Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

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

Nachname/ *Last name* ____________________

Vorname/ *First name* ____________________

Matrikelnr./ *Matriculation no* ____________________

Nachklausur

08.09.2023

- 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 31 Blättern: Einem Deckblatt, 26 Aufgabenblättern mit insgesamt 5 Theorieaufgaben und 3 Programmieraufgaben sowie 4 Seiten Man-Pages.

  The examination consists of 31 pages: One cover sheet, 26 sheets containing 5 theory assignments as well as 3 programming assignments, and 4 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.

PCB: ____________________

____

TCB: ____________________

____

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?

____

____

____

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?

____

____

____

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

Explain the term preemption.

____

____

____

____

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.*

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.*

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.

(+) ______________________________

____

____

(-) ______________________________

____

____

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?*

____

____

____

____

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?

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

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?

Total: 9.0pt

Matrikelnummer/*Matriculation number*

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.

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																											
2																														
3																														
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?

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?

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.

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.*

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.

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.*

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?

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?

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.

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.

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?

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.

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.

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?

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.

File name:

Name of containing directory:

File size:

Number of symbolic links:

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**

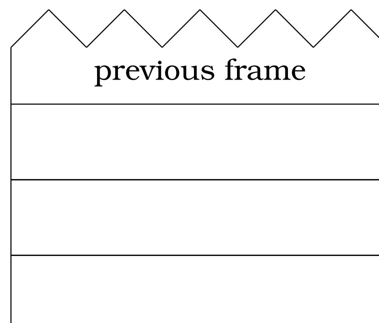
Hinweis: Gesucht ist ein Problem, das zum Absturz des Programms führen kann.

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.*

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.

$f(x) =$

Bedeutung von (*Meaning of*) x:

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).

```
int sum(int n) {
........................................................................
........................................................................
........................................................................
........................................................................
........................................................................
........................................................................
........................................................................
........................................................................
........................................................................
}
```

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;
}
```

____

____

____

____

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)`	
`sizeof(text2)`	
`sizeof(text3)`	
`strlen(text1)`	
`strlen(text2)`	
`strlen(text3)`	

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;
}
```

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.*

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()`.*

```
char *split(char *s, char **part2, int n) {
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
}
```

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.

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

```
#define INDENT "␣␣"
/* include statements for the required C headers */
```

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.

```
struct dirent *next_dir(DIR *s) {
}
```

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()`.

```
int skip_dir(char *name) {
}
```

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:

```
char *join_path(char *path, char *subpath) {
    size_t path_len = strlen(path);
    size_t sub_len  = strlen(subpath);
```

....

....

....

....

....

....

....

....

....

....

....

```
}
```

....

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.

```
int get_props(char *path) {
..........................................................................................
..........................................................................................
..........................................................................................
..........................................................................................
..........................................................................................
..........................................................................................
..........................................................................................
..........................................................................................
..........................................................................................
..........................................................................................
}
..........................................................................................
```

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.

```
void print(int indent, char *path, int is_dir) {
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
.................................................................
}
.................................................................
```

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);


void print_dir(int indent, char *path) {
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
.......................................................................
}
.......................................................................
```

Total: 15.0pt

Aufgabe P3: Virtuelle Maschine für eBPF

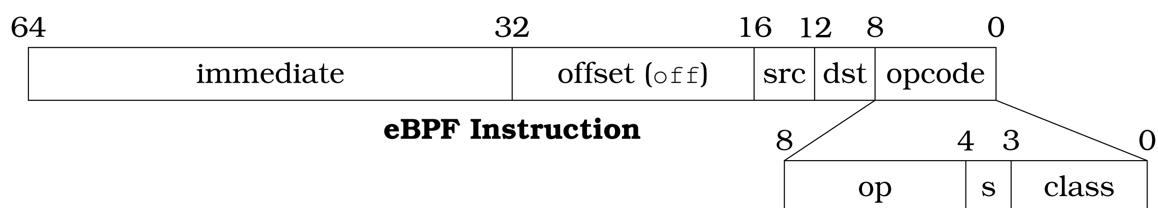
Assignment P3: eBPF Virtual Machine

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.



eBPF 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	
00000000 0000 06 b7	
00000001 0000 06 07	
00000000 0000 60 0f	
00000002 0001 06 15	
00000000 fffc 00 05	
00000000 0000 00 95	
registers:	

