Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

Prof. Dr. Frank Bellosa
Dipl.-Inform. Marc Rittinghaus

Musterlösung Scheinklausur

15.03.2016

Alle Punkteangaben ohne Gewähr!

- Bitte tragen Sie zuerst auf dem Deckblatt Ihren Namen, Ihren Vornamen und Ihre Matrikelnummer ein. Tragen Sie dann auf den anderen Blättern (auch auf Konzeptblättern) Ihre Matrikelnummer ein.
  Please fill in your last name, your first name, and your matriculation number on this page and fill in your matriculation number on all other pages (including draft pages).
- Die Prüfung besteht aus 15 Blättern: 1 Deckblatt, 14 Aufgabenblättern mit insgesamt 3 Aufgaben und 0 Blättern Man-Pages.
  The examination consists of 15 pages: 1 cover sheet, 14 sheets containing 3 assignments, and 0 sheets for man pages.
- Es sind keinerlei Hilfsmittel erlaubt!
  No additional material is allowed.
- Die Prüfung gilt als nicht bestanden, wenn Sie versuchen, aktiv oder passiv zu betrügen.
  You fail the examination if you try to cheat actively or passively.
- Wenn Sie zusätzliches Konzeptpapier benötigen, verständigen Sie bitte die Klausuraufsicht.
  If you need additional draft paper, please notify one of the supervisors.
- Bitte machen Sie eindeutig klar, was Ihre endgültige Lösung zu den jeweiligen Teilaufgaben ist. Teilaufgaben mit widersprüchlichen Lösungen werden mit 0 Punkten bewertet.
  Make sure to clearly mark your final solution to each question. Questions with multiple, contradicting answers are void (0 points).
- 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	1	2	3	Total
Max. Punkte	18	22	20	60
Erreichte Punkte				
Note				

Aufgabe 1: C Grundlagen

Assignment 1: C Basics

a) Wie werden die folgenden Operationen in der Programmiersprache C geschrieben? Nehmen Sie an, dass die folgenden Definitionen gelten: `int a, b, *c;` **2 pt**

How are the following operations expressed in the C programming language? Assume the following definitions: `int a, b, *c;`

Lösung:

bitwise not of `a`	`~a`
bitwise exclusive or of `a` and `b`	`a ^ b`
address of value pointed to by `c`	`c`
value pointed to by `c`	`*c`

b) Beschreiben Sie, wie sich die Länge einer Zeichenkette (`char*`) ohne die Verwendung von Hilfsfunktionen wie `strlen()` bestimmen lässt. **1 pt**

Explain how the length of a string (`char*`) can be determined without using any helper methods such as `strlen()`.

Lösung:

Strings must be terminated with a `'\0'`-character **(0.5 P)**. *The length can then be determined by counting the number of characters until the `'\0'`-character (e.g., with a `while`-loop)* **(0.5 P)**.

c) Betrachten Sie das C Programm aus Listing 1. In welchen Speichersegmenten liegen die Daten der jeweiligen Symbole bei der Ausführung? **2 pt**

Consider the C program in Listing 1. In which memory segments is the data of the following symbols stored during execution?

```
const char *str = "Hello World!";
int a = 0;

int* add(int p)
{
    int b = p + a;
    return &b;
}
```

Listing 1: Basic C Program

Lösung:

	Text	Stack	Heap	BSS	Data	RO-Data
`str`	☐	☐	☐	☐	☐	■
`a`	☐	☐	☐	■	☐	☐
`p`	☐	■	☐	☐	☐	☐
`add`	■	☐	☐	☐	☐	☐

d) Wieso kann ein Aufrufer der Funktion `add()` aus Listing 1 den Rückgabewert nicht gefahrlos dereferenzieren? **1 pt**

Why is it not safe for a caller of the function `add()` from Listing 1 to dereference the return value?

Lösung:

The variable `b` is stored on the stack of the executing thread. The function therefore provides an address, which might not be accessible after the stack frame of the function is freed on return. Even if the memory is still accessible, the value may have already been overwritten (e.g., through another function call or the definition of further local variables within the caller function). **(1 P)**

e) Erläutern Sie den Unterschied zwischen `unsigned int` und `uint32_t`. **1 pt**

Explain the difference between `unsigned int` and `uint32_t`.

