3A Synchronisation: Weitere Aufgaben

3A.1 Sprachunabhängige Anwendungsaufgaben

1. Gegeben ist ein Prozess-System, das über Semaphore synchronisiert wird:

Semaphore: SEM1.INIT(0); SEM2.INIT(1)		
Prozess A: Aktionen SEM1.V() SEM1.V()	Prozess B: SEM1.P() SEM2.P() Aktionen SEM2.V()	Prozess C: SEM1.P() SEM2.P() Aktionen SEM2.V()

   Wie nennt man die Synchronisationsbedingung, die durch SEM1 durchgesetzt wird, und die, die durch SEM2 durchgesetzt wird?

   SEM1: _____

   SEM2: _____

   1.)SEM1 Reihenfolge, SEM2 wechselseitiger Ausschluss.

   Beschreiben Sie kurz das zeitliche Verhalten des Gesamtsystems, das durch die Semaphoroperationen durchgesetzt wird.

   Prozess A wird als erster ausgeführt, dann Prozess B und Prozess C. B und C sind wechselseitig ausgeschlossen, ihre Reihenfolge untereinander ist nicht festgelegt.

2. P1, P2, P3 und P4 seien Prozesse. P4 soll erst dann weiterlaufen, wenn P1 einen Wert geliefert hat und wenn zudem P2 oder P3 einen Wert geliefert hat. Lösen Sie dieses Synchronisationsproblem mit Hilfe von zwei Semaphoren.

   2.)

Semaphore: SA.INIT(0); SB.INIT(0)			
P1: *arbeite;* *SA.V();*	*P2:* *arbeite;* *SB.V();*	*P3:* *arbeite;* *SB.V();*	*P4:* *SA.P();* *SB.P();* *weiter;*

Fällt Ihnen auch eine Lösung mit nur einem Semaphor ein?

Semaphor: S.INIT(0)			
P1: arbeite; S.V(); S.V();	*P2:* arbeite; S.V();	*P3:* arbeite; S.V();	*P4:* S.P(); S.P(); S.P(); weiter;

3. Ein System von drei Prozessen soll folgendermaßen ausgeführt werden: Zuerst soll Prozess A vollständig ablaufen. Anschließend sollen die Prozesse B und C jeweils vollständig ablaufen. Welcher von beiden zuerst ausgeführt wird, soll egal sein (also nicht durch die Programmstruktur von vornherein festgelegt sein); sie sollen allerdings stets hintereinander und nicht zeitlich durchmischt ablaufen. Zeigen Sie, wie mit Hilfe von Semaphoren diese Synchronisationsbedingungen durchgesetzt werden können.

3.)

Semaphore: S_REIHE_AB.INIT(0); S_REIHE_AC.INIT(0); S_WA.INIT(1)		
Prozess A: Aktionen S_REIHE_AB.V() S_REIHE_AC.V()	*Prozess B:* S_REIHE_AB.P() S_WA.P() Aktionen S_WA.V()	*Prozess C:* S_REIHE_AC.P() S_WA.P() Aktionen S_WA.V()

oder einfacher:

Semaphore: S_REIHE.INIT(0); S_WA.INIT(1)		
Prozess A: Aktionen S_REIHE.V() S_REIHE.V()	*Prozess B:* S_REIHE.P() S_WA.P() Aktionen S_WA.V()	*Prozess C:* S_REIHE.P() S_WA.P() Aktionen S_WA.V()

oder noch einfacher:

Semaphore: S.INIT(0);		
Prozess A: *Aktionen* *S.V()*	*Prozess B:* *S.P()* *Aktionen* *S.V()*	*Prozess C:* *S.P()* *Aktionen* *S.V()*

4. Eine Familie mit Vater, Mutter und Kind beginnt ihren Tag wie folgt: Nach dem Aufstehen geht zunächst jeder ins Bad. Die Reihenfolge dabei ist nicht festgelegt; es gibt jedoch nur ein Badezimmer, und es darf höchstens eine Person gleichzeitig im Bad sein. Wenn der Vater das Bad verlassen hat, bereitet er für das Kind einen Kakao zu; die Mutter schmiert dem Kind, nachdem sie aus dem Bad gekommen ist, ein Brötchen. Stehen Kakao und Brötchen auf dem Tisch, so frühstückt das Kind. Der Vater wartet, bis es damit fertig ist, und fährt es dann in die Schule.

