Inhalt

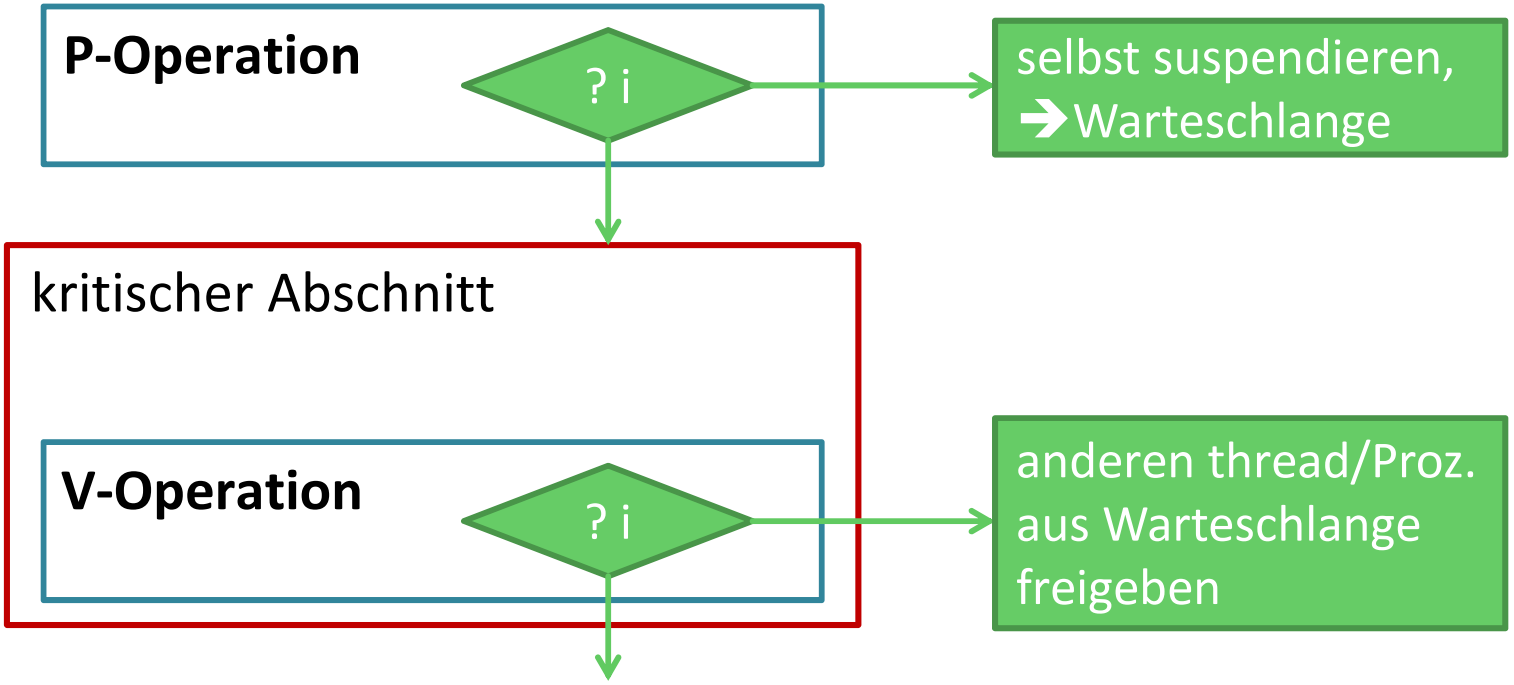
- Synchronisation mit Semaphoren

Semaphoren

- Zur Sperr- und Reihenfolgensynchronisation werden sog. Semaphoren eingesetzt.
- Begriff Semaphor:
  - griechisch: ~Zeichenträger
  - das Semaphor: Signalmast, (auch franz.: Leuchtturm)
  - der Semaphor, österreichisch
- Das Prinzip wurde entwickelt von E.W. Dijkstra 1965 entwickelt.
- Man unterscheidet
  - unbenannte Semaphoren (thread-Synchronisation)
  - benannte Semaphoren (Prozess-Synchronisation)

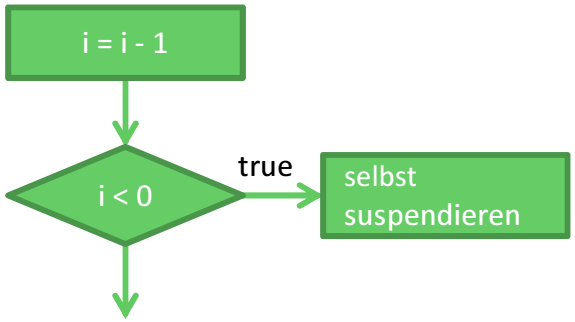
Ein Semaphor besteht aus:

- einer **Zählvariablen i**
- der **P-Operation** (niederl.: *passeren*, *probeeren*)
- der **V-Operation** (niederl.: *vrijgeven*, *verhogen*)
- einer **Liste/Schlange** von Prozessen/threads, die auf die V-Operation (Freigabe) warten.



