Grundlagen der Informatik III

Wintersemester 2010/2011

Wolfgang Heenes, Patrik Schmittat

TECHNISCHE UNIVERSITÄT DARMSTADT

11. Aufgabenblatt mit Lösungsvorschlag

31.01.2011

Hinweis: Der Schnelltest und die Aufgaben sollen in den Übungsgruppen bearbeitet werden. Die Hausaufgaben sind in der Kalenderwoche 6 (07.02. bis 11.02.) bei den Tutoren in **physikalischer Form** (handschriftlich oder gedruckt) abzugeben. Bei allen Abgaben ist der Name des Tutors und die Übungsgruppe deutlich anzugeben. Bei Teamabgaben wird nur eine Lösung eingereicht, die alle Namen der Teammitglieder enthält.

Schicken Sie Ihre Lösungen von Programmieraufgaben zusätzlich zur schriftlichen Abgabe per E-Mail an Ihren Tutor. Kommentieren Sie Ihren Quellcode.

Aufgabe 1: Schnelltest

Fragen	*Antworten*
1. *Welche der folgenden Aussagen sind korrekt ?*	☐ *Im Benutzermodus sind privilegierte (Prozessor)-Befehle verfügbar.* ■ *Asynchrone Interrupts sind im Allgemeinen nicht reproduzierbar.* ■ *Das Betriebssystem behandelt Interrupts selbständig.* ☐ *Multithreading sorgt für parallele Ausführung mehrerer Prozesse.* ■ *Threads teilen sich einen gemeinsamen Adressraum.*
2. *Zur Unterbrechungsbehandlung muss*	■ *das Prozessorstatuswort gesichert werden* ■ *das IC-Register neu geladen werden.* ☐ *der Cache vollständig geleert werden.*
3. *Welche Aussagen zu Semaphoren sind richtig?*	☐ *Mit Semaphoren läßt sich wechselseitiger Ausschluss verhindern.* ■ *Mit Semaphoren läßt sich wechselseitiger Ausschluss realisieren.* ■ *Semaphoren können Verklemmungen nicht verhindern.* ■ *Das Verändern von Semaphoren muss atomar ablaufen*
4. *Bei welchen der folgenden Schedulingverfahren können Prozesse verhungern?*	☐ *FIFO* ■ *Shortest Job First* ☐ *Round Robin* ■ *Statische Prioritätenvergabe* ☐ *Dynamische Prioritätenvergabe*

Aufgabe 2: Betriebssysteme

a) *Was ist im Zusammenhang von Prozesssynchronisation mit dem Begriff „atomar" gemeint? Warum müssen die Semaphor Operationen „P" und „V" atomar sein?*

Lösungsvorschlag:

Eine atomare Operation ist eine Operation die nicht unterbrochen werden kann. Wenn „P" und „V" nicht atomar ausgeführt werden treten „race conditions" auf, da auf eine gemeinsame Variable zugegriffen wird, d. h. wenn in einem ungünstigen Moment eine Unterbrechung auftritt, kann beispielsweise die Schreiboperation einer Semaphor Operation durch eine andere Semaphor Operation überschrieben werden.

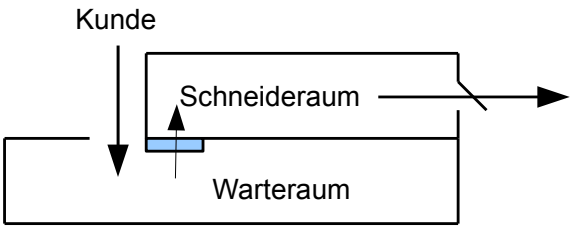
b) *Was sind die Anforderungen an eine Realisierung des wechselseitigen Aussschlusses?*

Lösungsvorschlag:

- Die kritischen Abschnitte der Prozesse sind wechselseitig auszuschließen.
- Es dürfen keine Annahmen über die Reihenfolge der Ausführung kritischer Abschnitte gemacht werden.
- Es dürfen keine Annahmen über die Länge der Ausführungszeiten kritischer Abschnitte gemacht werden.
- Kein Prozess muss unendlich lange darauf warten, seinen kritischen Abschnitt betreten zu dürfen („fairness condition").

