SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Jan Reineke
Dr. Andreas Abel
Lucas Biehl

Systemarchitektur SS 2021

Präsenzblatt 11 (Lösungsvorschläge)

Hinweis: **Dieses Aufgabenblatt wurde von Tutoren erstellt. Die Aufgaben sind für die Klausur weder relevant noch irrelevant, die Lösungsvorschlage weder korrekt noch inkorrekt.**

Aufgabe 11.1: Speichervirtualisierung

1. Wir haben in der Vorlesung verschiedene Ansätze kennengelernt, mit denen Speichervirtualisierung realisiert werden kann. Erläutern Sie jeweils kurz die Grundgedanken der folgenden Techniken und nennen Sie jeweils Vor- und Nachteile: Zeitmultiplexen, statische Relokation, dynamische Relokation, Segmentierung, Paging (diskutieren Sie dabei auch die verschiedenen Erweiterungen).

2. Betrachten Sie die folgende Codezeile aus einem C-Programm:

   ```
   a = a + 3;
   ```

   Dabei ist `a` eine globale Variable, die im (virtuellen) Speicher an Adresse `0x21c` steht und dort auch wieder hingeschrieben werden soll.

   Übersetzen Sie den Programmausschnitt in MIPS-Assemblercode. Der Code-Teil des Adressraums beginnt bei `0x000` und der Daten-Teil bei `0x200`. Beim Übersetzen beginnt der Code für die gegebene Zeile an der (virtuellen) Adresse `0x100`. Wie viele Speicherzugriffe sind nötig?

3. Unser (kleines) System hat 10-Bit virtuelle und 12-Bit physische Adressen, wobei jeder Speichereintrag ein Byte groß ist. Wir möchten diese Codezeile nun von zwei Prozessen ausführen lassen. Aufgrund von CPU-Scheduling wird der erste Prozess nach dem Ladebefehl unterbrochen und erst wieder fortgesetzt, wenn der zweite Prozess die Codezeile komplett ausgeführt hat.

   Geben Sie die zeitliche Abfolge aller Speicherzugriffe der zwei Prozesse für obige Codezeile an, wenn das System mit den folgenden Virtualisierungstechniken arbeitet:

   a) **Zeitmultiplexen:** Dabei wird nur der virtuelle Adressraum benutzt.

   b) **Statische bzw. dynamische Relokation:** Der Adressraum für Prozess 1 beginnt bei `0x000` und für Prozess 2 bei `0x400`. Geben Sie für statische Relokation den umgeschriebenen Assemblercode und für dynamische Relokation jeweils den Inhalt der MMU an.

   c) **Segmentierung:** Das Codesegment für Prozess 1 beginnt bei `0x000` und für Prozess 2 bei `0x800`, das Datensegment jeweils bei `0x200` bzw. `0xa00`.

   d) **Paging ohne TLBs:** Wir nutzen die letzten 8 Bits als Versatz. Die Seitentabellen der zwei Prozesse liegen hintereinander ab Adresse `0x300`, dabei enthält die Seitentabelle von Prozess 1 hintereinander die Werte `0xf`, `0x7`, `0x2` und `0x5`, während die Seitentabelle von Prozess 2 die Werte `0x0`, `0x4`, `0x8` und `0xa` enthält.

   Hinweise: Wie viele Einträge hat die Seitentabelle eines Prozesses? Wie groß ist jeder Seitentabelleneintrag? Können Sie Einträge dieser Größe gut im Speicher adressieren? Wie groß werden Sie jede Seitentabelle machen?

   e) **Paging mit TLBs:** Betrachten Sie einen initial leeren TLB der Größe 4. Nutzen Sie ansonsten die gleichen Voraussetzungen wie beim Paging ohne TLBs. Welchen Unterschied gibt es zwischen TLBs ohne und mit ASID?