```
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.

```
struct insn decode(uint64_t v) {









}
```

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.

```
uint64_t select_src(struct vm *vm, struct insn i) {





}
```

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`.*

```
void exec_alu(struct vm *vm, struct insn i) {
  uint64_t src = select_src(vm, i);
  uint64_t *dst =
  switch (i.opcode &                                                    ) {
  case BPF_ALU_ADD:                                                break;
  case BPF_ALU_XOR:                                                break;
  case BPF_ALU_MOV:                                                break;
  }
}
```

```
void exec_jmp(struct vm *vm, struct insn i) {
  uint64_t src = select_src(vm, i);
  uint64_t dst =
  switch (i.opcode &                                                    ) {
  case BPF_JMP_JA:                                                 break;
  case BPF_JMP_JEQ:


     break;
  }
}
```

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.*

```
struct insn decode(uint64_t v);
void exec_alu(struct vm *vm, struct insn i);
void exec_jmp(struct vm *vm, struct insn i);

uint64_t run_bpf(struct vm *vm) {
  for (;;) { /* infinite loop */














  }
}
```

Total: 15.0pt

NAME

stat, fstat – get file status

SYNOPSIS

#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

int stat(const char ***_pathname_, struct stat ***_buf_**);**
int fstat(int _fd_**, struct stat ***_buf_**);**

DESCRIPTION

These functions return information about a file, in the buffer pointed to by _buf_. No permissions are required on the file itself, but—in the case of **stat**()—execute (search) permission is required on all of the directories in _pathname_ that lead to the file.

stat() retrieves information about the file pointed to by _pathname_.

fstat() is identical to **stat**(), except that the file about which information is to be retrieved is specified by the file descriptor _fd_.

All of these system calls return a _stat_ structure, which contains the following fields:

```
struct stat {
    dev_t     st_dev;         /* ID of device containing file */
    ino_t     st_ino;         /* inode number */
    mode_t    st_mode;        /* file type and mode */
    nlink_t   st_nlink;       /* number of hard links */
    uid_t     st_uid;         /* user ID of owner */
    gid_t     st_gid;         /* group ID of owner */
    dev_t     st_rdev;        /* device ID (if special file) */
    off_t     st_size;        /* total size, in bytes */
    blksize_t st_blksize;     /* blocksize for filesystem I/O */
    blkcnt_t  st_blocks;      /* number of 512B blocks allocated */

    time_t st_atime;  /* Time of last access */
    time_t st_mtime;  /* Time of last modification */
    time_t st_ctime;  /* Time of last status change */
};
```

ERRORS

EACCES
Search permission is denied for one of the directories in the path prefix of _pathname_. (See also **path_resolution**(7).)

EFAULT
Bad address.

ELOOP
Too many symbolic links encountered while traversing the path.

ENAMETOOLONG
pathname is too long.

ENOENT
A component of _pathname_ does not exist or is a dangling symbolic link.

ENOENT
pathname is an empty string and **AT_EMPTY_PATH** was not specified in _flags_.

ENOTDIR
A component of the path prefix of _pathname_ is not a directory.

SEE ALSO

inode(7)

NAME

inode – file inode information

DESCRIPTION

Each file has an inode containing metadata about the file. An application can retrieve this metadata using **stat**(2) (or related calls), which returns a _stat structure_

The following is a list of the information typically found in, or associated with, the file inode, with the names of the corresponding structure fields returned by **stat**(2):

File type and mode
_stat.st_mode_

See the discussion of file type and mode, below.

The file type and mode

The _stat.st_mode_ field contains the file type and mode.

POSIX refers to the _stat.st_mode_ bits corresponding to the mask **S_IFMT** (see below) as the _file type_, the 12 bits corresponding to the mask 07777 as the _file mode bits_ and the least significant 9 bits (0777) as the _file permission bits_.

