

“If a **kernel-level thread (KLT)** executing in a **user-level monitor** acquires a **held kernel-lock**, then this **KLT** is **blocked** and the **monitor** is automatically **reopened** for other **KLTs**.”

korrekt

inkorrekt

8. „Zwei **mehrfädige (multi-threaded)** Anwendungen, von denen eine **nur aus PULTs** und die andere **nur aus KLTs** besteht, können **nicht** miteinander kommunizieren.“
 “Two **multi-threaded** applications, one consisting **solely of PULTs**, the other consisting **solely of KLTs**, **cannot** communicate with each other.”

korrekt

inkorrekt

9. „Wird bei einer **erweiterbaren Hashdatei** das **letzte Element** aus einem Datencontainer entnommen, so **muss** die **zugehörige Generationsnummer** im Basisvektor **verringert** und der leere **Datencontainer** mit seinem jeweiligen **Partnercontainer verschmolzen** werden.“
 “After having removed the **last element** from a data container of an **extensible hash file**, you **must decrement** the **according generation number** in the basis vector and **merge** the empty **data container** with its respective **partner** container.”

korrekt

inkorrekt

Aufgabe 2 / Question 2

(4 + 3 + 5)

1. „Gegeben sei ein Einprozessorsystem mit **nicht verdrängender Prioritätsplanungsstrategie (non-preemptive priority scheduling policy)** und **statischen Prioritäten**. **Unterstreichen** Sie von den folgenden Problemen all diejenigen, die auftreten können.“
 “Assume a single-processor system with a **non-preemptive priority scheduling policy** and **static priorities**. **Underline** all of the following problems that can occur.”

Prozesse können aushungern / processes can starve

Durch Prioritätsumkehr können niedrigst priore Prozesse aushungern / *lowest priority processes can starve due to priority inversion*

Die Auswahlfunktion arbeitet ineffizient bei vielen bereiten Prozessen / *the selection function works inefficiently if many processes are ready*

Die Verwendung von Spinlocks zur Sicherung kritischer Abschnitte kann zum Systemstillstand führen / Using spin locks to guard critical sections can lead to a system life-lock

2. „Welche **unerwünschten** Auswirkungen kann die **unbedarfte** Verwendung des **Test-And-Set-Befehls** zur **Synchronisation von Anwendungen** auf **Mehrprozessorsystemen** haben?“
 “What **undesired** effects can be caused by **carelessly** using the **Test-And-Set instruction** for **synchronizing applications** in **multi-processor systems**?”

Busy waiting in front of medium/long critical section wastes processor time (1)
Ping-pong effect (1) between cached synchronization variables slowing down the memory bus significantly
Live-lock of system in case of non-preemptive scheduling (1)

3. „Analysieren Sie die folgende Programmskizze des **allgemeinen Erzeuger/Verbraucher-Problems mit begrenztem Puffer** (*bounded buffer*) in gewohnter Weise! Verbessern Sie das Programm mit den gegebenen Semaphoren so, dass es eine **effektive Lösung** wird.“
*“Analyze the following program draft of the **general producer-consumer problem with a bounded buffer** in the usual way. Improve the program with the given semaphores to achieve an effective solution.”*

```

/* global data declarations */
item BUFFER[100];          /* bounded buffer of length 100 */
int i=0; int j =0;        /* indices to fetch & store */
bin_sem plock=1;         /* Only 1 producer can access buffer */
bin_sem clock=1;         /* Only 1 consumer can access buffer */
count_sem(free)=100;     /* controls # of free buffer slots */
count_sem(full)=0;      /* controls # of occupied buffer slots */

producer()
{
    item x;
    x = produce();        /* can be time consuming */
    p(free);
    p(plock);
    BUFFER[i] = x;       /* put product into buffer */
    i=(i+1)%100;
    v(plock);
    v(full)
}

consumer()
{
    item x;
    p(full);
    p(clock);
    x = BUFFER[j];       /* fetch product from buffer */
    v(clock);
    j=(j+1)%100;         /* program error, concurrent consumers
                          fetch from the same buffer entry */
    /* exchange the above two red code lines => OK */
    v(free);
    consume(x);          /* can be time consuming */
}

```

Mutual exclusion: NO, see comment above

Progress: YES, no process in its RS prevents a waiting process from entering

Portability: YES, no assumption about scheduling or # of CPUs

Bounded Waiting: YES, at least in case of strict semaphores

Aufgabe 3 / Question 3**(1 + 2 + 2 + 2 + 5 Punkte/marks)**

1. „Erläutern Sie die **Auswirkung** eines blockierenden Systemaufrufs in einem PULT!“
 “*Explain the **implication** of a blocking system call within a PULT.*”

If a PULT blocks due to a system call the entire task is blocked (1).

