《操作系统》

进程与线程

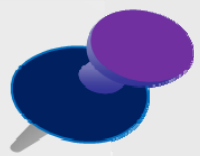
首都师范大学
信息工程学院
霍其润

Processes and Threads

- Processes
- Threads
- Interprocess communication
- Classical IPC problems
- Scheduling

CLASSICAL IPC PROBLEMS

- The Dining Philosophers Problem
- The Readers and Writers Problem

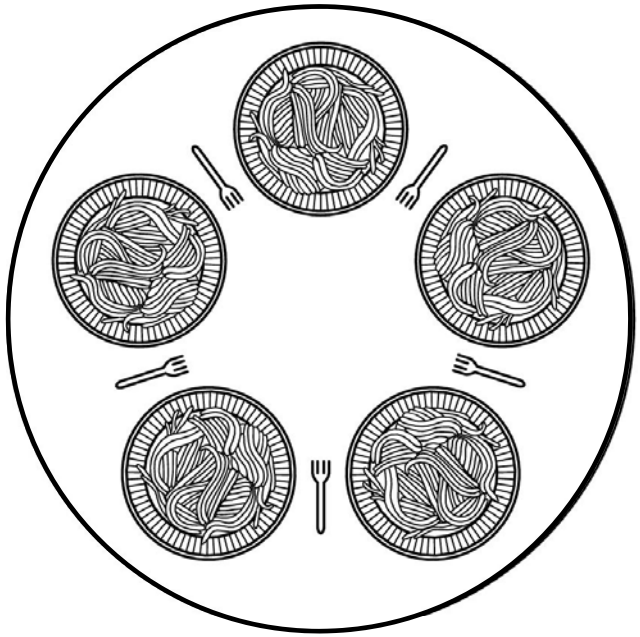


四 经典IPC问题
(Classical IPC problems)

1、哲学家餐桌问题

Dining Philosophers (1)

- Philosophers eat/think
- Eating needs 2 forks
- Pick one fork at a time
- How to do



Dining Philosophers (2)

```
#define N 5                                     /* number of philosophers */

void philosopher(int i)                         /* i: philosopher number, from 0 to 4 */
{
      while (TRUE) {
            think( );                           /* philosopher is thinking */
            take_fork(i);                       /* take left fork */
            take_fork((i+1) % N);               /* take right fork; % is modulo operator */
            eat( );                             /* yum-yum, spaghetti */
            put_fork(i);                        /* put left fork back on the table */
            put_fork((i+1) % N);                /* put right fork back on the table */
      }
}
```

A nonsolution to the dining philosophers problem

Dining Philosophers (3)

```
#define N                5                  /* number of philosophers */
#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                  /* philosopher is thinking */
#define HUNGRY           1                  /* philosopher is trying to get forks */
#define EATING           2                  /* philosopher is eating */
typedef int semaphore;                      /* semaphores are a special kind of int */
int state[N];                               /* array to keep track of everyone's state */
semaphore mutex = 1;                        /* mutual exclusion for critical regions */
semaphore s[N];                             /* one semaphore per philosopher */

void philosopher(int i)                     /* i: philosopher number, from 0 to N–1 */
{
      while (TRUE) {                        /* repeat forever */
            think( );                       /* philosopher is thinking */
            take_forks(i);                  /* acquire two forks or block */
            eat( );                         /* yum-yum, spaghetti */
            put_forks(i);                   /* put both forks back on table */
      }
}
```

Solution to dining philosophers problem (part 1)

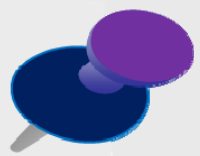
Dining Philosophers (4)

```
void take_forks(int i)                  /* i: philosopher number, from 0 to N–1 */
{
        down(&mutex);                   /* enter critical region */
        state[i] = HUNGRY;              /* record fact that philosopher i is hungry */
        test(i);                        /* try to acquire 2 forks */
        up(&mutex);                     /* exit critical region */
        down(&s[i]);                    /* block if forks were not acquired */
}

void put_forks(i)                       /* i: philosopher number, from 0 to N–1 */
{
        down(&mutex);                   /* enter critical region */
        state[i] = THINKING;            /* philosopher has finished eating */
        test(LEFT);                     /* see if left neighbor can now eat */
        test(RIGHT);                    /* see if right neighbor can now eat */
        up(&mutex);                     /* exit critical region */
}

void test(i)                            /* i: philosopher number, from 0 to N–1 */
{
        if (state[i] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING) {
                state[i] = EATING;
                up(&s[i]);
        }
}
```

Solution to dining philosophers problem (part 1)



四 经典IPC问题
(Classical IPC problems)

2、读者-写者问题

The Readers and Writers Problem (1)

- 问题描述：对共享资源的读写操作

  任一时刻“写者”最多只允许一个，而“读者”则允许多个

  - “读－写”互斥，
  - “写－写”互斥，
  - “读－读”允许

The Readers and Writers Problem (2)

```
typedef int semaphore;                  /* use your imagination */
semaphore mutex = 1;                    /* controls access to 'rc' */
semaphore db = 1;                       /* controls access to the database */
int rc = 0;                             /* # of processes reading or wanting to */

void reader(void)
{
    while (TRUE) {                      /* repeat forever */
        down(&mutex);                   /* get exclusive access to 'rc' */
        rc = rc + 1;                    /* one reader more now */
        if (rc == 1) down(&db);         /* if this is the first reader ... */
        up(&mutex);                     /* release exclusive access to 'rc' */
        read_data_base( );              /* access the data */
        down(&mutex);                   /* get exclusive access to 'rc' */
        rc = rc – 1;                    /* one reader fewer now */
        if (rc == 0) up(&db);           /* if this is the last reader ... */
        up(&mutex);                     /* release exclusive access to 'rc' */
        use_data_read( );               /* noncritical region */
    }
}

void writer(void)
{
    while (TRUE) {                      /* repeat forever */
        think_up_data( );               /* noncritical region */
        down(&db);                      /* get exclusive access */
        write_data_base( );             /* update the data */
        up(&db);                        /* release exclusive access */
    }
}
```

- 采用信号量机制：
  - db互斥信号量：实现读者与写者、写者与写者之间的互斥，初值是1。
  - rc读者共享的变量：表示“正在读”的进程数，初值是0；
  - mutex表示对rc的互斥操作，初值是1。

读者优先还是写者优先？

小 结

- 生产者-消费者问题
- 哲学家餐桌问题
- 读者-写者问题