SAARLAND UNIVERSITY
COMPUTER SCIENCE

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

Systemarchitektur SS 2021

Lösungsskizze 12

Aufgabe 12.1: Scheduling

(a) Übersicht der Scheduling Planungsverfahren:

	Beschreibung	Vorteile	Nachteile
FIFO/ FCFS	Prozesse kriegen Betriebsmittel nacheinander zugeteilt. Wenn zwei zeitgleich $\rightarrow$ lexikographisch.	einfach	keinerlei Berücksichtigung anderer Faktoren
SJF	Prozesse nach steigender Ausführungszeit geordnet. (bei Gleichheit egal)	optimal bzgl. ø–Umlaufzeit (wenn Jobs glztg. ankommen)	längere Prozesse müssen ggf. lange warten
STCF	Wähle Prozess mit niedrgst. Restlaufzeit. Wenn neuer Prozess eintritt ggf. laufenden unterbrechen	optm. bzgl. ø–Umlaufzeit	Verhungern (Starvation) sehr langer Jobs
Round Robin	Führe Prozesse abwechselnd für eine Zeitscheibe fester Länge aus	fair kein Verhungern gut bzgl. Antwortzeit	Prozesswechsel kosten Zeit schlecht bzgl. Umlaufzeit unflexible wg. fester Zeitscheibenlänge

(b) Die Ausführungen sind in Abbildung 2 dargestellt. Zu beachten ist, dass es sich dabei nur jeweils um eine mögliche Ausführung handelt, andere Ausführungen mit anderen Werten für Umlauf- und Antwortzeit sind möglich.

Übersicht über die sich ergebenden Umlauf– und Antwortzeiten:

Verfahren	ø–Umlaufzeit	ø–Antwortzeit
FIFO	$\frac{6+13+(16-2)+(19-5)}{4} = 11.75$	$\frac{0+6+(13-2)+(16-5)}{4} = 7$
SJF	$\frac{6+(9-2)+(12-5)+19}{4} = 9.75$	$\frac{0+(6-2)+(9-5)+12}{4} = 5$
STCF	$\frac{12+(5-2)+(8-5)+19}{4} = 9.25$	$\frac{0+(2-2)+(5-5)+12}{4} = 3$
Round Robin (Länge 1)	$\frac{17+(11-2)+(14-5)+19}{4} = 13.5$	$\frac{0+(2-2)+(5-5)+1}{4} = 0.25$

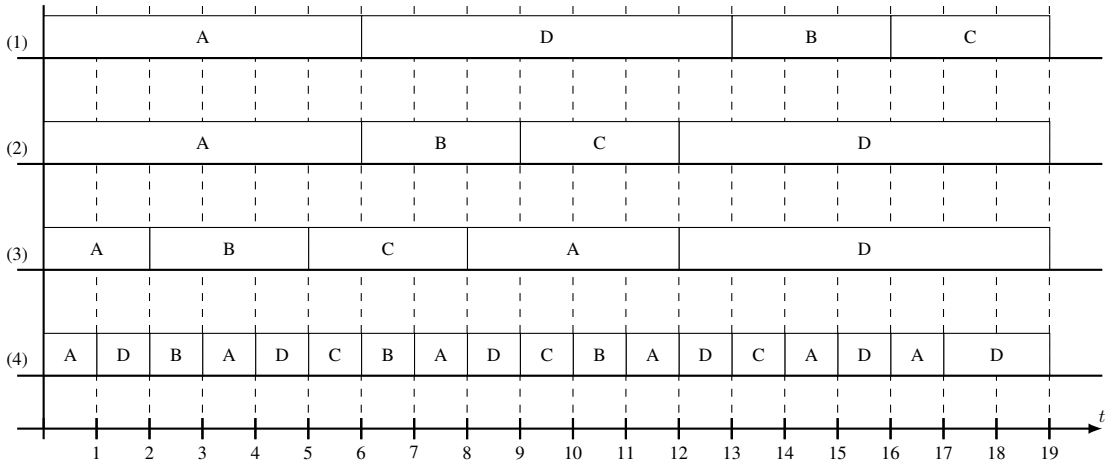

Abbildung 1: Ausführungen mit den verschiedenen Planungsverfahren (1–4 entsprechen FIFO, SJF, STCF und Round Robin in dieser Reihenfolge)

