Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

Prof. Dr. Frank Bellosa
Peter Maucher, M.Sc.
Daniel Habicht, M.Sc.
Dipl.-Inform. Thorsten Gröninger
Lukas Werling, M.Sc.

Musterlösung Nachklausur
05.09.2025

Alle Punkteangaben ohne Gewähr!

- Bitte tragen Sie zuerst auf dem Deckblatt Ihren Nachnamen, Ihren Vornamen und Ihre Matrikelnummer ein. Tragen Sie dann auf allen 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 27 Blättern: Einem Deckblatt, 20 Aufgabenblättern mit insgesamt 3 Aufgaben sowie 6 Seiten Man-Pages.

  The examination consists of 27 pages: One cover sheet, 20 sheets containing 3 assignments, and 6 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 also 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 according to the lecture.

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 (20 P)

Assignment 1: Virtualization (20 P)

In Ihrem Betriebssystem entwickeln Sie an der Unterstützung von Prozessen und Threads. Statt dazu zwei Systemaufrufe zu entwickeln, lassen Sie sich von Linux und dessen `clone`-Systemaufruf inspirieren, der für beide Zwecke genutzt werden kann. In dieser Aufgabe entwerfen und benutzen Sie diesen Systemaufruf mit folgender Signatur:

In your operating system you are developing the support for processes and threads. Instead of developing two system calls you take inspiration from Linux and its `clone` system call, which can be used for both purposes. In this assignment you design and use this system call with the following signature:

```
enum clone_flags {
  DUP_DATA = 1,  // duplicates all remaining modifyable properties into the child
  DUP_STACK = 2, // creates a new stack for the child
  DUP_REGS = 4,  // creates a new set of register descriptors for the child
};
int clone(enum clone_flags flags);
```

a) Begründen Sie für die folgenden 3 Kategorien, wie diese beim Erstellen eines neuen Threads kopiert und geändert werden müssen oder beibehalten werden können. **3 P**

Explain for the following 3 categories, how they must be copied and modified or kept identical on thread creation.

Solution:

Registers *Need to be* modified, *e.g., to point to the new stack pointer or to contain the return value that needs to be different between parent and child thread*

Stack *Needs to be* copied *to let each thread use its own working area. Also needs to be relocated in the address space*

Heap *Shared between all threads, no work needed and is kept* identical

b) Nennen Sie zwei weitere Prozesseigenschaften, die beim `fork()`-Aufruf modifiziert oder kopiert werden müssen, und beschreiben Sie kurz warum die Operation notwendig ist. **2 P**

Name two additional process properties which must be modified or copied using `fork()` and briefly describe why the operation is necessary.

Solution:

- *Local Open File Table: This references the same entries in the Global Open File Table, however to facilitate modifications down the road the road, the table itself needs to be copied 1:1*
- *Page Table: We need a new page table to reference the new address space, which also requires modifications in the intermediate steps*
- *Process ID: Unique per process*
- *Parent Process: Obviously the process that just called `fork()`*
- *Remove threads: Only the thread calling `fork()` is still running in the child, and only its stack (and TCB) remains*
- *PCB/TCB: new process needs new Process/Thread Control Block*

c) Begründen Sie, warum `clone()` im Kernel implementiert sein muss. **1 P**

Give reasons why `clone()` must be implemented in the kernel.

Solution:
- *Page table alterations required*
- *interfaces with kernel level scheduler*
- *creates new TCB/PCB*
- *changes PID, a kernel level property*

Common mistakes:
- *Congestion or resource exhaustion are not accepted as these are also possible with a user space program spamming a system call to the OS*
- *The process does not involve privileged instructions, only access to kernel-only memory (but accessed using normal instructions)*

d) `pthread` Creation

In dieser Teilaufgabe nutzen Sie den oben beschriebenen `clone()`-Systemaufruf, um darauf basierend das Starten eines Threads und das Einsammeln des Ergebnisses zu implementieren. Sie implementieren dazu die zwei Funktionen `pthread_create` und `pthread_join`. Beachten Sie dabei:

- Sie dürfen annehmen, dass es zwei weitere Systemaufrufe gibt:
  - `void exit_thread(void);` das den aufrufenden Thread, aber nicht den ganzen Prozess beendet und
  - `void wait_tid(int thread_id);` das blockiert, bis der angegeben Thread beendet ist. Sie müssen dazu im Parameter `thread_id` die Thread-ID übergeben.
- Sie müssen den *Output-Parameter* `thread` in `pthread_create` mit Werten füllen, um diese Werte in `pthread_join` nutzen zu können. Die Definition von `struct pthread_t` ist durch die Aufgabenstellung vorgegeben.
- Sie können für diese Aufgabe davon ausgehen, dass keine Fehler auftreten.
- Denken Sie daran, dass Sie alle allokierten Ressourcen wieder freigeben.
- *Hinweis:* Beim Erstellen eines Threads können Änderungen im Adressraum passieren. Stellen Sie sicher, dass Pointer valide bleiben.

