Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

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

Musterlösung Nachklausur

Theorie
29.09.2020

Alle Punkteangaben ohne Gewähr!

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

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

- Die Prüfung besteht aus 15 Blättern: Einem Deckblatt und 14 Aufgabenblättern mit insgesamt 5 Aufgaben.

  The examination consists of 15 pages: One cover sheet and 14 sheets containing 5 assignments.

- Es sind keinerlei Hilfsmittel erlaubt!

  No additional material is allowed!

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

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

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

  You can use the back side of the 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).

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

Aufgabe	1	2	3	4	5	Total
Max. Punkte	9	9	9	9	9	45
Erreichte Punkte						
Note						

Aufgabe 1: Grundlagen

Assignment 1: Basics

a) Nennen Sie eine zentrale Betriebssystemabstraktion. **0.5 pt**

Name a central operating system abstraction.

Solution:

Process/Thread, File, or Virtual Memory

Wo oder wann werden durch diese Abstraktion zusätzliche Systemressourcen verbraucht? Geben Sie zwei Beispiele an. Begründen Sie Ihre Antwort. **2 pt**

Where or when does this abstraction consume additional system resources? Give two examples. Justify your answer.

Solution:

Process *The process abstraction introduces a runtime overhead on context switches* **(1.0 P)**, *that is by saving and restoring register state. Each process/thread also requires its own stack, which increases the memory footprint of the application* **(1.0 P)**.

File *The additional indirection caused by translating file offsets to physical sector offsets slows down sector accesses* **(1.0 P)**. *In addition, respective data structures for address translation must be created, maintained, and stored on the storage device* **(1.0 P)**.

Virtual Memory *Similar to files, virtual memory comes with additional costs for address tanslation (TLB misses, page faults)* **(1.0 P)**. *Switching between virtual address spaces is also a costly operation, especially when TLBs have to be flushed (i.e., no tagging). Just like with files, keeping management structures such as page tables consume processor time and memory* **(1.0 P)**.

Beschreiben Sie eine Situation, in der die gleiche Abstraktion Systemressourcen einspart. Begründen Sie Ihre Antwort. **1 pt**

Describe a situation in which the same abstraction saves system resources. Justify your answer.

Solution:

Process *If the current process waits for I/O, switching to another process in the meantime can increase the overall system utilization, i.e., saves CPU time.*

File *The additional indirection provided by files allows unallocated areas to transparently return zero-sectors, saving the space on the storage device (sparse files).*

Virtual Memory *A huge benefit of virtual memory is the possibility to savely share identical data between processes (e.g., libraries), thus eliminating the need to hold multiple copies in memory.*

b) Wieso können Systemaufrufe in einem Betriebssystem wie Linux konzeptionell nicht durch Prozeduraufrufe ersetzt werden? **1 pt**

Why can system calls in an operating system like Linux conceptually not be replaced by procedure calls?

Solution:

An important task of a system call is to change the privilege level **(0.5 P)** *of the CPU when entering the kernel. This cannot be done with a procedure call* **(0.5 P)** *and would force the OS to give up the separation between user and kernel mode.*

Nennen Sie einen Grund neben Einfachheit, weshalb Anwendungen üblicherweise Systemaufrufe nicht selbst durchführen, sondern Sprach- oder Systembibliotheken dafür verwenden? **1 pt**

Give a reason besides simplicity why applications usually do not invoke system calls themselves but use language or system libraries instead?

Solution:

Compatibility/Portability. If the system call interface is not stable (e.g., Windows), updating the OS would require developers to also update their applications and potentially keep and maintain different versions. But even with a stable interface, invoking a system call is highly architecture specific. Language and system libraries help to abstract away from such details, making applications more portable.

Welcher andere CPU-Mechanismus neben der Trap-Instruktion könnte technisch benutzt werden, um synchron aus dem Benutzermodus in den Kernel zu gelangen? **0.5 pt**

What other CPU mechanism besides the trap instruction could technically be used to synchronously enter the kernel from user mode?