   Sie sollen nun diese Situation durch ein System von Prozessen modellieren, die sich mit Hilfe von Semaphoren synchronisieren.

   - Welches sind die Prozesse in diesem System? Wieviele und welche Synchronisationsbedingungen sind im Einzelnen durchzusetzen? Geben Sie bei jeder Synchronisationsbedingung an, um welche allgemeine Art von Bedingung (siehe die beiden grundlegenden Begriffe aus dem vorigen Kapitel) es sich dabei handelt.

   4.)Prozesse: Vater, Mutter, Kind
   4 Synchronisationsbedingungen:
   1.) Das Bad kann nur von einem gleichzeitig benutzt werden (wechselseitiger Ausschluss)
   2.) Das Kind frühstückt, nachdem der Vater Kakao zubereitet hat (Reihenfolge)
   3.) Das Kind frühstückt, nachdem die Mutter Brötchen geschmiert hat (Reihenfolge)
   4.) Der Vater fährt das Kind in die Schule, nachdem es gefrühstückt hat (Reihenfolge)

   - Stellen Sie dar, wie diese Synchronisationsbedingungen über Semaphore durchgesetzt werden können. Geben Sie dazu erstens an, welche Semaphore benötigt und mit welchen Werten sie initialisiert werden. Skizzieren Sie zweitens die Programmstücke, die die Aktionen der Prozesse beschreiben. Kommentieren Sie Ihre Lösung geeignet!

Semaphore: S1.INIT(1); S2.INIT(0); S3.INIT(0); S4.INIT(0)		
Vater: *S1.P()* *im Bad* *S1.V()* *bereite Kakao zu* *S2.V()* *S4.P()* *bring Kind zur Schule*	*Mutter:* *S1.P()* *im Bad* *S1.V()* *schmiere Brötchen* *S3.V()*	*Kind:* *S1.P()* *im Bad* *S1.V()* *S2.P()* *S3.P()* *frühstücke* *S3.V()*

- Wie würde sich die Lösung ändern, wenn es zwei Badezimmer gäbe?

Der erste Semaphor müsste mit 2 initialisiert werden – S1.INIT(2)

- Wir nehmen nun an, dass es ein zweites Kind gibt, das sich genau so wie das erste verhält. Wie müssen Vater und Mutter geändert werden?

Vater und Mutter müssen ihre Operationen auf S2, S3 und S4 jeweils zweimal aufrufen.

5. Drei Bauarbeiter (Anton, Bruno und Carl) ziehen zunächst ihre Arbeitskleidung an und brauchen dazu einen Umkleideraum. Dieser (einzige) Umkleideraum ist so klein, dass jeweils nur ein Arbeiter hineinpasst; in welcher Reihenfolge ihn die Arbeiter benutzen, ist egal. Wenn Anton sich umgezogen hat, holt er Hammer und Nägel; wenn Bruno sich umgezogen hat, holt er ein paar Bretter. Wenn Hammer, Nägel und Bretter da sind, beginnt Carl zu arbeiten. Anton und Bruno gehen währenddessen auf die Toilette, in die jeweils nur einer hineinpasst. Sie sollen nun diese Situation durch ein System von Prozessen modellieren, die sich mit Hilfe von Semaphoren synchronisieren.

