Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

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

Musterlösung Nachklausur
08.09.2022

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 22 Blättern: Einem Deckblatt, 21 Aufgabenblättern mit insgesamt 5 Theorieaufgaben und 3 Programmieraufgaben sowie 0 Seiten Man-Pages.

  The examination consists of 22 pages: One cover sheet, 21 sheets containing 5 theory assignments as well as 3 programming assignments, and 0 sheets with man pages.

- Es sind keinerlei Hilfsmittel erlaubt!

  No additional material is allowed!

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

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

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

  You can use the back side of the assignment sheets for your answers. If you need additional draft paper, please notify one of the supervisors.

- Bitte machen Sie eindeutig klar, was Ihre endgültige Lösung zu den jeweiligen Teilaufgaben ist. Teilaufgaben mit mehreren widersprüchlichen Lösungen werden mit 0 Punkten bewertet.

  Make sure to clearly mark your final solution to each question. Questions with multiple, contradicting answers are void (0 points).

- Programmieraufgaben sind gemäß der Vorlesung in C zu lösen.

  Programming assignments have to be solved in C.

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

Aufgabe	T1	T2	T3	T4	T5	P1	P2	P3	Total
Max. Punkte	9	9	9	9	9	15	15	15	90
Erreichte Punkte									

Aufgabe T1: Grundlagen

Assignment T1: Basics

a) Welchen Vorteil bietet die Trennung zwischen Mechanismus und Policy? **1 pt**

Which advantage does the separation between mechanism and policy provide?

Solution:

More flexibility: Replacing the policy does not require rewriting the mechanism.

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

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

Solution:

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.*

c) Erläutern Sie einen Vorteil und einen Nachteil von statischem gegenüber dynamischem Linking von Bibliotheken. **1 pt**

Explain an advantage and a disadvantage of static versus dynamic linking of libraries.

Solution:

+ *No external dependencies (no DLL hell) as all libraries are linked to a single executable*
+ *Link time optimization (LTO) possible as libraries are available at link time of executable*
+ *Potentially smaller memory footprint (at least virtual memory), because only required functionality is included*
+ *Smaller load time, because (a) see previous point, and (b) no dependency resolution and dynamic linking*
- *Higher physical memory consumption due to missing sharing opportunities*
- *Larger executable distribution and storage size (see previous point)*
- *Higher security risks, as all executables that share a vulnerable library have to be relinked and updated*

d) Geben Sie ein Segment einer ELF-Datei an, das zwischen Prozessen geteilt werden kann, und ein anderes Segment, das nicht geteilt werden kann. Begründen Sie, warum das jeweilige Segment geteilt bzw. nicht geteilt werden kann. **1.5 pt**

Name a segment of an ELF file that can be shared between processes, and another segment that cannot be shared. Give reasons why each segment can be shared or cannot be shared.

Solution:

Generally, ELF segments can be shared if they are read-only and thus do not receive process private data **(1 P)**. *Accordingly,* `.text` *and* `.rodata` **(0.5 P)** *can be shared, whereas* `.data` *and* `.bss` **(0.5 P)** *cannot be shared.*

e) Nennen Sie zwei Möglichkeiten, die Parameter eines Systemaufrufs an den Kernel zu übergeben. Welche dieser Möglichkeiten würden Sie wählen? Begründen Sie Ihre Antwort. **2 pt**

Name two ways of passing the parameters of a system call to the kernel. Which of these ways would you prefer? Explain your answer.

Solution:

Parameters can be passed either in registers **(0.5 P)** *or on the stack.* **(0.5 P)** *Passing them in registers is preferable,* **(0.5 P)** *because registers can be accessed faster than memory, which may improve system call performance* **(0.5 P)**.

f) Wie bemerkt das Betriebssystem, dass ein nicht privilegierter Prozess versucht hat, eine privilegierte Instruktion auszuführen? **0.5 pt**

How does the operating system know that a non-privileged process has tried to execute a privileged instruction?

Solution:

The CPU generates an exception in that case.

g) Wofür stehen die folgenden Abkürzungen im Kontext der Vorlesung? **1 pt**

What do the following abbreviations stand for in the context of the lecture?

Solution:

TCB: Thread Control Block **(0.5 P)**
IPC: Interprocess Communication **(0.5 P)**

Total: 9.0pt

Aufgabe T2: Prozesse und Threads

Assignment T2: Processes and Threads

a) Nennen Sie vier in der Vorlesung besprochene Bestandteile eines Prozesskontrollblocks. **2 pt**

Name four elements of a Process Control Block discussed in the lecture.

Solution:

- *process state*
- *process id*
- *program counter*
- *CPU registers*
- *scheduling information*
- *memory management information*
- *list of open files*
- *...*

b) Gegeben seien ein Lottery-Scheduler auf einem Uniprozessorsystem mit der Prozessliste $\{P_1, P_2, P_3\}$. P_1 habe 2 Tickets, P_2 habe 5 Tickets, P_3 habe 1 Ticket. Alle Prozesse sind permanent lauffähig. **3 pt**

Vervollständigen Sie den untenstehenden Plan (10 Kreuze). Die erste Zeile (TIME) gibt die Zeit an. Eingeplant wird jeweils beim Wechsel der Zeiteinheit. Um die Lotterie deterministisch zu gestalten, enthält die zweite Zeile (LOTTERY) das vorgegebene Lotterie-Ergebnis zum jeweiligen Anfang der Zeiteinheit.

