

# Solution System Architecture

( Examination of September 20, 2002 )

## Aufgabe/Question 1 (Zum Aufwärmen/Warm up, 3 + 3 + 1 + ... + 1 Punkte/marks)

1. „Zählen Sie **drei** verschiedene Möglichkeiten auf, wie im Rahmen einer **Kommunikation** (IPC) **Nachrichten** zwischen **zwei Adressräumen transportiert** werden können!“  
*“Enumerate **three** different ways, on how to transfer messages within an IPC between two different address spaces.”*

Register (if message is short enough)

Containers in a common shared memory, message contains address of container

Copy in/out the whole message via the kernel

2. „Zählen Sie **drei prinzipielle Methoden** im Umgang mit **Verklemmungen** (*deadlocks*) auf und geben Sie hierzu jeweils **eine konkrete Beispielmethode** an!“  
*“Enumerate **three principal schemes** for dealing with **deadlocks** and give a **concrete example** for each of these **methods**.”*

Prevention, e.g. install a linear ordering of resource types, successive requests for resources are only allowed according to that ordering

Avoidance, do not start a thread if all its request + those of the current active threads might lead to a deadlock

Detection + resolution, use the standard banker algorithm, swap out youngest threads

Einige der folgenden Aussagen sind korrekt, einige inkorrekt. Unterstreichen Sie „korrekt“, wenn die Aussage korrekt ist, unterstreichen Sie „inkorrekt“, wenn die Aussage inkorrekt ist.

*Some of the following statements are correct, some are incorrect. Underline “korrekt” if the statement is correct; underline “inkorrekt” if the statement is incorrect!*

3. „Das **Halbierungsverfahren** (*buddy system*) kann sowohl zu **internem** (*internal*) als auch zu **externem** (*external*) **Verschnitt** (*fragmentation*) führen.“  
*“The **buddy system** may lead to **internal** as well as to **external fragmentation**.”*

korrekt

inkorrekt

4. „Alle **Seitentabellen** eines **aktiven Adressraums** sind in einem System mit **mehrstufigem Adresstransformationsschema** im **physischen Hauptspeicher** (*main memory*) **abgebildet**.“  
*“All **page tables** of an **active address space** in a system with a **multi-level address translation scheme** are mapped to **physical main memory**.”*

korrekt

inkorrekt

5. „Das **Abfangen** (*interception*) einer Nachricht ist der **Bedrohungstyp** (*type of threat*), der in erster Linie die **Verfügbarkeit** (*availability*) eines Systems angreift.“  
*“The **interception** of a message is that **threat type** that first of all attacks **the availability** of a system.”*

korrekt

inkorrekt

6. „In einem **Mikrokern basierten** System ist bei **gleicher Anwendungslast** die **Zahl der Adressräume größer** als beim funktional vergleichbaren **monolithischen** System.“  
*“In a **micro kernel based** system together with a **load of applications**, the **number of address spaces** is **larger** than in a functionally comparable **monolithic** system running the same application load.”*

[korrekt](#)

inkorrekt

7. **Threadkontrolle** wird nur für **abhängige (dependent) Threads** benötigt.“  
*“**Thread control** is only needed for **dependent threads**.”*

korrekt

[inkorrekt](#)

8. „Um **Wettlaufsituationen (race conditions)** um ein gemeinsam benutztes Betriebsmittel zu vermeiden, benötigt man **Synchronisationsmechanismen**.“  
*“To avoid **race conditions** with a shared resource you need **synchronization mechanisms**.”*

[korrekt](#)

inkorrekt

### Aufgabe/Question 2

(5 + 3 + 2 + 2 Punkte/marks)

1. „Annahme: Der Dateisystempuffer und der Puffer für die Namensübersetzung eines **Unix** System seien beide **völlig leer**. **Wie viele Plattenblöcke** müssen **mindestens gelesen** werden, damit das **Millionste Byte** der Datei **“/a/big/file“** gelesen werden kann, wobei angenommen wird, dass Verzeichnisse in einen Plattenblock passen, wobei das Standarddateisystem **4 KB Blöcke** und **32-Bit Dateikopfnummern (inode numbers)** benutzt.“