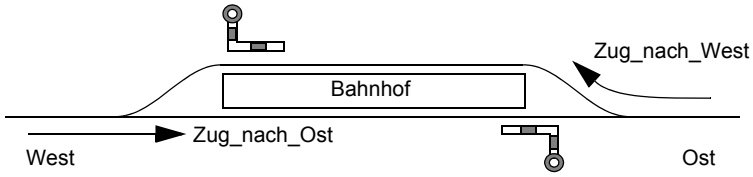
   - Was sind die Prozesse in diesem System? Welche Synchronisationsbedingungen sind im Einzelnen durchzusetzen? Geben Sie jeweils an, um welche allgemeine Art von Bedingung es sich handelt.

   5.)Prozesse: Anton, Bruno, Carl
   Synchronisationsbedingungen:
   1.) Umkleideraum kann nur von einem gleichzeitig benutzt werden (wechselseitiger Ausschluss)
   2.) Carl muss auf Hammer und Nägel von Anton warten (Reihenfolge)
   3.) Carl muss auf Bretter von Bruno warten (Reihenfolge)
   4.) Toilette kann nur von einem gleichzeitig benutzt werden (wechselseitiger Ausschluss)

   - Zeigen Sie nun, wie mit Hilfe von Semaphoren die Synchronisationsbedingungen durchgesetzt werden können.

Semaphore: S1.INIT(1); S2.INIT(0); S3.INIT(0); S4.INIT(1)		
Anton:	*Bruno:*	*Carl:*
S1.P()	*S1.P()*	*S1.P()*
umziehen	*umziehen*	*umziehen*
S1.V()	*S1.V()*	*S1.V()*
Hammer und Nägel holen	*Bretter holen*	*S2.P()*
S2.V()	*S3.V()*	*S3.P()*
S4.P()	*S4.P()*	*arbeiten*
Toilette	*Toilette*	
S4.V()	*S4.V()*	

6. Betrachten Sie die folgende eingleisige Eisenbahnstrecke mit Ausweichstelle:



   Aus West und Ost wird jeweils ein Zug erwartet, der auf ein vorgegebenes Gleis geleitet wird (siehe Pfeile,

also feste Weichenstellungen). Er findet auf seinem Gleis ein Signal vor, das ihn gegebenfalls stoppt. Ein Zug darf erst weiterfahren, wenn der Gegenzug in den Bahnhof eingelaufen ist.

Modellieren Sie dieses System durch zwei Prozesse *Zug_nach_Ost* und *Zug_nach_West*, die sich über Semaphore synchronisieren.

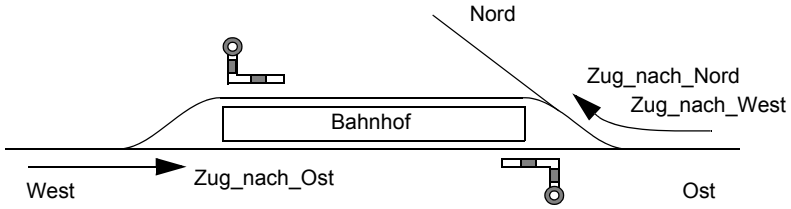
6.)

Semaphore: Sig_nach_Ost.INIT(0); Sig_nach_West.INIT(0)	
Zug_nach_West: *in Bahnhof einfahren* *Sig_nach_Ost.V(1)* *Sig_nach_West.P(1)* *weiterfahren*	*Zug_nach_Ost:* *in Bahnhof einfahren* *Sig_nach_West.V(1)* *Sig_nach_Ost.P(1)* *weiterfahren*

Stehen in ihrer Lösung die Signale auf "Halt" oder "Weiterfahrt", wenn die Züge den Bahnhof wieder verlassen haben?

auf "Halt", denn die P-Operationen zählen die Semaphorwerte wieder auf 0 herunter

Betrachten Sie nun die folgende Erweiterung: Aus Ost wird ein zweiter Zug erwartet, der ohne Halt nach Nord weiterfährt.