Aufgabe 12.2: Scheduling (2)

(a) Wir wollen durch Widerspruch zeigen, dass SJF optimal bzgl. ø–Umlaufzeit ist. Dazu nehmen wir an es gebe einen Scheduling-Algorithmus S $\neq$ SJF der optimal bezüglich ø–Umlaufzeit ist.

Zunächst beobachten wir, dass S Prozessausführungen nicht unterbricht (im Gegensatz zu beispielsweise STCF), da die ø–Umlaufzeit bei *gleichzeitig ankommenden* Prozessen nicht mehr optimal sein kann. Das liegt daran, dass sich die Terminierungen der Prozesse durch Unterbrechungen unnötig verzögern und dadurch die ø–Umlaufzeit erhöht wird. (Bedenke: Umlaufzeit ist die Differenz aus Ankunftszeit und Terminierungszeitpunkt.)

Nun, da wir ausschließen können, dass das verwendete Planungsverfahren Prozesse unterbricht, können wir den restlichen Beweis leicht führen: Da S $\neq$ SJF muss es 2 Prozesse A und B geben sodass $|A| < |B|$ aber B vor A ausgeführt wird. Dieser Schedule sieht also folgendermaßen aus:

$$\underbrace{\ldots}_{x} B \underbrace{\ldots}_{y} A \underbrace{\ldots}_{z} \tag{1}$$

Um den Widerspruch herbeizuführen beweisen wir jetzt dass Schedule 2 (in dem A und B vertauscht sind) eine geringere ø–Umlaufzeit hat - damit ist gezeigt dass S nicht optimal ist.

$$\underbrace{\ldots}_{x} A \underbrace{\ldots}_{y} B \underbrace{\ldots}_{z} \tag{2}$$

Da per Konstruktion x,y und z identisch sind unterscheidet sich ihr Beitrag zur Umlaufzeit nicht zwischen den beiden Schedules. Es reicht also aus zu zeigen, dass

$$Umlaufzeit_1(A) + Umlaufzeit_1(B) > Umlaufzeit_2(A) + Umlaufzeit_2(B)$$

$$\begin{aligned} Umlaufzeit_1(A) + Umlaufzeit_1(B) &= (x + |B| + y + |A|) + (x + |B|) \\ &= 2x + 2|B| + y + |A| \\ &\overset{|A|<|B|}{>} 2x + |B| + y + 2|A| \\ &= (x + |A|) + (x + |A| + y + |B|) \\ &= Umlaufzeit_2(A) + Umlaufzeit_2(B) \end{aligned}$$

■

(b) Gegenbeispiel:

Job	Ankunftszeit	Ausführungszeit
A	0	10
B	1	2
C	2	3

In diesem Fall ergibt sich für die ø–Umlaufzeit mit SJF–Verfahren:

$$\frac{(10-0)+(12-1)+(15-2)}{3} = \frac{34}{3} \approx 11,33$$

Für das STCF–Verfahren ergibt sich jedoch:

$$\frac{(15-0)+(3-1)+(6-2)}{3} = \frac{21}{3} = 7$$

Demnach ist SJF hier nicht optimal. Dieser Effekt findet sich im Foliensatz zu Scheduling auf Folie 10 unter dem Begriff "Konvoieffekt" und ist dadurch bedingt, dass die wesentlich kürzeren Jobs kurz nach dem deutlich längeren Job A ankommen, SJF jedoch im Gegensatz zu STCF Prozesse nicht unterbricht.

