Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

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

Musterlösung Scheinklausur

02. 03. 2015

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 dem Konzeptblatt) 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 the draft page).
- Die Prüfung besteht aus 21 Blättern: 1 Deckblatt, 17 Aufgabenblättern mit insgesamt 5 Aufgaben und 3 Blätter Man-Pages.
  The examination consists of 21 pages: 1 cover sheet, 17 sheets containing 5 assignments, and 3 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	4	5	Total
Max. Punkte	12	12	12	12	12	60
Erreichte Punkte						
Schein (Bonus)						

Aufgabe 1: C Grundlagen

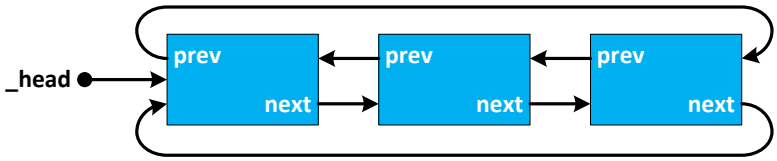
Assignment 1: C Basics

In der vorliegenden zirkulären, doppelt-verketteten Liste werden Elemente durch `entry` und die Zeiger auf das vorherige und nächste Element durch `prev` und `next` repräsentiert. Das letzte Element der Liste zeigt immer zurück auf das erste und umgekehrt. Der Kopf der Liste (`head`) zeigt auf das erste Element oder ist `NULL`, falls die Liste leer ist.

In the given circular, doubly-linked list, entries are represented by `entry` and the pointers to the previous and next entries by `prev` and `next`, respectively. The last entry always points back to the first one and vice versa. The list head (`head`) points to the first entry or is `NULL` if the list is empty.

```
typedef struct _entry {
    struct _entry *prev; // Pointer to previous entry in list
    struct _entry *next; // Pointer to next entry in list
    ...                  // Some data
} entry;

entry *_head = NULL;     // Pointer to first entry in list
```



a) Schreiben Sie eine Funktion, die ein neues Listenelement auf dem Heap alloziert. Bei Erfolg soll die Funktion einen Zeiger auf das Element zurückgeben, ansonsten `NULL`. **1 pt**

Write a function that allocates a new list entry on the heap. The function should return a pointer to the entry on success, `NULL` otherwise.

Lösung:

```
entry* allocateEntry(void) {
    return (entry*)malloc(sizeof(entry));
}
```

b) Schreiben Sie eine Funktion, die den Speicher eines bestehendes Listenelementes freigibt. Gehen Sie davon aus, dass das Element gültig (`e != NULL`) und nicht mehr Teil der Liste ist. **1 pt**

Write a function that frees the memory of an existing entry. Assume the entry to be valid (`e != NULL`) and not part of the list.

Lösung:

```
void freeEntry(entry *e) {
    free(e);
}
```

c) Schreiben Sie eine Funktion, die das Element `e` am Anfang der Liste einfügt. Gehen Sie davon aus, dass `e` auf ein gültiges Element zeigt (`e != NULL`). **3 pt**

Write a function that inserts the entry `e` at the head of the list. Assume `e` to point to a valid entry (`e != NULL`).

Lösung:

```
void insertEntry(entry *e) {
    if (_head == NULL) {
        e->prev = e;
        e->next = e;
    } else {
        e->next = _head;
        e->prev = _head->prev;

        _head->prev->next = e;
        _head->prev = e;
    }
    _head = e;
}
```

d) Schreiben Sie eine Funktion, die das Element `e` aus der Liste entfernt. Gehen Sie davon aus, dass `e` auf ein gültiges Element der Liste zeigt. **3 pt**

Write a function that removes the entry `e` from the list. Assume `e` to point to a valid entry in the list.