*“Assume: The file buffer and the name translation caches of a **UNIX** system *V* are both completely empty. What is the **minimum number of disk blocks** that must **be read** to be able to read the **millionth byte** of the file **“/a/big/file”** assuming a directory fits into a disk block and the standard file system uses **4KB** blocks, and **32 bit inode numbers**?”*

1. read root inode
2. read root directory and find "a"
3. read inode for "/a"
4. read directory chunk and find "big"
5. read inode for "/a/big"
6. read directory chunk and find "file"
7. read inode for "/a/big/file"
8. read indirect block for the file (since 4KB blocks hold 1024 inodes, files up to 4MB are in direct or indirect block)
9. read the data block

3

2. „Zählen Sie die **Felder** eines **UNIX System V Verzeichniseintrags** auf!“  
“Enumerate the *fields* of a *UNIX System V directory entry*.”

File name, length of file , inode number,

3. „In UNIX System V gibt es **zwei verschiedene Dateitabellentypen**, deren Einträge **Deskriptoren** von geöffneten Dateien enthalten: zum einen eine **prozessspezifische**, zum anderen eine **systemweite**. Weshalb hat man beide Dateitabellentypen eingeführt?“  
“In UNIX system V there are *two different types of file-tables* whose entries contain *descriptors of opened files*: a *process-related one* and a *system-wide one*. What are the objectives of both types of file tables?”

A task related-file table enables cheap and quick forking whereby the child inherits all open files of its parent. Furthermore, if the task terminates or even if you have to abort the task, it's easy to control whether all open files have been closed. Each descriptor in this file table contains a separate file pointer, thus enabling independent navigation within each opened file.

The system-wide file-table contains the descriptor pointing to the inode of this file, and pointers to all in memory data and indirect pointer-blocks. Furthermore, this entry contains the current number of tasks that have opened this file. As long as there is still at least one task accessing this file, a system call `close()` only decrements the task counter, but doesn't close the file itself.

4. „Zählen sie mindestens **zwei Einträge** des **Superblocks** eines **physischen UNIX Dateisystems** auf.“  
“Enumerate at least *two entries* of the *super block* of a *physical UNIX file system*.”

Length of inode table, number of data blocks, size of data block, ...

### Aufgabe 3 / Question 3:

(4 + 2 + 6 Punkte/marks)

1. „Erklären Sie die **Bedeutung** und den **Zweck** der **folgenden Kontrollbits!**“  
“Explain the *semantics* and the *objectives* of the *following control bits*.”

**1.1 Valid:** This bit indicates whether a page is currently mapped to main memory or not

**1.2 Reference:** This bit indicates whether a page has been referenced or not since the last inspection by the pager.

**1.3 Dirty:** This bit indicates whether a page has been modified or not since the last inspection by the pager.

**1.4 Pin:** This bit indicates that a page is temporarily pinned to main memory, i.e. it cannot be paged out by a pager.

2. „Welche spezifischen **HW/SW-Komponenten lesen** oder **setzen** das **Pin-Bit** zu welchem **Zwecke**?“  
 “What specific **HW/SW-components read** or **modify the pin-bit**, and for what **purpose**?”

The pin-bit is only read the pager to avoid undesired swapping of that page. The pin-bit is set and reset by the pager on behalf of an application or some high-level scheduler. Each paging algorithm can only use “unpinned pages” for page replacement.

3. „In einem Seitentableneintrag ist auch ein Feld für die **physische Adresse** vorgesehen. Von **welchen Systemgrößen** ist die **Länge** dieses Feldes **abhängig**?“  
 “In an entry of a page table there is also a field containing the **physical address**. The **length** of this field **depends on what system entities**?”

Size of a page, i.e. we do not have to regard the least significant address bits.  
 Size of main memory, the larger the physical address space, the more bits are needed

4. „Beschreiben Sie, wie ein **Kontextwechsel** einer **Applikation mit nur einem Thread** das virtuelle Speichersystem betrifft. Was muss das System gewährleisten, damit die **Speicherzugriffe des neuen Threads richtig übersetzt** werden?“  
 “Describe how a **context switch** of a **single-threaded application** affects the **virtual memory system**. What must the system do to **ensure that memory references** made by the **newly running thread** will be **properly translated**.”

