Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

Prof. Dr. Michael Beigl
Jonas Kiemel, M.Sc.
Mathias Gottschlag, M.Sc.

Musterlösung Scheinklausur
03.04.2019

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 `calculate()`.

Consider the following function `calculate()`.

```
1 double calculate(int *ptr, int n) {
2   assert(n != 0);
3   int result = 0;
4   int *ptr2 = ptr + n;
5   while (ptr < ptr2)
6     result += *ptr++;
7   return result / n;
8 }
```

Was berechnet die Funktion? **1 pt**

What does the function calculate?

Lösung:

`calculate()` returns the average of the values in the integer array of length `n` starting at `ptr`.

Welches Problem kann bei der Addition `+=` in Zeile 6 auftreten? Wie könnte dieses Problem zur Laufzeit erkannt werden? **2 pt**

Which issue might occur when the addition `+=` in line 6 is executed? How could this issue be detected at runtime?

Lösung:

The addition might overflow **(1 P)** *the `result` variable. Signed integer overflow is undefined behavior, so we cannot simply check for overflow after the addition. Instead, we could perform the addition with a larger type (such as `long`) and check whether it still fits the `int` afterwards* **(1 P)**. *Alternatively, we could compare `result` with the maximum (or minimum) `int` value subtracted by `*ptr`.*

Common wrong answers:

(a) Anything unrelated to the addition "+=", like issues with dereferencing and incrementing `ptr`.

(b) Atomicity issues. `result` is a function-local variable, so each thread would operate on its own copy.

(c) Detecting overflow by checking the sign after the addition. The compiler is allowed to remove such a check as it is undefined behavior.

Welches Problem kann bei der Division in Zeile 7 auftreten? Wie kann das Problem vermieden werden? **2 pt**

Which issue might occur when executing the division in line 7? How could this issue be avoided?

Lösung:

Dividing two integers results in another integer which is then converted to `double`*. Consequently, the result might be imprecise. Converting one of the operands to* `double` *would avoid the issue.*

Common wrong answers:

(a) Division by 0. The assert will abort in this case, so the division never happens.

*(b) The assert might be disabled in certain configurations, so division by 0 can still happen. The assertion at the beginning of the function is a precondition and indicates that the function should never be called with n = 0. Omitting assertions in "release"-configurations is a performance optimization that will not change the behavior of correct programs. (***(0.5 P)** *for the issue,* **(0.5 P)** *for a reasonable solution (e.g., returning 0)).*

(c) Issues with negative numbers. For negative n, `ptr < ptr2` *is true, so the function will return 0.*

b) Nennen Sie eine C-Bibliotheksfunktion, die nur manchmal einen Systemaufruf durchführt. Erklären Sie, in welchen Fällen ein Systemaufruf durchgeführt wird. **1 pt**

Name a C library function which does not always perform a system call. Explain the cases in which the function performs a system call.

Lösung:

stdio functions (e.g., `printf`*,* `fread`*,* `fwrite`*): only perform system calls when the standard library's buffer is empty or full, or (in case of line-buffered streams) at newlines.*

`malloc`*: can hand out free memory on the heap and only needs system calls to grow the heap.*

`free`*: usually just marks memory as free in the local allocator data structures, but may return larger allocations to the operating system with a system call.*

c) Betrachten Sie die folgende Funktionen `a()` und `b()`.

Consider the following functions `a()` *und* `b()`*.*

```
void set_configuration(struct config *);

void a() {
  struct config c;
  memset(&c, 0, sizeof(struct config));
  c.verbose = 1;
  set_configuration(&c);
}

void b() {
  struct config *c = malloc(sizeof(struct config));
  memset(c, 0, sizeof(struct config));
  c->verbose = 1;
  set_configuration(c);
}
```

Begründen Sie, warum es nicht möglich ist, eine Funktion `set_configuration()` zu schreiben, die von beiden Funktionen `a()` und `b()` korrekt aufgerufen werden kann. **2 pt**

Explain why it is not possible to write a function `set_configuration()` *that both* `a()` *and* `b()` *can call correctly.*

Lösung:

Function `b()` *allocates* `struct config` *on the heap but does not free it later on, so it expects* `set_configuration()` *to take ownership of the memory and call* `free()` *on the pointer. This would be wrong for function* `a()`*, which passes a pointer to the stack instead. Calling* `free()` *on that pointer would result in undefined behavior.*