Lösung:

```
void removeEntry(entry *e) {
    if (e->next == e) {
        _head = NULL;
    } else {
        e->next->prev = e->prev;
        e->prev->next = e->next;

        if (_head == e) {
            _head = e->next;
        }
    }
}
```

e) Welche der folgenden Aussagen sind korrekt, welche sind inkorrekt? **2 pt**
(falsches Kreuz: -0.5P, kein Kreuz: 0P, korrektes Kreuz: 0.5P)

Which of the following statements are correct, which are incorrect?
(incorrectly marked: -0.5P, not marked: 0P, correctly marked: 0.5P)

korrekt/ *correct*	inkorrekt/ *incorrect*	
☐	■	Der Code einer Funktion wird zusammen mit ihren lokalen Variablen im Stack-Segment abgelegt. *The code of a function is stored together with its local variables in the stack segment.*
■	☐	Initialisierte globale Variablen liegen im Daten-Segment. *Initialized global variables are placed in the data segment.*
☐	■	Globale Konstanten liegen im BSS-Segment. *Global constants are placed in the bss segment.*
☐	■	`malloc()` erweitert dynamisch das Stack-Segment. `malloc()` *dynamically extends the stack segment.*

f) Was ist der Wert der Ausdrücke nach Ausführung des Programmschnipsels? **2 pt**

What is the value of the expressions after execution of the given code snippet?

```
int a[] =
  { 0x00112233, 0x44556677,
    0x8899aabb, 0xccddeeff,
    0xbaadf00d, 0xdeadc0de };

char b  = a[2];
int *p0 = &a[0];
int *p1 = p0 + sizeof(uint32_t);
// <-- Evaluate expression here
```

Lösung:

Expression	*Value*
`a[5] && 1`	*1*
`b`	*0xbb*
`*p0`	*0x00112233*
`*p1`	*0xbaadf00d*

Note that adding a value to a pointer increments the pointer by the value multiplied by the size of the type pointed at. Also note that `*p0` and `*p1` are expressions that dereference the pointers and thus return the value, not the address.

Total: 12.0pt

Aufgabe 2: Prozesse und Threads

Assignment 2: Processes and Threads

Credit Scheduling arbeitet wie folgt: Jeder Thread im System verfügt über eine bestimmte Menge Laufzeit-"Guthaben". Immer, wenn ein Thread ausgeführt wird, verbraucht er einen Teil seines Guthabens. Wenn ein Thread die CPU abgibt, wird die verbrauchte Zeit von seinem Guthaben abgezogen, und der Scheduler wählt einen anderen Thread mit positivem Guthaben aus. Das Guthaben aller Threads wird periodisch um einen definierten Betrag bis zu einem definierten Maximum aufgefüllt. Guthaben, das nach dem Auffüllen dieses Maximum übersteigt, verfällt.

In dieser Aufgabe sollen Sie einen einfachen Credit Scheduler für ein Multiprozessorsystem implementieren.

- Denken Sie daran, dass das System nebenläufig ist. d.h. es kann mehrere, parallele Aufrufe Ihrer Funktionen geben. Nutzen Sie Funktionen der *pthreads*-Bibliothek zur Synchronisation.
- Definieren Sie, falls nötig, zusätzliche globale Variablen oder Datenstrukturen. Gehen Sie davon aus, dass alle Variablen mit 0 initialisiert werden.
- Sie müssen die Laufzeiten der Threads selbst erfassen. Messen Sie das Guthaben und den Verbrauch der Threads in CPU-Zyklen. Verwenden Sie dazu die Funktion `clock()`.
- Nehmen Sie an, dass es eine feste Anzahl von Threads (`NUM_THREADS`) im System gibt, und dass Threads nie beendet werden.

Credit scheduling works as follows: Each thread in the system has a certain runtime-"credit". Whenever the thread runs, it consumes some of that credit. Whenever a thread yields the CPU, the scheduler selects another thread with positive credit to run. The credit of all threads is replenished periodically by adding a defined amount up to a defined maximum. If after replenishing, a thread has more credit than the maximum, any credit exceeding the maximum expires.

Your objective is to write a simple credit scheduler for a multiprocessor system.