Aufgabe 3: Semaphore: Sleeping Barber Problem

Der Laden eines Friseurs besteht aus einem Warteraum, der beliebig viele Kunden aufnehmen kann und einem Haarschneideraum. Beide Räume besitzen eine gemeinsame, jeweils nur einen Eingang schließende Schiebetür.



- *Die Kunden betreten nacheinander den Warteraum.*
- *Hat der Friseur einen Haarschnitt beendet, so verläßt der Kunde den Schneideraum und der Friseur öffnet die Tür zum Warteraum und schaut nach, ob ein weiterer Kunde wartet.*
  - *Ist dies der Fall, so bittet er diesen in den Schneideraum;*
  - *andernfalls setzt er sich auf seinen Stuhl im Schneideraum und schläft.*
- *Tritt ein Kunde in den Warteraum und findet den Friseur schlafend,*
  - *so weckt er diesen und läßt sich dann die Haare schneiden;*
  - *andernfalls wartet der Kunde.*

Schreiben Sie Algorithmen für die Verhaltensweise des Friseurs und der Kunden unter Verwendung von Semaphoren. Als Syntax bietet sich die Systemprogrammiersprache aus der Vorlesung an.

Lösungsvorschlag:

Die Problemstellung entspricht in etwa der eines unendlichen Puffers, wobei der Friseur den Konsumenten und die Kunden die Produzenten darstellen.

Für die Ausführung einer V-Operation wird eine FIFO-Strategie zugrundegelegt

```
     door : SEMAPHORE == 1;
                -- mutex für Auftragsabwicklung

     ctr : INTEGER == 0;
                -- Anzahl der Kunden im Warteraum

     sleep : BOOLEAN == FALSE;
                -- Zustand des Friseurs

     barber : SEMAPHORE == 0;
                -- 'Warteplatz' des Friseurs

     customer : SEMAPHOR == 0;
                --  'Wartebank' der Kunden

     kunde : PROCESS;
       BEGIN
(1)      P(door);
         IF sleep
(2)         THEN
(3)           sleep := FALSE;
(4)           V(barber);
(5)           V(door);
(6)         ELSE
(7)           ctr := ctr + 1;
(8)           V(door);
(9)           P(customer);
            END;
         Haare schneiden lassen;
       END;

     friseur : PROCESS;
       BEGIN
         LOOP
(10)       P(door);
           IF ctr > 0
(11)          THEN
(12)            ctr := ctr - 1;
(13)            V(customer);
(14)            V(door);
(15)          ELSE
(16)            sleep := TRUE;
(17)            V(door);
(18)            P(barber);
              END;
           Haare schneiden;
         REPEAT;
       END;
```

(1) Falls sich gerade der Friseur im Warteraum umschaut (vergl. (10)), steht der Kunde vor verschlossener Schiebetür und muß warten, bis der Friseur in den Schneideraum zurückgegangen ist und die Tür zum Warteraum wieder geöffnet hat.

Spätestens dann kann der Kunde den Warteraum betreten und sich umschauen. Dazu öffnet er die Tür zum Schneideraum (`P(door)`), um zu überprüfen, ob der Friseur gerade arbeitet oder schläft. Weitere Kunden stehen dann vor verschlossener Tür zum Warteraum.

(2) Da der Friseur schläft

oder

gerade dabei ist, sich schlafenzulegen (d.h. er befindet sich gerade zwischen `V(door)` und `P(barber)`),

(3), (4) weckt der Kunde ihn auf oder teilt ihm mit, daß es keinen Sinn mehr hat, sich schlafenzulegen (in diesem Fall wird `V(barber)` vor `P(barber)` durchgeführt).

(5) Der Kunde betritt den Schneideraum und schließt die Tür zum Schneideraum hinter sich (`V(door)`).

(6) Da der Friseur gerade arbeitet oder darauf wartet, sich nach einem neuen Kunden umschauen zu können (vergl. (10)),

(7) wird der Kundenzähler erhöht.

(8) Der Kunde öffnet die Schiebetür zum Warteraum wieder und

(9) setzt sich auf die Wartebank.

