Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

Prof. Dr. Frank Bellosa
Peter Maucher, M.Sc.
Lukas Werling, M.Sc.
Yussuf Khalil, M.Sc.
Daniel Habicht, M.Sc.

Musterlösung Nachklausur
06.09.2024

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 19 Blättern: Einem Deckblatt, 18 Aufgabenblättern mit insgesamt 3 Aufgaben sowie 0 Seiten Man-Pages.

  The examination consists of 19 pages: One cover sheet, 18 sheets containing 3 assignments, and 0 sheets with man pages.

- Es sind keinerlei Hilfsmittel erlaubt!

  No additional material is allowed!

- Die Prüfung ist nicht bestanden, wenn Sie versuchen, aktiv oder passiv zu 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 assignment 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 mehreren 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	20	20	20	60
Erreichte Punkte				

Aufgabe 1: Virtualisierung

Assignment 1: Virtualization

a) Nennen Sie drei Dinge, die im Sinne der Vorlesung vom Betriebssystem virtualisiert werden, und erklären Sie kurz die Virtualisierung. **3 pt**

Name three things that are virtualized within the meaning of the lecture, and shortly explain the virtualization.

Solution:

- *Memory* **(0.5 P)***: physical memory is abstracted into a virtual address space* **(.5 P)**
- *CPU* **(0.5 P)***: the OS pretends that the application has the CPU to itself* **(0.5 P)**
- *Storage* **(0.5 P)***: the OS abstracts the storage medium into a file system* **(0.5 P)**
- *IO/Drivers/Devices* **(0.5 P)***: the OS hides implementation details and offers an interface to the application* **(0.5 P)**
- *Virtual File System* **(0.5 P)***: the OS hides implementation details and offers an interface to the application* **(0.5 P)**

b) Erklären Sie die Rolle des Loaders beim Starten eines Prozesses. **2 pt**

Explain the role of the loader when starting a process.

Solution:

The loader parses the executable file **(0.5 P)** *(*`.exe` *in Windows or (usually) extensionless in linux), and determines the sections* **(0.5 P)** *it finds. Some sections are copied into the address space of the newly created process* **(0.5 P)** *(e.g.,* `.text` *or* `.data`*), and some permissions might be additionally set* **(0.5 P)** *(e.g.,* `.rodata` *or again* `.text`*). Other segments, like* `.bss`*, are dynamically allocated and inserted into the address space* **(0.5 P)***. The loader then, depending on the mode, inserts the addresses into the global offset table* **(0.5 P)***, and finally jumps to the first instruction of the process* **(0.5 P)***.*

c) In dieser Aufgabe analysieren wir einen Scheduling-Algorithmus. Bevor mit der Analyse begonnen wird, implementieren Sie zwei Hilfsfunktion für die Verwaltung einer einfach verketteten Liste, welche dann im Algorithmus verwendet werden.

In dieser Aufgabe dürfen Sie davon ausgehen, dass alle Bibliotheksfunktionen erfolgreich ausgeführt werden, dass die Datenstruktur in einem korrekten Zustand war und dass alle Header, die Sie benötigen könnten, bereits inkludiert sind. Beachten Sie, dass alle Pointer korrekt gesetzt werden, sowie dass alle nicht mehr benötigten Ressourcen am Ende wieder freigegeben werden.

In this exercise, we analyse a scheduling algorithm. Before you start analysing, you are to implement two helper functions for the management of a singly linked list, which are used in the algorithm.

You can assume that all library functions execute successfully, that the data structure was in a correct state, and that all headers you might need are already included. Take care that all pointers are set correctly, and that all resources are cleaned up at the end.

Zunächst soll die Funktion `enqueue_back` implementiert werden, die ein `int` am Ende einer `singly_linked_list` einfügt. Beachten Sie, dass alle Pointer korrekt gesetzt werden, sodass das Element sowohl über die Liste als auch über den `end`-Pointer erreicht werden kann. **2 pt**

Eine leere Liste erkennen Sie daran, dass sowohl `start`- als auch `end`-Pointer auf `NULL` gesetzt sind.

First, implement the function `enqueue_back`, which inserts an `int` at the end of a `singly_linked_list`. Take care that all pointers are set correctly, so that the element can both be accessed via the list as well as the `end` pointer.

You can identify an empty list by it having both `start` and `end` pointer set to `NULL`.

```
#include <stdlib.h>
#include <stdbool.h>

struct sll_entry {
    // NULL if no more entries
    struct sll_entry *next;
    int payload;
};
struct singly_linked_list {
    struct sll_entry *start;
    struct sll_entry *end;
};

typedef struct sll_entry sll_entry;
typedef struct singly_linked_list singly_linked_list;

void enqueue_back(singly_linked_list *list, int entry) {
    sll_entry *wrapper = malloc(sizeof(struct sll_entry)); // .5P
    wrapper->payload = entry;
    if (list->start != 0) { // .5P
        list->end->next = wrapper; // .5P
        list->end = wrapper;       //
    } else {
        wrapper->next = 0;                  // .5P
        list->end = list->start = wrapper; //
    }
}
```

Als weitere Hilfsfunktion soll `dequeue_front` implementiert werden, die das erste `int`-Objekt am Anfang der `singly_linked_list` wieder entfernt. Beachten Sie die Hinweise aus der letzten Teilaufgabe sowie der Aufgabenbeschreibung. **2 pt**

Additionally, implement the helper function `dequeue_front`, which removes the first `int` object at the beginning of a `singly_linked_list`. Take note of the hints in the previous subquestion and the introductory text.

```
int dequeue_front(singly_linked_list *list) {
    sll_entry *result = list->start; // .5P
    list->start = result->next; // .5P
    if (!list->start) list->end = 0; // .5P
    int value = result->payload;
    free(result); // .5P
    return value;
}
```

Betrachten Sie den folgenden Scheduling-Algorithmus. Geben Sie die ersten zehn Scheduling-Entscheidungen für die unten vorgegebene Initialisierung an. Nehmen Sie an, dass anfangs ein Prozess mit ID `00` läuft. **1.5 pt**

Consider the following scheduling algorithm. State the first ten scheduling decisions for the initialization given below. Assume that a process with id `00` is initially running.

```
#define NUM_ENTRIES 3
static int index = 0;
static singly_linked_list scheduler_lists[NUM_ENTRIES] = {0};
int schedule_next_task(int last) {
    enqueue_back(scheduler_lists + index, last);
    index += 1;
    for (int i = 0; i < NUM_ENTRIES; ++i) {
        if (scheduler_lists[(index + i) % NUM_ENTRIES].start != 0) {
            index = (index + i) % NUM_ENTRIES;
            return dequeue_front(scheduler_lists + index);
        }
    }
    return 0;
}
```

Initialisierung / *Initialization*:

```
enqueue_back(scheduler_lists + 0, 01);
enqueue_back(scheduler_lists + 0, 02);
enqueue_back(scheduler_lists + 0, 03);
enqueue_back(scheduler_lists + 0, 04);
enqueue_back(scheduler_lists + 1, 10);
enqueue_back(scheduler_lists + 1, 11);
enqueue_back(scheduler_lists + 1, 12);
enqueue_back(scheduler_lists + 2, 20);
enqueue_back(scheduler_lists + 2, 21);
```

Scheduling Cycle	0	1	2	3	4	5	6	7	8	9	10
Process ID	00	10	20	01	11	21	02	12	20	03	10

Solution:

- **(0.5 P)** *for sequential behavior in one list*
- **(0.5 P)** *for sequential behavior between the lists*
- **(0.5 P)** *for circular behavior*

d) Beschreiben Sie das M-to-N-Modell bei Threads, und geben Sie einen Nachteil gegenüber Kernel-Level-Threads an. **1.5 pt**

Describe the M-to-N model for threads, and name one disadvantage as compared to kernel level threads.

Solution:

In the M-to-N model, user space and kernel space cooperate **(0.5 P)** *in the scheduling decisions. A set of user level threads is handled by (usually fewer) kernel level threads* **(0.5 P)**, *and user space is informed when scheduling decisions need to be made* **(0.5 P)**. *Problems include handling synchronous IO and blocking system calls* **(0.5 P)**, *as well as the need for upcalls* **(0.5 P)**.

e) Geben Sie außer Swapping zwei Fälle an, in denen Page Faults auftreten. **1 pt**

Except swapping, name two cases in which page faults occur.

Solution:

- *Access to illegal addresses* **(0.5 P)** *(not mapped, or wrong permissions)*
- *Nullpointer dereference (also gives points as this might also indicate missing error handling)* **(0.5 P)**
- *Demand paging* **(0.5 P)**
- *Copy on write* **(0.5 P)**

No points awarded for

- *Anything that amounts to swapping or page replacement*
- *TLB Misses or software page table walks*

Beschreiben Sie den Ablauf im Betriebsystem, welcher bei einem Speicherzugriff auf eine ausgelagerte Seite (*swapping*) durchlaufen wird. Nehmen Sie hierfür zunächst an, dass genügend freier physischer Speicher vorhanden ist. Schreiben Sie entweder einen Text oder zeichnen eine Grafik auf einer der Rückseiten der Klausur. Markieren Sie klar, wo und was Ihre Antwort ist! **2 pt**

Describe the procedure in the operating system for a memory access to a swapped out page, For now, assume that enough physical memory is available. Either write a text or draw a figure on one of the backsides of this exam. Clearly mark your answer and where you put it!

Solution:

First, a page fault occurs **(0.5 P)**, *which traps into the OS. The OS then figures out that the access is actually legal* **(0.5 P)**, *but that the page was swapped out* **(0.5 P)**. *The OS then allocates a free page frame* **(0.5 P)**, *copies the contents from the backing storage into that page frame* **(0.5 P)**, *inserts the page frame into the page table* **(0.5 P)** *and finally restarts the instruction* **(0.5 P)**.