- *Remember that your functions can be called multiple times concurrently. Use functions from the* phtreads*-library for synchronization.*
- *If necessary, define additional global variables or data structures. Assume that all variables are initialized to zero.*
- *You need to account each thread's runtime yourself. Measure each thread's credit and consumption in cpu cycles using the function* `clock()`*.*
- *Assume that there is a fixed number of threads in the system (*`NUM_THREADS`*) and that threads never terminate.*

a) Implementieren Sie die Funktion `void schedule(void)`. Diese Funktion wählt einen gerade **nicht laufenden** Thread mit **positivem Guthaben** aus, der als nächstes laufen soll. Falls kein entsprechender Thread existiert, starten Sie stattdessen den Idle-Thread mit der Thread ID `IDLE_THREAD`. Nehmen Sie an, dass eine Funktion `void restart_thread(int thread_id)` definiert ist, mit der Sie den ausgewählten Thread starten können. Achten Sie darauf, dass kein Thread mit positivem Guthaben verhungert!

Hinweise:

- `schedule()` muss nicht zwingend zurückkehren.
- `schedule()` wird nur von `yield()` aufgerufen.

6 pt

*Implement the function `void schedule(void)`. This function chooses a currently **not executing** thread with **positive credit** to be run next. If there is no such thread, launch the idle thread with the ID `IDLE_THREAD` instead. Assume that there is a function `void restart_thread(int thread_id)` which launches the chosen thread. Make sure that threads with positive credit cannot starve!*

Hints:

- *It is ok if `schedule()` does not return.*
- *`schedule()` is only called by `yield()`.*

b) Implementieren Sie die Funktion `void yield(int thread_id)`. Diese Funktion wird von Threads aufgerufen, um die CPU abzugeben. Gehen Sie davon aus, dass jeder Thread nach endlicher Zeit `yield()` aufruft und seine Thread ID korrekt angibt.

Hinweis: Die Threads müssen `yield()` nicht aufrufen, bevor ihr Guthaben verbraucht ist. Das Guthaben der Threads darf negativ sein. **3 pt**

Implement the function `void yield(int thread_id)`. This function is called by threads wanting to give up the CPU for the time being. Assume that each thread eventually calls `yield()` and passes its correct thread ID.

***Hint:** Threads do not necessarily call `yield()` before their entire credit is spent. Threads may have negative credit.*

c) Implementieren Sie die Funktion `void replenish(void)`. Diese Funktion erhöht das Guthaben aller Threads um den definierten Betrag `RP_CREDIT`. Das Maximum, bis zu dem das Guthaben aufgefüllt wird, wird durch `RP_MAX` vorgegeben. **3 pt**

Implement the function `void replenish(void)`. This function increases each thread's credit by the defined amount `RP_CREDIT`. The maximum credit a thread may have at any given time is defined by `RP_MAX`.

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

#define NUM_THREADS     42
#define IDLE_THREAD     -1
#define RP_CREDIT       10000
#define RP_MAX          100000

pthread_mutex_t mutex; // Assume that the mutex is properly initialized.

void restart_thread(int thread_id); // Does not return.
```

```
int credit[NUM_THREADS];
clock_t start_time[NUM_THREADS];
int running[NUM_THREADS];
int last_thread_to_run;

void schedule(void) {
  pthread_mutex_lock(&mutex);

  int next_thread = last_thread_to_run;

  do {
    next_thread++;
    if (next_thread >= NUM_THREADS)
      next_thread -= NUM_THREADS; // wrap to first thread in list

    if ((credit[next_thread] > 0) && (!running[next_thread])) {
      start_time[next_thread] = clock();
      running[next_thread] = 1;
      last_thread_to_run = next_thread;

      pthread_mutex_unlock(&mutex);
      restart_thread(next_thread);
    }

  } while (next_thread != last_thread_to_run);