	Beschreibung	Vorteile	Nachteile
Zeitmultiplexen	jedem Prozess wird kompletter Speicher zugewiesen	keine zusätzliche Logik nötig	Prozesswechsel viel zu teuer und damit unpraktikabel
statische Relokation	Statische Modifikation der Adressen vor Start des Prozesses	Speicher geteilt statt kopiert	Keine Isolation; Fragmentierung möglich
dynamische Relokation	dynamische Modifikation der Adressen zur Laufzeit durch MMU	Isolation durch Schranke in MMU, einfach zu realisieren	interne + externe Fragmentierung, kein gemeinsamer Speicher
Segmentierung	Adressraum wird in logische Segmente unterteilt	keine interne Fragmentierung, unterschiedlicher Schutz für Segmente	externe Fragmentierung
Paging	Adressraum in voneinander unabhängige, gleich große Seiten aufgeteilt	keine externe Fragmentierung, Auslagerung möglich	interne Fragmentierung, zusätzlicher Speicherzugriff (Lösung: TLBs), hoher Platzbedarf (Lösung: Multi-Level-Seitentabellen)

Lösungsvorschlag:

1. Übersicht über die verschiedenen Speichervirtualisierungstechniken:

2. Die Codezeile lässt sich zu folgendem MIPS-Assemblercode übersetzen:

```
0x100: lw $1, 0x21c($0)
0x104: addiu $1, $1, 3
0x108: sw $1, 0x21c($0)
```

   Es sind dabei zwei Speicherzugriffe nötig: ein lesender und ein schreibender Zugriff.

3. a) Mit Zeitmultiplexen:

Prozess 1	Prozess 2	Kommentar
0x100	-	
0x21c	-	
-	0x100	
-	0x21c	
-	0x104	
-	0x108	
-	0x21c	P2 hat Codezeile ausgeführt
0x104	-	
0x108	-	
0x21c	-	P1 hat Codezeile ausgeführt

   b) Bei statischer bzw. dynamischer Relokation:

Prozess 1	Prozess 2	Kommentar
0x100	-	
0x21c	-	
-	0x500	
-	0x61c	
-	0x504	
-	0x508	
-	0x61c	P2 hat Codezeile ausgeführt
0x104	-	
0x108	-	
0x21c	-	P1 hat Codezeile ausgeführt

Umgeschriebener Assemblercode für P2 bei statischer Relokation:

```
0x500: lw $1, 0x61c($0)
0x504: addiu $1, $1, 3
0x508: sw $1, 0x61c($0)
```

Inhalt der MMU für P1 (P2) bei dynamischer Relokation:
Basis: `0x000` (`0x400`)
Schranke: `0x400` (`0x400`)
Modus: `1` (entspricht user)

   c) Mit Segmentierung:

Prozess 1	Prozess 2	Kommentar
0x100	-	
0x21c	-	
-	0x900	
-	0xa1c	
-	0x904	
-	0x908	
-	0xa1c	P2 hat Codezeile ausgeführt
0x104	-	
0x108	-	
0x21c	-	P1 hat Codezeile ausgeführt

d) Bei (einfachem) Paging ohne TLBs:

Prozess 1	Prozess 2	Kommentar
0x301	-	Page 1 liegt an `0x700`
0x700	-	
0x302	-	Page 2 liegt an `0x200`
0x21c	-	
-	0x305	Page 1 liegt an `0x400`
-	0x400	
-	0x306	Page 2 liegt an `0x800`
-	0x81c	
-	0x305	Page 1 liegt an `0x400`
-	0x404	
-	0x305	Page 1 liegt an `0x400`
-	0x408	
-	0x306	Page 2 liegt an `0x800`
-	0x81c	P2 hat Codezeile ausgeführt
0x301	-	Page 1 liegt an `0x700`
0x704	-	
0x301	-	Page 1 liegt an `0x700`
0x708	-	
0x302	-	Page 2 liegt an `0x200`
0x21c	-	P1 hat Codezeile ausgeführt