Solution:

CPU exceptions / software interrupts

c) Nennen und erläutern Sie zwei grundlegende Techniken, um in einem Betriebssystem auf den Abschluss einer Ein-/Ausgabeoperation zu warten. **2 pt**

Name and explain two basic techniques for an operating system to wait for the completion of an input/output operation.

Solution:

Programmed I/O, Polling *The CPU busy waits for the completion of the I/O operation by repeatedly querying a status register of the device* **(1.0 P)**.

Interrupts *The device raises a previously determined interrupt line or crafts a message for an interrupt controller. The CPU then jumps to a configured interrupt service routine upon receiving the interrupt, thereby signaling the completion of the I/O operation* **(1.0 P)**.

d) Erläutern Sie, was unter *kooperativem Multitasking* zu verstehen ist. Welcher Systemaufruf spielt dabei eine besondere Rolle? **1 pt**

Explain what is meant by cooperative multitasking. *Which system call plays a special role in this context?*

Solution:

With cooperative multitasking, the operating system does not forcefuly preempt processes or threads to perform a context switch. Instead, tasks are expected to voluntarily release the CPU, that is cooperate with the other processes **(0.5 P)**. *Releasing the CPU is usually done by invoking the yield system call* **(0.5 P)**. **Total: 9.0pt**

Aufgabe 2: Prozesse und Threads

Assignment 2: Processes and Threads

a) Nennen Sie vier typische Ziele eines Scheduling-Algorithmus. **1 pt**

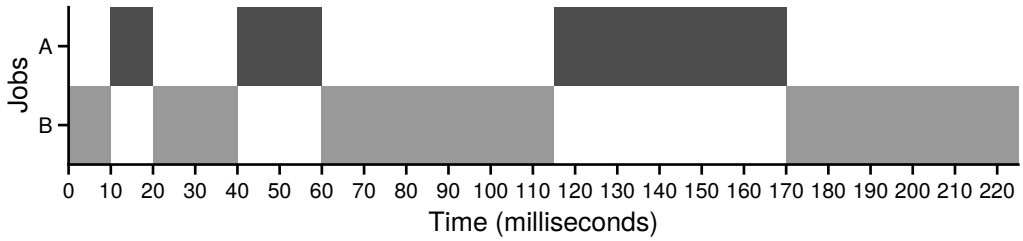
Name four typical goals of a scheduling algorithm.

Solution:

Fairness, balance, low response time, high throughput, low scheduling overhead, high CPU utlization

b) Die Abbildung unten zeigt zwei CPU-gebundene Prozesse in einem System mit einem MLFQ-Scheduler. Im weiteren Zeitverlauf nach 60 ms wechseln sich die beiden Prozesse regelmäßig ab.

The figure below shows two CPU-bound processes running in a system with an MLFQ scheduler. After 60 ms, the two processes alternate regularly.



Wie viele Warteschlangen nutzt der MLFQ-Scheduler? **0.5 pt**

How many queues does the MLFQ scheduler use?

Solution:

The scheduler uses three queues.

Welcher Scheduling-Algorithmus wird für Prozesse innerhalb einer Warteschlange verwendet? **0.5 pt**

Which scheduling algorithm is used for processes within one queue?

Solution:

Round Robin

Wie groß ist die Zeitscheibe in der Warteschlange höchster Priorität? **0.5 pt**

How long is the time slice in the highest-priority queue?

Solution:

10 ms

Wie groß ist die Zeitscheibe in der Warteschlange niedrigster Priorität? **0.5 pt**

How long is the time slice in the lowest-priority queue?

Solution:

55 ms

Nehmen Sie an, dass der Scheduler alle 500 ms alle Prozesse in die Warteschlange höchster Priorität verschiebt. Welches Problem wird dadurch gelöst? Beschreiben Sie ein Szenario, in dem das Problem auftreten würde. **1 pt**

Assume that the scheduler moves all processes to the highest-priority queue every 500 ms. Which problem does this strategy solve? Describe a scenario where this problem might occur.

