Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

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

Musterlösung Scheinklausur

22.03.2018

Alle Punkteangaben ohne Gewähr!

- Bitte tragen Sie zuerst auf dem Deckblatt Ihren Namen, Ihren Vornamen und Ihre Matrikelnummer ein. Tragen Sie dann auf den anderen Blättern (auch auf Konzeptblättern) Ihre Matrikelnummer ein.
  Please fill in your last name, your first name, and your matriculation number on this page and fill in your matriculation number on all other pages (including draft pages).
- Die Prüfung besteht aus 18 Blättern: 1 Deckblatt, 17 Aufgabenblättern mit insgesamt 3 Aufgaben und 0 Blättern Man-Pages.
  The examination consists of 18 pages: 1 cover sheet, 17 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.
- Sie können auch die Rückseite der Aufgabenblätter für Ihre Antworten verwenden. Wenn Sie zusätzliches Konzeptpapier benötigen, verständigen Sie bitte die Klausuraufsicht.
  You can use the back side of the task sheets for your answers. If you need additional draft paper, please notify one of the supervisors.
- Bitte machen Sie eindeutig klar, was Ihre endgültige Lösung zu den jeweiligen Teilaufgaben ist. Teilaufgaben mit widersprüchlichen Lösungen werden mit 0 Punkten bewertet.
  Make sure to clearly mark your final solution to each question. Questions with multiple, contradicting answers are void (0 points).
- 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	20	20	20	60
Erreichte Punkte				
Note				

Aufgabe 1: C Grundlagen

Assignment 1: C Basics

a) Betrachten Sie die folgende Funktion `read_input()`. Nehmen Sie an, dass die Datei, auf die der Filedeskriptor `fd` verweist, die folgende Folge von Zeichen enthält: a, b, c, d.

Consider the following function `read_input()`. Assume that the file which the descriptor `fd` refers to contains the following sequence of characters: a, b, c, d.

```
void read_input(int fd, char **buf) {
  *buf = malloc(1024);
  memset(*buf, 0, 1024);
  lseek(fd, 3, SEEK_SET);
  read(fd, *buf, 1023);
}
```

Welchen String enthält der Buffer `buf`, nachdem der `read()`-Aufruf zurückgekehrt ist? Nehmen Sie an, dass keine Fehler auftreten. **1 pt**

Which string does the buffer `buf` contain after the `read()` call returns? Assume that no errors occur.

Lösung:

`buf` contains the string "d" **(1.0 P)**. *The `lseek` call moves `fd`'s file pointer three bytes to the right, thus pointing at the character d. The `read()` call then attempts to read 1024 bytes starting at that byte, but encounters the end of the file after reading the first character. Therefore, a single d is written to the beginning of `buf`, and the rest of the buffer – which contains zeroes due to the `memset` call – is left untouched. As strings in C are terminated by 0 bytes, the buffer thus contains a valid string consisting only of the letter d.*

Was ist der Zweck des `memset()`-Aufrufs? **2 pt**

What is the purpose of the `memset()` call?

Lösung:

Strings in C must be terminated by a null byte. but `read()` will not append such a null byte to the data it has read. Therefore, the function must ensure that a null byte is appended to whatever `read()` writes into `buf`. **(1.0 P)** *One way to achieve this is `memset()`: Filling the buffer with zeroes guarantees that `buf` will always contain a valid, null-terminated string since whatever `read()` writes into the buffer will always be followed by a null byte.* **(1.0 P)**

Nehmen Sie an, mehrere Threads rufen gleichzeitig `read_input()` auf. Dabei übergeben alle Threads denselben Wert für `fd`, aber nicht für `buf`. Welches Problem kann in diesem Fall auftreten? **2 pt**

Assume that multiple threads simultaneously call `read_input()`, passing the same value for `fd` but not for `buf`. Which problem can occur in that case?