Jede Seitentabelle hat 4 Einträge, da die zwei höchstwertigen Bits einer virtuellen Adresse den Eintrag in der Tabelle angeben.
Jeder Seitentabelleneintrag ist $4bit$ groß, da wir 16 verschiedene physische Rahmen haben. Da wir aber nur byte-weise aus dem Speicher lesen können, spendieren wir jedem Eintrag doppelt so viele Bits.
Jede Seitentabelle ist damit $4 \cdot 8bit = 32bit = 4byte$ groß.

e) Bei Paging mit TLBs (mit ASID würden die mit (*) markierten Speicherzugriffe entfallen):

Prozess 1	Prozess 2	Kommentar
0x301	-	Page 1 liegt an `0x700`
0x700	-	
0x302	-	Page 2 liegt an `0x200`
0x21c	-	
-	0x305	Page 1 liegt an `0x400`
-	0x400	
-	0x306	Page 2 liegt an `0x800`
-	0x81c	
-	0x404	TLB-Hit
-	0x408	TLB-Hit
-	0x81c	TLB-Hit, P2 hat Codezeile ausgeführt
0x301	-	Page 1 liegt an `0x700` (*)
0x704	-	
0x708	-	TLB-Hit
0x302	-	Page 2 liegt an `0x200` (*)
0x21c	-	TLB-Hit, P1 hat Codezeile ausgeführt

Aufgabe 11.2: Speichervirtualisierung Quiz

Schauen Sie sich die folgenden Aussagen an und überlegen Sie sich, ob sie jeweils stimmen oder nicht. Geben Sie für falsche Aussagen an, warum diese inkorrekt sind.

1. Zeitmultiplexen bedeutet, dass jeder Prozess eine private CPU hat.
2. Ein wichtiger Vorteil der dynamischen Relokation ist, dass Prozesse isoliert werden.
3. Der Kernelmodus kann auf mehr Speicher zugreifen als der Benutzermodus.
4. Segmentierung ist wie dynamische Relokation, nur sicherer durch den Kernelmodus.
5. Bei Segmentierung gibt es interne Fragmentierung.
6. TLBs sind Caches, die für eine virtuelle Adresse den jeweiligen Datenpunkt ausgeben.

Lösungsvorschlag:

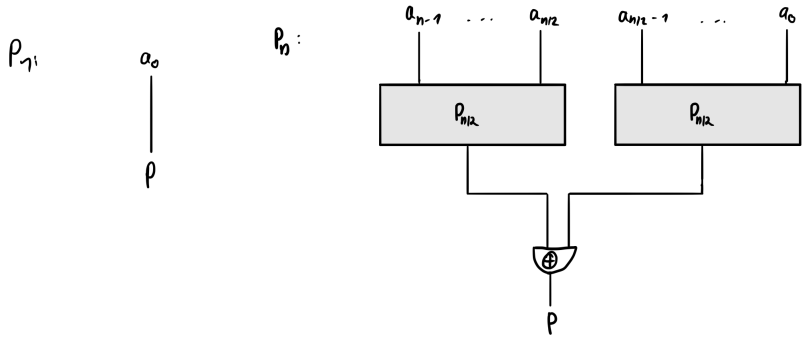
1. Falsch, es wird nur die Illusion erzeugt.
2. Richtig, die Adressräume werden getrennt.
3. Richtig, die Benutzerprozesse können nur auf ihren jeweiligen Speicher zugreifen.
4. Falsch, der Hauptvorteil, ist, dass Speicher optimiert werden kann, weil nicht jeder Prozess große Zusammenhängende Stücke Speicher benötigt. Der Kernelmodus hat damit nichts zu tun.
5. Falsch, der Adressraum kann spärlich allokiert werden, Stack und Heap wachsen unabhängig.
6. Falsch, TLBs geben die physische Adresse zurück.