Solution:

Starvation **(0.5 P)** *would occur if a CPU-bound process runs at the same time as lots of I/O-bound processes.* **(0.5 P)** *The I/O-bound processes would always stay in the high-priority queues, preventing the CPU-bound process in the low-priority queue from running.*

c) Erklären Sie den Unterschied zwischen einem Programm und einem Prozess. **1 pt**

Explain the difference between a program and a process.

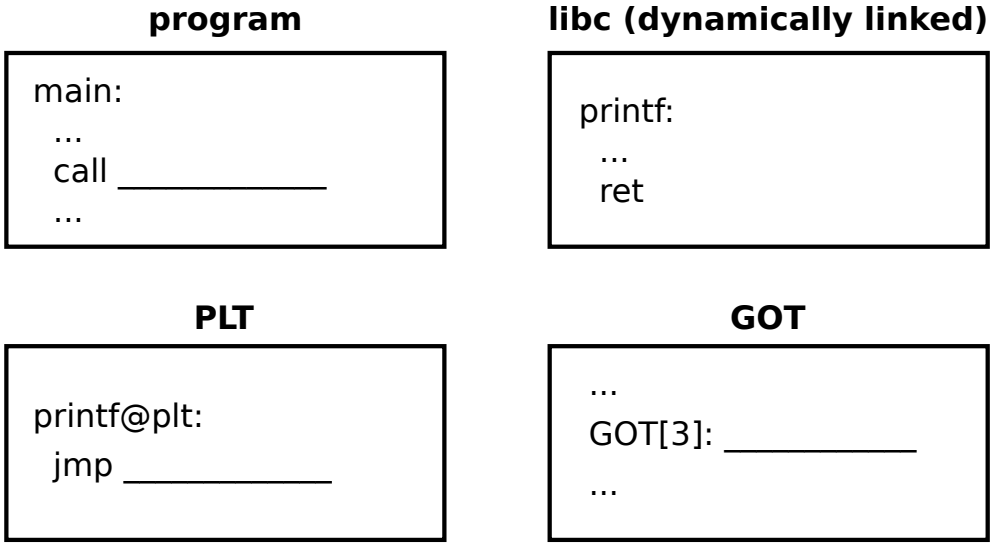
Solution:

Program vs. process is like recipe vs. cooking. **(1 P)**

A program describes the memory layout and CPU instructions, whereas a process is the activity of actually executing a program.

d) In der Grafik unten ist der Assembler-Code eines Programms dargestellt, das mit der libc dynamisch gebunden wurde. Füllen Sie die leeren Felder aus, sodass ein Aufruf der `printf`-Funktion in libc durchgeführt wird. Nehmen Sie an, dass `printf` bereits zuvor aufgerufen wurde. **1.5 pt**

The figure below shows the assembler code of a program linked dynamically with libc. Fill in the empty fields so that the program calls `printf` in libc. Assume that there have been previous calls to `printf`.



Solution:

```
call printf@plt
jmp GOT[3]
GOT[3]: &printf
```

Welchen Vorteil bringt *lazy dynamic binding*? Erklären Sie kurz. **1 pt**

What is the advantage of lazy dynamic binding*? Explain shortly.*

Solution:

Lazy dynamic binding reduces the program's startup time, because the individual functions in the dynamically-linked libraries are only resolved in PLT and GOT when they are first called.

e) Wenn ein Prozess `fork()` aufruft, erstellt das System eine fast exakte Kopie des Elternprozesses. Warum ist diese Kopie nur „fast exakt" und nicht „exakt"? **1 pt**

When a process calls `fork()`, the system creates a nearly exact copy of the parent process. Why is the copy only "nearly exact" and not an "exact" copy?