Consider a lottery scheduler on a uniprocessor system with process list $\{P_1, P_2, P_3\}$. P_1 has 2 lottery tickets, P_2 has 5 lottery tickets, and P_3 has 1 ticket.

Complete the scheduling plan given below (10 crosses). The first row (TIME) states the time. Scheduling occurs at time unit changes. In order to ensure deterministic results, the second row (LOTTERY) contains a pre-defined lottery result for each time unit start.

Solution:

TIME	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
LOTTERY	*4*	*5*	*0*	*0*	*1*	*3*	*2*	*4*	*1*	*6*
P_1			**X**	**X**	**X**				**X**	
P_2	**X**	**X**				**X**	**X**	**X**		**X**
P_3										

c) Erklären Sie, warum ein *FCFS-Scheduler* eine schlechte Wahl für Desktopsysteme ist. **1 pt**

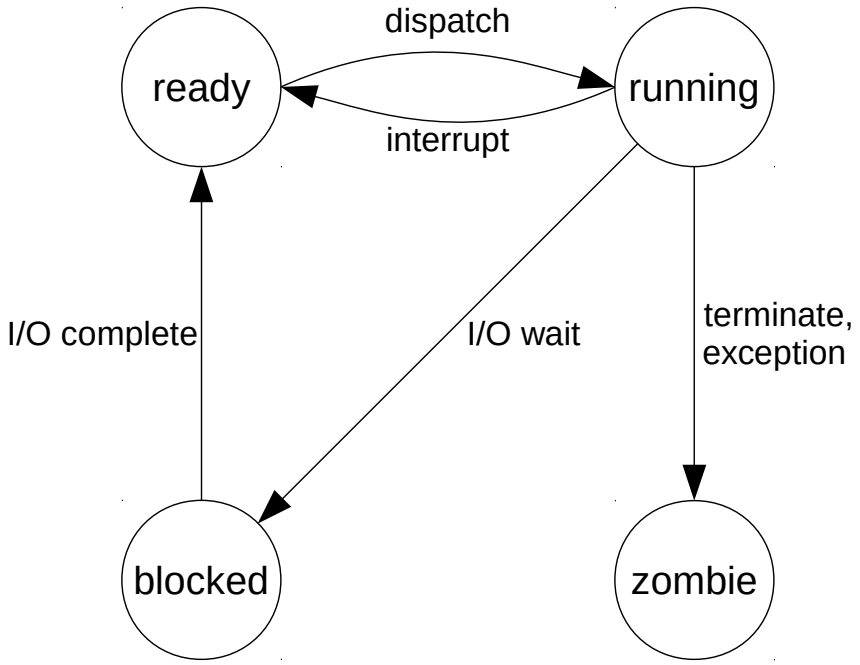
Explain why a FCFS-scheduler is a bad choice for desktop systems.

Solution:

FCFS does not preempt running processes and therefore does not provide good interactivity, which is important for desktop systems. If, for example, a single process runs on the CPU for an extended period of time, other processes, including the desktop environment, are never dispatched.

d) Nehmen Sie an, dass ein Betriebssystem die vier Prozesszustände „bereit“ („ready“), „rechnend“ („running“), „blockiert“ („blocked“) und „zombie“ unterstützt. Stellen Sie grafisch dar, zwischen welchen Zuständen Übergänge möglich sind und beschriften Sie jede Kante mit einem Ereignis, das den jeweiligen Übergang auslöst. **2.5 pt**

Consider an operating system, which supports the four process states “ready”, “running”, “blocked”, and “zombie”. Depict the possible state transitions and label each edge with an event causing the transition.



(0.5 P) per edge if labeled correctly **(-0.5 P)** per incorrect edge.

Durch welche Aktion verlässt ein Prozess den „Zombie“-Zustand? **0.5 pt**

Which action causes a process to leave the “zombie” state?

Solution:

A process leaves the “zombie” state once its parent process reads its state with `wait()`.

Total: 9.0pt

Aufgabe T3: Speicher

Assignment T3: Memory

a) Erläutern Sie, für welche Szenarien die folgenden Seitentabellentypen geeignet sind. Begründen Sie Ihre Antwort. **2 pt**

Explain for which scenarios the following page table types are suitable. Justify your answer.

Solution:

Linear page table *A linear page table is particularly suited for systems with small or mostly filled virtual address spaces* **(0.5 P)**. *In these cases, there is no benefit in using multiple levels for translation due to the small size and missing sparseness. Instead, the additional levels only increase translation costs and waste memory* **(0.5 P)**.

Inverted page table *An inverted page table makes sense if the physical address space is considerably smaller than the virtual address space. This way, less memory is wasted for page tables* **(0.5 P)**.