Aufgabe 11.3: Wiederholung: Schaltkreise

1. Konstruieren Sie einen rekursiven Schaltkreis P_n mit logarithmischer Tiefe, der die Funktion $p : \mathbb{B}^n \to \mathbb{B}, x \mapsto (\sum_{i=1}^{n} x_i) \mod 2$ berechnet.
2. Bestimmen Sie Funktionen $C(n)$ und $d(n)$ in rekursiver Darstellung für Kosten und Tiefe des Schaltkreises.
3. Bestimmen Sie eine Darstellung $C'(n)$ für $C(n)$ in geschlossener Form und beweisen Sie die Korrektheit dieser Darstellung.

Lösungsvorschlag:

1.



2.

$$C(1) = 0 \qquad C(n) = 2C(\frac{n}{2}) + 1 \quad n = 2^k > 1$$

$$d(1) = 0 \qquad d(n) = 1 + d(\frac{n}{2}) \quad n = 2^k > 1$$

3. $C'(n) = n - 1$ für $n = 2^k > 0$
   Beweis durch Induktion über $k \in \mathbb{N}_{\geq 0} : \forall k \in \mathbb{N}_{\geq 0} : C(2^k) = C'(2^k)$.

   IA: Sei $k = 0$. Dann ist $C(2^0) = C(1) \overset{Def\ C}{=} 0 = 1 - 1 \overset{Def\ C'}{=} C'(2^0)$.

   IH: Sei die Aussage für ein beliebiges aber festes $k \in \mathbb{N}_{>0}$ bereits bewiesen, d.h. $C'(2^k) = C(2^k)$.

   IS: Es gilt:

$$
\begin{aligned}
C(2^{k+1}) &= 2C(2^k) + 1 && \text{Def. C} \\
&= 2C'(2^k) + 1 && \text{IH} \\
&= 2(2^k - 1) + 1 && \text{Def. C'} \\
&= 2^{k+1} - 2 + 1 \\
&= 2^{k+1} - 1 \\
&= C'(2^{k+1}) && \text{Def. C'}
\end{aligned}
$$

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Jan Reineke
Dr. Andreas Abel
Lucas Biehl

System Architecture SS 2021

Tutorial Sheet 11 (Suggested Solutions)

Note: **This task sheet was created by tutors. The tasks are neither relevant nor irrelevant for the exam, the suggested solutions are neither correct nor incorrect.**

Problem 11.1: Memory Virtualization

1. In the lecture, we learned about different approaches with which memory virtualization can be realized. Briefly explain the basic ideas of each of the following techniques and name the advantages and disadvantages of each: time sharing, static relocation, dynamic relocation, segmentation, paging (also discuss possible extensions).

2. Consider the following line of code from a *C* program:

   ```
   a = a + 3;
   ```

   Where `a` is a global variable, which is stored in (virtual) memory at address `0x21c` and is to be written there again.

   Translate the program section into MIPS assembler code. The code part of the address space starts at `0x000` and the data part at `0x200`. When compiling, the code for the given line starts at the (virtual) address `0x100`. How many memory accesses are needed?

3. Our (small) system has 10-bit virtual and 12-bit physical addresses, where each memory entry is one byte in size. We now want this line of code to be executed by two processes. Due to CPU scheduling the first process is interrupted after the load command and will not resume until the second process has completely executed the line of code.

   Specify the time sequence of all memory accesses of the two processes for the above line of code when the system is using the following virtualization techniques:

   a) **Time sharing:** Only the virtual address space is used.

   b) **Static or dynamic relocation:** The address space for process 1 starts at `0x000` and for process 2 at `0x400`. For static relocation give the rewritten assembler code and for dynamic relocation the content of the MMU.

   c) **Segmentation:** The code segment for process 1 starts at `0x000` and for process 2 at `0x800`, the data segment respectively at `0x200` resp. `0xa00`.

   d) **Paging without TLBs:** We use the last 8 bits as offsets. The page tables of the two processes are one after the other starting at address `0x300`, where the page table of process 1 contains sequentially the values `0xf`, `0x7`, `0x2` and `0x5`, while the page table of process 2 contains the values `0x0`, `0x4`, `0x8` and `0xa`.