(2 P) *for explaining the* `free()` *issue, otherwise* **(0.5 P)** *for each:*

(a) `a()` *allocates on stack,* `b()` *allocates on heap*

(b) `set_configuration()` *can keep pointer passed by* `b()` *but not the one passed by* `a()`

Common wrong answers:

(a) There is some difference between a pointer to `struct config` *and an address of a* `struct config`*. The address-of operator in C (*`&`*) returns a pointer.*

(b) Function `a()` *is doing call-by-value whereas* `b()` *is doing call-by-reference. There are no references in C, every function passes by-value. Both functions here are passing pointers by value.*

Welches Problem könnte in Funktion `a()` auftreten, wenn der `memset()`-Aufruf entfernt wird? **1 pt**

Which issue might occur in function `a()` *when the call to* `memset()` *is removed?*

Lösung:

Variables declared on the stack have undefined contents, so they need manual initialization. After removing the call to `memset`*,* `set_configuration()` *might read uninitialized memory if* `struct config` *has additional fields.*

Vervollständigen Sie die Funktion `a_alternative()`, sodass sie funktional äquivalent zur Funktion `a()` ist. Verwenden Sie dabei keinerlei Funktionsaufrufe. **1 pt**

Complete the function `a_alternative()` *in a way that makes it functionally equivalent to the function* `a()`*. Do not call any functions in your solution.*

Lösung:

```
void a_alternative() {
  struct config c = {0};

  c.verbose = 1;
  set_configuration(&c);
}
```

Alternative: manual `memset()`*-like loop.*

```
void a_alternative() {
  struct config c;
  for (char *ptr = (char *) &c; ptr < (char *) &c + sizeof(struct config); ptr++)
    *ptr = 0;
```

```
  c.verbose = 1;
  set_configuration(&c);
}
```

d) In dieser Aufgabe sollen Sie Funktionen zur Nutzung eines Bitvektors implementieren.

- Ein Bitvektor speichert eine Menge an Flags.
- Jedes Flag wird durch genau ein Bit dargestellt. Wenn das Flag gesetzt ist, hat das Bit den Wert 1, ansonsten 0.
- Die einzelnen Flags werden mit einem Index assoziiert.
- Ihre Bitvektor-Implementierung soll eine beliebige, aber nach der Initialisierung feste Anzahl an Flags speichern können.

In this part, you will implement functions for using a bit vector.

- *A bit vector saves a set of flags.*
- *Every flag is represented by exactly one bit in the bit vector. If the flag is set, the bit is set to 1, otherwise the bit is set to 0.*
- *Every flag is associated with an index.*
- *Your bit vector implementation should save an arbitrary number of flags that is fixed after initialization.*

Wählen Sie einen geeigneten Integer-Typ für den Bitvektor. **0.5 pt**

- Der Typ sollte so gewählt werden, dass auf heute üblichen Prozessorarchitekturen effizient damit gerechnet werden kann.
- Achten Sie darauf, dass Ihre Implementierung der übrigen Teilaufgaben korrekt mit dem hier gewählten Typ funktioniert. Schreiben Sie portablen Code.

Choose an appropriate integer type for the bit vector.

- *Choose the type so that efficient operations are possible on current processor architectures.*
- *Make sure your implementation of the remaining parts of this excercise works correctly with the integer type you choose. Write portable code.*

Lösung:

```
typedef
  uint32_t /* or uint64_t or unsigned int or unsigned long */
  bv_int;
```

Note: For efficiency, we want a type that is close to the architecture's natural integer size. The C standard specifies that the `int` *type has such a size, but using a fixed-width type makes reasoning about individual bits a bit easier.*

(0.5 P) *for correct size and type.*

Signed is okay, but may need extra casts for bit shifts to avoid undefined behavior.

Declaring a struct is unnecessary, but okay.

Declaring a pointer or array type is not okay as it does not easily fit the signatures of the following functions. (Okay if all other functions are implemented correctly for the double pointers.)

Vervollständigen Sie die Definition von `bpe`. Die Konstante soll die Anzahl an Bits in einem Element des Bitvektors enthalten. **0.5 pt**

Complete the definition of `bpe`. The constant should contain the number of bits per element of the bit vector.

Lösung:

```
/* Number of bits per element */
static const size_t bpe = 8 * sizeof(bv_int);
```

Putting a constant here is okay for fixed-width integer types (e.g., 32 for `uint32_t`).