Solution:

The process id of the child needs to be unique and thus cannot be copied. **(0.5 P)** *To distinguish parent and child in the code, the fork() call returns 0 in the child and the child's pid in the parent.* **(0.5 P)**

f) In welchem Threadmodell kommen *Scheduler Activations* zum Einsatz? **0.5 pt**

Which thread model uses scheduler activations*?*

Solution:

Hybrid Threads / M-to-N

Total: 9.0pt

Aufgabe 3: Koordination und Kommunikation von Prozessen

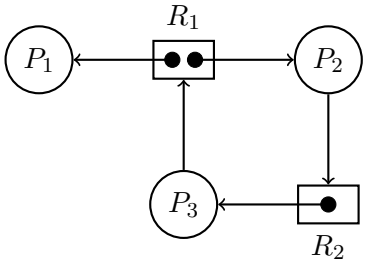
Assignment 3: Process Coordination and Communication

a) Zeichnen Sie einen *Resource Allocation Graph*, der einen Zyklus enthält, aber keinen Deadlock darstellt. **1 pt**

Draw a resource allocation graph *which contains a cycle but does not represent a deadlock.*

Solution:

One of the resources in the cycle must have multiple instances **(1.0 P)***:*



Note that it is not possible that processes wait for resources that are available, so all resources in the cycle have to be allocated to a process.

b) Erklären Sie den Begriff *Priority Donation*. **1 pt**

Explain the term priority donation*.*

Solution:

If a process waits for another process with lower priority **(0.5 P)***, priority donation temporarily increases the priority of that process to match that of the first process* **(0.5 P)***.*

Gegeben sei ein Scheduler, der Priority Donation implementiert. Löst diese Implementierung alleine das Problem der *Priority Inversion* bei Benutzung einfacher Spinlocks? Begründen Sie Ihre Antwort. **1.5 pt**

Assume a scheduler that implements priority donation. Does this implementation alone solve the problem of priority inversion *when using simple spinlocks? Justify your answer.*

Solution:

No **(0.5 P)***, because a simple spinlock does not yield control to the operating system, so the scheduler does not even know that the thread is waiting* **(1.0 P)** *or for which other thread a thread is waiting.*

Note: A simple spinlock does not store the ID of the process in the critical section. Even if the threads waiting would want to donate their priority, they could not identify the process blocking them.

c) Welches Problem ergibt sich bei Nutzung von synchronem Message Passing zwischen Prozessen in einem System mit User-Level-Threading? **1 pt**

Which problem arises when using synchronous message passing between processes in a system with user-level threading?

Solution:

Sychronous message passing is blocking, and in a system with user-level threading a blocking thread blocks all other threads as well.

d) Der folgende Code implementiert ein Spinlock. Ein Thread, der in den kritischen Abschnitt eintritt, schreibt mittels der atomaren `swap`-Instruktion seine eindeutige Thread-ID (>0) in die Spinlock-Variable und liest den vorherigen Wert aus. Falls die Variable 0 enthielt, tritt der Thread in den kritischen Abschnitt ein, ansonsten macht er den Schreibvorgang rückgängig.

Nennen Sie für die drei Anforderungen an einen Synchronisationsmechanismus jeweils, ob diese erfüllt sind oder nicht. Begründen Sie Ihre Antwort. **4.5 pt**

The following code implements a spinlock. A thread that enters the critical section writes its unique thread ID (>0) into the spinlock variable using the atomic `swap` instruction and reads the previous value. If the variable was 0, the thread enters the critical section. Otherwise, the write access is reverted.