Lösung:

All threads operate on the same file handle. However, each file handle only has a single file pointer. **(1.0 P)** *Therefore, each thread's `lseek()` call will move the file pointer for all other threads as well. The threads thus cannot predict which location in the file they will actually read from.* **(1.0 P)**

b) Ein Programm verwendet die folgende Funktion, um zu verhindern, dass geheime Daten aus freigegebenen Puffern im Speicher zurückbleiben. Obwohl nach dem `free()`-Aufruf noch einmal auf den Puffer `buf` zugegriffen wird, funktioniert die Funktion zunächst wie gewünscht. Erklären Sie, warum. **1 pt**

A program uses the following function to ensure that secret data from freed buffers does not remain in memory. Even though the buffer `buf` is accessed after the `free()` call, the function works as intended. Explain why.

```
void secure_free(void *buf, size_t size) {
  free(buf);
  memset(buf, 0, size);
}
```

Lösung:

`free()` does not return freed memory to the operating system immediately. Instead, each program has a pool of allocated but unused memory. `free()` simply returns memory to that pool; subsequent calls to `malloc()` attempt to allocate from that pool to reduce the number of system calls. Since the memory in this pool remains mapped in the application's address space, accesses to this memory will typically succeed.

Nachdem das Programm um mehrere Threads erweitert wird, treten gelegentlich Abstürze auf. Welches Problem verursacht wahrscheinlich diese Abstürze? **1 pt**

After the program is modified to use multiple threads, the program starts to crash occasionally. Which problem is probably causing these crashes?

Lösung:

Freed memory can be reused by subsequent `malloc()` calls. With multiple threads, such a `malloc()` call can happen before the `memset()` call in `secure_free()` returns. In that case, `memset()` can overwrite data from the allocating thread, leading to a crash.

c) Wieviele neue Prozesse startet die folgende Funktion? **1 pt**

How many new processes does the following function start?

```
void start_processes() {
  fork();
  fork();
  fork();
  return;
}
```

Lösung:

Seven **(1.0 P)**. *After each* `fork()`*, all following* `fork()`*s will be executed by all child processes. Every call to* `fork()` *thus doubles the total number of processes. After three calls, there are therefore* $2^3 = 8$ *processes in existence. However, one of these eight processes is the one* `start_processes()` *was originally called in. Therefore, only seven processes are actually new.*

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

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

```
&A[42]
```

Lösung:

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

e) Ist es in C problemlos möglich, einen Zeiger nach `int` zu casten? Begründen Sie Ihre Antwort. **1.5 pt**

In C, is it possible to cast a pointer to `int` *without causing problems? Justify your answer.*

Lösung:

No. **(0.5 P)** *An* `int` *is typically 32 bits long. Depending on the hardware architecture, a pointer may be longer than 32 bits. In that case, the upper 32 bits of the pointer are discarded during the cast, which may lead to segmentation faults later on if the int is cast back to a pointer.* **(1.0 P)**

f) In dieser Aufgabe sollen Sie eine einfache Hashtabelle implementieren.

- Die Tabelle verwendet den Datentyp `int` sowohl für Schlüssel (`key`) als auch für Werte (`value`).
- Die Position jedes Elements im Array entspricht dem Schlüssel des Elements modulo der Anzahl der Felder des Arrays (`LIST_SIZE`).
- Für den Fall, dass sich beim Einfügen eines Elements bereits ein anderes Element an der Zielposition befindet, verwendet die Tabelle *linear probing*: Beginnend mit der Zielposition werden alle Einträge des Arrays linear durchsucht, bis ein freier Eintrag gefunden wird.

In this problem, you are to implement a simple hash table.

- *The table uses the data type* `int` *for both keys and values.*
- *The position of each element in the array is the element's key modulo the number of slots in the array (*`LIST_SIZE`*).*
- *If an element is to be inserted into a slot that already contains a valid element, the table uses* linear probing*: Starting with the calculated slot, all slots in the array are linearly searched until a free slot is found.*

Definieren Sie zunächst eine Datenstruktur, die ein einzelnes Element der Tabelle repräsentiert. **1.5 pt**

First, define a data structure representing a single table element.

Lösung:

```
struct element {
  int valid;
  int key;
  int value;
};
```

(0.5 P) *for each member*

Vervollständigen Sie die Funktion `insert()`, die ein Element in die Tabelle einfügt. **3.5 pt**

- Nehmen Sie an, dass freie Einträge gemäß Ihrer Definition von `struct element` korrekt initialisiert sind, dass noch kein Eintrag mit dem selben Schlüssel existiert und dass mindestens ein freier Eintrag existiert.

Complete the function `insert()` which inserts an element into the table.

