Aufgabe 1: Virtual Memory

Zeichnen Sie ein System das Paging zur Speicherverwaltung verwendet. Die Zeichnung soll analog zur Vorlesungsfolie (vgl. Foliensatz 21 – Speichermanagement, Folie 20, rechts) gestaltet sein.

a) Entwerfen Sie das System mit folgenden Eigenschaften:

- Die Page-Größe muss unter 1025 liegen.
- Es wird byteweise adressiert.
- Es ist weniger Speicher verbaut als adressierbar ist.
- Die Page-Table ist dynamisch implementiert. Es verursachen somit nur verwendete Pages einen Eintrag in der Page-Table.

b) Geben Sie für Ihr System 4 virtuelle Adressen, deren Eintrag in der Page-Table und die dazugehörigen physikalischen Adressen an.

c) Wie sieht ein System mit einer *möglichst kleinen* Adresslänge aus, das die Anforderungen aus a) erfüllt?

Aufgabe 2: Netzwerke – IPv6

Gegeben ist folgendes IPv6 Netzwerk in CIDR Notation:

`0BAD:0000:CAFE:00FF:0000:0000:222.173.190.239/64`

a) Geben Sie das Netzwerk in minimaler Notation (vgl. Foliensatz 18 – Netzwerkprotokolle, Folie 34) an.

b) Wandeln Sie die Adresse in eine reine IPv6 Adresse um (d.h. ohne Dezimalnotation).

c) Wie viele Hosts sind im gegebenen Netzwerk adressierbar?

d) Teilen Sie das Netzwerk in 4294967296 Teile auf und geben Sie folgende Informationen an:

- Die erste IP des letzten Teilnetzes (CIDR Notation).
- Die Broadcast Adresse des letzten Teilnetzes.

Aufgabe 3: Netzwerke – Subnetting

Sie bekommen folgendes Netzwerk vom Provider zugeteilt: `128.192.224.0/23`

Entwerfen und zeichnen Sie ein Netzwerk, das folgende Anforderungen erfüllt:

- Das Netzwerk soll in mindestens 3 Teile zerlegt werden.
- Geben Sie für jeden Teil folgende Informationen an:
  - verwendbarer IP Adressbereich
  - Subnetzmaske
  - Broadcast Adresse
  - Netzwerkname
- Verteilen Sie 8 Clients auf mindestens 3 Subnetze Ihres Netzwerks. Vergeben Sie den Clients gültige IP Adressen und Subnetzmasken.
- Im Netzwerk befindet sich ein Router mit der IP Adresse `128.192.224.66`, der dahinter per NAT ein eigenes `/24` Netzwerk aufspannen soll. Geben Sie für das Netzwerk hinter der NAT einen gültigen IP Adressbereich sowie die Subnetzmaske an.

Aufgabe 4: Netzwerke – Routing

Gegeben ist folgende Routing-Tabelle:

Zieladresse	Netzmaske	Gateway	Interface
192.102.0.0	255.255.255.0	on-link	A
192.102.1.0	255.255.0.0	66.66.66.66	B
192.168.0.0	255.255.0.0	192.168.0.1	C
212.16.39.4	255.254.0.0	66.66.66.66	D
192.102.1.0	255.255.255.0	99.66.66.66	E
0.0.0.0	0.0.0.0	192.102.0.1	

Bei jedem Paket, das zugestellt werden muss, wird die Routing-Tabelle von oben nach unten nach einem passenden Eintrag durchsucht. *On-link* bedeutet, der Adressraum befindet sich im lokalen Netz und die Pakete können direkt zugestellt werden.

a) Über welches Interface werden Pakete an die IP Adresse `192.102.128.1` gesendet?

b) Wie groß ist das lokale Netz?

c) Welcher Gateway ist für die IP-Adresse `192.102.1.4` zuständig?

d) Wohin wird die IP Adresse `212.17.39.4` geroutet?

e) Wohin wird die IP Adresse `212.18.39.4` geroutet?

Aufgabe 5: Scheduling – Preemptive Scheduling

Die folgende Prozessliste soll mit verschiedenen Scheduling-Algorithmen abgearbeitet werden. Alle Algorithmen sind *preemptive*. Verwenden Sie die angegebenen Symbole beim Eintragen in die Tabellen.