The operating system must locate the page table for the task that is to start running. It must set the base register in the MMU so that it points to the page table in memory, and it must set the length register if present. Finally, it must clear all now-invalid cached address translations of the old application from the TLB.

5. „Erläutern Sie den Unterschied zwischen **Residentmenge** (*resident set*) und **Arbeitsmenge** (*working set*)!“  
 “Explain the difference between **resident set** and **working set**.”

The resident set contains all currently mapped pages of an address space

The working set contains all pages of an address space which have been referenced within the past  $\tau$  time units.

**Aufgabe/Question 4****(4 + 2 + 6 Punkte/marks)**

1. „Gegeben sei die unten skizzierte Applikation für ein begrenztes Pufferproblem, bestehend aus einem Elterthread `parent` und den beiden Kinderthreads `producer` und `consumer`. Analysieren Sie die **beiden signifikantesten Probleme** dieser Programmlösung. Wie könnte man diese Probleme beheben?“

“Given the application of a bounded buffer problem (listed below) consisting of a parent thread `parent` and the two child threads `producer` and `consumer`. Analyze the **two most significant problems** of this program solution. How could you eliminate these problems?”

```
Initialization;
in:=out:=0;
buffer[N];
```

```
parent:
{
  for {i = 0; i++; i<N}
    buffer[i] = empty;
  start consumer; start producer;
}
```

```
producer:
while (true)
/* produce item v */
  while ((in+1)%N == out)
    /*do nothing */;
  b[in] = v;
  in = (in+1)%N;
}
consumer:
while (true)
{
  while (in == out)
    /*do nothing */;
  w = b[out];
  out = (out+1)%N;
  /* consume item w */
}
```

Only N-1 slots can be used  
Active waiting wastes CPU

Active waiting wastes CPU

Use a variable count for controlling the number of filled or empty buffer slots

Use semaphores instead of active waiting

2. „Weswegen **benötigt** man in **Monitoren** manchmal **Bedingungsvariablen** (*condition variables*) und mit **welchen Operationen** kann man auf diese Bedingungsvariablen zugreifen?“  
 “Why do we **need** sometimes **condition variables** within **monitors** and with **what operations** can one access these condition variables?”

A condition variable represents a certain condition (e.g. an event) that has to be met before a thread may continue to execute one of the monitor procedures. If such an event occurs we have to leave the monitor in a way that another thread can use a related monitor procedure to react on this event. The appropriate interface operations are: `CondWait(cv)` and `CondSignal(cv)`.

3. „**Skizzieren** Sie auf dem Beiblatt oder unten eine Programmschablone für eine Lösung des **begrenzten Pufferproblems mittels Monitore**. Legen Sie weniger Wert auf korrekte Syntax als vielmehr auf verständliche Semantik.“  
 “Outline below or on one of the additional draft sheets a program solution of the **bounded buffer problem via monitors**. Do not focus too much on correct syntax, but more on understandable semantics.”

```
monitor module bounded_buffer
  export fetch, deposit
  import CondSignal, CondWait
  buffer_object = record
    array buffer[1..n] of datatype
    head: integer = 1
    tail: integer = 1
    count: integer = 0
    not_full: cond = true
    not_empty: cond = false
  end
  procedure deposit(b:buffer_object, d:datatype)
  begin
    while b.count = n do CondWait(b.not_full)
      /* block thread, not monitor*/
    b.buffer[b.tail] := d
    b.tail := b.tail mod n +1
    b.count := b.count + 1
    CondSignal(b.not_empty)
  end
  procedure fetch(b:buffer_object, result:datatype)
  begin
    while b.count = 0 do CondWait(b.not_empty)
      /* block thread, not monitor*/
    result := b.buffer[b.head]
    b.head := b.head mod n +1
    b.count := b.count - 1
    CondSignal(b.not_full)
  end
end monitor modul
```

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

1. „Vergleichen und analysieren Sie die **Vor- und Nachteile** der folgenden **drei Schedulingstrategien für ein Einprozessorsystem**.“  
“Compare and analyze the **pros and cons** of the **three following scheduling policies for a single-processor system**.”

