Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

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

Musterlösung Nachklausur
08.09.2023

Alle Punkteangaben ohne Gewähr!

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

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

- Die Prüfung besteht aus 23 Blättern: Einem Deckblatt, 22 Aufgabenblättern mit insgesamt 5 Theorieaufgaben und 3 Programmieraufgaben sowie 0 Seiten Man-Pages.

  The examination consists of 23 pages: One cover sheet, 22 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) Nennen Sie je zwei Dinge, die im Prozesskontrollblock (PCB) bzw. im Threadkontrollblock (TCB) gespeichert werden. **2 pt**

Name two things that are saved in the process control block (PCB) and in the thread control block (TCB), respectively.

Solution:

PCB *Address space, open files, child processes, pending alarms*

TCB *Instruction pointer, registers, stack, scheduling state*

b) Nennen Sie zwei Möglichkeiten, die Parameter eines Systemaufrufs an den Kernel zu übergeben. Welche dieser Möglichkeiten bietet eine höhere Geschwindigkeit? **1 pt**

Name two ways of passing the parameters of a system call to the kernel. Which of these ways offers better performance?

Solution:

Parameters can be passed either in registers or the stack **(0.5 P)**. *Passing them in registers is faster* **(0.5 P)** *because memory access is generally slower.*

c) Wie bemerkt das Betriebssystem, dass ein nicht privilegierter Prozess versucht hat, eine privilegierte Instruktion auszuführen? **1 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.

d) Erklären Sie den Begriff *preemption*. **1 pt**

Explain the term preemption.

Solution:

Forcefully take away a resource from, e.g., a process.

***Note**: Answers that gave a concrete example (e.g., "scheduler takes away CPU from a process") instead of a general explanation got* **(0.5 P)**.

e) Erklären Sie kurz das Problem der *Priority Inversion* anhand eines Beispiels, wann es auftreten kann. **2 pt**

Briefly explain priority inversion *based on an example of when it may occur.*

Solution:

Priority inversion may occur when, for example, two threads are scheduled with static priorities and share a lock. If the low priority thread holds the lock, the high priority thread has to wait for the low priority thread to release the CPU. However, the low priority thread will never be dispatched.

f) Moderne Linux-Systeme bieten ein Interface namens *io_uring*. *io_uring* ermöglicht User-Space-Prozessen, I/O-Aufgaben (bspw. Dateioperationen) in einen Puffer zu schreiben. Der Kernel liest asynchron die Aufgaben aus diesem Puffer und führt diese dann aus. Erläutern Sie einen Vor- und einen Nachteil dieses Konzepts gegenüber den aus der Vorlesung bekannten I/O-Interfaces. **2 pt**

Modern Linux systems offer an interface known as io_uring. io_uring *allows user-space processes to write I/O tasks (e.g., file operations) into a buffer. The kernel then fetches the tasks asynchronously from this buffer and executes them. Explain one advantage and one disadvantage of this concept compared to the I/O interfaces known from the lecture.*

Solution:

io_uring *allows to significantly reduce the overhead incurred by context switches due to system calls, particularly for I/O-heavy processes. However, the latency of specific operations may be increased due to the asynchronous nature of the concept. There may be an additional delay between the time a process submits an I/O operation and the time the kernel is able to fetch the operation.*

Total: 9.0pt

Aufgabe T2: Prozesse und Threads

Assignment T2: Processes and Threads

a) Betrachten Sie einen Drucker, der ununterbrochen neue Aufträge von vielen Nutzern erhält. Immer wenn ein Auftrag fertig bearbeitet ist, bearbeitet der Drucker als nächstes den Auftrag mit der niedrigsten Seitenanzahl. Erklären Sie je einen Vor- und Nachteil dieser Strategie. **2 pt**

Consider a printer which continuously receives jobs from many users. Whenever the printer has finished printing a job, it selects the job with the least amount of pages to print next. Explain an advantage and a disadvantage of this policy.

Solution:

(+) *Selecting the shortest job first maximizes overall throughput.*

(-) *Shortest job first leads to starvation: assuming that new short jobs are steadily coming in, longer jobs have to wait indefinitely.*

b) In welchen Threadmodell können *Scheduler Activations* zum Einsatz kommen? Welches Problem wird damit gelöst? **1 pt**

Which thread model do scheduler activations *apply to? Which problem do they solve?*

Solution:

Scheduler activiations can improve ***hybrid threads*** *(M-to-N model)* **(0.5 P)** *by notifying the user space scheduler that a thread will block, allowing a switch to a different thread. Without scheduler activations, a blocking thread also blocks all other threads on the same kernel-level thread.*

c) Ein System ist mit zwei CPU-Kernen ausgestattet. Das Betriebssystem konfiguriert den einen Kern (Kern 1) so, dass er alle 5 Millisekunden unterbrochen wird, und den anderen Kern (Kern 2) so, dass er alle 500 Millisekunden unterbrochen wird. **2 pt**

Warum und für welche Systeme kann solch eine Aufteilung sinnvoll sein?

A system is equipped with two CPU cores. The operating system configures one core (core 1) to be interrupted every 5 milliseconds, and the other core (core 2) to be interrupted every 500 milliseconds.

Why and for which systems might such a division make sense?