Aufgabe 12.3: Multi-Level Feedback Queue (MLFQ)

(a) Regeln für eine MLFQ:

Regel 1: *Priorität*$(A) >$ *Priorität*$(B) \rightarrow$ *führe* A *aus.*

Ermöglicht es, dass Prozesse aufgrund ihrer Prioritäten unterschiedlich behandelt werden. Durch das Setzen der Prioritäten kann entschieden werden, welcher Prozess ausgeführt werden soll.

Regel 2: *Priorität*$(A) =$ *Priorität*$(B) \rightarrow$ *Round Robin zwischen* A *und* B.

Prozesse mit gleicher Priorität werden gleichermaßen bearbeitet, verhindert "Verhungern" von Prozessen bei gleicher Priorität.

Regel 3: *Neue Jobs erhalten höchste Priorität.*

Neue Jobs werden zuerst ausgeführt, damit sie, falls sie nur wenig Zeit beanspruchen, schnell abgearbeitet werden können. Lange Jobs werden so unterbrochen, eine Art STCF wird umgesetzt. So wird eine kurze Antwortzeit garantiert.

Regel 4: *Senke die Priorität eines Jobs, wenn er sein Budget auf einem Prioritätslevel erschöpft hat.*

Falls ein Job zu lange braucht, muss er nach kurzer Zeit warten. Dies realisiert den Teil von STCF, der garantiert, dass nur kurze Jobs einen langen Prozess dauerhaft unterbrechen können. Der Scheduler "lernt", ob es sich um einen kurzen Job handelt.

Regel 5: *Setze die Priorität eines Jobs nach* S *Zeiteinheiten zurück auf die höchste Stufe.*

Sorgt dafür, dass lange Jobs nicht bei niedriger Prioriät verhungern, wenn immer wieder neue, kürzere Jobs auftauchen.

(b) Die MLFQ degeneriert zu einem Round Robin-Scheduler, wenn immer alle Jobs die gleiche Priorität haben. Dies lässt sich dadurch erreichen, dass immer nur genau eine Prioritätsebene in der MLFQ genutzt wird, beispielsweise indem festgelegt wird, dass es nur eine Priorität gibt, oder indem Prioritäten niemals abgesenkt oder direkt wieder erhöht werden, sobald sie abgesenkt werden.

Aufgabe 12.4: Segmentierung

Die Lösung wird in `http://pages.cs.wisc.edu/~remzi/OSTEP/vm-segmentation.pdf`, Kapitel 16.3, genauer beschrieben. Die Idee ist hier, den MMU-Zustand um ein Bit zu erweitern, das kennzeichnet in welche Richtung sich das Segment von der Basisadresse aus erstreckt. Abhängig von diesem Bit muss bei der Adressübersetzung der Offset von der maximalen Segmentgröße abgezogen werden bevor das (dann negative) Ergebnis auf die Basisadresse addiert wird. Ebenso muss abhängig von dem Bit der Bounds-Check angepasst werden um den Absolutwert des (negativen) Ergebnisses mit der Segmentgröße zu vergleichen.

Aufgabe 12.5: Speichervirtualisierung

(a) **Zeitmultiplexen:** Der Prozesskontrollblock enthält eine Festplattenadresse. Beim Prozesswechsel wird der Hauptspeicher auf die Festplatte geschrieben und die Daten an der Festplattenadresse des neuen Prozesses in den Hauptspeicher geladen. Aus Optimierungsgründen möchte man vielleicht den genutzten Speicher im PCB vermerken, damit keine ungenutzten Speicherbereiche gesichert werden.

statische Relokation: Der Prozesskontrollblock enthält die Basisadresse und die Speichermenge die der Prozess benötigt. Beim Prozesswechsel muss nichts besonderes beachtet werden. Ausschließlich beim Starten eines Prozesses muss man darauf achten die verwendeten Addressen auf eine freie Basisadresse umzuschreiben (die Information in den PCBs benötigt man um einen freien Addressbereich auszuwählen).

dynamische Relokation: Wie statische Relokation. In diesem Fall müssen allerdings bei jedem Prozesswechsel die Basisadresse und eventuell die Größe in die entsprechenden MMU-Register geschrieben werden. Dafür ist keine Sonderbehandlung beim Prozesstart mehr nötig.

