Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

Prof. Dr. Frank Bellosa
Peter Maucher, M.Sc.
Lukas Werling, M.Sc.

Musterlösung Hauptklausur
01.03.2023

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 30 Blättern: Einem Deckblatt, 29 Aufgabenblättern mit insgesamt 5 Theorieaufgaben und 3 Programmieraufgaben sowie 0 Seiten Man-Pages.

  The examination consists of 30 pages: One cover sheet, 29 sheets containing 5 theory assignments as well as 3 programming 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	T1	T2	T3	T4	T5	P1	P2	P3	Total
Max. Punkte	9	9	9	9	9	15	15	15	90
Erreichte Punkte									

Aufgabe T1: Grundlagen

Assignment T1: Basics

a) Beschreiben Sie den Unterschied zwischen Programm und Prozess. **1 pt**

Describe the difference between program and process.

Solution:

A program is an entity stored on the drive, which can be loaded and executed, **(0.5 P)** *and a process is a single running instance which executes the code of a program* **(0.5 P)**.

b) Nennen Sie zwei Arten privilegierter CPU-Instruktionen und erklären Sie jeweils, warum diese Instruktionen privilegiert sein müssen. **2 pt**

Name two types of privileged CPU instructions. For each type, explain why these instructions must be privileged.

Solution:

Instructions that manipulate control registers for memory translation *must be privileged to ensure memory isolation. Otherwise, processeses could access arbitrary physical memory.*

Instructions that disable or enable interrupts. *With disabled interrupts, a process could run forever since the scheduler would never be invoked.*

Instructions that access platform devices. *The operating system must be able to mediate access to devices without interference from user processes.*

c) Warum führt Programmed I/O üblicherweise zu hoher Prozessorauslastung, wenn das Betriebssystem schnell auf Eingaben von Geräten reagieren soll? **1 pt**

Why does programmed I/O cause high CPU utilization if the OS should quickly react to device input?

Solution:

To be able to quickly react to input from devices, the driver needs to poll **(0.5 P)** *the device frequently to check its state: The CPU is busy-waiting and, in a loop, reading the status of the I/O device or operation* **(0.5 P)**, *therefore CPU utilization is high.*

d) Erklären Sie den Unterschied zwischen freiwilligen und unfreiwilligen Aufrufen des Betriebssystems und nennen Sie je ein Beispiel. **2 pt**

Explain the difference between voluntary and involuntary calls of the operating system and name one example each.

Solution:

In a voluntary jump to the OS, the userspace application intends to call the kernel **(0.5 P)**, *e.g. with a system call* **(0.5 P)**. *In an involuntary jump to the OS, the userspace application intended to continue computing* **(0.5 P)**, *but either a CPU interrupt occured* **(0.5 P)** *or the application triggered an exception* **(0.5 P)** *like an illegal-instruction exception or a page fault.*

Common errors:

· *For the involuntary case, either interrupt or exception were given as an explanation, but not both, and without the actual distinction, this answer is not accepted.*

· *Voluntary and involuntary are independent of synchronous and asynchronous.*

e) Beim Kopieren einer großen Datei von einer SSD auf eine Festplatte liegt die Kopiergeschwindigkeit für einige Zeit deutlich über der maximalen Schreibgeschwindigkeit der Festplatte und fällt dann abrupt auf das Maximum ab. Wie erklären Sie diese Beobachtung? **2 pt**

While copying a large file from an SSD to an HDD the copy speed greatly exceeds the HDD's maximum write speed and then suddenly drops to the maximum. How do you explain this?

Solution:

The file is copied using the operating system's ***file system cache****, caching the destination file. During the first period there is still free RAM to let the cache grow and use the full read speed of the SSD. However, the slow HDD cannot keep up with the high rate of incoming data and eventually the cache grows to its maximum size, capping the read speed to the HDD's write speed (i.e., the speed with which memory in the cache can be replaced).*

Answers that referred to the HDD's internal cache were also accepted.

f) Nennen Sie die zwei grundsätzlichen Kommunikationsmodelle der Interprozesskommunikation. Hinweis: Gefragt sind nicht die verschiedenen Designparameter, sondern die grundlegenden Modelle. **1 pt**

Enumerate the two fundamental models of interprocess communication (IPC). Note: We do not ask for the various design parameters, but for the fundamental models.

Solution:

- *shared memory* **(0.5 P)**
- *message passing* **(0.5 P)**

Total: 9.0pt

Aufgabe T2: Prozesse und Threads

Assignment T2: Processes and Threads

a) Erläutern Sie den Unterschied zwischen Nebenläufigkeit und Parallelismus. **2 pt**

Explain the difference between concurrency and parallelism.

Solution:

Concurrency only means that multiple processes may co-exist at the same time on a system. Parallelism, in contrast, means that they are executed truly in parallel (e.g., on multiple processor cores).

Kreuzen Sie die Konzepte an, die zur Implementation von Multiprogrammierung verwendet werden können. **0.5 pt**

Tick the concepts that may be used to implement multiprogramming.

Nebenläufigkeit/ *Concurrency*	Parallelismus/ *Parallelism*
■	■

b) Nennen Sie ein Beispiel für einen Daemon auf einem typischen Computersystem. **1 pt**

Name an example for a daemon on a typical computer system.

Solution:

Web server, display server, window manager, audio server, DNS server, printer server, . . .

More specific examples are also accepted, e.g., systemd, nginx, dwm.exe, Xorg, svchost.exe. . .

*A garbage collector is **not** a daemon.*

c) Auf POSIX-Systemen können neue Prozesse mittels `fork()` und `exec()` gestartet werden. Derselbe Vorgang kann auch mittels `posix_spawn()` in einem einzelnen Systemaufruf durchgeführt werden. Nennen Sie zwei Vorteile gegenüber der Verwendung von `fork()` und `exec()`. **2 pt**

New processes can be started on POSIX systems by using `fork()` and `exec()`. The same result may also be achieved with a single system call by using `posix_spawn()` instead. Name two advantages of using `posix_spawn()` over the combination of `fork()` and `exec()`.

Solution:

- *Less overhead for system calls*
- *No need to copy the entire address space of the calling process first*

d) Wann wird für `malloc()` ein Systemaufruf ausgeführt, wann nicht? **1 pt**

In which cases is it required to perform a system call for a `malloc()` operation? When is it not required?

Solution:

A system call is only required if there is no sufficiently large free area in the process's heap space. The memory allocator then performs a system call to acquire additional heap space from the operating system kernel. Otherwise, it simply returns a pointer to a free area.

Wie heißt der Systemaufruf, der hierfür auf POSIX-Systemen verwendet wird? **0.5 pt**

What is the name of the system call that is used on POSIX systems for that purpose?