Solution:

This setup allows for both high responsivity for latency-critical, interactive or IO-bound applications as well as high throughput for long-running and compute-intensive applications. **(1 P)** *The high-responsivity tasks should be scheduled on core 1 and the long-running tasks on core 2.* **(0.5 P)***. This decision only makes sense if a system operates both kinds of processes at the same time* **(0.5 P)***, e.g., in a workstation which both offers a user interface and runs computations. If the system is specialized on one type of task, the scheduler should be built to fit, too* **(0.5 P)***.*

Verteilen Sie die folgenden Prozesse (mit je einem Thread) auf die Kerne 1 und 2, und begründen Sie Ihre Entscheidung kurz. **2 pt**

Distribute the following processes (each containing one thread) to the cores 1 and 2, and give a reason for your decision.

	runtime	max allowed turnaround time	average time between syscalls
Process 1	5000 ms	25000 ms	2000 ms
Process 2	2000 ms	16 ms	2 ms
Process 3	5000 ms	25000 ms	2 ms
Process 4	200 ms	5000 ms	200 ms

Solution:

Process 1: Core 2, as the process is apparently compute-bound.
Process 2: Core 1, as the process is latency-critical and frequently calls the OS, and therefore seems to be IO-bound.
Process 3: Core 1, as even though the process is not latency-critical, it frequently calls the OS and therefore seems to be IO-bound.
Process 4: Core 2, as even though the process does not use up the whole time slice, it still runs to completion without being interrupted and therefore is compute-bound.

Wann ergibt es Sinn, einen Prozess der eigentlich auf dem einen Kern ausgeführt werden sollte, auf den anderen Kern zu schieben? **2 pt**

When does it make sense to move a process that should have been scheduled to one core to the other core?

Solution:

Scheduling on the other core only makes sence if the optimal core is already filled with tasks **(0.5 P)**. *If core 2 is already full, compute-bound processes can also make progress on core 1, though slower, therefore moving the task makes sence* **(0.5 P)**. *On the other hand, if core 1 is full, moving a tasks only makes sense sometimes: If the task is latency-critical, moving to core 2 might mean that the task is only ever scheduled after the deadline was already missed, so moving these tasks should be avoided* **(0.5 P)**. *If the task is only IO-bound though, moving these tasks over means litte progress as they might get starved by compute-bound processes, but no deadlines are missed and some progress is made. Therefore, moving these tasks makes sence* **(0.5 P)**.

Answers discussing tasks that change characteristics during runtime are awarded **(1.5 P)** *as these answers still take into account the static nature of the partitioning, and another* **(0.5 P)** *is awarded if starvation is discussed.*

Note: *For answers only stating that tasks should be well-distributed or that tasks should be moved to the idle core, only* **(0.5 P)** *is given as these answers do not discuss the nuances of possible starvation and degredation of a otherwise statically partitioned system.*

Total: 9.0pt

Aufgabe T3: Speicher

Assignment T3: Memory

a) Beschreiben Sie Demand-Paging und Pre-Paging, und nennen Sie jeweils einen Anwendungsfall, bei dem das Verfahren Vorteile gegenüber dem jeweils anderen hat. **2 pt**

Describe demand-paging and pre-paging, and name a use case each where the respective technique is advantageous to the other.

Solution:

demand-paging: only reserve space in the address space without actually adding backing pages **(0.5 P)**, *good for processes which allocate, but not actually use all of the memory, or require little startup latency* **(0.5 P)**.
pre-paging: add backing pages directly at the allocation **(0.5 P)**, *good for processes that actually use the all of the memory directly* **(0.5 P)**. ***Note:*** *Pre-paging is not limited to file reads, as the kernel can map multiple empty anonymous pages in one call, too. (see, e.g., the flag* `MAP_POPULATE` *in the linux* `mmap` *call).*

b) Sie haben ein System mit 4 freien Frames (f) ohne TLB, das Clock-Replacement als Page Replacement Strategie nutzt. Berechnen Sie die Abfolge, in der Seiten ersetzt werden. Am Anfang sind keine Seiten im Speicher. Tragen Sie in der Tabelle neben der aktuellen Seite („page“, p) auch den Status des Referenced-Bits (r) und die Position des Clock-Zeigers (c) ein. **3 pt**

You have a system with 4 free frames, which does not have a TLB. Said system uses clock replacement as its page replacement strategy. At the beginning, there are no pages in memory. Fill the table with the current page (p), the status of the referenced bit (r) as well as the clock position (c).

f	page ⇒	1			2			3			4			5			2			6			4			3		
	p	r	c	p	r	c	p	r	c	p	r	c	p	r	c	p	r	c	p	r	c	p	r	c	p	r	c	p	r	c
1			x	1	x		1	x		1	x		1	x	x	5	x		5	x		5	x		5	x		5		
2						x	2	x		2	x		2	x		2		x	2	x	x	2			2			3	x	
3									x	3	x		3	x		3			3			6	x		6	x		6	x	x
4												x	4	x		4			4			4		x	4	x	x	4		

c) Sie haben ein System mit einem hardware-managed TLB. Wie können Sie damit einen software-managed TLB nachbauen? **1 pt**

You have a system with a hardware-managed TLB. How can you use it to emulate a software-managed TLB?

Solution:

You initialize the hardware pagetable to map as invalid for every entry **(0.5 P)**. *As the hardware walker fails, it generates a page fault, in which the OS can execute the software page table walk* **(0.5 P)**, *and the OS then inserts the found address into the CPU.* Inserting into the CPU is ISA-specific and might contain actually modifying the hardware page table temporarily so that the hardware walker can be restarted.

d) In Ihrem Programm kommen viele Objekte (aus C-`struct`) bestehend aus 6 `long long`-Ganzzahlen mit je 8 byte vor, die in einem Array gespeichert werden. Welches Problem bekommen Sie auf einem System mit einem Cache mit 64-Byte Cachelines, und wie beheben Sie das? **1 pt**

In your program there are a lot of objects (from C-`struct`s) containing 6 `long long` integers with 8 bytes each, which are stored in an array. Which problem do you encounter on a system with a cache with 64-byte cache lines, and how do you fix that problem?