(10) Falls sich gerade ein Kunde im Friseursalon *umschaut*, muß der Friseur mit dem Eintritt in den Warteraum warten, *obwohl die Tür für ihn offen ist*! Nachdem der Kunde die Tür zum Schneideraum wieder geschlossen hat (`V(door)` aus (8)), kann der Friseur die `P(door)`-Operation beenden und damit die Tür zum Schneideraum wieder öffnen und den Warteraum betreten.

Andernfalls (kein Kunde schaut sich gerade um) öffnet sich der Friseur durch die `P(door)`-Operation die Tür vom Schneideraum zum Warteraum und schließt sie damit gleichzeitig (kurzfristig) für einen neuen Kunden.

(11) Da mindestens ein Kunde im Warteraum vorhanden ist (entweder sitzt er bereits auf der Wartebank oder ist gerade dabei sich zu setzen (Unterbrechung zwischen `V(door)` und `P(customer)`)),

(12), (13) ruft er den sich am längsten im Warteraum befindenden Kunden auf, geht mit ihm in den Schneideraum und

(14) und öffnet Kunden wieder die Tür zum Warteraum.

(15) Da sich kein Kunde im Warteraum befindet,

(16) markiert der Friseur, daß er nun gedenkt sich schlafenzulegen.

(17) Er geht in den Schneideraum zurück, schließt die Tür hinter sich und öffnet damit den Kunden wieder die Tür zum Warteraum.

(18) Schließlich setzt er sich auf seinen Stuhl und schläft (sofern ihn nicht gerade ein Kunde beim Hinsetzen anspricht, d. h. zwischen `V(door)` und `V(barber)`).

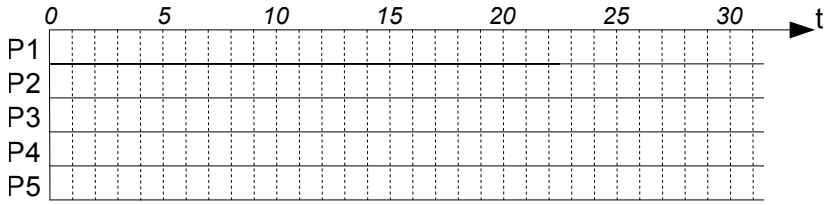
Aufgabe 4: Scheduling

Gegeben seien fünf Prozesse mit jeweiligen Start- und CPU-Zeiten, sowie deren Priorität (Prozesse mit kleinerem Prioritätswert kommen vor Prozessen mit größeren Werten):

Prozess	*Startzeit*	*CPU-Zeit*	*Statische Priorität*
P1	*0*	*8*	*6*
P2	*2*	*4*	*7*
P3	*3*	*2*	*3*
P4	*3*	*7*	*4*
P5	*6*	*3*	*2*

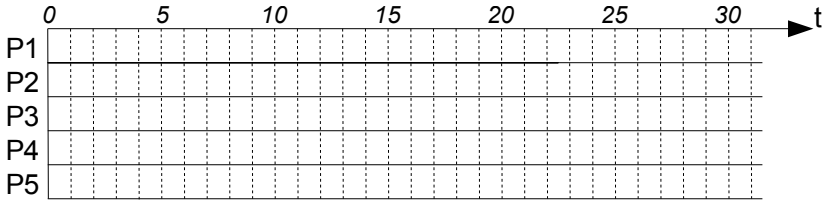
Betrachten Sie nur die Prozesszustände bereit (B) und laufend (L).

Zur Darstellung des Schedulings sind Gantt-Diagramm sehr gut geeignet (vgl. Vorlesung 24). Ein Gantt-Diagramm hat folgendes Aussehen:

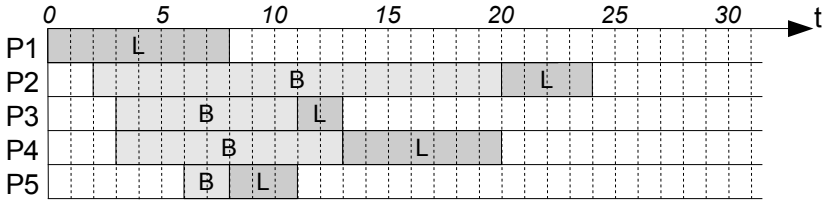


Zeichnen Sie für jedes der folgenden Schedulingverfahren ein Gantt-Diagramm der Prozesse.