The following mask values are defined for the file type:

S_IFMT 0170000 bit mask for the file type bit field

S_IFSOCK 0140000 socket
S_IFLNK 0120000 symbolic link
S_IFREG 0100000 regular file
S_IFBLK 0060000 block device
S_IFDIR 0040000 directory
S_IFCHR 0020000 character device
S_IFIFO 0010000 FIFO

Thus, to test for a regular file (for example), one could write:

```
stat(pathname, &sb);
if ((sb.st_mode & S_IFMT) == S_IFREG) {
    /* Handle regular file */
}
```

SEE ALSO

stat(2)

NAME

exit – cause normal process termination

SYNOPSIS

#include <stdlib.h>

noreturn void exit(int *status***);**

DESCRIPTION

The **exit**() function causes normal process termination and the least significant byte of *status* (i.e., *status & 0xFF*) is returned to the parent (see **wait**(2)).

The C standard specifies two constants, **EXIT_SUCCESS** and **EXIT_FAILURE**, that may be passed to **exit**() to indicate successful or unsuccessful termination, respectively.

RETURN VALUE

The **exit**() function does not return.

NOTES

After **exit**(), the exit status must be transmitted to the parent process. There are three cases:

- If the parent has set **SA_NOCLDWAIT**, or has set the **SIGCHLD** handler to **SIG_IGN**, the status is discarded and the child dies immediately.
- If the parent was waiting on the child, it is notified of the exit status and the child dies immediately.
- Otherwise, the child becomes a "zombie" process: most of the process resources are recycled, but a slot containing minimal information about the child process (termination status, resource usage statistics) is retained in process table. This allows the parent to subsequently use **waitpid**(2) (or similar) to learn the termination status of the child; at that point the zombie process slot is released.

NAME

printf, fprintf, dprintf, sprintf – formatted output conversion

SYNOPSIS

#include <stdio.h>

int printf(const char *restrict *format***, ...);**
int fprintf(FILE *restrict *stream***,**
    const char *restrict *format***, ...);**
int dprintf(int *fd***,**
    const char *restrict *format***, ...);**
int sprintf(char *restrict *str***,**
    const char *restrict *format***, ...);**

DESCRIPTION

The functions in the **printf**() family produce output according to a *format*. The function **printf**() writes output to *stdout*, the standard output stream; **fprintf**() writes output to the given output *stream*; **sprintf**() writes to the character string *str*.

The function **dprintf**() is the same as **fprintf**() except that it outputs to a file descriptor, *fd*, instead of to a **stdio**(3) stream.

RETURN VALUE

Upon successful return, these functions return the number of characters printed (excluding the null byte used to end output to strings).

If an output error is encountered, a negative value is returned.

NAME

malloc, free – allocate and free dynamic memory

SYNOPSIS

#include <stdlib.h>

void *malloc(size_t *size***);**
void free(void **ptr***);**

DESCRIPTION

The **malloc**() function allocates *size* bytes and returns a pointer to the allocated memory. *The memory is not initialized*. If *size* is 0, then **malloc**() returns either NULL, or a unique pointer value that can later be successfully passed to **free**().

The **free**() function frees the memory space pointed to by *ptr*, which must have been returned by a previous call to **malloc**(). Otherwise, or if *free(ptr)* has already been called before, undefined behavior occurs. If *ptr* is NULL, no operation is performed.

RETURN VALUE

The **malloc**() function return a pointer to the allocated memory, which is suitably aligned for any built-in type. On error, these functions return NULL. NULL may also be returned by a successful call to **malloc**() with a *size* of zero.

The **free**() function returns no value.

ERRORS

malloc() can fail with the following error:

ENOMEM
Out of memory. Possibly, the application hit the **RLIMIT_AS** or **RLIMIT_DATA** limit described in **getrlimit**(2).

NOTES

By default, Linux follows an optimistic memory allocation strategy. This means that when **malloc**() returns non-NULL there is no guarantee that the memory really is available. In case it turns out that the system is out of memory, one or more processes will be killed by the OOM killer. For more information, see the description of */proc/sys/vm/overcommit_memory* and */proc/sys/vm/oom_adj* in **proc**(5), and the Linux kernel source file *Documentation/vm/overcommit-accounting*.

Normally, **malloc**() allocates memory from the heap, and adjusts the size of the heap as required, using **sbrk**(2). When allocating blocks of memory larger than **MMAP_THRESHOLD** bytes, the glibc **malloc**() implementation allocates the memory as a private anonymous mapping using **mmap**(2). **MMAP_THRESHOLD** is 128 kB by default, but is adjustable using **mallopt**(3). Prior to Linux 4.7 allocations performed using **mmap**(2) were unaffected by the **RLIMIT_DATA** resource limit; since Linux 4.7, this limit is also enforced for allocations performed using **mmap**(2).