Solution:

`mmap()`, `brk()`, or `sbrk()` were accepted answers.

e) Typische Multithreading-Bibliotheken wie `pthreads` verwenden ausschließlich Kernel-Level-Threads, um Parallelismus zu ermöglichen. In einigen jüngeren Programmiersprachen (bspw. Go) ist es gängig, stattdessen auf hybrides Multithreading gemäß dem M-zu-N-Modell zu setzen. Erklären Sie, wie ein derartiger Ansatz die Performance von Anwendungsprogrammen verbessern kann. **2 pt**

Typical multithreading libraries such as `pthreads` exlusively use kernel-level threads to enable parallelism. In several recent programming languages (e.g., Go) it is common to employ hybrid multithreading according to the M-to-N model instead. Explain how such an approach may improve the performance of user applications.

Solution:

*Spawning new threads does not necessarily involve a system call anymore. Instead, a user-level scheduler is responsible for mapping user-level threads to kernel-level threads. This makes it cheap to spawn a large number of threads. Further, the program may decide for itself which threads it wants to execute at a certain time (the kernel is merely responsible for deciding **how many** of the program's threads may run concurrently), so that it can dynamically prioritize threads according to their current importance.*

Total: 9.0pt

Aufgabe T3: Speicher

Assignment T3: Memory

a) Nennen Sie neben mehrstufigen Seitentabellen zwei weitere Übersetzungsverfahren von virtuellen auf physische Adressen und jeweils einen Anwendungsfall wofür dieses Verfahren gut geeignet ist. **2 pt**

Other than multi-level page tables, name two other mechanisms that translate virtual to physical addresses, and for each name a use case the mechanism is well-catered for.

Solution:

Linear page table (0.5 P) *A linear page table is particularly suited for systems with small or mostly filled virtual address spaces* **(0.5 P)**.
Common error: *Fast translation alone does not suffice, because a TLB is still faster and commonly available.*

Inverted page table (0.5 P) *An inverted page table makes sense if the physical address space is considerably smaller than the virtual address space. This way, less memory is wasted for page tables* **(0.5 P)**.
Alternative: An inverted table is also useful as an extension to a regular forward page table **(0.5 P)**.
Common error: *If only few mappings are used, but physical memory is large, the lookup times make this mechanism infeasible, so the notion of small physical memory needs to be added.*

Hashed inverted page table (0.5 P) *Combines low memory usage of inverted page tables with fast lookup using hashes* **(0.5 P)**.

Segmentation (0.5 P) *If the hardware implementation should be simple.* **(0.5 P)**
Alternative: To avoid internal fragmentation. **(0.5 P)**

Base+Limit Registers (0.5 P)
If the hardware implementation should be simple. **(0.5 P)**
Alternatives: To avoid internal fragmentation. **(0.5 P)**
If no actual translation is necessary, only protection. **(0.5 P)**

b) Nennen Sie vier Informationen, die pro Page in in der Seitentabelle gespeichert werden. **2 pt**

State four pieces of information stored in the page table per page.

Solution:

Page Frame Number (0.5 P)

Valid Bit (0.5 P)

Write Bit (0.5 P)

Caching Bit (0.5 P)

Accessed Bit (0.5 P)

Modified Bit (0.5 P)

Kernel-Only Page (0.5 P)

(No) Execute Bit (0.5 P) *- Not discussed in the lecture but accepted as an answer*

Location on Disk (0.5 P) *- Accepted in conjunction with swapping*

"Permissions" (0.5 P) *- Accepted as long as no permissions were given explicitly*

Common errors:

Data *Referenced by the page table, but not stored there*

(Virtual) Page Number *Not explicitly stored in common page table implementations*

ASID/Address Space ID, Process ID, ... *These IDs are stored once per process and are either handled by the OS or in special CPU registers*

"Offset" *Not enough on its own, "physical offset" is accepted as a replacement for PFN*

Base+Limit *Not part of most page table implementations*

c) Nennen Sie zwei Gründe für einen Page Fault in einem Prozess, nach welchen der Prozess weiterlaufen kann. **1 pt**

Name two reasons for page faults which do not lead to termination of the process.

Solution:

page has been swapped out and is now accessed again **(0.5 P)**

page was demand-paged and has not been accessed yet **(0.5 P)**

page is mapped read-only for copy-on-write **(0.5 P)**

Common errors:

TLB Miss *Not an exception for Hardware-Managed TLBs, otherwise not "Page Fault" but "TLB-Miss Exception"*

OS can handle *Not accepted as this answer was already in the exercise*

Not mapped *Not accepted without further explanation as this might also lead to process termination*

Scenarios which DO *lead to process termination*

d) Beschreiben Sie das Phänomen Thrashing. **2 pt**

Describe the phenomenon thrashing.

Solution:

The idea behind swapping **(0.5 P)** *is to unload unused pages. If we get so high memory pressure* **(0.5 P)** *that we need to start swapping out pages in the current working set of the process* **(0.5 P)**, *other pages from the working set start getting replaced by the page we just swapped in* **(0.5 P)**. *This can happen both if the main memory is too small for all processes* **(0.5 P)**. *Therefore, the system is mostly busy swapping and makes little progress* **(0.5 P)**. *A remedy is to completely stop some processes until others complete to decrease memory pressure* **(0.5 P)**. *One effect is that CPU utilization lowers* **(0.5 P)** *as all processes are now IO-bound, which might encourage the OS to increase the degree of multiprocessing* **(0.5 P)**.
Only a subset of these answer options was expected.
Common errors:

Thrashing is not related to the TLB directly

While the amount of page faults does increase, this is not the reason for the described behavior. So without further explanation, this answer is not accepted.

e) Gegeben sei ein System, das virtuelle Adressen mittels einer dreistufigen Seitentabelle übersetzt. Die Seitengröße beträgt 4 KiB. Das System ist außerdem mit einem TLB ausgestattet, der 1024 Einträge fasst. Ein Cache ist nicht vorhanden. **2 pt**

Auf diesem System wird ein Prozess ausgeführt, der innerhalb eines zusammenhängenden 3 MiB-Speicherbereichs zufällige Anfragen macht. Als Sie den Speicherbereich auf 5 MiB vergrößern, messen Sie bei einigen Speicherzugriffen größere Latenzen. Woran liegt das? Bitte geben Sie die Rechnung an.

Consider a system which translates virtual addresses with a three-level page table. The page size is 4 KiB. Additionally, the system includes a TLB. There are no CPU caches.

The system runs a process which randomly accesses memory in a 3 MiB area of contiguous memory. After you increased the memory area to 5 MiB, some memory accesses show higher latency. What is the reason? Please note down your calculations.

Solution:

*In this system, the TLB reach is 4 MiB (1024 * 4 KiB)* **(0.5 P)**. *For the 3 MiB buffer, each translation can be cached in the TLB, so only one memory access is needed* **(0.5 P)**. *With the 5 MiB buffer, on average 20% of the accesses need to be translated with the pagetable* **(0.5 P)**, *which requires extra time and increases memory access latency* **(0.5 P)**.

Total: 9.0pt

Aufgabe T4: Koordination und Kommunikation von Prozessen

Assignment T4: Process Coordination and Communication

a) Nennen Sie die vier notwendigen Bedingungen für einen Deadlock. **2 pt**

Name the four necessary deadlock conditions.