Alternative: An inverted table is also useful as an extension to a regular forward page table **(0.5 P)** *to enable fast reverse mapping (e.g., to find virtual mappings of a PFN)* **(0.5 P)**.

b) Ein Prozess ruft `fork()` auf. Beschreiben Sie für den Eltern- und den Kindprozess, wie der jeweilige Adressraum konfiguriert werden muss, um Copy-On-Write (COW) zu ermöglichen. Begründen Sie Ihre Antwort. **1.5 pt**

A process calls `fork()`*. Describe for the parent and the child process, how each address space has to be configured to allow copy-on-write (COW). Explain your answer.*

Solution:

Both, the parent and the child processes, map to the same page frames. Therefore, both address spaces need to be write-protected **(0.5 P)**. *This way, a write access triggers a page-fault and the operating system is invoked, which can create a private copy* **(1.0 P)**.

c) Erläutern Sie knapp den Begriff *ASID*. **0.5 pt**

Briefly explain the term ASID*.*

Solution:

ASID is the abbreviation for address space identifier and it associates a TLB entry with an address space to avoid having to flush all TLB entries on a context switch **(0.5 P)**.

d) Gegeben sei ein System, das virtuelle Adressen mit einer vierstufigen Seitentabelle übersetzt. Die Tabelle nutzt 32 von 1024 Einträgen in der obersten Stufe und alle 1024 Einträge in den folgenden Stufen. Die Seitengröße beträgt 4 KiB. Jeder Tabelleneintrag ist 4 Bytes groß. **2 pt**

Bei gleichem Speicherverbrauch für die Seitentabelle, wieviel größer wäre der maximal adressierbare virtuelle Speicher in Bytes bei einer linearen Seitentabelle, wenn alle Einträge genutzt werden?

Consider a system which translates virtual addresses with a four-level page table. The table uses 32 of 1024 entries in the top-most level and all 1024 entries in each following level. The page size is 4 KiB. Each table entry is 4 bytes in size.

With the same memory consumption for the page table, how much larger would the maximum addressable virtual memory in bytes be for a linear page table, if all entries are used?

Solution:

The fourth level in the hierarchical page table is also required in the linear page table **(0.5 P)**. *We thus only consider the higher levels. This gives us a memory consumption of* 1 *page for the top-most level,* 2^5 *pages for the second level, and* $2^5 * 2^{10}$ *pages for the third level* **(0.5 P)**. *In a linear page table each of these pages could be used to address additional 1024 pages of 4 KiB* **(0.5 P)**.

This gives us a total of $(2^0 + 2^5 + 2^{15}) \cdot 2^{10} \cdot 2^{12}$ *bytes* **(0.5 P)**.

e) Worin unterscheidet sich ein software- von einem hardwaregesteuerten TLB? **1 pt**

What is the difference between a software- and a hardware-managed TLB?

Solution:

The software-managed TLB does not walk the page table in memory on a TLB miss **(0.5 P)**. *Instead, the OS has to resolve the TLB miss and manually add the TLB entry* **(0.5 P)**.

f) Erklären Sie die Begriffe Demand Paging und Pre-Paging. Bei welchem Verfahren ist mit mehr Speichernutzung zu rechnen? Warum kann dieses Verfahren dennoch von Vorteil sein? **2 pt**

Explain the terms demand paging and pre-paging. Which method is expected to use more memory? Why can this method nevertheless be advantaguous?

Solution:

With demand paging, pages are loaded into the address space only on their first access **(0.5 P)**. *In contrast, pre-paging loads all contents into memory at process start* **(0.5 P)**. *Consequently, with pre-paging the probability is higher that more memory is used* **(0.5 P)**. *Nevertheless, the method avoids page faults at runtime and can thus be faster* **(0.5 P)**.

Total: 9.0pt

Aufgabe T4: Koordination und Kommunikation von Prozessen

Assignment T4: Process Coordination and Communication

a) Erläutern Sie, was man unter einer *Race Condition* versteht. **1 pt**

Explain what is meant by a race condition.

Solution:

Concurrent access to shared data by different threads **(0.5 P)** *can lead to wrong results.* **(0.5 P)**

Alternatively: Computed results depend on the execution order of involved concurrently running threads or processes. **(1 P)**

b) Unter welchen Voraussetzungen funktioniert das Deaktivieren von Interrupts als Synchronisationsmechanismus? Begründen Sie Ihre Antwort. **1 pt**

In which circumstances may disabling interrupts work as a synchronization mechanism? Explain why.

Solution:

You may implement synchronization by disabling interrupts only on single processor systems **(0.5 P)**. *On a multi-processor system, disabling interrupts (even globally) will not keep other CPUs from entering a critical section and therefore cannot prevent race conditions* **(0.5 P)**.

c) Welche der Bedingungen für einen Deadlock wird nicht erfüllt, wenn sämtliche Ressourcen sortiert und ausschließlich in dieser Reihenfolge alloziert werden? Begründen Sie Ihre Antwort. **1 pt**

Which of the requirements for a deadlock is negated by ordering all resources and always acquiring them in this order? Justify your answer.

