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.

# Musterlösung Hauptklausur 17.03.2025

## Alle Punkteangaben ohne Gewähr!

- 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 24 Blättern: Einem Deckblatt, 21 Aufgabenblättern mit insgesamt 3 Aufgaben sowie 2 Seiten Man-Pages.

  *The examination consists of 24 pages: One cover sheet, 21 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.*

**Solution:**

- *Jump into the OS* ***(0.5 P)*** *using* `yield` *(*`sched_yield` *in posix)* ***(0.5 P)****, which now also sets the CPU mode to kernel mode* ***(0.5 P)***
- *Store old register information (including program counter,* `pc`*)* ***(0.5 P)*** *into PCB* ***(0.5 P)*** *so that the OS can use the whole CPU*
- *Select a new process (might be same as old process)* ***(0.5 P)*** *from the ready queue* ***(0.5 P)*** *using the OS scheduler* ***(0.5 P)***
- *Insert old process into the ready queue* ***(0.5 P)***
- *Set the new address space* ***(0.5 P)*** *(This is only a register operation, no need to store or even swap out the old address space)*
- *Load register values of new process* ***(0.5 P)*** *from the PCB*
- *Return the CPU into user mode* ***(0.5 P)***
- *Jump to the stored* `pc` ***(0.5 P)***

**Also accepted:**

- *An explanation of what happens on an* `exit()` *call*
- *A general discussion of system call handling in an OS*

**Common mistakes:**

- *The address space is (usually) already in main memory, only the CPU needs to be reconfigured. If the new AS is not im memory, page faults will occur during execution, so no work is required now either way.*
- *With a blocking system call (like* `read()` *the process is not intending to give up the CPU, but would rather continue as fast as possible. In many cases, especially with cached results, real kernels like Linux usually return the control back to the calling process for performance reasons, and only reschedule if the results needs actual waiting.*

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.*

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

*Name and briefly describe a different translation mechanism.*

**Solution:**

- *Paging* ***(0.5 P)***
  - *Separate the address into index (single level) or indices (multi-level) and offset* ***(0.5 P)***
    - *Size of offset usually 12 bit* ***(0.5 P)***
  - *Single level: find base frame by indexing into an array pointed to by a base address* ***(0.5 P)*** *(called the `CR3` register in x86)*
  - *Multi level: find base frame by recursively indexing* ***(0.5 P)***
  - *Find the physical address by concatenating offset to base frame address* ***(0.5 P)***
- *Base+Limit* ***(0.5 P)***
  - *No translation, only protection* ***(0.5 P)***
  - *Two registers,* base *and* limit ***(0.5 P)***
  - *Address valid iff* $\text{base} \leq \text{address} < \text{base} + \text{limit}$ ***(0.5 P)***

***Also accepted****:*

- *Inverted page table for translating physical to virtual (as an accelerator for lookup) is accepted, but somewhat misses that segmentation usually maps the other way round (i.e., from virtual to physical)*

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.*

**Solution:**

- `setSTBR(void*)` *and* `setSTLR(int)` *need to be kernel-only,* ***(0.5 P)*** *as otherwise the process could alter its own segment table and thus gain full access to the whole memory.* ***(0.5 P)***
- `flushTLB()` *on the other hand can be called from user space,* ***(0.5 P)*** *as the only effect would be that the process slows down itself as it forces the CPU (or OS) to re-insert all mappings.* ***(0.5 P)***

**Alternative solution for `flushTLB()`, only accepted with explanation:**

- *Processes should not be allowed to slow the system down, therefore `flushTLB()` should be limited to kernel space* ***(1 P)***

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.*

**Solution:**

| *virt addr.* | *segment id* | *valid* | *phy addr.* |
|---|---|---|---|
| *0x1234* | *1* | *yes* | *0x234* |
| *0x201* | *0* | *no* | |
| *0x2400* | *2* | *yes* | *0x2400* |

- ***(1.5 P)** if 2 rows out of 3 correct*
- ***(1 P)** if 1 row out of 3 correct*
- ***(0.5 P)** when one column is correct, but no row*