Lösung:

In C, the size of data types depends on the platform for which the code is compiled **(0.5 P)**. *By using `uint32_t` one can define a 32 bit unsigned integer in a platform independent* **(0.5 P)** *way.*

f) Welchen Wert hat die Variable `addr` nach der Ausführung? Begründen Sie Ihre Antwort. **2 pt**

What value does the variable `addr` have after execution? Explain your answer.

```
uint32_t addr = (uint32_t)((uint64_t*)0xffffffff + 0x01);
```

Lösung:

The correct value is `0x00000007` **(1 P)**. *By casting the address `0xffffffff` to a pointer, the addition increments the address by the size of the pointer data type (pointer arithmetic), which is 8 (bytes) for a 64 bit pointer* **(0.5 P)**. *This gives `0x0000000100000007`. By casting the value back to a 32 bit unsigned integer, the high 32 bits are truncated* **(0.5 P)**.

g) Erklären Sie, warum es in C wichtig ist, lokale Variablen vor der ersten Verwendung explizit zu initialisieren. **2 pt**

Explain why it is important in C to explicitly initialize local variables before their first use.

Lösung:

In C, all local variables contain random values before they are initialized. **(1 P)** *This is because the value of the variable will thus correspond to whatever was on the stack before in the space now occupied by the new variable.* **(1 P)** *Since the program cannot know what data was on the stack before, it cannot rely on the variable holding any particular value. The only safe way of dealing with this is to overwrite the variable with a defined value.*

h) Erklären Sie, warum es auf Systemen mit wenig RAM bei der Verwendung von Rekursion zu Problemen kommen kann. **1 pt**

Explain why the use of recursion can lead to problems on systems with little RAM.

Lösung:

Recursion means that a function calls itself, potentially many times. Each of these calls will allocate a new stack frame. **(0.5 P)** *If the system has little RAM, these stack frames could easily exhaust the available memory.* **(0.5 P)**

i) Erklären Sie, was unter einem Void-Pointer (`void*`) zu verstehen ist. **1 pt**

Explain the meaning of a void-pointer (`void*`).

Lösung:

A void-pointer is a pointer pointing to data of an unspecified type.

j) C unterscheidet beim Datentyp `char` zwischen *signed* und *unsigned*. Ist diese Unterscheidung sinnvoll? Begründen Sie Ihre Antwort. **1 pt**

For the data type `char`, C distinguishes between signed *and* unsigned*. Does this distinction make sense? Explain your answer.*

Lösung:

Yes, it makes sense because chars can not only be used as actual characters, but also as one byte numbers. **(1 P)** *In the latter case, the distinction makes sense, as it does for any other numerical data type.*

k) Welche der folgenden Aussagen sind richtig?
(falsches Kreuz: -1P, kein Kreuz: 0P, korrektes Kreuz: 1P) **4 pt**

Which of the following statements are correct?
(incorrectly marked: 1P, not marked: 0P, correctly marked: 1P)

korrekt/ *correct*	inkorrekt/ *incorrect*	
☐	■	`(a >> 2) == (a / 2)`
■	☐	`sizeof(int) == sizeof(unsigned int)`
■	☐	Ein C-`struct` kann ein Array als Feld enthalten. *A C `struct` can contain an array as a member.*
☐	■	`free(NULL)` führt zum Absturz des Programms. *`free(NULL)` leads to a program crash.*

Total: 18.0pt

Aufgabe 2: Druckerserver

Assignment 2: Printer Spooler

Schreiben Sie einen Druckerserver (*spooler*) und einen Client, der Aufträge an den Server schickt.

Client und Server nutzen POSIX Message Queues und temporäre Dateien, um Druckaufträge und die zu druckenden Daten auszutauschen.

Beim Starten initialisiert der Druckerserver zunächst eine POSIX Message Queue und wartet dann auf eingehende Nachrichten. Um einen Druckauftrag abzusetzen, speichert der Client die zu druckenden Daten in eine temporäre Datei und sendet dann den Namen der temporären Datei über die Message Queue an den Druckerserver. Sobald der Druckerserver eine Nachricht empfängt, liest er den Druckauftrag aus der temporären Datei und schreibt die Daten in die Character-Device Datei `/dev/lp0` des Druckers.

Binden Sie in allen Teilaufgaben die notwendigen C-Header ein und lassen Sie niemals vom Betriebssystem angeforderte Objekte (z.B. Dateideskriptoren) ungenutzt zurück (*leak*).

Write a printer spooler and a client for sending jobs to the spooler.

The client and spooler make use of POSIX message queues and temporary files to exchange print requests and the print jobs' data, respectively.

When starting up, the printer spooler first initializes a POSIX message queue and then waits for incoming messages. For launching a print job, a client stores the job's data in a temporary file and then sends the file name of the temporary file for printing to the spooler via the message queue. Once the spooler receives a request, it reads the print job from the temporary file and writes it to the printer's character device file `/dev/lp0`*.*

In all assignments, include necessary C headers and never leak allocated operating system objects (e.g., file descriptors).

a) Wie trägt ein *Spooler* (so wie der Druckerserver in dieser Aufgabe) zur Vermeidung von Deadlocks *(deadlock prevention)* bei? **2 pt**

How does a spooler *(e.g., for a printer, as in this assignment) contribute to deadlock prevention?*

Lösung:

A spooler avoids that clients that want to use a resource (e.g., a printer) need to acquire exclusive access to that resource. **(1 P)** *Thereby, a spooler breaks one of the necessary conditions for deadlocks for the resource it manages, which results in the prevention of deadlocks.* **(1 P)**

b) Vervollständigen Sie die Funktion `init_message_queue()` für den Server, die eine POSIX Message Queue mit dem Namen `\exam-print-spooler` anlegt und öffnet.

- Die maximale Nachrichtengröße der Message Queue soll `MAX_PATH` betragen.
- Die Puffergröße der Message Queue soll acht Nachrichten betragen.
- Die Funktion gibt ein Handle auf die neu erzeugte Message Queue zurück.
- Nehmen Sie an, dass keine Fehler auftreten.
- Beachten Sie, dass der Druckerserver und die Clients unter verschiedenen Benutzer IDs laufen können.

4 pt

Complete the function `init_message_queue()` for the printer spooler, which creates and opens a POSIX message queue called `\exam-print-spooler`.

- *The maximum size of messages to be accepted into the queue should be `MAX_PATH`.*
- *The queue should store up to eight messages in its buffer.*
- *The function returns a handle to the newly created message queue.*
- *Assume that no errors occur.*
- *Keep in mind that printer spooler and clients might run with different user ids.*

Lösung:

```
#include <fcntl.h>
#include <sys/stat.h> // not needed with numeric constant for mode
#include <mqueue.h>

#define MAX_PATH 256

mqd_t init_message_queue(void) {
    struct mq_attr attr = {0};
    mqd_t mq;

    /* flags, maxmsg, msgsize, curmsgs */
    attr.mq_msgsize = MAX_PATH;
    attr.mq_maxmsg  = 8;
    /* others set to zero by initializer above */

    mq = mq_open("/exam-print-spooler",
        O_RDWR | O_CREAT, 0666, &attr);
    /* mode 0722 is ok, as well */

    return mq;
}
```

Includes **(0.5 P)**, *initializing attributes* **(1 P)**, *opening the message queue* **(2 P)**, *returning the handle* **(0.5 P)**; *downgrading for leaking malloc* **(-0.5 P)**.

c) Wie können Sie beeinflussen, ob sich die Sende- und Empfangsoperationen einer POSIX Message Queue synchron oder asynchron verhalten?

Kann sich dieses Verhalten zwischen Sender und Empfänger einer Nachricht unterscheiden? Begründen Sie Ihre Antwort kurz.

2 pt

How do you control if send and receive operations on a POSIX message queue should be synchronous or asynchronous?

Can this behavior be different between sender and receiver of a message? Briefly explain your answer.

Lösung:

The flag `O_NONBLOCK` *in the call to* `mq_open()` **(1 P)** *controls whether send and receive should behave asynchronous (a.k.a. non-blocking, when the flag is set) or synchronous (when the flag is omitted).*

Yes, this setting can differ between sender and receiver **(0.5 P)**, *as each process may call* `mq_open()` *with individual flags.* **(0.5 P)**

d) Sollte der Druckerserver im gegebenen Szenario synchrone oder asynchrone Empfangsoperationen nutzen? Begründen Sie Ihre Antwort. **1 pt**

In the given scenario, should the printer spooler use synchronous or asynchronous receive operations? Explain your answer.

Lösung:

The spooler should synchronously receive messages (i.e., blocking wait for new messages). **(0.5 P)**

The printer spooler has no other tasks than servicing print jobs, so it cannot do meaningful work instead of waiting for new jobs. Repeated asynchronous receives (i.e., polling for new messages) therefore would only waste CPU time. **(0.5 P)**

e) Vervollständigen Sie die Funktion `send_print_job()`, mit der ein Client einen Druckauftrag an den Server schicken an.

- Die Funktion `store_data_to_tmp()` speichert die zu druckenden Daten in einer temporären Datei und liefert den Pfad zu dieser Datei zurück. Der Speicher des Pfads muss nicht freigegeben werden.
- Senden Sie den Pfadnamen der temporären Datei (sonst nichts) als Druckauftrag über die Message Queue `\exam-print-spooler` an den Druckerserver.
- Die Priorität der Nachricht in der Message Queue soll `0` betragen.
- Die Funktion gibt bei Erfolg `0` zurück, `-1` andernfalls.

3 pt

Complete the function `send_print_job()`, *which allows a client to send a print job to the spooler.*

- *The function* `store_data_to_tmp()` *stores the print job's data in a temporary file and returns the path name of that file. The path name does not need to be freed.*
- *Send the path name of the temporary file (nothing else) as a request to the printer spooler via the message queue* `\exam-print-spooler`.
- *The priority of the message in the queue should be* `0`.
- *The function returns* `0` *on success,* `-1` *otherwise.*

Lösung:

```c
#include <fcntl.h>
/* unnecessary: #include <sys/stat.h> */
#include <mqueue.h>

char* store_data_to_tmp(char *jobname, void *jobdata, size_t joblength);

#define MAX_PATH 256

int send_print_job(char *jobname, void *jobdata, size_t joblength) {
    char *path = store_data_to_tmp(jobname, jobdata, joblength);
    mqd_t mq;
    int r;

    mq = mq_open("/exam-print-spooler", O_WRONLY);
    if (mq == (mqd_t)-1)
        return -1;

    r = mq_send(mq, path, strlen(path), 0);

    mq_close(mq);

    return r;
}
```

Includes **(0.5 P)**, *opening the message queue* **(0.5 P)**, *sending the path* **(1 P)**, *closing the handle* **(0.5 P)**, *error handling and return value* **(0.5 P)**; *downgrading when setting O_NONBLOCK without handling EAGAIN* **(-0.5 P)**.

f) Vervollständigen Sie die `main()`-Funktion des Druckerservers, die zunächst die Initialisierungsfunktion aus Teilaufgabe b) aufruft und dann in einer Endlosschleife eingehende Druckaufträge verarbeitet.

- Die Funktion empfängt Druckaufträge über die Message Queue und sendet sie über die Funktion `print_job()` an den Drucker.
- Nehmen Sie an, dass keine Fehler auftreten können.

3 pt

Write the spooler's `main()`-function, which first calls the initialization function from assignment b) and then enters an infinite loop servicing client requests.

- *The function receives print jobs from the message queue and sends them to the printer by calling `print_job()`.*
- *Assume that no errors occur.*

Lösung:

```c
#include <mqueue.h>

int print_job(char *jobpath);

int main(int argc, char *argv[]) {
    mqd_t mq = init_message_queue();
    char jobpath[MAX_PATH + 1];

    do {
        memset(jobpath, 0, sizeof(jobpath));
```

```
        mq_receive(mq, jobpath, sizeof(jobpath) - 1, NULL);

        print_job(jobpath);
    } while (1);

    return 0;
}
```

Includes **(0.5 P)**, *calling initialization method* **(0.5 P)**, *defining a buffer to receive the path* **(0.5 P)**, *only when in infinite loop: receiving the print job* **(0.5 P)**, *ensuring that the path is null terminated* **(0.5 P)**, *calling the print function* **(0.5 P)**.

g) Vervollständigen Sie die Funktion `print_job()`, die einen Druckauftrag aus einer temporären Datei liest und ihn an den Drucker schickt, indem sie die Daten in die Character-Device Datei `/dev/lp0` schreibt.