Welche zusätzlichen Schritte muss das Betriebssystem durchführen, wenn kein nicht genug physischer Speicher zur Verfügung steht? Beschreiben Sie oder erweitern Sie ihre Grafik aus der vorherigen Teilaufgabe. Markieren Sie in diesem Fall ihre Erweiterungen der Grafik klar als solche. **2 pt**

Which additional steps does the operating system need to execute if not enough physical memory is available? Describe or extend your figure from the previous exercise. In this case clearly mark your extensions in the figure.

Solution:

Before you can allocate an empty page frame, you need to reclaim it **(0.5 P)**: *Firstly, the OS decides which used frame to reuse* **(0.5 P)**, *unmaps the frame from the address space(s) it is currently used in* **(0.5 P)**, *copies the contents to the swap area* **(0.5 P)**, *and then initialized the reclaimed page frame for the new process.*

f) Erklären Sie den Unterschied zwischen synchroner und asynchroner IO, und beschreiben Sie, wie man das eine durch das jeweils andere abbilden kann. **3 pt**

Describe the difference between synchronous and asynchronous IO, and describe how you can emulate one with the respective other.

Solution:

In case of synchronous IO, every request blocks **(0.5 P)***, that is the system call stops the thread until the request has completed* **(0.5 P)***. In case of asynchronous IO however, the system call returns immediately* **(0.5 P)***, and the process needs to poll* **(0.5 P)** *to determine completion. One can emulate synchronous IO with asynchronous IO using busy-waiting* **(0.5 P)***, and asynchronous IO using synchronous IO by spawning an extra thread for the request* **(0.5 P)***.*
Common misconception: *Synchronous and asynchronous IO are a user-space concept. In the kernel, even when polling the actual completion is independent from the kernel or even the CPU, and has to be awaited. An interrupt only relaxes the necessity to actively wait.*

Total: 20.0pt

Aufgabe 2: Nebenläufigkeit

Assignment 2: Concurrency

a) Erläutern Sie die zwei Phasen einer *Two-Phase Lock* (`futex`). **2 pt**

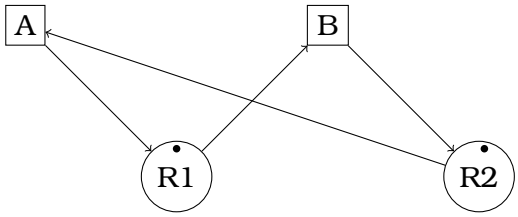
Explain the two phases of a Two-Phase Lock *(`futex`).*

Solution:

*In **phase 1**, the program tries to acquire a spinlock in userspace* **(1 P)***. If that fails, the program enters **phase 2** and instead uses a blocking system call-based lock* **(1 P)***.*

b) Zeichen Sie einen *Resource Allocation Graph*, der ein Deadlock darstellt. **2 pt**

Draw a Resource Allocation Graph *that represents a deadlock.*



c) Nennen Sie die vier notwendigen Bedingungen für Deadlocks. **2 pt**

Name the four necessary conditions for deadlocks.

Solution:

Mutual exclusion, hold and wait, no preemption, circular wait. **(0.5 P)** *each.*

```
1  #include <sched.h>
2
3  struct account {
4      int id;
5      int balance;
6  };
7
8  void update_balance(struct account *acc, int amount) {
9      acc->balance += amount;
10 }
11
12 void do_transaction(struct account *from, struct account *to, int amount) {
13     update_balance(from, -amount);
14     update_balance(to, amount);
15     sched_yield();
16 }
```

d) Betrachten Sie das obige Programm. Nehmen Sie an, dass für jede Transaktion jeweils ein neuer Thread gestartet und in diesem `do_transaction()` ausgeführt wird. Das Programm funktioniert auf einem Single-Core-Prozessor einwandfrei. Es wird neue Hardware gekauft und das System hat nun einen Multi-Core-Prozessor, das Betriebssystem allerdings bleibt das gleiche. Plötzlich funktioniert das Programm nicht mehr wie erwartet. Was könnte der Grund dafür sein? Was sagt das über das verwendete Betriebssystem aus? Nennen Sie außerdem die konkrete Zeilennummer, in der das Problem auftritt. **3 pt**

Consider the program above. Assume that for each transaction a new thread is started and `do_transaction()` is executed in this thread. The program is working fine on a single-core processor. New hardware is bought and now the system has a multi-core processor, but the operating system remains the same. Suddenly, the program is not working as expected anymore. What could be the reason for this? What does this say about the operating system in use? Also, name the specific line number where the problem occurs.

