Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

Prof. Dr. Hans P. Reiser
Mathias Gottschlag, M.Sc.
Dipl.-Inform. Marc Rittinghaus

Musterlösung Hauptklausur
02.03.2018

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 pages (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 gilt als 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 task sheets for your answers. If you need additional draft paper, please notify one of the supervisors.
- Bitte machen Sie eindeutig klar, was Ihre endgültige Lösung zu den jeweiligen Teilaufgaben ist. Teilaufgaben mit widersprüchlichen Lösungen werden mit 0 Punkten bewertet.
  Make sure to clearly mark your final solution to each question. Questions with multiple, contradicting answers are void (0 points).

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

Aufgabe	1	2	3	4	5	Total
Max. Punkte	12	12	12	12	12	60
Erreichte Punkte						
Note						

Aufgabe 1: Grundlagen

Assignment 1: Basics

a) Erläutern Sie kurz an je einem Beispiel, wo ein Betriebssystem Multiplexing in Zeit und Raum durchführt. **2 pt**

Explain briefly, using one example each, where an operating systems performs multiplexing in time and space.

Lösung:

With multiplexing a resource can be shared by multiple processes.

Time *Computation time* **(0.5 P)** *is the primary resource provided by a CPU. The operating system multiplexes CPU time with context switches, where the state of the CPU (i.e., the contents of registers) is swapped with a saved state in memory* **(0.5 P)**.

Space *Memory* **(0.5 P)** *is multiplexed by the operating system by using virtual memory and dedicated address spaces, thus allowing each process to work with the full range of virtual addresses* **(0.5 P)**.

b) Wie verbessert Multiprogramming die Systemauslastung bei E/A-Operationen? **1 pt**

How does multiprogramming improve the system utilization during I/O operations?

Lösung:

When a process initiates an I/O operation, it is often necessary to block it until the operation has finished. When having only a single program running concurrently, the CPU is thus forced into an idle phase, which wastes precious computation time **(0.5 P)**. *By having multiple programs running concurrently, the operating system can perform a context switch to another process and continue execution. This way, the CPU and I/O devices can be utilized at the same time, improving the overall system load* **(0.5 P)**.

c) Erläutern Sie die Begriffe *Interrupt Vector* und *Interrupt Service Routine*. **2 pt**

Describe the terms interrupt vector *and* interrupt service routine.

Lösung:

Interrupt Vector *An interrupt vector is either directly the entry address of an interrupt handler (e.g., on MIPS) or the index into an array of such addresses, called the interrupt vector table (e.g., on x86)* **(1.0 P)**.

Interrupt Service Routine *An ISR is the piece of system code that handles an interrupt* **(1.0 P)**. *It is often part of a device driver may be reading or writing device registers and device memory.*

d) Das Betriebssystem löst einen Interrupt aus, wenn ein Prozess eine ungültige Instruktion ausführt. **0.5 pt**

The operating system generates an interrupt when a process executes an invalid instruction.

Lösung:

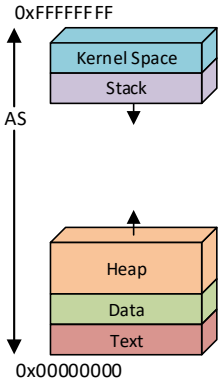
☐ *Ja / Yes* ■ *Nein / No*

e) Skizzieren Sie grafisch den üblichen Aufbau eines virtuellen Adressraums. **3 pt**

Visually depict the usual layout of a virtual address space.

Lösung:

Kernel **(0.5 P)**
Stack **(0.5 P)**
Heap **(0.5 P)**
Text/Data **(0.5 P)**
Correct position of kernel **(0.5 P)**
Correct position of heap and stack **(0.5 P)**



f) Ein Systemaufruf umfasst mehr Argumente, als CPU Register zur Verfügung stehen. Nennen Sie, in richtiger Reihenfolge, zwei zentrale Schritte, die das Betriebssystem bei der Parameterbehandlung unternehmen muss, um sich vor böswilligen Aufrufern zu schützen. **2 pt**

A system call has more arguments than there are CPU registers. Give, in the right order, two important steps the operating system has to take to protect itself from malicious callers when handling the parameters.