2. „Erläutern Sie, wie die **K42** Systemarchitekten die obige Auswirkung gemildert haben!“
 “*Explain how the system architects of **K42** improved the implication above.*”

Whenever a PULT blocks due to a syscall and its dispatcher’s time slice has not been expired, the kernel sends a notification to the user-level scheduler (1), enabling him to schedule another runnable PULT (1) (similar to scheduler activations). The PULT state is provided in the corresponding dispatcher structure (~ ULTCB) (1 bonus P).

3. „Erläutern Sie den Begriff **Seitendiebstahl** (*page stealing*)! **Wo** und **wann** kann Seitendiebstahl vorkommen?“
 “*Explain the notion **page stealing**. **Where** and **when** can page stealing occur?*”

Page stealing can occur only in virtual memory systems with a global page replacement policy (1) and indicates that a page-fault in an address space A can be resolved by taking a frame away from a different address space B and giving it to address space A instead (1).

4. „Kann das folgende System aus zwei Prozessen A und B sowie den zwei exklusiven Betriebsmitteln R und S verklemmen? **Begründen** Sie Ihre Meinung!“
 “*Can the following system consisting of the two processes A and B and the two exclusive resources R and S enter a deadlock? **Exemplify** your choice.*”

A: allocate(R), allocate(S), release(S), release(R)

B: allocate(S), release(S), allocate(R), release(R)

No, the system is free of deadlocks (1).

Reasoning (1), alternatively

(a) B violates the hold-and-wait-condition

(b) There is no circular wait: Whenever A waits for B, B will release the held resource and not wait for A. If B waits for A, B does not hold any resources, so A need not wait for B.

Solutions without explicitly naming the conditions are equally valid.

5. „Gegeben sei ein Einprozessorsystem mit drei exklusiven Betriebsmittel R, S und T, sowie vier Prozessen A(1), B(2), C(3) und D(4) (die Zahlen in Klammern seien die Prioritäten der Prozesse). Vervollständigen Sie den untenstehenden Ablaufplan, wobei **striktes Prioritäten-scheduling mit Verdrängung** und das Betriebsmittelvergabeprotokoll **Prioritätsvererbung** angenommen werden. Beachten Sie, dass die Prozesse erst zu den fett markierten Zeitpunkten (also 0, 3, 6, und 7) bereit werden. Tragen Sie gemäß der Vorlage an allen Stellen, an denen sich die Prioritäten der Prozesse ändern, die jeweils neuen Prioritätswerte ein.“

“Assume a single-processor system with three exclusive resources R, S, and T, and four processes A(1), B(2), C(3), and D(4), where the numbers in parentheses denote the priorities of these processes. Complete the schedule below according to **strict priority driven scheduling with preemption** and the resource allocation protocol **priority inheritance**. Pay attention: Processes become ready only at the bold-marked points in time (i.e., 0, 3, 6, and 7). Mark all time periods with changing priorities.”

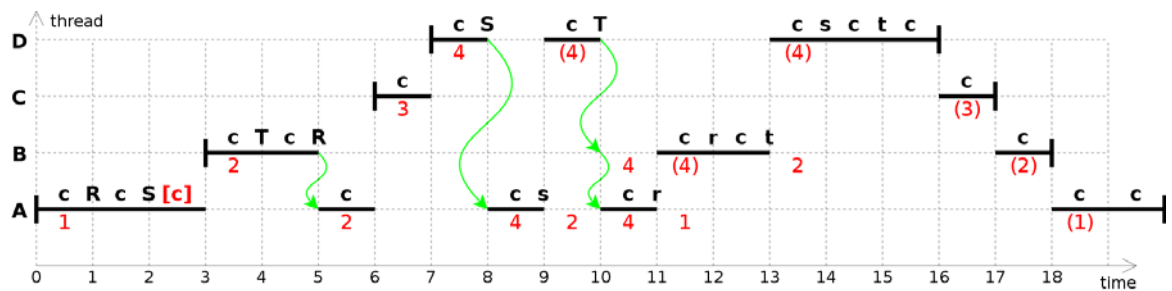
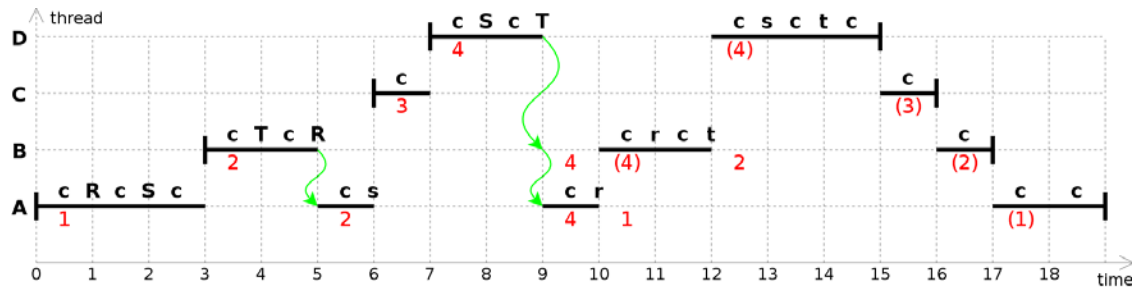
Operationsfolge pro Prozess, wobei eine Rechenphase “c” jeweils eine Zeiteinheit, alle anderen Operationen jeweils 0 Zeiteinheiten dauern und stets am Ende einer Rechenphase c ausgeführt werden sollen:

Sequence of operations per process; each compute phase “c” takes exactly one time unit, all other operations need 0 time units and will take place at the end of a compute phase c.

- D: c, allocate(S), c, allocate(T), c, release(S), c, release(T), c, exit
- C: c, c, exit
- B: c, allocate(T), c, allocate(R), c, release(R), c, release(T), c, exit
- A: c, allocate(R), c, allocate(S), c, c, release(S), c, release(R), c, c, exit

Wenn Sie in der unten angeführten Skizze das Ereignis „allocate(X)“ für $X \in \{R, S, T\}$ markieren wollen, benutzen Sie den Großbuchstaben X, für das Ereignis „release(X)“ benutzen Sie x.

Use a capital X if you want to mark the event “allocate(X)” with $X \in \{R, S, T\}$ in the chart below; for the event “release(X)” use a lowercase x.



The top diagram shows the “correct” solution, the diagram below shows a frequent solution (extra compute phase (-1)). At any time the thread of highest priority executes (1), threads block on held resources (1), non-transitive priority inheritance (1), transitive inheritance (1), drop priority after release (1).

Additional or incorrectly placed phases or events (-1 total), missing or wrong events (-1 total), missing priority and no arrow (-1 total).

Both # and c (even nothing) may be used for compute phases. Arrows denoting inheritance are optional iff the new priority is correctly indicated.

Aufgabe 4 / Question 4**(2 + 2 + 2 + 4 + 2 Punkte/marks)**

1. „Erläutern Sie, warum beim Threadwechsel zwischen Kernel-Level Threads (KLTs) innerhalb der **Prozedur** `THREAD_SWITCH` der **Befehlszeiger** (*instruction pointer*) **nicht gerettet** werden muss, während dies beim **Makro** `THREAD_SWITCH` **nötig** ist.“

*“Describe why you **do not have** to save the **instruction pointer** when switching between two kernel-level threads (KLTs) using the **procedure** `THREAD_SWITCH`, whereas this is **necessary** when using a **macro** `THREAD_SWITCH`.”*

In the procedure `THREAD_SWITCH` switching the stack pointer always occurs at the very same instruction pointer, thus you do not have to save this value. The corresponding IP of the caller is always saved via the procedure call. (0.5) In a macro this can happen at many different IPs, the code of `THREAD_SWITCH` is expanded to various “callers”, thus it must be saved manually inside the macro `THREAD_SWITCH`(0.5)

2. „Warum ist der `fork()` Systemaufruf in modernen Linux-Implementierungen nicht mehr so aufwändig wie er dies in frühen Unix Versionen war? **Erläutern** Sie **ausführlich**, was das System für die **neue Task bereitstellt**.“

*“Why is the system call `fork()` in modern Linux implementations no longer as complex as it was in early Unix versions. **Explain in detail** what the system has to **provide** for the **new task**.”*

In early Unix versions despite the code segment all other segments have to be copied explicitly. (1) Nowadays, you only copy the address space description, i.e. page-tables etc. However, you are using the technique: copy-on-write, i.e. whenever a parent or its child writes to a COW-page, then and only then a copy has to be made. (1) All COW-pages in both page tables have to be reorganized as READ-ONLY pages, before parent or child can run again. (1)

3. „Diskutieren Sie die **Orthogonalität** der folgenden beiden Kommunikationsvarianten: **verbindungsorientierte** bzw. **lose IPC** und **direkte** bzw. **indirekte IPC!**“

*“Discuss the **orthogonality** of the following two communication variants: **connection-oriented** versus **connectionless IPC** and **direct** versus **indirect IPC**.”*

The following combinations are useful:

- 1. Connection-oriented and indirect IPC (socket)**
- 2. Connectionless and direct IPC (see L4) or indirect IPC (see Unix)**