Solution:

The various threads of the program are now running in parallel on different cores. This can lead to race conditions when accessing the same data simultaneously **(1 P)**. *The operating system likely does not use preemptive scheduling, otherwise the race conditions would have been detected earlier* **(1 P)**. *The problem occurs in line 9* **(1 P)**.

e) Modifizieren Sie den Code so, dass er auch auf einem Multi-Core-Prozessor korrekt funktioniert. Falls Sie dafür eine oder mehrere Zeile(n) zwingend entfernen müssen, streichen Sie diese durch. Wenn nötig, fügen Sie Code zur Initialiserung in `initialize_account()` hinzu. **3 pt**

Modify the code so that it works correctly on a multi-core processor. If you have to remove one or more line(s) for this, strike them through. If necessary, add initialization code in `initialize_account()`.

Solution:

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

struct account {
    int id;
    int balance;
    pthread_mutex_t lock;
};

void update_balance(struct account *acc, int amount) {
    pthread_mutex_lock(&acc->lock);
    acc->balance += amount;
    pthread_mutex_unlock(&acc->lock);
}

void do_transaction(struct account *from, struct account *to, int amount) {
    update_balance(from, -amount);
    update_balance(to, amount);

    sched_yield();
}

void initialize_account(struct account *acc) {
    pthread_mutex_init(&acc->lock, NULL);
}
```

- **(0.5 P)** *for adding the lock to* `struct account`
- **(0.5 P)** *for initializing the lock in* `initialize_account()`
- **(1 P)** *for acquiring the lock in* `update_balance()`
- **(1 P)** *for releasing the lock in* `update_balance()`
- **(-1 P)** *if any code was removed*
- **(-1 P)** *if* `pthread.h` *is missing*
- **(-0.5 P)** *if lock is added as a pointer without proper initialization of the underlying memory (i.e., missing* `malloc()`*)*

f) In einem weiteren Hardwareupgrade erhält das System einen Prozessor, der *transaktionalen Speicher* unterstützt. Hierbei können innerhalb einer Transaktion mehrere Speicherzugriffe atomar ausgeführt werden. Erst beim *Commit* werden die innerhalb der Transaktion modifizierten Speicherbereiche aktualisiert. Sollte ein anderer Prozessorkern die Speicherbereiche gleichzeitig modifizieren, so schlägt der *Commit* fehl und die Transaktion wird abgebrochen und muss wiederholt werden. **2 pt**

Schreiben Sie `do_transaction()` so um, dass es transaktionalen Speicher verwendet unter Zuhilfenahme der bereitgestellten Funktionen. Nutzen Sie nur so viele Transaktionen wie zwingend nötig.

In another hardware upgrade, the system receives a processor that supports transactional memory*. With this, multiple memory accesses can be executed atomically within a transaction. Only when* committing*, the memory areas modified within the transaction are updated. If another processor core modifies the memory areas at the same time, the* commit *fails and the transaction is aborted and must be repeated.*

Rewrite `do_transaction()` *to use transactional memory using the provided functions. Use only as many transactions as absolutely necessary.*

Solution:

```
void do_transaction(struct account *from, struct account *to, int amount) {
    do {
        begin();
        update_balance(from, -amount);
        update_balance(to, amount);
    } while (!commit());
}
```

- **(0.5 P)** *for correctly calling* `begin()`
- **(0.5 P)** *for correctly calling* `commit()`
- **(1 P)** *for looping correctly*
- **(-0.5 P)** *if more than one transaction is used*

Jemand in Ihrem Team schlägt ein alternatives Programmdesign vor, wo nicht ein Thread pro Transaktion gestartet wird, sondern für jedes Konto (`struct account`) wird zum Programmstart jeweils ein einzelner eigener Thread initialisiert. Die Threads arbeiten jeweils die anstehenden Kontostandsaktualisierungen für ihr jeweiliges Konto ab. Die entsprechenden Aktualisierungsaufgaben werden dazu in einem Puffer abgelegt, der eine fixe Anzahl an Einträgen aufnehmen kann. Eine beliebige Zahl anderer Threads fügt Aufgaben in diesen Puffer ein.

Someone in your team suggests an alternative program design where not one thread is started per transaction, but at program start, a single thread is initialized for each account (`struct account`*). The threads each process the pending account balance updates for their respective account. The corresponding update tasks are stored in a buffer that can hold a fixed number of entries. Any number of other threads add tasks to this buffer.*

g) Bei Verwendung des Puffers ergibt sich ein klassisches Synchronisationsproblem. Benennen Sie dieses. **1 pt**

When using the buffer, a classic synchronization problem arises. Name it.

Solution:

The producer-consumer problem *(or* bounded-buffer problem*).*

h) Diskutieren Sie die Eignung der folgenden drei Synchronisationsprimitive für die Lösung des Problems: *Mutex*, *Condition Variable*, *Semaphore*. Legen Sie dar, welches am besten geeignet ist und warum, und erläutern Sie jeweils einen Nachteil der anderen beiden. **3 pt**

Discuss the suitability of the following three synchronization primitives for solving the problem: mutex, condition variable, semaphore*. Explain which one is best suited and why, and explain the disadvantages or problems for each of the other two.*

Solution:

Semaphores are best suited as they carry a state, namely the amount of elements in the buffer, ensuring that the buffer's limits are correctly enforced. Also, they ensure that the thread is woken up as soon as at least one element is added to the buffer.

Mutexes are unsuited as there is no solution that correctly ensures that the program is free from race conditions and deadlocks, but also wakes up correctly in every case. Condition variables would work, however, they do not carry a state, and therefore it would be required for the programmer to track the state separately.

See lecture slides for code examples.

i) Benennen und erklären Sie ein weiteres klassisches Synchronisationsproblem. **2 pt**

Name and explain another classic synchronization problem.

Solution:

- *Readers-writers problem: many threads compete to read or write the same data.*
- *Dining philosophers problem: multiple threads compete for multiple resources and can deadlock each other.*

(0.5 P) *for naming the problem,* **(1.5 P)** *for a correct explanation.*

Total: 20.0pt

Aufgabe 3: Persistenz

Assignment 3: Persistence

In dieser Aufgabe implementieren Sie eine vereinfachte Variante vom Kommandozeilenwerkzeug `du`, um für ein gegebenes Verzeichnis den auf dem Hintergrundspeicher allozierten Platz für alle enthaltenen Dateien rekursiv zu ermitteln. Für die Teilaufgaben a) bis d) sollen Sie annehmen, dass für jeden vorkommenden *inode* exakt ein Verzeichniseintrag existiert und dass alle Bibliotheksaufrufe bzw. Systemaufrufe erfolgreich sind, sofern keine ungültigen Argumente verwendet werden. **Es muss keine Fehlerbehandlung implementiert werden**. Weiter können Sie annehmen, dass alle notwendige Header bereits inkludiert sind.

In this exercise, you implement a simple version of the command line utility `du` for calculating the allocated disk space of all files in a given directory recursively. For exercise a) to d), you should assume that for each inode there is exactly one directory entry, and that all library calls and system calls complete successfully if you do not use invalid arguments. ***You do not have to implement any error handling.*** *Further, you may assume that all necessary header files are already included.*

a) Implementieren Sie die Funktion `get_alloced_size()`, welche die Menge an allozierten Platz in Bytes auf dem Hintergrundspeicher für die im Dateideskriptor `fd` übergebene Datei ermittelt. Der Dateideskriptor darf nicht in `get_alloced_size()` geschlossen werden. Sie dürfen annehmen, dass `fd` ein gültiger Dateideskriptor auf eine reguläre Datei ist. **2 pt**

Hinweis: der allozierte Platz entspricht nicht zwangsläufig der Dateigröße.

Implement the function `get_alloced_size()` that returns the allocated disk space in bytes for the file passed in the file descriptor `fd`. You must not close the file descriptor in `get_alloced_size()`. You may assume that `fd` is a valid file descriptor on a regular file.

Hint: the allocated space does not necessarily equal the file size.

Solution:

```
size_t get_alloced_size(int fd) {
    struct stat sb;
    fstat(fd, &sb);
    return sb.st_blocks << 9;
}
```

`fstat` call **(1 P)**, *allocated file size (`st_blocks`) returned* **(0.5 P)** *as number of bytes* **(0.5 P)**

b) Weshalb könnte die Dateigröße von der Menge an alloziertem Platz auf dem Hintergrundspeicher abweichen? Nennen Sie einen möglichen Grund. **1 pt**

Why could the file size differ from the allocated disk space? Give one possible reason.

Solution:

- *Implementation of sparse files*
- *File system / OS allocates entire disk sectors (or multiples of the sector size) for files. The File size might not be aligned to the sector/block size (internal fragmentation).*
- *Preallocation of blocks to combat fragmentation (e.g., XFS speculative prealloc)*

c) Implementieren Sie die Funktion `is_special_name()`, welche ermittelt, ob es sich beim Dateinamen `name` um einen der beiden Verzeichniseinträge `"."` oder `".."` handelt. Für die beiden Namen `"."` und `".."` soll 1 zurückgegeben werden, ansonsten 0. `name` ist ein nullterminierter String. **1 pt**

Implement the `is_special_name()` function that determines if the file name `name` matches on of the directory entries `"."` or `".."`. For the names `"."` and `".."`, return 1, otherwise 0. `name` is a null-terminated string.

Solution:

```
int is_special_name(const char *name) {
    return !strcmp(name, ".") || !strcmp(name, "..");
}
```

correct classification of name for all possible inputs **(1 P)**,

d) Vervollständigen Sie die Funktion `walk_dir()`, welche die Summe des allozierten Platzes auf dem Hintergrundspeicher über alle Dateien im von `dir_fd` beschriebenen Verzeichnis rekursiv ermittelt. Geben Sie den ermittelten Platzverbrauch in Bytes zurück. Sie dürfen annehmen, dass `dir_fd` ein gültiger Dateideskriptor auf ein Verzeichnis ist. **4.5 pt**

- Verwenden Sie für die Ermittlung des Platzverbrauchs einer Datei die in a) implementierte `get_alloced_size()`.
- Überspringen Sie besondere Verzeichnisse (`"."` und `".."`). Verwenden Sie hierfür `is_special_name()` aus Teilaufgabe b).
- Geben Sie alle in dieser Funktion allozierten Ressourcen (z.B. Dateideskriptoren) vor dem Verlassen der Funktion frei. `dir_fd` muss vom Aufrufer freigegeben werden.

Hinweise:

- Sie können `openat()` verwenden, um einen von `dir_fd` ausgehenden relativen Pfad zu öffnen.
- `closedir()` gibt den zugrundeliegenden Dateideskriptor frei.
- Das Öffnen eines Verzeichnisses für schreibenden Zugriff ist verboten.

Complete the function `walk_dir()` that calculates the sum of allocated disk space over all files contained in the directory described by `dir_fd` recursively. Return the calculated disk usage in bytes. You may assume that `dir_fd` is a valid file descriptor on a directory.

- *For calculating the disk usage of a file, use `get_alloced_size()` implemented in exercise a).*
- *Skip special directories (`"."` and `".."`) using `is_special_name()` implemented in exercise b).*
- *Free all resources (e.g., file descriptors) allocated in this function before returning. `dir_fd` must be freed by the caller.*

Hints:

- *You can use `openat()` for opening a path relative to `dir_fd`.*
- *`closedir()` closes the underlying file descriptor.*
- *Opening directories for writing is forbidden.*

Solution:

```
size_t get_alloced_size(int fd);
int is_special_name(const char *name);

size_t walk_dir(int dir_fd)
{
    size_t size = 0;

    DIR *dirp = fdopendir(dup(dir_fd)); // open dir stream

    struct dirent *de;
    while ((de = readdir(dirp)) != NULL) {

        if (is_special_name(de->d_name))
            continue;

        int fd;
        switch(de->d_type) {
        case DT_DIR:
            fd = openat(dir_fd, de->d_name, O_PATH);
            size += walk_dir(fd);
            close(fd);
            break;
        case DT_REG:
            fd = openat(dir_fd, de->d_name, O_PATH);
            size += get_allocated_file_size(fd);
            close(fd);
            break;
        default:
            break;
        }
    }

    closedir(dirp);
    return size;
}
```

- *iterate through dir with* `readdir()` **(1 P)**
- *skip special directory entries (*`"."` *and* `".."`*)* **(0.5 P)**
- `openat()` *on file/directory with valid open mode* **(1 P)**
- *call* `walk_dir()` *for directories* **(0.5 P)**
- *call* `get_alloced_size()` *for regular files* **(0.5 P)**
- *close all opened file descriptors* **(0.5 P)** *(only applies when at least one additional fd opened)*
- *use* `d_type` *in* `switch` *statement* **(0.5 P)**
- *Valid* `openat` *modes:*
  - `O_PATH` *with and without additional* `O_RDONLY` *valid*
  - `O_DIRECTORY` *not required (no error handling) and only valid for opening directories*
  - *using only* `O_RDONLY` *valid for both* `openat()` *calls*
  - `O_RDWR` *not allowed on directories, acceptable on regular file only*
  - *empty open mode, i.e.,* `openat(fd, 0)`*, invalid*

e) Implementieren Sie die Funktion `calc_disk_usage()`, welche die Anzahl an allozierten Bytes mittels der in c) implementierten `walk_dir()` für das durch `path` beschriebene Verzeichnis ermittelt. Sie dürfen annehmen, dass `path` ein gültiges Verzeichnis beschreibt. Geben Sie alle in dieser Funktion allozierten Ressourcen vor dem Verlassen frei. **2 pt**

Implement the function `calc_disk_usage()` that calculates the number of allocated bytes for the directory described by `path` using `walk_dir()` from exercise c). You may assume that `path` describes a valid directory. Free all resources allocated in this function before returning.

Solution:

```
size_t get_alloced_size(int fd);
int is_special_name(const char *name);
size_t walk_dir(int dir_fd);