Solution:

(0.5 P) *per condition: Mutual exclusion, hold and wait, no preemption, circular wait.*

b) Datenbanksysteme ermöglichen es Anwendungen, große Datenmengen strukturiert zu speichern und abzufragen. Prozesse kommunizieren mit einem Datenbanksystem mittels sogenannter *Transaktionen*. Innerhalb einer Transaktion können beliebig viele Datensätze abgerufen und modifiziert werden. Beim Abrufen wird ein Datensatz automatisch gesperrt, sodass parallel ablaufende Transaktionen warten müssen, bevor sie auf den Datensatz zugreifen können. Erst beim Beenden einer Transaktion (*Commit*) werden alle Locks wieder freigegeben. Im Folgenden wird eine vereinfachte Pseudo-Abfragesprache verwendet, um den Ablauf von Transaktionen anzugeben. Diese Befehle stehen zur Verfügung:

- `START TRANSACTION` initiiert eine neue Transaktion.
- `SELECT x` ruft den Datensatz `x` aus der Datenbank ab und sperrt ihn.
- `UPDATE x SET y` aktualisiert den Datensatz `x` in der Datenbank auf den Wert `y` (ohne ihn dabei zu entsperren).
- `COMMIT` beendet eine Transaktion und gibt alle gehaltenen Locks frei.

Database systems enable applications to store and retrieve large amounts of data in a structured manner. Processes can communicate with a database system using transactions*. Within a transaction, the client may read and modify an arbitrary amount of datasets. When a dataset is read, it is locked automatically. In turn, other transactions running at the same time will need to wait before they can access the dataset. All locks are freed as soon as a transaction is completed (*commit*). In this task, we will use a simplified query language to specify transaction sequences. The following commands are available:*

- *`START TRANSACTION` initiates a new transaction.*
- *`SELECT x` retrieves the dataset with the ID `x` from the database and locks it.*
- *`UPDATE x SET y` updates the dataset `x` in the database and sets its value to `y`. Note that `UPDATE` does not unlock datasets.*
- *`COMMIT` finishes a transaction and frees all locks held by it.*

Zwei Prozesse A und B führen die folgenden Transaktionen aus.

Two processes A and B execute the following transactions.

Process A

```
1  START TRANSACTION
2  SELECT 7
3  UPDATE 7 SET 3
4  COMMIT
5
6  START TRANSACTION
7  SELECT 4
8  SELECT 10
9  UPDATE 10 SET 1
10 SELECT 5
11 SELECT 7
12 UPDATE 7 SET 1337
13 COMMIT
```

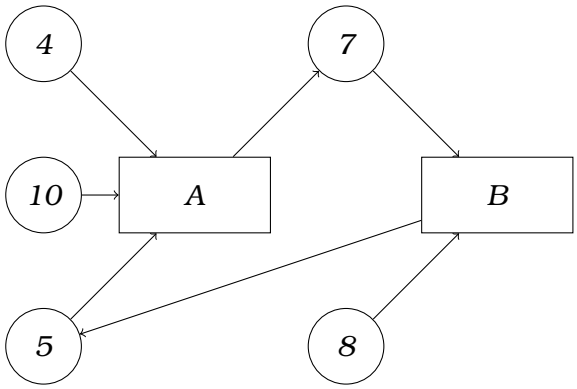
Process B

```
1  START TRANSACTION
2  SELECT 8
3  SELECT 7
4  SELECT 5
5  UPDATE 7 SET 1338
6  COMMIT
```

Im dargestellten Szenario kann bei paralleler Ausführung ein *Deadlock* zwischen den Prozessen A und B auftreten. Zeichnen Sie basierend auf dem Szenario einen *Resource Allocation Graph* (RAG), der den Zustand zum Zeitpunkt eines aufgetretenen *Deadlock* illustriert. Achten Sie darauf, dass alle gehaltenen Ressourcen eingezeichnet sind. **3.5 pt**

The scenario above may cause a deadlock *between the processes A and B when they are executed in parallel. Draw a* resource allocation graph *(RAG) based on the scenario that illustrates the locking state when a deadlock has occurred. Make sure to draw all held resources.*

Solution:



(0.5 P) *for each correct edge in the graph.* **(-0.5 P)** *for each wrong edge in the graph. Process A's first transaction is not involved in the deadlock situation, and therefore, there must not be an edge from dataset 7 to process A.*

c) Beschreiben Sie präzise, wie Sie **eine** der Transaktionen so verändern würden, dass kein Deadlock mehr auftreten kann. Entfernen Sie dabei keine Operationen und verändern Sie nicht die Semantik einzelner Operationen. Markieren Sie eindeutig, welche der Transaktionen Sie verändern. **1 pt**

*Accurately describe how you would modify **one** of the transactions to prevent deadlocks from occurring. Do not remove any operations and do not modify the semantics of individual operations. Clearly indicate the transaction that you modify.*

Prozess A, erste Transaktion/ *Process A, first transaction*	Prozess A, zweite Transaktion/ *Process A, second transaction*	Prozess B/ *Process B*
☐	■	☐

Solution:

```
1 START TRANSACTION
2 SELECT 4
3 SELECT 10
4 UPDATE 10 SET 1
5 SELECT 7
6 SELECT 5
7 UPDATE 7 SET 1337
8 COMMIT
```

(1 P) *for correctly reordered operations. No points when only a transaction was selected without actually rewriting the transaction. No points when operations are missing or semantics are changed. No points when no transaction or wrong transaction was selected.*

Alternative solution:

Prozess A, erste Transaktion/ *Process A, first transaction*	Prozess A, zweite Transaktion/ *Process A, second transaction*	Prozess B/ *Process B*
☐	☐	■

Solution:

```
1 START TRANSACTION
2 SELECT 8
3 SELECT 5
4 SELECT 7
5 UPDATE 7 SET 1338
6 COMMIT
```

d) Welche der vier notwendigen Bedingungen für einen Deadlock haben Sie durch das Verändern der Transaktion in der vorherigen Teilaufgabe gebrochen? **1 pt**

Which of the four necessary deadlock conditions did you break by modifying the transaction in the previous question?

Solution:

Circular wait.

e) Das Datenbanksystem führt automatisch einen RAG und nutzt diesen, um Deadlocks zu erkennen. Im Falle eines aufgetretenen Deadlocks wird eine der beteiligten Transaktionen abgebrochen und die vorgenommenen Änderungen rückgängig gemacht. Die andere(n) Transaktion(en) dürfen dann weiterlaufen. Welche der Transaktionen sollte im obigen Szenario sinnvollerweise abgebrochen werden? **0.5 pt**

The database system automatically maintains an RAG to detect deadlocks. In case a deadlock has occurred, it aborts one of the involved transactions and rolls all its changes back. The other transactions may then continue to execute. Which of the transactions should reasonably be aborted in the given scenario?

Prozess A, erste Transaktion/ *Process A, first transaction*	Prozess A, zweite Transaktion/ *Process A, second transaction*	Prozess B/ *Process B*
☐	☐	■

Warum? **1 pt**

Why?

Solution:

At the time when the deadlock occurs, process A holds three locks and has modified a single dataset. Process B, however, holds only two locks and has not changed any datasets. Therefore, terminating process A's active transaction would cause more progress to be lost than when process B's transaction is aborted.