Solution:

This breaks circular wait **(0.5 P)**, *because a cycle in the resource acquisition graph would require at least one process to acquire a resource with a lower order than the resource it already holds* **(0.5 P)**.

d) Warum muss beim Reader-Writer-Problem der Eintritt von schreibenden Threads in den kritischen Abschnitt gegenüber lesenden Threads priorisiert werden, wenn Bounded Waiting erfüllt werden soll? **1 pt**

Why does the reader-writer problem require the entry of writer threads into the critical section to be prioritized over the entry of reader threads if bounded waiting shall be fulfilled?

Solution:

If reader threads are admitted into the critical section whenever they could enter it, a large number of reader threads can starve writer threads. If additional reader threads arrive at the section before the previous reader threads have left the section, there always is a reader thread in the section, so writer threads can never get in **(1 P)**.

e) Welche zwei weiteren Bedingungen neben Bounded Waiting sind für die gültige Lösung des Problems kritischer Abschnitte notwendig? Erläutern Sie diese kurz. **2 pt**

Besides bounded waiting, what two other requirements must be met for a valid solution to the critical section problem? Explain them briefly.

Solution:

Progress (0.5 P): *If no process is in the CS, one of the processes trying to enter will eventually get in. Processes that are not trying to enter do not hinder processes that try to enter from getting in* **(0.5 P)**.

Mutual Exclusion (0.5 P): *Only one thread can be in the CS at any time* **(0.5 P)**.

f) Erklaren Sie den Begriff *Mailbox* im Kontext von Message Passing. **1 pt**

Explain the term mailbox *in the context of message passing.*

Solution:

A mailbox is a indirection mechanism: With mailboxes, communicating processes do not name the communication partner directly but rather deposit messages to a mailbox or receive them from the mailbox **(1 P)**.

Note that the mailbox is not a buffer, but only an addressing mechanism. There are, for example, mailbox implementations with synchronous unbuffered message passing such as L4Re IPC gates.

g) Erklären Sie, wie ein *Two-Phase-Lock* funktioniert. Ist die Verwendung auch im Kernel sinnvoll? **2 pt**

Explain how a two-phase lock *works. Does it make sense to use it also in the kernel?*

Solution:

The lock first spins for a fixed amount of time **(0.5 P)**. *If the lock cannot be acquired within this time, the thread executes a syscall to sleep until the lock is available* **(0.5 P)**.

Yes, if the cost of performing the blocking wait (i.e., including a context switch) is generally larger than the time spent spinning **(1.0 P)**. *However, since no syscalls are involved, the cost for blocking is lower in the kernel, which will reduce the appropriate spinning time.*

Total: 9.0pt

Aufgabe T5: I/O, Hintergrundspeicher und Dateisysteme

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

a) Wie ist es möglich, einem einzelnen Benutzer außerhalb der eigenen Benutzergruppe Zugriff auf ein Verzeichnis zu geben, ohne die Gruppe des Benutzers zu ändern? **1 pt**

How is it possible to give a single user outside your user group access to a directory without changing the user's group?

Solution:

ACLs **(1 P)** *can be used to grant access to only a single user.*

b) Wenn in einem Unix Dateisystem ein symbolischer Link gelöscht wird, dann muss die zu der verlinkten Datei gehörige inode nicht modifiziert werden. Bei harten Links ist dies allerdings der Fall. Warum? **2 pt**

When a symbolic link is deleted in a Unix file system, the inode of the linked file does not need to be modified. This is not true for hard links. Why is this the case?

Solution:

A hard link is a reference to an inode. **(0.5 P)** *Symbolic links on the other hand are special files that refer to another file by its path, which may not exist.* **(0.5 P)** *If there is no more hard link pointing to an inode, the inode needs to be deleted. Thus, we need to keep track of the number of hard links with a reference count.* **(1 P)**

c) Ein Verbund aus vier Festplatten kann für unterschiedliche RAID Level konfiguriert werden. Wie viele Festplatten können in der jeweiligen Konfiguration ausfallen, bevor Daten endgültig verloren gehen? Begründen Sie jeweils Ihre Antwort. **4 pt**

An array of four hard disks can be configured for different RAID levels. How many disks can fail in each configuration before data is permanently lost? Explain your answer.

Solution:

RAID 0: 0 **(0.5 P)**, *striping, no redundancy* **(0.5 P)**.
RAID 10: 1 or 2 **(0.5 P)**, *can handle two failing drives if they are in separate RAID 1 groups* **(0.5 P)**.
RAID 4: 1 **(0.5 P)**, *data is stored on three disks, parity information is stored on a separate disk, thus reconstruction is only possible if one disk fails* **(0.5 P)**.
RAID 5: 1 **(0.5 P)**, *same as RAID 4, but parity information is distributed across all disks* **(0.5 P)**.

d) Mit dem Programm `fsck` können unter Unix Dateisystemfehler gefunden und behoben werden. Geben Sie für die folgenden Fehlermeldungen an, was `fsck` im inkonsistenten Dateisystem gesehen haben könnte um auf den Fehler zu schließen. **2 pt**

Unix file systems can be checked and fixed with the `fsck` program. What could `fsck` have seen to generate the following error messages in an inconsistent file system.

Der Inode Linkzähler ist 1, er sollte 2 sein.
The inode link count is 1, it should be 2.