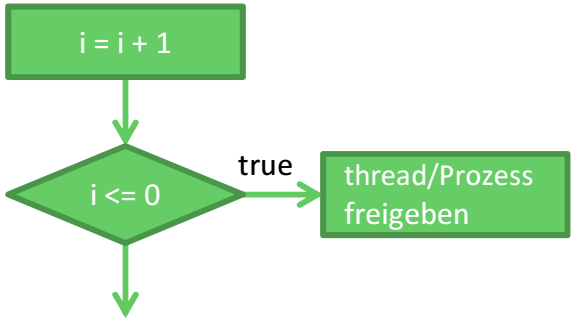
P-Operation:

- Zählvariable dekrementieren
- falls i < 0, diesen thread/Prozess blockieren und in Warteschlange einfügen



V-Operation:

- Zählvariable inkrementieren
- falls i <= 0, einen anderen thread/Prozess aus der Warteschlange freigeben



Zählvariable i:

- der **positive Wert von i** gibt die Anzahl der threads/Prozesse an, die gleichzeitig „passieren“ dürfen
- der **negative Wert von i** gibt die Anzahl der threads/Prozesse an, die auf die Freigabe dieses Semaphors warten

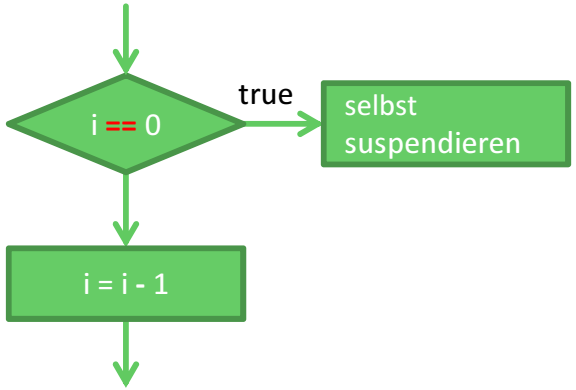
Die P- und V-Operationen müssen atomar ausgeführt werden!

POSIX Semaphoren

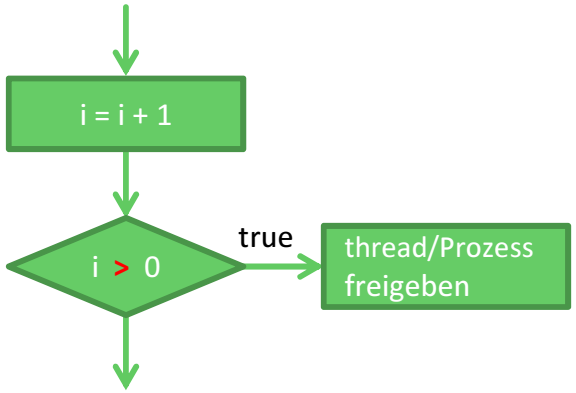
- Variable vom Typ **`sem_t`** (in `semaphore.h`)
- der Zählerwert ist immer >= 0

  ➔ die Anzahl der wartenden threads kann nicht abgefragt werden

POSIX-P-Operation **`sem_wait`**:



POSIX-V-Operation **`sem_post`**:



- die P-Operation lautet: **`sem_wait(&semX)`**

- die V-Operation lautet: **`sem_post(&semX)`**

- benannte Semaphoren sind im System bekannt
  ➔ Prozesskommunikation

- unbenannte Semaphoren sind nur innerhalb eines Prozesses bekannt
  ➔ thread-Kommunikation

- die **`sem_trywait(&semX)`** Funktion blockiert nicht, sondern kehrt bei „gesperrt“ mit -1 zurück

Beispiele

- Sperrsynchronisation, Semaphor S als Mutex:
  - Initialwert **i = 1**, d.h. *frei*
  - die erste P-Operation bekommt Zugang
  - jeder thread führt P(S)...V(S) aus

- Reihenfolgensynchronisation
  - mehrere Semaphoren werden **wechselseitig** belegt
  - Beispiel, zwei threads abwechselnd ausführen:
    - zwei Semaphoren S1, S2 jeweils mit **i = 0** initialisieren
    - die threads starten mit P(S1) bzw. P(S2) ➔ beide blockieren
    - dritter thread (z.B. main) gibt Startsignal mit V(S1) oder V(S2)
    - die threads führen fortlaufend P(S1)... V(S2) und P(S2)... V(S1) aus und wechseln sich dadurch ab

POSIX Beispiel: `main` steuert `faden` mit `semA`

```
sem_t semA;
int main()
{
  pthread_t faden;
  sem_init(&semA, 0, 1);
  pthread_create(&faden, NULL,
                 machWatt, NULL);
  int i;
  for(i = 1; i <= 25; i++)
  {
    sem_wait(&semA);
    schlafZufaellig(50, 1000);
    sem_post(&semA);

    // dem anderen eine Chance ...:
    usleep(10 * 1000);
  }
  return 0;
}
```

i=1 → frei

sperren

freigeben

```
void* machWatt()
{
  char c = 'A';
  while(1)
  {
    sem_wait(&semA);
    printf("%c ", c++);
    fflush(stdout);
    sem_post(&semA);

    if(c > 'Z')
        c = 'A';

    // dem anderen eine Chance ...:
    usleep(10 * 1000);
  }

  return NULL;
}
```

main-Freigabe abwarten

Kontrolle an main abgeben

→ `faden` bekommt von **`main`** zufällige Zeitabschnitte zur Ausführung zugewiesen

Reihenfolgensynchronisation, zwei threads wechseln sich ab: