

Some comments on the voluntary examinations.

### Question 1:

„Zählen Sie **drei konkrete Anwendungen** auf, in denen man einen **m:n-Kommunikationskanal** verwenden kann, wobei  $m, n > 1$ !“

“Enumerate **three concrete applications** for which you can use a **m:n-channel** ( $m, n > 1$ )”.

→ *What is a m:n-channel?*

- *Software- or Hardware-object?*

→ *How to use this object for an application?*

→ *Enumerate 3 concrete applications*

← *Only few solved this question completely.*

„Geben Sie **drei verschiedene konkrete Anwendungen** des **Timeout-Mechanismus** an!“

“Enumerate **three different concrete applications** of the **time-out mechanism**.”

→ *What is a time-out?*

- *Used for signaling a thread after some limited time*

→ *When to use such a timeout signal?*

→ *Enumerate 3 different applications*

← *About 50 % solved this question completely.*

Auf einem **Mehrprozessorsystem** mit **Mehrprogrammbetrieb** (*multi programming*) können **gleichzeitig mehrere „Reine User-Level-Threads“** (*PULTs*) im Threadzustand **„Rechnend“** (*running*) sein.“

“On a **multi-processor** with **multi-programming** **multiple pure-user-level threads** may be in the thread state **“Running”** at the same time.”

→ *SMP+ Multi-programming already answer this question*

← *Only few people did not solve this question*

„Ein **Anwendungssystem** aus  $n > 1$  **Kernel-Level-Threads** kann verklemmen, wenn diese Threads **geschachtelte Monitore** verwenden.“

“An **application system** of  $n > 1$  **kernel-level-threads** may deadlock, if its threads use **nested monitors**.”

→ *Nested monitor?*

- *Interface function of monitor M1 calls interface function of another Monitor M2*

→ *Potential for a deadlock?*

← *About 50 % solved this question completely.*

„Annahme: Eine Applikation benutzt eine **Semaphore S**, um den **exklusiven** Zugriff ihrer Threads auf ein Betriebsmittel R zu **synchronisieren**. Wenn in einem der Threads, die das Betriebsmittel R benutzen, ein **P(S)** Aufruf fehlt, so ist die **exklusive Nutzung** des Betriebsmittels R **nicht mehr garantiert**.“

“Suppose an application is using a **semaphore S** to **synchronize** its threads for **exclusive** access to a resource R. If a **P(S)** is missing in one of its thread using the resource R the **exclusive usage** of this resource R is **no longer guaranteed**.”

←Only few people did not solve this question

„Jeder **blockierende Systemaufruf** (system call) bewirkt einen **Taskwechsel**.“

“Every **blocking system call** invokes a **task switch**.”

→ Thread  $\neq$  task  $\Rightarrow$ ?

←Only few people did not solve this question

„Annahme: Außer einer Anwendung (application) existiert ansonsten keine Anwendungslast auf dem System. Diese Applikation aus reinen „User-Level Threads“ kann in einem Mehrprozessorsystem nicht schneller als auf einem Einprozessorsystem abgearbeitet werden.“

“Suppose besides the application there is no other application load on the system. This application with pure user-level-threads can not be executed faster on a multiple-processor system than on a single processor-system.”

←Only few people did not solve this question

„In **Java** werden zur Synchronisation von Threads **Monitore** angeboten“.

“Java offers **monitors** to synchronize threads .”

←Only few people did not solve this question

## Question 2

„Diskutieren Sie die Implikationen, wenn die zentrale Umschaltfunktion **thread\_switch** als **Makro** implementiert wird?“

“Discuss the implications if the central switch function **thread\_switch** is implemented as a **macro**.”

→ What is a macro compared to a procedure?

→ Implications? Direct ones and side effects?

→ Code size and code timing

→ What kind of instruction pointer may be affected?

Some partially **incomplete, imprecise, wrong** answers:

- Error in macro **thread\_switch** may lead to a subsystem breakdown
- Source code is larger
- To implement **thread\_switch** as a macro you have to do that at user level
- More effective usage of the CPU, less idle-time
- It's more difficult to implement exclusive access to the ready queue
- Higher speed on SMP, because every CPU might have the macro in its cache
- A macro implementation of **thread\_switch** is clearer

„Erklären Sie die **Hauptunterschiede** zwischen **Kernel-Level-(managed) Threads** und **User-Level-(managed) Threads**“

“Explain the **main differences** between **kernel-level-(managed)-threads** and **user-level-(managed)-threads**.”

→ Location of TCBs

→ Knowledge to kernel

→ Internal or external scheduling

→ Reaction to blocking system calls or blocking exceptions

→ Thread-switching overhead, ...