In this subtask you use the previously described `clone()` system call to implement the starting of a thread and collection of its result. For this, you implement two functions `pthread_create` and `pthread_join`. Keep in mind:

- *You can assume that there a two additional system calls:*
  - *`void exit_thread(void);` which terminates the calling thread, but not the whole process and*
  - *`void wait_tid(int thread_id);` which blocks until the named thread terminates. You have to pass the thread-id in parameter `thread_id`.*
- *You have to fill the* output parameter *`thread` in `pthread_create` with values to use said values in `pthread_join`. The definition of `struct pthread_t` is given by the exercise.*
- *In this subtask you can assume that no errors occur.*
- *Keep in mind that you must free all allocated resources.*
- Hint: *During thread creation, changes to the address space can happen. Ensure that pointers remain valid.*

```
typedef struct pthread_t {
    void **return_value; // hint: double pointer
    int thread_id;
} pthread_t;
```

i) Der `clone()`-Systemaufruf soll einen `int`-Wert zurückgeben. Diskutieren Sie, was in diesen Wert kodiert werden muss. **1 P**

Hinweis: Denken Sie zum Beispiel daran, wie in POSIX neue Prozesse erstellt werden.

The `clone()` system call should return an `int` value. Discuss what must be encoded in this value.

Hint: *For example, think about how new processes are created in POSIX.*

Solution:
The system call needs to return the new process id/thread id in the parent (given that we use the same system call, PID and TID are returned as the same[1]). However, in the child we just return 0 to be compatible with `fork()`.

In the most general sense, we need to differentiate between parent ***(0.5 P)*** *and child* ***(0.5 P)****, and also need to communicate errors.*

ii) Implementieren Sie den `pthread_create`-Funktionsaufruf mithilfe des `clone()`-Systemaufrufs. **3.5 P**

Sie können hier den Parameter `attr` ignorieren.

Implement the `pthread_create` *function call by using the* `clone()` *system call.*

Here, you can ignore the parameter `attr`.

Solution:

```
int pthread_create(pthread_t *thread, pthread_attr_t *attr,
    void *(*start_routine)(void *), void *argument) {
  thread->return_value = malloc(sizeof(void*)); // .5P (putting onto stack does
  not work as struct pthread_t needs to be copyable)
  int result = clone(DUP_STACK | DUP_REGS); // .5P
  if (result != 0) {  // .5P
    thread->thread_id = result; // .5P
    return 0;
  } else {
    void* function_result = start_routine(arguments); // .5P
    *(thread->return_value) = function_result; // .5P
    exit_thread(); // .5P
  }
}
```

iii) Implementieren Sie den `pthread_join`-Funktionsaufruf, sodass Sie das Ergebnis des Funktionsaufrufs im Kindsthread erhalten. **2.5 P**

Implement the `pthread_join` *function call so that you get the result of the function call in the child thread.*

Solution:

```
int pthread_join(pthread_t thread, void **retval) {
  wait_tid(thread.thread_id); // 1P
  if(retval) // .5P
    *retval = *thread.return_value; // .5P
  free(thread.return_value); // .5P
  return 0;
}
```

[1]This is true even in a modern Linux system, even though the "PID" is the same for all threads in a process. "Real" Linux solves this with a concept called a *thread group* which bundles processes (which is what threads really are in linux). However, these technicalities are far beyond the scope of the lecture, and even more so the exam. If this has peaked your interest though, see the linux `clone(2)` or `exit_group(2)`-manpages.

e) Ihr `clone()`-Systemaufruf muss bei manchen Verwendungen große Mengen Speicher kopieren. Nennen Sie die Optimierung, die dieses Kopieren in manchen Fällen vermeiden kann, und beschreiben Sie das Verfahren. **2 P**

Your `clone()` system call sometimes has to copy large amounts of memory. Name the optimization which in some cases can avoid copying, and describe the technique.

Solution:
Many operating systems implement Copy on Write (CoW), a technique where all pages shared between parent and child are marked as read-only (in which case sharing is safe). If a process writes to such a page, a due to the access violation a page fault happens which then traps into the OS kernel. The kernel can then copy the pages as required and remap the pages as read/write in both parent and child.

f) Führen Sie die folgenden Scheduling-Algorithmen aus und berechnen Sie die durchschnittliche Turnaround-Time des Systems. Zu jedem Zeitschritt kann dabei eine Scheduling-Entscheidung getroffen werden. Neue Prozesse werden vorne in Warteschlangen eingefügt. Sie haben die folgenden Prozesse gegeben: **5 P**

Execute the following scheduling algorithms and calculate the average turnaround time of the system. At each time step a new scheduling decision can be made. New processes are added at the front of waiting queues. The following processes are given:

ID	Arrival Time	Runtime
P1	$t = 0.5$	3
P2	$t = 1.5$	4
P3	$t = 3.5$	2
P4	$t = 4.5$	1

Solution:

$t =$	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
FCFS	*P1*	*P1*	*P1*	*P2*	*P2*	*P2*	*P2*	*P3*	*P3*	*P4*
RR	*P1*	*P2*	*P1*	*P3*	*P4*	*P2*	*P1*	*P3*	*P2*	*P2*