Total: 9.0pt

Aufgabe T5: I/O, Hintergrundspeicher und Dateisysteme

Assignment T5: I/O, Secondary Storage, and File Systems

a) Anwendungen nutzen häufig Code ähnlich dem unten gegebenen, um ihre Dateien zu speichern.

Applications often use code similar to that in the snippet below for saving files.

```
1 /* open new temporary file */
2 fd = open("config.tmp", O_WRONLY | O_CREAT);
3 /* write new file contents */
4 write(fd, data, data_size);
5 /* close the temporary file */
6 close(fd);
7 /* replace the original file */
8 rename("config.tmp", "config.txt");
```

Das System stürzt nach dem Abschließen des Speichervorgangs ab. Nach einem Neustart stellen Sie fest, dass die Datei `config.txt` existiert, aber leer ist (d.h. Länge 0 hat) und somit die geschriebenen Daten verloren gingen. Wie kann es dazu kommen? **1.5 pt**

The system crashes after completing the save operation. After a restart, you notice that the file `config.txt` exists, but is empty (i.e., has length 0), thus losing the written data. How can this happen?

Solution:

The operating system caches writes in the file system cache (or buffer cache) **(0.5 P)**. *In case of a crash, the contents of the file system cache are lost* **(0.5 P)**. *If the file system persists the rename operation before flushing the file system cache, we end up with an empty file* **(0.5 P)**.

Note: *This issue became known as the "ext4 data loss" incident of 2009.*

Beheben Sie das oben beschriebene Problem durch Einfügen einer einzelnen Codezeile. **1 pt**

Fix the problem described above by inserting a single line of code.

Solution:

Code: `fsync(fd);` *or* `sync();` **(0.5 P)**
Insert after line: 4 **(0.5 P)**

Common error: `fflush()` *flushes user-space buffers managed by libc to the kernel. The code snippet above does not use these functions.*

Wie hilft *Journaling* dabei, das beschriebene Problem zu verhindern? **1 pt**

How does journaling *help prevent the described problem?*

Solution:

With journaling, all changes are committed to an ordered journal. Thus, it is not possible for a later change to be persisted without all earlier changes.

b) Das untenstehende Shellskript wird auf einem Linux-System ausgeführt. Welchen Dateiinhalt haben die einzelnen Dateien danach? Füllen Sie die Tabelle aus. Schreiben Sie „X“, falls auf eine Datei nicht zugegriffen werden kann. **2.5 pt**

The shell script below is run on a Linux system. What content does each file have afterwards? Fill in the table. Write "X" if a file cannot be accessed.

```
# -n: no newline
echo -n A > a
echo -n B > b
ln a c
ln c d
# -s: symlink
ln -s b e
# >>: append to file
echo -n 1 >> c
# mv source dest
mv b c
echo -n 2 >> d
echo -n 3 >> a
```

File	Contents
a	A123
b	**X**
c	B
d	A123
e	**X**

c) Welche drei Benutzerklassen kennt das traditionelle Modell der UNIX-Zugriffskontrolle? **1 pt**

Which three classes of users are present in the traditional UNIX access control model?

Solution:

user/owner, group, others **(1 P)** *(***(0.5 P)** *if one is missing/incorrect)*

Note: `chmod` *also allows setting permission for "all", but this is merely a shortcut to modify the three classes (user, group, others) at once.*

d) Was ist der wichtigste Vorteil einer File Allocation Table (FAT) gegenüber Chained Allocation? **1 pt**

What is the most important advantage of a file allocation table (FAT) compared to Chained Allocation?

Solution:

Compared to the linked list used in chained allocation, the FAT is more compact and can thus be cached in main memory more easily **(0.5 P)**. *This potentially reduces the number of disk accesses when a block in the middle of a file is requested* **(0.5 P)**.

Alternative: With a FAT, finding a block in the middle of a file does not require traversing all file blocks **(0.5 P)**. *Instead, the file system driver only reads the corresponding FAT entries. The FAT is more compact, so these read operations cause fewer disk seeks* **(0.5 P)**.

e) Welche Datenstrukturen durchläuft das Betriebssystem, um einen Dateideskriptor zu einer Inode aufzulösen? **1 pt**

What data structures does the operating system traverse to resolve a file descriptor to an inode?

Solution:

The file descriptor is an index into the local open file table **(0.5 P)** *which provides another index into the* global open file table **(0.5 P)** *which references the inode.*

Total: 9.0pt

Aufgabe P1: C-Grundlagen

Assignment P1: C Basics

a) In dem untenstehenden Code haben sich 5 Fehler eingeschlichen. Markieren Sie die fehlerhaften Zeilen mit einem X und korrigieren Sie den Code. Gehen Sie von einem 64-Bit-System aus. **5 pt**

There are 5 errors in the code below. Mark the incorrect lines with an X and correct the code. Assume a 64-bit system.

```
struct meta;
typedef struct meta meta;

size_t size(meta *buffer);
size_t capacity(meta *buffer);
```

```
struct meta {
    size_t element_size;
    void *begin;
    void *end;
    void *capacity;
};
```

Solution:

```
meta *allocate_meta(size_t element_size) {
    meta *pointer = malloc(sizeof(meta));
    if (!pointer) return 0;
    *pointer = (meta) {element_size, 0, 0, 0}; // missing *
    return pointer;
}
bool append(meta *buffer, void *data) {
    if(!data) return false;
    size_t buff_size = size(buffer); // & not needed
    size_t buff_bytes =
        buff_size * buffer->element_size; // . instead of ->
    size_t buff_cap = capacity(buffer);
    if (buff_size >= buff_cap) {
        size_t new_bytes =
            (buff_size * 2 + 1) * buffer->element_size;
        void *new_begin = malloc(new_bytes);
        if (!new_begin) return false;
        void *new_end = (char *)new_begin + buff_bytes;
        void *new_capacity =
            (char *)new_begin + new_bytes;
        memcpy(new_begin, buffer->begin, buff_bytes);
        free(buffer->begin); // fix memory leak
        buffer->begin = new_begin;
        buffer->end = new_end;
        buffer->capacity = new_capacity;
    }
    memcpy(buffer->end, data, buffer->element_size);
    buffer->end =
        (char *)buffer->end + buffer->element_size;
    return true;
}
```

```
void *get(meta *buffer, size_t where) {
    return (char *)buffer->begin +
        where * buffer->element_size; // missing return
}
size_t size(meta *buffer) {
    return ((char *)buffer->end - (char *)buffer->begin) /
        buffer->element_size;
}
size_t capacity(meta *buffer) {
    return ((char *)buffer->capacity - (char *)buffer->begin) /
        buffer->element_size;
}
```

Common errors:

`NULL` vs `0` *The literal `0` is also a valid constant to mark a `NULL`-pointer*

`!pointer` *Is a valid way to check whether a pointer is invalid*

`capacity(&buffer)` *Instead of fixing `size(buffer)`, some added an `&`. buffer is already a pointer in function `append`, so changing the type for `size` and `capacity` is not needed as they already accept a pointer.*

When to grow the dynamic array? *to grow the dynamic array, this needs to happen when capacity and size are equal (as then there is no space for new elements anymore). Sometimes, an `+buffer->element_size` was added, which is wrong for two reasons:*

(a) `size` and `capacity` deal in number of elements, not in number of bytes, so the error is of variable size.

(b) even if only one element was added in the comparison, we would be writing past the end of the allocated array

`meta` vs `struct meta` *In the header of the exercise, we stated `typedef struct meta meta;`, which aliased the `struct meta` name to be used for the struct to the (shorter) name `meta`. After that point, those two names are synonymous, so writing `meta` suffices.*

`void` vs `void*` *the `get()`-function returns a pointer to the element with index `where`. But, given that a generic dynamic array was implemented, the function can not specify the correct type and has to return `void*`. If its return type was `void`, it would return nothing, so `get()` would be without function.*

`(char*)` *The code is packed with casts of pointers to `char*`. The C language standard does not assign a size to type `void`, so pointer arithmetic on a pointer to `void` does not make sense. Therefore, we need to cast to a pointer to a type of defined size, and `sizeof(char)` $= 1$ by the standard. Therefore, with these casts all the pointer arithmetic makes sense.*

Welche Datenstruktur wird hier implementiert? **0.5 pt**

Which data structure is implemented here?

Solution:

This is the implementation of a simple dynamically sized array.
Also accepted: *vector, growing buffer, growing array, "like (java.util.)ArrayList"*
Common errors: *Queue/FiFo, Heap*[1]*, List, Array*

Schreiben Sie eine Funktion `free_meta()`, die ein `meta`-Objekt samt Daten freigibt. **1.5 pt**

Write a function `free_meta()`*, which frees a* `meta` *object including its data.*

Solution:

```
void free_meta(meta *buffer) {
    free(buffer->begin);
    free(buffer);
}
```

max **(1 P)** *for "freeing the data":* `free(buffer->begin)`

(-0.5 P) *for* `free(buffer->end);free(buffer->capacity);`

(-0.5 P) *for freeing the elements instead of the buffer (not allowed as these elements need not be pointers)*

(-0.5 P) *for freeing* `buffer->begin` *after* `buffer`

(0.5 P) *for* `free(buffer)`
Common errors: *In addition to stated above: writing* `meta` *instead of* `buffer`

[1] The dynamic array neither satisfies the heap property nor is it managed by the OS

b) Was ist der Wert der Ausdrücke nach Ausführung des Programmschnipsels? **2 pt**

What is the value of the expressions after execution of the given code snippet?

```
uint32_t a[] =
  { 0x00112233, 0x44556677,
    0x8899aabb, 0xccddeeff,
    0xff00ff00, 0x0123abcd };

uint16_t b = a[2] - a[1];
uint32_t c = b >> 2;
// <-- Evaluate expression here
```

Solution:

Expression	*Value*
`a[2] ^ a[2]`	*0x0*
`a[4] & a[5]`	*0x0100ab00*
`b || ~b`	*0x1*
`c`	*0x1111*

Solution:

Common errors:

`a[4] & a[5]` *: `0xff23ffcd`: `&` (bitwise and) was confused with `|` (bitwise or)*

`b || ~b` *: `0xffff`: `||` (logical or) was confused with `|` (bitwise or). Also, because of a feature called "integer promotion", even with `b | ~b` the answer would be `0xffffffff` on common 32 bit or 64 bit systems where the datatype `int` has 32 bits instead of 16 bits. (This answer has not been given once.)*

`c` *: `0x44`: shift by two was thought to be in terms of hex characters*

`c` *: `0x11111111`: The number of bits has been confused due to the type of `c`, but `b` is of type `uint16_t`, so the first two bytes (four hex digits) have never been assigned and thus been set to zero because `b` is unsigned.*

c) Nennen Sie die 3 Verwendungen des Zeichens `*` in C. **1.5 pt**

List the three usages of the symbol `*` in C.

Solution:

- *to declare a pointer* **(0.5 P)***: `int *ptr = NULL;`*
- *to dereference a pointer* **(0.5 P)***: `int value = *ptr;`*
- *to multiply two values* **(0.5 P)***: `int square = value * value;`*

Technically Correct is the Best Kind of Correct:

- *As a part of a multiline comment* **(0.5 P)***: `/* */`*
- *In a string literal* **(0.5 P)***: `const char* string = "Look, there is a * here!";`*

d) Eine ABI kann verschiedene Orte spezifizieren, um Parameter beim Funktionsaufruf zwischenzuspeichern. Nennen Sie zwei davon. **1 pt**

Where can the ABI specify to store arguments during a function call? Name two options.

Solution:

- *In registers* **(0.5 P)** *(modern x86-64 ABIs)*
- *On the stack* **(0.5 P)** *(cdecl, or when not enough registers are available)*

Common errors: *Heap: Even if large data is allocated to the heap, the function signature always accepts a pointer, which resides either on the stack or in a register.*

e) Warum ist dieser Code gefährlich, und wie könnte man das beheben? **1.5 pt**

Why is this code dangerous, and how to fix it?

```
int *ptr = some_pointer_returning_function();
int storage = (int)ptr;
// some time later
int *new_pointer = (int*)storage;
int value = *new_pointer;
```

Solution:

The pointer is casted to an `int`*, which is not necessarily big enough to hold it* **(0.5 P)***. When it is later casted back and dereferenced, the address might have been altered and an illegal or invalid value might be loaded* **(0.5 P)***. To fix it, use a type that is specified to be large enough* **(0.5 P)***, like* `intptr_t` *or* `uintptr_t`*. (A pointer type for* `storage` *is also accepted.)*

Common errors:

Casting by itself is not dangerous

`new_pointer` *should point to the same address as* `ptr`*, so storing the value pointed to by* `ptr` *in storage is not correct.*

It does not matter for `free`*ing the pointer whether the address is stored in a number or pointer datatype.*

`uint64_t` *or* `int64_t` *do not solve the issue fundamentally.*

f) Welche Zahlenwerte haben die folgenden `enum`-Werte? **1 pt**

Which integer values are represented by the following `enum` values?

```
enum JUST_A_NAME {
  FOO,
  BAR,
  BAZ = 15,
  ALICE,
  BOB,
};
```

Expression	Value
BAR	1
ALICE	16

Solution:

Common errors:

`BAR`, `ALICE`: 0. Enums count up

`BAR`: 14. Even though `BAZ` was assigned, the numbering for all elements before started at zero for `FOO` and increased by one for `BAR`.

g) Beschreiben Sie, wie sich die Länge einer Zeichenkette (`char*`) ohne die Verwendung von Hilfsfunktionen wie `strlen()` bestimmen lässt. **1 pt**

Explain how the length of a string (`char*`) can be determined without using any helper methods such as `strlen()`.