Some partially **incomplete, imprecise, wrong** answers:

- Task blocks, when one of its PULTs blocks, but if a PULT is blocked within a blocking system\_call, its complete task is blocked
- All PULTs of same task must run on same CPU
- With PULTs every process manages its own threads by itself in a table
- System call always interrupts the complete KLT
- Heavy-weight KLT always needs system calls
- KLTs run in kernel-mode
- If one of the PULTS gets blocked by an interrupt, all other PULTS in the same task are blocked, too.
- Stacks of KLTs are in kernel land, stacks of PULTs in user land.
- KLTs run in kernel-mode, that's why you have to enter the kernel if you want to switch them
- Certain system\_calls are more expensive, because they have to be implemented within the kernel
- It's more difficult to handle blocking system\_calls with ULTs
- On a CPU at most 1 KLT is running

„Sie sollen das **allgemeine Produzenten Verbraucherproblem** ( $p, c \geq 1$ ) lösen, wobei unabhängig voneinander mal Produzenten, mal Konsumenten schneller vorankommen können.“

“Solve the general producer/consumer problem whereby independently of each other producers or consumers may outrun the other.

Welches Threadmodell werden Sie verwenden?

What thread-model will you apply?

Begründen Sie Ihre Wahl?

Reason your choice.

Skizzieren Sie eine effiziente Lösung!

Outline an efficient solution.

→ What is the general producer/consumer problem?

→ We have to synchronize up to  $p$  producers and up to  $c$  consumers

← Majority of wrong outlines didn't consider  $p, c > 1$

Some didn't distinguish between critical- and non critical section

Some additional imprecise answers:

1. KLTs, because otherwise no mutual-exclusion can be guaranteed when accessing the buffer
2. 3 thread-state model (running, ready, blocked)
3. Monitor, Semaphore, Mutual Exclusion, ...
4. KLTs, because you can use TLS-operations, otherwise you must use monitors
5. KLTs, because they cannot corrupt each others stacks
6. ULTs, because the common buffer is in the same address space, which enhances accesses to it

### Question 3:

„Skizzieren Sie inklusive aller Übergangsfunktionen des Threadzustandsdiagramm von Java!“

“Outline the thread state diagram of Java including all state-transition functions.”

- Which thread states exist?
- Names of state-transition functions?
- In JAVA? (90% forgot that!)

Some imprecise answers:

- In Java there are 3 thread states
- Transition functions with no names
- There is no difference between Java-TSM and a 3-TSM.
- Java has a „blocking“, a „waiting“ and a „sleeping“ state
- `signal()` is used for: „not running“→„running“
- Obscure arrows between empty circles

„Beschreiben Sie so knapp wie möglich, gleichwohl so ausführlich wie nötig das Lotterieverfahren!“

“Describe as short as possible, but as completely as necessary lottery scheduling”

- Why are tickets used?
- How are the Tickets distributed?
- What is the jackpot in the lottery?

Some imprecise answers:

- Jeder Thread bekommt ein Los.
- Es wird zufällig ein Thread aus der Ready-Queue gewählt.
- Das hat was mit Zufall zu tun.
- Es wird zufällig ein Thread gelost, der für eine zufällige Zeit läuft.

„Beschreiben Sie so knapp wie möglich das Prinzip: „Trennung von Strategie und Mechanismus“, und geben Sie hierfür ein einleuchtendes Beispiel an!“

“Describe as short as possible the meaning of the principle : “separation of policy and mechanism”, and give a clear example.”

→ *Why Separation?*

→ *Do not forget the Example!*

→ *Show clearly the policy and the mechanism in your example!*

*Some imprecise answers:*

- Policy and Mechanism are *different parts* of a Scheduler.
- This principle is *only used* in scheduling.
- The principle is the *same as modularization*.
- Mechanism means *parameterization*.

*„Das Leser-/Schreiberproblem kann auf verschiedene Arten formuliert werden, wenn man berücksichtigt, welche Art von Prozessen (Leser oder Schreiber) zu welchem Zeitpunkt gestartet werden kann. Beschreiben Sie sorgfältig zwei verschiedene Lösungsvarianten des Problems. Schildern sie für jede Variante, was passiert, wenn ein Leser oder Schreiber bereit wird, auf die Datenbank zuzugreifen, und was passiert, wenn ein Prozess terminiert.“*

*“The reader-/writer-problem can be formulated in several ways with regard to the ordering in which category of processes (reader or writer) can be started. Carefully describe two different variations of the problem. For each variation, specify what happens when a reader or writer becomes ready to access the data base, and what happens when a process terminates.”*

→ *What is the reader-/writer Problem?*

→ *Which issues need to be solved?*

→ *Find two **different** solutions!*

*Some imprecise answers:*

- *Transactions solve the reader-/writer problem* mentioned in this lecture.
- The *Producer-Consumer-Problem* is just another Name for the Reader-Writer-Problem.
- Mutual exclusion (i.e. only one thread works) is sufficient.
- There exist different locations where data is read or written.
- *There is always only one* writer.
- *Two different starting situations* make two different solutions.
- Everything is said after the first process is started.
- *Termination of a process always means abnormal termination.*
- New processes do *never* arrive

#### **Question 4:**

*„Ein Rechner hat acht Bandlaufwerke, um die n Prozesse konkurrieren. Jeder Prozess braucht bis zu 2 Bandlaufwerke. Für welche n kann es keine Verklemmung (deadlock) geben?“*