- *Assume that unused entries are initialized correctly according to your definition of `struct element`, that no entry with the same key exists and that at least one unused entry exists.*

Lösung:

```
void insert(int key, int value) {
  int index = key % TABLE_SIZE;
  int i, cur;

  for (i = 0; i < TABLE_SIZE; i++) {
    cur = (index + i) % TABLE_SIZE;
    if (table[cur].valid == 0) {
      table[cur].key = key;
      table[cur].value = value;
      table[cur].valid = 1;
      return;
    }
  }

  return; //this line should never be reached
}
```

- **(0.5 P)** *for computing the starting point*
- **(0.5 P)** *for the loop*
- **(1.0 P)** *for correctly wrapping around at the end of the table*
- **(0.5 P)** *for checking for unused elements*
- **(0.5 P)** *for correctly inserting key and value*
- **(0.5 P)** *for making the new element valid*

Vervollständigen Sie die Funktion `find()`, die ein Element mit einem bestimmten Schlüssel aus der Tabelle liest. **4.5 pt**

- Der Rückgabewert der Funktion soll 1 sein, wenn ein entsprechendes Element gefunden wurde, und 0 falls nicht.
- Falls ein Element gefunden wurde, soll die Funktion seinen Wert im Parameter `value` ablegen.
- Ihre Implementierung soll so wenige Arrayelemente wie möglich durchsuchen.

Complete the function `find()` which reads an element with a given key from the table.

- *The function shall return 1 if an element with the given key was found and 0 if not.*
- *If an element was found, the function shall store the element's value into the parameter `value`.*
- *Your implementation shall search as few array elements as possible.*

Lösung:

```
int find(int key, int *value) {
  int index = key % TABLE_SIZE;
  int i, cur;

  for (i = 0; i < TABLE_SIZE; i++) {
    cur = (index + i) % TABLE_SIZE;
    if (table[cur].valid == 0) {
      /* if an element with the correct key existed,
         it would have been inserted here */
      return 0;
    }

    if (table[cur].key == key) {
      *value = table[cur].value;
      return 1;
    }
  }

  return 0;
}
```

- **(0.5 P)** *for computing the starting point*
- **(0.5 P)** *for the loop*
- **(1.0 P)** *for correctly wrapping around at the end of the table*
- **(1.0 P)** *for aborting on the first invalid element*
- **(0.5 P)** *for checking the key*
- **(0.5 P)** *for returning the value*
- **(0.5 P)** *for returning whether an element has been found*

Total: 20.0pt

Aufgabe 2: Dateisystemgröße

Assignment 2: File System Size

Schreiben Sie ein Programm, das rekursiv den Platzverbrauch aller Dateien in einem Verzeichnis und seinen Unterordnern auf der Festplatte bestimmt.

- Das Programm soll Hardlinks anhand der I-Node-Nummer erkennen, sodass der Platzverbrauch von Dateien nur einmal pro I-Node gezählt wird.
- Das Programm wird mithilfe einer festen Anzahl von Threads parallelisiert, indem Unterverzeichnisse in eine zentrale Warteschlange eingetragen werden, die von den Threads abgearbeitet wird. Verwenden Sie, wo notwendig, Mutexe, um gleichzeitigen Zugriff auf Daten zu synchronisieren.
- Geben Sie alle vom Betriebssystem angeforderte Ressourcen (z. B. Speicher) explizit zurück.
- Binden Sie die in den Teilaufgaben notwendigen C-Header in dem gekennzeichneten Bereich ein.
- Soweit nicht anderweitig bestimmt, gehen Sie davon aus, dass bei Systemaufrufen und Funktionen der Standardbibliothek keine Fehler auftreten.

Write a program which recursively determines the disk space consumption of all files in a directory and all its subdirectories.

- *The program shall recognize hard links by their i-node number, so that the space consumption of files is only counted once per i-node.*
- *The program is parallelized with a fixed number of threads, and subdirectories are inserted into a central queue from which they are processed by the threads. Use mutexes where necessary to synchronize concurrent data accesses.*
- *Explicitly return all allocated operating system resources (e. g., memory).*
- *Include necessary C headers in the marked area.*
- *Unless stated otherwise, assume that system calls and standard library functions do not fail.*

```c
/* global variables */
pthread_mutex_t lock;
unsigned long long total = 0; /* total file size */

/* include statements for the required C headers */
#include <errno.h>
#include <dirent.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
```

a) Vervollständigen Sie die Funktion `listDir()`, die die Einträge eines Verzeichnisses auflistet, für jeden Eintrag die Funktion `processEntry()` aufruft und ihr jeweils den Verzeichnispfad (`path`-Parameter von `listDir()`) und den Verzeichniseintrag übergibt. **5.5 pt**

- Behandeln Sie Fehler sämtlicher Systemaufrufe, indem Sie mit `exit(-1)` das Programm beenden.