Solution:

A second directory entry has been found to an inode with link count 1.

Der Frei-Bitmap Eintrag für Block 1234 ist 0 (frei), er sollte 1 sein (zugewiesen).
The free-bitmap entry for block 1234 is 0 (free), it should be 1 (allocated).

Solution:

An inode entry points to a block which is not marked to be allocated.

Total: 9.0pt

Aufgabe P1: C Grundlagen

Assignment P1: C Basics

a) In dem untenstehenden Code haben sich 6 Fehler eingeschlichen. Markieren Sie die fehlerhaften Zeilen mit einem X und korrigieren Sie den Code. Gehen Sie von einem 64-Bit-System aus. **6 pt**

There are 6 errors in the code below. Mark the incorrect lines with an X and correct the code. Assume a 64-bit system.

```
typedef struct {
    int valid;
    char *key;
    int value;
} element;
```

```
typedef struct {
    element *elements;
    size_t capacity;
} hashtable;
```

Solution:

```
uint64_t hashString(char *c) {
    uint64_t hash = 5381;
    while (*c) /* missing * */
        hash = hash * 33 + *c++;
    return hash;
}
int insert(hashtable *table, char *key, int value) {
    size_t start = hashString(key) % table->capacity;
    size_t pos = start;
    while (table->elements[pos].valid) {
        if (strcmp(key, table->elements[pos].key) == 0) /* key dereference */
            break; /* break instead of continue */
        pos = (pos + 1) % table->capacity;
        if (pos == start)
            return 0;
    }
    table->elements[pos] = (element) {1, key, value}; /* wrong order */
    return 1;
}
int find(hashtable *table, char *key, int *value) {
    size_t start = hashString(key) % table->capacity;
    size_t pos = start;
    while (table->elements[pos].valid) {
        if (strcmp(key, table->elements[pos].key) == 0) { /* missing == 0 */
            *value = table->elements[pos].value; /* missing * */
            return 1;
        }
        pos = (pos + 1) % table->capacity;
        if (pos == start)
            break;
    }
    return 0;
}
```

Wie löst die Hashtabelle Kollisionen auf? **0.5 pt**

How does the hash table resolve collisions?

Solution:

The hash table uses linear probing *to resolve collisions.*

Schreiben Sie eine Funktion `allocHashtable()`, die eine leere, auf dem Heap allozierte Hashtabelle mit der gegebenen Kapazität zurückgibt. Bei einem Fehler soll die Funktion `NULL` zurückgeben. **3 pt**

Write a function `allocHashtable()` which shall return an empty hash table allocated on the heap with the given capacity. In case of an error, the function shall return `NULL`.

Solution:

```
hashtable *allocHashtable(size_t cap) {
  hashtable *table = malloc(sizeof(hashtable));
  if (table == NULL) return NULL;
  table->capacity = cap;
  table->elements = calloc(cap, sizeof(elements));
  if (table->elements == NULL) {
    free(table);
    return NULL;
  }
  return table;
}
```

(0.5 P) *malloc for* `table`, **(0.5 P)** *error check for* `table`, **(0.5 P)** *setting* `capacity` *and* `elements`, **(0.5 P)** *calloc or malloc+memset for* `elements`, **(1 P)** *error check for* `elements`

Wie kann durch Anpassung von `element` Speicher eingespart werden? Wie viel Speicher kann pro Eintrag eingespart werden? Erklären Sie. **1.5 pt**

How can we save memory by changing `element`? How much memory can we save per entry? Explain.

Solution:

Assuming an `int` size of 4 bytes, there is padding after `valid` and after `value` to ensure alignment of the 64 bit pointer `key`. **(0.5 P)** *By moving `valid` after `key`* **(0.5 P)**, *we can save 8 bytes per element* **(0.5 P)**.

Note: Changing `valid` to a smaller data type will just increase the amount of padding.

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

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

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

uint16_t b = a[0] + a[2];
uint32_t c = b << 8;
// <-- Evaluate expression here
```

Solution:

Expression	*Value*
`a[4] | a[1]`	*0xff55ff77*
`~a[5] >> 16`	*0xfedc*
`b ^ ~b`	*0xffff*
`c`	*0xccee00*

c) `A` sei ein Array. Was bedeutet der folgende Ausdruck? **1 pt**

Let `A` *be an array. What is the meaning of the following expression?*

```
&A[42]
```

Solution:

The statement refers to the memory address **(0.5 P)** *of the 43rd element* **(0.5 P)** *of A.*

Schreiben Sie einen äquvalenten Ausdruck, der ohne den `&`-Operator auskommt. **0.5 pt**

Write an equivalent expression without the `&` *operator.*

Solution:

```
A + 42
```

d) Wie werden in der cdecl-Aufrufkonvention Funktionsargumente übergeben? **0.5 pt**

In the cdecl calling convention, how are arguments passed to functions?

Solution:

Arguments are passed on the stack.

Total: 15.0pt

Aufgabe P2: Dateikompression

Assignment P2: File Compression

Schreiben Sie ein Programm zur Kompression von Dateien. Die Pfade der Eingabedateien sowie die entsprechenden Ausgabepfade für die komprimierten Daten sollen als Paare über Kommandozeilenargumente übergeben werden. Ein Aufruf des Programms könnte wie folgt aussehen:

```
compress a.txt a.txt.out b.txt b.txt.out
```

Das Programm lädt dabei die Inhalte der Dateien `a.txt` und `b.txt` in den Hauptspeicher, komprimiert diese und schreibt sie in die jeweiligen Ausgabedateien (z. B. `a.txt` → `a.txt.out`).

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

Write a program that compresses files. The paths to input files as well as the corresponding output paths for the compressed data are passed to the program as pairs of command line arguments. As an example, consider the following command line:

```
compress a.txt a.txt.out b.txt b.txt.out
```

The program loads the contents of the files `a.txt` and `b.txt` into memory, compresses them, and writes them into the corresponding output files (e.g., `a.txt` → `a.txt.out`).

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

```c
/* include statements for the required C headers */
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h> /* not requested */

/* global variables */
int parallel_processes = 0;
```

a) Vervollständigen Sie die Funktion `get_file_size()`, die die Größe der Datei an dem angegebenen Pfad zurückgibt. **2 pt**

Complete the function `get_file_size()`, which returns the size of the file at the specified path.

Solution:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
size_t get_file_size(const char *path) {
    struct stat info;
    stat(path, &info);
    return info.st_size;
}
```

Adding necessary includes **(0.5 P)**, *Calling `stat` on the file* **(1 P)**, *Returning `st_size`* **(0.5 P)**.

b) Vervollständigen Sie die Funktion `read_file()`, die die Datei an dem gegebenen Pfad `path` öffnet und die ersten `buf_len` Bytes der Datei mithilfe des `read()` Systemaufrufs in den Puffer `buf` liest. Die Funktion gibt in jedem Fall die Anzahl tatsächlich gelesener Bytes zurück. **6.5 pt**

- Schreiben Sie nicht über das Ende des Puffers hinaus (`buf_len`).
- Implementieren Sie Fehlerbehandlung für sämtliche Systemaufrufe. Die Funktion soll auch dann `buf_len` Bytes einlesen, wenn Signale während der Ausführung der Funktion auftreten. Sonstige Fehler (inkl. Erreichen des Dateiendes) sollen zum Abbruch des Lesevorgangs führen.

Complete the function `read_file()`, which opens the file at the specified path `path` and reads the first `buf_len` bytes of the file into the buffer `buf` using the `read()` system call. In any case, the function returns the actual number of bytes read.

- *Do not write beyond the end of the buffer (`buf_len`).*
- *Implement error handling for all system calls. The function shall read `buf_len` bytes, even if signals occur during the execution of the function. Other errors (incl. reaching the end of file) shall abort the read operation.*

Solution:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
size_t read_file(const char *path, char *buf, size_t buf_len) {
    int fd = open(path, O_RDONLY);
    if (fd == -1)
        return 0;

    size_t n = 0;
    while (n < buf_len) {
        ssize_t count = read(fd, buf, buf_len - n);
        if (count == 0) {
            break; /* End of file */
```

```
        } else if (count == -1) {
            if (errno == EINTR)
                continue; /* Signal interrupt */

            break; /* Other error */
        }

        buf += count;
        n   += count;
    }

    close(fd);

    return n;
}
```

Adding necessary includes **(1 P)**, *Opening the file* **(1 P)**, *Aborting on* `open()` *error* **(0.5 P)**, *Reading the file* **(1 P)**, *Handling signals* **(0.5 P)**, *Handling errors* **(0.5 P)**, *Stopping on EOF* **(0.5 P)**, *Stopping on buffer full* **(0.5 P)**, *Closing the file (no leak)* **(0.5 P)**, *Returning the number of bytes read* **(0.5 P)**

c) Vervollständigen Sie die Funktion `spawn_process()`, die einen separaten Prozess erzeugt, in dem die gegebene Eingabedatei asynchron komprimiert wird. **3.5 pt**

- Verwenden Sie `compress_file()` zur Kompression.
- Die Funktion soll sicherstellen, dass nie mehr als 4 zusätzliche Prozesse gleichzeitig ausgeführt werden.
- Sie können globale Variablen in dem zu Beginn der Aufgabe P2 reservierten Platz einfügen.

Complete the function `spawn_process()`*, which creates a separate process that asynchronously compresses the specified input file.*

- *Use* `compress_file()` *for compression.*
- *The function shall ensure that there are never more than 4 additional processes running at the same time.*
- *You may add global variables to the reserved space at the beginning of Assignment P2.*

Solution:

```
int parallel_processes = 0;

#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
void spawn_process(const char *in, const char *out) {
    /* wait until only 3 or less compression processes remain */
    if (parallel_processes >= 4) {
        wait(NULL);
    } else {
        parallel_processes++;
    }