Solution:

Strings must be terminated with a `'\0'`-character **(0.5 P)***. The length can then be determined by counting* **(0.5 P)** *the number of characters until the `'\0'`-character (e.g., with a `while`-loop).*
Common errors: *Sometimes, if the string literal is allocated by the compiler, the compiler knows the arrays size, and the length of a string can be computed using `sizeof()`. However, this fails with all runtime-generated strings, so it is not accepted as an answer.*

Total: 15.0pt

Aufgabe P2: Datenbankserver

Assignment P2: Database Server

In dieser Aufgabe sollen Sie einen einfachen Datenbankserver implementieren. Der Server nimmt Verbindungen von Prozessen entgegen und erzeugt einen neuen Thread für jede Verbindung. In diesem Thread werden die auf der Verbindung eingehenden Befehle abgearbeitet. Die Kommunikation läuft in sogenannten *Transaktionen* ab. Innerhalb einer Transaktion können beliebig viele Datensätze abgerufen und modifiziert werden. Beim Abrufen wird ein Datensatz automatisch gesperrt, sodass parallel ablaufende Transaktionen warten müssen, bevor sie auf den Datensatz zugreifen können. Erst beim Beenden einer Transaktion (*Commit*) werden alle Locks wieder freigegeben. Während echte Datenbanksysteme komplexe Tabellenstrukturen abbilden können, nehmen wir in dieser Aufgabe an, dass genau 32 einfache Integer-Werte gespeichert werden. Die unterstützten Befehle sind dieselben wie in Aufgabe T4:

- `START TRANSACTION` initiiert eine neue Transaktion.
- `SELECT x` ruft den Datensatz `x` aus der Datenbank ab und sperrt ihn.
- `UPDATE x SET y` aktualisiert den Datensatz `x` in der Datenbank auf den Wert `y` (ohne ihn dabei zu entsperren).
- `COMMIT` beendet eine Transaktion und gibt alle gehaltenen Locks frei.

- Geben Sie vom Betriebssystem angeforderte Ressourcen (z. B. Speicher) explizit zurück.
- Binden Sie die in den Teilaufgaben notwendigen C-Header in dem gekennzeichneten Bereich ein.
- Sofern nicht anderweitig bestimmt, gehen Sie davon aus, dass (1) bei Systemaufrufen und Speicherallokationen keine Fehler auftreten, (2) Systemaufrufe nicht durch Signale unterbrochen werden und (3) Funktionsparameter valide sind.

In this assignment, you will implement a simple database server. The server accepts incoming connections from other processes and creates a new thread for every connection. This thread handles the commands sent via the associated connection. The server communicates with the clients via so-called transactions*. Within a transaction, the client may retrieve and modify an arbitrary amount of datasets. When a dataset is retrieved it is locked automatically, so that other transactions running in parallel will have to wait before accessing the dataset. Only when a transaction is finished (*committed*), all locks are freed again. While real-world database systems may store complex table structures, we assume for this task that exactly 32 simple integer values may be stored. The supported commands are the same as in assignment T4:*

- *`START TRANSACTION` initiates a new transaction.*
- *`SELECT x` retrieves the dataset `x` from the database and locks it.*
- *`UPDATE x SET y` updates the dataset `x` to the value `y` in the database (without unlocking it).*
- *`COMMIT` terminates a transaction and frees all held locks.*

- *Explicitly return allocated operating system resources (e.g., memory).*

- *Include necessary C headers in the marked area.*
- *Unless stated otherwise, assume that (1) system calls and memory allocations do not fail, (2) system calls are not interrupted by signals, and (3) function arguments are valid.*

```
#include <stdbool.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NUM_VALUES 32

struct database {
    int values[NUM_VALUES];
    pthread_mutex_t locks[NUM_VALUES];
}
struct database db;

struct conn {
    pthread_t thread;
    int fd;
    bool in_trans;
    bool locked[NUM_VALUES];
}
```

a) Wenn eine neue Verbindung eingeht, sollen deren Befehle auf einem eigenständigen Thread abgearbeitet werden. Eine Verbindung wird durch einen Dateideskriptor (*file descriptor*, `fd`) repräsentiert. Allozieren und initialisieren Sie hierzu zunächst eine Struktur für die Verbindungsdaten (`struct conn`) und übergeben Sie diese dann als Argument an den neu gestarteten Thread. Der Thread soll die Funktion `conn_thread()` ausführen. Geben Sie die Verbindungsdaten-Struktur zurück. **2.5 pt**

When a new connection is established, a new thread shall be spawned to handle its commands. A connection is represented by a file descriptor (`fd`). Start by allocating and initializing a structure for the data associated with the connection (`struct conn`) and then pass it as an argument to the newly created thread. The thread shall execute the function `conn_thread()`. Return the connection data structure.

```
struct conn *on_new_connection(int fd) {
    struct conn *data = malloc(sizeof(struct conn));
    memset(data, 0, sizeof(struct conn));
    data->fd = fd;

    pthread_create(&data->thread, NULL, conn_thread, data);

    return data;
}
```

Für jeden Befehl gibt es eine Funktion, welche für dessen Abarbeitung im Rahmen einer Verbindung zuständig ist. Implementieren Sie diese Funktionen analog der im Einleitungstext gegebenen Befehlsbeschreibung. Beachten Sie dabei folgendes:

- Nutzen Sie die vordefinierten Strukturen. Sie müssen keine eigenen Strukturen definieren.
- Die einzige Instanz der Datenbank ist in der Variable `db` vorgegeben.
- Gehen Sie davon aus, dass alle Mutexe und die Verbindungsstruktur korrekt initialisiert sind.
- Geben Sie in allen Funktionen `true` zurück, wenn der Befehl erfolgreich ausgeführt wurde und `false`, wenn ein Fehler aufgetreten ist.
- Prüfen Sie die im Befehl gegebenen Argumente (bspw. `sel_pos`) auf Plausibilität.
- `UPDATE`-Befehle dürfen nur auf zuvor gesperrten Datensätzen ausgeführt werden.
- Sämtliche Befehle außer `START TRANSACTION` dürfen nur innerhalb von Transaktionen ausgeführt werden. `START TRANSACTION` innerhalb einer bereits laufenden Transaktion ist nicht erlaubt.
- `SELECT` darf innerhalb einer Transaktion mehrfach für den gleichen Datensatz ausgeführt werden. In diesem Fall darf der Datensatz nicht erneut gesperrt werden.
- `SELECT` soll den gelesenen Wert über den Dateideskriptor der Verbindung ausgeben. Nach dem Zahlenwert soll zusätzlich ein Zeilenumbruch (`\n`) ausgegeben werden. **Hinweis**: Ziehen Sie die Nutzung einer der Varianten von `printf()` hierfür in Betracht.