4. „Geben Sie möglichst präzise an, in welchen Adressbereichen (*regions, sections*) die jeweiligen Variablen des folgenden C++ Programms in einem modernen Linux System abgespeichert werden.“

“Indicate as precisely as possible in which address regions (sections) the variables of the following program will be stored.”

```
static long A[4];           /* A not initialized global data */
unsigned B[4] = {1, 2, 3, 4}; /* B initialized global data */
```

```

long mult(int j, long *X, long *Y) { /*stack, stack, stack */
    for (int i = 0; i < j; i++) { /* i stack */
        X[i] = Y[i] * Y[i];
    }
}

void main(void) {
    long TEMP[4]; /* TEMP stack */
    mult(4, &TEMP[0], &A[0]);
    for (int i = 0; i < 4; i++) { /* i stack */
        printf("%ld\n", TEMP[i]);
    }
}

```

5. „Beim Seitentausch auf Verlangen (*demand paging*) ist es mitunter ratsam, **Kacheln mit Nullen zu füllen**, bevor sie der Anwendung zugänglich gemacht werden. **Welche Seiten** sollten „genullte“ Kacheln erhalten und **warum „nullt“** man die Kacheln überhaupt?“
*“With paging on demand it can be advisable to **zero-fill page frames** before making them accessible to the application. **Which pages** should be backed by zeroed frames and **why** should you **zero-fill** them at all?”*

Pages with non initialized global data or heap or stack (1) to support lazy programmers and to avoid a covered channel, i.e. the next owner of a page frame might first try to read the complete page to get the information of the previous page owner (1).

Aufgabe 5 / Question 5

(1 + 1 + 4 + 3 + 3 Punkte/marks)

1. „Welche Angaben enthält ein Verzeichniseintrag (*directory entry*) im **Linux EXT2-Dateisystem**?“
*“Which fields does a directory entry of the **Linux file systems EXT2** contain?”*

**Filename (0.5), inode number (0.5),
link to the next valid directory entry (0.5)**

2. „Welches der folgenden **Dateiattribute** ist **nicht** Bestandteil des Dateikopfes (*inode*) einer Linux EXT2 Datei?“ (Zutreffendes unterstreichen.)
*“Which of the following **file attributes** is **not** part of an inode of a Linux EXT2 file?”
(Underline appropriately.)*

Dateilänge / *file length*

Anzahl symbolischer Links / number of symbolic links

Zugriffsrechte / *accessrights*

Eigentümer / *owner*

3. „Erläutern Sie so knapp und präzise wie möglich, wie Sie erreichen können, dass Sie stets den aktuellsten und konsistenten Inhalt einer **mehrfach geöffneten B*-Indexsequentiellen Datei** erhalten, egal ob Sie sequentiell oder über einen Schlüssel zugreifen.“
*“Describe as short but as precisely as possible how you can achieve that you always get the most current and consistent content of a **multiply opened B*-indexed sequential file**, regardless of whether you are accessing the file sequentially or via a key.”*

Use read-write locks (4), i.e. as long as concurrent readers are active, no writer is allowed, and no two writers are allowed.

4. „Nennen Sie die **drei Softwareschichten**, die im Schichtenmodell zwischen E/A-Geräten und Anwendung angesiedelt sind und erläutern Sie für jede Schicht deren **Hauptaufgabe!**“
*“Enumerate the **three software layers** that reside between the I/O device and the applications and explain for each layer its **specific main task**.”*

Logical I/O or device independent subsystem or VFS: identify requested device driver, translate high-level I/O request into device-class specific request

Physical I/O or (device specific) driver: translate device class interface (block or character device interface) into a sequence of device specific commands

Interrupt handler: identify device events. Trigger their handling by unblocking a previously blocked driver thread (or create a new handler thread).

5. „Zählen Sie **drei** verschiedene RAID-Architekturen auf und erläutern Sie jeweils deren **wesentliche Charakteristik**.“
*“Enumerate **three** different RAID architectures and explain their **main characteristics**.”*

RAID 0 (0.5): Block Striping (0.5), multiples disk drive to contain different stripes of a file. Improved performance, no gain in reliability.

RAID 1 (0.5): Mirroring (0.5), you keep two identical disk volumes, i.e. when you write to a file you update both file contents on both disk volumes. In some cases, the mirrored disk volume is only used as a fast backup media. Loss in performance, but better reliability.

RAID 2 (0.5): Hamming Code

RAID 3 (0.5): Bit-interleaved parity

RAID 4 (0.5): Block-level parity on a single parity disk & block striping (0.5). Increased reliability, as long as parity disk lives.

RAID 5 (0.5): Block-level parity on different disks & block striping (0.5)

RAID 10 (0.5): RAID 0 and RAID 1 combined (0.5), RAID 0 consisting of several RAID 1