Complete the function `listDir()` which lists the entries of a directory and which calls `processEntry()` for each entry, each time passing the directory path (`path` parameter of `listDir()`) and the directory entry to it.

- *Handle errors from all system calls by terminating the program with `exit(-1)`.*

Lösung:

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

void listDir(const char *path) {
    /* open the directory */
    DIR *dir = opendir(path);
    if (dir == NULL) {
        exit(-1);
    }

    /* loop through all directory entries */
    errno = 0;
    struct dirent *entry;
    while ((entry = readdir(dir)) != NULL) {
        processEntry(path, entry);
    }
    if (errno != 0) {
        exit(-1);
    }

    /* close the directory again */
    if (closedir(dir) != 0) {
        exit(-1);
    }
}
```

Adding necessary includes for the directory listing **(1.0 P)**, *opening the directory* **(0.5 P)**, *closing the directory* **(0.5 P)**, *error handling after opendir and closedir* **(0.5 P)**, *calling readdir* **(0.5 P)**, *error handling after readdir and including `errno.h`* **(1.0 P)**, *calling processEntry* **(0.5 P)**, *loop and exit condition* **(1.0 P)**.

b) Vervollständigen Sie die Funktion `makePath()`, die einen Pfad aus einem Ordnerpfad und einem Dateinamen zusammensetzt. So soll `makePath("/home/student", "exam.txt")` zum Beispiel den String `"/home/student/exam.txt"` zurückgeben. **3.5 pt**

Complete the function `makePath()` which joins a directory path and a file name to get a path to the file. For example, `makePath("/home/student", "exam.txt")` shall return the string `"/home/student/exam.txt"`.

Lösung:

```c
#include <string.h>
#include <stdlib.h>

char *makePath(const char *dirpath, const char *name) {
    /* one byte for the "/", one byte for the final null byte */
    size_t dirpathlen = strlen(dirpath);
    size_t len = dirpathlen + 1 + strlen(name) + 1;

    /* allocate memory for the result */
    char *path = malloc(len);

    /* concatenate the individual strings */
    strcpy(path, dirpath);
    path[dirpathlen] = '/';
    strcpy(path + dirpathlen + 1, name);

    return path;
}
```

Adding necessary includes **(1.0 P)**, *calculating the correct string length* **(0.5 P)**, *calling `malloc`* **(0.5 P)**, *building the resulting string* **(1.5 P)**.

c) Vervollständigen Sie die Funktion `processEntry()`, die für den übergebenen Verzeichniseintrag je nach Typ die passende Funktion zum Verarbeiten des Eintrags aufruft. **3 pt**

- Für Dateien soll `processFile()` aufgerufen werden, um die Dateigröße zu bestimmen.
- Für Verzeichnisse soll `addDirectory()` aufgerufen werden. Die Funktion löst eine rekursive Auflistung des Unterverzeichnisses aus.
- Beide Funktionen erhalten als Argument den vollständigen Pfad des Verzeichniseintrags.
- Ignorieren Sie die Verzeichniseinträge "." und "..".

Complete the function `processEntry()` which, for the directory entry passed to it, calls the appropriate function to process the entry depending on the type.

- *For files, `processFile()` shall be called to determine the file size.*
- *For directories, `addDirectory()` shall be called to trigger a recursive listing of the subdirectory.*
- *Both functions are passed the full path to the directory entry.*
- *Ignore the directory entries "." und "..".*

Lösung:

```
#include <string.h>
#include <dirent.h>

void processEntry(const char *dirpath, struct dirent *entry) {
    char *path = makePath(dirpath, entry->d_name);
    /* ignore "." and ".." */
    if (!strcmp(entry->d_name, "..") || !strcmp(entry->d_name, ".")) {
        return;
    }
    /* ignore everything but regular files and directories */
    if (entry->d_type == DT_DIR) {
        addDirectory(path);
    } else if (entry->d_type == DT_REG) {
        processFile(path);
    }
    free(path);
}
```

Constructing the full path of the entry **(0.5 P)**, *calling* `addDirectory` *for directories* **(0.5 P)**, *excluding "." and ".."* **(0.5 P)**, *calling* `processFile` *for regular files* **(0.5 P)**, *ignoring other types of entries* **(0.5 P)**, *freeing the memory again* **(0.5 P)**.

d) Vervollständigen Sie die Funktion `processFile()`, die die Größe der Datei an dem übergebenen Pfad bestimmt und auf die Gesamtgröße der betrachteten Dateien aufsummiert. **4.5 pt**

- Die Funktion wird parallel aus verschiedenen Threads des Thread-Pools aufgerufen. Achten Sie auf korrekte Synchronisierung.
- Ignorieren Sie die Datei, falls ein anderer Hardlink auf den selben I-Node bereits betrachtet wurde. Verwenden Sie hierfür die gegebene globale Hashtabelle, um eine Liste bereits betrachteter I-Nodes zu halten.
- Verwenden Sie die Funktion `contains()`, die in dieser Tabelle prüft, ob ein bestimmter I-Node enthalten ist. Ist der I-Node enthalten, gibt die Funktion 1 und ansonsten 0 zurück.
- Verwenden Sie die Funktion `insert()`, um I-Nodes in die Tabelle einzutragen.
- Beide Funktionen führen keine interne Synchronisierung von Zugriffen auf die Tabelle durch.

Complete the function `processFile()` *which determines the size of the file at the specified path and adds it to the total size of all processed files.*

- *The function is called in parallel from multiple threads of the thread pool. Ensure proper synchronization.*
- *Ignore the file if another hard link to the same i-node has already been processed. Use the given global hash table to maintain a list of i-nodes which have already processed.*
- *Use the function* `contains()` *to check whether this table contains the specified i-node. If the i-node is present, the function returns 1, else it returns 0.*
- *Use the function* `insert()` *to insert i-nodes into the table.*
- *Both functions do not internally synchronize accesses to the table.*

Lösung:

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

void processFile(const char *path) {
    /* retrieve file attributes */
    struct stat s;
    stat(path, &s);
    ino_t inode = s.st_ino;

    /* we only want to count every inode once, so we check whether the hash table
     * already contains the inode */
    pthread_mutex_lock(&lock);
    if (!contains(inode)) {
        insert(inode);
        total += s.st_blocks * 512; /* s.st_size; also accepted */
    }
    pthread_mutex_unlock(&lock);
}
```

Adding necessary includes (`sys/stat.h`, `unistd.h` and `pthread.h`) **(1.0 P)**, *calling `stat` with a correct destination pointer* **(1.0 P)**, *checking whether the inode has already been visited* **(0.5 P)**, *inserting the inode into the hash table* **(0.5 P)**, *increasing the total size* **(0.5 P)**, *correct locking* **(1.0 P)**.

e) Vervollständigen Sie die `main()`-Funktion des Programmes. **3.5 pt**

- Initialisieren Sie globale Variablen und starten Sie 10 Threads, die als Teil des oben genannten Thread-Pools jeweils die Funktion `doWork()` ausführen.
- `addDirectory()` übergibt einen Pfad zur rekursiven Gesamtgrößenberechnung an den Thread-Pool und kehrt dann sofort zurück. Die Threads des Thread-Pools beenden sich, sobald die Berechnung abgeschlossen ist.
- Geben Sie am Funktionsende verbliebene Resourcen frei.

Complete the `main()` function of the program.

- *Initialize global variables and start 10 threads which all execute the function `doWork()` as part of the thread pool mentioned above.*
- *`addDirectory()` passes a path to the thread pool to recursively determine its total size. The function then returns immediately. The threads of the thread pool terminate once the computation of the size is complete.*
- *At the end of the function, release any remaining resources.*

Lösung:

```
#include <pthread.h>
#include <stdio.h>

int main(int argc, char **argv) {
    pthread_t threads[10];
    int i;
```

```
    /* initialize the mutex */
    pthread_mutex_init(&lock, NULL);
    /* spawn the threads */
    for (i = 0; i < 10; i++) {
        pthread_create(&threads[i], NULL, doWork, NULL);
    }
    /* start scanning the directory */
    addDirectory(argv[1]);
    /* wait for the threads to quit */
    for (i = 0; i < 10; i++) {
        pthread_join(threads[i], NULL);
    }
    /* free remaining resources */
    pthread_mutex_destroy(&lock);
    /* print the result */
    printf("Size: %lld bytes\n", total);
    return 0;
}
```

Initializing and destroying the mutex **(1.0 P)**, *creating the threads* **(1.5 P)**, *waiting for the threads* **(1.0 P)**. **Total: 20.0pt**