Round Robin Queue

$t =$	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
	P1	*P2*	*P1*	*P3*	*P4*	*P2*	*P1*	*P3*	*P2*	*P2*
		P1	*P2*	*P2*	*P2*	*P1*	*P3*	*P2*		
				P1	*P1*	*P3*	*P2*			
					P3					

Turnaround Time: *A processes turnaround time is the time between arrival and completion of a process, and the systems turnaround time is the average process turnaround time.*

FCFS $\mathrm{TT}_{\mathrm{P1}} = 4 - 0.5 = 3.5$; $\mathrm{TT}_{\mathrm{P2}} = 8 - 1.5 = 6.5$;

$\mathrm{TT}_{\mathrm{P3}} = 10 - 3.5 = 6.5$; $\mathrm{TT}_{\mathrm{P4}} = 11 - 4.5 = 6.5$;

$\mathrm{TT}_{\mathrm{sys}} = \frac{\sum_{i=1}^{4} \mathrm{Pi}}{4} = \frac{3.5+6.5+6.5+6.5}{4} = \frac{23}{4} = 5.75$

RR $\mathrm{TT}_{\mathrm{P1}} = 8 - 0.5 = 7.5$; $\mathrm{TT}_{\mathrm{P2}} = 11 - 1.5 = 9.5$;

$\mathrm{TT}_{\mathrm{P3}} = 9 - 3.5 = 5.5$; $\mathrm{TT}_{\mathrm{P4}} = 6 - 4.5 = 1.5$;

$\mathrm{TT}_{\mathrm{sys}} = \frac{\sum_{i=1}^{4} \mathrm{Pi}}{4} = \frac{7.5+9.5+5.5+1.5}{4} = \frac{24}{4} = 6$

Total: 20 P

Aufgabe 2: Nebenläufigkeit (20 P)

Assignment 2: Concurrency (20 P)

a) Sie haben bereits einige Funktionen für das Threading implementiert. Im Folgenden arbeiten Sie an einer Kompatibilitätsbibliothek, die erlaubt, alten Code auf unterschiedlichen Plattformen auszuführen.

Diese Platformen haben:
- Mehr als einen Prozessorkern
- Unterschiedliche Prozessorfunktionalität
- Unterschiedliche POSIX-kompatibel Betriebssysteme

You already implemented some functions for threading. In the following you are working on a compatibility library which runs old code on various platforms.

These platforms have:
- *More than one processor core*
- *Different processor functionality*
- *Different POSIX-compatible operating systems*

i) Man kann Nebenläufigkeit mit verschiedenen Ansätzen erreichen. Beschreiben Sie kurz, welche Funktionalität ein Betriebssystem nicht bzw. nur eingeschränkt bereitstellt, wenn man auf User Level Threads (ULT) zurückgreifen muss. Welche Threading-Modelle werden von so einem Betriebssystem nicht unterstützt? Geben Sie zusätzlich sowohl einen Vorteil, als auch einen Nachteil oder Problem für ULT Implementierungen an. **2.5 P**

You can achieve concurrency with various means. Briefly describe which functionality of the operating system is missing or limited, so User Level Threads (ULT) must be used. Which threading models are not supported by such operating system? Additionally, give both one advantage and also one disadvantage or problem for ULT implementations.

Solution:

ULT:

ULT must be used if the operating system does not support kernel level threads and can only schedule processes. ***(0.5 P)***

Missing Threading Models:
- *1-to-1 model* ***(0.5 P)***
- *N-to-M model* ***(0.5 P)***

Pro: (0.5 P)
- *Faster thread management operations (up to 100 times)*
- *Flexible scheduling policy*
- *Few system resources*

Con: (0.5 P)
- *No parallel execution*
- *Whole process blocks if only one user thread blocks*
- *Need to re-implement parts of the OS (e.g., scheduler)*
- *Unknown to the kernel (thus blocking)*

ii) ULTs können sowohl kooperativ als auch nicht-kooperativ implementiert werden. Beide Implementierungen können sich auf Locks auswirken, die einen kritischen Abschnitt schützen. **2 P**

Wie erfolgt der Wechsel zu einem anderen Thread bei einer kooperativen Implementierung? Welche Funktion muss von der Laufzeitumgebung bereitgestellt werden und was muss diese Funktion machen?

Wie könnten Locks im kooperativen Fall realisiert werden? Sind Locks notwendig? Begründen Sie ihre Antwort!

ULTs can be implemented either as cooperative or non-cooperative. Both implementations can affect how locks guarding a critical section work.

How does the switch to another thread work in a cooperative implementation? What function must be provided by the runtime environment and what must this function do?

How could locks be realized in the cooperative case? Are locks necessary? Justify your answer!

Solution:
Cooperative:
The runtime needs to provide yield(), ***(0.5 P)*** *which releases the CPU, swaps context (e.g., saving and restoring all registers), and gives control to the (user) scheduler.* ***(0.5 P)***

Realization of Locks:
Locks are not required, as only one thread can enter a critical section and decides when to give up the CPU in the current process. ***(1 P)***

iii) Nennen und beschreiben Sie kurz einen Mechanismus, welchen das Betriebssystem oder der Prozessor bereitstellen muss, um nicht-kooperative ULTs zu ermöglichen. **2 P**

Wie könnten Locks (für kritische Abschnitte) im nicht-kooperativen Fall realisiert werden? Sind Locks notwendig? Begründen Sie ihre Antwort!