Lösung:

(a) Copy arguments to kernel memory **(1.0 P)** *(e.g., the kernel stack).*
(b) Check the arguments using the copy **(1.0 P)**.

Was muss für Pufferadressen (z. B. ein Zielpuffer beim Einlesen einer Datei) gelten, die an einen Systemaufruf übergeben wurden? Begründen Sie Ihre Antwort. **1.5 pt**

What must apply to buffer addresses (e.g., a destination buffer when reading a file) which were passed to system calls? Justify your answer.

Lösung:

Buffer addresses must be in user space **(0.5 P)**. *Otherwise, the caller could lead the kernel into overwriting arbitrary memory or leaking sensitive information* **(1.0 P)**.

Total: 12.0pt

Aufgabe 2: Prozesse und Threads

Assignment 2: Processes and Threads

a) Nennen Sie je eine Datenstruktur, die beim Erstellen eines Kernel-Level-Threads im User- bzw. Kernel-Adressraum angelegt wird. **1 pt**

Name one data structure that is allocated in user address space and one that is allocated in kernel address space during the creation of a kernel level thread.

Lösung:

- *User space: (User-)Stack*
- *Kernel space: Thread control block (TCB)*

(0.5 P) *each*

b) Nennen Sie drei Gründe, aus denen ein Prozess terminiert werden kann. **1.5 pt**

Name three reasons for terminating a process.

Lösung:

- *Voluntary termination: The process has finished the task it was started for.*
- *Fatal error: The process has encountered an error or the operating system kills the process because of an exception (e.g., a segmentation fault).*
- *External termination: The process is killed by a signal from another process.*

(0.5 P) *per answer. The explanations are not required.*

c) Erklären Sie den Begriff *Preemption* im Kontext der Vorlesung. Welche Funktion muss die Hardware zur Verfügung stellen, damit Preemption möglich ist? **1.5 pt**

Explain the term preemption *in the context of the lecture. Which feature must the hardware provide for preemption to be possible?*

Lösung:

Preemption means that the operating system can interrupt running threads **(0.5 P)**, *storing their state so their execution can be seamlessly resumed later* **(0.5 P)**. *For preemption to work, the hardware must provide a timer interrupt* **(0.5 P)**.

d) Haben *Daemon-Prozesse* in interaktiven Systemen normalerweise eine hohe oder eine niedrige Schedulingpriorität? Begründen Sie Ihre Antwort. **1.5 pt**

Do daemon processes *in interactive systems typically have a high or low scheduling priority? Justify your answer.*

Lösung:

A low priority **(0.5 P)**. *Daemons are background processes that are not directly visible to the user. The user is less likely to notice slow-running daemons, which is why resources should preferably be allocated to foreground processes that the user interacts with* **(1.0 P)**.

e) Kann im Many-to-One-Threadmodell *Priority Inversion* auftreten? Begründen Sie Ihre Antwort. **1.5 pt**

Can priority inversion *occur in the many-to-one thread model? Justify your answer.*

Lösung:

Yes **(0.5 P)**. *Priority inversion is a scheduling problem and therefore unrelated to the type of thread used* **(1.0 P)**. *If a high-priority thread* ***A*** *waits on a spinlock held by low-priority thread* ***B****, A cannot continue if the scheduler constantly selects threads with a higher priority than B for execution. Whether that scheduler executes in user or kernel space makes no difference.*

f) Welche Daten über einen Zombie-Prozess muss ein POSIX-kompatibles Betriebssystem **mindestens** speichern, bis der Prozess endgültig beendet wird? **1.5 pt**

Which data about a zombie process must a POSIX-compatible operating system ***at least*** *store until that process is terminated for good?*

Lösung:

- ***Exit status****: This is what is collected by* `waitpid()`
- ***Process ID (PID)****: Since the parent process can wait for a specific child, the operating system must know which exit status belongs to which process*
- ***Parent process****: Since processes can only wait for their own children, the operating system must store parent-child relations for zombies to allow for permission checking. This relationship can be stored either in the child (as a link to the parent process) or the parent (as a list of children)*

(0.5 P) *per answer. The explanations are not required.*

g) Macht es Sinn, den virtuellen Speicher im User-Adressraum eines Prozesses freizugeben, wenn dieser Prozess zum Zombie wird? Begründen Sie Ihre Antwort. **1.5 pt**

Does it make sense to free all virtual memory in the user mode address space of a process when that process becomes a zombie? Justify your answer.

Lösung:

Yes **(0.5 P)**. *A process can only become a zombie after it exits. This implies that no activity can take place in the address space of a zombie process, which is why the contents of the address space are not needed any more* **(1.0 P)**.

h) Betrachten Sie ein Betriebssystem, das einen Round-Robin-Scheduler mit fester Zeitscheibenlänge verwendet. Das Betriebssystem versucht, den wichtigen Threads mehr CPU-Zeit zu geben, indem es diese Threads mehrfach in die Warteschlange des Schedulers einfügt. Das System stürzt allerdings oft ab, sobald solch ein Thread blockiert.

Welches Problem ist vermutlich aufgetreten? Schlagen Sie eine Lösung für dieses Problem vor, die die übrigen Eigenschaften des Schedulers erhält. **2 pt**

Consider an operating system using a round-robin scheduler with a fixed timeslice length. This operating system attempts to give more CPU time to important threads by adding these threads to the scheduler's queue multiple times. However, the system often crashes as soon as such a thread blocks.

Which problem probably caused the crash? Suggest a solution for this problem which otherwise preserves the properties of the scheduler.

Lösung:

The problem with this scheme is that if one thread blocks for I/O, there will be more entries for this thread in the scheduler's queue. The scheduler may thus select a thread to run that is not ready to run **(1.0 P)**.

Possible solutions include (others may be acceptable): **(1.0 P)**

- *Whenever a thread blocks, scan the scheduler's queue and remove all entries for this thread.*
- *Store a timeslice length in each thread's TCB, and run each thread for its stored timeslice length once selected. This way, some threads can be granted more time without the need for multiple entries per thread in the scheduler's queue.*
- *Have the scheduler check the thread's state before actually scheduling it, and advance to the next queue entry if the current entry refers to a blocked thread.*

Answers which are ***not*** *acceptable include:*

- *Adding static priorities instead of using a variable timeslice length (introduces the possibility of starvation, which is not possible with the suggested scheduler).*
- *Generally, replacing the scheduler with a completely different one (e.g., MLFQ), as that would fundamentally change the scheduler's properties.*

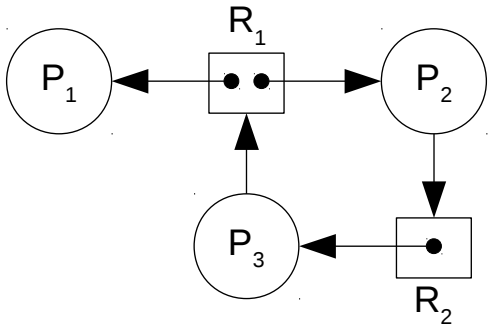
Total: 12.0pt

Aufgabe 3: Koordination und Kommunikation von Prozessen

Assignment 3: Process Coordination and Communication

a) Handelt es sich bei der unten dargestellten Situation um einen Deadlock? Begründen Sie Ihre Antwort. **1 pt**

Is the situation displayed below a deadlock? Give a reason for your answer.



Lösung:

No, there is no deadlock. Because P_1 *has all required resources and can execute* **(0.5 P)**, *it will eventually return its instance of* R_1, *so* P_3 *can also acquire all required resources* **(0.5 P)**.