To avoid corruption in multithreaded applications, mutexes are used internally to protect the memory-management data structures employed by these functions. In a multithreaded application in which threads simultaneously allocate and free memory, there could be contention for these mutexes. To scalably handle memory allocation in multithreaded applications, glibc creates additional *memory allocation arenas* if mutex contention is detected. Each arena is a large region of memory that is internally allocated by the system (using **brk**(2) or **mmap**(2)), and managed with its own mutexes.

SUSv2 requires **malloc**() to set *errno* to **ENOMEM** upon failure. Glibc assumes that this is done (and the glibc versions of these routines do this); if you use a private malloc implementation that does not set *errno*, then certain library routines may fail without having a reason in *errno*.

Crashes in **malloc**() or **free**() are almost always related to heap corruption, such as overflowing an allocated chunk or freeing the same pointer twice.

The **malloc**() implementation is tunable via environment variables; see **mallopt**(3) for details.

NAME
memmove – copy memory area

LIBRARY
Standard C library (*libc*, −*lc*)

SYNOPSIS
#include <string.h>

void *memmove(void ****dest*, const void ****src***, size_t** *n***);**

DESCRIPTION
The **memmove**() function copies *n* bytes from memory area *src* to memory area *dest*. The memory areas may overlap: copying takes place as though the bytes in *src* are first copied into a temporary array that does not overlap *src* or *dest*, and the bytes are then copied from the temporary array to *dest*.

RETURN VALUE
The **memmove**() function returns a pointer to *dest*.

NAME
strcpy – copy a string

LIBRARY
Standard C library (*libc*, −*lc*)

SYNOPSIS
#include <string.h>

char *strcpy(char *restrict *dst***, const char *restrict** *src***);**

DESCRIPTION
This function copies the string pointed to by *src*, into a string at the buffer pointed to by *dst*. The programmer is responsible for allocating a destination buffer large enough, that is, *strlen(src) + 1*.

RETURN VALUE
This function returns *dst*.

CAVEATS
The strings *src* and *dst* may not overlap.

If the destination buffer is not large enough, the behavior is undefined.

NAME
strcmp, strncmp – compare two strings

SYNOPSIS
#include <string.h>

int strcmp(const char ****s1*, const char ****s2***);**
int strncmp(const char ****s1*, const char ****s2***, size_t** *n***);**

DESCRIPTION
The **strcmp**() function compares the two strings *s1* and *s2*. It returns an integer less than, equal to, or greater than zero if *s1* is found, respectively, to be less than, to match, or be greater than *s2*.

The **strncmp**() function is similar, except it compares only the first (at most) *n* bytes of *s1* and *s2*.

RETURN VALUE
The **strcmp**() and **strncmp**() functions return an integer less than, equal to, or greater than zero if *s1* (or the first *n* bytes thereof) is found, respectively, to be less than, to match, or be greater than *s2*.

NAME
memset – fill memory with a constant byte

SYNOPSIS
#include <string.h>

void *memset(void ****s*, int** *c***, size_t** *n***);**

DESCRIPTION
The **memset**() function fills the first *n* bytes of the memory area pointed to by *s* with the constant byte *c*.

RETURN VALUE
The **memset**() function returns a pointer to the memory area *s*.

NAME
memcpy – copy memory area

SYNOPSIS
#include <string.h>

void *memcpy(void ****dest*, const void ****src***, size_t** *n***);**

DESCRIPTION
The **memcpy**() function copies *n* bytes from memory area *src* to memory area *dest*. The memory areas must not overlap. Use **memmove**(3) if the memory areas do overlap.

RETURN VALUE
The **memcpy**() function returns a pointer to *dest*.

NOTES
Failure to observe the requirement that the memory areas do not overlap has been the source of significant bugs. (POSIX and the C standards are explicit that employing **memcpy**() with overlapping areas produces undefined behavior.) Most notably, in glibc 2.13 a performance optimization of **memcpy**() on some platforms (including x86-64) included changing the order in which bytes were copied from *src* to *dest*.