Prozess	Startzeit	Laufzeit	Priorität
A	0	5	0
B	1	3	0
C	3	1	2
D	3	3	4
E	4	2	2
F	6	1	1

Zustand	Symbol
RUNNING	X
READY	—
BLOCKED	B

a) *Priority Scheduling*, wobei höhere Prioritäten höhere Werte haben. Bei gleicher Priorität bzw. nicht definierten Scheduling-Entscheidungen wird der Prozess mit der kürzesten Restlaufzeit bevorzugt:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A																
B																
C																
D																
E																
F																

b) *Shortest Remaining Time*, bei nicht definierten Scheduling-Entscheidungen wird jene mit den wenigsten Context Switches bevorzugt:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
A																
B																
C																
D																
E																
F																

Aufgabe 6: Scheduling – Non-Preemptive Scheduling

Auf einem System werden Prozesse nach einem *Non-Preemptive Scheduling* Verfahren mit folgenden Besonderheiten ausgeführt:

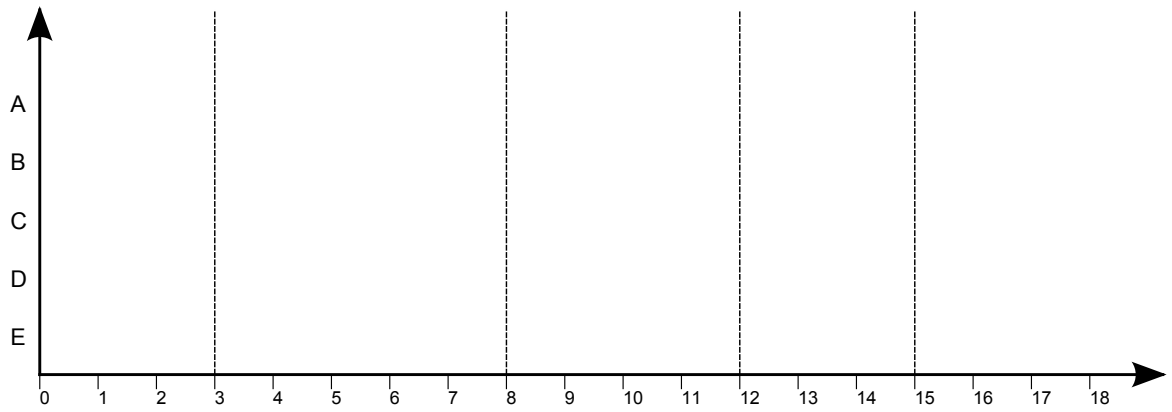
- Falls sich kein Prozess im Zustand RUNNING befindet, wählt der Scheduler jenen Prozess aus, der sich in der Warteschlange (*Ready-Queue*) an vorderster Position befindet.
- Die Prozesse selbst verhalten sich kooperativ, indem sie sich nach drei Zeiteinheiten selbst unterbrechen, falls die Restlaufzeit mehr als zwei Zeiteinheiten beträgt.

  Beispiel: Der Prozess A hat eine Laufzeit von 7 Zeiteinheiten. Nach 3 Zeiteinheiten unterbricht er sich selbst, weil die Restlaufzeit noch 4 Zeiteinheiten beträgt. Nachdem A den Prozessor wiedererlangt hat, läuft er ohne Unterbrechung durch, weil nach 3 Zeiteinheiten nur mehr 1 Zeiteinheit als Restlaufzeit übrig bleibt.
- Ein Prozess, der sich selbst unterbricht, wird hinter allen in der Warteschlange wartenden Prozessen angereiht. Neu gestartete Prozesse werden zum Startzeitpunkt ebenfalls hinter allen in der Warteschlange wartenden Prozessen angereiht.

Gegeben ist die folgende Prozess-Tabelle mit Startzeitpunkt und Ausführungsdauer:

Prozess	Startzeit	Laufzeit
A	0	7
B	1	2
C	6	6
D	10	2
E	7	1

a) Stellen Sie grafisch (vgl. Foliensatz 19-20 – Prozessmanagement, Folie 54) den Ablauf der Prozesse dar, wenn der Scheduler den oben beschriebenen Algorithmus verwendet.



b) Geben Sie den Zustand der Ready-Queue zu den Zeitpunkten 3, 8, 12 und 15 an. Falls zum jeweiligen Zeitpunkt ein Prozess-Wechsel stattfindet, wählen Sie für Ihre Darstellung genau den Zeitpunkt *nachdem* der laufende Prozess unterbrochen wurde und *bevor* dem nächsten Prozess der Prozessor zugeteilt wird.

Aufgabe 7: Multicore-Scheduling

Auf einem Multiprozessorsystem mit 2 CPUs arbeitet der Scheduler nach einem *Round Robin*-Verfahren mit Zeitscheibenlänge 3. Der Scheduler verwendet eine gemeinsame Warteschlange (Ready-Queue) zur Verwaltung der Prozesse. Er wird immer nur alle 3 Zeiteinheiten, am Ende einer Zeitscheibe, aktiv. Sollte ein Prozess daher vor dem Ende der Zeitscheibe terminieren, wird der jeweiligen CPU erst am Ende der Zeitscheibe ein neuer Prozess zugeteilt.