    /* start a new process */
```

```
    if (fork() == 0)
        compress_file(in, out);
}
```

Adding necessary includes **(0.5 P)**, *Defining global process counter* **(0.5 P)** *Maintaining the correct process count* **(0.5 P)**, *Using* `wait` *to wait for a child to exit* **(1 P)**, *Using* `fork` *to create a new process* **(0.5 P)**, *Calling* `compress_file` *in the child* **(0.5 P)**.

d) Vervollständigen Sie die `main()`-Funktion des Programms, die für je zwei aufeinanderfolgende Kommandozeilenargumente `spawn_process()` aufruft. **3 pt**

- Das Programm soll sich erst beenden, nachdem alle zusätzlich gestarteten Prozesse terminiert sind.
- Gehen Sie von einer beliebigen Anzahl von Kommandozeilenargumenten aus. Bei ungerader Anzahl kann das letzte Argument ignoriert werden.

Complete the program's `main()` *function, which calls* `spawn_process()` *for each pair of two consecutive command line arguments.*

- *The program shall terminate only after all additionally spawned processes have terminated.*
- *Assume an arbitrary number of command line arguments. In case of an odd number of arguments, the last one can be ignored.*

Solution:

```
#include <sys/types.h>
#include <sys/wait.h>
int main(int argc, char **argv) {
    /* read the arguments and spawn a process for each input file */
    int i;
    for (i = 1; i + 1 < argc; i += 2)
        spawn_process(argv[i], argv[i + 1]);

    /* wait until all spawned processes have finished */
    while (parallel_processes > 0) {
        wait(NULL);
        parallel_processes--;
    }

    return 0;
}
```

Adding necessary includes **(0.5 P)**, *Looping through argument pairs* **(0.5 P)**, *Correctly handling an odd number of arguments* **(0.5 P)**, *Calling* `spawn_process` *on each pair* **(0.5 P)**, *Calling* `wait` *for all remaining processes* **(1 P)**. **Total: 15.0pt**

Aufgabe P3: Speicherverwaltung

Assignment P3: Memory Management

Gegeben sei ein System mit 32-Bit Addressraum, zweistufiger Seitentabelle, 4 KiB Seiten und Software-verwaltetem TLB.

Ein virtueller Adressraum (VAS) wird im Betriebssystem mit der `vas_t`-Struktur beschrieben. Die `vma_t`-Struktur stellt einen gültigen Speicherbereich (VMA) innerhalb eines Adressraums dar. Alle VMAs eines Adressraums sind in einer unsortierten, einfach-verketteten Liste abgelegt, wobei ein `vma_t`-Zeiger von `NULL` das Listenende bzw. eine leere Liste markiert.

Assume a system with 32-bit address space, two-level page table, 4 KiB pages, and software-managed TLB.

The operating system describes a virtual address space (VAS) with the `vas_t` structure. The `vma_t` structure represents a valid virtual memory area (VMA) within an address space. All VMAs of an address space are stored in an unsorted, singly-linked list, where a `vma_t` pointer of `NULL` indicates the list's end, or an empty list.

```
typedef struct pte {      // Page table entry (PTE)
    uint32_t pfn;         // Page frame number (PFN)
} pte_t;

typedef struct pt {       // Page table (PT)
    pte_t ptes[1024];
} pt_t;

typedef struct pd {       // Page directory (PD)
    pt_t *tables[1024];
} pd_t;

typedef struct vma {      // Virtual memory area (VMA)
    struct vma *next;     // Pointer to next VMA in list, or NULL
    uint32_t start;       // First virtual page number in VMA
    uint32_t end;         // Last virtual page number in VMA
} vma_t;

typedef struct vas {      // Virtual address space (VAS)
    vma_t *vmas;          // Unsorted list of VMAs in address space
    pd_t pdir;            // Page directory
} vas_t;

#define INV_PFN ((uint32_t)(-1)) // Invalid page frame number (PFN)

// Allocates a number of bytes from kernel virtual memory
// (not zero-initialized) on success, NULL otherwise
void* kmalloc(size_t bytes);

// Frees a physical page frame with the given PFN.
// Calling with INV_PFN is undefined behavior!
void freeFrame(uint32_t pfn);

// Flushes the TLB, thereby invalidating all TLB entries
void flushTlb(void);
```

a) Vervollständigen Sie die Funktion `allocVas()`, die einen neuen Adressraum (`vas_t`) aus Kernelspeicher alloziert, initialisiert und zurückgibt. **3 pt**

- Der Adressraum soll keine gültigen Speicherbereiche enthalten.
- Der durch die Seitentabellenhierarchie belegte Platz soll möglichst klein sein.
- Die Funktion gibt im Fehlerfall `NULL` zurück.

Complete the function `allocVas()`, which allocates, initializes, and returns a new address space (`vas_t`) from kernel memory.

- *The address space shall not contain any valid virtual memory areas.*
- *The memory required by the page table hierarchy shall be minimal.*
- *The function returns `NULL` on error.*

Solution:

```
void* kmalloc(size_t bytes);

vas_t* allocVas(void) {
    vas_t *vas = kmalloc(sizeof(vas_t));
    if (vas == NULL)
        return NULL;