Explain for each of the three requirements for a synchronization primitive whether they are fulfilled or not. Justify your answer.

```
1  int lock_var = 0;
2
3  void lock(void) {
4      int x;
5      while (1) {
6          x = thread_id();
7          swap(&x, &lock_var); /* atomic */
8          if (x == 0)
9              break; /* enter section */
10         else
11             swap(&x, &lock_var); /* revert access */
12     }
13 }
14
15 void unlock(void) {
16     int x = 0;
17     swap(&x, &lock_var);
18 }
```

Solution:

Mutual exclusion *Yes* **(0.5 P)**. *`lock_var` is only zero if no thread is in the critical section* **(0.5 P)**. *A thread only enters when `lock_var` is zero, and the variable is atomically set to a non-zero value on entry* **(0.5 P)**.

Progress *No* **(0.5 P)**. *For example, if one thread unlocks the lock, but only afterwards another thread reverts an unsuccessful lock attempt, the lock variable is left with a non-zero value even though no thread is in the critical section* **(1.0 P)**.

Bounded Waiting *No* **(0.5 P)**. *Threads enter the critical section in random order, so an individual thread might wait arbitrarily long* **(1.0 P)**.

Total: 9.0pt

Aufgabe 4: Speicher

Assignment 4: Memory

a) Erläutern Sie eine Methode, um bei seitenbasierter Speicherverwaltung auf x86 im Betriebssystem festzustellen, dass auf einen Speicherbereich geschrieben wurde. Geben Sie dabei die Genauigkeit an, mit der Ort, Zeit und Anzahl des Zugriffe bestimmt werden können. **1.5 pt**

Explain a method for the operating system to determine that a memory area has been written to when using page-based memory management on x86. Specify the accuracy with which the location, time, and number of accesses can be determined.

Solution:

Dirty Bit *On a write access, the MMU sets the dirty bit in the page table entry (PTE) of the target page. By periodically inspecting and resetting dirty bits, the OS can detect that at least one write access to a page (i.e., page granularity) has been taken place since the last reset. However, it is not possible to derive the exact offset within the page, the number of accesses, or the exact point in time.*

Page Faults (Variant 1) *If the pages of the respective memory area are write protected, page faults can be used to detect write accesses. However, page faults are much more costly and single-stepping or emulation is needed to allow the attempted write access to actually be performed. This leads to excessive runtime overhead. Nonetheless, this method provides the exact addresses, points in time, number of the accesses.*

Page Faults (Variant 2) *If the write protection is released after the first page fault, the overhead is much lower and no single-stepping or emulation is needed. Instead, the offending instruction can simply be re-started. However, this method only provides detailed information on the first access and it cannot be determined if, when, and where potential further accesses occurred.*

b) In einer hypothetischen Datenstruktur werden alle Elemente sequentiell auf separaten 4-KiB Seiten gespeichert. Um die Datenstruktur zu sortieren, sollen die virtuellen Addressen der Elemente über die Seitentabellen angepasst werden, anstelle die Elemente im Speicher hin- und herzukopieren. Diskutieren Sie Vor- und Nachteile dieses Vorgehens. **1.5 pt**

In a hypothetical data structure all elements are stored sequentially on separate 4-KiB pages. To sort the data structure, the virtual addresses of the elements should be adjusted through the page tables instead of copying the elements back and forth in memory. Discuss the advantages and disadvantages of this approach.

Solution:

***Pro**: We only have to adjust two PTEs to exchange two elements in the data structure* **(0.5 P)**. *This is especially advantageous if the elements are large (i.e., occupy a whole page) and we save copy time.*

***Con**: Since the respective TLB entries have to be invalidated, this method leads to a large number of TLB misses and consecutive page table walks* **(1.0 P)**. *At the same time, page table structure caches may be less effective as the page tables have been modified.*

c) Nennen Sie zwei Eigenschaften vorwärtsgerichteter Seitentabellen, die die Größe des virtuellen Adressraums bestimmen. **1.0 pt**

Name two properties of a forward page table that determine the size of the virtual address space.

