

```
1 /*
2  Uppgiften är att synkronisera koden och
3  undvika alla "busy-wait" som kan uppstå.
4  Din lösning ändrar eller ersätter relevant
5  kod med att använda lås och semaforer
6  (conditions om du vill).
7
8  Det finns:
9  - Tio filosofer.
10 - Ett runt bord med fem platser.
11 - En skål mat i mitten på bordet.
12 - Två servitörer som med jämna mellanrum
13   fyller på skålen.
14 - En tallrik per plats och en gaffel mellan
15   varje plats.
16
17 Krav:
18 - Varje filosof behöver en plats, vilken som.
19 - En filosof behöver just de gafflar som
20   finns på vardera sida om platsen.
21 - Det går inte att ta mat om skålen är tom.
22 - Är skålen tom lämnar filosofen bordet
23   omedelbart.
24 - En filosof skall kunna äta samtidigt som de
25   andra filosoferna.
26
27 OBS: De funktionerna som ej är implementerade
28   (endast deklarerade) är trådsäkra.
29
30 Tänk även på att undvika deadlock!
31 */
```

```

1
2 // begin of dummy declarations that are only
3 // designed to get the code through compiler
4 typedef enum      {false , true } bool;
5 struct semaphore { int count; };
6 struct lock      { int owner; };
7 struct condition { int placeholder; };
8
9 // The five functions are thread-safe.
10 void cook_more_rations();
11 void eat_randomly_long_time();
12 void sleep_randomly_long_time();
13 void swap(int*, int*);
14
15 void sema_init(struct semaphore*, int);
16 void sema_down(struct semaphore*);
17 void sema_up(struct semaphore*);
18
19 void lock_init(struct lock*);
20 void lock_acquire(struct lock*);
21 void lock_release(struct lock*);
22
23 void cond_init (struct condition*);
24 void cond_wait (struct condition*, struct lock*);
25 void cond_signal (struct condition*, struct lock*);
26
27 int atomic_swap(int*, int);
28 int test_and_set(int *);
29 int compare_and_swap(int *, int , int);
30 // end of dummy declarations

```

```

1 bool seat_taken[5];
2 bool fork_taken[5];
3 int rations_of_food = 0;
4
5 void init() // executed before threads
6 {
7     for (int i = 0; i < 5; ++i)
8     {
9         seat_taken[i] = false;
10        fork_taken[i] = false;
11    }
12 }
13
14 void clerk_thread() // two of those
15 {
16     for ( ; ; )
17     {
18         // Add more food to table
19         rations_of_food = rations_of_food + 1;
20         cook_more_rations();
21     }
22 }

```

```

1 void philosopher_thread() // ten of those
2 {
3     for ( ; ; )
4     {
5         int seat_no = 0;
6
7         // Find a free seat.
8         while ( seat_taken[seat_no] )
9             seat_no = (seat_no + 1) % 5;
10        seat_taken[seat_no] = true;
11
12        // Grab two forks.
13        int fork1 = seat_no;
14        int fork2 = (seat_no + 1) % 5;
15
16        while ( fork_taken[fork1] )
17            ;
18        fork_taken[fork1] = true;
19
20        while ( fork_taken[fork2] )
21            ;
22        fork_taken[fork2] = true;
23
24        // Check for food. Eat, or leave hungry.
25        if ( rations_of_food > 0)
26        {
27            rations_of_food = rations_of_food - 1;
28            eat_randomly_long_time();
29        }
30
31        // Replace forks and leave seat.
32        fork_taken[fork1] = false;
33        fork_taken[fork2] = false;
34        seat_taken[seat_no] = false;
35
36        // Digest the food.
37        sleep_randomly_long_time();
38    }
39 }

```