**Common mistakes:**

- *Translation table numbered by segment table: It is not valid to look up the $n$-th row of the translation table in the $n$-th row of the segment table, the translation table would work in any colum order.*
- *$0x201_{16}$ ($513_{10}$) is equal to $0x0201_{16}$, not equal to $0x2010_{16}$ ($8208_{10}$)*
- *For virtual address $0x201_{virt}$ = $0x0201_{virt}$, no valid translation can be found, so the field* "phy addr." *should be empty. However, if one does the translation and ignores both the access violation (no accesses allowed) and the limit violation ($0x201_{offset} > 0x0_{limit}$), the resulting physical address would also be $0x201_{phy}$, so this one number is also accepted.*

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.*

**Solution:**

```
int insertSegmentTable(struct segment_table *s, void *base, int len, int prot) {
  if (s->len >= 16) return -1; // .5P
  s->segments[s->len] = (struct segment) {base, len, prot}; // .5P
  setSTLR(s->len + 1); // .5P
  s->len++;          // .5P
  return s->len - 1; //
}
```

**Note:**

- *Accepted both `s->len` and `s->len - 1` as the return value as the segment number might be 1-indexed ("first segment")*

**Common mistakes:**

- *Forgot return of segment number*
- *Forgot `setSTLR(int)`.* Note: *`flushTLB(int)` does not suffice here as the hardware would need to .*
- *Used `malloc()` (which is not easily available in the kernel). However, if there was a matching `free()` on every path, no points were deducted.*

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.*

**Solution:**

```
int extendSegment(struct segment_table *s, int segm, int amount) {
  int amount = s->segments[segm].len + amount; // .5P
  if (amount < 0 || amount > 0xFFF) return -1;     // .5P
  s->segments[segm].len = amount; // .5P
  flushTLB(); // .5P
  return amount; // required for full points
}
```

**Common mistakes:**

- *Overlapping segments are ok (and indeed sometimes useful, e.g. for file system caches), but segments cannot be longer than the offset of a pointer*

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;
```

**Solution:**

```
void *findPhysical(size_t size) {
  struct list *curr = begin; //.5P
  struct list *last = curr;
  while (curr != NULL) { //.5P
    if (curr->length > size) { //.5P
      curr->length -= size;                          //.5P
      return ((char *)curr->storage_pointer) + length; //
    } else if (curr->length == size) { //.5P
      if (curr == begin) {  //.5P
        begin = curr->next;      //.5P
      } else {                   //
        last->next = curr->next; //
      }
      return curr->storage_pointer; //.5P
    }
    last = curr;
    curr = curr->next; //.5P
  }
  return NULL; //.5P
}
```

**Common mistakes:**

- *No valid handling of either first or last element of list (e.g., skipping checking the element pointed to by* `begin`*)*
- *Forgetting to update pointer, length or both for the "segment larger than requested size" case*
- *Returning a memory area that is also still valid in the allocator*
- `free()`*-ing the list element*

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)?*

**Solution:**

- *Free-list* ***(0.5 P)***
  - *Idea: locate list directly in memory to be allocated and generate returned pointer from directly from the list element location.* ***(0.5 P)***
- *Use dynamic array* ***(0.5 P)***
  - *No next pointer required* ***(0.5 P)***
- `free()`*-ing the list element that was skipped before* ***(0.5 P)***

**Common mistakes:**

- *Alignment/padding does not work (*`size_t` *and pointers have the same size)*
- *Bitmap allocator: given that we need a bit for every fixed size block, a bitmap allocator will be worse than any kind of free list in the case of one contiguous, empty segment. (The bitmap allocator is in* $O(\text{blocks})$ *which scales linearily with the number bytes of physical address space and the linked list is in* $O(\text{segments})$*, and* segments *can be* $1$*)*

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?*

**Solution:**

*The kernel can just reserve one segment (e.g., the first one) for its own usage,* ***(0.5 P)*** *and disallowing user access to this page.* ***(0.5 P)***

Note: *Using the first segment (with index 0) also has the nice advantage of making a* `NULL` *pointer dereference quite literally a segmentation fault.*

**Common mistakes:**

- *Address space identifiers do not work as the kernel has to work in every address space, as a system call does not change the AS. Therefore, the kernel cannot tag its memory and has to configure each address space individually.*

**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.*

**Solution:**

```
void spin_acquire(int *lock) {
  while(test_and_set(lock, 1));
}

void spin_release(int *lock) {
  test_and_set(lock, 0);
  // alternative solution:
  *lock = 0;
}
```

- ***(1 P)*** *for correct `spin_release()` implementation*
- ***(0.5 P)*** *for correct `test_and_set()` call in `spin_acquire()`*
- ***(0.5 P)*** *for looping correctly in `spin_acquire()`*

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?*

**Solution:**
*A low-priority thread may hold a spinlock while a high-priority thread is waiting for the lock.* ***(0.5 P)*** *In this situation, the scheduler will always choose the high-priority (spinning) thread, so the low-priority thread will never run and cannot release the lock.* ***(0.5 P)*** *Neither thread will make progress.*

*The scheduler can prevent this problem with one of the following:*

- *Priority inheritance: The low-priority thread temporarily receives the priority of the waiting thread, allowing it to run.* ***(1 P)***
- *Prevent lock holder preemption: The scheduler must ensure that a thread cannot be preempted while holding a lock.* ***(1 P)***

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.*

**Solution:**
*A spinlock implementation based on atomic store operations as introduced in the lecture does not fulfill the requirements **Bounded Waiting** and **Performance**.*

***Bounded Waiting*** *Since a simple spinlock does not explicitly coordinate waiters, the CPU's implementation of atomics decides which waiter may enter the critical section. The spinlock therefore cannot guarantee an upper bound on waiting time.*

***Performance*** *The spinlock is wasting CPU time while spinning.*

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.*

**Solution:**

```
#include <pthread.h>

#define NUM_WORK 10

int produce_work(void);
void consume_work(int work_desc);
int work_buf[NUM_WORK] = {0};
int work_len = 0;

pthread_cond_t cv_empty = PTHREAD_COND_INITIALIZER;
```

```
pthread_cond_t cv_fill = PTHREAD_COND_INITIALIZER;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void init() {
  /* Instead of using the initializers above, you may also call
     pthread_mutex_init() and pthread_cond_init() here. */
}

void producer() {
    while (1) {
        int work_desc = produce_work();
        pthread_mutex_lock(&lock);
        while (work_len >= NUM_WORK)
            pthread_cond_wait(&cv_empty, &lock);

        work_buf[work_len++] = work_desc;
        cond_signal(&cv_fill);
        pthread_mutex_unlock(&lock);
    }
}

void consumer() {
    while (1) {
        pthread_mutex_lock(&lock);
        while (work_len <= 0)
            pthread_cond_wait(&cv_fill, &lock);

        int work_desc = work_buf[--work_len];

        pthread_cond_signal(&cv_empty);
        pthread_mutex_unlock(&lock);
        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.*

**Solution:**

- Many-to-One model (user-level threads)*: All threads are managed by a single kernel thread. The kernel is not aware of the existence of user-level threads. The user-level thread library is responsible for scheduling and managing the threads.*
- One-to-One model (kernel-level threads)*: Each user-level thread is mapped to a kernel thread. The kernel is aware of all threads and schedules them independently.*

- Many-to-Many model (hybrid threads)*: Multiple user-level threads are mapped to multiple kernel threads. The kernel is aware only of the kernel-level threads. The user-level thread library is responsible for scheduling the user-level threads on the kernel-level threads.*

***(0.5 P)** each for names, **(0.5 P)** each for descriptions.*

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.*

**Solution:**
- *Mutual exclusion*
- *Progress*
- *Bounded waiting / Fairness*
- *Performance*

***(0.5 P)** each.*

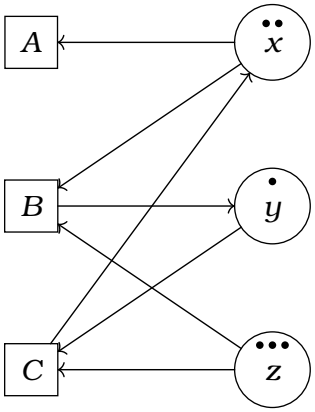
e) Gegeben sei folgender Systemzustand:
- Prozesse A, B und C
- Ressourcen x (2×), y (1×), z (3×)
- A hält x, B hält x und z, C hält y und z
- B möchte y, C möchte x