Aufgabe 3: Speicherdeduplikation

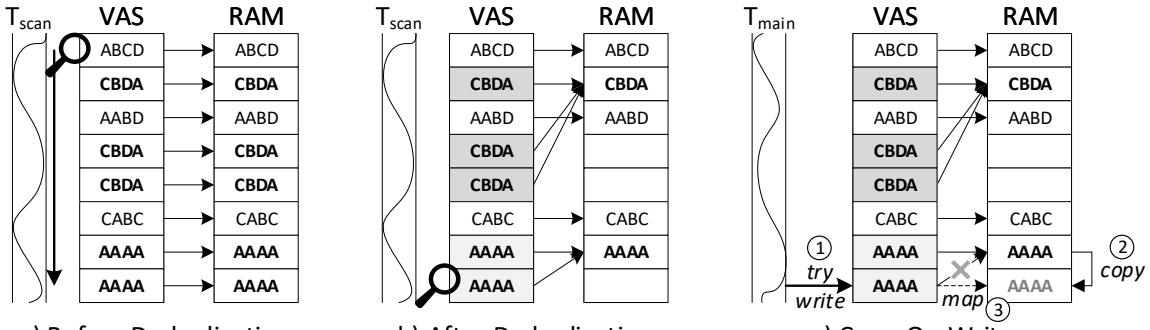
Assignment 3: Memory Deduplication

Um physischen Speicher einzusparen, sollen in einem virtuellen Adressraum (VAS) die Seitentabelleneinträge (PTEs) derart verändert werden, dass Seiten mit identischem Inhalt auf die gleiche physische Seite (PFN) zeigen. Ein (nicht optimales) Verfahren identifiziert Duplikate, indem die Seiteninhalte gehasht und über eine Hashtabelle abgeglichen werden. Das Verfahren verwendet Copy-On-Write (CoW), um bei schreibendem Zugriff deduplizierte Seiten wieder aufzubrechen.

- Die Seitengröße beträgt 4 KiB.
- Adressräume enthalten keine schreibgeschützten Bereiche wie `.rodata`.
- Das OS gibt nicht referenzierte physische Seiten automatisch frei.
- Die Suche läuft parallel zum Hauptthread T_{main} in einem eigenen Thread T_{scan}.

To save physical memory the page table entries (PTEs) in a virtual address space (VAS) should be adjusted so that pages with identical contents point to the same physical page (PFN). A best-effort approach identifies duplicates by hashing the pages' contents and matching them with a hash table. The approach uses copy-on-write (CoW) for breaking deduplicated pages on write access.

- *The page size is 4 KiB.*
- *Address spaces do not contain read-only areas such as* `.rodata`*.*
- *The OS frees not referenced physical pages automatically.*
- *The search runs in parallel to the main thread T_{main} in a dedicated thread T_{scan}.*



a) Before Deduplication b) After Deduplication c) Copy-On-Write

```
/* PTE: Bits[12:31] = PFN, Bits[4:11] = unused, Bits[0:3] = flags      */
#define PTE_P_MASK   0x1 /* Present - if set, PTE is valid mapping     */
#define PTE_RO_MASK  0x2 /* Read-only - if set, page fault on write    */
#define PTE_COW_MASK 0x4 /* CoW - if set, CoW enabled (ignored by MMU) */

/* Returns or sets PTE for a given virtual address */
uint32_t getPte(uint32_t va);
void setPte(uint32_t va, uint32_t pte);

/* Locks or unlocks the virtual address space.
 * Calling lockVas() will block a thread if the VAS is already locked */
void lockVas(void);
void unlockVas(void);

#define INV_PFN ((uint32_t)-1) /* Invalid physical frame number */

/* Returns a pointer to the contents of a physical page, NULL for INV_PFN */
uint32_t *getPage(uint32_t pfn);
```

a) Vervollständigen Sie die Funktion `hash()`, die für eine gegebene physische Seite `pfn` den 32-bit djb2-Hash $h(i) = h(i-1) * 33 + p[i]$ zurückgibt, wobei $p[i]$ das i-te 32-bit Wort der Seite ist und $h(1023)$ den Hashwert der Seite enthält. **3.0 pt**

- Wählen Sie $h(-1) = 5381$.
- Verwenden Sie zur Implementierung der Hashfunktion keine Multiplikation.
- `pfn` ist eine valide physische Seite (`pfn` $\neq$ `INV_PFN`).