A function is defined for each command that is called for its execution during a connection. Implement the corresponding functions based on the definitions given in this task's introductory text. Observe the following restrictions:

- *Make use of the predefined C structures. You do not need to define your own structures.*
- *The only instance of the database is already defined in the variable `db`.*
- *Assume that all mutexes and the connection data structure are correctly initialized.*
- *Return `true` when the command was successful and `false` when an error has occurred.*
- *Check all command arguments (e.g., `sel_pos`) for plausibility.*
- *`UPDATE` commands may only be executed on datasets that were previously locked.*
- *All commands except `START TRANSACTION` may only be executed within a transaction. `START TRANSACTION` within an already running transaction is not allowed.*
- *`SELECT` may be executed on the same dataset multiple times within a single transaction. Do not lock the dataset again in that case.*
- *`SELECT` shall output the retrieved value via the connection's file descriptor. Additionally print a line break (`\n`) after the value. **Hint**: Consider using one of the variants of `printf()`.*

b) Implementieren Sie `do_start_transaction()` für den Befehl `START TRANSACTION`. **1.5 pt**

Implement `do_start_transaction()` for the `START TRANSACTION` command.

```
bool do_start_transaction(struct conn *data) {
    if(data->in_trans)
        return false;

    data->in_trans = true;

    return true;
}
```

c) Implementieren Sie `do_commit()` für den Befehl `COMMIT`. **3.5 pt**

Implement `do_commit()` for the `COMMIT` command.

```
bool do_commit(struct conn *data) {
    if(!data->in_trans)
        return false;

    for(int i = 0; i < NUM_VALUES; i++) {
        if(data->locked[i]) {
            pthread_mutex_unlock(&db.locks[i]);
            data->locked[i] = false;
        }
    }

    data->in_trans = false;

    return true;
}
```

d) Implementieren Sie `do_update()` für den Befehl `UPDATE`. **2.5 pt**

Implement `do_update()` for the `UPDATE` command.

```
bool do_update(struct conn *data, unsigned int upd_pos, int upd_value) {
    if(!data->in_trans)
        return false;

    if(upd_pos >= NUM_VALUES)
        return false;

    if(!data->locked[upd_pos])
        return false;

    db.values[upd_pos] = upd_value;

    return true;
}
```

e) Implementieren Sie `do_select()` für den Befehl `SELECT`. **5 pt**

Implement `do_select()` for the `SELECT` command.

```
bool do_select(struct conn *data, unsigned int sel_pos) {
    if(!data->in_trans)
        return false;

    if(sel_pos >= NUM_VALUES)
        return false;

    if(!data->locked[sel_pos])
        pthread_mutex_lock(&db.locks[sel_pos]);

    data->locked[sel_pos] = true;

    if(dprintf(data->fd, "%i\n", db.values[sel_pos]) < 0)
        return false;

    return true;
}
```

Total: 15.0pt

Aufgabe P3: Log-strukturierter Speicher

Assignment P3: Log-Structured Storage

In einem Log-strukturierten Speichersystem werden neue Einträge immer an das Ende des Logs geschrieben. In dieser Aufgabe setzen Sie ein einfaches Log-strukturiertes Speichersystem um.

Jeder Eintrag besteht aus einem Header (`struct record`) und einer variablen Menge an Daten. Die Länge der Daten ist immer ein Vielfaches der Blockgröße `BS`. Jeder Eintrag besitzt eine eindeutige ID, die für gelöschte Einträge auf `ID_INV` gesetzt wird. An der aktuellen Einfügeposition `tail_off` steht immer ein Tail-Block, ein ungültiger Eintrag mit der Länge des verbleibenden freien Speichers.

- Nutzen Sie die Funktionen `write_block()` und `read_block()`, die jeweils einen einzelnen Block vom Hintergrundspeicher schreiben oder lesen.
- Die Funktion `copy_blocks()` kopiert Blöcke sequentiell innerhalb des Hintergrundspeichers.

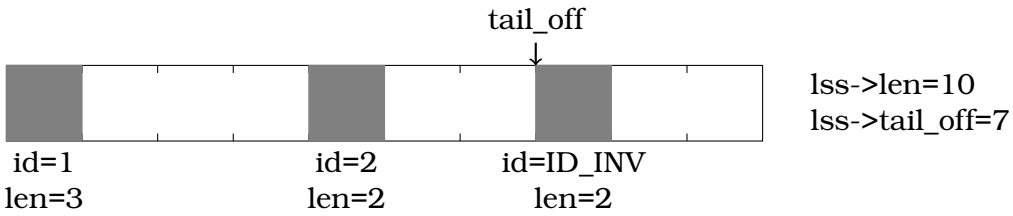
In a log-structured storage system, new entries are always written to the end of the log. In this assignment, you will realize a simple log-structured storage system.

Every entry has a header (`struct record`) and a variable amount of data. The data length is always a multiple of the block size `BS`. Every entry has a unique id which is set to `ID_INV` for deleted entries. There is always a tail block at the current insertion position `tail_off`. A tail block is an invalid entry with length equal to the remaining free space.

- *Use the functions `read_block()` and `write_block()` which read or write a single block from the background storage.*
- *The function `copy_blocks()` copies blocks sequentially within the background storage.*

```
#define BS 512                    /* block size */
#define ID_INV ((uint64_t) -1)    /* id for invalid records */

struct lss {
  uint64_t len;       /* total length of the storage area in blocks */
  uint64_t tail_off;  /* offset of the tail in blocks */
  /* fields for block storage access omitted */
};

struct record {
  uint64_t id;   /* user-defined id, or ID_INV */
  uint64_t len;  /* length of data in blocks after the record header */
  uint32_t type; /* user-defined type */

  char _padding[BS - (2*8 + 4)]; /* or BS - 20 or 492 */

};

void write_block(struct lss *lss, uint64_t off, void *mem);
void read_block(struct lss *lss, uint64_t off, void *mem);
void copy_blocks(struct lss *lss, uint64_t target, uint64_t source,
                 uint64_t len);
```

Beispiel mit zwei Einträgen: / *Example with two entries:*



a) Vervollständigen Sie die Deklaration von `_padding`, sodass `struct record` eine Gesamtgröße von `BS` Byte hat. **1 pt**

Complete the declaration of `_padding` so that `struct record` has a total size of `BS` bytes.

b) Vervollständigen Sie die Funktion `ensure_space()`, die sicherstellt, dass mindestens `len` freie Blöcke am Ende des Logs frei sind. **2.5 pt**

- Rufen Sie die Funktion `compact_log()` auf, falls nicht ausreichend viele Blöcke frei sind.
- Falls anschließend immer noch nicht genügend Blöcke frei sind, gibt die Funktion `-ENOSPC` zurück, ansonsten 0.

Complete the function `ensure_space()` which ensures that at least `len` free blocks are available at the tail of the log.

- *Call the function `compact_log()` if there are not enough free blocks.*
- *If the amount of free blocks is still not sufficient, the function returns `-ENOSPC`, otherwise 0.*

Solution:

```
int ensure_space(struct lss *lss, uint64_t len) {
  if (lss->len - lss->tail_off < len) {
    compact_log(lss);
    if (lss->len - lss->tail_off < len)
      return -ENOSPC;
  }
  return 0;
}
```

Correct check **(1 P)**, *Call to `compact_log()`* **(0.5 P)**, *Repeated check* **(0.5 P)**, *Returns* **(0.5 P)**

Note: *In the solution above, we assume that `len` includes the header. Excluding the header (with an appropriate call in `append_record()`) is also correct.*

c) Vervollständigen Sie die Funktion `write_tail()`, die wie oben beschrieben einen Tail-Block an das Ende des Logs schreibt. **1.5 pt**

- Sie müssen den alten Tail-Block nicht anfassen.
- Aktualisieren Sie schließlich `tail_off`.