Segmentierung: Der PCB enthält Basis und Grenze für jedes Segment. Beim Prozesswechsel werden diese in die passenden MMU-Register geschrieben.

Paging: Der PCB enthält die Adresse der Page Table des Prozesses. Beim Prozesswechsel wird diese in das passende MMU-Register geladen.

Paging mit TLB: Wie Paging, allerdings muss man beim Prozesswechsel nun auch den TLB leeren (*flushen*). Systeme die den TLB in Software befüllen (z.B. MIPS) müssen die Adresse der Page Table der MMU nicht mitteilen.

(b) Segmentierung: Das Codesegment von Prozess 1 startet bei 0, das Codesegment von Prozess 2 startet bei 0x2000. Das Datensegment von Prozess 1 startet bei 0x3000, das von Prozess 2 bei 0x1000.

Prozess 1	Prozess 2	Kommentar
0x0100	-	
0x0104	-	
0x3234	-	
-	0x2100	
-	0x2104	
-	0x1234	
-	0x2108	
-	0x210c	
-	0x1234	Prozess 2 terminiert
0x0108	-	
0x010c	-	
0x3234	-	

(c) Paging: Die Pagetables von Prozess 1 und 2 liegen bei 0 und 0x0800 (jeweils 4 Einträge). Die Seitengröße ist 4 kB.

Prozess 1	Prozess 2	Kommentar
0x0000	-	Page 0 liegt an 0x9000
0x9100	-	
0x0000	-	Page 0 liegt an 0x9000
0x9104	-	
0x0004	-	Page 1 liegt an 0x7000
0x7234	-	
-	0x0800	Page 0 liegt an 0x3000
-	0x3100	
-	0x0800	Page 0 liegt an 0x3000
-	0x3104	
-	0x0804	Page 1 liegt an 0x4000
-	0x4234	
-	0x0800	Page 0 liegt an 0x3000
-	0x3108	
-	0x0800	Page 0 liegt an 0x3000
-	0x310c	
-	0x0804	Page 1 liegt an 0x4000
-	0x4234	Prozess 2 terminiert
0x0000	-	Page 0 liegt an 0x9000
0x9108	-	
0x0000	-	Page 0 liegt an 0x9000
0x910c	-	
0x0004	-	Page 1 liegt an 0x7000
0x7234	-	

(d) Paging mit TLB: Die Pagetables von Prozess 1 und 2 liegen bei 0 und 0x0800 (jeweils 4 Einträge). Die Seitengröße ist 4 kB. Wir nehemen einen TLB der Größe 4 an, der initial leer ist.

Prozess 1	Prozess 2	Kommentar
0x0000	-	Page 0 liegt an 0x9000
0x9100	-	
0x9104	-	Übersetzung liegt im TLB vor
0x0004	-	Page 1 liegt an 0x7000
0x7234	-	
-	0x0800	Page 0 liegt an 0x3000
-	0x3100	
-	0x3104	Übersetzung liegt im TLB vor
-	0x0804	Page 1 liegt an 0x4000
-	0x4234	
-	0x3108	Übersetzung liegt im TLB vor
-	0x310c	Übersetzung liegt im TLB vor
-	0x4234	Prozess 2 terminiert
0x0000	-	Page 0 liegt an 0x9000
0x9108	-	
0x910c	-	Übersetzung liegt im TLB vor
0x0004	-	Page 1 liegt an 0x7000
0x7234	-	

Unterstützt der TLB *Address Space Identifiers* hätten die beiden letzten Zugriffe auf die Pagetable eingespart werden können.

SAARLAND UNIVERSITY
COMPUTER SCIENCE

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

System Architecture SS 2021

Solution Sketch 12

Problem 12.1: Scheduling

(a) Overview of scheduling algorithms:

	Description	Advantages	Disadvantages
FIFO/ FCFS	Resources are allocated to jobs one after the other.	simple	no consideration of other factors
SJF	Jobs ordered by increasing execution time.	optimal w.r.t. ø-turnaround time (if jobs arrive at the same time)	longer jobs may have to wait a long time
STCF	Select job with lowest remaining time. If new job arrives, preempt running job if necessary.	optimal w.r.t. ø-turnaround time	Starvation of very long jobs is possible
Round Robin	Execute jobs alternately for a time slice of fixed length.	fair no starvation good w.r.t. response time	Context switches cost time bad w.r.t. turnaround time inflexible due to fixed time slice length

(b) The schedules are shown in Figure 2. Note that other schedules with different round-trip or response times may also be possible.

Overview of the resulting turnaround and response times:

Algorithm	ø–Turnaround Time	ø–Response Time
FIFO	$\frac{6+13+(16-2)+(19-5)}{4} = 11.75$	$\frac{0+6+(13-2)+(16-5)}{4} = 7$
SJF	$\frac{6+(9-2)+(12-5)+19}{4} = 9.75$	$\frac{0+(6-2)+(9-5)+12}{4} = 5$
STCF	$\frac{12+(5-2)+(8-5)+19}{4} = 9.25$	$\frac{0+(2-2)+(5-5)+12}{4} = 3$
Round Robin (length 1)	$\frac{17+(11-2)+(14-5)+19}{4} = 13.5$	$\frac{0+(2-2)+(5-5)+1}{4} = 0.25$

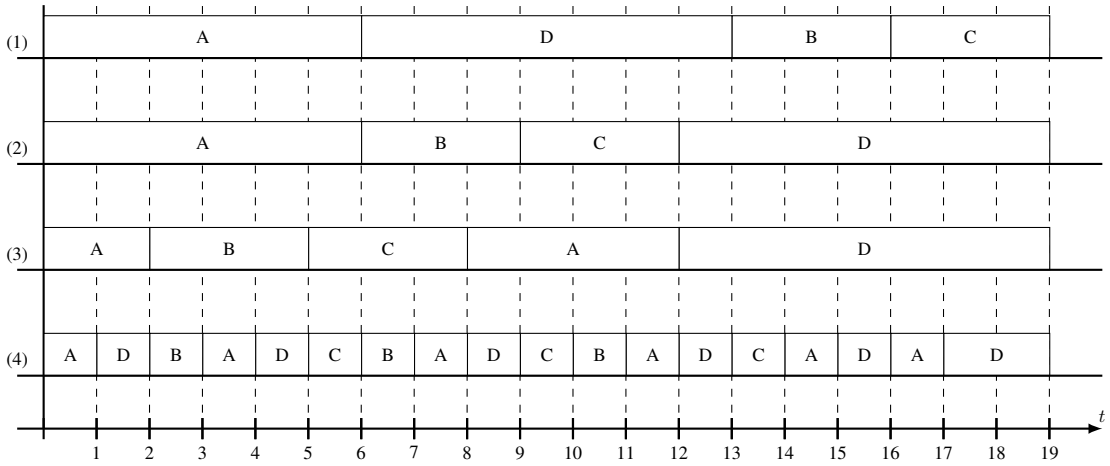

Figure 2: Schedules for the different algorithms (1–4 correspond to FIFO, SJF, STCF and Round Robin in this order)

Problem 12.2: Scheduling (2)

(a) We want to show by contradiction that SJF is optimal with respect to the average turnaround time. For this, we assume that there is a scheduling algorithm S $\neq$ SJF that is optimal with respect to the average turnaround time.

