Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

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

Musterlösung Nachklausur

Programmieren

29.09.2020

Alle Punkteangaben ohne Gewähr!

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

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

- Die Prüfung besteht aus 14 Blättern: 1 Deckblatt, 13 Aufgabenblättern mit insgesamt 3 Aufgaben und 0 Blättern Man-Pages.

  The examination consists of 14 pages: 1 cover sheet, 13 sheets containing 3 assignments, and 0 sheets for man pages.

- Es sind keinerlei Hilfsmittel erlaubt!

  No additional material is allowed.

- Die Prüfung ist nicht bestanden, wenn Sie aktiv oder passiv 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	15	15	15	45
Erreichte Punkte				
Note				

Aufgabe 1: C Grundlagen

Assignment 1: C Basics

a) Was gibt der unten stehende Code bei der Ausführung der Funktion `print_test()` aus? **1 pt**

What does the code below print when running the function `print_test()`?

```
struct test {
    int a, b, c;
};

void print_test(void) {
    struct test t = {.b = 2};
    printf("%d/%d/%d", t.a, t.b, t.c);
}
```

Solution:

0/2/0

b) Geben Sie für alle Felder des unten stehenden `struct mystruct` die Größe des Feldes und die Größe des Paddings *nach* dem Feld in Bytes an. Schreiben Sie „0“, falls kein Padding eingefügt wird. Gehen Sie von einem 64-Bit-System aus. **3 pt**

For each field of the `struct mystruct` below, give the field's size and the size of the padding after *the field in bytes. Write "0" if the compiler does not insert any padding. Assume a 64-bit system.*

Code	Field size [Bytes]	Padding size [Bytes]
`struct mystruct {`	—	—
`uint32_t a;`	4	0
`char b;`	1	3
`void *c;`	8	0
`int16_t d;`	2	6
`};`	—	—

(1.0 P) for all field sizes, **(0.5 P)** per padding size

c) Definieren Sie ein C-Makro `MIN`, das das kleinere von zwei übergebenen Argumenten zurückgibt. **1.5 pt**

Define a C macro `MIN` that returns the smaller of the supplied arguments.

Examples:

```
assert(MIN(3, 5) == 3);
assert(MIN(3, 0) == 0);
assert(MIN(1 | 2, 13 & 5) == 3);
```

Solution:

```
#define MIN(a, b) ((a) > (b) ? (b) : (a))
```

macro syntax **(0.5 P)**, *parentheses everywhere* **(0.5 P)**, *ternary expression* **(0.5 P)**

Welches Problem mit dem Makro `MIN` könnte im Codeausschnitt unten auftreten? **1 pt**

Which issue with the macro `MIN` could occur in the code snippet below?

```
int f() { /* ... */ }

int main() {
    /* upper bound 10 */
    int x = MIN(10, f());
    /* ... */
}
```

Solution:

The function `f()` might be called twice by the macro **(0.5 P)**. *Any side effects of `f()` could thus happen twice* **(0.5 P)**.

d) Geben Sie ein Beispiel eines *Include-Guards* an. Machen Sie deutlich, wo jedes Stück Code stehen würde. Nutzen Sie keine Spracherweiterungen. **1.5 pt**

Give an example of an include guard. *Clearly indicate where each piece of code would be placed. Do not use any language extensions.*

Solution:

```
#ifndef FOO_H
#define FOO_H
/* header contents */
#endif
```

ifndef/endif **(0.5 P)**, *define* **(0.5 P)**, *position information* **(0.5 P)**

Wann and warum werden Include-Guards in C benötigt? **1 pt**

When and why are include guards required in C?

Solution:

We need include guards to avoid duplicate declarations **(0.5 P)** *if a header file is included multiple times* **(0.5 P)**.

e) Schreiben Sie eine Funktion `concat()`, die zwei Strings aneinanderhängt und als neuen String zurückgibt. **3 pt**

- Allozieren Sie genau so viel Heap-Speicher für den neuen String, wie nötig.
- Von den C-Stringfunktionen dürfen Sie lediglich `strlen()` verwenden.

Implement the function `concat()` that concatenates two strings and returns the result as a new string.

- *Allocate exactly as much heap memory for the new string as necessary.*
- *Of the C string functions you may only use `strlen()`.*

Solution:

```
char *concat(const char *s1, const char *s2) {
    char *str = malloc(strlen(s1) + strlen(s2) + 1);
    char *s = str;
    while (*s1) *s++ = *s1++;
    while (*s2) *s++ = *s2++;
    *s = 0;
    return str;
}
```

call to malloc **(0.5 P)**, *correct size* **(0.5 P)**, *copying first string* **(0.5 P)**, *copying second string* **(0.5 P)**, *null-terminator* **(0.5 P)**, *return* **(0.5 P)**

Warum kann `concat()` bei gleichbleibender Signatur nicht stattdessen Speicher auf dem Stack allozieren? **0.5 pt**

Why is it not possible for `concat()` *to allocate memory on the stack instead? Assume you cannot change the function signature.*

Solution:

Any memory that `concat()` *might allocate on the stack will be released when the function returns and thus cannot be used by the caller.*

f) C `enum`s werden häufig verwendet, um Bitflags deskriptive Namen zu geben. Füllen Sie die Deklaration von `enum permission` mit drei unterschiedlichen Bitflags `READ`, `WRITE` und `EXECUTE` aus. **0.5 pt**

C `enum`*s are often used to give bitflags descriptive names. Fill in the declaration of* `enum permission` *with three different bitflags* `READ`*,* `WRITE`*, and* `EXECUTE`*.*

Solution:

```
enum permissions {
  READ = 1,
  WRITE = 2,
  EXECUTE = 4
};
```

Schreiben Sie eine Funktion `set_rw()`, die die `READ`- und `WRITE`-Flags aus `enum permissions` im übergebenen Wert setzt und diesen anschließend zurückgibt. **1 pt**

Write a function `set_rw()` *that sets the* `READ` *and* `WRITE` *flags of* `enum permissions` *in the passed value and then returns this value.*

Solution:

```
unsigned set_rw(unsigned flags) {
    return flags | READ | WRITE;
}
```

g) Nach welcher Aufrufkonvention werden Funktionsparameter in C ausschließlich über den Stack übergeben? **0.5 pt**

According to which calling convention are function parameters in C passed exclusively via the stack?

Solution:

cdecl

Die Funktion `and()` wird vom C-Compiler wie folgt nach Intel x86 übersetzt. In welchen Registern werden die Parameter übergeben? Eine exakte Zuordnung zu `a` und `b` ist nicht notwendig. **0.5 pt**

The function `and()` is translated by the C compiler to Intel x86 as follows. In which registers are the parameters passed? An exact mapping to `a` and `b` is not required.

```
int and(int a, int b) {
    return a & b;
}
```

```
and:
    mov     eax, edi
    and     eax, esi
    ret
```

Solution:

The arguments are passed in `edi` and `esi`.

Total: 15.0pt

Aufgabe 2: Volltextsuche

Assignment 2: Full-Text Search

Sie sollen ein Programm schreiben, das die Vorkommen eines Schlüsselworts in einer Datei sucht. Die Suche soll innerhalb der Datei abschnittsweise mittels mehrerer Prozesse parallelisiert werden. Wird das Schlüsselwort gefunden, so soll auf der Kommandozeile die Position in der Datei in Bytes ausgegeben werden. So soll, wenn das Schlüsselwort „test“ in einer Datei mit dem Inhalt „**testest**asdf**test**“ gesucht wird, das Programm die Zahlen 0, 3 und 11 ausgeben.

- Sie können davon ausgehen, dass es sich um eine Textdatei ohne Null-Bytes handelt.
- Sie müssen keine C-Header inkludieren.
- Sie müssen in dieser Aufgabe keine Fehlerbehandlung implementieren.
- Geben Sie jegliche im Code angeforderten Ressourcen wieder frei.

*You shall write a program which searches a file for a keyword. Inside the file, the search shall be parallelized section-wise using multiple processes. When the keyword is found, the position in the file in bytes shall be printed to the command line. For example, if the keyword “test” is searched in a file containing “**testest**asdf**test**”, the program shall output the numbers 0, 3, and 11.*

- *You may assume that the file is a text file without null bytes.*
- *You do not need to include any C headers.*
- *You do not have to implement any error handling.*
- *Free all resources allocated in the code.*

a) Vervollständigen Sie die Funktion `print_u64()`, die den übergebenen 64-Bit-Integer und einen Zeilenumbruch ausgibt. **1 pt**

Complete the function `print_u64()` which prints the specified 64-bit integer and a line break.

Solution:

```
void print_u64(uint64_t x) {
    printf("%"PRId64"\n", x);
}
```

`printf()` with a correct format specifier **(0.5 P)** *and a linebreak* **(0.5 P)**.

b) Vervollständigen Sie die Funktion `search_str()`, die alle Vorkommnisse des Schlüsselwortes `word` innerhalb des Strings `buffer` mittels `print_u64()` ausgibt. **4 pt**

- `buffer` enthält die Inhalte der Datei beginnend an der Position `offset`. Die Funktion soll also vor der Ausgabe `offset` auf die jeweilige Position des Schlüsselworts addieren.
- Sie können `strstr()` zum Suchen des Schlüsselworts verwenden.

Complete the function `search_str()`*, which prints all occurrences of the keyword* `word` *in the string* `buffer` *using* `print_u64()`*.*

- `buffer` *contains the file contents starting at the position* `offset`*. The function shall therefore add* `offset` *to the position of each occurrence before printing.*
- *You may use* `strstr()` *to search for the keyword.*

Solution:

```
void search_str(uint64_t offset, const char *buffer, const char *word) {
    const char *cursor = buffer;
    while (1) {
        const char *found = strstr(cursor, word);
        if (found) {
            print_u64(offset + (found - buffer));
        } else {
            break;
        }
        cursor = found + 1;
    }
}
```

Correct usage of `strstr()` *or alternative implementation* **(1.0 P)**, *print the correct offset* **(1.0 P)** *if the keyword is found* **(0.5 P)**, *loop and continue to search for the next occurrence at the correct location* **(1.0 P)**, *correct loop exit condition* **(0.5 P)**.

c) Vervollständigen Sie die Funktion `search_fd()`, die einen Abschnitt der gegebenen Datei beginnend an der Position `offset` mit der Länge `len` in den Speicher lädt und dann mittels `search_str()` nach dem Schlüsselwort `word` durchsucht. **3.5 pt**

- Beachten Sie, dass `search_str()` nullterminierte Strings erwartet.

Complete the function `search_fd()` *which reads a section of the specified file starting at the position* `offset` *with the length* `len` *into memory and then uses* `search_str()` *to search for the keyword* `word`*.*

- *Note that* `search_str()` *expects null-terminated strings.*

Solution:

```
void search_fd(int fd, uint64_t offset, size_t len, const char *word) {
    ssize_t bytes_read;
    char *buffer = malloc(len + 1);

    bytes_read = pread(fd, buffer, len, offset);
    buffer[bytes_read] = 0;
    search_str(offset, buffer, word);

    free(buffer);
}
```

Allocate a buffer **(1.0 P)**, *use* `pread()` *to read the correct data* **(1.0 P)**, *zero-terminate the buffer contents* **(0.5 P)**, *search for the keyword* **(0.5 P)**, *free the buffer* **(0.5 P)**.

Note: If this function cannot cope with an end-of-file condition, the solution to question e) needs to be written in a way that no end-of-file condition is reached. If this is not the case, the corresponding **(0.5 P)** *are deducted here, as the problem usually surfaces in the form of incorrect zero termination in end-of-file scenarios.*

d) Vervollständigen Sie die Funktion `spawn_search()`, die einen neuen Prozess startet, der `search_fd()` mit den gegebenen Parametern ausführt. **2 pt**

Complete the function `spawn_search()` which starts a new process that executes `search_fd()` with the given parameters.