Name and briefly describe a mechanism the operating system or processor has to provide to enable non-cooperative ULTs.

How could locks (for critical sections) be realized in the non-cooperative case? Are locks necessary? Justify your answer!

Solution:
Non-cooperative:
For non-cooperative ULTs, the process must be periodically notified by the operating system similar to a timer interrupt, allowing the user scheduler to preempt the currently-running thread. ***(1 P)*** *(e.g., with a SIGALRM signal)*

Realization of Locks: *(max* ***(1 P)****)*
Locks are necessary for mutual exclusion. ***(0.5 P)***

Disabling the periodic notification (e.g., by masking the signal) during critical sections is sufficient, like disabling interrupts on single CPU systems. ***(1 P)***

Note that spinlocks never work for ULTs as threads do not run in parallel.

b) Atomare Instruktionen werden gewöhnlich für lock-freie Algorithmen und Spinlocks eingesetzt.

Im Folgenden schreiben Sie weiter an einer Kompatibilitätsbibliothek, die auf verschiedenen Plattformen (unterschiedliche Prozessorarchitekturen und Betriebssystemen) läuft.

Atomic instructions are commonly used to implement lock-free algorithms and spinlocks.

In the following you continue writing a compatibility library that runs on various platforms (different processor architectures and operating systems).

i) Implementieren Sie `AtomicFetchAndAdd()` für eine Plattform, die diese Funktionalität nicht bereitstellt. Sie können die `acquire_lock()` und `release_lock()` verwenden, welche typische Lock-Semantik haben. Minimieren Sie die vom Lock geschützte Region. **3.5 P**

Implement `AtomicFetchAndAdd()` for a platform that does not provide this functionality. You can use the `acquire_lock()` and `release_lock()` which have common lock semantics. Minimize the region protected by the lock.

Solution:

```
long tmp;

acquire_lock(&glLock);
  tmp = *Addend;
  *Addend += Value;
release_lock(&glLock);

return tmp;
```

- ***(0.5 P)*** *for each correct line*
- ***(–0.5 P)*** *for incorrect semantics*
- ***(–0.5 P)*** *wrong return value*

ii) Begründen Sie warum ihre Lösung minimal ist. Wieso kann die Region nicht weiter verkleinert werden? **1 P**

Give reasons why your solution is minimal! Why can the region not be reduced?

Solution:
The complete read-modify-write operation must be within the locked region. Otherwise, a different core could already see partial results and thus no atomicity could be guaranteed.

iii) Nennen Sie zwei weitere atomare Instruktionen, die Lese- und Schreibinstruktionen verbinden. **1 P**

Name two additional atomic instructions which combine load and store instructions.

Solution:
TestAndSet, CompareAndSwap / CompareXCHG, LinkedLoad-ConditionalStore, all x86 interlockable operations, also all valid instructions from other platforms

c) Spinlocks werden häufig zur Realisierung von Locks auf Multiprozessorsystemen eingesetzt. Sie haben bei mobilen Anwendungen einen Nachteil und verhindern das effektive Power-Management. Wie können Spinlocks verbessert werden, damit sich im Idealfall nicht so viel Energie verbrauchen? Nennen Sie eine Verbesserung und beschreiben Sie diese kurz. Welches bekannte Lock kann stattdessen verwendet werden? **2 P**

Spinlocks are commonly used to realize locks on multiprocessor systems. They have a disadvantage for mobile applications and hinder effective power management. How can spinlocks be improved so they do not waste that much energy in the ideal case? Name one optimization and briefly describe it. What known lock could be used instead?

Solution:
Spinlocks waste CPU cycles for spinning and can be improved by limiting the time spent spinning. ***(0.5 P)*** *Once such a limit is reached, the lock should yield / sleep / defer to a blocking lock.* ***(1 P)***

Name of Improved Lock: *two-phase lock, hybrid lock* ***(0.5 P)***

d) Nennen Sie zwei Methoden zur Behandlung von Verklemmungssituationen (Deadlocks), und beschreiben Sie kurz ihre Tauglichkeit für den Alltag. Welche dieser Methoden ist am besten für neugeschriebene Programme geeignet und kann proaktiv genutzt werden? Benennen Sie hierfür auch zwei Deadlockbedingungen, die mit dieser Methode gelöst werden können! **2 P**

Name two methods to handle deadlocks and briefly describe their practical suitability. Which method is best suited for newly written programs and can be proactively used? Also name two deadlock conditions which can be solved with this method!