First, we observe that S does not preempt job executions (unlike, for example, STCF) because the average turnaround time may no longer be optimal for *simultaneously arriving* jobs. This is because preemptions unnecessarily delay the completion of jobs, thereby increasing the average turnaround time. (Remember: turnaround time is the difference between arrival time and completion time).

Now that we can rule out that the scheduling algorithm preempts jobs, we can easily do the rest of the proof: Since S $\neq$ SJF, there must be two processes A and B such that $|A| < |B|$ but B is executed before A. So this schedule looks as follows:

$$\underbrace{\ldots}_{x} B \underbrace{\ldots}_{y} A \underbrace{\ldots}_{z} \tag{3}$$

To obtain the contradiction, we now prove that schedule 2 (in which A and B are interchanged) has a lower average turnaround time; this shows that S is not optimal.

$$\underbrace{\ldots}_{x} A \underbrace{\ldots}_{y} B \underbrace{\ldots}_{z} \tag{4}$$

Since by construction x,y, and z are identical, their contribution to the turnaround time does not differ between the two schedules. Thus, it is sufficient to show that

$$Turnaround_1(A) + Turnaround_1(B) > Turnaround_2(A) + Turnaround_2(B)$$

$$\begin{aligned} Turnaround_1(A) + Turnaround_1(B) &= (x + |B| + y + |A|) + (x + |B|) \\ &= 2x + 2|B| + y + |A| \\ &\overset{|A|<|B|}{>} 2x + |B| + y + 2|A| \\ &= (x + |A|) + (x + |A| + y + |B|) \\ &= Turnaround_2(A) + Turnaround_2(B) \end{aligned}$$

■

(b) Counterexample:

Job	Arrival Time	Run-Time
A	0	10
B	1	2
C	2	3

In this case, the average turnaround time with SJF is:

$$\frac{(10-0)+(12-1)+(15-2)}{3} = \frac{34}{3} \approx 11,33$$

However, for STCF, it is:

$$\frac{(15-0)+(3-1)+(6-2)}{3} = \frac{21}{3} = 7$$

Thus, SJF is not optimal here. This effect is described in the slide deck on scheduling on slide 12 under the term "convoy effect", and it is caused by the fact that the much shorter jobs arrive shortly after the much longer job A, but SJF, unlike STCF, does not preempt jobs.

Problem 12.3: Multi-Level Feedback Queue (MLFQ)

(a) Rules for an MLFQ:

Rule 1: *Priority*(A) > *Priority*(B) $\rightarrow$ *execute* A

Allows processes to be handled differently based on their priorities. By setting the priorities, it is possible to decide which process should be executed.

Rule 2: *Priority*(A) = *Priority*(B) $\rightarrow$ *Round Robin between* A *and* B

Processes with the same priority are handled equally, prevents "starvation" of processes with the same priority.

Rule 3: *Processes start at top priority.*

New jobs are executed first, so that they can be processed quickly if they take only little time. Long jobs may be preempted, a kind of STCF is implemented. This guarantees a short response time.

Rule 4: *Reduce priority of a job when it has exhausted its budget at a priority level*

If a job takes too long, it has to wait after a short time. This implements the part of STCF that guarantees that only short jobs can permanently preempt a long process. The scheduler "learns" whether a job is short.

Rule 5: *Every* S *time units boost the priority of all jobs (that haven't been scheduled)*

Ensures that long jobs don't starve at low priority as new, shorter jobs keep popping up.

(b) The MLFQ degenerates to a round robin scheduler if all jobs always have the same priority. This can be achieved by always using exactly one priority level in the MLFQ, for example by specifying that there is only one priority, or by never reducing priorities, or by raising them again as soon as they are reduced.