Vervollständigen Sie die Funktion `div_round_up()`, die die übergebenen Parameter dividiert und das Ergebnis aufrundet. **1 pt**

Complete the function `div_round_up()`. It should divide the passed arguments and should round the result up.

$$result = \left\lceil \frac{dividend}{divisor} \right\rceil$$

Lösung:

```
size_t div_round_up(size_t dividend, size_t divisor) {
  return dividend / divisor + (dividend % divisor != 0);
}
```

(1 P) *if everything is correct.*

(0.5 P) *if the implementation does not handle certain numbers correctly:*

- *Solutions using `double` have precision issues for large numbers if `size_t` has 64 bit.*
- *`(dividend - 1)` will overflow if dividend is 0.*

Vervollständigen Sie die Funktion `allocate_bitvector()`, die einen Bitvektor allokiert, der die gegebene Anzahl an Flags speichern kann. **1.5 pt**

- Der Bitvektor soll so initialisiert werden, dass keine Flags gesetzt sind.
- Allokieren Sie nicht mehr Speicher als notwendig.

Complete the function `allocate_bitvector()`. It should allocate a bit vector that can save the given number of flags.

- *In the new bit vector, all flags should be unset.*
- *Do not allocate more memory than necessary.*

Lösung:

```
bv_int *allocate_bitvector(size_t num_flags) {
  /* Note: need to round up the result of the division. */
  size_t size = sizeof(bv_int) * div_round_up(num_flags, bpe);
  bv_int *vec = malloc(size);
  memset(vec, 0, size);
  return vec;
}
```

(0.5 P) *for correct size calculation,* **(0.5 P)** *for memory allocation call and* `return`*,* **(0.5 P)** *for zero-initializing the array*

Vervollständigen Sie die Funktion `set_flag()`, die das Flag an i-ter Stelle in dem Bitvektor setzt. **2.5 pt**

Complete the function `set_flag()`*. It should set the bit vector's flag at position i.*

Lösung:

```
void set_flag(bv_int *vec, size_t i) {
  vec[i / bpe] |= ((bv_int) 1) << (i % bpe);
}
```

(0.5 P) *for correct index in* `vec`*,* **(0.5 P)** *for correct index in the vec item,* **(0.5 P)** *for bit shift,* **(0.5 P)** *for shifting the correct type (must have correct size, must be unsigned),* **(0.5 P)** *for correct OR*

Note*: The question does not specify the bit layout, so it is also correct to put the first flag at bit* $(bpe - 1)$*. For the same reason, one-based numbering is also okay, but not advisable as it makes the implementation more complex. In both cases, the implementations of* `set_flag` *and* `find_first_set` *need to be consistent.*

Vervollständigen Sie die Funktion `find_first_set()`, die den Index des ersten gesetzten Flags im Bitvektor zurückgibt. **4 pt**

- Der Parameter `num_flags` ist der Wert, der zuvor an `allocate_bitvector()` übergeben wurde.
- Falls keine Flags gesetzt sind, soll `num_flags` zurückgegeben werden.

Complete the function `find_first_set()`*. It should return the index of the first set bit in the bit vector.*

- *The parameter* `num_flags` *is the value that was previously passed to* `allocate_bitvector()`*.*
- *If there are not any flags set, the function should return* `num_flags`*.*

Lösung:

```
size_t find_first_set(const bv_int *vec, size_t num_flags) {
  size_t index = 0;
  bv_int e;
  while (index < num_flags) {
    if (*vec) {
      e = *vec;
      while (!(e & 1)) {
```

```
        e >>= 1;
        index++;
      }
      return index;
    }
    index += bpe;
    vec++;
  }
  return num_flags;
}
```

(1.5 P) *for correct loop (stepping through* `vec`*),* **(0.5 P)** *for correctly calculating the flag index for each element,* **(1.5 P)** *for correctly finding the first set bit in the non-zero element,* **(0.5 P)** *for returning* `num_flags` *if nothing is set*

Note*: In real code outside of the exam, this solution could be optimized by replacing the inner loop with a "find first set" or "count trailing zeros" instruction. This is not possible in standard C, but most compilers provide special intrinsic functions (e.g.,* `__builtin_ffs` *in GCC) that compile to such instructions depending on the target architecture.*

Alternate (less efficient) solution:

```
size_t find_first_set(const bv_int *vec, size_t num_flags) {
  for (size_t i = 0; i < num_flags; i++) {
    if ((vec[i / bpe] >> (i % bpe)) & 1)
      return i;
  }
  return num_flags;
}
```

(1 P) *for correct loop,* **(2.5 P)** *for correctly checking each flag,* **(0.5 P)** *for returning* `num_flags` *if nothing is set*