Warum lässt sich für den gegebenen Resource-Allocation-Graphen kein Wait-For-Graph erstellen? **1 pt**

Why is it not possible to construct a wait-for graph for the given resource allocation graph?

Lösung:

Because there is more than one instance of R_1 **(0.5 P)**, P_3 *waits on either* P_1 *or* P_2, *which cannot be represented in a wait-for graph* **(0.5 P)**.

b) Warum eignen sich Spinlocks nicht für kritische Abschnitte, auf die Threads bei Eintritt oft lange warten müssen (Lock Congestion)? **1 pt**

Why are spinlocks not suitable for critical sections for which threads often have to wait for a long time when trying to enter (lock congestion)?

Lösung:

Spinlocks use busy waiting **(0.5 P)**, *which wastes CPU and blocks other threads from executing useful code* **(0.5 P)**.

Nennen Sie ein Synchronisierungsprimitiv, das dieses Problem nicht hat. **0.5 pt**

Name a synchronization primitive which does not suffer from this problem.

Lösung:

Semaphore or mutex **(0.5 P)**.

c) Die folgende Implementierung eines Synchronisierungsprimitivs weist wartenden Threads eine fortlaufende Nummer zu (wie „Nummern ziehen“ in einer Warteschlange).

Welche der Anforderungen an eine Lösung für das Critical-Section-Problem erfüllt dieser Code? Begründen Sie für alle der Anforderungen, warum sie erfüllt oder nicht erfüllt werden. **6 pt**

The following implementation of a synchronization primitive assigns a consecutive number to waiting threads (similar to the ticket dispenser in a waiting queue).

Which of the desired properties for solutions to the critical section problem are fulfilled by this code? Give reasons for all these properties on why or why not they are fulfilled.

```
1  /* aligned to cache lines */
2  volatile int next = 0;
3  volatile int executing = 0;
4
5  lock_acquire() {
6      /* atomic version of "ticket = next; next += 1;" */
7      int ticket = fetch_and_add(next, 1);
8      /* busy wait until the counter matches */
9      while (ticket != executing) {}
10 }
11
12 lock_release() {
13     executing++;
14 }
```

Lösung:

- *Mutual exclusion is fulfilled* **(0.5 P)***: A thread only enters the critical section when* `executing == ticket`*. As every thread has a different ticket* **(0.5 P)***, and* `executing` *only changes when no thread is in the critical section* **(0.5 P)***, only one thread can ever be in the critical section* **(0.5 P)***.*
  Solutions which argue that 2^{32} *waiting threads cause the counter to overflow are accepted as well.*
- *Progress is fulfilled* **(0.5 P)***: Threads in the remainder section do not block threads from entering the critical section* **(0.5 P)***. When there is no thread in the critical section, then* `executing` *equals one of the tickets of the waiting threads, so that the thread can enter the section* **(1.0 P)***.*
  Note that progress is fulfilled even though the first waiting thread blocks other waiting threads from getting into the critical section until it is scheduled, as any such blocking is temporary (a similar argument could be made for any lock where unrelated threads with higher priority can prevent threads from entering the critical section). Also, the complete definition of progress only includes threads in the remainder section.

- *Bounded waiting is fulfilled* **(0.5 P)***: Threads enter the critical section in the order of their tickets* **(1.0 P)***, so every thread only has to wait until the threads with lower ticket numbers have left the critical section* **(0.5 P)***.*

  Note that bounded waiting is defined in a way that excludes solutions which argue that threads can be terminated while waiting for the lock or while holding the lock.

In all cases, correct usage of the functions has to be assumed. Otherwise, traditional spinlocks would not guarantee either mutual exclusion or progress.

Warum ist es nicht notwendig, in Zeile 13 eine atomare Instruktion für das Inkrementieren des Zählers zu verwenden? **1 pt**

Why is it not necessary to use an atomic instruction in line 13 to increment the counter?