Solution:

The structs each use 48 byte, which is unaligned with the 64 byte cache lines **(0.5 P)**, *and can lead to variance in the load times. To fix that, add 16 byte padding at the end of the `struct`. A different solution that fixes this particular problem (but might raise different ones like major latency differences between struct parts and more fragmentation) is to split the `struct` into multiple parts and to store each one in its own cache-line aligned array.* **(0.5 P)**

e) Gegeben sei ein System, das virtuellen Adressen mittels einer dreistufigen Seitentabelle übersetzt. Die Seitengröße beträgt 8 KiB, die Größe eines Eintrages in der Seitentabelle beträgt 8 Byte. Das System ist außerdem mit einem TLB ausgestattet, der 256 Einträge fasst. Es gibt außerdem einen physisch indizierten und physisch getaggten, 4 MiB fassenden Cache mit 64 Byte Cachezeilen. Sowohl der TLB als auch der Cache sind am Anfang leer. **2 pt**

Ein Prozess kopiert einen Puffer von einer Größe von 1 MiB in einen neuen Puffer, und flusht danach alle Cachezeilen mit modifiziertem Inhalt. Wie viele Speicherzugriffe sind dafür mindestens nötig?

Zweierpotenzen genügen als Antwort.

Consider a system which translates virtual addresses with a three-level page table. The page size is 8 KiB, the size of a single page table entry is 8 bytes. Additionally, the system contains a 256-entry TLB, and a physically-indexed, physically-tagged cache of size 4 MiB with 64-byte cachelines. Both the TLB as well as the cache are initially empty.

A process copies a buffer of size 1 MiB into a new buffer, and flushes all cache lines with modified content. How many memory accesses are required at a minimum?

Powers of two suffice as an anwer.

Solution:

For each cacheline in the buffer, one load of the source and one store in the target are needed, which are 1 MiB / 64 B = $2^{20}/2^6 = 2^{14}$ *accesses for each of the buffers, so* $2 \cdot 2^{14} = 2^{15}$ *accesses total* **(0.5 P)**. *This fills only 2 MiB of the 4MiB cache, so there is enough space for the complete needed page table* **(0.5 P)**.

Each 8 KiB page can store 8 KiB/ 8 B = 1024 entries. 1 MiB consists of 128 pages, which means that 256 entries are needed (128 per buffer) **(0.5 P)**. *We need to load the pointer to the second and to the third level page table (one access each), and need 256 B * 8 / 64 B =* $2^8 * 2^3/2^6 = 2^5$ *accesses for the third level. Summed up, the result is* $1 + 1 + 2^5 + 2^{15}$ **(0.5 P)**.

Total: 9.0pt

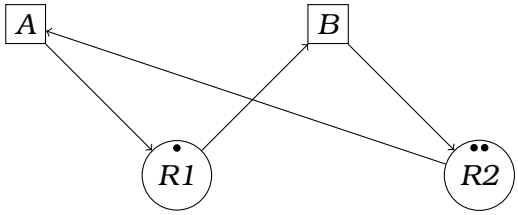
Aufgabe T4: Koordination und Kommunikation von Prozessen

Assignment T4: Process Coordination and Communication

a) Zeichnen Sie einen *Resource Allocation Graph* (RAG), der einen Kreis enthält, aber kein Deadlock. **2 pt**

Draw a valid resource allocation graph *(RAG) that contains a cycle, but does not depict a deadlock.*

Solution:



***Note**: Theoretically, there is an infinite amount of correct answers to this question.*

Beschreiben Sie, wie Sie den Graphen verändern würden, sodass ein Deadlock zu sehen ist. **1 pt**

Describe how you would change the graph so that it shows a deadlock.

Solution:

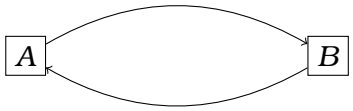
Remove R2's second resource.

***Note**: Depending on your answer to the previous question, other solutions may be correct.*

Zeichen Sie einen *Wait-For Graph* (WFG) basierend auf dem obigen RAG mit dem Deadlock aus der vorherigen Teilaufgabe. **1 pt**

Draw a wait-for graph *(WFG) that corresponds to the RAG above and contains the deadlock described in the previous question.*

Solution:



b) Auf welchen Systemen ist das Maskieren von Interrupts hinreichend, um das Problem kritischer Abschnitte zu lösen? Warum? **1 pt**

On what kinds of systems is it sufficient to mask interrupts to correctly implement critical sections? Why?

Solution:

Single-processor systems. By disabling interrupts, we can ensure that no interrupt handler can potentially enter the critical section. Further, as no other processor core can execute code at the same time, it is guaranteed that no other thread may enter the critical section simultaneously.

c) Spinlocks können bspw. mit atomaren Swap-Instruktionen implementiert werden, die den Inhalt eines Registers mit dem Inhalt an einer Speicheradresse vertauschen. Da Speicherzugriffe wesentlich langsamer sind als Registerzugriffe, würde es sich bei ausreichend vielen unbenutzten Registern anbieten, stattdessen zwei Register zu vertauschen. Warum würde eine derartige Implementation für Spinlocks nicht funktionieren? **1 pt**

Spinlocks may be implemented using atomic swap instructions that exchange a register's content with the data at a specified memory address. As memory accesses are much slower than register accesses, it seems favorable to exchange the contents of two registers instead if enough unused registers are available. Why would such an implementation not work for spinlocks?

Solution:

Register values are specific to each thread, and hence, other threads can not see changes to a thread's register set. Exchanging the contents of two registers would therefore be useless.