Total: 20.0pt

Aufgabe 2: Erkennung von Dateiänderungen

Assignment 2: Detection of Modified Files

Sie sollen ein Programm schreiben, das eine Liste von Dateien betrachtet und ihr Änderungsdatum mit dem auf einem Server gespeicherten Datum vergleicht und dadurch Änderungen der Dateien erkennt. Das Programm soll die folgenden Eigenschaften haben:

- Die Liste der zu überprüfenden Dateien wird aus einer Textdatei gelesen.
- Das vorherige Änderungsdatum wird vom Server durch Kommunikation über POSIX Message Queues abgefragt.
- Binden Sie die in den Teilaufgaben notwendigen C-Header in dem gekennzeichneten Bereich ein.
- Außer in Teilaufgabe a) müssen Sie keine Fehlerbehandung implementieren. Sie können in den restlichen Teilaufgaben davon ausgehen, dass bei Systemaufrufen und Funktionen der Standardbibliothek keine Fehler auftreten.
- Geben Sie jegliche in ihrem Code angeforderte Resourcen wieder frei.

Your task is to write a program which analyzes a list of files and compares their modification date with a modification date stored on a server in order to detect changes to the files. The program shall have the following properties:

- *The list of the files to be checked is read from a text file.*
- *The previous modification date is fetched from the server via communication through POSIX message queues.*
- *Include necessary C headers in the marked area.*
- *Except in subtask a) you do not have to implement error handling. In the other subtasks, you can assume that system calls and standard library functions do not fail.*
- *Free all resources allocated in your code.*

```c
/* include statements for the required C headers */
#include <string.h>    /* strlen */
#include <stdio.h>     /* printf, fopen, fclose, fgets */
#include <sys/types.h> /* stat */
#include <sys/stat.h>  /* stat, mq_open */
#include <unistd.h>    /* stat */
#include <fcntl.h>     /* mq_open */
#include <mqueue.h>    /* mq_open, mq_close, mq_send, mq_receive */

#define MAX_PATH 256

/* global variables */
/* name of the message queue to send the path of a file to the server */
const char *mq_path_name = "/mq-path";
/* name of the message queue to receive the modification date from the server */
const char *mq_mtime_name = "/mq-mtime";
```

a) Vervollständigen Sie die Funktion `getTimeOfModification()`, welche die Zeit der letzten Änderung einer Datei zurückgibt. **3 pt**

- Die Funktion erhält als Argument den absoluten Pfad zur Datei.
- Fehler, welche von Funktionsaufrufen innerhalb von `getTimeOfModification()` verursacht werden, sollen zu einem Rückgabewert von 0 führen.

Complete the function `getTimeOfModification()` which returns the time of the last modification of a file.

- *The function is passed the absolute path to the file.*
- *Errors caused by function calls within `getTimeOfModification()` shall lead to a return value of 0.*

Lösung:

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

time_t getTimeOfModification(const char *path){
    struct stat buf;
    if (stat(path, &buf) == -1) {
        return 0;
    }
    return buf.st_mtime;
}
```

Adding the necessary includes **(1.0 P)**, *calling `stat` with a pointer to a `struct stat`* **(1.0 P)**, *checking the return code and returning 0* **(0.5 P)**, *returning `st_mtime`* **(0.5 P)**.

b) Vervollständigen Sie die Funktion `checkFileStatus()`, die das dem Server bekannte Änderungsdatum einer Datei holt und prüft, ob das aktuelle lokale Änderungsdatum abweicht. **7 pt**

- Die Funktion erhält als Argument den absoluten Pfad zur Datei.
- Die Funktion schickt den absoluten Pfad der Datei über eine POSIX Message Queue mit dem Namen "/mq-path" synchron an einen Server.
- Ist die Datei dem Server bekannt, so sendet der Server das Datum der letzten Modifikation der Datei, gespeichert als `time_t`, synchron über eine Message Queue mit Namen "/mq-mtime" an Ihr Programm. Anderenfalls sendet der Server eine nur aus Nullen bestehende Nachricht.
- Stimmt das zugeschickte Modifikationsdatum mit dem lokal bestimmten Änderungsdatum überein, so soll die Funktion 1 zurückgeben, ansonsten 0.
- Der Server übernimmt die korrekte Initialisierung der Message Queues, sodass Ihr Programm die Queues öffnen kann ohne sie zuvor erzeugen zu müssen.
- Die von `checkFileStatus()` zu sendende Nachricht soll eine Priorität von 0 haben.
- Nehmen Sie an, dass die Message Queue "/mq-mtime" vor dem Anfordern der Daten leer ist. Nachrichten können ohne Unterbrechung in einem Stück eingelesen werden.
- Alle Strings sollen null-terminiert sein. Sie können annehmen, dass Pfade aus maximal MAX_PATH Zeichen bestehen.

Complete the function `checkFileStatus()` *which fetches the modification date of a file as known to the server and checks whether it matches the current local date of modification.*

- *The function is passed the absolute path to the file.*
- *The function synchronously sends the absolute file path to the server via a posix message queue with the name "/mq-path".*
- *If the file is known to the server, the server sends the last modification date of the file (stored as* `time_t`*) synchronously to your program via a message queue with the name "/mq-mtime". Otherwise, the server sends a message consisting of zeros.*
- *If the received modification date matches the locally determined date, the function shall return 1, else it shall return 0.*

- *The server undertakes the correct initialization of the message queues meaning that your program can open the queues without creating them before.*
- *The message to be sent by* `checkFileStatus()` *should have a priority of 0.*
- *Assume that the message queue "/mq-mtime" is empty before requesting data. All messages can be read in one piece without interruptions.*
- *All strings shall be null-terminated. You can assume that no paths will be longer than MAX_PATH characters.*

Lösung:

```
#include <fcntl.h>           /* For O_* constants */
#include <mqueue.h>

int checkFileStatus(const char *path){
    time_t mtimeLocal = getTimeOfModification(path);
    mqd_t mq_path;
    mqd_t mq_mtime;
    mq_path = mq_open(mq_path_name, O_WRONLY);
    mq_mtime = mq_open(mq_mtime_name, O_RDONLY);
    mq_send(mq_path, path , strlen(path) + 1 , 0);
    /* strlen(path) + 1 or MAX_PATH */
    /* otherwise the terminating \0 character is not sent */
    time_t mtimeServer = 0;
    mq_receive(mq_mtime, (char*) &mtimeServer, sizeof(time_t), NULL);
    mq_close(mq_path);
    mq_close(mq_mtime);
    if (mtimeLocal == mtimeServer){
        return 1;
    } else {
        return 0;
    }
}
```

Adding the required includes **(1.0 P)**, *getting the modification time* **(0.5 P)**, *opening the two queues* **(1.5 P)**, *sending the path* **(1.5 P)**, *receiving the time* **(1.5 P)**, *closing the queues* **(0.5 P)**, *checking the time* **(0.5 P)**.

Warum ist dieses Verfahren nicht geeignet, wenn mehrere Programme gleichzeitig mit dem Server kommunizieren sollen? Was müsste geändert werden, um solch gleichzeitigen Zugriff auf den Server zu ermöglichen? **2 pt**

Why is this method not suitable if multiple programs shall communicate with the server concurrently? What needs to be changed in order to allow such concurrent access to the server?

Lösung:

- *With the implementation given above, the server cannot detect the sender of a request. All clients will try to receive messages from the same message queue. A client might therefore receive messages that were intended for other clients* **(1 P)**.
- *Potential mitigation include:*
  - *Encode information about the client in the message that is sent to the server so that the server can create one message queue per client for its answer.*
  - *Use a shared mutex so that only one client can send and receive messages at one time*

  (1 P) *for a valid mitigation*

c) Vervollständigen Sie die Funktion `readFileList()`, welche die Textdatei mit der Liste der zu prüfenden Dateien einliest. Für jede aufgelistete Datei prüft die Funktion, ob die Datei seit der letzten Synchronisation mit dem Server verändert wurde. **8 pt**

- Die Funktion erhält als Argument den absoluten Pfad zur einzulesenden Textdatei.
- Jede Zeile der Textdatei enthält den absoluten Pfad zu einer zu überprüfenden Datei.
- Wenn eine Datei nicht modifiziert wurde, sollen Sie den Pfad der Datei und "not modified" auf der Konsole ausgeben. Ansonsten sollen Sie den Pfad und "modified or unknown" ausgeben.
- Verwenden Sie `fgets()`, um die Textdatei zeilenweise einzulesen. Berücksichtigen Sie, dass `fgets()` den newline-Charakter `'\n'` am Ende jeder Zeile nicht abschneidet, dieser jedoch nicht Teil des Pfades ist.
- Sie dürfen annehmen, dass jede Zeile der Textdatei maximal `MAX_PATH` Zeichen enthält.
- Leere Zeilen in der Textdatei sollen ignoriert werden.

Complete the function `readFileList()` which reads the text file with the list of files to be checked. For each listed file, the function checks, whether the file was changed since the last synchronization with the server.

- *The function is passed the absolute path to the text file to be read.*
- *Each line of the text file contains an absolute path to a file to be checked.*
- *If a file was not modified, you shall print the path of the file and "not modified" to the console. Otherwise the path and "modified or unknown" shall be printed.*
- *Use `fgets()` to read the text file line by line. Note that `fgets()` does not remove the newline character `'\n'` at the end of each line. This newline character is not part of the file path.*

- *You may assume that each line of the text file contains at most* `MAX_PATH` *characters.*
- *Empty lines in the text file shall be ignored.*

Lösung:

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

void readFileList(const char *path){
    FILE* file = fopen(path, "r");
    char line[MAX_PATH+1]; // char line[MAX_PATH] also accepted
    while (fgets(line, sizeof(line), file)) {
        /* fgets does not strip the terminating \n */
        if (strlen(line) != 0 && line[strlen(line) - 1] == '\n') {
            line[strlen(line)-1] = '\0';
        }
        if (strlen(line) >0) {
            if (checkFileStatus(line)) {
                printf("%s:␣not␣modified.", line);
            } else {
                printf("%s:␣modified␣or␣unknown.", line);
            }
        }
    }
    fclose(file);
}
```

Adding the necessary includes **(1.0 P)**, *opening the file* **(1.0 P)**, *declare an array of characters like line with the correct size* **(0.5 P)** *loop to read all lines from the file* **(0.5 P)**, *correct fgets call* **(1.0 P)**, *ignoring empty lines* **(0.5 P)**, *removing the newline* **(1.5 P)**, *checking whether the file has been modified* **(0.5 P)**, *printing the result* **(1.0 P)**, *closing the file* **(0.5 P)**.

Total: 20.0pt

Aufgabe 3: Multilevel Feedback Queue

Assignment 3: Multilevel Feedback Queue

Ein Multilevel Feedback Queue Scheduler (MLFB) bestimmt Prioritäten dynamisch basierend darauf, ob die Tasks ihre Zeitscheiben voll ausnutzen oder nicht. Implementieren Sie einen solchen Scheduler mit den folgenden Eigenschaften:

- Der Scheduler unterscheidet zwischen drei unterschiedlichen Prioritäten. Zwischen Task gleicher Priorität wird Round-Robin-Scheduling verwendet.
- Wenn ein Task seine Zeitscheibe vollständig ausnutzt, wird seine Priorität um eins reduziert.
- Wenn ein Task seine Zeitscheibe dreimal direkt in Folge nicht vollständig ausnutzt, wird seine Priorität um eins erhöht.
- Sie können davon ausgehen, dass zu jedem Zeitpunkt mindestens ein Task lauffähig ist, d.h. mindestens eine Queue ist nicht leer.
- Alle Zeiten werden als Vielfaches der Timer-Periode angegeben.

A multilevel feedback queue scheduler (MLFB) determines priorities dynamically depending on whether tasks fully utilize their time slices or not. Implement such a scheduler with the following properties:

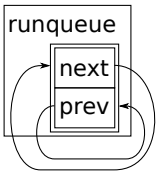
- *The scheduler distinguishes between three different priorities. Round-robin scheduling is used for tasks of equal priority.*
- *If a task fully uses its time slice, its priority is reduced by one.*
- *If a task does not fully use its time slice three times directly in a row, its priority is increased by one.*
- *You can assume that at all times at least one task is runnable, i.e., at least one queue is non-empty.*
- *All times are stated as multiples of the timer period.*

```
struct queue_entry {
    struct queue_entry *next;
    struct queue_entry *prev;
};

struct task {
    uint32_t ts;           // Remaining part of current time slice
    uint32_t prio;         // Priority of the task (index into queue array)
    struct queue_entry rq; // Scheduler queue links

    uint32_t block_cnt; // Counter of partially unused timeslices - when the
                        // counter reaches 3, increase the priority of the task.
                        // The counter is set to 0 whenever the timeslice expires.

    // ... (task context omitted for brevity)
};

// list of runnable (not blocked) tasks of the same priority
struct runqueue {
    struct queue_entry head;
};
```

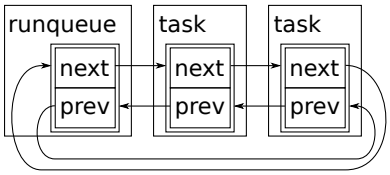
```
#define NUM_PRIO 3 // Number of different priorities
                   // (2 is the highest priority, 0 the lowest)
// Length of the time slice of each priority
const int TS_LEN[NUM_PRIO] = { 20, 10, 5 };

struct runqueue rqs[NUM_PRIO]; // MLFB queues (one per priority)
struct task *current;          // Currently executed task

void context_switch(struct task *t); // Performs a context switch
                                     // if the task has changed.
```

a) Jede der Queues ist, wie in der folgenden Abbildung zu sehen, als doppelt verkettete Liste realisiert. Leere Queues enthalten der Einfachheit halber einen Zyklus, sodass `next` und `prev` des Listenkopfes auf den Listenkopf selbst zeigen.

Each of the queues is, as shown in the following figure, implemented as a double linked list. For the sake of simplicity, empty queues contain a cycle, so that `next` and `prev` of the list head point back to the list head.



(a) Empty task queue



(b) Task queue with two tasks

Vervollständigen Sie die Funktion `from_task()`, die aus einem Pointer auf einen Task den Pointer auf den Queue-Eintrag (`rq`-Feld) des Tasks berechnet. **1.0 pt**

Complete the function `from_task()` which takes a pointer to a task and calculates the pointer to the queue entry (`rq` field) of the task.

Lösung:

```
struct queue_entry *from_task(struct task *t) {
    return &t->rq;
}
```

Access to `rq` member **(0.5 P)**, *taking a pointer to the member* **(0.5 P)**.

Vervollständigen Sie die Funktion `from_list_entry()`, die umgekehrt aus einem Pointer auf einen Queue-Eintrag den Pointer auf den dazugehörigen Task berechnet. Nehmen Sie dabei an, dass der Compiler die Reihenfolge der Felder der Struktur nicht ändert und kein Padding zwischen den Feldern einfügt. **1.5 pt**

Complete the function `from_list_entry()` which vice versa takes a pointer to a queue entry and computes the pointer to the corresponding task. Assume that the compiler does not reorder the fields of the struct and does not insert padding between the fields.

Lösung:

```
struct task *from_queue_entry(struct queue_entry *e) {
    return (struct task*)((char*)e - 8);
}
```

Subtraction of 8 bytes from the pointer **(0.5 P)**, *correct type conversion* **(1 P)**.

Arithmetic on `void*` *is a GCC extension and undefined in standard C, as* `void` *does not have a size. Therefore,* `(void*)e - 8` *is not valid.*

Vervollständigen Sie die Funktion `append()`, die den Task `t` an das Ende der Queue `rq` anhängt. **3 pt**

Complete the function `append()` *which appends the task* `t` *at the end of the queue* `rq`.

Lösung:

```
void append(struct runqueue *rq, struct task *t) {
    struct queue_entry *e = from_task(t);
    e->next = &rq->head;
    e->prev = rq->head.prev;
    rq->head.prev->next = e;
    rq->head.prev = e;
}
```

Computing the list entry of the task **(0.5 P)**, *correctly accessing both* `rq->head` *and* `rq->head.prev` **(0.5 P)**, *correctly setting the task's list entry pointers* **(1.0 P)**, *correctly setting the* `next` *pointer of the last entry and the* `prev` *pointer of the list head* **(1.0 P)**.

Vervollständigen Sie die Funktion `remove_first()`. Falls die Liste leer ist, gibt diese Funktion `NULL` zurück. Ansonsten entfernt die Funktion das erste Element aus der Liste und gibt einen Pointer auf den dazugehörigen Task zurück. **3 pt**

Complete the function `remove_first()`. *If the list is empty, the function returns* `NULL`. *Else, the function removes the first element from the list and returns a pointer to the corresponding task.*

Lösung:

```
struct task *remove_first(struct runqueue *rq) {
    if (rq->head.next == &rq->head) {
        return NULL;
    }
    struct task *t = from_queue_entry(rq->head.next);
    rq->head.next = rq->head.next->next;
    rq->head.next->prev = &rq->head;
    return t;
}
```

Checking whether the list is empty and returning `NULL` **(1.0 P)**, *retrieving the first element* **(0.5 P)**, *modifying the pointers of the list head and the new first element* **(1.0 P)**, *returning the task* **(0.5 P)**.

b) Vervollständigen Sie die Funktion `highest_prio_task()`, die basierend auf der Priorität den nächsten auszuführenden Task aus den Queues wählt. Der Task soll aus der Queue entfernt und zurückgegeben werden. Falls sämtliche Queues leer sind, wird NULL zurückgegeben. **3 pt**

Complete the function `highest_prio_task()` which, based on the priority, chooses the next task to be executed from the queues. The task shall be removed from the queue and be returned. If all queues are empty, NULL is returned.