size_t calc_disk_usage(const char *path) {
    int dir_fd = open(path, O_PATH);
    size_t size = walk_dir(dir_fd);
    close(dir_fd);
    return size;
}
```

- *open path* **(0.5 P)**, *see solution of 3 d) for discussion on valid open mode*
- *call `walk_dir()` on opened path* **(0.5 P)**
- *close fd before returning* **(0.5 P)**, *only applies when new fd opened*
- *return size returned by `walk_dir()`* **(0.5 P)**

f) Beschreiben Sie den Unterschied zwischen symbolischen Links und *hard links*. Geben Sie zudem an, auf welcher Ebene des Betriebssystems symbolische Links aufgelöst werden. **1.5 pt**

Describe the difference between symbolic links and hard links. Further, name the layer of the operating system that resolves symbolic links.

Solution:

- *symbolic links are special files that contain a path* **(0.5 P)** *while hard links are directory entries pointing to an inode* **(0.5 P)**
- *Deleting a symbolic link does not affect the linked file (link may point to nonexistent files/directories), deleting the last hard link frees the underlying file (a hard link cannot point to a nonexistent resource)*
- *symbolic links are resolved by the VFS* **(0.5 P)**

g) Nehmen Sie an, dass die Menge an *hard links* pro *inode* nicht auf eins beschränkt ist und dass ausschließlich reguläre Dateien und Verzeichnisse existieren. Ist es möglich, dass `calc_disk_usage()` bzw. `walk_dir()` nicht den korrekten Platzverbrauch berechnen? Begründen Sie Ihre Antwort. **1 pt**

Assume that the number of hard links per inode is not limited to one and that only regular files and directories exist. Is it possible that `calc_disk_usage()` or `walk_dir()` do not calculate the correct disk usage? Justify your answer.

Solution:

If the given path contains two (or more) hard links pointing to the same inode **(0.5 P)**, *the allocated storage space of this file is counted twice (multiple times)* **(0.5 P)**. *Therefore, `calc_disk_usage()` is only correct under the assumption of a single hard link per inode.*

h) Nun sollen auch symbolische Links bei der Ermittlung des verbrauchten Hintergrundspeicherplatzes unterstützt werden. Wieso reicht es für eine korrekte Implementation nicht aus, den symbolischen Link aufzulösen und wie eine reguläre Datei oder ein Verzeichnis zu behandeln? Beschreiben Sie ein mögliches Problem. **1 pt**

Now, we want to also support symbolic links when calculating the disk usage. Why does it not suffice for a correct implementation to simply resolve a symbolic link and handle it like a regular file or a directory? Describe one possible problem.

Solution:

- *Symbolic links could create cycles in the file system hierarchy. A correct implementation would have to detect cycles and only recuse into each subdirectory once.*
- *Symbolic links could point to files whose disk usage is already accounted for.*
- *The resource that the symbolic link resolves to could be unavailable/nonexistent.*
- *The symbolic link consumes storage itself (only the allocated space of the linked file is accounted)*

i) Nennen Sie den Zweck, welchen das *Flash Translation Layer (FTL)* einer SSD erfüllt. **0.5 pt**

Give the purpose that the flash translation layer (FTL) of a SSD fulfills.

Solution:

enables wear leveling for SSDs

j) Beschreiben Sie das Problem, welches beim Überschreiben von Daten auf einer HDD mit *Shingled Magnetic Recording (SMR)* auftreten kann. **0.5 pt**

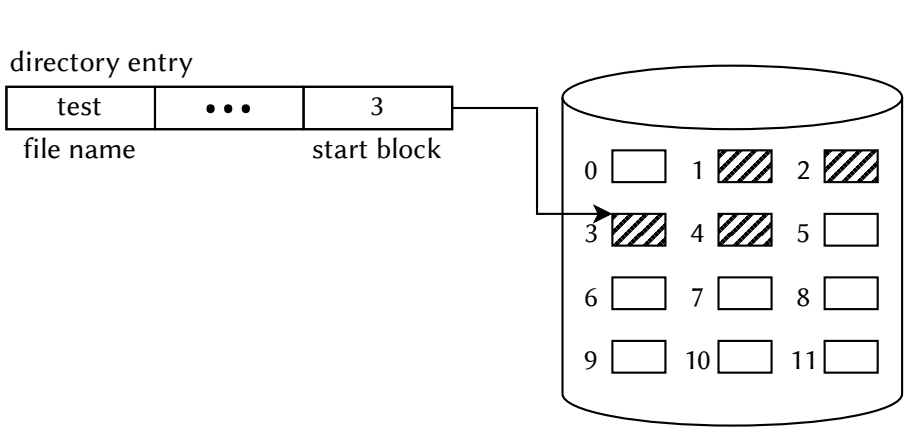