      Hint: How many entries does the page table of a process have? What is the size of each page table entry? Can you address entries of this size well in memory? How large will you make each page table?

   e) **Paging with TLBs:** Consider an initially empty TLB of size 4. Otherwise, use the same assumptions as for paging without TLBs. What is the difference between TLBs without and with ASID?

	description	advantages	disadvantages
Time sharing	each process is allocated the complete memory	no additional logic needed	process change way too expensive and therefore impracticable
Static relocation	static modification of addresses before starting the process	memory shared instead of copied	no isolation; fragmentation possible
Dynamic relocation	dynamic modification of addresses at runtime by MMU	isolation by barrier in MMU, easy to realize	internal + external fragmentation, no shared memory
Segmentation	address space is divided into logical segments	no internal fragmentation, different protection for segments	external fragmentation
Paging	Address space divided into independent pages of equal size.	no external fragmentation, swapping possible	internal fragmentation, additional memory access (solution: TLBs), high space requirements (solution: multi-level page tables)

Suggested solution:

1. Overview of the different storage virtualization techniques:

2. The code line translates to the following MIPS assembly code:

```
0x100: lw $1, 0x21c($0)
0x104: addiu $1, $1, 3
0x108: sw $1, 0x21c($0)
```

   Two memory accesses are necessary: one read and one write access.

3. a) With time sharing:

Process 1	Process 2	Comment
0x100	-	
0x21c	-	
-	0x100	
-	0x21c	
-	0x104	
-	0x108	
-	0x21c	P2 has executed code line
0x104	-	
0x108	-	
0x21c	-	P1 has executed code line

   b) With static or dynamic relocation:

Process 1	Process 2	Comment
0x100	-	
0x21c	-	
-	0x500	
-	0x61c	
-	0x504	
-	0x508	
-	0x61c	P2 has executed code line
0x104	-	
0x108	-	
0x21c	-	P1 has executed code line

Rewritten assembly code for P2 on static relocation:

```
0x500: lw $1, 0x61c($0)
0x504: addiu $1, $1, 3
0x508: sw $1, 0x61c($0)
```

Contents of the MMU for P1 (P2) with dynamic relocation:
Base: `0x000` (`0x400`)
Bounds: `0x400` (`0x400`)
Mode: `1` (equivalent to user)

   c) With segmentation:

Process 1	Process 2	Comment
0x100	-	
0x21c	-	
-	0x900	
-	0xa1c	
-	0x904	
-	0x908	
-	0xa1c	P2 has executed code line
0x104	-	
0x108	-	
0x21c	-	P1 executed code line

d) With (simple) paging without TLBs:

Process 1	Process 2	Comment
0x301	-	Page 1 is at `0x700`
0x700	-	
0x302	-	Page 2 is at `0x200`
0x21c	-	
-	0x305	Page 1 is at `0x400`
-	0x400	
-	0x306	Page 2 is at `0x800`
-	0x81c	
-	0x305	Page 1 is at `0x400`
-	0x404	
-	0x305	Page 1 is at `0x400`
-	0x408	
-	0x306	Page 2 is at `0x800`
-	0x81c	P2 has executed code line
0x301	-	Page 1 is at `0x700`
0x704	-	
0x301	-	Page 1 is at `0x700`
0x708	-	
0x302	-	Page 2 is at `0x200`
0x21c	-	P1 has executed code line

Each page table has 4 entries, since the two most significant bits of a virtual address specify the entry in the table.
Each page table entry is $4bit$ in size, since we have 16 different physical frames. However, since we can only read from memory one byte at a time, we give each entry twice as many bits.
Each page table is thus $4 \cdot 8bit = 32bit = 4byte$ in size.

e) With paging with TLBs (with ASID memory accesses marked with (*) are not necessary):

Process 1	Process 2	Commentar
0x301	-	Page 1 is at `0x700`
0x700	-	
0x302	-	Page 2 is at `0x200`
0x21c	-	
-	0x305	Page 1 is at `0x400`
-	0x400	
-	0x306	Page 2 is at `0x800`
-	0x81c	
-	0x404	TLB-Hit
-	0x408	TLB-Hit
-	0x81c	TLB-Hit, P2 has executed code line
0x301	-	Page 1 is at `0x700` (*)
0x704	-	
0x708	-	TLB-Hit
0x302	-	Page 2 is at `0x200` (*)
0x21c	-	TLB-Hit, P1 has executed code line

Problem 11.2: Memory Virtualization Quiz

Look at the following statements and consider whether or not each is true. For wrong statements, state why it is incorrect.