Lösung:

```
struct task *highest_prio_task(void) {
    for (int i = NUM_PRIO - 1; i >= 0; i--) {
        struct task *t = remove_first(&rqs[i]);
        if (t != NULL) {
            return t;
        }
    }
    return NULL;
}
```

Looping through all runqueues **(1.0 P)** *in the correct order* **(0.5 P)**, *trying to remove the first task* **(1.0 P)**, *returning the task if successful* **(0.5 P)**.

c) Vervollständigen Sie die Funktion `current_task_blocked()`, die aufgerufen wird, wenn der aktuelle Task blockiert, d.h., wenn er seine Zeitscheibe nicht vollständig ausnutzt. Die Funktion soll einen anderen Task wählen und ihn in die Variable `current` schreiben. Die Funktion soll die Priorität erhöhen, wenn ein Task dreimal direkt hintereinander seine Zeitscheibe nicht vollständig ausnutzt. **3 pt**

Complete the function `current_task_blocked()` which is called when the current task blocks, i.e., wenn it does not completely use its timeslice. The function shall choose a different task and write it to the variable `current`. The function shall increase the priority if a task does not fully use its timeslice three times directly in sequence.

Lösung:

```
void current_task_blocked(void) {
    // After three partially used time slices, increase priority
    current->block_cnt++;
    if (current->block_cnt == 3) {
        if (current->prio < NUM_PRIO - 1) {
            current->prio++;
        }
        // Reset counter so that the priority is not increased too quickly
        // (optional, implied by the comment at the declaration of block_cnt)
        current->block_cnt = 0;
    }
    // We do not add the previous task to any queue - it is not runnable
    current = highest_prio_task();

    context_switch(current);
}
```

Count the number of incompletely used timeslices **(0.5 P)**, *update the priority* **(0.5 P)** *if the counter is 3* **(0.5 P)**, *limit the priority to `MAX_PRIO-1`* **(1.0 P)**, *select a different task* **(0.5 P)**. *Blocked (i.e., not runnable) tasks must not be appended to any queue.*

d) Vervollständigen Sie die Funktion `timer_interrupt()`, die einmal pro Timer-Periode vom Timer Interrupt aufgerufen wird. **5.5 pt**

- Legen Sie bei Ablauf der Zeitscheibe den aktuell laufenden Task in seine Queue und wählen Sie einen anderen Task, dessen Adresse Sie in die Variable `current` schreiben. Beachten Sie dabei die Regeln für Prioritätsänderungen und ändern Sie gegebenenfalls `block_cnt`.
- Füllen Sie wo notwendig die Zeitscheibe (`ts`-Feld) der Tasks wieder auf.
- Gehen Sie davon aus, dass `highest_prio_task()` nie NULL zurück gibt.

Complete the function `timer_interrupt()` which is called once per timer period by the timer interrupt.

- *Upon expiration of the time slice, place the currently executing task in its queue and choose a different task, whose address shall be placed in the variable `current`. Follow the rules for priority changes and modify `block_cnt` when necessary.*
- *Replenish the tasks' timeslice (`ts` field) where necessary.*
- *Assume that `highest_prio_task()` never returns NULL.*

Lösung:

```
void timer_interrupt(void) {
    // Deduct one unit of time from the task's remaining time
    current->ts--;

    // If the time slice is not elapsed, do nothing
    if (current->ts != 0) {
        return;
    }
    // Timeslice elapsed, reduce priority
    if (current->prio != 0) {
        current->prio--;
    }
    // Replenish the time slice - can also happen after highest_prio_task()
    current->ts = TS_LEN[current->prio];
    // Reset the counter of unused time slices
    current->block_cnt = 0;

    append(&rqs[current->prio], current);

    // Choose the next task
    current = highest_prio_task();

    context_switch(current);
}
```

doing nothing if the timeslice has not elapsed **(1.0 P)**, *decreasing the priority of the previous task* **(1.0 P)**, *replenishing the time slice* **(1.0 P)**, *resetting the previous task's "unused timeslices" counter* **(0.5 P)**, *returning the task to its queue* **(1.5 P)**, *choosing the next task* **(0.5 P)**.

Note that `append()` needs to be called before `highest_prio_task()`, as the same task might be scheduled again if no other is runnable.

Total: 20.0pt