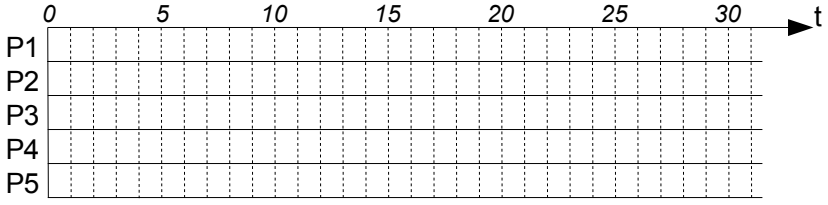
a) *Statische Priorität (non-preemptive):*



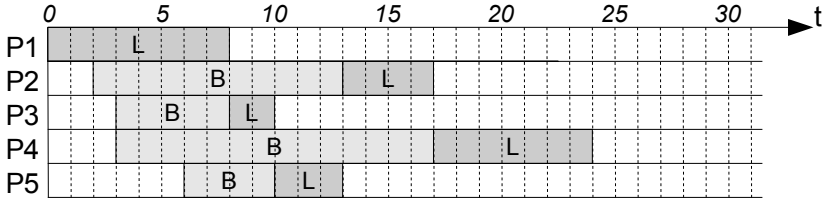
Lösungsvorschlag:



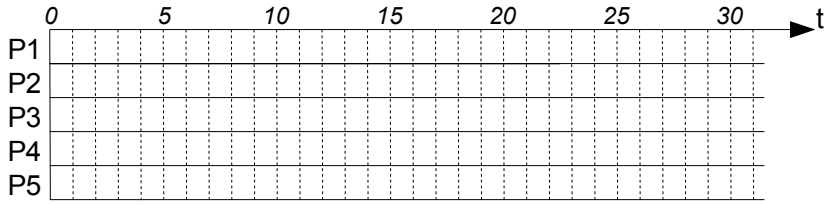
b) *Shortest Job First (non-preemptive):*



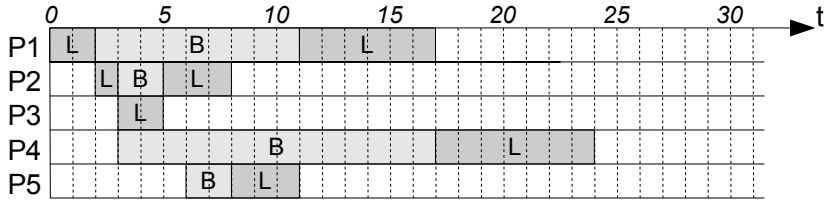
Lösungsvorschlag:



c) *Shortest remaining time next (preemptive):*



Lösungsvorschlag:



d) *Berechnen Sie die mittlere Wartezeit für jedes Schedulingverfahren.*

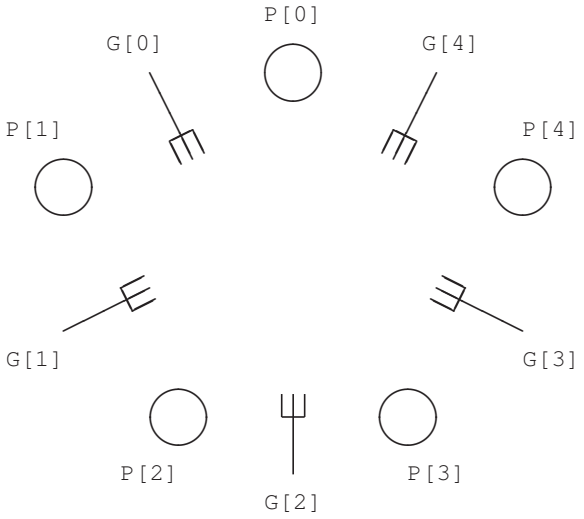
Prozess	*Prio.*	*SJF*	*SRTN*
P1			
P2			
P3			
P4			
P5			
Mittlere Wartezeit			

Lösungsvorschlag:

Prozess	Prio.	SJF	SRTN
P1	0	0	9
P2	18	11	2
P3	8	5	0
P4	10	14	14
P5	2	4	2
Mittlere Wartezeit	7,6	6,8	5,4

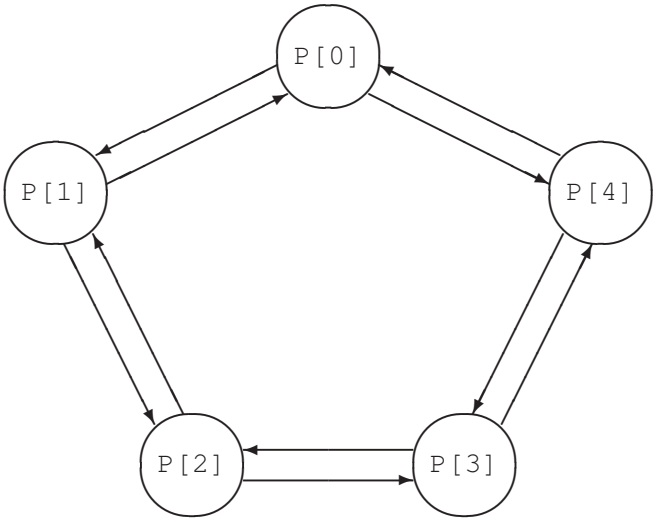
Hausaufgabe 1: Semaphore: 5-Philosophenproblem (6 Punkte)

Fünf Philosophen sitzen gemeinsam an einem runden Tisch, an dem sie unabhängig voneinander von Zeit zu Zeit Spaghetti essen. Jeder Philosoph hat rechts neben seinem Teller nur eine Gabel, benötigt aber zum Essen zwei Gabeln, also auch die seines linken Nachbarn. Aus diesem Grund können nicht alle Philosophen gleichzeitig essen, sondern sie müssen sich bezüglich ihrer kritischen Abschnitte **Essen** *synchronisieren*



a) *Formulieren Sie die Synchronisationsbedingung mittels eines Ausschlußgraphen.*

Lösungsvorschlag:



b) *Geben Sie eine Programmlösung unter Verwendung der P- und V-Operationen an, bei der es zu keinerlei Verklemmungen kommen kann. Als Syntax für die Programmlösung bietet sich die Systemprogrammiersprache aus der Vorlesung an.*

Hinweise:

- *Die Philosophen sollten als Prozesse modelliert werden.*
- *Jeder Philosoph erlebt zyklisch drei Zustände*
  - *denkend,*
  - *hungrig und*
  - *essend*
- *Die Gabeln stellen die Betriebsmittel dar.*

Lösungsvorschlag:

Primitivlösung:

- vier Philosophen beantragen zunächst die *rechte* und dann die *linke* Gabel
- der fünfte Philosoph zunächst die *linke* und dann die *rechte*

Reihenfolge der Gabelbelegung: `G[4]`, `G[3]`, `G[2]`, `G[1]`, `G[0]`

```
TYPE philos = (0..4);

fork : ARRAY philos OF SEMAPHORE == [EACH == 1];

philosopher : PROCESS (i : IN philos);
  BEGIN
    LOOP
      denken;
      IF i = 0
         THEN
           P(fork[4]);
           P(fork[0]);
         ELSE
           P(fork[i]);
           P(fork[i-1]);
         END;
      essen;
      IF i = 0
         THEN
           V(fork[4]);
           V(fork[0]);
         ELSE
           V(fork[i]);
           V(fork[i-1]);
         END;
    REPEAT;
  END;
```

Eigenschaften: Lösung ist verklemmungsfrei

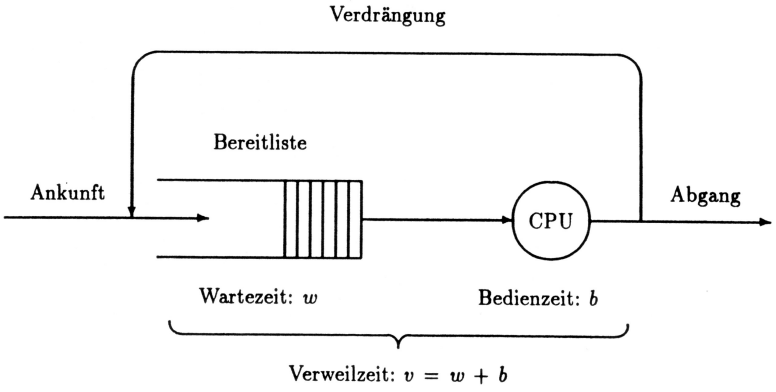
- Im Wartegraphen kann aufgrund der Anforderungsstrategie für die Gabeln kein Zyklus auftreten.