1. Time sharing means that each process has a private CPU.
2. An important advantage of dynamic relocation is that processes are isolated.
3. Kernel mode can access more memory than user mode.
4. Segmentation is like dynamic relocation, only more secure through kernel mode.
5. With segmentation there is internal fragmentation.
6. TLBs are caches which output the respective data point for a virtual address.

Suggested solution:

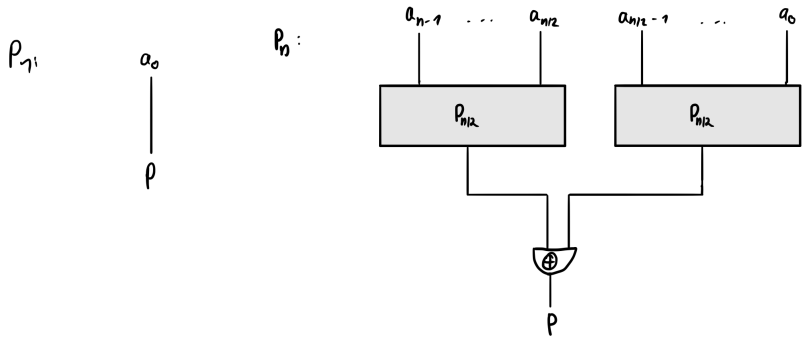
1. False, only the illusion is created.
2. Correct, the address spaces are separated.
3. True, the user processes can only access their respective memory.
4. Wrong, the main advantage is that memory can be optimized because not every process needs large chunks of memory. Kernel mode has nothing to do with this.
5. Wrong, the address space can be allocated spatially, stack and heap grow independently.
6. False, TLBs return the physical address.

Problem 11.3: Recap: Circuits

1. Construct a recursive circuit P_n with logarithmic depth that calculates the function $p : \mathbb{B}^n \to \mathbb{B}, x \mapsto (\sum_{i=1}^{n} x_i) \mod 2$.
2. Determine functions $C(n)$ and $d(n)$ in recursive representation for cost and depth of the circuit.
3. Determine a representation $C'(n)$ for $C(n)$ in closed form and prove the correctness of this representation.

Suggested solution:

1.



2.

$$C(1) = 0 \qquad C(n) = 2C(\frac{n}{2}) + 1 \quad n = 2^k > 1$$

$$d(1) = 0 \qquad d(n) = 1 + d(\frac{n}{2}) \quad n = 2^k > 1$$

3. $C'(n) = n - 1$ for $n = 2^k > 0$
   Proof by induction over $k \in \mathbb{N}_{\geq 0} : \forall k \in \mathbb{N}_{\geq 0} : C(2^k) = C'(2^k)$.

   BC: Let $k = 0$. Then $C(2^0) = C(1) \overset{Def\ C}{=} 0 = 1 - 1 \overset{Def\ C'}{=} C'(2^0)$.

   IH: Let the statement for any but fixed $k \in \mathbb{N}_{>0}$ already be proved, i.e. $C'(2^k) = C(2^k)$.

   IS: It holds:

$$\begin{aligned}
C(2^{k+1}) &= 2C(2^k) + 1 && \text{Def. C} \\
&= 2C'(2^k) + 1 && \text{IH} \\
&= 2(2^k - 1) + 1 && \text{Def. C'} \\
&= 2^{k+1} - 2 + 1 \\
&= 2^{k+1} - 1 \\
&= C'(2^{k+1}) && \text{Def. C'}
\end{aligned}$$