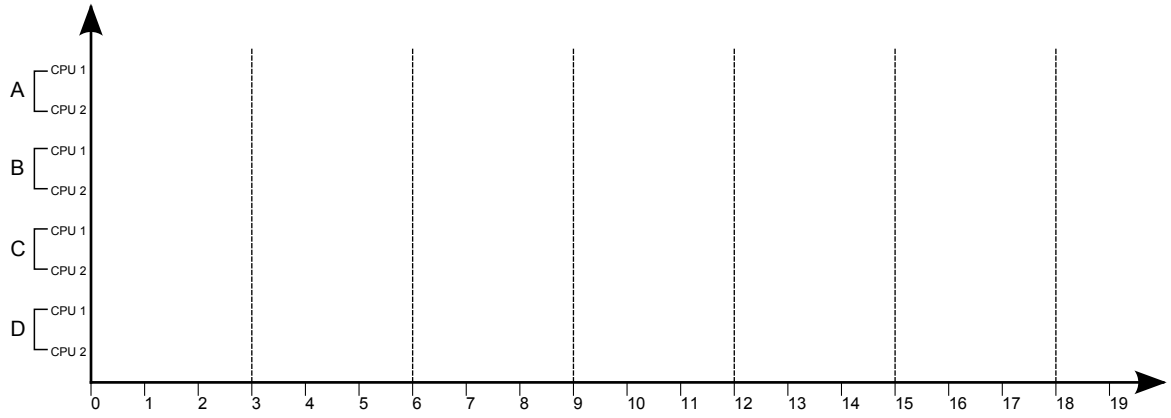
Der Scheduler weist die beiden CPUs den ersten beiden Prozessen in der Warteschlange nach folgendem Verfahren zu, wobei die CPUs durch eine fix vergebene CPU-Nummer (1 und 2) identifiziert werden:

- Ein unterbrochener Prozess wird bevorzugt an jene CPU vergeben, auf der er zuletzt ausgeführt wurde. Der zweite Prozess wird dann an die jeweils andere CPU vergeben.
- Falls eine bevorzugte Zuordnung nicht möglich ist (beispielsweise weil beide Prozesse dieselbe CPU beanspruchen oder weil beide Prozesse noch nie ausgeführt wurden), werden die Prozesse in der Reihenfolge, in der sie aus der Warteschlange entnommen wurden, vergeben: Also der erste Prozess an CPU 1, der zweite Prozess an CPU 2.
- Nach Ablauf der Zeitscheibe werden die laufenden Prozesse in aufsteigender Reihenfolge der CPU-Nummern in der Warteschlange hinten angereiht.

Auf dem System werden die folgenden Prozesse abgearbeitet:

Prozess	Startzeit	Laufzeit
A	1	8
B	4	2
C	0	13
D	8	7

a) Stellen Sie den Ablauf der Prozesse A bis D grafisch dar:



b) Geben Sie den jeweiligen Zustand der Warteschlange jeweils am Ende einer Zeitscheibe an, und zwar *nachdem* die laufenden Prozesse unterbrochen und in die Warteschlange eingereiht wurden und *bevor* der Scheduler die nächsten beiden Prozesse an die CPUs vergeben hat.

Zeitpunkt	3	2	1	0
0:				
3:				
6:				
9:				
12:				
15:				
18:				

Aufgabe 8: Prozesse – Prozesszustand

Gegeben ist folgendes Programm in Pseudo-Code Notation:

```
   int main() {
1:   i= get_input();
2:   f();
3:   return 0;
   }

   f() {
4:    if (i > 0)
5:       i= i-1;
6:       f();
      endif;
7:    return;
   }
```

Das Programm wird in zwei voneinander unabhängigen Prozessen $P1$ und $P2$ gestartet, wobei $P1$ zum Zeitpunkt 0 und $P2$ zum Zeitpunkt 1 startet. Die Prozesse werden in einem *Round Robin* Verfahren mit einer Zeitscheibenlänge von 4 Zeiteinheiten abwechselnd abgearbeitet, d.h. es werden 4 Schritte von $P1$ ausgeführt, dann 4 Schritte von $P2$, dann wieder 4 Schritte von $P1$ u.s.w.
Es wird weiters angenommen, dass `get_input()` in $P1$ den Wert 3 und in $P2$ den Wert 1 einliest.

Tragen Sie in nachfolgende Tabelle für jeden Schritt die Zustände der Prozesse $P1$ und $P2$ *am Beginn* des jeweiligen Schrittes ein. Verwenden Sie die im Pseudo-Code links eingetragenen Zahlen als Wert für den *Program Counter (PC)* und tragen Sie im Stack die jeweils abgespeicherte Folge der Rücksprungadressen ein. Gehen Sie dabei analog zu Foliensatz 19-20 – Prozessmanagement, Folien 13ff vor.