Lösung:

`lock_release()` *is only called by the thread which has acquired the critical section* **(0.5 P)***. Other threads can only acquire the critical section after the value has been incremented* **(0.5 P)***, therefore no races are possible.*

d) Gegeben sei ein System, das Message-Passing mit asynchronem Senden und synchronem Empfangen unterstützt. Erklären Sie kurz, wie man darauf aufbauend ohne Modifikation des Kernels Message-Passing mit synchronem Senden und synchronem Empfangen implementieren kann. **1.5 pt**

Assume a system which supports message passing with non-blocking send and blocking receive operations. Explain briefly how it is possible to implement message passing with blocking send and blocking receive on top of this system, without modifying the kernel.

Lösung:

After the non-blocking send, the sender executes a blocking receive **(0.5 P)***. Upon receiving a message, the receiver sends an acknowledgment message* **(0.5 P)***. The sender blocks until that acknowledgment has been received* **(0.5 P)***.* **(-0.5 P)** *for solutions which perform the handshake before sending the actual data, due to a race condition when multiple processes try to send data.*

Solutions which use a zero-length buffer to force the sender to block are incorrect because asynchronous sending always copies into the buffer, therefore the corresponding system call will always fail. **Total: 12.0pt**

Aufgabe 4: Speicher

Assignment 4: Memory

a) Wann erfährt in Linux ein Prozess einen *Segmentation Fault*, unter Windows auch als *Access Violation* bekannt? **1 pt**

When does a process in Linux experiences a segmentation fault, *also known as* access violation *in Windows?*

Lösung:

A segmentation fault or access violation is an exception that is raised by the operating system if a process attempts to access a virtual address that is either not assigned or for which the process does not posses the required access permissions **(1.0 P)**.

b) Welche Operation bestimmt zu einem wesentlichen Teil die Geschwindigkeit des `fork()`-Systemaufrufs? **0.5 pt**

Which operation mainly determines the speed of the `fork()` *system call?*

Lösung:

Copying the process's address space **(0.5 P)**.

Nennen und erläutern Sie eine Technik, die auf modernen Systemen eingesetzt wird, um die Geschwindigkeit von `fork()` deutlich zu erhöhen. **2 pt**

Give and explain a technique that is used on modern systems to considerably increase the performance of `fork()`.

Lösung:

Copy-on-write **(0.5 P)** *creates page tables that share the page frames between the parent and child processes, without having to copy the data itself* **(0.5 P)**. *To maintain isolation the pages are marked read-only in both processes* **(0.5 P)**. *When either process attempts to modify a page, the sharing is broken and a private writable copy of the page frame is created* **(0.5 P)**.

c) Nennen Sie den in der Vorlesung vorgestellten Seitenersetzungsalgorithmus, der sich die temporale Lokalität von Seitenzugriffen zunutze macht. Wieso wird dieser in der Praxis lediglich approximiert? **1.5 pt**

Name the page replacement algorithm presented in the lecture which makes use of the temporal locality of page accesses. Why is it only approximated in practice?

Lösung:

Least Recently Used (LRU) **(0.5 P)**. *LRU is usually approximated, because the page tables are missing timestamps that are updated on each access* **(1.0 P)**.

d) Betrachten Sie ein System, das mittels einer hierarchischen Seitentabelle virtuelle in physische Speicheradressen übersetzt. Der virtuelle Adressraum umfasst 512 GiB, die Seitengröße ist 4 KiB, eine Seitentabelle beinhaltet 512 Einträge.

Berechnen Sie die Anzahl der Stufen der Seitentabellenhierarchie. **2 pt**

Consider a system that translates virtual addresses to physical addresses using a hierarchical page table. The virtual address space is 512 GiB in size, the page size is 4 KiB, a page table contains 512 entries.

Calculate the number of levels in the page table hierarchy.

Lösung:

Each page is $4\,KiB = 2^{12}\,bytes$*. The address space is* $512\,GiB = 2^9 * 2^{30}\,bytes = 2^{39}\,bytes$*. The number of total pages is thus* $\frac{2^{39}\,bytes}{2^{12}\,bytes} = 2^{27}$ **(1.0 P)**.