*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).*

**Solution:**

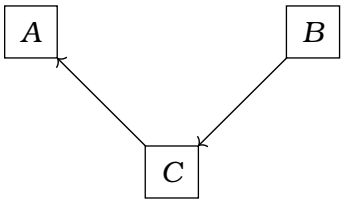



***(–0.5 P)** for any missing or incorrect edge.*

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

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

**Solution:**




*An additional edge from C to B is also accepted.* ***(–0.5 P)*** *for any missing or incorrect edge.*

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

*Is there a deadlock?*

**Solution:**
*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.*

**Solution:**
*Problem name: livelock **(0.5 P)** (or starvation)*

*Approaches, each **(1.5 P)**:*

- *programmed I/O / polling*
  - *CPU busy-waits for new packets to arrive by polling the device state*
  - *suitable for fast devices (busy-waiting has less overhead than frequent interrupts)*
- *mask/disable interrupts*
  - *temporarily mask interupts to allow for some progress (unmask them afterwards)*
  - *if interrupts are masked for too long, the NIC might drop packets*
- *interrupt coalescing*
  - *instead of sending multiple interrupts, send a single interrupt for multiple packet*
  - *packets are buffered upon arrival*
  - *raise an interrupt when the packet buffer is full/reached a certain threshold or after a predefined amount of time has passed since the arrival of a packet (if the buffer threshold is not reached)*

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.*

**Solution:**
*Name of concept: Direct Memory Acess (DMA)* ***(0.5 P)***

*Sequence of a basic copy process using a DMA controller: To copy data between system memory and an I/O device, the CPU sets up a DMA transfer with the DMA controller, which includes specifying the source address, the length, and the destination of data to copy* ***(0.5 P)****. When the DMA controller completes the transfer, it notifies the CPU by raising an interrupt* ***(0.5 P)****.*

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.*

**Solution:**
*blocks per single indirect:* $2^{13}/2^3 = 2^{10}$ blocks ***(0.5 P)***

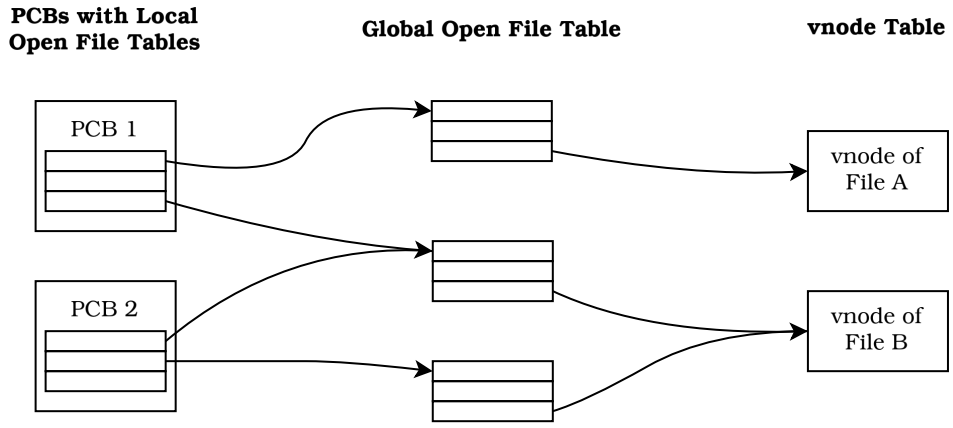
*blocks per double indirect:* $\left(2^{10}\right)^2 = 2^{20}$ blocks ***(0.5 P)***

*max file size:* ***(1.5 P)***

$$
\begin{aligned}
& 128 * 8\text{KiB} + 32 * 2^{10} * 8\text{KiB} + 1 * 2^{20} * 8\text{KiB} \\
&= \left(2^7 + 2^{15} + 2^{20}\right) * 2^{13}\text{B} \\
&= \left(2^0 + 2^8 + 2^{13}\right) * 2^{20}\text{B} \\
&= 1\text{MiB} + 256\text{MiB} + 8192\text{MiB} = 8449\text{MiB}
\end{aligned}
$$

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*?*

**Solution:**
*A file descriptor serves as an index into the local open file table and refers to single entry.*

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*.*

**Solution:**
- *current file position*
- *access rights* ***that file was opened with*** *(also known as access mode)*
  *Note: the access mode might be different to the permissions of this file*
- *(certain) open flags (mostly file status flags, e.g.,* `O_APPEND`*,* `O_SYNC`*, ...)*