Wie muss Ihre Lösung erweitert werden, so dass *Zug_nach_Ost* erst weiterfährt, wenn beide anderen Züge die Strecke aus Osten verlassen haben? Führen Sie einen dritten Prozess *Zug_nach_Nord* ein, geben Sie an, was er tut, und geben Sie auch an, wie sich *Zug_nach_Ost* ändert. Sie müssen sich dabei nicht um die Weichenstellung und die Koordination zwischen *Zug_nach_Nord* und *Zug_nach_West* kümmern.

Zug_nach_West unverändert
Zug_nach_Nord: auf das Gleis Richtung Nord fahren; Sig_nach_Ost.V(1); weiterfahren;
Zug_nach_Ost: ... wie bisher ...; Sig_nach_Ost.P(2); weiterfahren

7. In einer Behörde füllen Antragsteller Anträge aus und reichen sie bei einem Sachbearbeiter ein. Die genaue Vorgehensweise ist dabei wie folgt:

   Bei Öffnung der Behörde ist eine bestimmte Anzahl von Stiften (anz_stifte) und eine bestimmte Anzahl von leeren Formularen (anz_formulare) vorhanden. Ein Antragsteller, der die Behörde besucht, führt folgende Tätigkeiten aus: Er nimmt einen Stift und ein leeres Formular, wobei er zunächst warten muss, wenn gerade kein Stift frei bzw. kein Leerformular mehr vorhanden ist. Nachdem er das Formular ausgefüllt hat, legt er den Stift zurück und legt das ausgefüllte Formular bei einem Sachbearbeiter auf einen Stapel. Anschließend verläßt er die Behörde; er wartet also nicht die Bearbeitung des Antrags ab.

Der Sachbearbeiter nimmt jeweils ein ausgefülltes Formular vom Stapel und bearbeitet es. Ist der Stapel leer, so macht er eine Pause, bis wieder ein Formular bei ihm abgegeben wird. Neben dem Sachbearbeiter gibt es noch einen Boten, der in bestimmten Zeitabständen jeweils anz_formulare leere Formulare nachlegt.

```
7.)Initialisierung der Semaphore:
SEM_WA_STIFT.INIT(anz_stifte);  /* Semaphor für Stifte */
SEM_LEERFORM(anz_formulare);  /* Semaphor für Leerformulare */
SEM_VOLLFORM(0);  /* Semaphor für ausgefüllte Formulare */

Antragsteller-Prozess:
SEM_WA_STIFT.P();  /* Stift nehmen */
SEM_LEERFORM.P();  /* leeres Formular nehmen */
... Formular ausfüllen ...
SEM_WA_STIFT.V();  /*Stift freigeben */
SEM_VOLLFORM.V();  /* ausgefülltes Formular auf Stapel legen */

Sachbearbeiter-Prozess:
while (true) {
 SEM_VOLLFORM.P();  /* ausgefülltes Formular vom Stapel nehmen */
 ... Antrag bearbeiten ...
}

Boten-Prozess:
while (true) {
 ... neue Formulare holen ...
 for (i=0;i<anz_formulare;i++)
   SEM_LEERFORM.V();  /* Leerformular nachlegen */
}
```

Erweitern Sie Ihre Lösung wie folgt: Manche Antragsteller (aber nicht alle) sind Diebe, die ihren Stift nicht wieder zurücklegen, sondern mitnehmen. Der Boten legt daher bei jedem Besuch zwei neue Stifte nach.

```
Antragsteller-Prozess:
SEM_WA_STIFT.P();  /* Stift nehmen */
SEM_LEERFORM.P();  /* leeres Formular nehmen */
... Formular ausfüllen ...
if (!dieb)
   SEM_WA_STIFT.V();  /*Stift freigeben */
SEM_VOLLFORM.V();  /* ausgefülltes Formular auf Stapel legen */

Boten-Prozess:
while (true) {
 ... neue Formulare holen ...
 for (i=0;i<anz_formulare;i++)
   SEM_LEERFORM.V();  /* Leerformular nachlegen */
 SEM_WA_STIFT.V();  /* neue Stifte nachlegen */
 SEM_WA_STIFT.V();
}
```

8. In dieser Aufgabe soll die Essensausgabe der Mensa durch Prozesse modelliert werden, die sich über Sem-

aphore synchronisieren. Geben Sie jeweils zunächst an, welche Semaphore benutzt werden (inkl. ihrer Anfangswerte). Skizzieren Sie dann (wie seinerzeit in der Übung) jeweils die Ablaufschritte der einzelnen Prozesse mit den P- und V-Operationen.