- kein Philosoph verhungert
  - Da jeder Philosoph die Gabeln nach dem Essen wieder zurückgibt, muß ein vor einem Semaphor wartender Philosoph nach endlicher Zeit diese `P`-Operation passieren.
- Vertauschen der `V`-Operationen im Austrittsprotokoll des kritischen Abschnitts
  - An den Eigenschaften *Verklemmungsfreiheit* und *Aushungerungsfreiheit* ändert sich nichts. Es ändert sich die Reihenfolge, in der hungrige Philosophen zu essen *anfangen* können.
- es ist möglich, daß alle Gabeln belegt sind und trotzdem nur ein Philosoph ißt

Beispiel: `P[3]` hat `G[3]`
`P[2]` hat `G[2]`
`P[1]` hat `G[1]`
`P[0]` hat `G[0]` und `G[4]`

Hausaufgabe 2: Scheduling (4 Punkte)

Die folgende Abbildung beschreibt ein einfaches Modell für die Prozeßbearbeitung.



Fünf Prozesse treffen zu gegebenen Zeitpunkten in der Bereitliste ein. Es ist bekannt, wieviel Bedienzeit (=Rechenzeit) sie benötigen. Jeder Prozess hat eine Priorität (0 stellt die höchste Priorität dar).

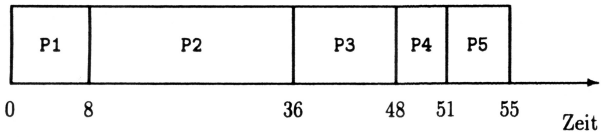
Die folgende Tabelle gibt die Ankunftszeitpunkte, Bedienzeiten und Prioritäten der einzelnen Prozesse wieder.

Prozeß	Ankunftszeit	Bedienzeit	Priorität
1	0	8	4
2	3	28	1
3	7	12	0
4	9	3	2
5	15	4	3

Die Prozesse sollen nun unter verschiedenen Dispatcher-Strategien laufen.

- *Ohne Verdrängung, d. h. nach der Zuteilung der CPU arbeitet ein Prozess, bis er von selbst die Prozessor abgibt.*
  - *a) First-In-First-Out (FIFO)*
  - *b) Highest-Priority-First (HPF)*
  - *c) Shortest-Processing-Time-First (SPTF)*
    Dies entspricht einer HPF-Strategie, bei der die Priorität nach der Dauer der Bedienzeit vergeben werden.
- *Mit Verdrängung*
  - *d) Round-Robin (RR) mit dem Zeitscheibenwert 5*
  - *e) HPF mit Verdrängung bei der Ankunft eines Prozesses (mit höherer Priorität)*
  - *f) Shortest-Remaining-Time-First*
    Dies entspricht einer HPF-Strategie, bei der nach der Ankunft eines neunen Prozesses derjenige ausgewählt wird, der die kürzeste Restbedientzeit besitzt.

Das Gantt-Diagramm für die Dispatcher-Strategie a) ist in der folgenden Abbildung angegeben.