	Prozess $P1$			Prozess $P2$		
Zeit	PC	i	Stack	PC	i	Stack
0						
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21						
22						
23						
24						

Aufgabe 9: Interprozesskommunikation – Race Conditions

Gegeben sind zwei Prozesse $P1$ und $P2$, die über zwei gemeinsame Variablen A und B miteinander kommunizieren. Die Prozesse benutzen für die Kommunikation folgende Befehle:

read(X) liest den Wert der Variable X.

write(X,n) schreibt den Wert n in die Variable X.

Die Prozesse sind wie folgt programmiert, wobei angenommen wird, dass jede Befehlszeile genau einen Takt zur Ausführung benötigt:

```
   P1() {
1:   write(A,n);
2:   n= n+1;
3:   write(B,n);
   }
```

```
   P2() {
1:   read(B);
2:   read(A);
   }
```

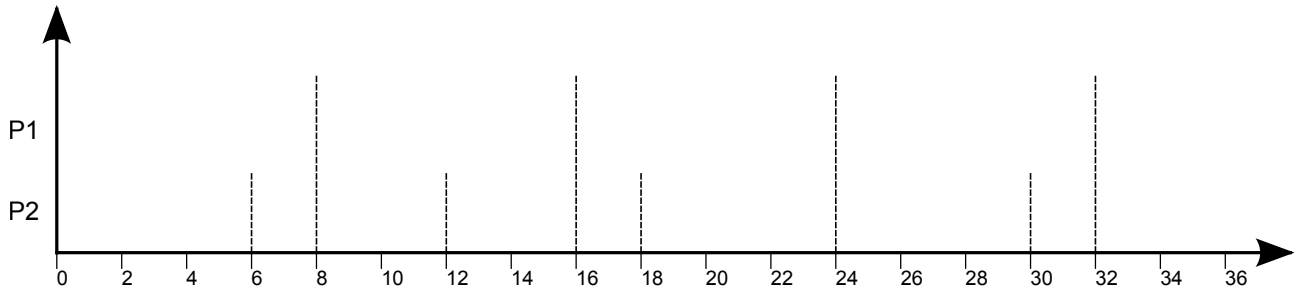
Hinweis: Die Variable n ist eine statische, lokale Variable des Prozesses $P1$. Sie ist vor dem ersten Aufruf von $P1$ mit 0 initialisiert und behält ihren Wert zwischen aufeinanderfolgenden Aufrufen von $P1$ bei.

Die Prozesse werden wie in der nachfolgenden Tabelle angegeben periodisch aufgerufen. Eine Periode k bedeutet, dass der jeweilige Prozess von der erstmaligen Startzeit T weg alle k Zeiteinheiten neu gestartet wird, insgesamt also zu den Zeitpunkten T, $T + k$, $T + 2k$, u.s.w.:

Prozess	Erste Startzeit	Laufzeit	Periode
P1	0	3	8
P2	6	2	6

Die Prozesse werden mit einem *Round Robin Scheduling*, das eine Zeitscheibenlänge von 1 Takt benutzt, ausgeführt. Sollten beide Prozesse zum gleichen Zeitpunkt starten, wird immer zuerst $P1$ und danach $P2$ eine Zeitscheibe zugewiesen.

a) Zeichnen Sie grafisch das Ablaufmuster der beiden Prozesse.



b) Analysieren Sie, wie das Ablaufmuster der beiden Prozesse die Kommunikation stört.

Hinweis: Achten Sie darauf, ob die Werte von A und B, die $P2$ liest, von $P1$ im selben Programmdurchlauf geschrieben wurden oder nicht.

c) Wie kann man durch ein geeignetes Ablaufmuster sicherstellen, dass die Werte, die $P2$ liest, von $P1$ im selben Programmdurchlauf geschrieben wurden? Geben Sie eine geeignete Bedingung dafür an.

Aufgabe 10: Interprozesskommunikation – Semaphore

Gegeben ist das Timing-Diagramm dreier Prozesse (A,B,C). Darüber hinaus gibt es eine gemeinsam genutzte Ressource, anhand derer die Prozesse kommunizieren können. Der Zugriff auf diese Ressource wird mithilfe eines Semaphores S synchronisiert. Die Operationen des Semaphores sind dabei im Timing-Diagramm gekennzeichnet.

a) Tragen Sie in den mit S : gekennzeichneten freien Feldern unterhalb des Diagramms die jeweiligen Werte des Semaphors *zwischen* zwei Befehlen (nach der Ausführung des erstens bzw. vor der Ausführung des zweiten) ein.

b) Tragen Sie in den mit *Zustand* gekennzeichneten Feldern den Zustand des Prozesses (Y = *Ready*, R = *Running*, B = *Blocked*) ein, in dem sich der jeweilige Prozess zwischen den markierten Zeitpunkten befindet.