- Schreiben Sie als erstes den Rahmen des Programms für einen Gast, der an die Essensausgabe herantreten (in der Mensa gibt es beliebig viele Gäste, von denen jeder dieses Programm ausführt). In dem Programmrahmen soll zunächst nur sichergestellt werden, dass sich höchstens ein Gast gleichzeitig an der Essensausgabe befindet.

8.)Semaphor "WA", der mit 1 vorbesetzt ist
Gast:
WA.P()
Essen entgegennehmen
WA.V()

- Am Füllen eines Tellers sind drei Küchenhilfen beteiligt: Die erste schaufelt Kartoffeln auf den Teller, die zweite legt danach ein Schnitzel dazu und die dritte gießt schließlich Soße darüber. In den drei entsprechenden Prozessen, die nun zu schreiben sind, soll diese zeitliche Reihenfolge durchgesetzt werden.

Zwei Semaphore "Reihe12" und "Reihe23", die mit 0 vorbesetzt sind.

Küchenhilfe 1:	*Küchenhilfe 2:*	*Küchenhilfe 3:*
Kartoffeln auf Teller	*Reihe12.P()*	*Reihe23.P()*
Reihe12.V()	*Schnitzel auf Teller*	*Soße auf Teller*
	Reihe23.V()	

- Erweitern Sie jetzt Ihre Lösung so, dass Gäste und Küchenhilfen zeitlich synchronisiert werden:
  Die erste Angestellte beginnt mit dem Füllen eines Tellers erst, wenn ein Gast da ist; ansonsten blokkiert sie, bis ein Gast kommt.
  Ein Gast entblockiert die erste Angestellte, wenn er kommt, und muss seinerseits blockieren, bis die dritte Angestellte den Füllvorgang abgeschlossen hat.
  Eine Angestellte soll ihren Arbeitsgang beliebig oft hintereinander durchführen können (Schleife benutzen!).
  Damit sich die Angestellten nicht überarbeiten, soll jede von ihnen nach Abschluss eines Arbeitsgangs 15 Sekunden schlafen, bevor sie sich um den nächsten Teller bemüht.

Zwei weitere Semaphore "GastDa" und "EssenDa", die mit 0 vorbesetzt sind.

Gast:	*Küchenhilfe 1:*	*Küchenhilfe 2:*	*Küchenhilfe 3:*
GastDa.V()	*while (true) {*	*while (true) {*	*while (true) {*
WA.P()	*GastDa.P()*	*Reihe12.P()*	*Reihe23.P()*
EssenDa.P()	*Kartoffeln auf Teller*	*Schnitzel auf Teller*	*Soße auf Teller*
Essen nehmen	*Reihe12.V()*	*Reihe23.V()*	*EssenDa.V()*
WA.V()	*sleep(15) }*	*sleep(15) }*	*sleep(15) }*

9. In dieser Aufgabe geht es um die Synchronisation dreier Ballspieler: Die Spieler (Anton, Berta und Chris) stehen im Dreieck. Zu Beginn hat Anton den Ball und spielt ihn zu Berta, Berta spielt ihn dann zu Chris, dieser wieder zurück zu Anton usw. Jeder hält dabei den Ball für eine Sekunde lang fest. Wenn ein Spieler den Ball zehnmal hatte, geht er duschen. Da nur eine Dusche vorhanden ist, wird eine Reihenfolge festgelegt: Zuerst duscht Berta, dann Anton und dann Chris, wozu sie jeweils zehn Minuten brauchen. Zu Beginn ist die Dusche frei, und sie wird von keinen anderen Leuten benutzt.