  // no threads with positive credit
  pthread_mutex_unlock(&mutex);
  restart_thread(IDLE_THREAD);
}

void yield(int thread_id) {
  pthread_mutex_lock(&mutex);

  clock_t end = clock();
  credit[thread_id] -= (end - start_time[thread_id]);
  running[thread_id] = 0;

  pthread_mutex_unlock(&mutex); // Releasing the mutex in schedule() is ok
  schedule();
}

void replenish(void) {
  int i;
  pthread_mutex_lock(&mutex);

  for (i = 0; i < NUM_THREADS; ++i) {
    credit[i] += RP_CREDIT;
    if (credit[i] > RP_MAX)
      credit[i] = RP_MAX;
  }

  pthread_mutex_unlock(&mutex);
}
```

Total: 12.0pt

Aufgabe 3: Koordination von Prozessen

Assignment 3: Process Coordination

Implementieren Sie Synchronisationsprimitive mit Hilfe von atomaren Instruktionen. Verwenden Sie die unten gegebenen Funktionen.

In this assignment you will implement synchronization based on atomic instructions. You may use any of the functions given below.

```
// atomically add i to a variable
// in memory, return the previous
// value of the variable
int atomic_add(int *var, int i) {
    /* ... */
}
```

```
// atomically swap x with a variable
// in memory, then return the previous
// value of the variable
int atomic_swap(int *var, int x) {
    /* ... */
}
```

a) Implementieren Sie ein einfaches Spinlock. Schreiben Sie hierzu zwei Funktionen `acquire` und `release` zum Holen und Freigeben eines Spinlocks. Setzen Sie voraus, dass jedes Lock als `int`-Variable repräsentiert und mit `0` initialisiert ist. Ihre Implementierung soll alle 1000 Iterationen *busy waiting* die CPU freiwillig abgeben (yield). **3 pt**

Implement a simple spinlock. Write two functions `acquire` and `release` that acquire and release a spinlock, respectively. Assume that each lock is represented by an `int` variable, which is initialized to `0`. Your implementation should voluntarily give up the CPU (yield) every 1000 iterations of busy waiting.

Lösung:

```
void acquire(int *spinlock) {
    int i=0;
    while (atomic_swap(spinlock, 1)) {
        i++;
        if (i % 1000 == 0) sched_yield();
    }
}

void release(int *spinlock) {
    *spinlock = 0;
}
```

- *Correct loop for busy waiting* **(1 P)**
- *Correct frequency of calling `sched_yield`* **(1 P)**
- *Correct `release` function* **(1 P)**

b) Begründen Sie, warum ein einfaches Spinlock keine Fairness garantiert. **1 pt**

Explain why a simple spinlock does not guarantee fairness.

Lösung:

Atomic operations are executed in arbitrary order. **(0.5 P)** *Thus, it is unspecified in which order waiting threads will eventually acquire the lock, making guarantees such as fairness impractical.* **(0.5 P)**

c) Implementieren Sie eine Barrieren-Synchronisation als Funktion `void barrier()`. Die Funktion muss sich von mehreren Threads aufrufen lassen und darf erst zurückkehren, wenn alle beteiligten Threads sie betreten haben. Benutzen Sie eine der oben definierten atomaren Funktionen. Sie dürfen globale Variablen als Datenstruktur der Barriere einsetzen. Ihre Barriere braucht nur einmal zu funktionieren, Sie brauchen sie nicht zurückzusetzen. **4 pt**

Implement a barrier synchronization as a function `void barrier()`*. That function must be safe to be called by multiple threads and must only return once all participating threads have entered. Base your code on one of the atomic functions defined above. You may introduce global variables as the barrier's data structures. Your barrier needs to work only once, no need to re-initialize it.*

Lösung:

```
extern int N; // number of participating threads
int simple_barrier = 0;

void barrier() {
    atomic_add(&simple_barrier, 1);

    while(simple_barrier < N) {
        sched_yield(); // be nice, busy wait is fine, too.
    }
}
```

- *Correctly count waiting threads in barrier variable* **(2 P)**
- *Busy waiting till all threads have reached the barrier* **(2 P)**
- *You were not allowed to use mutexes. However, we will grade your answer with up to* **(1 P)** *if you provide a correct solution based on mutexes.*

d) Implementieren Sie eine Barrieren-Synchronisation wie in der vorherigen Teilaufgabe. Stellen Sie nun sicher, dass Ihre Implementierung mehrfach benutzt werden kann. Hinweise: Überlegen Sie sich, welcher Thread die Datenstruktur zurücksetzen soll. Wann ist ein geeigneter Zeitpunkt dafür? Ein Thread könnte erneut die Barriere betreten, bevor alle anderen Threads sie verlassen haben. **4 pt**

Implement a barrier as in the assignment above. However, make sure that your barrier implementation can be called several times in a row. Hints: Which thread should reset the barrier's data? When is a safe point in time to do that? A thread could enter the barrier again while other threads are still leaving.

Lösung:

```
extern int N; // number of participating threads
int barrier = 0;
void barrier() {

    // wait for previous re-initialization to complete
    while (barrier > N);

    int oldval = atomic_add(&barrier, 1);

    if (oldval != 0) {
        // regular thread, wait till all threads have entered
        while (barrier < N);
        // tell leader that I have passed
        atomic_add(&barrier, 1);

    } else /* oldval == 0 */ {
        // I am the leader, wait till threads have passed
        while (barrier < 2 * N - 1);
        // now reset the barrier
        barrier = 0;
    }
}
```

- *Threads re-entering the barrier need to wait for cleanup to be complete. No downgrading if you forgot that detail, though.*
- *Your solution still needs to work as a barrier.* **(2 P)**
- *Reset the barrier's variable(s) at the right instant, without introducing race conditions.* **(2 P)**

Total: 12.0pt

Aufgabe 4: Speicher

Assignment 4: Memory

Bei der Clock-Seitenersetzungsstrategie werden alle gültigen Seiten in einer zirkulären Liste gehalten. Jede Seite verfügt über ein Referenz-Bit (R-Bit), welches die MMU bei Zugriff auf 1 setzt. Der Zeiger der Uhr startet auf der ersten Seite. Bei Speicherdruck betrachtet der Algorithmus die aktuelle Seite nach folgender Regel:

- R-Bit 0 $\rightarrow$ Verwende Seite für Ersetzung, bewege Zeiger eine Seite weiter
- R-Bit 1 $\rightarrow$ Setze R-Bit auf 0 und suche weiter

In dieser Aufgabe sollen Sie den Algorithmus implementieren. Gehen Sie dabei von folgenden Rahmenbedingungen aus:

- Eine Seite ist über ihre Nummer (VPN) eindeutig definiert.
- Das System hat 1024 physische Seiten, die alle stets in Verwendung sind.
- Seiten werden nicht freigegeben, sondern nur durch den Algorithmus ersetzt.
- Eine physische Seite wird maximal von einer virtuellen Seite abgebildet.

The clock page replacement algorithm holds all valid pages in a circular list. Every page has a reference bit (r-bit), which the MMU sets to 1 on access. The clock hand starts at the first page. On memory pressure the algorithm examines the current page according to the following rule:

- *r-bit 0 $\rightarrow$ Use page as victim and advance hand*
- *r-bit 1 $\rightarrow$ Set r-bit to 0 and continue scanning*

In this question you will implement the clock algorithm. You can assume the following constraints:

- *Every page is uniquely identified by its number (vpn).*
- *The system has 1024 page frames, which are all permanently in use.*
- *Pages are not freed, but only replaced by the algorithm.*
- *A page frame is mapped at most by one page.*

a) Deklarieren Sie für Ihre Implementierung alle nötigen Variablen, um den Zustand der Uhr abzubilden. Setzen Sie passende Initialwerte. Sie müssen keine Initialisierung der verwendeten Seitenliste durchführen. **3 pt**

For your implementation, declare all necessary variables to represent the state of the clock. Set appropriate initial values. You do not have to initialize the clock's page list.

b) Implementieren Sie die Funktion `int replacePage(int new_vpn)`, die gemäß dem Clock-Algorithmus nach einer Seite sucht, die durch die neue Seite `new_vpn` ersetzt wird. Der Rückgabewert der Funktion ist die Nummer der zu ersetzenden Seite. Verwenden Sie die Funktionen `getRBit()` und `resetRBit()`, um auf das Referenz-Bit einer Seite zuzugreifen. **6 pt**

Implement the function `int replacePage(int new_vpn)`, which uses the clock algorithm to search for a page that should be replaced with the new page `new_vpn`. The function should return the number of the selected page for replacement. Use the functions `getRBit()` and `resetRBit()` to access a page's reference bit.