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.*

**Solution:**
*A process has an open file descriptor and calls* `fork()` ***(0.5 P)****.*

*Note: As the local open file table is copied on* `fork()` *(but not the global open file table that is not part of the PCB), the inherited file descriptors point to the same open file handles as before.*

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.*

**Solution:**
*A file is opened multiple times **(0.5 P)**.*

*Note: Even when a single process calls `open()` multiple times with identical arguments, each call creates a new open file handle in the global open file table.*

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.*

**Solution:**
*The page cache buffers file contents from storage devices in system memory as storage devices are typically much slower (i.e., higher access lantency and lower throughput) than system memory **(0.5 P)**. Doing so allows the page cache to speed up repeated reads (spacial and/or temporal locality) and to buffer writes in fast memory without having to wait for the storage device **(0.5 P)**.*

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.*

**Solution:**
*reasons for write-back, each **(0.5 P)**:*
- *explicit write-back through syscalls or commands (e.g., `fsync()`, `msync()`, `sync`, ...)*
- *periodic write-back of dirty pages done by the OS*
- *page cache evictions on memory pressure (OS reclaims memory used for buffering file contents)*

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.*

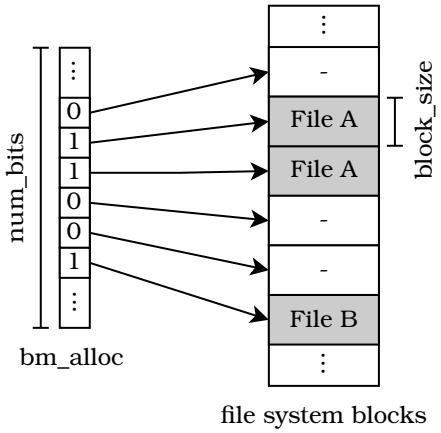
**Solution:**
*When contents buffered in the page cache were not modified since the last write-back, their write-back can be skipped **(0.5 P)**. This is because these buffered contents are in sync with the backing storage **(0.5 P)** (i.e., the in-memory copy is identical to the copy stored on the storage device).*

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`*.*

**Solution:**

```
bool bm_find_next(bitmap_t *bm, long *cur, bool bit) {
  for (long i = *cur; i < bm->num_bits; i++) {  // .5P
    if (bm_test(bm, i) == bit) {                // .5P
      *cur = i;                                 // |
      return true;                              // .5P
    }
  }

  return false;                                 // .5P
}
```

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 $B_{\text{bm}}[i] = B_{\text{bm}}[i+1] = ... = B_{\text{bm}}[i+\text{len}-1] =$ `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 $B_{\text{bm}}[i] = B_{\text{bm}}[i+1] = ... = B_{\text{bm}}[i+\text{len}-1] =$ `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.*

**Solution:**

```
bool bm_find_range(bitmap_t *bm, long *cur, long len, bool bit) {
  while(1) {                              // .5P
    if (!bm_find_next(bm, cur, bit))      // |
      return false;                       // .5P

    long end = *cur + 1;
    if (!bm_find_next(bm, &end, !bit))    // .5P
      end = bm->num_bits;                 // .5P

    if (end - start >= len)               // |
      return true;                        // .5P

    *cur = end;
  }
}
```

***Common mistakes:***

- *off-by-one errors when checking the length of a bit range*
- *out-of-bounds accesses / calling `bm_test()` with an invalid index*
- *ignoring the return value of `bm_find_next()`*
- *not handling ranges at the end of the bitmap (lookup of end fails)*
- *not returning the start index in *cur*
- *not advancing *cur and ending up in an endless loop*
- *operator precedence (postfix increment vs. pointer dereference)*

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`.*

**Solution:**

```
long fs_alloc_file(fs_t *fs, long file_sz) {
  long n_blocks = (file_sz + fs->block_size - 1) / fs->block_size;  // 1P

  long s = 0;
  if (!bm_find_range(fs->bm_alloc, &s, n_blocks, false))           // |
    return -1;                                                     // .5P

  bm_set_range(fs->bm_alloc, s, n_blocks);                         // .5P

  return s;                                                        // .5P
}
```

***Common mistakes:***

- *ignoring the return value of* `bm_find_range()`
- *mixing up copy-by-value and copy-by-reference arguments*
- *not rounding up the number of blocks*

**Total: 20 P**