    // alt: memset(vas, 0, sizeof(vas_t))
    vas->vmas = NULL;
    for (int i = 0; i < 1024; ++i)
        vas->pdir.tables[i] = NULL;

    return vas;
}
```

Allocation with `kmalloc()` **(1 P)**, *error handling* **(0.5 P)**, *initialization of `vmas`* **(0.5 P)**, *initialization of page directory* **(0.5 P)**, *returning vas* **(0.5 P)**

b) Vervollständigen Sie die Funktion `addVma()`, die den gegebenen virtuellen Speicherbereich `vma` in die Liste der gültigen Speicherbereiche des Adressraums `vas` einfügt. **3.5 pt**

- Die Operation schlägt fehl, wenn der neue Speicherbereich sich mit einer bereits eingetragenen Region überschneidet (siehe Abbildung).
- Die Funktion gibt im Fehlerfall `-1` zurück, ansonsten `0`.

Complete the function `addVma()`, which adds the specified virtual memory area (`vma_t`) to the list of valid VMAs in the address space `vas`.

- *The operation fails if the new area overlaps with an existing area (see figure).*
- *The function returns `-1` on error, `0` otherwise.*

Solution:

```
int addVma(vas_t *vas, vma_t *vma) {
    vma_t *ex = vas->vmas;
    while (ex) {
        if (((vma->end >= ex->start) &&
             (vma->end <= ex->end)) ||
            ((vma->start >= ex->start) &&
             (vma->start <= ex->end)) ||
            ((vma->start <= ex->start) &&
             (vma->end >= ex->end)))
```

```
            return -1;

        ex = ex->next;
    }

    vma->next = vas->vmas;
    vas->vmas = vma;
    return 0;
}
```

*To determine if the given region overlaps with any existing area, we have to check for three cases (***(0.5 P)** *for each case,* **(0.5 P)** *for the loop). Note that the last case is implicitely checked for by the first two cases.*

Adding VMA **(1 P)**, *returning correct values* **(0.5 P)**

c) Vervollständigen Sie die Funktion `getPte()`, die einen Pointer auf den Seitentabelleneintrag (PTE) zu der virtuellen Adresse `vaddr` liefert. **4.5 pt**

- Die Funktion allokiert eine neue Seitentabelle falls nötig und initialisiert deren PTEs, sodass diese auf keine gültige physische Seite zeigen.
- Sie müssen keine Fehlerbehandlung durchführen.

Complete the function `getPte()`, *which returns a pointer to the page table entry (PTE) for the virtual address* `vaddr`.

- *The function allocates a new page table if necessary and initializes the PTEs to point to no valid physical frame.*
- *You do not need to handle errors.*

Solution:

```
void* kmalloc(size_t bytes);

pte_t* getPte(vas_t *vas, uint32_t vaddr) {
    uint32_t idx1 = vaddr >> 22;
    uint32_t idx2 = (vaddr >> 12) & 0x3ff;  // alt: & ((1 << 10) - 1)

    pt_t *t = vas->pdir.tables[idx1];
    if (!t) {
        t = kmalloc(sizeof(pt_t));

        for (int i = 0; i < 1024; ++i)
            t->ptes[i].pfn = INV_PFN;

        vas->pdir.tables[idx1] = t;
    }

    return &t->ptes[idx2];
}
```

We have a 32-bit address space which means 32-bit virtual addresses. With a two-level table, 1024 entries each, each level is addresses by 10 bits. The remaining 12 bits specify the page offset.

Extracting the first-level table index **(0.5 P)**, *extracting the second-level table index* **(1 P)**, *checking if the second-level table exists* **(0.5 P)**, *allocating new page table* **(1 P)**, *initializing PTEs* **(1 P)**, *returning address of PTE* **(0.5 P)**

d) Vervollständigen Sie die Funktion `invalidateVma()`, die alle Abbildungen von virtuellen auf physische Seiten des Speicherbereichs `vma` aufhebt und betroffene physische Seiten freigibt. **4.0 pt**

- Sie können die Funktion `getPte()` verwenden.
- Allozierte Seitentabellen müssen nicht freigegeben werden.
- Beachten Sie, dass nicht alle PTEs gültige Abbildungen auf physische Seiten enthalten müssen.

Complete the function `invalidateVma()`, which invalidates all mappings of virtual to physical pages for the memory area `vma`, and frees the physical frames.

- *You can use the function `getPte()`.*
- *Allocated page tables do not need to be released.*
- *Keep in mind that not all PTEs may contain valid mappings to physical frames.*

Solution:

```
pte_t* getPte(vas_t *vas, uint32_t vaddr);
void freeFrame(uint32_t pfn);
void flushTlb(void);

void invalidateVma(vas_t *vas, vma_t *vma) {
    for (uint32_t p = vma->start; p <= vma->end; ++p) {
        pte_t *pte = getPte(vas, p << 12);
        if (pte->pfn != INV_PFN) {
            freeFrame(pte->pfn);
            pte->pfn = INV_PFN;
        }
    }

    flushTlb();
}
```

Loop **(1.0 P)**, *correct use of `getPte()`* **(1.0 P)**, *handling of non-existing mappings* **(0.5 P)**, *release of frames* **(0.5 P)**, *invalidation of PTEs* **(0.5 P)**, *flush of TLB* **(0.5 P)**. **Total: 15.0pt**