Each level of the page table hierarchy contains $512 = 2^9$ *entries. This means we need three levels, as* $\frac{27}{9} = 3$ **(1.0 P)**.

Ein Programm kopiert einen 4 MiB Puffer im virtuellen Adressraum. Quell- und Zielpuffer überlappen sich nicht und sind an Seitengrenzen ausgerichtet. Die CPU verfügt über keinen Cache, jedoch einen leeren TLB. Die Wortbreite beträgt 8 Bytes.

Wie viele Speicherzugriffe sind für den Kopiervorgang mindestens nötig? **3 pt**

A program copies a 4 MiB buffer in the virtual address space. The source- and destination buffers do not overlap and are page aligned. The CPU does not have a cache, but has an empty TLB. The word size is 8 bytes.

How many memory accesses are at least required for the copy operation?

Lösung:

The buffer size is $4\,MiB = 2^2 * 2^{20}\,bytes = 2^{22}\,bytes$*. To carry out the copy operation we have to read the entire buffer. This requires* $\frac{2^{22}\,bytes}{2^3\,bytes} = 2^{19}$ *accesses* **(1.0 P)**.

For each of the $\frac{2^{22}\,bytes}{2^{12}\,bytes} = 2^{10}$ *pages* **(0.5 P)**, *we have to perform one address translation. Each address translation requires one access per page table level, i.e.* 3*. We thus have a total of* $2^{10} * 3$ *extra read operations* **(0.5 P)**.

Since we have to copy the buffer the same amount of work has to be done when writing the data to the destination **(0.5 P)**.

In total this gives: $2 * (2^{19} + 2^{10} * 3) = \boxed{2^{20} + 2^{11} * 3}$ *accesses* **(0.5 P)**.

Das System wird mit einem Cache erweitert. Nennen Sie zwei Gründe, weshalb die Kopiergeschwindigkeit steigt. **2 pt**

The system is extended with a cache. Give two reasons why the copy performance increases.

Lösung:

- *A cache usually consists of cache lines that are larger than a single CPU word (e.g., 64 bytes). Thus when reading the buffer, RAM accesses occur with larger width, which increases the memory access bandwidth.*
- *Having a cache allows the CPU to prefetch (read-ahead) data from the source.*
- *When writing to the destination the cache buffers the write accesses, thus reducing the latency for each write operation. Just like when reading, the memory access bandwidth increases.*

(1.0 P) *for each reason.* **Total: 12.0pt**

Aufgabe 5: I/O, Hintergrundspeicher und Dateisysteme

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

a) Beschreiben Sie RAID 4. Wie werden die Nutzdaten auf den Festplatten verteilt? Welche Art von Redundanz wird verwendet, und wo wird sie gespeichert? **2 pt**

Describe RAID 4. How is the data distributed among the disks? What type of redundancy is used, and where is it stored?

Lösung:

RAID 4 has multiple data disks, and uses block-level **(0.5 P)** *striping* **(0.5 P)** *of the data. A separate disk* **(0.5 P)** *holds the parity* **(0.5 P)** *of the data disks.*

Warum kann RAID 4 zum vorzeitigen Ausfall einer der Festplatten führen? **1 pt**

Why can RAID 4 cause premature failure of one of the hard disks?

Lösung:

The parity disk is accessed on every write access **(0.5 P)**. *It is therefore accessed more often than the data disks, leading to increased wear* **(0.5 P)**.

b) Warum ist der Flash-Speicher in einer SSD meist größer als die Kapazität der SSD, die dem Betriebssystem zur Verfügung steht? **2 pt**

Why is the amount of flash memory in a SSD usually larger than the capacity available to the OS?

Lösung:

The additional memory is used as spare blocks **(0.5 P)**. *Modified block contents are written to those spare blocks* **(0.5 P)**.