Describe the problem that may occur when overwriting data on an HDD with shingled magnetic recording (SMR).

Solution:

Data overlapping the to-be-written tracks requires a rewrite (this leads to write amplification and reduced performance).

k) Geben Sie an, welche Dateiallokationsstrategie in der folgenden Abbildung dargestellt wird. Geben Sie zudem an, wie viele Zugriffe auf den Hintergrundspeicher notwendig sind, um den vierten Block der Datei "test" (Startblock 3) zu lesen. Nehmen Sie hierfür an, dass durch Caching im Hauptspeicher keine weiteren Zugriffe für die Dateitabelle (*file table*) anfallen. **1 pt**

Give the name of the file allocation strategy depicted in the following figure. Further, give the number of disk accesses that are required for reading the fourth block of the file "test" (start block 3). For this, you shall assume that all accesses on the file table do not require additional disk accesses due to caching in system memory.



file table

block	next
0	⊥
1	2
2	4
3	1
4	⊥
5	⊥
6	⊥
7	⊥
8	⊥
9	⊥
10	⊥
11	⊥

Solution:

File Allocation Strategy: *chained allocation with FAT in RAM* **(0.5 P)** *(or File Allocation Table (FAT)).*

Number of Disk Accesses: *1 (accesses on file table do not count as we assumed that the table is cached in RAM)* **(0.5 P)**

Geben Sie einen Vor- und einen Nachteil der oben gezeigten Dateiallokationsstrategie an. **1 pt**

Give one advantage and one disadvantage of the file allocation strategy shown above.

Solution:

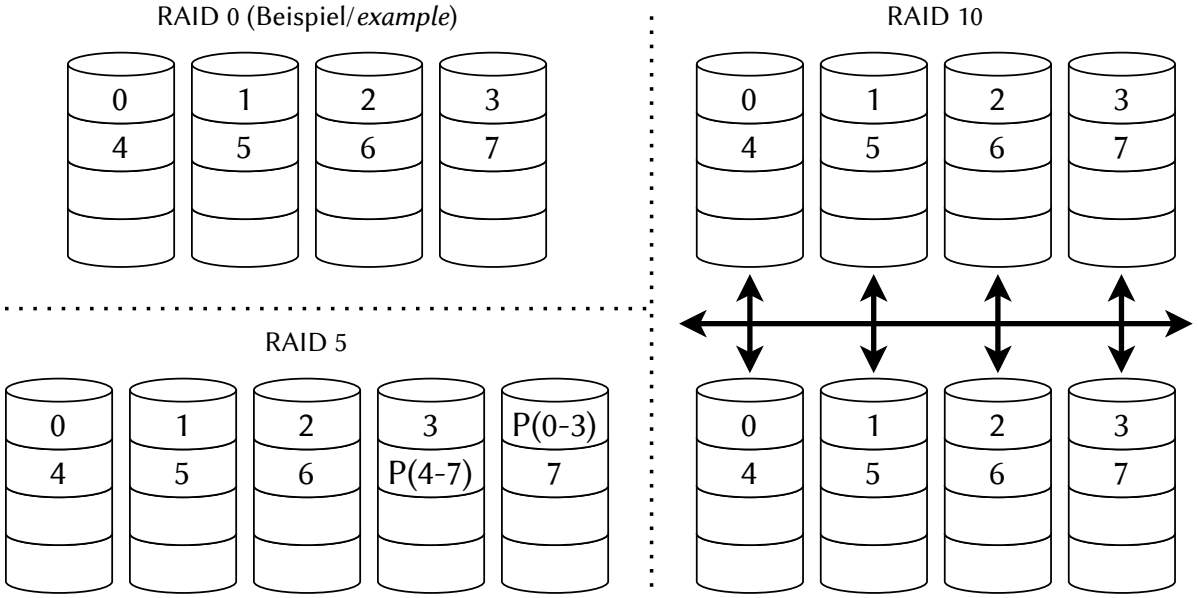
(+): *efficient random access* **(0.5 P)**

(-): *FAT size depends on disk size –> inefficient for large storage devices* **(0.5 P)**

l) In der folgenden Abbildung finden Sie die Verteilung von den ersten acht Datenblöcken in einem RAID 0-Verbund über vier Hintergrundspeicher. Vervollständigen Sie die Verteilung für die ersten acht Datenblöcke für RAID 5 und RAID 10. Sollten Paritätsblöcke in einem der RAID-Level verwendet werden, sollen die Paritätsblöcke in dieser Aufgabe über vier Datenblöcke gebildet werden. Ein Paritätsblock über die Blöcke n bis m soll als P(n - m) angegeben werden. Gehen Sie beim Verteilen der Blöcke erst von links nach rechts, dann von oben nach unten vor. RAID 0 dient hierfür als Beispiel. **2 pt**

The following figure shows the distribution of the first eight data blocks in a RAID 0 array with four disks. Complete the distribution of the first eight data blocks for RAID 5 and RAID 10. If a RAID level requires parity blocks, assume that parity blocks are formed over four data blocks. Use P(n - m) as notation for a parity block over the blocks n to m. When distributing the blocks, first go from left to right, then from top to bottom. RAID 0 serves as example for this.

Solution:



- **(1 P)** *for each correct RAID-level.*

Erklären Sie in wenigen Worten, in welchen RAID-Konfigurationen ein RAID 10- einem RAID 01-Verbund vorzuziehen ist und warum das der Fall ist. **1 pt**

Explain in few words in which RAID configuration a RAID 10 setup should be preferred over a RAID 01 setup and why this is the case.

Solution:

RAID 10 has a lower probability of failure than RAID 01 **(0.5 P)** *for 6 or more drives* **(0.5 P)**.

For RAID 10, the array only fails when two disks in the same group (i.e., the disk and its mirror) fail. For RAID 01, the array fails when one disk in each RAID-0 group fails. Thus, RAID 10 has a lower probability of failure in most configurations.

Total: 20.0pt