Complete the function `hash()`, which returns for a given physical page `pfn` the 32-bit djb2-hash $h(i) = h(i-1) * 33 + p[i]$, where $p[i]$ is the i-th 32-bit word in the page and $h(1023)$ is the hash of the page.

- *Choose $h(-1) = 5381$.*
- *Do not use multiplication for implementing the hash function.*
- *`pfn` is a valid physical page (`pfn` $\neq$ `INV_PFN`).*

Lösung:

```
uint32_t hash(uint32_t pfn) {
    uint32_t *p = getPage(pfn);
    uint32_t h  = 5381;
    int i;

    for (i = 0; i < 1024; ++i)
        h = ((h << 5) + h) + p[i];

    return h;
}
```

get pointer to page **(0.5 P)**, *hash start value* **(0.5 P)**, *for-loop* **(0.5 P)**, *translation of hash function* **(1.0 P)**, *return hash* **(0.5 P)**

b) Wo sehen Sie potentiell Probleme bei der Verwendung eines Hashwertes zur Identifikation gleicher Seiteninhalte? **1 pt**

Where do you see potential problems in the use of a hash function for identifying equal page contents?

Lösung:

A simple hash function (such as the 32-bit djb2-hash) is very likely to produce hash collisions **(0.5 P)**. *It is therefore possible that pages with different contents are deduplicated* **(0.5 P)**, *thereby corrupting the memory of the process.*

c) Vervollständigen Sie die Funktion `updatePte()`, die den Seitentabelleneintrag für die virtuelle Seite mit der Adresse `va` aktualisiert. **3.0 pt**

- Nutzen Sie das im Kommentar gegebene PTE-Format und die Bitmasken.
- Bei `pfn = INV_PFN` darf das Present-Bit nicht gesetzt werden.
- `ro` und `cow` bestimmen, ob die Seite als Read-Only oder Copy-On-Write markiert werden soll.

Complete the function `updatePte()`, which updates the page table entry for the virtual page with the address `va`.

- *Use the PTE format and the bit masks described in the comments.*
- *For `pfn = INV_PFN` the present bit must not be set.*
- *`ro` and `cow` determine, if the page should be marked read-only or copy-on-write.*

```
/* PTE: Bits[12:31] = PFN, Bits[4:11] = unused, Bits[0:3] = flags   */
#define PTE_P_MASK   0x1 /* Present - if set, PTE is valid mapping  */
#define PTE_RO_MASK  0x2 /* Read-only - if set, page fault on write */
#define PTE_COW_MASK 0x4 /* CoW - if set, copy-on-write enabled     */
                         /* (CoW flag ignored by MMU)               */
```

Lösung:

```
void updatePte(uint32_t va, uint32_t pfn, int ro, int cow) {
    uint32_t pte = pfn << 12;

    if (pfn != INV_PFN)
        pte |= PTE_P_MASK;

    if (ro)  pte |= PTE_RO_MASK;
    if (cow) pte |= PTE_COW_MASK;

    setPte(va, pte);
}
```

add PFN to PTE **(1.0 P)**, *add P-flag* **(0.5 P)**, *add RO-flag* **(0.5 P)**, *add COW-flag* **(0.5 P)**, *set PTE* **(0.5 P)**

d) Vervollständigen Sie die Funktion `getPfn()`, die die physischen Seitennummer (PFN) aus dem PTE der virtuellen Seite mit der Adresse `va` zurückgibt. **1.5 pt**

- Geben Sie `INV_PFN` zurück, falls das Present-Bit nicht gesetzt ist.

Complete the function `getPfn()`, which returns the physical frame number (PFN) from the PTE of the virtual page with the address `va`.

- *Return `INV_PFN` if the present bit is not set.*

Lösung:

```
uint32_t getPfn(uint32_t va) {
    uint32_t pte = getPte(va);

    return (pte & PTE_P_MASK) ? (pte >> 12) : INV_PFN;
}
```

get PTE from VA **(0.5 P)**, *return INV_PFN for invalid mapping* **(0.5 P)**, *return PFN otherwise* **(0.5 P)**

e) Vervollständigen Sie die Funktion `tryMerge()`, die versucht die virtuelle Seite mit der Adresse `va` zu deduplizieren. **6.5 pt**

- T_{scan} sucht in Richtung aufsteigender Adressen und ruft `tryMerge()` für jede gültige (Present-Bit gesetzt) Seite einmal auf.
- Verwenden Sie `lockVas()`/`unlockVas()` sowie Schreibschutz wo nötig.
- Verwenden Sie das CoW-Flag nur für redundante Seiten.
- Bei einem Seitenfehler wird eine schreibgeschützte Seite unter dem VAS-Lock aus der Hashtabelle entfernt.
- Übereinstimmende Hashes sind zur Deduplikation ausreichend.