Solution:

Deadlock Prevention *Make deadlocks impossible to occur by negating one of the required deadlock conditions.* ***(0.5 P)***
This is a practical solution that can be used proactively. ***(0.5 P)***

Deadlock Avoidance *On every resource request, decide if the system stays in a safe state.* ***(0.5 P)***

Deadlock Detection *Detect deadlocks using a wait-for graph, then recover.* ***(0.5 P)***

2 Conditions: (0.5 P) *for two*
- *mutual exclusion*
- *hold and wait*
- *no preemption*
- *circular wait*

e) Implementieren Sie basierend auf `AtomicFetchAndAdd()` oder kurz `AFAA()` einen einfachen Ringpuffer fester Größe für das Logging in ihrer Bibliothek. Ist am Ende des Ringpuffers nicht ausreichend Platz um eine beliebig große Allokation zu erfüllen, muss die Allokation wiederholt werden. Der bereits reservierte Bereich bleibt ungenutzt. **2 P**

Hinweis: `WRAP_BUFFER()` stellt sicher, dass die Eingabe innerhalb der definierten Puffergröße bleibt. Betrachten Sie z.B. den Index 0x10002, welcher die Puffergröße um 3 überschreitet. Nach Anwendung von `WRAP_BUFFER` wird dieser auf 0x00002 reduziert.

Implement based on `AtomicFetchAndAdd()` or short `AFAA()` a simple ring buffer of fixed size for the logging in your library. If there is not enough space at the end of the ring buffer to fullfill the variable-sized allocation, the allocation must be retired. The already reserved region remains unused.

Hint: *`WRAP_BUFFER()` ensures that the input stays within the defined buffer size. Note, e.g., the index 0x10002 which exceeds buffer size by 3. After applying `WRAP_BUFFER` it is reduced to 0x00002.*

Solution:

```
#define LOG_BUF_SIZE 0x10000
#define WRAP_BUFFER(x) (((unsigned long)x) % LOG_BUF_SIZE)

char datalog[LOG_BUF_SIZE];
/// Assume long will correctly wrap around on supported platforms
volatile long curpos = 0;

char* reserveLogEntry(uint8_t size)
{
  long checkSize, retIdx;
  long tpos;

  do {
      tpos = AtomicFetchAndAdd(&curpos, size);

      checkSize = size + WRAP_BUFFER( tpos );
  } while (checkSize > LOG_BUF_SIZE);

  retIdx = WRAP_BUFFER( tpos );

  return &datalog[retIdx];
}
```

(1 P) **(0.5 P)** **(0.5 P)**

f) Der Ringpuffer verwendet eine atomare Operation und bedarf, je nach Verwendung, einer Schleife. Wie könnte das Reservieren alternativ gelöst werden, damit die Allokation nicht wiederholt werden muss? **1 P**

Begründen Sie ihre Antwort.

The ring buffer uses one atomic operation and needs, depending on the use case, a loop. How could the reservation be solved alternatively, so the allocation must not be retried?

Give reasons for your answer.

Solution:
Atomic instructions typically only guard one load-modify-store operation, thus it is impossible to load the value, compare the result against a different value and only store the result on success. The wrap-around operation must be included in the guarded region and this is not possible with AtomicFetchAndAdd.

This could be solved with:
- *locks guarding the whole reservation process*
- *transactional memory*
- *(new) more complex instructions.*

g) Welchen Mechanismus könnten Sie verwenden, um andere Prozesse über einen vollen Ringpuffer zu informieren? **1 P**

Nennen und beschreiben Sie kurz einen Mechanismus.

What mechanism could you use to inform other processes about a full ring buffer?

Name and briefly describe a mechanism.

Solution:
IPC mechanism *(e.g., message passing, sockets, pipe, signal)*
Send a message to the other process.
Semaphore *Increment the semaphore when writing an entry to the ring buffer.*
Condition Variable *Signal the condition variable when writing an entry to the ring buffer.*

Total: 20 P

Aufgabe 3: Persistenz (20 P)

Assignment 3: Persistence (20 P)

a) Write-Ahead-Logging ist eine Journaling-Technik, bei der Änderungen an einer Datenbank vor ihrer Ausführung in einen sogenannten Write-Ahead-Log (WAL) geschrieben und auf einen Hintergrundspeicher persistiert werden. Für jede Änderung wird ein weiterer Eintrag hinten am WAL angehängt. In dieser Aufgabe implementieren Sie diese Technik in einem einfachen Key-Value-Store mit nur einem Thread. Nehmen Sie für die folgenden Aufgaben an, dass alle notwendigen Header bereits inkludiert sind und dass Sie keine Fehlerbehandlung implementieren müssen. Der Zustand des Key-Value-Stores wird durch folgende Datenstruktur beschrieben:

Write-ahead logging is a journaling technique that writes updates of a database to a so-called write-ahead log (WAL) before executing them and persists them on a storage device. For each update, an additional entry is appended to the WAL. In this exercise, you will implement this technique in a basic key-value store with only a single thread. Assume for this exercise that all necessary headers are already included and that error handling does not need to be implemented. The state of the key-value store is described using the following data structure:

```
struct kv_ctx {
      int wal_fd;
      // ... in-memory kv representation
};
```

i) Implementieren Sie die Funktion `open_wal`, welche die Datei `wal` im aktuellen Verzeichnis öffnet und den Dateideskriptor in `wal_fd` (Teil von `struct kv_ctx`) speichert. Die geöffnete Datei soll folgende Anforderungen erfüllen: **1.5 P**

