Karlsruher Institut für Technologie

Betriebssysteme
(Operating Systems)

Prof. Dr. Frank Bellosa
Peter Maucher, M.Sc.
Daniel Habicht, M.Sc.
Yussuf Khalil, M.Sc.
Lukas Werling, M.Sc.

Nachname/ *Last name*	Vorname/ *First name*	Matrikelnr./ *Matriculation no*

Hauptklausur
17.03.2025

- Bitte tragen Sie zuerst auf dem Deckblatt Ihren Nachnamen, Ihren Vornamen und Ihre Matrikelnummer ein. Tragen Sie dann auf allen 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 23 Blättern: Einem Deckblatt, 20 Aufgabenblättern mit insgesamt 3 Aufgaben sowie 2 Seiten Man-Pages.

  The examination consists of 23 pages: One cover sheet, 20 sheets containing 3 assignments, and 2 sheets with man pages.

- Es sind keinerlei Hilfsmittel erlaubt!

  No additional material is allowed!

- Die Prüfung ist 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 also use the back side of the assignment 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 mehreren 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 according to the lecture.

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				

Aufgabe 1: Virtualisierung (20 P)

Assignment 1: Virtualization (20 P)

a) Beschreiben Sie, welche Schritte ablaufen, wenn ein Prozess freiwillig die CPU abgibt. Nehmen Sie an, dass ein weiterer Prozess lauffähig sei. **3 P**

Describe which steps are taken when a process voluntarily gives up the CPU. Assume that another process is runnable.

b) **Segmentation**

In dem folgenden Abschnitt schauen wir uns Segmentierung genauer an.

Wir nehmen dafür die folgenden Eigenschaften über unsere CPU an: Die CPU habe 16-bit-Adressen, von denen die höchstwertigsten *(most significant)* 4 bit in die Segmenttabelle indizieren. Die CPU habe außerdem zwei Register, das Segment Table Base Register (`STBR`) und das Segment Table Length Register (`STLR`), die über die Intrinsics `setSTBR(void*)` und `setSTLR(int)` gesetzt werden können.

Um die Übersetzung zu beschleunigen, sei außerdem ein Puffer verbaut, der einige Segmentbeschreibungen zwischenspeichert, sodass die CPU bei einem Speicherzugriff nicht jedesmal die Segmenttabelle lädt. Wir nennen diesen Puffer angelehnt an die Vorlesung ebenfalls Translation Lookaside Buffer (TLB). Dieser Buffer wird beim Setzen der Register `STBR` und `STLR` invalidiert, oder über das Intrinsic `flushTLB()`.

Nehmen Sie für die folgenden Implementierungen an, dass alle notwendigen Header bereits inkludiert sind.

In the following section, we will take a closer look at segmentation.

We assume the following properties of our CPU: The CPU shall have 16 bit addresses, of which the most significant 4 bit index into the segment table. Additionally, the CPU shall have two registers, the segment table base register (`STBR`) and the segment table length register (`STLR`), which can be set using the intrinsics `setSTBR(void*)` and `setSTLR(int)`.

To accelerate the translation, the CPU shall also have a buffer which caches some segment descriptions so that the CPU does not always load the segment table on a memory access. Following the lecture, we also call this buffer a translation lookaside buffer (TLB). This buffer is invalidated when setting the registers `STBR` or `STLR`, or via the intrinsic `flushTLB()`.

For the following implementations, assume that all required headers are already included.

Matrikelnummer/*Matriculation number*

1) Nennen und beschreiben Sie kurz ein anderes Übersetzungsverfahren. **2 P**

Name and briefly describe a different translation mechanism.

2) Diskutieren Sie, welche der Intrinsics `setSTBR(void*)`, `setSTLR(int)` sowie `flushTLB()` im User Space aufgerufen werden dürfen. **2 P**

Discuss which of the intrinsics `setSTBR(void*)`, `setSTLR(int)`, and `flushTLB()` may be called in user space.

3) Das Betriebssystem hat die folgende Segmenttabelle gesetzt: **2 P**

The operating system has set the following segment table:

`STLR`: 3

base	limit	prot
`0x0`	`0x0`	*none*
`0x0`	`0x500`	rw
`0x2000`	`0x600`	rw

Übersetzen Sie die folgenden virtuellen Adressen mit der gegebenen Tabelle.

Translate the following virtual addresses using the given table.

virt addr.	segment id	valid	phy addr.
`0x1234`			
`0x201`			
`0x2400`			

4) Die CPU gibt den Aufbau der Segmentbeschreibung vor, welche das Betriebssystem durch `struct segment` nutzbar macht. Außerdem verwaltet das Betriebssystem für jeden Prozess eine Segmenttabelle als `struct segment_table`, die es direkt an die CPU geben kann. **2 P**

The CPU specifies the layout of the segment descriptor that the operating system makes available as `struct segment`. Additionally, the operating system manages a segment table for each process in `struct segment_table` that it can pass directly to the CPU.

```
struct segment {
  void *base;
  int length;
  int prot;
};
struct segment_table {
  int len;
  segment segments[16];
};
```

Implementieren Sie die Funktion `insertSegmentTable(struct segment_table *s, void *base, int len, int prot)`, die ein neues Segment in die Segmenttabelle einfügt und das Segment aktiviert. Geben Sie `-1` zurück, wenn kein Platz mehr ist, ansonsten die Segmentnummer. Nehmen Sie an, dass alle sichtbaren Segmente auch genutzt werden.

Implement the function `insertSegmentTable(struct segment_table *s, void *base, int len, int prot)`, which inserts a new segment into the segment table and activates the segment. Return `-1` if no space is available, otherwise return the segment number. Assume that all visible segments are in use.

```
int insertSegmentTable(struct segment_table *s, void *base, int len, int prot) {

-------------------------------------------------------------------------

-------------------------------------------------------------------------

-------------------------------------------------------------------------

-------------------------------------------------------------------------

-------------------------------------------------------------------------

-------------------------------------------------------------------------

-------------------------------------------------------------------------

-------------------------------------------------------------------------

-------------------------------------------------------------------------

-------------------------------------------------------------------------

-------------------------------------------------------------------------

-------------------------------------------------------------------------
}
```

5) Implementieren Sie außerdem die Funktion `extendSegment(struct segment_table *s, int segment, int amount)`, die ein gegebenes Segment um eine gegebene Menge an Bytes (`amount`) verlängert. Stellen Sie sicher, dass die CPU die neue Länge auch benutzt. Geben Sie `-1` zurück, wenn es einen Fehler gab, ansonsten die neue Länge des Segments. Sie müssen die Parameter nicht überprüfen. Verifizieren Sie aber, dass das entstehende Segment valide ist. **2 P**

Furthermore, implement the function `extendSegment(struct segment_table *s, int segment, int amount)` which extends a given segment by a given number of bytes (`amount`). Ensure that the CPU actually uses the new length. Return `-1` if there was an error, otherwise return the new length of the segment. You do not need to check the parameters, but verify that the resulting segment is valid.

```
int extendSegment(struct segment_table *s, int segment, int amount) {

--------------------------------------------------------------------

--------------------------------------------------------------------

--------------------------------------------------------------------

--------------------------------------------------------------------

--------------------------------------------------------------------

--------------------------------------------------------------------

--------------------------------------------------------------------

--------------------------------------------------------------------
}
```

6) Implementieren Sie die Funktion `void *findPhysical(size_t size)`, die einen freien Bereich im physischen Speicher sucht. Das Betriebssystem verwaltet den physischen Speicher mittels einer einfach verketteten Liste aller freien kontinuierlichen Abschnitte, aus denen Sie mittels *first fit* einen passenden Bereich entfernen sollen. Der Pointer `begin` zeigt auf das erste Objekt der Liste, oder ist `NULL` wenn es keinen freien Speicher gibt. Am Ende der Liste gilt `next == NULL`. **5 P**

Wird der gesamte Abschnitt benötigt, so sollen Sie diesen auch aus der Liste entfernen, das dazugehörige Listenelement aber nicht freigeben. Ansonsten verkleinern Sie den Abschnitt. Geben Sie im Erfolgsfall einen physischen Pointer auf den Beginn des Abschnitt zurück und passen Sie die Liste an, ansonsten geben Sie `NULL` zurück.

Hinweis: Hier müssen Sie nicht auf Alignment achten.

Implement the function `void *findPhysical(size_t size)` which searches a free area in physical memory. The operating system manages physical memory using a singly linked list of all free contiguous spaces, from which you shall remove an appropriate area using first fit*. The pointer `begin` points to the first object in the list, or `NULL` if there is no free space. At the end of the list, `next == NULL` holds true.*

If the whole space is required, also remove it from the list without freeing the associated list element. Otherwise, shrink the memory area. In case of success, return a physical pointer to the start of the space and alter the list, otherwise return `NULL`.

Hint: *You do not need to consider alignment.*

```
struct list {
  size_t len; // length of this segment in the list
  struct list *next; // next list element or NULL if last element
  void *storage_pointer; // physical pointer to the memory area
};
typedef struct list list;
static struct list *begin;

void *findPhysical(size_t size) {
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
----------------------------------------------------------------------------------
}
```

7) Wie können Sie den Speicherverbrauch der in Teilaufgabe 6) genutzten Liste reduzieren? **1 P**

How can you reduce the memory usage of the list used in subassignment 6)?

8) Nehmen Sie an, dass Ihre CPU (nach dem Start) nur virtuelle Adressen unterstützt. Wie kann der Kernel sicherstellen, dass er in jedem Prozess für seinen eigenen Addressraum die gleichen Adressen nutzen kann? **1 P**

Assume that your CPU (after startup) only supports virtual addresses. How can the kernel assure that it can use the same addresses for its own address space in each process?

Total: 20 P

Aufgabe 2: Nebenläufigkeit (20 P)

Assignment 2: Concurrency (20 P)

a) In der Vorlesung haben Sie Spinlocks als ein Synchronisationsprimitiv kennengelernt.

In the lecture, you learned about spinlocks as a synchronization primitive.

1) Implementieren Sie `spin_acquire()` und `spin_release()`, um ein Spinlock gemäß der Vorlesung umzusetzen. Nutzen Sie keine Funktionen aus der `pthread`-Bibliothek. Sie dürfen eine Funktion **`int`** *`test_and_set`*`(`**`int`** `*`*`ptr`*`,` **`int`** *`new`*`)` verwenden, die atomar den Wert von `*ptr` auf `new` setzt und den alten Wert zurückgibt. **2 P**

Implement `spin_acquire()` and `spin_release()` to implement a spinlock according to the lecture. Do not use any functions from the `pthread` library. However, you may use a function **`int`** `test_and_set(`**`int`** `*ptr,` **`int`** `new)`*, which atomically sets the value of `*ptr` to `new` and returns the old value.*

```
int test_and_set(int *ptr, int new);
void spin_acquire(int *lock) {

--------------------------------------------------------------------------

--------------------------------------------------------------------------

--------------------------------------------------------------------------

--------------------------------------------------------------------------
}
void spin_release(int *lock) {

--------------------------------------------------------------------------

--------------------------------------------------------------------------

--------------------------------------------------------------------------

--------------------------------------------------------------------------
}
```

2) Erläutern Sie, welches Problem bei der Verwendung von Spinlocks in Verbindung mit statischen Prioritäten auftritt. Was muss der Scheduler beachten, um dieses Problem zu vermeiden? **2 P**

Explain what problem occurs when using spinlocks in conjunction with static priorities. What does the scheduler need to consider to avoid this problem?

3) Nehmen Sie eine einfache Spinlock-Implementation an, welche atomare Schreiboperationen verwendet und zwischen zwei Zuständen unterscheidet. Diese Implementierung erfüllt nicht alle Anforderungen an eine korrekte Lösung des Synchronisationsproblems. Nennen Sie beide Anforderungen, welche ein solcher Spinlock verletzt, und erklären Sie, warum das der Fall ist. **3 P**

Assume a simple spinlock implementation that uses atomic store operations and distinguishes between two states. This implementation does not meet all requirements to a correct solution to the synchronization problem. Name both requirements that this spinlock violates and explain why this is the case.

b) In dieser Aufgabe lösen Sie das Producer-Consumer-Problem mittels *pthread condition variables*. Die Funktionen `producer()` und `consumer()`, welche jeweils von mehreren Threads gleichzeitig ausgeführt werden, nutzen das Array `work_buf` zum Austauschen von Arbeit. Vervollständigen Sie den gegebenen Code-Abschnitt für eine korrekte Lösung des Producer-Consumer-Problems. Inkludieren Sie alle notwendigen Header. Nutzen Sie bei Bedarf die Funktion `init()` für die dynamische Initialisierung weiterer globaler Variablen, welche ihre Lösung benötigt. Sie müssen weder dynamisch initialisierte Ressourcen freigeben noch Fehlerbehandlung implementieren. **4.5 P**

In this exercise, you solve the producer-consumer problem using pthread condition variables. The `producer()` and `consumer()` functions are each executed by multiple threads and use the `work_buf` array for exchanging work. Complete the given code template for a correct solution to the producer-consumer problem. Include all headers required. If needed, use the function `init()` for the dynamic initialization of additional global variables required by your solution. You neither have to free dynamically initialized resources nor implement error handling.

```
// space for headers

----------------------------------------------------------------------

----------------------------------------------------------------------

----------------------------------------------------------------------

----------------------------------------------------------------------
```

```
#define NUM_WORK 10
int produce_work(void); void consume_work(int work_desc);
int work_buf[NUM_WORK] = {0};
int work_len = 0;
// definitions of global variables

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
// dynamic initialization
void init() {

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
}
void producer() {
  while (1) {
    int work_desc = produce_work();

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
    work_buf[work_len++] = work_desc;

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
  }
}
```

```
void consumer() {
  while(1) {

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    int work_desc = work_buf[--work_len];

--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
    consume_work(work_desc);
  }
}
```

c) In der Vorlesung wurden drei verschiedene Threadmodelle behandelt. Nennen und beschreiben Sie diese **kurz**. **3 P**

*The lecture covered three different thread models. Name and **briefly** describe them.*

d) Nennen Sie (ohne Erklärung) alle vier Eigenschaften, die zur Lösung des Problems kritischer Abschnitte nötig sind. **2 P**

Name (without explanation) all four properties required to solve the critical section problem.

e) Gegeben sei folgender Systemzustand:
- Prozesse A, B und C
- Ressourcen x (2×), y (1×), z (3×)
- A hält x, B hält x und z, C hält y und z
- B möchte y, C möchte x

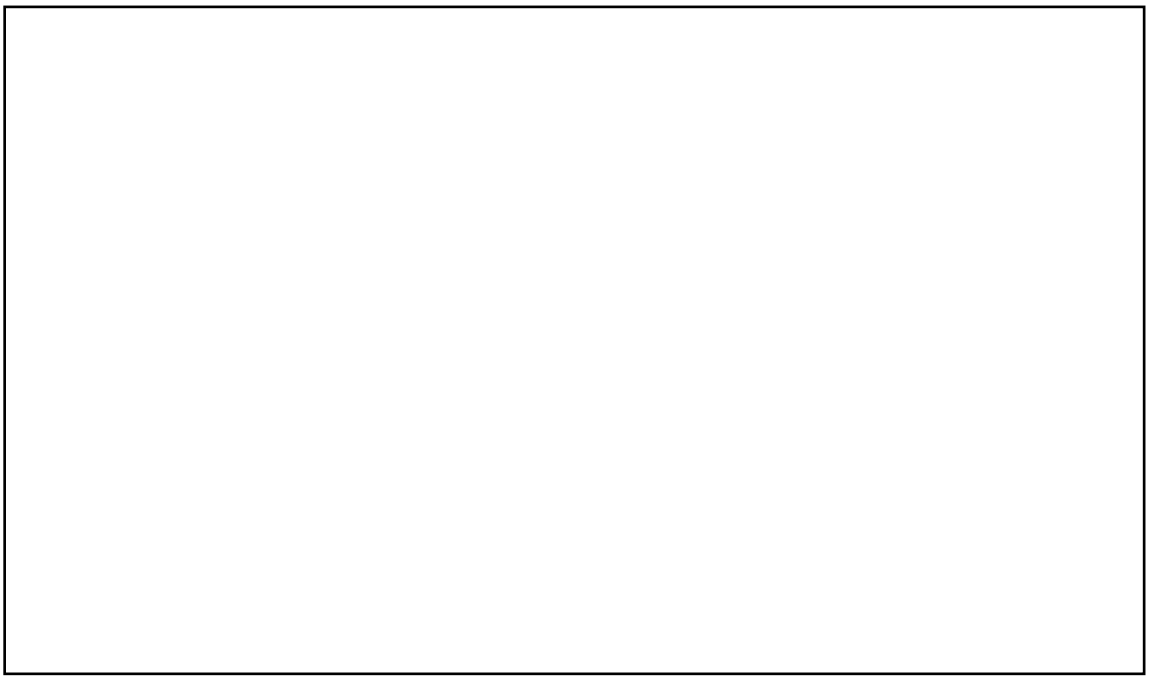
Given the following system state:
- *Processes A, B, and C*
- *Resources x (2×), y (1×), z (3×)*
- *A holds x, B holds x and z, C holds y and z*
- *B wants y, C wants x*

1) Zeichnen Sie den zugehörigen *Resource Allocation Graph* (RAG). **1.5 P**

Draw the corresponding Resource Allocation Graph *(RAG).*

2) Zeichnen Sie den zugehörigen *Wait-for Graph* (WFG). **1.5 P**

Draw the corresponding Wait-for Graph *(WFG).*



3) Liegt ein Deadlock vor? **0.5 P**

Is there a deadlock?

☐ Ja / *Yes* ☐ Nein / *No*

Total: 20 P

Aufgabe 3: Persistenz (20 P)

Assignment 3: Persistence (20 P)

a) Stellen Sie sich ein System vor, welches eine Netzwerkkarte enthält, die für jedes erhaltene Netzwerkpaket einen Interrupt auslöst. Unter hoher Netzwerklast wendet das Betriebssysteme den Großteil der CPU-Zeit für die Behandlung von Interrupts auf, was Anwendungen davon abhält, Fortschritt zu machen. **3.5 P**

Nennen Sie den Begriff, welchen Sie in der Vorlesung für eine solche Situation, in der kein Deadlock vorliegt aber Anwendungen trotzdem keinen Fortschritt machen, kennengelernt haben. Beschreiben Sie zudem zwei Strategien aus der Vorlesung, welche bei der Abmilderung einer Überflutung mit Interrupts helfen können.

Imagine a system containing a network interface card that raises an interrupt for each network packet received. Under a heavy networking load, the operating system spends a large amount of the CPU time on handling interrupts, which prevents applications from making progress.

Give the name that you have learned in the lecture for a situation where there is no deadlock but applications are blocked from making progress regardless. Further, describe two strategies from the lecture that help with mitigating a flood of interrupts.

Problem:

1:

2:

b) Die CPU größere Datenmengen zwischen einem I/O-Gerät und dem Hauptspeicher kopieren zu lassen, nimmt eine signifikante Menge an CPU-Zeit ein. In der Vorlesung wurde ein Konzept vorgestellt, welches einen großen Teil dieser Last von der CPU nehmen kann. Benennen Sie dieses Konzept und beschreiben Sie den Ablauf eines Kopiervorgangs. **1.5 P**

Letting the CPU copy large amounts of data between I/O devices and main memory takes up a significant amount of CPU time. The lecture introduced a concept that can take a large part of this load off the CPU. Give the name of this concept and describe the sequence of a copy process.

Konzept/*Concept*: _____

Kopiervorgang/*Copy Process*: _____

c) Im Folgenden betrachten Sie ein Dateisystem mit *indexed allocation*. Jeder *inode* hat 128 direkte Blockzeiger, 32 einfach indirekte Blockzeiger und 1 doppelt indirekten Blockzeiger. Nehmen Sie an, dass jeder Dateisystemblock 8 KiB groß ist und jeder Blockzeiger 8 B benötigt. **2.5 P**

Berechnen Sie zuerst die Anzahl an Blöcken pro einfach indirektem Blockzeiger und pro doppelt indirektem Blockzeiger. Berechnen Sie im Anschluss die maximale Dateigröße und geben Sie das Ergebnis vollständig gekürzt in MiB an.

In the following, you will have a look at a file system with indexed allocation*. Each* inode *has* 128 *direct block pointers,* 32 *single indirect block pointers, and* 1 *double indirect block pointer. Assume that each file system block is* 8 *KiB in size and each block pointer requires* 8 *B.*

First, calculate the number of blocks per single indirect block pointer and per double indirect block pointer. After that, calculate the maximum file size reduced to a single number of MiB.

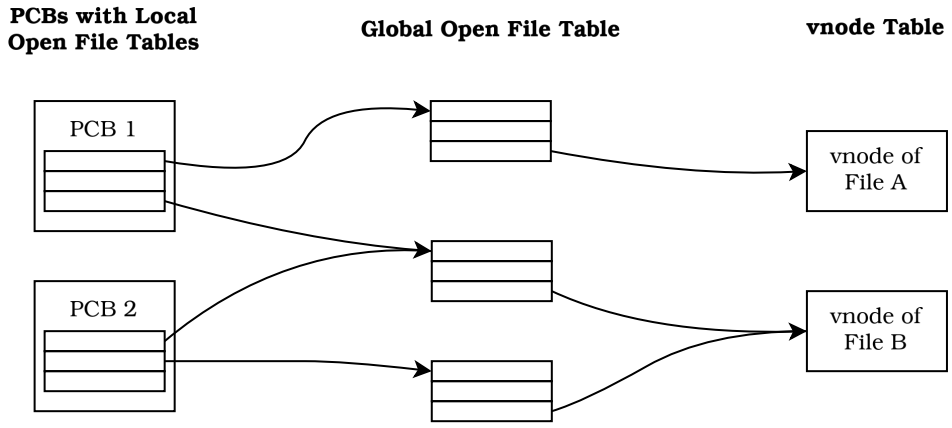
einfach indirekt/*single indirect*: _____

doppelt indirekt/*double indirect*: _____

Dateigröße/*file size*: _____

d) Die folgende Grafik beschreibt die *local open file tables* und die *global open file table* für zwei Prozesse mit offenen Dateien.

The following figure describes the local open file tables *and the* global open file table *for two processes with open files.*



1) Was ist die Beziehung zwischen Dateideskriptoren und der *local open file table*? **0.5 P**

What is the relationship between file descriptors and the local open file table*?*

2) Nennen Sie zwei Attribute, welche zusätzlich zur Referenz auf einen *vnode* für jeden Eintrag in der *global open file table* gespeichert werden. **1 P**

Give two attributes that, in addition to the reference on the vnode*, are stored for each entry in the* global open file table*.*

1:

2:

3) In der obigen Abbildung können Sie sehen, dass zwei unterschiedliche Einträge in den *local open file tables* auf den selben Eintrag in der *global open file table* zeigen. Erklären Sie, wie eine solche Situation enstehen kann. **0.5 P**

In the figure above, you can see that two different entries in local open file tables *point to the same entry in the* global open file table*. Explain how such a situation can occur.*

4) Weiter können Sie in der Abbildung sehen, dass zwei unterschiedliche Einträge in der *global open file table* auf einen *vnode* zeigen. Erklären Sie auch hier, wie eine solche Situation entstehen kann. **0.5 P**

In addition, you can see in the figure that two different entries in the global open file table *point to a single* vnode*. Again, explain how such a situation can occur.*

e) Der Page-Cache (in der Vorlesung auch als Buffer-Cache bezeichnet) ist eine zentrale Komponente im I/O-Stack vieler Betriebssysteme.

The page cache (also referred to as buffer cache in the lecture) is a central component in the I/O stack of many operating systems.

1) Erklären Sie die Motivation hinter der Verwendung dieses Caches im Kontext von I/O. **1 P**

Explain the motivation behind using this cache in the context of I/O.

2) Nennen Sie zwei fundamental unterschiedliche Ereignisse, die zum Zurückschreiben von Inhalten des Page-Caches führen können. **1 P**

Give two fundamentally different events that can cause the write-back of contents in the page cache.

1:

2:

3) Wann können im Page-Cache gepufferte Inhalte beim Zurückschreiben übersprungen werden? Erklären Sie, warum das der Fall ist. **1 P**

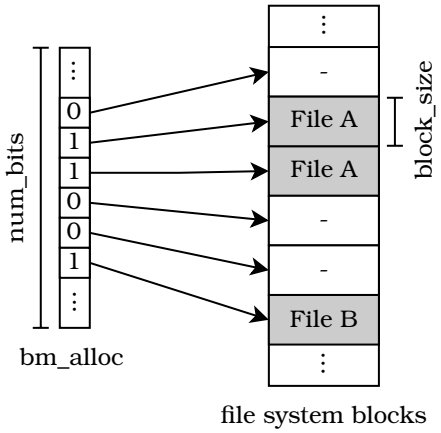
When can contents buffered in the page cache be skipped during write-back? Explain why this is the case.

f) In dieser Aufgabe geht es um die Implementation eines Dateisystems mit *contiguous allocation*, welches eine einfache Bitmap für die Verwaltung von Blockallokationen verwendet. Im Folgenden bezeichnet $B_{\text{bm}}[i] \in \{0, 1\}$ das i-te Bit der Bitmap `bm`. Der i -te Block ist genau dann alloziert, wenn $B_{\text{fs.bm\_alloc}}[i] = 1$ gilt. Sie dürfen annehmen, dass alle verwendeten Datentypen über ausreichende Genauigkeit verfügen und dass alle notwendigen Header bereits inkludiert sind.

This exercise is about implementing a file system with contiguous allocation *that uses a simple bitmap for managing block allocations. In the following,* $B_{\text{bm}}[i] \in \{0, 1\}$ *describes the* i*-th bit of the bitmap* `bm`*. The* i*-th block is allocated if and only if* $B_{\text{fs.bm\_alloc}}[i] = 1$*. You may assume that all data types have enough precision and that all headers required are already included.*

```
typedef struct bitmap {
  long num_bits;
  uint64_t map[];
} bitmap_t;

typedef struct fs {
  bitmap_t *bm_alloc;
  int block_size; // in bytes
  // FAT and other fs metadata ...
} fs_t;
```



Listing 1: Metadaten-Struktur & Bitmap / *Metadata structure & bitmap*

1) Implementieren Sie als erstes die Funktion `bm_find_next(bitmap_t *bm, long *cur, bool bit)`, welche den kleinsten Index $i \geq$ `*cur` sucht, für den $B_{\text{bm}}[i] =$ `bit` gilt. Falls solch ein Index i existiert, sollen Sie `true` zurückgeben und `*cur` auf i setzen. Anderenfalls sollen Sie `false` zurückgeben. Nutzen Sie die Funktion `bm_test(bitmap_t *bm, long n)` um $B_{\text{bm}}[\text{n}]$, also das Bit am Index `n`, zu bestimmen. **2 P**

First, implement the function `bm_find_next(bitmap_t *bm, long *cur, bool bit)` *that searches the smallest index* $i \geq$ `*cur` *such that* $B_{\text{bm}}[i] =$ `bit`*. If such an index* i *exists, return* `true` *and set* `*cur` *to* i*. Otherwise, return* `false`*. Use the function* `bm_test(bitmap_t *bm, long n)` *to determine* $B_{\text{bm}}[\text{n}]$*, namely the bit at index* `n`*.*

```
bool bm_test(bitmap_t *bm, long n); // test if n-th bit is set in bm

bool bm_find_next(bitmap_t *bm, long *cur, bool bit) {

--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
}
```

2) Im nächsten Schritt sollen Sie die Funktion `bm_find_range(bitmap_t *bm, long *cur, long len, bool bit)` implementieren, welche den kleinsten Index $i \geq$ `*cur` bestimmt, sodass $B_{\mathrm{bm}}[i] = B_{\mathrm{bm}}[i+1] = ... = B_{\mathrm{bm}}[i+\mathrm{len}-1] =$ `bit` gilt. Verwenden Sie hierfür die in 1) implementierte `bm_find_next()`. Geben Sie auch hier beim Finden eines passenden Indexes i `true` zurück und setzen Sie `*cur` auf i. Anderenfalls soll `false` zurückgegeben werden. **2.5 P**

Hinweis: Sie können mit `bm_find_next()` sowohl den Beginn als auch das Ende (nächstes Bit mit anderem Wert) des gesuchten Bereichs finden.

For the next step, implement the function `bm_find_range(bitmap_t *bm, long *cur, long len, bool bit)` that determines the smallest index $i \geq$ `*cur`, such that $B_{\mathrm{bm}}[i] = B_{\mathrm{bm}}[i+1] = ... = B_{\mathrm{bm}}[i+\mathrm{len}-1] =$ `bit`. For this, use `bm_find_next()` implemented in 1). Like before, return `true` when an appropriate index i is found and set `*cur` to i. Otherwise, return `false`.

Hint: *Using `bm_find_next()`, you can find the start as well as the end (next bit with different value) of the target range.*

```
bool bm_test(bitmap_t *bm, long n); // test if n-th bit is set in bm
bool bm_find_next(bitmap_t *bm, long *cur, bool bit);

bool bm_find_range(bitmap_t *bm, long *cur, long len, bool bit) {

--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
--------------------------------------------------------------------------
}
```

3) Implementieren Sie abschließend die Funktion `fs_alloc_file(fs_t *fs, long file_sz)`, welche mittels *first fit* eine Allokation vornimmt. Bestimmen Sie dafür mittels `fs.bm_alloc` den ersten ausreichend großen Speicherbereich und nutzen Sie davon so wenig wie nötig. Beachten Sie, dass `file_sz` in Bytes und nicht in Blöcken angegeben ist. Gibt es einen solchen Bereich, allozieren Sie ihn durch das Setzen der entsprechenden Bits in `fs.bm_alloc` und geben Sie den Index des ersten Blocks dieses Bereichs zurück. Anderenfalls soll -1 zurückgegeben werden. Verwenden Sie die Funktion `bm_set_range(bitmap_t *bm, long n, long len)` für das Setzen eines Bit-Intervalls $[n, n + \text{len} - 1]$ in einer Bitmap `bm`. **2.5 P**

Finally, implement the function `fs_alloc_file(fs_t *fs, long file_sz)` that makes an allocation using first fit*. For this, determine the first sufficiently large memory area using `fs.bm_alloc` and use as little of it as possible. Note that `file_sz` is given as number of bytes, not blocks. If such a range exists, allocate it by setting the associated bits in `fs.bm_alloc` and return the index of the first block of this range. Otherwise, return* -1*. Use the function `bm_set_range(bitmap_t *bm, long n, long len)` for setting the bit interval* $[n, n + \text{len} - 1]$ *in a bitmap `bm`.*

```
bool bm_test(bitmap_t *bm, long n); // test if n-th bit is set in bm
bool bm_find_next(bitmap_t *bm, long *cur, bool bit);
bool bm_find_range(bitmap_t *bm, long *cur, long len, bool bit);
void bm_set_range(bitmap_t *bm, long n, long len);

long fs_alloc_file(fs_t *fs, long file_sz) {

-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
-------------------------------------------------------------------------
}
```

Total: 20 P

NAME

pthread_cond_init, pthread_cond_signal, pthread_cond_broadcast, pthread_cond_wait, pthread_cond_destroy – operations on conditions

SYNOPSIS

```
#include <pthread.h>

pthread_cond_t cond = PTHREAD_COND_INITIALIZER;

int pthread_cond_init(pthread_cond_t *cond,
                      pthread_condattr_t *cond_attr);
int pthread_cond_signal(pthread_cond_t *cond);
int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);
int pthread_cond_destroy(pthread_cond_t *cond);
```

DESCRIPTION

A condition (short for "condition variable") is a synchronization device that allows threads to suspend execution and relinquish the processors until some predicate on shared data is satisfied. The basic operations on conditions are: signal the condition (when the predicate becomes true), and wait for the condition, suspending the thread execution until another thread signals the condition.

A condition variable must always be associated with a mutex, to avoid the race condition where a thread prepares to wait on a condition variable and another thread signals the condition just before the first thread actually waits on it.

pthread_cond_init initializes the condition variable *cond*, using the condition attributes specified in *cond_attr*, or default attributes if *cond_attr* is **NULL**. The LinuxThreads implementation supports no attributes for conditions, hence the *cond_attr* parameter is actually ignored.

Variables of type **pthread_cond_t** can also be initialized statically, using the constant **PTHREAD_COND_INITIALIZER**.

pthread_cond_signal restarts one of the threads that are waiting on the condition variable *cond*. If no threads are waiting on *cond*, nothing happens. If several threads are waiting on *cond*, exactly one is restarted, but it is not specified which.

pthread_cond_broadcast restarts all the threads that are waiting on the condition variable *cond*. Nothing happens if no threads are waiting on *cond*.

pthread_cond_wait atomically unlocks the *mutex* (as per **pthread_unlock_mutex**) and waits for the condition variable *cond* to be signaled. The thread execution is suspended and does not consume any CPU time until the condition variable is signaled. The *mutex* must be locked by the calling thread on entrance to **pthread_cond_wait**. Before returning to the calling thread, **pthread_cond_wait** re-acquires *mutex* (as per **pthread_mutex_lock**).

Unlocking the mutex and suspending on the condition variable is done atomically. Thus, if all threads always acquire the mutex before signaling the condition, this guarantees that the condition cannot be signaled (and thus ignored) between the time a thread locks the mutex and the time it waits on the condition variable.

pthread_cond_destroy destroys a condition variable, freeing the resources it might hold. No threads must be waiting on the condition variable on entrance to **pthread_cond_destroy**. In the LinuxThreads implementation, no resources are associated with condition variables, thus **pthread_cond_destroy** actually does nothing except checking that the condition has no waiting threads.

CANCELLATION

pthread_cond_wait and **pthread_cond_timedwait** are cancelation points. If a thread is cancelled while suspended in one of these functions, the thread immediately resumes execution, then locks again the *mutex* argument to **pthread_cond_wait** and **pthread_cond_timedwait**, and finally executes the cancelation. Consequently, cleanup handlers are assured that *mutex* is locked when they are called.

RETURN VALUE

All condition variable functions return 0 on success and a non-zero error code on error.

SEE ALSO

pthread_condattr_init(3), **pthread_mutex_lock**(3), **pthread_mutex_unlock**(3)

NAME

pthread_condattr_init, pthread_condattr_destroy – condition creation attributes

SYNOPSIS

```
#include <pthread.h>

int pthread_condattr_init(pthread_condattr_t *attr);
int pthread_condattr_destroy(pthread_condattr_t *attr);
```

DESCRIPTION

Condition attributes can be specified at condition creation time, by passing a condition attribute object as second argument to **pthread_cond_init**(3). Passing **NULL** is equivalent to passing a condition attribute object with all attributes set to their default values.

The LinuxThreads implementation supports no attributes for conditions. The functions on condition attributes are included only for compliance with the POSIX standard.

pthread_condattr_init initializes the condition attribute object *attr* and fills it with default values for the attributes. **pthread_condattr_destroy** destroys a condition attribute object, which must not be reused until it is reinitialized. Both functions do nothing in the LinuxThreads implementation.

RETURN VALUE

pthread_condattr_init and **pthread_condattr_destroy** always return 0.

SEE ALSO

pthread_cond_init(3).

NAME

pthread_mutex_init, pthread_mutex_lock, pthread_mutex_unlock, pthread_mutex_destroy − operations on mutexes

SYNOPSIS

```
#include <pthread.h>

pthread_mutex_t fastmutex = PTHREAD_MUTEX_INITIALIZER;

int pthread_mutex_init(pthread_mutex_t *mutex,
                       const pthread_mutexattr_t *mutexattr);
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

DESCRIPTION

A mutex is a MUTual EXclusion device, and is useful for protecting shared data structures from concurrent modifications, and implementing critical sections and monitors.

A mutex has two possible states: unlocked (not owned by any thread), and locked (owned by one thread). A mutex can never be owned by two different threads simultaneously. A thread attempting to lock a mutex that is already locked by another thread is suspended until the owning thread unlocks the mutex first.

pthread_mutex_init initializes the mutex object pointed to by *mutex* according to the mutex attributes specified in *mutexattr*. If *mutexattr* is **NULL**, default attributes are used instead.

Variables of type **pthread_mutex_t** can also be initialized statically, using the constant **PTHREAD_MUTEX_INITIALIZER**.

pthread_mutex_lock locks the given mutex. If the mutex is currently unlocked, it becomes locked and owned by the calling thread, and **pthread_mutex_lock** returns immediately. If the mutex is already locked by another thread, **pthread_mutex_lock** suspends the calling thread until the mutex is unlocked. If the mutex is already locked by the calling thread, the calling thread is suspended until the mutex is unlocked, thus effectively causing the calling thread to deadlock.

pthread_mutex_unlock unlocks the given mutex. The mutex is assumed to be locked and owned by the calling thread on entrance to **pthread_mutex_unlock**. **pthread_mutex_unlock** always returns it to the unlocked state.

pthread_mutex_destroy destroys a mutex object, freeing the resources it might hold. The mutex must be unlocked on entrance. In the LinuxThreads implementation, no resources are associated with mutex objects, thus **pthread_mutex_destroy** actually does nothing except checking that the mutex is unlocked.

CANCELLATION

None of the mutex functions is a cancelation point, not even **pthread_mutex_lock**, in spite of the fact that it can suspend a thread for arbitrary durations. This way, the status of mutexes at cancelation points is predictable, allowing cancelation handlers to unlock precisely those mutexes that need to be unlocked before the thread stops executing. Consequently, threads using deferred cancelation should never hold a mutex for extended periods of time.

RETURN VALUE

pthread_mutex_init always returns 0. The other mutex functions return 0 on success and a non-zero error code on error.

SEE ALSO

pthread_mutexattr_init(3)

NAME

pthread_mutexattr_init, pthread_mutexattr_destroy − initialize and destroy a mutex attributes object

LIBRARY

POSIX threads library (*libpthread*, *−lpthread*)

SYNOPSIS

```
#include <pthread.h>

int pthread_mutexattr_init(pthread_mutexattr_t *attr);
int pthread_mutexattr_destroy(pthread_mutexattr_t *attr);
```

DESCRIPTION

The **pthread_mutexattr_init**() function initializes the mutex attributes object pointed to by *attr* with default values for all attributes defined by the implementation.

The results of initializing an already initialized mutex attributes object are undefined.

The **pthread_mutexattr_destroy**() function destroys a mutex attribute object (making it uninitialized). Once a mutex attributes object has been destroyed, it can be reinitialized with **pthread_mutexattr_init**().

The results of destroying an uninitialized mutex attributes object are undefined.

RETURN VALUE

On success, these functions return 0. On error, they return a positive error number.

STANDARDS

POSIX.1-2008.

HISTORY

POSIX.1-2001.

NOTES

Subsequent changes to a mutex attributes object do not affect mutex that have already been initialized using that object.

SEE ALSO

pthread_mutex_init(3)