***Note**: We recommend students preparing for the exam to read up on fundamental concepts such as registers to obtain a thorough understanding, not only to learn that different types of memory exist in a computer, but also* why *and* how *they differ. Our experience is that many misconceptions about registers and main memory exist among students, and we truly believe that a good comprehension is vital for grasping the implications for operating systems.*

d) Nennen Sie eine weitere atomare Instruktion (außer eines simplen Swaps), die zur Implementation von Spinlocks verwendet werden kann. **1 pt**

Name another atomic instruction (except a simple swap) that can be used to implement spinlocks.

Solution:

test-and-set, fetch-and-add, compare-and-swap, . . .

e) Betrachten Sie ein System mit einem Mehrkernprozessor, auf dem ein Prozess mit mehreren Threads läuft. Beschreiben Sie, wann in diesem Szenario Spinlocks besser geeignet sind im Vergleich zu blockierenden Locks für kritische Abschnitte und umgekehrt. **2 pt**

Given a system with a multi-core CPU that is running a process with multiple threads. Describe when spinlocks are better suited for critical sections in this scenario compared to blocking locks and vice versa.

Solution:

Generally, spinlocks are well-suited for short critical sections and low contention of locks, i.e., where the time spent spinning is typically very short so that not much CPU time is wasted spinning. Blocking locks are better when the time required to acquire a lock is long, i.e., the time may be better spent executing other tasks instead of spinning.

Total: 9.0pt

Aufgabe T5: I/O, Hintergrundspeicher und Dateisysteme

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

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

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

Solution:

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

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

b) Wir haben in der Vorlesung drei Dateiallokationsstrategien kennengelernt: (a) Indexed Allocation, (b) Chained Allocation und (c) Contiguous Allocation. Sortieren Sie die Strategien nach der Reihenfolge ihrer Eignung in den folgenden Szenarien. Begründen Sie für jede Strategie kurz, warum sie sich eignet/nicht eignet.

We have discussed three file allocation strategies in the lecture: (a) indexed allocation, (b) chained allocation, and (c) contiguous allocation. Order the strategies based on how well they perform in each of the following scenarios. Provide a short explaination why each strategy performs well/bad.

Schnelles Auslesen aufeinanderfolgender Blöcke einer Datei. **1.5 pt**

Fast reading of consecutive blocks of a file.

Solution:

(c) Contiguous allocation: best, since file blocks are contiguous

(a) Indexed allocation: worse, since file blocks may not be contiguous

(b) Chained allocation: even worse, since it is not possible to load blocks in parallel

Effiziente Datenspeicherung durch wenig Metadaten. Gehen Sie davon aus, dass der Speicher unfragmentiert ist. **1.5 pt**

Efficient storage of data in terms of few metadata. Assume that the memory is defragmented.

Solution:

(c) Contiguous allocation: best, since only start address and length need to be stored for a file

(b) Chained allocation: pointer at the end of blocks in addition to start address

(a) Indexed allocation: separate block pointers with additional index structure

c) Nennen Sie vier Metadaten, die das Betriebssystem für eine geöffnete Datei halten muss. Welche der Informationen werden systemweit nur einmal, welche pro geöffneter Instanz gespeichert? **2 pt**

Give four pieces of metadata that the operating system has to store for an open file. Which information is stored once per system and which once per open file instance?

Solution:

per system: *file-open counter* **(0.5 P)**, *information on cached file parts* **(0.5 P)**
per instance: *access mode* **(0.5 P)**, *file pointer* **(0.5 P)**

d) Erklären Sie für die folgenden Daten, wo diese in einem Unix-Dateisystem gespeichert werden bzw. wie die Information hergeleitet werden könnte. **3 pt**

Explain where the following data is stored in a Unix file system or how the information may be inferred.

The file name is represented as directory entry **(0.5 P)**

The name of containing directory may be ambiguous and is only implicitly stored through the directory structure. Files may be stored in multiple directories (hard links). **(1 P)**

The file size is stored in the inode. **(0.5 P)**

The number of symbolic links is not stored at all and can only be obtained through searching the entire directory tree. **(1 P)**

Total: 9.0pt

Aufgabe P1: C-Grundlagen

Assignment P1: C Basics

a) Gegeben sei folgende Funktion, die rekursiv die Summe $\sum_{k=1}^{n} k$ für eine Zahl n berechnet.

The following function recursively calculates the sum $\sum_{k=1}^{n} k$ for a number n.

```
int sum(int n) {
    if(n == 1) return 1;

    return n + sum(n - 1);
}
```

Benennen Sie das Problem, das bei Ausführung von `sum()` mit sehr großen Eingaben auftreten kann. **1 pt**

Hinweis: Gesucht ist ein Problem, das zum Absturz des Programms führen kann.

Name the problem that may occur when calling `sum()` with very large input numbers.

***Hint**: We are looking for an issue that may cause the program to crash.*

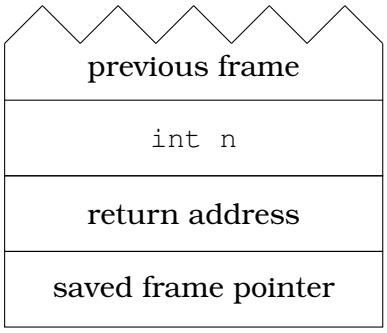
Solution:

Stack overflow.

***Explanation** (not required in the exam): `sum()` recursively calls itself. Every call requires a new stack frame, however, the stack only has limited space available. When calling `sum()` with large inputs, the stack will eventually have to grow beyond its limit, causing the program to crash.*

Tragen Sie alle Elemente des Stack-Frames eines Aufrufs von `sum()` ein. Gehen Sie von `cdecl` als verwendeter Aufrufkonvention auf einem 32-bit-x86-Prozessor aus. **1.5 pt**

Fill in all fields of the following stack frame representing a call to `sum()`. Assume that the `cdecl` calling convention is used on a 32-bit x86 CPU.