- ermöglicht lesenden und schreibenden Zugriff
- schreibende Zugriffe sollen immer am Dateiende angehängt werden (*append*)
- falls die Datei nicht existiert, soll sie angelegt werden (übergeben Sie `0660` als `mode`-Argument)

Implement the function `open_wal`, that opens the file `wal` in the current directory and stores the file descriptor in `wal_fd` (part of `struct kv_ctx`). The opened file should adhere to the following requirements:

- *allows read and write access*
- *writes should always be appended to the end of the file*
- *if the file does not exist, it shall be created (pass `0660` as `mode` argument)*

Solution:

```
void open_wal(struct kv_ctx *ctx) {
  ctx->wal_fd = open("wal", O_CREAT | O_APPEND | O_RDWR, 0660);
}
```

Note: even when the file offset is identical after creating the file using `open()` with `O_CREAT` and `creat()`, the resulting file descriptors will behave differently with respect to writes due to the lack of `O_APPEND`.

Common mistakes:
- *open flags separated by* `||` *or comma*
- *returning a file descriptor obtained from* `creat()` *(fd is write-only and lacks* `O_APPEND` *file status flag, see note)*

ii) Vervollständigen Sie die Funktion `write_wal_cmd`, welche mittels `dprintf` die durch `cmd` beschriebene Operation in den WAL schreibt. Nutzen Sie die vordefinierten Format-Strings (`PR_UPDATE_FMT` und `PR_DEL_FMT`) und stellen Sie durch geeignete Systemaufrufe sicher, dass in den WAL geschriebene Operationen vor dem Verlassen der Funktion garantiert im Hintergrundspeicher persistiert sind. **1.5 P**

Complete the function `write_wal_cmd`*, that uses* `dprintf` *for writing the operation described by* `cmd` *to the WAL. Use the predefined format strings (*`PR_UPDATE_FMT` *and* `PR_DEL_FMT`*) and ensure by using suitable system calls that operations written to the WAL get persisted to the backing storage before leaving the function.*

```
struct cmd {
  enum cmd_op op;
  char key[16]; // alphanumeric C-String
  char value[16]; // alphanumeric C-String
};
```

```
enum cmd_op {
  OP_GET,
  OP_UPDATE,
  OP_DEL,
};
```

Solution:

```
void write_wal_cmd(int wal_fd, struct cmd *cmd)
{
  switch(cmd->op) {
  case OP_UPDATE:
    dprintf(wal_fd, PR_UPDATE_FMT, cmd->key, cmd->value);
    break;
  case OP_DEL:
    dprintf(wal_fd, PR_DEL_FMT, cmd->key);
    break;
  default: // only called for writes, i.e., UPDATE and DEL
  }

  fdatasync(wal_fd);
}
```

Common mistakes:
- *no fd passed to* `dprintf()`
- *arguments specified in format string missing*

iii) Eignet sich die *contiguous allocation* Dateiallokationsstrategie für einen WAL auf einem stark fragmentierten Datenträger? Begründen Sie Ihre Antwort. **1 P**

Hinweis: bedenken Sie, dass für jede Änderung am Datenbestand ein neuer Eintrag am WAL angehängt wird.

Is the contiguous file allocation strategy suitable for a WAL on a heavily fragmented storage device? Explain your answer.

Hint*: keep in mind that for each update of the dataset, a new entry is appended to the WAL.*

Solution:
*No, assuming that the maximum WAL size is not known at the time of allocation, contiguous allocation is not suitable **(0.5 P)**. This is because the WAL continuously grows requiring the file to be resized frequently which is not possible without reallocating when there is no unallocated space behind the file. On a heavily fragmented storage device, finding a sufficiently large continuous range of free blocks is further complicating WAL resizing **(0.5 P)**.*

iv) Um den Datenbestand aus dem WAL wiederherzustellen, wird dieser Zeile für Zeile eingelesen. Vervollständigen Sie dafür die Implementation von `read_line`, welche die nächste Zeile aus `fd` liest und in den übergebenen Puffer `line` kopiert. Ihre Implementation soll dabei wie folgt vorgehen: **3.5 P**

- Lesen Sie die nächsten 64 Bytes aus `fd` in `line` ein. Sie dürfen annehmen, dass `read()` nur weniger Bytes als angefordert liest, wenn das Ende der Datei (*EOF*) erreicht ist, und dass jede Zeile im WAL aus weniger als 64 Zeichen besteht.
- Lokalisieren Sie das Ende der ersten Zeile im Puffer `line` mittels `memchr`.
- Stellen Sie sicher, dass auf **die erste Zeile** in `line` direkt ein terminierendes NULL-byte folgt.
- Setzen Sie abschließend die aktuelle Leseposition mittels `lseek` auf das erste Zeichen, welches nicht zur gelesenen Zeile gehört. Sollte `memchr` keinen Zeilenumbruch in `line` finden, setzen Sie die Leseposition auf das Offset vor dem `read`-Aufruf.