- Skizzieren Sie für die drei Spieler je einen Prozess, in denen die geschilderten Bedingungen mit Semaphoroperationen durchgesetzt werden. Geben Sie dabei an, welche Semaphoren benutzt werden, wozu sie dienen und mit welchem Anfangswert sie vorbesetzt werden.

9.)Semaphore:
AB.INIT(1): AB läßt Anton auf den Ball warten
BB.INIT(0): BB läßt Berta auf den Ball warten
CB.INIT(0): CB läßt Chris auf den Ball warten
AD.INIT(0): AD läßt Anton auf die Dusche warten
CD.INIT(0): CD läßt Chris auf die Dusche warten

Anton:
for (i=0;i<10;i++) {
AB.P(1); warten auf Ball
sleep(1); Ball eine Sekunde lang festhalten
BB.V(1); Ball an Berta weitergeben
}
AD.P(1); auf Dusche warten
sleep(600); duschen
CD.V(1); Dusche für Chris freigeben

Berta:
for (i=0;i<10;i++) {
BB.P(1); warten auf Ball
sleep(1); Ball eine Sekunde lang festhalten
CB.V(1); Ball an Chris weitergeben
}
sleep(600); duschen
AD.V(1); Dusche für Anton freigeben

Chris:
for (i=0;i<10;i++) {
CB.P(1); warten auf Ball
sleep(1); Ball eine Sekunde lang festhalten
AB.V(1); Ball an Anton weitergeben }
}
CD.P(1); auf Dusche warten
sleep(600); duschen

- Erweitern Sie Ihre Lösung um einen Wärter (= einen neuen Prozess), der zuerst von allen drei Spielern je eine Mark kassiert, bevor er Berta die Dusche aufschließt.
  Hinweis: Berta kann nicht duschen, bevor der Wärter die Dusche aufgeschlossen hat, und der Wärter schließt nicht auf, bevor er das Geld erhalten hat. Sie müssen also auch die bereits vorhandenen Prozesse ergänzen und neue Semaphore einführen.

Semaphore:
GELD.INIT(0): blockiert den Wärter, bis das Geld da ist
DUSCHE.INIT(0): läßt Berta auf die Dusche warten

Anton, Chris:

for (...) { ... wie bisher ... }
GELD.V(1); Geld bezahlen
... weiter wie bisher ...

Berta:
for (...) { ... wie bisher ... }
GELD.V(1); Geld bezahlen;
DUSCHE.P(1); auf Aufschließen der Dusche warten
... weiter wie bisher ...

Wärter:
GELD.P(3); auf drei Mark warten
DUSCHE.V(1); Dusche aufschließen

3A.2 Programmierung unter UNIX/Linux

1. Betrachten Sie das folgende Teilstück eines UNIX-C-Programms:

```
int semid;
unsigned short initarray[2] = {1,0};
struct sembuf op0, op1;
semid = semget(IPC_PRIVATE,2,IPC_CREAT|0777);
semctl(semid,2,SETALL,initarray);
op0.sem_num = 0;  op1.sem_num = 1;
op0.sem_op = -1;  op1.sem_op = -1;
op0.sem_flg = -1; op1.sem_flg = 0;
semop(semid,&op0,1);
semop(semid,&op1,1);
```

- Was bewirken die Aufrufe von `semget` und `semctl` in diesem Programm?

1.)Der semget-Aufruf erzeugt eine neue Gruppe von 2 Semaphoren. Der semctl-Aufruf initialisiert den ersten Semaphor in dieser Gruppe mit dem Wert 1 und den zweiten mit dem Wert 0.

- Was ist der Effekt des ersten `semop`-Aufrufs für die Werte der Zählvariablen der Semaphore? Wird der Prozess blockiert oder läuft er weiter? (Wir gehen davon aus, dass andere Prozesse zwischenzeitlich keine semop-Aufrufe gemacht haben.)