```
// Returns the status of the reference bit (1 set, 0 not set) for the
// page with the number supplied in vpn.
int getRBit(int vpn);

// Resets the reference bit of the supplied page to 0.
void resetRBit(int vpn);

#define NUM_FRAMES 1024 // Number of frames in the system

// We need to represent the mapped pages in the system. Since all frames
// are permanently in use we can assume a constant number of elements in
// the clock. We store for each frame, which page maps the respective frame.
// The frame number is thus the index into the array.
int pages[1024];

// We start the clock by setting the hand one element before the beginning
// of the list. This is necessary to allow the increment logic in replacePage().
int hand = -1;

int replacePage(int new_vpn) {
    // Iterate over the mappings until we find one
    // that has the reference bit not set.
    int vpn, ref;
    do {
        // Advance the hand to the next page. Wrap it if necessary.
        hand = (hand + 1) % NUM_FRAMES;

        vpn = pages[hand];
        ref = getRBit(vpn);
        resetRBit(vpn);
    } while (ref == 1);

    // We found a page for replacement. Set the new mapping.
    pages[hand] = new_vpn;

    // Return the page we selected for replacement.
    return vpn;
}
```

c) Welche andere Seitenersetzungsstrategie wird durch den beschriebenen Algorithmus approximiert? **1 pt**

Which other page replacement policy does the given algorithm approximate?

Lösung:

Least Recently Used (LRU)

d) Welche zusätzliche Information müsste der Clock-Algorithmus berücksichtigen, um geteilten Speicher (Shared Memory) zu unterstützen? **1 pt**

What additional information would the clock algorithm need to consider to support shared memory?

Lösung:

To support shared memory, the clock algorithm would need to know if a frame is mapped multiple times and what pages map the frame. This is necessary because evaluating a single page's reference bit is not enough for shared memory.

e) Bewerten Sie folgende Aussage: "Der Systemaufruf `malloc()` dient zur dynamischen Allozierung von Benutzerspeicher". Begründen Sie Ihre Aussage. **1 pt**

Evaluate the following statement: "The `malloc()` system call dynamically allocates user-space memory". Explain your answer.

Lösung:

The statement is wrong. `malloc()` is not a system call. **Total: 12.0pt**

Aufgabe 5: Dateizugriff

Assignment 5: File Access

Eine gegebene Bibliothek abstrahiert Dateizugriffsoperationen. Es sind zwei Funktionen `isFileReadable()` und `isFileWriteable()` implementiert, die die Zugriffsrechte des aktuellen Benutzers auf eine Datei überprüfen.

A given library abstracts file access operations. Two functions `isFileReadable()` and `isFileWriteable()` are defined, which check the user's access permissions on a file.

```
// Returns 1, if the file can be read, 0 otherwise.
int isFileReadable(const char *filename){
        return (open(filename, O_RDONLY) >= 0) ? 1 : 0;
}
```

a) Welches Problem kann bei einer häufigen Verwendung dieser Funktion auftreten? **1 pt**

Which problem can occur, if this function is used frequently?

Lösung:

This function never closes the file descriptor returned by open, thus it exhausts system resources.

b) Implementieren Sie die Funktion `isFileWriteable()`, die überprüft ob für die gegebene Datei Lese- und Schreibrechte vorhanden sind. Vermeiden Sie das in Teilaufgabe a) identifizierte Problem. **1 pt**

Implement the function `isFileWriteable()`, which checks if the caller has read and write access to the given file. Avoid the problem identified in question a).

Lösung:

```
// Returns 1, if the file can be read and written, 0 otherwise.
int isFileWriteable(const char *filename) {
        int handle = open(filename, O_RDWR);
        if (handle >= 0) {
                close(handle);
                return 1;
        } else {
                return 0;
        }
}
```

c) Was stellen diese Funktionen auf einem multitaskingfähigem Betriebssystem nicht sicher? **1 pt**

What do these functions not ensure on a multi-tasking operating system?

Lösung:

This file permissions can be changed by a concurrently running process after the function returns.

Das Programm `tree` listet die Verzeichnisstruktur ausgehend von dem angegebenen Pfad hierarchisch auf. Das Programm zählt dabei wieviele Unterverzeichnisse es gibt und zeigt die Anzahl im fehlerfreien Fall am Ende an. Es ignoriert dabei alle Verzeichnisse, deren Namen mit einem Punkt beginnen – z.B. `/home/user/.ssh`. In dieser Aufgabe sollen Sie Teile des Programms implementieren.

The program `tree` prints the directory hierarchy, starting from the specified path. The program counts all encountered subdirectories and prints the number at the end if no error occurred. It ignores all directories, starting with a dot – e.g., `/home/user/.ssh`. Your objective is to implement some of the program's main functions.

Gewünschte Ausgabe von/*Desired output of:* `tree /home/user`

/home/user
-documents
--exams
-music
Directories: 3

d) Schreiben Sie die Funktion `isDir()`, die überprüft ob ein gegebener Verzeichniseintrag ein Unterverzeichnis ist, dessen Name nicht mit einem Punkt beginnt. Nehmen Sie an, dass das Programm nur auf ext2 Dateisystemen ausgeführt wird. **1 pt**

Implement the function `isDir()`, which checks if a given directory entry is a subdirectory, whose name does not start with a dot. Assume the program runs only on ext2 file systems.

e) Schreiben Sie die Funktion `enumDirs()`, die für einen gegebenen Pfad die Hierarchie der Unterverzeichnisse ausgibt und dabei den Verzeichniszähler aktualisiert. Nutzen Sie dabei die gegebenen Hilfsfunktionen. **6 pt**

Implement the function `enumDirs()`, which outputs the hierarchy of subdirectories for a given path and updates the directory counter. Use the supplied helper functions.

f) Schreiben Sie die Einsprungsfunktion `main()`. Das Programm soll mit dem aufzulistenden Verzeichnis als einzigen Parameter gestartet werden. Führen Sie eine Überprüfung der Parameter durch und rufen Sie anschließend die Funktion `enumDirs()` auf. Achten Sie auf die korrekte Ausgabe des Programms. **2 pt**

Implement the entry-point function `main()`. The program should be called with the directory to list as only parameter. Perform a parameter check and call the `enumDirs()` function. Pay attention to the correct output of the program.

```
int dirCount = 0;       // Directory counter
void printStats(void); // Prints the directory counter.

// Appends newDir to curDir and adds the path separator.
// Memory is handled internally - do not free the returned string.
// appendDir("home", "documents") => "home/documents"
char* appendDir(const char *curDir, const char *newDir);

// Prints a single directory with the supplied number of leading
// dashes to indicate the directory depth.
// printDir("home", 5) => "-----home\n"
void printDir(const char *dir, int depth);
```

```c
// Returns 1, if dentry is a directory, 0 otherwise.
int isDir(const dirent *dentry) {
    return ((dentry->d_type == DT_DIR) && (dentry->d_name[0] != '.'));
}

// Returns 0 on success, -1 otherwise.
int enumDirs(const char *dir, int depth) {
    DIR *handle;
    struct dirent *dentry;
    int status = 0;

    handle = opendir(dir);
    if (handle == NULL) {
        return -1; // or errno
    }

    dentry = readdir(handle);
    while (dentry != NULL) {
        if (isDir(dentry)) {
            char *subDir = appendDir(dir, dentry->d_name);

            printDir(dentry->d_name, depth);
            status |= enumDirs(subDir, depth + 1);

            dirCount++;
        }
        dentry = readdir(handle);
    }

    closedir(handle);
    return status;
}

// Returns 0 on success, -1 otherwise.
int main(int argc, char *argv[]) {
    int status = -1;
    if (argc == 2) {
        printf("%s\n", argv[1]);
        status = enumDirs(argv[1], 0);
        if (status == 0) {
            printStats();
        }
    }

    return status;
}
```

Total: 12.0pt