To recover the dataset from the WAL, it is read line by line. For this, complete the implementation of `read_line` that reads the next line from `fd` and copies it into the buffer `line`. Your implementation should work as follows:

- *Read the next 64 bytes from `fd` into `line`. You may assume that `read()` only returns fewer bytes than requested when the end of the file (EOF) is reached, and that each line in the WAL consists of less than 64 characters.*
- *Locate the end of the first line in the buffer `line` using `memchr`.*
- *Ensure that **the first line** in `line` is directly followed by a terminating NULL byte.*
- *Finally, set the current read position using `lseek` to the first character, that does not belong to the line read. If `memchr` does not find a line break in `line`, set the read position to the offset before the call to `read`.*

Solution:

```
// maximum length of line in wal (including line break)
#define WAL_LINE_LEN 64

char *read_line(int fd, char line[WAL_LINE_LEN + 1]) {
  ssize_t bytes = read(fd, line, WAL_LINE_LEN);            // .5P
  if (bytes <= 0)                                          // .5P
    return NULL;

  char *end = memchr(line, '\n', bytes);                   // .5P
  if (end) {
    size_t len = (end - line) + 1;
    line[len] = '\0';                                      // .5P
    lseek(fd, -(bytes - len), SEEK_CUR);                   //  1P
    return line;
  }

  lseek(fd, -bytes, SEEK_CUR);                              // .5P
  return NULL;
}
```

Common mistakes:

- *passing an array pointer to functions expecting a char/void pointer (e.g., `read(fd, &line, 64)`)*
- *returning NULL when reading less than 64 B (WAL entries are **up to** 64 B but might be shorter)*

Note: while using a fixed-size array in the function signature documents the assumed buffer size of the implementation, the array size is not strictly enforced and only serves as hint. While certain compilers might emit warnings when passing a fixed-size array that is too small, this is not always the case. Therefore, you should not rely on this syntax when you want the compiler to enforce the array size of `line`.

If you want the compiler to enforce the array size, you can either use array pointers (e.g., `char *read_line(int fd, char (*line)[WAL_LINE_LEN + 1]))`, the `static` keyword[2] (e.g., `char *read_line(int fd, char line[static WAL_LINE_LEN + 1]))`, or wrap the array in a struct. Note, however, that there are subtle differences between each of these variants.

b) Bei einem Stromausfall gehen alle Änderung am Dateisystem, welche in volatilem Speicher gepuffert sind, verloren. Wenn nicht vorsichtig vorgegangen wird, könnte dabei das Dateisystem in einem inkonsistenten Zustand hinterlassen werden. Geben Sie ein Beispiel für einen inkonsistenten Zustand an, welcher beim Verschieben einer Datei auftreten kann, und beschreiben Sie, was passieren muss, dass sich dieser inkonsistente Zustand manifestiert. Nennen Sie zudem den Namen des **2 P**

[2] supported in C99 and onwards

Werkzeugs, das typischerweise für die Erkennung von inkonsistenten Zuständen verwendet wird, nachdem sie bereits aufgetreten sind.

On power loss, all file system updates buffered in volatile memory are lost. When no proper care is taken, this might leave the file system in an inconsistent state. Give one example of an inconsistent state that might occur while moving a file and describe what needs to happen for this inconsistent state to manifest itself. Further, name the tool that is typically used for detecting inconsistent states after they have already occurred.

Solution:
*Exemplary solution: Invalid hard link count **(0.5 P)**: the new directory entry was already created in the target directory but the source directory entry was not removed before the power loss occurred **(0.5 P)**. Assuming that the file system does not change the hard link count during a rename operation (as the number of hard links before/after the rename is identical), the hard link count now deviates from the actual number of directory entries present on the file system **(0.5 P)**.*

Note: even when the file system increases/decreases the hard link count during the rename operation, we can come up with a scenario where the hard link count becomes invalid (e.g., the power loss occurs after the hard link count was updated (and also persisted) but before the new directory entry is persisted on the storage device). Another potential outcome could be the "loss" of the file (source hard link was removed before new hard link was persisted on disk). In general, this problem comes down to the atomicity of the move operation (or the lack thereof) with regards to a system crash. The property of a file system to remain in a consistent state no matter the sequence of operations or the occurrence of unexpected crashes is called crash consistency. Journaling is a commonly used technique to ensure crash consistency for file system metadata.

*The `fsck` (file system check) **(0.5 P)** tool is used for checking the file system for inconsistencies. It can optionally be used for repairing file systems (see `man 8 fsck` for further details). As checking and repairing a file system depends on its implementation, each file system has its own file system checker. Note that not all types of inconsistent states are repairable.*

c) Erklären Sie, warum sequentielles Lesen auf Festplatten erheblich schneller als wahlfreies Lesen ist. Beschreiben Sie weiter, wie sich das ein Betriebssystem zu Nutze machen kann, um den I/O-Durchsatz zu erhöhen. **1.5 P**

Explain why reading form hard drives sequentially is significantly faster than random reads. Further, describe how an operating system can make use of this to increase the I/O throughput.