Solution:

- *Number of levels*
- *Size of page*
- *Size of PTE (i.e., number of PTEs per page)*

d) Was passiert mit dynamisch alloziertem Speicher nach Programmende, wenn dieser zuvor nicht mit `free()` freigegeben wurde? **0.5 pt**

What happens to dynamically allocated memory after program termination, if it has not been freed with `free()` *before?*

Solution:

The memory is automatically freed by the operating system in the course of releasing the underlying memory pages **(0.5 P)**.

e) Was versteht man unter *Bélády's Anomalie*? **1 pt**

What is meant by Bélády's anomaly*?*

Solution:

With FIFO page replacement **(0.5 P)**, *it is possible to construct a reference string for a number* N *of page frames that performs worse with* $N + 1$ *frames* **(0.5 P)**.

f) Erläutern Sie, bei welchem Allokationsmuster die Verwendung von Arena-Allokation sinnvoll ist. Welche Vorteile bietet diese in dem Fall? **1.5 pt**

Explain for which allocation pattern it makes sense to use arena allocation. What advantages does it offer in this case?

Solution:

Arena allocation is suitable with peak allocation patterns, where many allocations are done without any intermediate frees. Instead, the objects are freed all at once at the end **(1.0 P)**. *Using area allocation in this case saves metadata as an allocation is just a pointer increment and the free just returns the whole arena in one call* **(0.5 P)**.

g) Wie werden im Speicher abgebildete Dateien (engl. Memory-Mapped Files) im Betriebssystem grundsätzlich implementiert? Gehen Sie dabei insbesondere darauf ein, wie die Konsistenz der Abbildung bei prozessfremden Schreibzugriffen (z. B. mit dem `write()` Systemaufruf) gewährleistet wird. **2 pt**

How are memory mapped files basically implemented in an operating system? In particular, explain how the consistency of the mapping is ensured in the presence of writes from foreign processes (e.g., with the `write()` system call).

Solution:

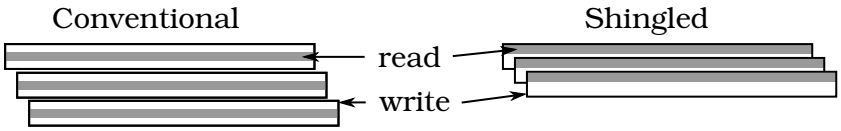
The operating system loads the file into a system-wide file cache in kernel mode **(0.5 P)**. *Memory mappings of the file are established by mapping the same cache pages into the respective user address spaces* **(0.5 P)**. *To guarantee consistency, writes via the `write()` system call are directed to the cache pages in kernel memory* **(0.5 P)**. *The OS periodically flushes the data to the actual file on disk to keep the on-disk representation consistent (write-back cache)* **(0.5 P)**. **Total: 9.0pt**

Aufgabe 5: I/O, Hintergrundspeicher und Dateisysteme

Assignment 5: I/O, Secondary Storage, and File Systems

a) Manche moderne Festplatten nutzen Shingled Magnetic Recording (SMR) zur Verbesserung der Datendichte. Beim Schreiben auf eine SMR-Spur werden die Daten auf benachbarten Spuren ebenso überschrieben. Daher müssen auch bei kleinen Änderungen auf wenigen Blöcken oft mehrere Spuren neu geschrieben werden.

Some modern hard disks use Shingled Magnetic Recording (SMR) to improve data density. A write to an SMR track destroys data on neighboring tracks. Consequently, multiple tracks might need a rewrite even when only few blocks are updated.



Eine Anwendung schreibt in eine Datei, die auf einer SMR-Festplatte abgelegt ist. Welcher existierende Betriebssystemmechanismus kann unabhängig vom verwendeten Dateisystem die Schreibgeschwindigkeit verbessern? Beschreiben Sie eine Situation, in der eine Verbesserung eintritt. **1 pt**

An application writes data to a file located on an SMR drive. Which existing operating system mechanism can improve performance independent from the file system in use? Describe a situation that shows improvement.

Solution:

The file system cache **(0.5 P)** *can improve performance if the application issues multiple writes close to each other that will end up on neighboring tracks on the SMR drive* **(0.5 P)**.