This mechanism has two effects (describing one is enough):

- *Updating data requires erasing the flash memory first, which is expensive* **(0.5 P)**. *If spare blocks are used for new data, erasing can take place in the background, which increases the performance of the SSD* **(0.5 P)**.
- *The number of erase cycles per block is limited* **(0.5 P)**. *By spreading accesses over a larger number of blocks, the durability of the SSD is increased* **(0.5 P)**.

c) Ihr aktuelles Arbeitsverzeichnis ist `/a/b/c/`. Geben Sie zwei unterschiedliche relative Pfade für die Datei `/a/d/e` an. **1 pt**

Your current working directory is `/a/b/c/`. *Give two different relative paths for the file* `/a/d/e`.

Lösung:

(0.5 P) *per correct relative path. Correct paths include* `../../d/e`, `../../../a/d/e`, `../c/../../d/e`, `../../d/./e`

d) Dateisystem-Caches verwenden die Write-Back-Strategie, um die Anzahl der Festplattenzugriffe zu reduzieren. Warum sollte hierbei sichergestellt werden, dass veränderte Daten periodisch auf die Festplatte geschrieben werden? **1 pt**

File system caches use the write-back strategy to reduce the number of hard disk accesses. Why should the OS ensure that modified data is periodically written back to the disk?

Lösung:

The longer modified data is only held in main memory, the higher is the risk of data loss due to a system crash. If modified data is periodically written to disk, the amount of lost data is reduced **(1.0 P)**.

e) Welchen Vorteil bietet eine Bitmap zur Verwaltung freier Blöcke im Vergleich zu einer verketteten Liste? **1 pt**

What advantage does a bitmap have over a linked list when used to track free blocks?

Lösung:

Bitmaps provide significantly faster random access **(0.5 P)**, *which improves the performance when searching for multiple contiguous free blocks* **(0.5 P)** *in order to reduce fragmentation.*

f) Beschreiben Sie, wie sich die Liste der freien Blöcke aus sonstigen Metadaten des Dateisystems wiederherstellen lässt, falls sie beschädigt wird. **1 pt**

Describe how the list of free blocks can be recovered from other metadata of the file system if it is damaged.

Lösung:

Metadata such as i-nodes or file allocation tables hold the blocks used by files **(0.5 P)**. *The list of free blocks can be constructed as the list of blocks which neither contain file contents nor the metadata itself* **(0.5 P)**.

g) Gegeben sei ein Dateisystem, das die Inhalte kleiner Dateien immer direkt im Verzeichniseintrag der Dateien speichert. Wie wirkt sich dies auf die Geschwindigkeit von Dateizugriffen aus? Begründen Sie Ihre Antwort. **1 pt**

Assume a file system which stores the contents of small files always directly in the directory entries of the files. How does this method affect the performance of file accesses? Justify your answer.

Lösung:

Accessing the contents of the files is significantly faster **(0.5 P)** *as no additional disk seek operation is required to find the file contents* **(0.5 P)**.

Note that disk performance is mainly dominated by seek times, thus answers which only argue that scans through the directory become slower are not sufficient.

Lassen sich in diesem Dateisystem Hardlinks implementieren? Begründen Sie Ihre Antwort. **1 pt**

Is it possible to implement hardlinks in this file system? Justify your answer.

Lösung:

No **(0.5 P)**. *Because hardlinks are additional references to the same blocks on the disks, implementing hardlinks requires an additional level of indirection to let multiple directory entries point to the same file contents* **(0.5 P)**.

Alternative: Yes, by converting small files to use the conventional i-node structure as soon as they are referenced by a second hardlink **(1.0 P)**.

Lassen sich in diesem Dateisystem Softlinks implementieren? Begründen Sie Ihre Antwort. **1 pt**

Is it possible to implement softlinks in this file system? Justify your answer.

Lösung:

Yes **(0.5 P)**. *Softlinks contain a path which references another directory entry, therefore separate i-nodes are not required* **(0.5 P)**. **Total: 12.0pt**