Geben Sie eine mathematische Funktion $f(x)$ an, mit der die größtmögliche Zahl bestimmt werden kann, die als Eingabe für `sum()` auf dem gegebenen System möglich ist. Benennen Sie die Bedeutung des Parameters x. **1.5 pt**

Define a mathematical function $f(x)$ that can be used to determine the largest number that may be passed as input for `sum()` on the system defined above. Describe the meaning of the parameter x.

$f(x) = x/3$ **or** $f(x) = x/12$
Bedeutung von (*Meaning of*) x: Stack size in 32-bit words **or** bytes

Schreiben Sie eine Implementation von `sum()`, die für beliebig große Eingaben (innerhalb der Limits des Datentyps) funktioniert. **2 pt**

Write an implementation of `sum()` that works for arbitrarily large inputs (within the limits of the data type).

Solution:

```
int sum(int n) {
    return n * (n + 1) / 2;
}
```

b) Welches Problem tritt ber der Verwendung von `init_work_item()` auf? **1 pt**

Which problem occurs when using `init_work_item()`?

```
struct work_item *init_work_item() {
    struct work_item item;
    memset(&item, 0, sizeof(struct work_item));
    return &item;
}
```

Solution:

`init_work_item()` returns a pointer to a value within its stack frame.

c) Geben Sie die Werte der folgenden Ausdrücke nach Ausführung des Codes auf der linken Seite an. Gehen Sie von einem 32-bit-x86-Prozessor aus. **3 pt**

Give the values of the following expressions after the code on the left side has run. Assume a 32-bit x86 CPU.

```
char *text1 = "Hello";
char text2[] = "Hello";
char *text3 = malloc(6);
strcpy(text3, text1);
```

Expression	Value
`sizeof(text1)`	4
`sizeof(text2)`	6
`sizeof(text3)`	4
`strlen(text1)`	5
`strlen(text2)`	5
`strlen(text3)`	5

d) Ihr Programm enthält folgenden Codeschnipsel. Das Programm funktioniert auf Ihrem Rechner problemlos. Als Sie das Programm einem Freund zum Ausprobieren schicken, klagt dieser über Abstürze. Warum funktioniert das Programm bei Ihnen und warum nicht bei Ihrem Freund? Nehmen Sie an, dass `item` und `item->is_last` immer korrekt gesetzt sind. **1 pt**

Your program contains the following code snippet. The program works flawlessly on your computer. However, your friend reports crashes as they try out the program. Why is the program working on your computer and not on your friend's? Assume that `item` and `item->is_last` are always set correctly.

```
struct work_item *get_next(struct work_item *item) {
    if (item->is_last) return NULL;

    unsigned int address = (unsigned int) item;
    address += sizeof(struct work_item);
    return (struct work_item*) address;
}
```

Solution:

The size of a pointer (i.e., an address) is larger than the size of an `int` *on my friend's system. Thus, casting* `item` *to an* `unsigned int` *cuts off the upper bits of the pointer, and in turn, the calculated* `address` *is invalid. Trying to dereference the returned pointer later hence causes the program to crash.*

Wie würden Sie das Problem beheben? Beschreiben Sie eine Lösung, die auf **allen** denkbaren Systemen funktioniert. **1 pt**

*How would you solve the issue? Describe a soluton that works on **all** plausibly imaginable systems.*

Solution:

Remove the entire address calculation and replace it with

```
    return address + 1;
```

e) Implementieren Sie die Funktion `split()`, die einen String nach `n` Zeichen aufteilt. Der erste Abschnitt soll direkt (als Rückgabewert) zurückgegeben werden, der zweite in `part2`. Beide Abschnitte sollen jeweils korrekt `NULL`-terminiert sein. **2 pt**

- Der Buffer von `s` hat genau ein Byte mehr Platz, als der String inkl. `NULL`-Byte lang ist.
- Sie dürfen **ausschließlich** folgende Funktionen der C-Standardbibliothek benutzen: `memcpy()`, `memmove()`, `strlen()`.

Implement the function `split()` that splits a string after `n` characters. The first part shall be returned directly via the function's return value, the other via `part2`. Both parts shall be correctly `NULL`-terminated.

- *The buffer of `s` is exactly one byte longer than the string (including the `NULL` byte).*
- ***Do not** use any C standard library functions other than `memcpy()`, `memmove()`, and `strlen()`.*

Solution:

```
char *split(char *s, char **part2, int n) {
    memmove(s + n + 1, s + n, strlen(s) - n);
    *part2 = s + n + 1;
    s[n] = 0;
    return s;
}
```

Implementieren Sie einen korrekten Aufruf der Funktion `split()`. Der String `s` soll in der Mitte geteilt werden und ist für diesen Zweck korrekt initialisiert. **1 pt**

Implement a valid call to `split()`. Split the string `s` in the middle. Assume that `s` is initialized correctly for this purpose.

Solution:

```
void test_split(char *s) {
    char *part1;
    char *part2;

    part1 = split(s, &part2, strlen(s) / 2);

    printf("%s|%s|\n", part1, part2);
}
```

Total: 15.0pt

Aufgabe P2: Dateisystembaum

Assignment P2: File System Tree

In dieser Aufgabe sollen Sie eine Anwendung schreiben, die Ihren Dateisystembaum ausgibt. Die Anwendung steigt dafür rekursiv in die Unterordner eines Ordners ab, und gibt alle Elemente darin aus, eingerückt nach der Tiefe der Ordnerstruktur.