Welcher Dateisystemtyp ist für SMR-Festplatten optimal, da fast alle Schreibzugriffe sequentiell stattfinden? **0.5 pt**

Which file system type is optimal for SMR drives, as it issues sequential writes almost exclusively?

Solution:

Log-Structured File System

SSDs besitzen ähnliche Einschränkungen wie SMR-Festplatten. Beschreiben Sie diese Einschränkungen. **1 pt**

SSDs have limitations similar to those of SMR drives. Describe these limitations.

Solution:

SSDs need an erased block to write to **(0.5 P)**, *but an erase operation deletes an entire page that consists of multiple blocks* **(0.5 P)**.

Similarly, an SMR drive should only write to an empty zone (= multiple tracks that can be written independently) to avoid losing data.

SSDs besitzen einen Flash Translation Layer (FTL), der neben Wear Leveling auch bei den oben genannten Einschränkungen Abhilfe bietet. Könnte ein solcher Mechanismus auch bei SMR-Festplatten sinnvoll eingesetzt werden? Begründen Sie. **1 pt**

SSDs use a Flash Translation Layer (FTL) that primarily provides wear leveling, but also helps with the limitations above. Might a similar mechanism make sense for SMR drives as well? Justify.

Solution:

Moving data across the HDD at fine granularity (e.g., block or track) like on SSDs might introduce long, unexpected seek times on HDD. The operating system expects that accessing sequential LBAs translates to a sequential access on disk as well. Thus, introducing an FTL-like mechanism is not viable for SMR drives.

At coarser granularity, a mapping layer might make sense. An SMR zone (= multiple tracks) could be relocated to make partial changes more efficient.

b) Wofür steht die Abkürzung ACL im Kontext von Dateisystemen? **0.5 pt**

In the context of file systems, what does the abbreviation ACL stand for?

Solution:

Access Control List

Welchen Vorteil haben ACLs im Vergleich zu klassischen Unix-Berechtigungen? **1 pt**

What is the advantage of ACLs compared to standard Unix permissions?

Solution:

With Unix permissions, we are constrained to read/write/execute permissions for one user, one group and everyone else. ACLs in contrast are completely flexible and allow defining permissions for any number of users or groups.

c) Geben Sie jeweils ein Beispiel für einen absoluten und einen relativen Pfad an. **0.5 pt**

Give an example of an absolute and a relative path.

Absolute	Relative
/usr/bin/env	../../foo.txt

Welche zusätzliche Information wird zum Auflösen eines relativen Pfads benötigt? Wo wird diese Information üblicherweise gespeichert? **1 pt**

Which additional information is necessary for resolving relative paths? Where is that information usually stored?

Solution:

The working directory, stored in the process control block.

d) Warum wird bei Netzwerkgeräten üblicherweise ein Ringpuffer anstelle eines einfachen Puffers eingesetzt? **1 pt**

Why do network interfaces usually use ring buffers instead of a simple buffer?

Solution:

Network packets often come in bursts and cannot be predicted by the operating system. The circular buffer gives the OS enough time to handle the packets, even if interrupt handling might be delayed.

e) Nennen Sie eines der in der Vorlesung genannten Verfahren zur Dateiblockallokation, welches einen sequentiellen und wahlfreien Zugriff erlaubt. **0.5 pt**

Name one of the file block allocation methods presented in the lecture that supports sequential and random access.

Solution:

Contiguous allocation or indexed allocation.

f) Nennen Sie einen Vorteil (+) und einen Nachteil (-) davon, Verzeichniseinträge in einem Baum zu speichern. **1 pt**

Name an advantage (+) and a disadvantage (-) of storing directory entries in a tree structure.

Solution:

(+) *Faster directory lookup in large directories.*

(-) *Needs more storage, complex, not that efficient for small number of files.* **Total: 9.0pt**