Complete the function `write_tail()`, which writes a tail block at the end of the log, as specified above.

- *You do not need to touch the old tail block.*
- *Finally, update `tail_off`.*

Solution:

```
void write_block(struct lss *lss, uint64_t off, void *mem);

void write_tail(struct lss *lss, uint64_t off) {
  struct record tail;
  tail.id = ID_INV;
  tail.len = lss->len - off - 1;
  write_block(lss, off, &tail);
  lss->tail_off = off;
}
```

Correct values for `id` *and* `len` **(0.5 P)**, *Writing the block* **(0.5 P)**, *Updating* `tail_off` **(0.5 P)**

Note: *Allocating the tail block with* `malloc()` *is accepted if you also free that memory.*

d) Vervollständigen Sie die Funktion `append_record()`, die einen Eintrag an den Log anhängt. **4.5 pt**

- Stellen Sie zunächst durch einen Aufruf von `ensure_space()` sicher, dass genügend Speicherplatz für den `record`, die Daten und den Tail-Block vorhanden ist. Geben Sie auftretende Fehler zurück, ansonsten 0.
- Schreiben Sie dann die Daten sowie einen neuen Tail-Block.
- Schreiben Sie schließlich den `record`.

Complete the function `append_record()`*, which appends an entry to the log.*

- *First, by calling* `ensure_space()`*, make sure that there is enough space left for the* `record`*, the data, and the tail block. Return any errors that may occur and otherwise 0.*
- *Then, write the data and the new tail block.*
- *Finally, write the* `record`*.*

Solution:

```
int ensure_space(struct lss *lss, uint64_t len);
void write_tail(struct lss *lss, uint64_t off);
void write_block(struct lss *lss, uint64_t off, void *mem);

int append_record(struct lss *lss, struct record *rec, void *data) {
  uint64_t off = lss->tail_off;

  /* Make sure we have enough space left for two records and the data. */
  int err;
  if ((err = ensure_space(lss, rec->len + 2)) < 0)
    return err;

  /* ensure_space() may modify lss->tail_off */
  off = lss->tail_off;

  /* Write data and tail block first */
  for (uint64_t i = 0; i < rec->len; i++)
    write_block(lss, off + 1 + i, (char *)data + i*BS);
  write_tail(lss, off + 1 + rec->len);

  /* Write record */
  write_block(lss, off, rec);
```

```
  return 0;
}
```

Call to `ensure_space` **(1.5 P)**, *writing the data* **(2 P)**, *updating the tail* **(0.5 P)**, *writing the record* **(0.5 P)**.

Note: *We added the local variable* `off` *to save you some writing effort.* **(0.5 P)** *extra if you notice that* `ensure_space()` *may modify* `lss->tail_off` *and update* `off` *accordingly.*

Note: *The order of the calls is important for crash consistency. By writing the* `record` *last as described in the assignment, it is impossible to read incomplete data.*

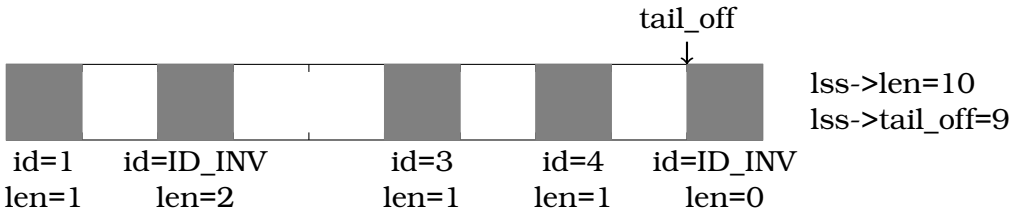
e) Vervollständigen Sie die Funktion `compact_log()`, die ungültige Einträge aus dem Log entfernt und folgende gültige Blöcke verschiebt, um Lücken zu füllen. **4 pt**

- Durchlaufen Sie den Log von dem `record` an Offset 0 aus.
- Aktualisieren Sie zum Schluss den Tail-Block.

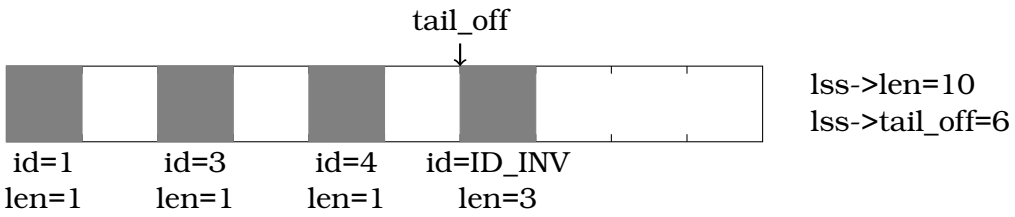
Complete the function `compact_log()`, *which removes invalid entries from the log and which moves the following valid blocks to fill any gaps.*

- *Walk the log starting from the* `record` *at offset 0.*
- *Finally, update the tail block.*

Before:



After:



Solution:

```
void write_tail(struct lss *lss, uint64_t off);
void read_block(struct lss *lss, uint64_t off, void *mem);
void copy_blocks(struct lss *lss, uint64_t target, uint64_t source,
                 uint64_t len);

void compact_log(struct lss *lss) {
  struct record rec;
  uint64_t off = 0, move = 0;

  while (off < lss->len) {
    read_block(lss, off, &rec);
    if (rec.id == ID_INV)
      move += rec.len + 1;
    else if (move > 0)
```

```
      copy_blocks(lss, off - move, off, rec.len + 1);
    off += rec.len + 1;
  }

  /* Finally, update the tail */
  write_tail(lss, off - move);
}
```

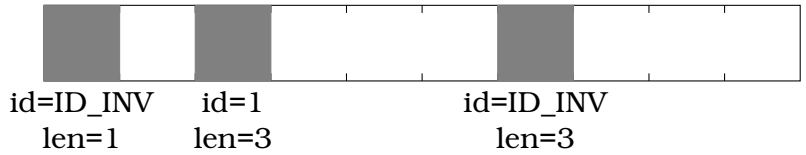
Correct loop **(1.5 P)**, *Reading* **(0.5 P)**, *Correct target offset* **(0.5 P)**, *Copying* `record` *and data* **(1 P)**, *Updating tail* **(0.5 P)**

f) Warum können Daten verloren gehen, wenn das System während der Ausführung von `compact_log()` abstürzt? Skizzieren Sie eine problematische Situation. **1.5 pt**

Why may there be data loss if the system crashes during execution of `compact_log()`*? Draw a problematic situation.*

Solution:

If the free space is smaller than the entry we are trying to move, we overwrite the original entry while moving it to the new position. Thus, we would lose data if the system crashes during the move operation.



Note: Only answers where data is irrevocably lost by overwriting are accepted. **Total: 15.0pt**