Problem 12.4: Segmentation

The solution is described in more detail in chapter 16.3 of `http://pages.cs.wisc.edu/~remzi/OSTEP/vm-segmentation.pdf`. The idea here is to extend the MMU state by a bit that identifies in which direction the segment extends from the base address. Depending on this bit, the offset must be subtracted from the maximum segment size during address translation before the (then negative) result is added to the base address. Also depending on the bit the bounds check must be adjusted to compare the absolute value of the (negative) result with the segment size.

Problem 12.5: Memory Virtualization

(a) **Time sharing:** The process control block contains a hard disk address. Upon a process switch, the main memory is written to the hard disk and the data at the hard disk address of the new process is loaded into the main memory. For optimization reasons, one may want to note which memory is actually used in the PCB, so that no unused memory areas are saved.

Static relocation: The process control block contains the base address and the amount of memory required by the process. Upon a process switch, nothing special has to be considered. Only when starting a process one has to take care to rewrite the used addresses to a free base address (the information in the PCBs is needed to select a free address area).

Dynamic relocation: Like static relocation. In this case, however, the base address and possibly the size must be written into the corresponding MMU registers at every process switch. In return, no special handling at the process start is necessary any more.

Segmentation: The PCB contains base and bound for each segment. Upon a process switch, these are written to the corresponding MMU registers.

Paging: The PCB contains the address of the page table of the process. Upon a process switch, this address is loaded into the corresponding MMU register.

Paging with TLB: Like paging, but now the TLB must also be flushed upon a process switch. Systems which fill the TLB in software (e.g., MIPS) do not have to tell the MMU the address of the page table.

(b) Segmentation: The code segment of process 1 starts at 0, the code segment of process 2 starts at 0x2000. The data segment of process 1 starts at 0x3000, that of process 2 at 0x1000.

Process 1	Process 2	Comment
0x0100	-	
0x0104	-	
0x3234	-	
-	0x2100	
-	0x2104	
-	0x1234	
-	0x2108	
-	0x210c	
-	0x1234	Process 2 terminates
0x0108	-	
0x010c	-	
0x3234	-	

(c) Paging: The pagetables of process 1 and 2 are at 0 and 0x0800 (4 entries each). The page size is 4 kB.

Process 1	Process 2	Comment
0x0000	-	Page 0 is at 0x9000
0x9100	-	
0x0000	-	Page 0 is at 0x9000
0x9104	-	
0x0004	-	Page 1 is at 0x7000
0x7234	-	
-	0x0800	Page 0 is at 0x3000
-	0x3100	
-	0x0800	Page 0 is at 0x3000
-	0x3104	
-	0x0804	Page 1 is at 0x4000
-	0x4234	
-	0x0800	Page 0 is at 0x3000
-	0x3108	
-	0x0800	Page 0 is at 0x3000
-	0x310c	
-	0x0804	Page 1 is at 0x4000
-	0x4234	Process 2 terminates
0x0000	-	Page 0 is at 0x9000
0x9108	-	
0x0000	-	Page 0 is at 0x9000
0x910c	-	
0x0004	-	Page 1 is at 0x7000
0x7234	-	

(d) Paging with TLB: The pagetables of process 1 and 2 are at 0 and 0x0800 (4 entries each). The page size is 4 kB. We use a TLB of size 4, which is initially empty.

Process 1	Process 2	Comment
0x0000	-	Page 0 is at 0x9000
0x9100	-	
0x9104	-	Translation is available in the TLB
0x0004	-	Page 1 is at 0x7000
0x7234	-	
-	0x0800	Page 0 is at 0x3000
-	0x3100	
-	0x3104	Translation is available in the TLB
-	0x0804	Page 1 is at 0x4000
-	0x4234	
-	0x3108	Translation is available in the TLB
-	0x310c	Translation is available in the TLB
-	0x4234	Process 2 terminates
0x0000	-	Page 0 is at 0x9000
0x9108	-	
0x910c	-	Translation is available in the TLB
0x0004	-	Page 1 is at 0x7000
0x7234	-	

If the TLB supports *Address Space Identifiers*, the last two accesses to the page table could have been avoided.