- Behandeln Sie den Fall, dass die temporäre Datei des Druckauftrags oder die Gerätedatei des Druckers nicht geöffnet werden können.
- Nehmen Sie an, dass keine Fehler auftreten können, wenn Sie Dateien lesen oder schreiben.
- Beachten Sie, dass die Systemaufrufe `read()` und `write()` zurückkehren können, obwohl sie weniger Bytes gelesen bzw. geschrieben haben, als angefordert.
- Die Funktion gibt bei Erfolg `0` zurück, `-1` andernfalls.

7 pt

Complete the function `print_job()`, *which reads a print job from a temporary file and sends it to the printer by writing it to the printer's character device file* `/dev/lp0`.

- *Handle the case that the temporary file or the device file cannot be opened.*
- *Assume that no errors can occur when reading or writing to files.*
- *Keep in mind that the* `read()` *and* `write()` *system calls may return with fewer bytes read/written than requested.*
- *The function returns* `0` *on success,* `-1` *otherwise.*

Lösung:

```
#include <fcntl.h>
/* unnecessary: #include <sys/stat.h>  */
/* unnecessary: #include <sys/types.h> */
#include <unistd.h>

int print_job(char *jobpath) {
    int jobfd, lpfd;
    char buf[512];
    ssize_t bytes_read;

    jobfd = open(jobpath, O_RDONLY);
    if (jobfd == -1)
        return -1;

    lpfd = open("/dev/lp0", O_APPEND | O_WRONLY);
    if (lpfd == -1) {
        close(jobfd);
        return -1;
    }
```

```
    while ((bytes_read = read(jobfd, buf, 512)) != 0) {
        char *bufpos = buf;

        while (bytes_read > 0) {
            ssize_t bytes_written;
            bytes_written = write(lpfd, bufpos, bytes_read);

            bytes_read -= bytes_written;
            bufpos += bytes_written;
        }
    }

    close(jobfd);
    close(lpfd);

    return 0;
}
```

Includes **(0.5 P)**, *defining temporary buffer* **(0.5 P)**, *opening job file* **(0.5 P)**, *opening character device* **(0.5 P)**, *error handling* **(1.5 P)**, *read-loop* **(0.5 P)**, *reading from job file* **(0.5 P)**, *write-loop* **(0.5 P)**, *writing to character device* **(0.5 P)**, *handling the buffer pointer* **(0.5 P)**, *closing file descriptors* **(0.5 P)**, *returning right values* **(0.5 P)**. *(Alternatively, a single loop reading and writing 1 B each iteration works as well, much less efficient, though).*

Total: 22.0pt

Aufgabe 3: Speicherverwaltung

Assignment 3: Memory Management

Betrachten Sie ein System mit 31 bit virtuellen und 45 bit physischen Adressen. Die Seitengröße beträgt 8 KiB. Zur Übersetzung kommt ein durch Software gefüllter TLB zum Einsatz.

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

Nutzen Sie zur Lösung der Teilaufgaben die folgenden Typen, Konstanten und Funktionen:

Consider a system with 31 bit virtual and 45 bit physical addresses. The page size is 8 KiB. The system uses a software-filled TLB for address translation.