Solution:
*HDDs have mechanical read/write heads that need to be positioned before accessing the disk (next to the seek time there is additional rotational delay) **(0.5 P)**. When accessing the disk sequentially, head movement (as well as rotational delay) during access is minimal whereas the random access performance suffers from many expensive seeks **(0.5 P)**.*

*Operating systems typically feature I/O schedulers that can reorder outstanding IOs such that head movement is minimized **(0.5 P)**. (Alternative: The operating system /*

file system can aim to write related/adjacent data to disk sequentially such that accessing large chunks of related data (e.g., a file) requires few seeks.)

d) Nehmen Sie an, dass das folgende Shell-Script in einem leeren Verzeichnis ausgeführt wird. Beschreiben Sie die Dateiinhalte nach der Ausführung des Scripts, indem Sie Tabelle 2 ausfüllen. Wenn nicht auf eine Datei zugegriffen werden kann, schreiben sie X in den entsprechenden Tabelleneintrag. Tabelle 1 dient als Beispiel und zeigt die Dateiinhalte, nachdem das Script bis zu Zeile 5 ausgeführt wurde. **2.5 P**

Hinweis: der Befehl `rm` löscht die angegebene Datei ohne dabei Links zu folgen.

Assume that the following shell script is executed in an empty directory. Describe the file contents after running the script by filling out table 2. If a file cannot be accessed, write X into the corresponding table entry. Table 1 serves as an example and shows the file contents after executing the script up until line 5.

Hint*: the `rm` command deletes the given file without following links.*

Solution:

a	***b***	***c***	***d***	***e***
A	***X***	***3***	***X***	***B12***

***(0.5 P)** for each correct entry*

Note: `>>` does not overwrite the current file contents but appends additional contents at the end of the file.

e) Beschreiben Sie den Unterschied zwischen *mandatory file locks* (verpflichtend) und *advisory file locks* (empfehlend/beratend). **1 P**

Describe the difference between mandatory file locks and advisory file locks.

Solution:
When using mandatory file locks, the operating system enforces acquired locks, preventing access to locked files. ***(0.5 P)***

The lock status of advisory file locks can be queried by applications but only serve as hint to other applications (i.e., locks might be ignored by other applications). The operating system does not enforce acquired file locks in this case. ***(0.5 P)***

f) **UNIX Access Rights**

i) Welche Zugriffsrechte werden durch `rwxr-xr--` im klassischen UNIX-Zugriffsmodell beschreiben? Vervollständigen Sie die folgende Tabelle für jedes Subjekt (ja/✓ für gewährte Berechtigungen, nein/✗ für fehlende Berechtigungen). **1.5 P**

Which access rights are described by `rwxr-xr--` in the traditional UNIX access model? Complete the following table for each subject (yes/✓ for granted permissions, no/✗ for missing permissions).

Solution:

***Subjekt*/Subject**	***Read***	***Write***	***Execute***
User/Owner	✓	✓	✓
Group	✓	✗	✓
Other	✓	✗	✗

***(0.5 P)** for each fully correct row*

ii) Welchen Zweck erfüllt die Execute-Berechtigung auf Verzeichnissen? **0.5 P**

What purpose does the execute permission serve on directories?

Solution:
The eXecute permissions on a directory states whether the respective subject is allowed to access files in the directory.

Note: for directories, the execute permission is sometimes also referred to as search permission. It is required for accessing a file and querying its stats, or changing the working directory to this directory (or a subdirectory). Listing the contents of a directory requires the read permission.

iii) Klassische UNIX-Zugriffsrechte bieten aufgrund der begrenzten Anzahl an Subjekte nicht immer die benötigte Flexibilität. Nennen Sie ein anderes UNIX-Konzept, was hier Abhilfe schaffen kann, und beschreiben Sie kurz, wie es funktioniert. **1 P**

Traditional UNIX access rights sometimes fail to offer enough flexibility due to the limited number of subjects. Name a different UNIX concept that can help here, and describe briefly how it works.

Solution:
*Access Control Lists (ACLs) **(0.5 P)** are attached to files and consist of ACL entries. Each ACL entry specifies additional permissions (rwx) for a specific user or group **(0.5 P)**.*

g) Nehmen Sie an, Sie verwenden einen Festplattenverbund aus fünf Festplatten. Wie viele Festplatten können in den folgenden RAID-Leveln ausfallen, **bevor** Daten unwiederbringlich verloren gehen? Begründen Sie Ihre Antwort für jedes RAID-Level. **2.5 P**

Geben Sie zudem an, wofür das Akronym RAID steht.

*Assume that you use a hard drive array consisting of five hard drives. How many hard drives can fail **before** data is lost irrecoverably? Justify your answer for each RAID level.*

Further, state what the acronym RAID stands for.

Solution:

Level	***#Ausfälle*/#Failures**	***Erklärung*/Explanation**
RAID 0	*0*	*Blocks distributed across all drives (striping), no redundancy or mirror available*
RAID 5	*1*	*Data and parity blocks distributed across all drives. When a single drive fails, the lost data can be reconstructed using the remaining blocks and their parity data.*

***RAID**: Redundant Array of Independent Disks (or Redundant Array of Inexpensive Disks)*

Total: 20 P