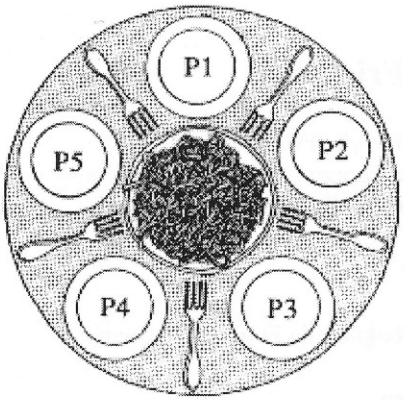

Das Philosophenproblem

```
#define N 5                     /* Anzahl der Philosophen      */

void philosopher(int i)         /* i:0..N-1, welcher Philosoph */
{
   while (TRUE) {
      think();                  /* Denken                      */
      take_fork(i);             /* Greife linke Gabel          */
      take_fork((i+1)%N);       /* Greife rechte Gabel         */
      eat();                    /* Essen                       */
      put_fork(i);              /* Ablegen linke Gabel         */
      put_fork((i+1)%N);        /* Ablegen rechte Gabel        */
   }
}
```

Annahme: Alle Philosophen greifen die linke Gabel durch `take_fork(i)`.

Dann sind alle für immer blockiert. Es liegt ein sogenannter Deadlock vor

```
#define N 5                     /* Anzahl der Philosophen      */
semaphore mutex = 1;

void philosopher (int i)        /* i:0..N-1, welcher Philosoph */
{
   while (TRUE) {
      think();                  /* Denken                      */
      down(&mutex);
      take_fork(i);             /* Greife linke Gabel          */
      take_fork((i+1)%N);       /* Greife rechte Gabel         */
      eat();                    /* Essen                       */
      put_fork(i);              /* Ablegen linke Gabel         */
      put_fork((i+1)%N);        /* Ablegen rechte Gabel        */
      up(&mutex)
   }
}
```

unbefriedigend: nur ein Philosoph kann gleichzeitig essen !

Semaphoren

- Jeder Prozeß führt die Prozedur *philosopher* als Hauptprogramm aus, die anderen Prozeduren sind gewöhnliche Unterprogramme, keine separaten Prozesse.

- Das array *state* speichert die aktuellen Zustände der Philosophen: essend, denkend, hungernd (versucht, Gabeln zu bekommen).

- Jeder Philosoph blockiert an einem ihm zugeordneten Semaphor *s[i]*, wenn die benötigten Gabeln nicht verfügbar sind.

- Das Semaphor *mutex* sichert den kritischen Abschnitt der Benutzung der Zustandsinformation.

- Die Lösung ist korrekt, sie enthält keinen Deadlock und kein Verhungern.

- Die Lösung läßt eine möglichst hohe Nebenläufigkeit zwischen den Philosophen zu.

- Die Lösung ist für eine beliebige Anzahl von Philosophen korrekt.

```
#define N           5           /* Anzahl der Philosophen            */
#define LEFT  (i-1)%N           /* Nummer des linken Nachbarn von i  */
#define RIGHT (i+1)%N           /* Nummer des rechten Nachbarn von i */
#define THINKING    0           /* Zustand: Denkend                  */
#define HUNGRY      1           /* Zust: Versucht, Gabeln zu bekommen*/
#define EATING      2           /* Zustand: Essend                   */

/* gemeinsame Variablen                                              */
int state[N];                   /* Zustaende aller Philosophen       */
semaphore mutex = 1;            /* fuer wechselseitigen Ausssschluss */
semaphore s[n];                 /* Semaphor fuer jeden Philosoph     */

void philosopher(int i) {       /* i:0..N-1, welcher Philosoph       */
   while (TRUE) {
      think();                  /* Denken                            */
      take_forks(i);            /* Greife beide Gabeln oder blockiere*/
      eat();                    /* Essen                             */
      put_forks(i);             /* Ablegen beider Gabeln             */
   }
}
void take_forks(int i) {        /* i:0..N-1, welcher Philosoph       */
   down(&mutex);                /* tritt in krit. Bereich ein        */
   state[i] = HUNGRY;           /* zeige, dass du hungrig bist       */
   test(i);                     /* versuche, beide Gabeln zu bekommen*/
   up(&mutex);                  /* verlasse krit. Bereich            */
   down(&s[i]);                 /* bockiere, falls Gabeln nicht frei */
}

void put_forks(int i) {         /* i:0..N-1, welcher Philosoph       */
   down(&mutex);                /* tritt in krit. Bereich ein        */
   state[i] = THINKING;         /* zeige, dass du fertig bist        */
   test(LEFT);                  /* kann linker Nachbar jetzt essen ? */
   test(RIGHT);                 /* kann rechter Nachbar jetzt essen ?*/
   up(&mutex);                  /* verlasse krit. Bereich            */
}

void test(int i) {              /* i:0..N-1, welcher Philosoph       */
   if (state[i]== HUNGRY && state[LEFT]!=EATING && state[RIGHT]!=EATING)
   {
      state[i]=EATING;          /* jetzt kann Phil i essen !         */
      up(&s[i]);                /* "sage es ihm"                     */
   }
}
```