->type = type;

            return n;
        }
    }

    return -1;
}
```

loop **(0.5 P)**, *find free inode* **(0.5 P)**, *configure inode* **(1.0 P)**, *return inode number* **(0.5 P)**, *return -1* **(0.5 P)**

b) Vervollständigen Sie die Funktion `fs_alloc_dirent()`, die einen `dirent_t`-Eintrag an ein Verzeichnis anfügt. **2 pt**

- `dir` ist die Inode-Nummer des Verzeichnisses, `inode` gibt die Inode-Nummer und `name` den Namen des neuen Eintrags an. Gehen Sie davon aus, dass dieser nicht länger als `MAX_NAME` ist und der Referenzzähler der Inode bereits angepasst ist.
- Nutzen Sie die Funktion `fs_pwrite()`, um Daten zu schreiben.

Complete the function `fs_alloc_dirent()`, which appends a `dirent_t` entry to a directory.

- *`dir` is the inode number of the directory, `inode` supplies the inode number and `name` specifies the name of the new entry. Assume that the name is not longer than `MAX_NAME` and that the reference counter of the inode has already been adjusted.*
- *Use the function `fs_pwrite()` to write data.*

```
/* writes count bytes of buf into the data area of the inode using the
   given offset (in bytes). Updates file length. Never fails. */
void fs_pwrite(filesystem* fs, int inode, void* buf, size_t count,
    off_t offset);
```

Solution:

```
void fs_alloc_dirent(filesystem* fs, int dir, int inode, const char* name) {
    dirent_t d = {0};
    d.inode = inode;
    strcpy(d.name, name);

    fs_pwrite(fs, dir, &d, sizeof(dirent_t), fs->inodes[dir].length);
}
```

initialize inode field **(0.5 P)**, *initialize string field* **(0.5 P)**, *write entry* **(1.0 P)**

c) Vervollständigen Sie die Funktion `fs_find()`, die in einem Verzeichnis nach einem passenden Eintrag zur gegeben Pfadkomponente sucht und deren Inode-Nummer zurückgibt. **4.5 pt**

- `dir` ist die Inode-Nummer des Verzeichnisses. Die Pfadkomponente umfasst die ersten `len` Zeichen des Strings `path`. Gehen Sie davon aus, dass `len` $\leq$ `MAX_NAME` ist.
- Nutzen Sie die Funktion `fs_pread()`, um Daten zu lesen.
- Geben Sie `-ENOENT` zurück, wenn kein passender Eintrag existiert.

Complete the function `fs_find()`, which searches in a directory for a matching directory entry to the given path component and returns its inode number.

- *`dir` is the inode number of the directory. The path component consists of the first `len` characters of the `path` string. Assume `len` $\leq$ `MAX_NAME`.*
- *Use the function `fs_pread()` to read data.*
- *Return `-ENOENT` if no matching entry exists.*

```
/* reads count bytes from inode's data area into buf using the given
   offset (in bytes). Returns the number of bytes read. Returns 0, when
   reading beyond the end of the data area; otherwise never fails. */
size_t fs_pread(filesystem* fs, int inode, void* buf, size_t count,
    off_t offset);
```

Solution:

```
int fs_find(filesystem* fs, int dir, const char* path, size_t len) {
    dirent_t d;
    off_t offset = 0;
    while (fs_pread(fs, dir, &d, sizeof(dirent_t), offset) > 0) {
        if ((strncmp(d.name, path, len) == 0) &&
            (d.name[len] == '\0')) {
            return d.inode; /* found! */
        }

        offset += sizeof(dirent_t);
    }

    return -ENOENT;
}
```

loop **(0.5 P)**, *read entry* **(1.0 P)**, *compare name* **(1.5 P)**, *increment offset* **(0.5 P)**, *return inode on success* **(0.5 P)**, *return -ENOENT otherwise* **(0.5 P)**

d) Vervollständigen Sie die Funktion `fs_open()`, die den absoluten Pfad `path` traversiert und die entsprechende Inode-Nummer der bezeichneten Datei zurückgibt. **5.5 pt**

- `inode` ist zu Beginn die Inode-Nummer des Wurzelverzeichnisses. Sie können `fs_open()` rekursiv aufrufen.
- Benutzen Sie die Funktion `split_path()`, um die nächste Pfadkomponente zu erhalten. Gehen Sie davon aus, dass diese nicht länger als `MAX_NAME` ist. Der Pfad hat die Form `/dir/file"` mit einer beliebigen Anzahl von Unterverzeichnissen.
- Sie können die Funktion `fs_find()` nutzen.
- Fehlt eine Pfadkomponente, geben Sie `-ENOENT` zurück.
- Enthält `flags` das `O_DIR` flag, öffnet die Funktion ausschließlich Verzeichnisse und gibt für Dateien `-ENOTDIR` zurück. Im umgekehrten Fall (d. h. `O_DIR` ist nicht gesetzt und das Ziel ist ein Verzeichnis) geben Sie `-EISDIR` zurück.

Complete the function `fs_open()`, which traverses the absolute path `path` and returns the inode number of the corresponding file.

- *Initially, `inode` is the inode number of the root directory. You may call `fs_open()` recursively.*
- *Use the function `split_path()` to get the next path component. Assume that each component is not longer than `MAX_NAME`. The path has the form `"/dir/file"` with an arbitrary number of subdirectories.*
- *You may use the function `fs_find()`.*
- *If a path component is missing, return `-ENOENT`.*
- *If `flags` contains the `O_DIR` flag, the function only opens directories and returns `-ENOTDIR` for files. In the opposite case (i.e., `O_DIR` is not set and `path` is a directory), return `-EISDIR`.*

```
/* returns the beginning of the next path component within the same path
   string. len returns the length of the path component in characters.
   The function skips leading path separators ('/'). Examples:
     "/dir/file" => "dir/file", len=3
     "file"      => "file", len=4
     ""          => "", len=0 */
const char* split_path(const char* path, size_t* len);

typedef enum {
    O_NONE  = 0,        /* open file entry, -ENOENT if not existing */
    O_DIR   = 1         /* open directory instead of file */
} open_flags;

/* returns the inode number of entry with name [path,path+len] in
   directory dir, or -ENOENT if no such entry exists. */
int fs_find(filesystem* fs, int dir, const char* path, size_t len);
```

Solution:

```
int fs_open(filesystem* fs, int inode, const char* path, open_flags flags) {
    inode_t* node = &fs->inodes[inode];

    size_t len;
    path = split_path(path, &len);
    if (len == 0) {
        if ((node->type == F_DIR) && !(flags & O_DIR)) {
            return -EISDIR;
        } else if ((node->type == F_FILE) && (flags & O_DIR)) {
            return -ENOTDIR;
        }

        return inode;
    }

    /* find directory entry for [path,path+len] in directory inode */
    int r = fs_find(fs, inode, path, len);
    if (r >= 0) {
        return fs_open(fs, r, path + len, flags);
    }

    /* not found */
    return -ENOENT;
}
```

call split_path() correctly **(1.0 P)**, *call fs_find() correctly* **(1.0 P)**, *traverse path correctly* **(2.0 P)**, *check for/return -EISDIR* **(0.5 P)**, *check for/return -ENOTDIR* **(0.5 P)**, *check for/return -NOENT* **(0.5 P)**

Total: 15.0pt