NAME
strlen – calculate the length of a string

SYNOPSIS
#include <string.h>

size_t strlen(const char ****s*);**

DESCRIPTION
The **strlen**() function calculates the length of the string pointed to by *s*, excluding the terminating null byte ('\0').

RETURN VALUE
The **strlen**() function returns the number of characters in the string pointed to by *s*.

NAME

opendir, closedir – open/close a directory
readdir – read a directory

SYNOPSIS

#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char ***_name_);**
struct dirent *readdir(DIR ***_dirp_);**
int closedir(DIR ***_dirp_);**

DESCRIPTION

The **opendir**() function opens a directory stream corresponding to the directory _name_, and returns a pointer to the directory stream. The stream is positioned at the first entry in the directory.

The **closedir**() function closes the directory stream associated with _dirp_. A successful call to **closedir**() also closes the underlying file descriptor associated with _dirp_. The directory stream descriptor _dirp_ is not available after this call.

The **readdir**() function returns a pointer to a _dirent_ structure representing the next directory entry in the directory stream pointed to by _dirp_. It returns NULL on reaching the end of the directory stream or if an error occurred.

On Linux, the _dirent_ structure is defined as follows:

```
struct dirent {
    ino_t        d_ino;       /* inode number */
    char         d_name[256]; /* filename */
};
```

The data returned by **readdir**() may be overwritten by subsequent calls to **readdir**() for the same directory stream.

RETURN VALUE

The **opendir**() function returns a pointer to the directory stream. On error, NULL is returned, and _errno_ is set appropriately.

On success, **readdir**() returns a pointer to a _dirent_ structure. (This structure may be statically allocated; do not attempt to **free**(3) it.) If the end of the directory stream is reached, NULL is returned and _errno_ is not changed. If an error occurs, NULL is returned and _errno_ is set appropriately.

ERRORS

ENOENT Directory does not exist, or _name_ is an empty string.

ENOTDIR _name_ is not a directory.

EBADF Invalid directory stream descriptor.

NAME

write – write to a file descriptor

SYNOPSIS

#include <unistd.h>

ssize_t write(int _fd_**, const void ***_buf_**, size_t** _count_**);**

DESCRIPTION

write() writes up to _count_ bytes from the buffer pointed _buf_ to the file referred to by the file descriptor _fd_.

The number of bytes written may be less than _count_ if, for example, there is insufficient space on the underlying physical medium, or the **RLIMIT_FSIZE** resource limit is encountered (see **setrlimit**(2)), or the call was interrupted by a signal handler after having written less than _count_ bytes.

For a seekable file (i.e., one to which **lseek**(2) may be applied, for example, a regular file) writing takes place at the current file offset, and the file offset is incremented by the number of bytes actually written. If the file was **open**(2)ed with **O_APPEND**, the file offset is first set to the end of the file before writing. The adjustment of the file offset and the write operation are performed as an atomic step.

RETURN VALUE

On success, the number of bytes written is returned (zero indicates nothing was written). On error, −1 is returned, and _errno_ is set appropriately.

If _count_ is zero and _fd_ refers to a regular file, then **write**() may return a failure status if one of the errors below is detected. If no errors are detected, 0 will be returned without causing any other effect. If _count_ is zero and _fd_ refers to a file other than a regular file, the results are not specified.

ERRORS

EAGAIN
The file descriptor _fd_ refers to a file other than a socket and has been marked nonblocking (**O_NONBLOCK**), and the write would block.

EBADF
_fd_ is not a valid file descriptor or is not open for writing.

EFAULT
buf is outside your accessible address space.

EINTR
The call was interrupted by a signal before any data was written; see **signal**(7).

EINVAL
_fd_ is attached to an object which is unsuitable for writing; or the file was opened with the **O_DIRECT** flag, and either the address specified in _buf_, the value specified in _count_, or the current file offset is not suitably aligned.

EIO A low-level I/O error occurred while modifying the inode.

ENOSPC
The device containing the file referred to by _fd_ has no room for the data.

NOTES

If a **write**() is interrupted by a signal handler before any bytes are written, then the call fails with the error **EINTR**; if it is interrupted after at least one byte has been written, the call succeeds, and returns the number of bytes written.