*“A computer has eight tape drives, with n processes competing for them. Each process may need up to two tape drives. For which values of n is the system deadlock free?”*

- **$n \leq 7$  oder  $n < 8$**
- **$n = 7$  is not correct**

„In einem elektronischen Überweisungssystem laufen Tausende von identischen Prozessen. Jeder Prozess liest die Überweisungssumme und die beiden Kontonummern von einem Eingabegerät. Anschließend sperrt er beide Konten, überweist das Geld und gibt dann die Konten wieder frei. Die Verklemmungsgefahr ist in diesem System sehr hoch. Lassen Sie sich ein Protokoll einfallen, um Verklemmungen zu verhindern. Dabei darf kein Konto wieder freigegeben werden, bevor die Überweisung beendet ist. (Mit anderen Worten, eine **Lösung, die ein Konto sperrt und es sofort wieder freigibt, falls das zweite Konto gesperrt ist, gilt nicht.**)“

“In an electronic funds transfer system, there are thousands of identical processes that work as follows. Each process reads an input line specifying an amount of money, the account to be credited, and the account to be debited. Then it locks both accounts and transfers the money, releasing the locks when done. With many processes running in parallel, there is a very real danger that having locked account  $x$  it will be unable to lock the other account  $y$ , because  $y$  has been locked by a process now waiting for  $x$ . Devise a protocol that avoids deadlocks. Do not release the lock of an account until you have completed the transaction. (In other words, solutions that lock one account and the release it immediately if the other lock is already locked are not allowed.)”

→ How to avoid deadlocks?

→ Use the appropriate avoidance policy!

→ Use the ordering scheme:

Assume, there is a global ordering amongst all accounts (just use their account numbers with an appropriate prefix)

Locking protocol:

1. Lock account with lower account number
2. Lock account with higher account number

Fünf Prozesse, P1 bis P5, kommen gleichzeitig in einem Einprozessorsystem an. Die Ausführungszeiten, Sollzeitpunkte und Prioritäten (5 ist die höchste Priorität) sind durch folgende Tabelle gegeben. Stellen Sie für jede der drei folgenden Umschaltstrategien den Ablauf durch ein Gantt-Diagramm dar.“

“Five processes, P1 ... P5 arrive at a single-processor system at the same time. The execution times, the deadlines, and the priorities are given in the following table. Show the resulting schedule for the following three scheduling policies in the appropriate Gantt diagram.”

←Some minor errors due to wrong priority interpretation

## Question 5:

„Erläutern Sie den Begriff: Erweiterbarer Kern! Geben Sie ein konkretes Beispiel für einen erweiterbaren Kern an!“

“Explain the notion: Extensible Kernel. Give a concrete example for an extensible kernel.”

→ What is a kernel, i.e. which functionality does it offer?

→ NOTE: kernel  $\neq$  systems

→ What is an extensible kernel?

→ When can a kernel be extended?

Some imprecise or **wrong** answers:

L4KA is an extensible microkernel (NO,NO,NO!!)

Liedtke and Druschel both believe(d): Extensible kernels are obsolete

Erweiterbarer Kern bedeutet, dass die Befehle (Systemaufrufe), die im Kern bereits vorhanden sind, durch den Benutzer erweiterbar sind

Im Kern sind nur die wichtigsten Dinge

Einem erweiter. Kern können Funktionen via Bibliotheken hinzugefügt werden, ohne dass sich dessen Struktur ändert und er neu übersetzt werden müsste.

Ein Mikrokern ist durch die Auslagerung von nicht essentiellen Servern ins Userland gerade dadurch erweiterbar.

Möglichkeit ein SMP um weitere Prozessoren zu erweitern, ohne neuen Kern zu benötigen.

Ein erweiterbarer Kern ist ein Kern, der auf Applikationsebene durch Dienste erweiterbar ist, welche Funktionen des BS übernehmen können

„Zählen Sie wenigstens 3 verschiedenartige Bibliotheksfunktionen auf, die **keinen Systemcall aufrufen**.“

“Enumerate at least three different library functions that do **not invoke a system call**.”

← What is a system call?

→ Mode switch from user mode to kernel mode, i.e. call of a function, hided within the kernel, e.g. **send\_message( )**

← What is a library function?

→ Call within user land, e.g. **fopen( )**

„Worin liegt der wesentliche Unterschied zwischen einem **Systemcall** und sonstigen **Ausnahmen** (exception)?“

“What is the main difference between a **system call** and **other exceptions**?”

When does a `system_call` happen?

When do exceptions happen?

Where are they handled?

Some incomplete or wrong answers:

Other exceptions, which are asynchronous, may happen in one run, but not in the next.

Bei einem `system_call` wird in den Kernmodus gewechselt.

System\_Calls sind nicht reproduzierbar, bei einem Durchlauf kann es beispielsweise 100 pagefaults, beim anderen nur 10 pagefaults geben