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

For solving the questions, use the following types, constants, and functions:

```
typedef struct _vma_t {
    struct _vma_t *next; // Pointer to next VMA in list
    uint32_t start;      // Number of first virtual page in VMA
    uint32_t end;        // Number of last virtual page in VMA
} vma_t;

typedef struct {
    vma_t *vmas;         // Unsorted list of VMAs in address space
    page_dir_t pdir;     // Page directory
} vas_t;

// Constant to return or set for invalid page frame numbers (PFNs)
#define INVALID_PFN ((uint32_t)(-1))

// Allocates a number of bytes from kernel virtual memory
// (not zero-initialized). Returns a pointer to the newly allocated
// memory on success, NULL otherwise.
void* kmalloc(size_t bytes);

// Allocates a free physical page frame (zero-initialized).
// Returns the page frame number (PFN) on success,
// INVALID_PFN otherwise.
uint32_t allocFrame(void);

// Frees a physical page frame previously allocated by allocFrame().
// Calling freeFrame() with INVALID_PFN is undefined behavior.
void freeFrame(uint32_t pfn);

// Flushes the software-filled TLB, thereby invalidating all TLB entries.
void flushTlb(void);
```

a) Definieren Sie die folgenden Strukturen für eine zweistufige Seitentabelle zur Übersetzung von virtuellen in physische Adressen in dem gegebenen System. Beginnen Sie mit der Berechnung der Anzahl von Einträgen pro Tabellenstufe.

- Tabellen der zweiten Ebene sollen nur bei Bedarf alloziert werden.

4 pt

Define the following structures for a two-level page table for the translation of virtual to physical addresses in the given system. Start with calculating the number of entries per page table level.

- *Second-level page tables should be allocated on demand only.*

Lösung:

Before defining the two-level page table, we need to calculate how many entries in each level will be needed. For the 31 bit virtual address space with 8 KiB pages, we have $\frac{2^{31}}{2^{13}} = 2^{18}$ *pages. A two-level page table will thus have* $2^9 = 512$ *entries in each level (*$2^9 * 2^9 = 2^{18}$*)* **(1 P)**.
We also have to determine how to structure the page table hierarchy. The first-level page table (the page directory) must point to the second-level page tables. An array of 512 pointers to second-level page tables is thus reasonable **(1 P)**. *Not using pointers, but an array of full second-level page tables would defeat the purpose of the multi-level page table. The second-level page tables need to contain 512 PTEs each* **(1 P)**. *Since we do not have any additional requirements than storing the mapping, a PTE only needs to contain the physical frame number (PFN)* **(0.5 P)**. *A frame is 8 KiB in size. The offset is thus 13 bits and a single PFN requires 32 bits. We can therefore use the* `uint32_t` *type for a PFN (this type is also used in the support routines such as* `allocFrame()`*)* **(0.5 P)**. *A complete page table hierarchy should look like this:*

```
typedef struct {
    uint32_t pfn;
} pte_t;

typedef struct {
    pte_t ptes[512];
} page_table_t;

typedef struct {
    page_table_t *tables[512];
} page_dir_t;
```

b) Vervollständigen Sie die Funktion `allocVas()`, die einen neuen Adressraum (`vas_t`) alloziert, initialisiert und zurückgibt.

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

2 pt

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

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

Lösung:

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

    vas->vmas = NULL;
    for (int i = 0; i < 512; ++i)
        vas->pdir.tables[i] = NULL;

    return vas;
}
```

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

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

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

3 pt

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

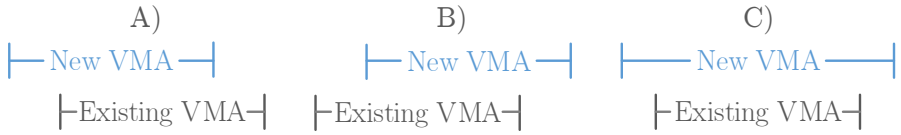
- *The operation fails if the supplied area overlaps with an existing area.*
- *The function returns* `-1` *on error,* `0` *otherwise.*

Lösung:

```
int addVma(vas_t *vas, vma_t *vma) {
    for (vma_t *cur = vas->vmas; cur; cur = cur->next) {
        if (!((vma->start > cur->end) ||
              (vma->end < cur->start)))
            return -1;
    }

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

Loop **(0.5 P)**, *checking for overlap with any existing area* **(1.5 P)**. *We can also check for the three cases (***(0.5 P)** *for each case):*




```
(((vma->end >= cur->start) &&   // Case A
  (vma->end <= cur->end)) ||
 ((vma->start >= cur->start) && // Case B
  (vma->start <= cur->end)) ||
 ((vma->start <= cur->start) && // Case C
  (vma->end >= cur->end)))
```

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

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

- Die Funktion liefert `NULL`, wenn keine Seitentabelle für die Adresse alloziert ist.
- Gehen Sie davon aus, dass Bit 32 in `vaddr` stets 0 ist.