- 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. Bei erwünschter Fehlerbehandlung überprüfen Sie, ob Sie den Fehler innerhalb Ihres Programms verarbeiten können. Ansonsten beenden Sie das Programm.

In this assignment, you will implement an application which prints your file system tree. To achieve this, the application recursively descends into subfolders of a folder and prints all elements it contains, indented by the depth of the folder structure.

- *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. If error handling is required check whether you can handle the error in your program. Otherwise, terminate the application.*

Solution:

```c
#include <sys/types.h> // exercise d)
#include <sys/stat.h> // exercise d)
#include <dirent.h> // exercises a), f)
#include <errno.h> // missing manpage; not required
#include <stdio.h> // exercise e)
#include <string.h> // optionally b), c)
#include <stdlib.h> // exercises a), c), d)
#include <unistd.h> // exercise  d)
```

a) Implementieren Sie die Funktion, die das nächste Element im Verzeichnis aus dem Verzeichnisstrom `s` heraussucht. Behandeln Sie Fehler der entsprechenden Bibliotheksfunktion. **2 pt**

Next, implement the function that finds the next element in the directory stream `s`. Handle the errors of the respective library function.

Solution:

```
struct dirent *next_dir(DIR *s) {
    errno = 0;
    struct dirent *ptr = readdir(s); // 1 P
    if(!ptr && errno) // 0.5 P
        exit(1); // 0.5 P
    return ptr;
}
```

b) Damit Sie beim Iterieren über das Verzeichnis nicht in eine Endlosschleife laufen, müssen Sie die `.` und `..` Verzeichnisse überspringen. Implementieren Sie dazu die Funktion `skip_dir()`. **1 pt**

To avoid an infinite loop when iterating the directory, you need to skip the `.` and `..`-directories. Implement the function `skip_dir()`.

Solution:

```
int skip_dir(char *name) {
    return (name[0] == '.' && name[1] == 0) ||
           (name[0] == '.' && name[1] == '.' && name[2] == 0);
}
```

oder

```
int skip_dir(char *name) {
    return !strcmp(name, ".") || !strcmp(name, "..");
} // the ! is important because of short-circuit evaluation
```

c) Im Verlauf des Programms müssen Sie rekursiv in Unterverzeichnisse absteigen. Dazu müssen Sie aus zwei Teilpfaden einen neuen Pfad zusammensetzen. Implementieren Sie dazu die folgende Funktion: **2 pt**

In your program, you will need to recursively descent into subfolders. To do that, you need to concatenate two paths into one new path. Implement the following function:

Solution:

```
char *join_path(char *path, char *subpath) {
    size_t path_len = strlen(path);
    size_t sub_len  = strlen(subpath);
    // old path + / + new length + 0-character
    size_t newbytes = path_len + sub_len + 2;
    char *newpath = (char*)malloc(newbytes); // 0.5 P
    memcpy(newpath, path, path_len); // 0.5 P for copies
    newpath[path_len] = '/'; // 0.5 P
    memcpy(newpath + path_len + 1, subpath, sub_len);
    newpath[newbytes - 1] = 0; // 0.5 P
    return newpath;
}
```

d) Um herauszufinden, ob ein Pfad auf einen Ordner zeigt, nutzen Sie den `stat`-Systemaufruf. Bestimmte Fehler dieses Systemaufrufs können Sie durch überspringen des Pfades lösen. Geben Sie 1 zurück, wenn es sich um ein Verzeichnis handelt, 2, wenn Sie diesen Pfad überspringen müssen, und 0 sonst. **2.5 pt**

To find out whether a path points to a folder, you use the `stat` system call. Certain errors of this system call can be solved by skipping the path. Return 1 if the path points to a directory, 2, if you need to skip this path, or 0 otherwise.

Solution:

```
int get_props(char *path) {
    struct stat statresult; // 0.5 P (not if malloced+leaked!)
    if(stat(path, &statresult)) { // 0.5 P
        if (errno == ENOENT || errno == ELOOP)
            return 2; // errno-check 0.5 P, correct errno 0.5 P

        exit(1); // 0.5 P
    }
    return (statresult.st_mode & S_IFMT) == S_IFDIR; // 0.5 P
}
```

e) Als letzte Teilfunktionionalität sollen Sie das aktuelle Element ausgeben. Geben Sie zunächst entsprechend des Parameters die Einrückung aus, benutzen Sie dafür den String `INDENT`. Danach geben Sie den übergebenen String aus, außerdem ein `/` am Ende eines Verzeichnisses. Zuletzt fügen Sie einen Zeilenumbruch ein. **1.5 pt**

The last remaining subfunctionality is to print the current element. Print the indention according to the respective parameter, using the string `INDENT`. Afterwards, print the string passed to the function, and a `/` at the end of a directory. Lastly, print a line break.

Solution:

```
void print(int indent, char *path, int is_dir) {
    for(int i = 0; i < indent; ++i)  //
        printf("%s", INDENT);        // 0.5 P combined
    printf("%s", path);      //
    if (is_dir) printf("/"); // 1 P combined
    printf("\n");            //
}
```

f) Fassen Sie jetzt die in den Aufgaben a) bis e) implementierten Funktionalitäten in einer rekursiv aufrufbaren Funktion zusammen. Aufgerufen wird die Funktion initial aus der gegebenen `main`-Funktion. Benutzen Sie folgendes Vorgehen: Zunächst öffnen Sie für jeden Aufruf das Verzeichnis, und für jeden Eintrag im Verzeichnis überprüfen Sie dann, ob Sie diesen Eintrag behandeln, welcher Typ von Eintrag es ist, geben den Eintrag aus und rufen falls nötig Ihre Funktion rekursiv auf. **6 pt**