Solution:

```
void spawn_search(int fd, uint64_t offset, size_t len, const char *word) {
    int pid = fork();
    if (pid == 0) {
        search_fd(fd, offset, len, word);
        exit(0);
    }
}
```

*Create a new process with `fork()` **(1.0 P)** execute `search_fd()` in the child process **(0.5 P)**, exit the child process **(0.5 P)**.*

Die Datei soll stückweise mittels `spawn_search()` durchsucht werden. Wie sehr müssen sich die von `offset` und `len` bestimmten Blöcke überlappen, damit sämtliche Vorkommnisse des Wortes gefunden werden? **1 pt**

The file shall be searched piece-wise by `spawn_search()`. How much do the blocks specified by `offset` and `len` need to overlap so that all occurrences of the word are found?

Solution:

The blocks need to overlap by the length of the keyword minus 1 byte. More overlap causes matches to be printed twice, whereas less overlap causes matches to be missed.

e) Vervollständigen Sie die Funktion `main()`, die die zu durchsuchende Datei öffnet und dann eine blockweise parallele Suche mittels `spawn_search()` durchführt. **3.5 pt**

- Das Programm soll auf sämtliche Kindprozesse warten, bevor es sich beendet.

Complete the function `main()` which opens the file to be searched and then executes a block-wise parallel search using `spawn_search()`.

- *The program shall wait for all child processes before terminating.*

Solution:

```
int main(int argc, char **argv) {
    const char *file = argv[1];
    const char *word = argv[2]; /* assume that the string is non-empty */

    int fd = open(file, O_RDONLY);
    uint64_t size = file_size(fd); /* length of the file */
    const size_t BLOCK = 4096; /* block size - add the overlap! */
    uint64_t num_blocks = (size + BLOCK - 1) / BLOCK; /* number of blocks */

    uint64_t offset = 0;
    size_t i;
    size_t search_len = BLOCK + strlen(word) - 1;
```

```
    for (i = 0; i < num_blocks; i++) {
        off_t offset = i * BLOCK;
        spawn_search(fd, offset, search_len, word);
    }

    for (i = 0; i < num_blocks; i++) {
        wait(NULL);
    }


    close(fd);
    return 0;
}
```

Calculate the correct amount of data required per block as per last question **(0.5 P)**, *loop through the blocks* **(0.5 P)**, *calculate the correct offset for each block* **(0.5 P)**, *search the block* **(1.0 P)**, *wait for all the created processes* **(1.0 P)**. **Total: 15.0pt**

Aufgabe 3: Dateisystem-Cache

Assignment 3: File System Cache

Der Dateisystem-Cache puffert zur Beschleunigung von Dateizugriffen den Dateiinhalt im Hauptspeicher. Für jede offene Datei muss das Betriebssystem eine Datenstruktur speichern, die einen Datei-Offset in eine Adresse im Cache übersetzt.

Im Folgenden soll zu diesem Zweck ein Radix-Baum implementiert werden. Ein Radix-Baum ist eine dynamisch wachsende Struktur, die einen Ganzzahlindex auf einen Pointer übersetzt.

Ähnlich der Adressübersetzung bei seitenbasierter Speicherverwaltung teilt der Radix-Baum den Index in mehrere gleich große Teile auf. Jeder Teil bestimmt die nächste Ebene im Baum. Die Höhe des Baums wird dabei durch den größten gespeicherten Index festgelegt.

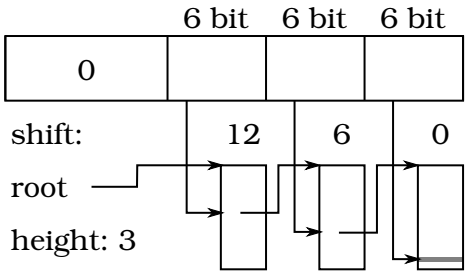
- Ein Baum der Höhe 0 kann genau einen Eintrag bei Index 0 speichern.
- Die `void*`-Pointer sind in der letzten Ebene des Baums Nutzerdaten und in allen anderen Ebenen Pointer auf `rt_node`.

The file system cache buffers file contents in main memory to speed up file access. For every open file, the operating system needs to keep a data structure that translates file offsets to a memory address in the cache.