Der Wert des ersten Semaphors wird um 1 gesenkt; der Prozess läuft weiter.

- Wie sieht es beim zweiten `semop`-Aufruf aus?

Der Prozess wird blockiert, die Werte beider Semaphore bleiben unverändert.

Das Programm wird nun wie folgt geändert:

```
int semid;
unsigned short initarray[2] = {1,0};
struct sembuf op[2];
semid = semget(IPC_PRIVATE,2,IPC_CREAT|0777);
semctl(semid,2,SETALL,initarray);
op[0].sem_num = 0;  op[1].sem_num = 1;
op[0].sem_op = -1;  op[1].sem_op = -1;
```

```
op[0].sem_flg = 0;  op[1].sem_flg = 0;
semop(semid,op,2);
```

- Was ist hier der Effekt des `semop`-Aufrufs für die Werte der Zählvariablen der Semaphore? Wird der Prozess blockiert oder läuft er weiter ?

Nach Änderung: Der Prozess wird blockiert, die Werte beider Semaphore bleiben unverändert.

2. Betrachten Sie das folgende Teilstück eines UNIX/Linux-C-Programms:

```
int semid;
unsigned short initarray[3] = {1,1,0};
struct sembuf operat[2];
semid = semget(IPC_PRIVATE,3,IPC_CREAT|0777);
semctl(semid,3,SETALL,initarray);
operat[0].sem_num = 0; operat[1].sem_num = 2;
operat[0].sem_op = operat[1].sem_op = -1;
operat[0].sem_flg = operat[1].sem_flg = 0;
semop(semid,operat,2);
```

- Was bewirken die Aufrufe von `semget` und `semctl` in diesem Programm?

2.)Der semget-Aufruf erzeugt eine neue Gruppe von 3 Semaphoren. Der semctl-Aufruf initialisiert den ersten und zweiten Semaphor in dieser Gruppe mit dem Wert 1 und den dritten mit dem Wert 0.

- Was ist der Effekt des semop-Aufrufs erstens für den ausführenden Prozess und zweitens für die Werte der Zählvariablen der Semaphore? (Wir gehen davon aus, dass andere Prozesse zwischenzeitlich keine `semop`-Aufrufe gemacht haben.)

Der Prozess wird blockiert, da die Zählvariable von Semaphor Nr. 2 (also dem dritten in dieser Gruppe) den Wert 0 hat und daher nicht weiter gesenkt werden kann. Die Werte sämtlicher Semaphore bleiben vorerst unverändert („Alles-oder-nichts-Prinzip“).

3. Gegeben ist das folgende Programmstück:

```
int s;
unsigned short anf[1];
struct sembuf op1, op2;
anf[0] = 0;
op1.sem_num = 0;   op2.sem_num = 0;
op1.sem_op  = 1;  op2.sem_op  = -1;
op1.sem_flg = 0;   op2.sem_flg = 0;
s = semget(IPC_PRIVATE,1,IPC_CREAT|0777);
semctl(s,0,SETALL,anf);
if (fork()==0) {
  ... Aktionen 1 ...
  semop(s,&op1,1);
  exit();
 }
if (fork()==0) {
  semop(s,&op2,1);
  ... Aktionen 2 ...
  exit();
 }
```

Wie viele Semaphore werden hier erzeugt? Kurze Begründung!

3.)einer (siehe zweiter Parameter von semget()).

Wie nennt man die Synchronisationsbedingung, die hier durchgesetzt wird? (ein Begriff)

Reihenfolge.

Welchen Wert hat/haben der/die Semaphor(e), wenn beide Sohnprozesse vollständig abgelaufen sind? Kurze Begründung!

0 (Semaphor wird mit 0 initialisiert, p1 zählt zunächst den Semaphorwert um 1 hinauf, p2 zählt ihn dann um 1 herunter).

Schreiben Sie eine C-Anweisung, mit der die Semaphorgruppe wieder gelöscht wird.

semctl(s,0,IPC_RMID,0)