Combine the functionality of exercises a) to e) into a recursively callable function. This function is called as presented in the given `main`-function. Use the following approach: For each call, first open the directory, and for each entry in the directory check if you need to handle the entry, and the type of the entry. Then, print the entry and call your function recursively if needed.

```
struct dirent *next_dir(DIR *s);
int skip_dir(char *name);
char *join_path(char *path, char *subpath);
int get_props(char *path);
void print(int indent, char *path, int is_dir);
```

Solution:

```
void print_dir(int indent, char *path) {
    DIR *stream = opendir(path); // 0.5 P
    while(1) {
        struct dirent *ptr = next_dir(stream); // 0.5 P
        if(!ptr) break; // 0.5 P

        if(skip_dir(ptr->d_name)) // 0.5 P
            continue; // 0.5 P

        char *newpath = join_path(path, ptr->d_name); // 0.5 P
        int statresult = get_props(newpath); // 0.5 P
        if(statresult == 2) {
            free(newpath); // 0.5 P if not leaked (see below)
            continue; // 0.5 P
        }

        int is_directory = statresult == 1;
        print(indent, newpath, is_directory); // 0.5 P
        if (is_directory)
            print_dir(indent + 1, newpath); // 0.5 P

        free(newpath);  // (see above)
    }
    closedir(stream); // 0.5 P
}
```

Total: 15.0pt

Aufgabe P3: Virtuelle Maschine für eBPF

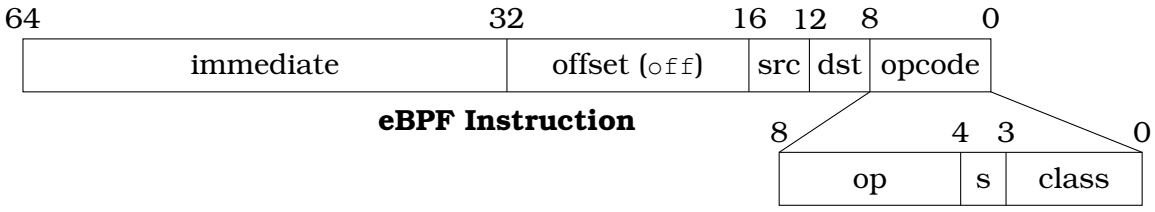
Assignment P3: eBPF Virtual Machine

In dieser Aufgabe werden Sie eine virtuelle Maschine für eine Teilmenge von eBPF schreiben. Ursprünglich für Linux entwickelt, erlaubt eBPF die Ausführung von Nutzercode direkt im Kernel. Ein eBPF-Programm ist eine Folge von RISC-Instruktionen, die jeweils 64 bit lang sind. Vor der Ausführung eines eBPF-Programms prüft ein Verifier, dass das Programm gültig ist.

Alle eBPF-Instruktionen haben die selben in der Abbildung unten dargestellten Komponenten. Der Opcode wird weiter in eine Instruktionsklasse (ALU oder JMP), ein Quelloperandenbit (s=0: Immediate, s=1: Register), sowie die auszuführende Operation aufgeteilt. In der Tabelle unten sind alle Instruktionen aufgelistet, die unsere VM unterstützt. Die elf Register (`r0-r10`) werden durch die `dst` und `src`-Felder adressiert. Alle ALU-Instruktionen schreiben das Ergebnis in das Register `dst`. Bei dem Offset-Feld von Sprüngen handelt es sich um eine vorzeichenbehaftete Zahl im Zweierkomplement relativ zu der folgenden Instruktion.

In this assignment, you are going to write a virtual machine for a subset of eBPF. Originating from Linux, eBPF allows running user code directly in the kernel. An eBPF program is a sequence of RISC instructions, each 64 bit long. Before running an eBPF program, a verifier ensures that the program is valid.

All eBPF instruction have the same components, as shown in the figure below. The opcode is further split into an instruction class (ALU or JMP), a source operand bit (s=0: immediate, s=1: register), and the operation to perform. The table below shows the instructions that our VM will support. The eleven registers (`r0-r10`) are addressed by the fields `dst` and `src`. All ALU instructions store the result in register `dst`. For jumps, the offset field is a signed number (two's complement) relative to the following instruction.



class	op	mnemonic	description
ALU 0x7	0x0	`add dst, src`	add src to dst
	0xa	`xor dst, src`	calculate exclusive or of dst and src
	0xb	`mov dst, src`	set value of dst to src
JMP 0x5	0x0	`ja +off`	jump (always) by `off` instructions
	0x1	`jeq dst, src, +off`	if dst equals src, jump like ja
	0x9	`exit`	exit the program, return r0

a) Dekodieren Sie die folgenden als Hexadezimalzahlen geschriebenen Instruktionen. Geben Sie die Inhalte aller relevanten Register nach Ausführung des Programms an. **3 pt**

Decode the following instructions, written as hexadecimal numbers. Give all relevant register contents after running the program.