1.1 Kürzester Prozess zuerst (*Shortest Job First*):

Pros: Lowest average response-, turnaround-, waiting-time of all non preemptive scheduling policies, high throughput

Con: You need a priori knowledge or a good estimation of the pure execution times, Long runners may starve

1.2 Striktes statisches Prioritätsschema ohne Verdrängung (*Strict Static Priority Scheme without Preemption*):

Pros: User can express a preference of his application threads, simply to implement

Con: Possible starvation of threads with low priority, Without preemption still a danger of priority inversion

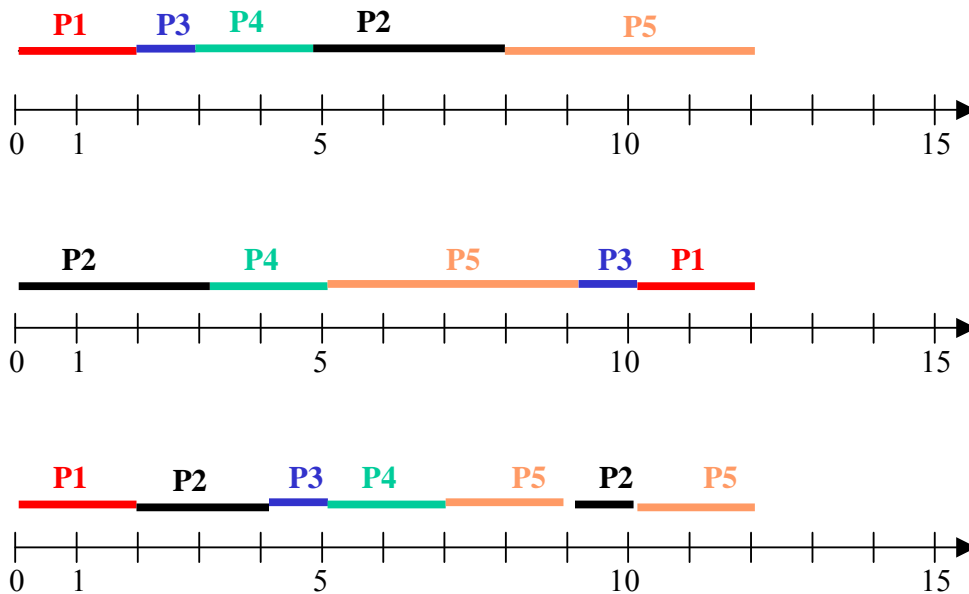
1.3 Round-Robin mit systemweiter Zeitscheibe (*Round robin with system wide time slice*):

Pros: Absolute fair policy,

Con: Hard to find the proper size of a time slice, either too small or too large, If too small, then too much overhead, if too large, then as FCFS

2. „Nehmen Sie an, dass auf einem Einprozessorsystem die folgenden fünf Prozesse zu den angegebenen Zeiten aktiviert werden (**Ankunftszeit** = *arrival time*). Deren **reine Ausführungszeit** (*pure execution time*) und deren **Priorität** seien ebenfalls vorab bekannt, wobei der Prioritätswert 0 die niedrigste Priorität bedeutet. Berechnen sie die **mittlere Verweilzeit** (*average turn around time*) dieser 5 Prozesse für die drei Schedulingstrategien: **Kürzester Prozeß zuerst** (shortest job first), **striktes Prioritätsschema ohne Verdrängung** und **Round-Robin mit Zeitscheibe 2 Zeiteinheiten**.“
- “Assume the following processes will be activated on a single-processor system at the below mentioned *arrival times*. The *pure execution time* and the *priority value* of each process are also known in advance, where the priority value 0 means lowest priority. Calculate the *average turn around time* of these 5 processes if the following scheduling algorithms are used: *shortest job first*, *strict priority without preemption*, and *round robin with time slice value 2*.”

Prozess (process)	Ankunftszeit (arrival time)	Reine Ausführungszeit (pure execution time)	Priorität (priority)
1	0	2	0
2	0	3	1
3	1	1	2
4	3	2	3
5	4	4	4



Average Turnaround Time (ShortestJobfirst) = 4.4

Average Turnaround Time (Strict Priority) = 6.2

Average Turnaround Time (Round Robin with Time Slice 2) = 5.6