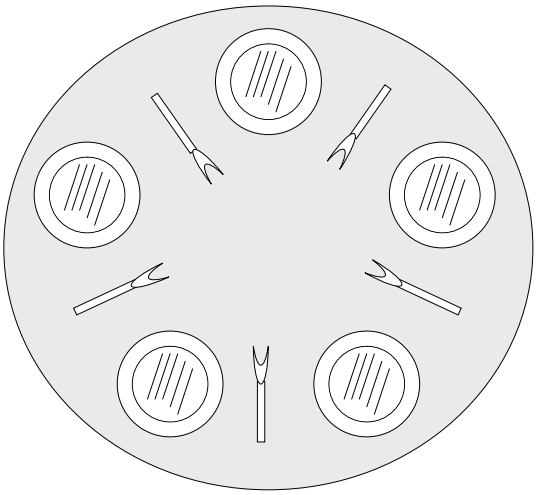
Classical IPC and Synchronization Problems

CS 342 – Operating Systems

Ibrahim Korpeoglu
Bilkent University
Computer Engineering Department

Dining Philosophers Problem

- 5 philosophers want to eat dinner (spaghetti)
- There are 5 plates and 5 forks on the table
- Each philosopher needs to use two forks to eat.
- When it want to read, he need to pickup left and right forks one at a time in any order.



Dining Philosophers Problem

- **Problem**: Write a program for each philosopher that simulates the situation and never get stuck?
  - A philosopher is either thinking or eating.
  - Before it eats, it has two acquire two forks.

Initial attempt for solution

```
#define N 5

void philosopher (int i)
{
        while (TRUE) {
                think();            /* just think – not eat */
                take_fork(i);       /* take left fork */
                take_fork( i + 1 % N);     / * take right fork */
                eat();                     /* eat the spagetti */
                put_fork(i);
                put_fork(i + 1 % N);
}
```

- What if all five philosophers take their left fork at the same time?
- This solution does not work!

Improvement

```
#define N 5
semaphore  mutex;

void philosopher (int i)
{
        while (TRUE) {
                think();            /* just think – not eat */
                down (mutex);
                take_fork(i);       /* take left fork */
                take_fork( i + 1 % N);     / * take right fork */
                eat();                     /* eat the spagetti */
                put_fork(i);
                put_fork(i + 1 % N);
                up (mutex);
}
```

- Use semaphore.
- Problem: only one philosopher can eat, although it is possible two of them to east simultaneously.

Solution

```
#define N    5
#define LEFT      (i+N-1)%N    /* number of i's left neighbor */
#define RIGHT     (i+1)%N      /* number of i's right neighbor */
#define THINKING  0
#define HUNGRY    1
#define EATING    2

typedef int semaphore;
int state[N];              /* state of each philosopher */
semaphore mutex = 1; /*mutual exlusion for critical regions */
semaphore s[N];         /* one semaphore per philosopher */
```

```
void philosopher (int i)      /* i is between 0 and N-1 */
{
        while (TRUE) {
                think(i);
                take_forks(i);  /* acquire two forks or block */
                eat();
                put_forks(i);    /* put both forks on the table */
        }
}

void take_forks (int i)
{
        down(&mutex);              /* enter critical region */
        state[i] = HUNGRY;         /* express interest to eat  */
        test(i);                   /* try to acquire two forks */
        up(&mutex);
        down(&s[i]);               /* block if forks were not acquired */
}
```



```
void put_forks (i)
{
        down(&mutex);
        state[i] = THINKING;
        test[(LEFT]);
        test(RIGHT);
        up(&mutex);
}

void test (i)
{
        if (state[i] == HUNGRY &&
            state[LEFT] != EATING &&
            state[RIGHT] != EATING)
        {
                state[i] = EATING;
                up(&s[i]);
        }
}
```


1
thinking
hungry
eating
2
thinking
hungry
eating
3
thinking
hungry
eating
4
thinking
hungry
eating
5
thinking
hungry
eating