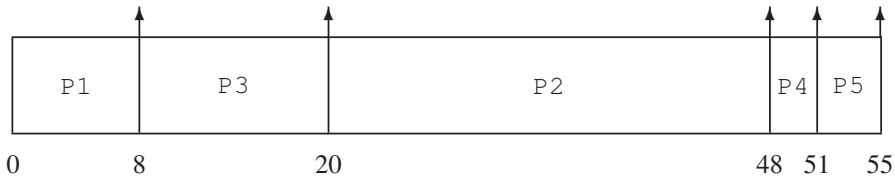


a) *Zeichnen Sie die Gantt-Diagramme für die Dispatcher-Strategien b) bis f).*

Lösungsvorschlag:

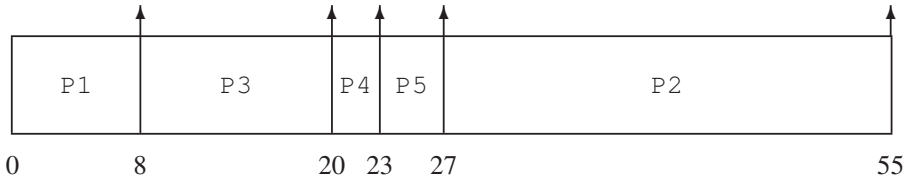
Bei den folgenden fünf Gantt-Diagrammen sind jeweils die Prozesse angeben, die dem Dispatcher zu dem aktuellen Zeitpunkt zur Verfügung stehen. Der Wert hinter der Prozeßnummer stellt das Auswahlkriterium für den Dispatcher dar (Priorität, Restlaufzeit, ...).

Gantt-Diagramm für die **HPF**-Strategie



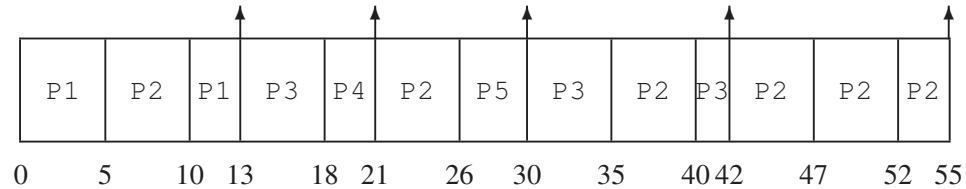

```
0: P1 (4)    8: P2 (1)    20: P2 (1)    48: P4 (2)    51: P5 (3)
             P3 (0)           P4 (2)        P5 (3)
                              P5 (3)
```

Gantt-Diagramm für die **SPTF**-Strategie



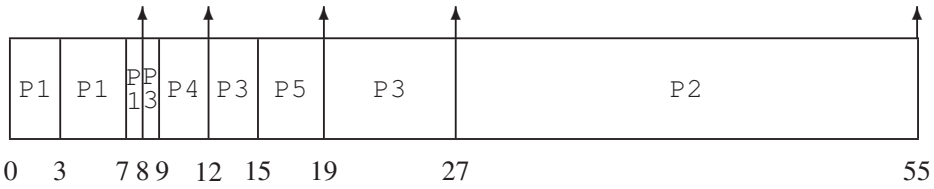

```
0: P1 (8)    8: P2 (28)    20: P2 (28)    23: P2 (28)    27: P2 (28)
             P3 (12)           P4 (3)         P5 (4)
                               P5 (4)
```

Gantt-Diagramm für die **RR**-Strategie




```
P1   P2   P1 P3   P4 P2   P5 P3   P2 P3   P3P2   P2   P2 P2
     P1   P3 P4   P2 P5   P3 P2   P3 P2
          P4 P2   P5 P3   P2
          P2      P3
```

Gantt-Diagramm für die **SRTF**-Strategie



0: P1 (8)

3: P1 (5)
P2 (28)

7: P1 (1)
P2 (28)
P3 (12)

8: P2 (28)
P3 (12)

9: P2 (28)
P3 (11)
P4 (3)

12: P2 (28)
P3 (11)

15: P2 (28)
P3 (8)
P5 (4)

19: P2 (28)
P3 (8)

27: P2 (28)

Gantt-Diagramm für die **HPF**-Strategie **mit Verdrängung bei Ankunft eines Prozesses**

0: P1 (8)

3: P1 (4)
P2 (1)

7: P1 (4)
P2 (1)
P3 (0)

9: P1 (4)
P2 (1)
P3 (0)
P4 (2)

15: P1 (4)
P2 (1)
P3 (0)
P4 (2)
P5 (3)

19: P1 (4)
P2 (1)
P4 (2)
P5 (3)

43: P1 (4)
P4 (2)
P5 (3)

46: P1 (4)
P5 (3)

50: P1 (4)