Example:

instruction	mnemonic
`56253667 0000 00 b7`	`mov r0, 0x56253667`
`88888888 0000 07 b7`	`mov r7, 0x88888888`
`00000000 0000 70 af`	`xor r0, r7`
`00000000 ffff 07 1d`	`jeq r7, r0, -1`
`00000000 0000 00 95`	`exit`
registers: `r0 = 0xdeadbeef, r7 = 0x88888888`	

Assignment:

instruction	mnemonic
`00000000 0000 00 b7`	`mov r0, 0x0`
`00000000 0000 06 b7`	`mov r6, 0x0`
`00000001 0000 06 07`	`add r6, 0x1`
`00000000 0000 60 0f`	`add r0, r6`
`00000002 0001 06 15`	`jeq r6, 2, +1`
`00000000 fffc 00 05`	`ja -4`
`00000000 0000 00 95`	`exit`
registers: `r0 = 3, r6 = 2`	

(2 P) for instructions, **(1 P)** for registers
(-0.5 P) per mistake

```c
struct vm {
  uint64_t *text;   /* pointer to code */
  uint64_t pc;      /* program counter: index into text */
  uint64_t reg[11]; /* general purpose registers */
};

struct insn {
  uint8_t opcode;
  uint8_t dst, src;
  int16_t off;
  uint32_t imm;
};

/* See table above for an explanation of these constants */
enum {
  /* instruction classes */
  BPF_CLS_ALU = 0x07,
  BPF_CLS_JMP = 0x05,
  /* op values */
  BPF_ALU_ADD = 0x00,
  BPF_ALU_XOR = 0xa0,
  BPF_ALU_MOV = 0xb0,
  BPF_JMP_JA = 0x00,
  BPF_JMP_JEQ = 0x10,
  BPF_JMP_EXIT = 0x90
};
```

b) Vervollständigen Sie die Funktion `decode()`, die eine einzelne eBPF-Instruktion dekodiert. **3 pt**

Complete the function `decode()` which decodes a single eBPF instruction.
Solution:

```
struct insn decode(uint64_t v) {
  struct insn i;

  i.opcode = v & 0xff;
  i.dst = (v >> 8) & 0xf;
  i.src = (v >> 12) & 0xf;
  i.off = (v >> 16) & 0xffff;
  i.imm = (v >> 32);

  return i;
}
```

Declaration and return **(0.5 P)**, *per field* **(0.5 P)**.

c) Vervollständigen Sie die Funktion `select_src()`, die anhand des `s`-Bits im Opcode den Quelloperanden aussucht und dessen Wert zurückgibt. **1.5 pt**

Complete the function `select_src()`, which selects the source operand based on the `s` bit in the opcode and returns its value.
Solution:

```
uint64_t select_src(struct vm *vm, struct insn i) {
  if (i.opcode & 0x8)
    return vm->reg[i.src];
  return i.imm;
}
```

Condition **(0.5 P)**, *source register* **(0.5 P)**, *immediate* **(0.5 P)**

d) Vervollständigen Sie die Funktionen `exec_alu()` und `exec_jmp()`, die jeweils eine ALU- bzw. JMP-Instruktion ausführt. **3.5 pt**

- Durch den Verifizierer ist sichergestellt, dass alle Werte gültig sind.
- Modifizieren Sie das Zielregister bei ALU-Instruktionen über den Pointer `*dst`.

Complete the functions `exec_alu()` and `exec_jmp()`, which execute an ALU or a JMP instruction, respectively.

- *The verifier ensures that all values are valid.*
- *Modify the destination register of ALU instructions through the pointer `*dst`.*

Solution:

```
void exec_alu(struct vm *vm, struct insn i) {
  uint64_t src = select_src(vm, i);
  uint64_t *dst = &vm->reg[i.dst];

  switch (i.opcode & 0xf0) {
  case BPF_ALU_ADD: *dst += src; break;
  case BPF_ALU_XOR: *dst ^= src; break;
  case BPF_ALU_MOV: *dst = src;  break;
  }
}
```

```
void exec_jmp(struct vm *vm, struct insn i) {
  uint64_t src = select_src(vm, i);
  uint64_t dst = vm->reg[i.dst];

  switch (i.opcode & 0xf0) {
  case BPF_JMP_JA:  vm->pc += i.off; break;
  case BPF_JMP_JEQ: if (dst == src) vm->pc += i.off; break;
  }
}
```

dst register (two times) **(0.5 P)**, *mask in switch (two times)* **(0.5 P)**, *5 operations* **(2.5 P)**

e) Vervollständigen Sie die Funktion `run_bpf()`, die ein eBPF-Programm ausführt. **4 pt**

- Gehen Sie davon aus, dass alle Werte in `vm` korrekt initialisiert sind.
- Lesen Sie die nächste Instruktion anhand des Programmzählers `vm->pc`.
- Prüfen Sie die Instruktionsklasse, um `exec_alu()` oder `exec_jmp()` aufzurufen.
- Die `exit`-Instruktion beendet des eBPF-Programm. Geben Sie dann den Wert des Registers `r0` zurück.
- Inkrementieren Sie schließlich den Programmzähler um 1.

Complete the function `run_bpf()`, which runs an eBPF program.

- *Assume that all values in `vm` are initialized correctly.*
- *Read the next instruction via the program counter `vm->pc`.*
- *Check the instruction class and call either `exec_alu()` or `exec_jmp()`.*
- *The `exit` instruction terminates the eBPF program. Return the value of register `r0`.*
- *Finally, increment the program counter by 1.*

Solution:

```
uint64_t run_bpf(struct vm *vm) {
  for (;;) { /* infinite loop */
    struct insn i = decode(vm->text[vm->pc]);
    switch (i.opcode & 0x7) {
    case BPF_CLS_ALU:
      exec_alu(vm, i);
      break;
    case BPF_CLS_JMP:
      if ((i.opcode & 0xf0) == BPF_JMP_EXIT)
        return vm->reg[0];
      exec_jmp(vm, i);
      break;
    }
    vm->pc++;
  }
}
```

Read and decode instruction **(0.5 P)**, *switch on instruction class* **(1 P)**, *call `exec_alu()`* **(0.5 P)**, *call `exec_jmp()`* **(0.5 P)**, *increment `pc`* **(0.5 P)**, *exit handling* **(1 P)**

Total: 15.0pt