3 pt

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

- *The function returns `NULL` if no page table exists for the virtual address.*
- *Assume that bit 32 of `vaddr` is always 0.*

Lösung:

```
pte_t* getPte(vas_t *vas, uint32_t vaddr) {
    uint32_t idx1 = vaddr >> 22;
    uint32_t idx2 = (vaddr >> 13) & 0x1ff;
    if (vas->pdir.tables[idx1])
        return &vas->pdir.tables[idx1].ptes[idx2];

    return NULL;
}
```

Extracting the first level table index **(0.5 P)**, *extracting the second level table index* **(1 P)**, *checking if the second level table exists* **(0.5 P)**, *returning address of PTE* **(0.5 P)**, *returning NULL* **(0.5 P)**

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

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

3 pt

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

- *You can use the function `getPte()`.*
- *Dynamically allocated page tables do not need to be released.*
- *Keep in mind that not all page tables must be allocated and that not all PTEs must contain valid mappings to physical pages.*

Lösung:

```
void invalidateVma(vas_t *vas, vma_t *vma) {
    for (uint32_t p = vma->start; p <= vma->end; ++p) {
        pte_t *pte = getPte(vas, p << 13);
        if (pte && (pte->pfn != INVALID_PFN)) {
            freeFrame(pte->pfn);
            pte->pfn = INVALID_PFN;
        }
    }

    flushTlb();
}
```

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

f) Vervollständigen Sie die Funktion `pagefault()`, die bei einem Seitenfehler aufgerufen wird und für die zugegriffene virtuelle Seite eine gültige Abbildung auf eine leere physische Seite herstellt.

- Die Adresse der virtuellen Seite ist in einen Index `idx1` in die erste Ebene und einen Index `idx2` in die zweite Ebene der Seitentabelle zerlegt worden.
- Der Zugriff auf eine Adresse außerhalb eines gültigen Speicherbereichs (VMA) ist ein Fehlerfall.
- Die Funktion liefert die Nummer der physischen Seite (PFN) bei Erfolg und `INVALID_PFN` im Fehlerfall.

5 pt

Complete the function `pagefault()`, *which is called on a page fault and establishes a mapping from the accessed virtual page to an empty physical page.*

- *The address of the virtual page has been split into an index* `idx1` *into the first level and an index* `idx2` *into the second level of the page table.*
- *Accessing an address that is not within a valid memory area (VMA) is an error.*
- *The function returns the number of the physical page (PFN) on success,* `INVALID_PFN` *on error.*

Lösung:

```
uint32_t pagefault(vas_t *vas, uint32_t idx1, uint32_t idx2) {
    uint32_t vpn = (idx1 << 9) | idx2;

    for (vma_t *vma = vas->vmas; vma; vma = vma->next) {
        if ((vpn >= vma->start) &&
            (vpn <= vma->end)) {
            page_table_t **t = &vas->pdir.tables[idx1];

            if (!*t) {
                *t = kmalloc(sizeof(page_table_t));
                if (*t == NULL)
                    return INVALID_PFN;

                for (int i = 0; i < 512; ++i)
                    *t->ptes[i].pfn = INVALID_PFN;
            }

            uint32_t pfn = allocFrame();
            *t->ptes[idx2].pfn = pfn;
            return pfn;
        }
    }

    return INVALID_PFN;
}
```

Calculating vpn **(1 P)**, *finding/checking for valid VMA* **(1 P)**, *allocating a page table if necessary* **(0.5 P)**, *initializing new page table* **(0.5 P)**, *allocating a new frame* **(0.5 P)**, *setting the new mapping* **(0.5 P)**, *returning the new pfn* **(0.5 P)**, *error handling* **(0.5 P)**

Total: 20.0pt