In the following, you will implement a radix tree for this purpose. A radix tree is as a dynamically growing structure that translates an integer index to a pointer.

Similar to address translation with paging, the radix tree partitions the index into equally-sized parts. Every part determines the next level of the tree. The largest index determines the height of the tree.

- *A tree with height 0 can store exactly one element at index 0.*
- *The `void*` pointers at the last level of the tree are user data. At all other levels they are pointers to `rt_node`.*




```
#define RT_SHIFT 6              /* number of index bits per level */
#define RT_SIZE (1 << 6)        /* number of entries per level */
#define RT_MASK (RT_SIZE - 1)

typedef struct _rt_node {
    uint8_t shift;             /* index shift for this level */
    void *entries[RT_SIZE];
} rt_node;
```

```
typedef struct _rt_tree {
  uint8_t height;              /* number of levels */
  void *root;                  /* root entry */
} rt_tree;
```

a) Vervollständigen Sie die Funktion `entry_idx()`, die für einen Index `idx` den Offset in dem `entries`-Array einer `rt_node`-Struktur mit dem gegebenen `shift` berechnet. **1 pt**

Complete the function `entry_idx()`, which takes an index `idx` and calculates the offset in the `entries` array of an `rt_node` structure with the given `shift`.

Solution:

```
int entry_idx(uint64_t idx, uint8_t shift) {
    return (idx >> shift) & RT_MASK;
}
```

b) Das Einfügen in den Radix-Baum besteht aus zwei Schritten: Dem Hinzufügen von Ebenen in den Baum, sodass der Index passt, und dem Durchlaufen des Baumes bis zum passenden Blattknoten.

Inserting into the tree requires two steps: adding levels to the tree until the index fits, and then descending the tree to the correct leaf node.

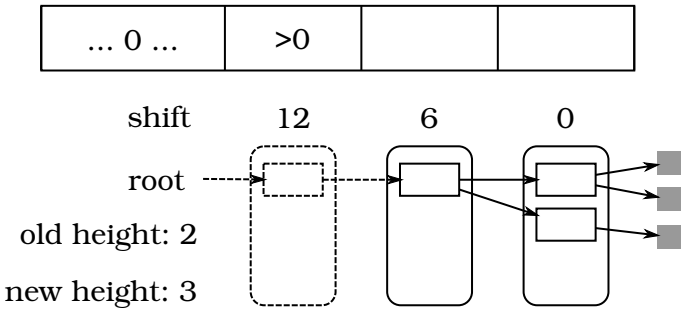
```
void add_levels(rt_tree *rt, uint64_t idx);
void **descend(rt_tree *rt, uint64_t idx);

// Insert an element (ptr) into the radix tree (rt) at index (idx).
void rt_insert(rt_tree *rt, uint64_t idx, void *ptr) {
    add_levels(rt, idx);

    void **entry = descend(rt, idx);
    *entry = ptr;
}
```

Vervollständigen Sie die Funktion `add_levels`, die dem Baum so lange Ebenen hinzufügt, bis der Index `idx` abgebildet werden kann (`idx` < 2^{shift}). **4 pt**

- Die Funktion `alloc_node()` gibt einen neuen Knoten zurück, dessen Felder alle auf 0 initialisiert sind.
- Falls der Baum nicht leer ist, wird der bisherige Wurzelknoten zum 0. Eintrag im neuen Wurzelknoten.
- Für einen leeren Baum sollen Sie keine neuen Knoten allozieren, passen Sie `rt->height` aber in jedem Fall entsprechend an.

Complete the function `add_levels()`, which adds levels to the tree until the index `idx` can be mapped (`idx` < 2^{shift}).

