Synchronisationssysteme

- PV_1-Systeme:
  - ➥ bestehend aus nur einer Komponente S:

Operations-bezeichnung	Vorbedingung	Effekt
P()	$S > 0$	$S \leftarrow S - 1$
V()	$S < 1$	$S \leftarrow S + 1$

- Up/Down-Systeme:

bestehend aus einem Komponentenvektor

Operations-bezeichnung	Vorbedingung	Effekt
$\{S_{i_r} \mid 1 \leq r \leq k\}: up(S_j)$	$\sum_{r=1}^{k} S_{i_r} \geq 0$	$S_j \leftarrow S_j + 1$
$\{S_{i_r} \mid 1 \leq r \leq k\}: down(S_j)$	$\sum_{r=1}^{k} S_{i_r} \geq 0$	$S_j \leftarrow S_j - 1$

- ➥ `shmflg`: legt die Abbildung fest:
  - SHM_RND: falls ein Adreßwunsch angegeben wurde, wird diese Adresse auf eine systemspezifische Seiten- bzw. Segmentgrenze gerundet (SHMLBA: shared memory low boundary address multiply)
  - SHM_RDONLY: Das Segment wird nur für lesenden Zugriff eingeblendet
- ➥ Returnwert: tatsächliche Adresse, an der das Segment eingeblendet wurde

◆ Aufheben der Einblendung:

```
int shmdt(void *shmaddr)
```

- ➥ `shmaddr`: Adresse des auszublendenden Segments

◆ Statuskontrolle und Löschen:

```
int shmctl(int shmid, int cmd,
              struct shmid_ds *buf)
```

- ➥ `shmid`: Identifier des shared memory Segments
- ➥ `cmd`: Kommando zum Ändern der Eigenschaften des Segmente, u.a.:
  - SHM_LOCK: sperrt das shared memory Segment
  - SHM_UNLOCK: hebt Sperre wieder auf
  - IPC_RMID: markiert Segment als gelöscht. Das Segment wird aber erst dann tatsächlich gelöscht, wenn der letzte Prozeß es aus seinem Adreßraum ausgeblendet hat.

Shared Memory

- Speicherbereich, auf den mehrere Prozesse gemeinsam lesend und schreibend zugreifen können
- Die beteiligten Prozesse müssen nicht miteinander verwandt sein
- Die Zugriffe werden vom System nicht koordiniert, d.h. die Koordinierung muß durch den Benutzer (z.B. mittels Semaphore) vorgenommen werden.
- Include-Dateien: `<sys/types.h>`
  `<sys/ipc.h>`
  `<sys/shm.h>`
- Systemschnittstelle:
  - Erzeugen eines neuen *shared memory* Segments, bzw. Zugriff auf ein bestehendes:
    ```
    int shmget(key_t key, size_t size, int shmflg)
    ```
    - ➥ `size`: Mindestgröße des *shared memory* Segments
    - ➥ Returnwert: *shared memory* Identifier
  - Einblenden des *shared memory* Segments in den virtuellen Adreßraum des Prozesses:
    ```
    char *shmat(int shmid, void *shmaddr,
                int shmflg)
    ```
    - ➥ `shmaddr`: Gewünschte Adresse, an der das Segment eingeblendet werden soll (0, falls kein Wunsch)

Semaphore

- Eine semop-Aufruf wird immer atomar ausgeführt, d.h. nur wenn alle geforderten Teiloperationen ausgeführt werden können, werden sie ausgeführt. Andernfalls bleiben alle Semaphorkomponenten unverändert und die Funktion blockiert, bzw. liefert einen Fehler.

Semaphore

- Semaphor-Operationen (Vektoraddition):

```
int semop(int semid, struct sembuf *sops,
          size_t nsops)
```

  - ➥ `semid`: Semaphoridentifier
  - ➥ `nsops`: Anzahl der in `sops` enthaltenen Operationen
  - ➥ `sops`: Vektor der die Semaphoroperation bestimmt. Jede Komponente dieses Vektors legt eine Teiloperation fest:

```
    struct sembuf {
       short sem_num;
       short sem_op;
       short sem_flg;
    };
```