Complete the function `tryMerge()`, which tries to deduplicate the virtual page with the address `va`.

- T_{scan} *scans in the direction of increasing addresses and calls `tryMerge()` once per valid (present bit set) page.*
- *Use `lockVas()`/`unlockVas()` and write protection where necessary.*
- *Use the CoW-flag for redundant pages only.*
- *On a page fault a write-protected page is removed from the hash table under the VAS lock.*
- *Matching hashes are sufficient for deduplication.*

```
/* Inserts a new <hash, va>-pair into the hash table.
 * Every hash value can occur only once in the table. */
void insert(uint32_t hash, uint32_t va);

/* Returns the virtual address for the given hash, or
 * 0 if the hash cannot be found in the hash table */
uint32_t find(uint32_t hash);
```

Lösung:

```
void tryMerge(uint32_t va) {
    uint32_t va2, pfn, pfn2, h;

    lockVas();          /* Synchronize with writeFault() */
    pfn = getPfn(va);

    /* Prevent main thread from writing to page after hashing */
    updatePte(va, pfn, 1, 0);

    h = hash(pfn);
    va2 = find(h);
    if (va2 == 0) {     /* No duplicate found */
        insert(h, va);
    } else {            /* Duplicate found, merge them */
        pfn2 = getPfn(va2);
        updatePte(va, pfn2, 1, 1);
        updatePte(va2, pfn2, 1, 1);
    }
    unlockVas();
}
```

lock VAS **(0.5 P)**, *unlock VAS (on exit)* **(0.5 P)**, *get PFN* **(0.5 P)**, *set write protection* **(1.0 P)**, *hash page* **(0.5 P)**, *find hash* **(0.5 P)**, *check if hash found* **(0.5 P)**, *insert hash if no match* **(0.5 P)**, *get PFN of matched VA* **(0.5 P)**, *update PTEs* **(1.5 P)**

f) Vervollständigen Sie die Funktion `writeFault()`, die bei schreibendem Zugriff auf eine gültige, schreibgeschützte virtuelle Seite mit der Adresse `va` aufgerufen wird. **5.0 pt**

- Verwenden Sie `lockVas()`/`unlockVas()` wo nötig.
- Die PTEs von anderen Seiten außer `va` müssen nicht angepasst werden.
- Nutzen Sie `memcpy()`, um Seiteninhalte zu kopieren. Kopieren Sie Seiten nur, wenn sie dedupliziert wurden.

Complete the function `writeFault()`, which is called on write access to a valid, write-protected virtual page with the address `va`.

- *Use `lockVas()`/`unlockVas()` where necessary.*
- *The PTEs of pages other than `va` do not need to be updated.*
- *Use `memcpy()` for copying page contents. Only copy pages which have been deduplicated.*

```
/* Computes the hash of the virtual address and removes it from the table.
 * If the hash does not exist, does nothing */
void remove(uint32_t va);

/* Allocates a free physical page and returns the PFN. Never fails! */
uint32_t allocPfn(void);
```

Lösung:

```
void writeFault(uint32_t va) {
    uint32_t src, dst;

    /* Synchronize with tryMerge() */
    lockVas();

    if (getPte(va) & PTE_COW_MASK) {
        /* Break deduplication */
        src = getPfn(va);
        dst = allocPfn();

        memcpy(getPage(dst), getPage(src), 1 << 12);
    } else {
        /* Just remove write protection */
        dst = getPfn(va);
    }

    remove(va);

    /* Remove write protection to resolve the page fault.
     * Since we are not changing PTEs of other pages an
     * write access to the last page of a deduplication set
     * will still be a copy-on-write case */
    updatePte(va, dst, 0, 0);

    unlockVas();
}
```

lock VAS **(0.5 P)**, *unlock VAS (on exit)* **(0.5 P)**, *check for COW-flag* **(0.5 P)**, *get PFN* **(0.5 P)**, *if cow, allocate destination PFN* **(0.5 P)**, *if cow, memcpy* **(1.0 P)**, *call remove* **(0.5 P)**, *call updatePte* **(1.0 P)**

Total: 20.0pt