- *The function `alloc_node()` returns a new node with all fields initialized to 0.*
- *If the tree is not empty, the old root node is the 0th entry in the new root node.*
- *For empty trees, you shall not allocate new nodes, but always adjust `rt->height` accordingly.*



Solution:

```
rt_node *alloc_node(); /* does not fail */
void add_levels(rt_tree *rt, uint64_t idx) {
    uint8_t shift = rt->height * RT_SHIFT;
    while (idx >> shift) {
        if (rt->root != NULL) {
            rt_node *n = alloc_node();
            n->shift = shift;
            n->entries[0] = rt->root;
            rt->root = n;
        }

        rt->height++;
        shift += RT_SHIFT;
    }
}
```

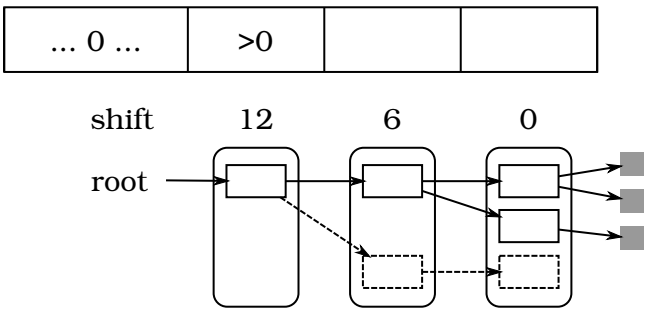
loop condition **(1.0 P)**, *do not create nodes in empty trees* **(0.5 P)**, *allocate and initialize node* **(1.5 P)**, *set fields in* `rt` **(1.0 P)**

c) Vervollständigen Sie die Funktion `descend()`, die den Baum von der Wurzel bis zu dem indizierten Element durchläuft und einen Pointer auf dieses zurückgibt. **4 pt**

- Fehlende Zwischenknoten müssen mit `alloc_node()` hinzugefügt werden.

Complete the function `descend()`*, which walks the tree from the root to the indexed element and returns a pointer to it.*

- *Add missing intermediate nodes using* `alloc_node()`*.*



Solution:

```
rt_node *alloc_node(); /* does not fail */
void **descend(rt_tree *rt, uint64_t idx) {
    uint8_t shift = rt->height * RT_SHIFT;
    void **entry  = &rt->root;
    while (shift > 0) {
        shift -= RT_SHIFT;

        rt_node *n = *entry;
        if (n == NULL) {
            n = alloc_node();
            n->shift = shift;
            *entry = n;
        }

        entry = &n->entries[entry_idx(idx, shift)];
    }

    return entry;
}
```

adjust `shift` *in loop* **(0.5 P)**, *get or allocate node* **(2.0 P)**, *update* `entry` **(1.5 P)**

d) Vervollständigen Sie die Funktion `count()`, die die Anzahl gültiger (nicht `NULL`) Einträge in einem Knoten zurückgibt. **2 pt**

Complete the function `count()`*, which returns the number of valid (not* `NULL`*) entries in a node.*

Solution:

```
int count(rt_node *n) {
    int c = 0;
    for (int i = 0; i < RT_SIZE; ++i) {
        if (n->entries[i] != NULL) {
            c++;
        }
    }
    return c;
}
```

loop **(0.5 P)**, *counting valid entries* **(1.0 P)**, *returning count* **(0.5 P)**

e) Vervollständigen Sie die Funktion `shrink()`, die unnötige Ebenen des Baums entfernt. **4 pt**

- Unnötige Ebenen besitzen lediglich einen gültigen Eintrag an Index 0 oder überhaupt keinen gültigen Eintrag.
- Nutzen Sie die Funktion `free_node()`, um Knoten freizugeben.
- Passen Sie `rt->height` entsprechend an.

Complete the function `shrink()`*, which removes unnecessary levels from the tree.*

- *An unnecessary level has only one valid entry at index 0 or no valid entry at all.*
- *Call the function* `free_node()` *to free nodes.*
- *Adjust* `rt->height` *accordingly.*

Solution:

```
void shrink(rt_tree *rt) {
    if (rt->height == 0) {
        return;
    }

    rt_node *n = rt->root;
    if ((count(n) == 1) && (n->entries[0] != NULL)) ||
        (count(n) == 0)) {

        rt->height--;
        rt->root = n->entries[0];
        free_node(n);
        shrink(rt);
    }
}
```

loop or recursion **(0.5 P)**, *shrinking condition* **(1.5 P)**, *adjusting* `rt` **(1.0 P)**, *free* **(0.5 P)** **Total: 15.0pt**