    - `sem_num`: Nummer der Semaphorkomponente, auf die sich die Operation bezieht
    - `sem_op < 0` : P-Operation
      `sem_op` wird auf die Semaphorkomponente addiert, wenn dadurch die Komponente nicht kleiner Null wird. Sonst blockiert die Operation oder liefert einen Fehler, je nach Wert von `sem_flg`.
    - `sem_op = 0`: Test auf 0
      Die Operation testet, ob die Semaphorkomponente den Wert Null hat. Bei einem Wert ungleich Null blockiert die Operation, bzw. liefert einen Fehler.
    - `sem_op > 0`: V-Operation
      `sem_op` wird auf die Semaphorkomponente addiert. Diese Operation blockiert nicht.
    - `sem_flg`: Beeinflußt das Verhalten, z.B.:
      IPC_NOWAIT: die Operation blockiert nicht, sondern liefert ggf. einen Fehler

Semaphore

- Vektoradditionssystem
  - ➥ mehrere Semaphor-Komponenten können mit einer Operation getestet und individuell dekrementiert (“P”) bzw. inkrementiert (“V”) werden

- Include-Dateien: `<sys/types.h>` `<sys/ipc.h>` `<sys/sem.h>`

- Systemschnittstelle:
  - ◆ Erzeugen eines neuen Semaphorvektors, bzw. Zugriff auf einen bestehenden:
    `int semget(key_t key, int nsems, int semflg)`
    - ➥ `nsems`: Anzahl der Komponenten, die der Semaphorvektor umfassen soll
    - ➥ Returnwert: global gültiger Semaphoridentifier
  - ◆ Statuskontrolle, Initialisierung und Löschen:
    `int semctl(int semid, int semnum, int cmd, ...)`
    - ➥ `semid`: Semaphoridentifier
    - ➥ `semnum`: Nummer der betroffenen Sem.-Komponente
    - ➥ `cmd`: Angabe des Kommandos, u.a.:
      - GETVAL: Wert der Komponente zurückliefern
      - SETVAL: Komponente auf Wert des optionalen vierten Parameters setzen
      - GETALL: Liefert die Werte aller Komponenten
      - SETALL: Setzt die Werte aller Komponenten

- Flags:
  IPC_CREAT: Erzeugen eines Mechanismus
  IPC_EXCL: Fehler falls Mech. schon existiert

- jedem Mechanismus ist ein control-Systemaufruf zugeordnet
  - Beispiel:
    `int semctl(int semid, int semnum, int cmd, ...)`
    - `semid`: allgemein ein Identifier für den Mechanismus, hier für die Semaphore
    - `semnum`: semaphorspezifischer Parameter
    - `cmd`: Angabe des zu erfolgenden Kommandos:
      - IPC_STAT: Abfrage von Zustandsinformationen des Mechanismus
      - IPC_SET: Setzen einzelner Zustandsinformationen
      - IPC_RMID: Löschen des Mechanismus

- **Kommandos** zum Abfragen und Beeinflussen von IPC-Mechanismen:
  - ipcs(1): Anzeige von Statusinformation über die Mechanismen
  - ipcrm(2): Löschen von IPC-Einträgen

IPC-Verwaltungsmechanismen

- jedem IPC-Mechanismus ist eine Tabelle zugeordnet, deren Einträge (siehe **intro(2)**) die einzelnen Message Queues, Shared Memory Segmente bzw. Semaphore beschreiben

- jedem Eintrag ist ein numerischer Key zugeordnet, über den der Eintrag angesprochen werden kann
  - mit Hilfe der Funktion **ftok()** kann aus einem Dateinamen ein Key erzeugt werden
  - Keys können aber auch frei gewählt werden

- Jedem Mechanismus ist ein **get**-Systemaufruf zugeordnet zum Erzeugen neuer und zum Zugriff auf existierende Einträge
  - Beispiel:
    `int semget(key_t key, int nsems, int semflg)`
    - ➥ `key`: numerischer Schlüssel für den IPC-Eintrag
      (kann auch dem Wert IPC_PRIVATE entsprechen)
    - ➥ `nsems`: spezifischer Wert für Semaphore
    - ➥ `semflg`: Mischung aus Flags und Zugriffsrechten:
      - Zugriffsrechte ähnlich denen für Dateien:

0400	READ by user
0200	ALTER by user
0040	READ by group
0020	ALTER by group
0004	READ by others
0002	ALTER by others