Sistemas Operacionais I

Problemas Clássicos de IPC

Prof. Carlos Eduardo de B. Paes
Departamento de Ciência da Computação
Pontifícia Universidade Católica de São Paulo

SEMANA 6
Estrutura da Aula

- Problemas Clássicos de IPC
  - Problema dos Filósofos Jantando
  - Problema dos Leitores e Escritores
  - Problema do Barbeiro Dorminhoco

Problemas Clássicos de IPC

- A literatura de Sistemas Operacionais está repleta de problemas interessantes
- Problemas clássicos que envolvem comunicação e sincronização interprocessos
- Vamos estudas o problema geral e sua solução implementada em pseudo-C usando semáforos

Problemas Clássicos de IPC
Filósofos Jantando

- Problema de sincronização proposto por Dijkstra em 1965
- Cinco filósofos estão sentados em torno de uma mesa.
- Cada filósofo tem para si um prato de espaguete.
- O espaguete é tão escorregadio que cada filósofo necessita de dois garfos para comê-lo.

Problemas Clássicos de IPC
Filósofos Jantando

- Entre cada par de pratos existe um garfo, como mostra a figura:

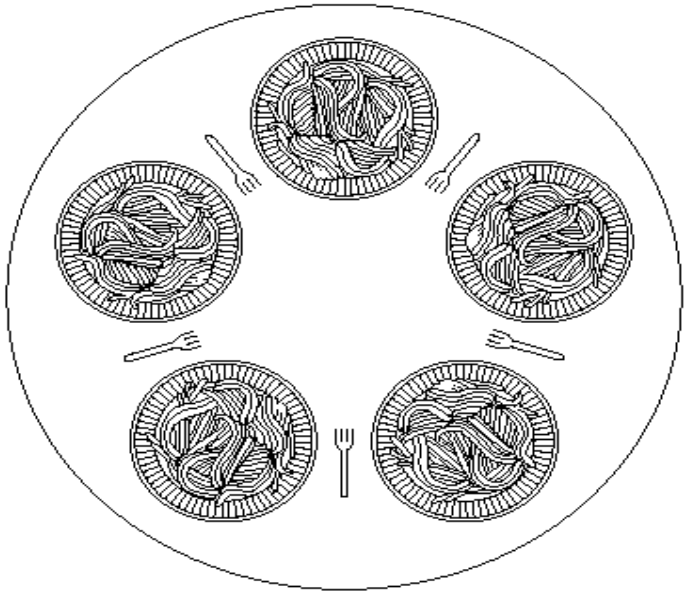


Figure 2-16. Lunch time in the Philosophy Department.

Problemas Clássicos de IPC
Filósofos Jantando

- A vida de um filósofo consiste em períodos alternados de comer e pensar.
- Filósofo com fome → tenta pegar o garfo da esquerda e da direita, um de cada vez, em qualquer ordem
- Filósofo come (durante um tempo) → se conseguir obter os dois garfos

Problemas Clássicos de IPC
Filósofos Jantando

- Solução obvia → procedimento *take_fork* espera até que o garfo especificado esteja disponível e, então, pega-o.

```
#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 * /
    }
}
```

Problemas Clássicos de IPC
Filósofos Jantando

- Problema com a solução anterior → os cinco filósofos pegam seus garfos esquerdos simultaneamente.
- Com isso nenhum filósofo será capaz de pegar seus garfos direito e haverá um deadlock

Problemas Clássicos de IPC

Filósofos Jantando

- Vamos tentar modificar o programa → depois de pegar o garfo esquerdo o filósofo verifica se o garfo direito está disponível
- Problema → esta solução também tem problemas
- Os filósofos podem iniciar o algoritmo simultaneamente, pegando seus garfos esquerdo, vendo que seus garfos direitos não estão disponíveis, colocando na mesa seus garfos esquerdo, esperando, e assim por diante, eternamente.

Problemas Clássicos de IPC

Filósofos Jantando

- Outra solução → definir tempos aleatórios de espera
- Problema → Não podemos garantir total segurança em termos de funcionamento !
- Vamos analisar uma solução que está correta
- Definir um semáforo binário para proteger o acesso dos filósofos aos garfos

Problemas Clássicos de IPC

Filósofos Jantando

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

typedef int semaphore;
semaphore mutex = 1;

void philosopher(int i) / * i: philosopher number, from 0 to 4 * /
{
      while (TRUE) {
            think(); / * philosopher is thinking * /
            down(mutex);
            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 * /
            up(mutex);
            }
}
```

Problemas Clássicos de IPC
Filósofos Jantando

- Falha na solução apresentada → apenas um filósofo pode estar comendo de cada vez.
- Com cinco garfos disponíveis → pode-se ter dois filósofos comendo ao mesmo tempo
- Vamos analisar um solução correta e também permite o máximo de paralelismo para um número arbitrário de processos.

Problemas Clássicos de IPC
Filósofos Jantando

```
#define N 5 / * number of philosophers * /
#define LEFT (i-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 * /
```

Problemas Clássicos de IPC

Filósofos Jantando

```
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 * /
    }
}
```

Problemas Clássicos de IPC

Filósofos Jantando

```
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 * /
}
```

Problemas Clássicos de IPC

Filósofos Jantando

```
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]);
     }
}
```

Problemas Clássicos de IPC
Leitores e Escritores

- Modela o acesso concorrente a um Banco de Dados (BD)
- Temos processo concorrentes querendo ler e escrever em um BD
- É permitido ter múltiplos processos lendo o BD ao mesmo tempo
- Mas se um processo estiver atualizando (escrevendo) o BD, nenhum outro poderá ter acesso ao BD, nem mesmo os leitores.

Problemas Clássicos de IPC
Leitores e Escritores

- Pergunta: Como programar os leitores e escritores?
- Vamos analisar um solução:

```
typedef int semaphore; / * use your imagination * /

semaphore mutex = 1; / * controls access to 'rc' * /
semaphore db = 1; / * controls access to the data base * /
int rc = 0; / * # of processes reading or wanting to * /
```

Problemas Clássicos de IPC

Leitores e Escritores

```
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 * /
     }
}
```

Problemas Clássicos de IPC
Leitores e Escritores

```
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 * /
      }
}
```

Problemas Clássicos de IPC
Barbeiro Dorminhoco

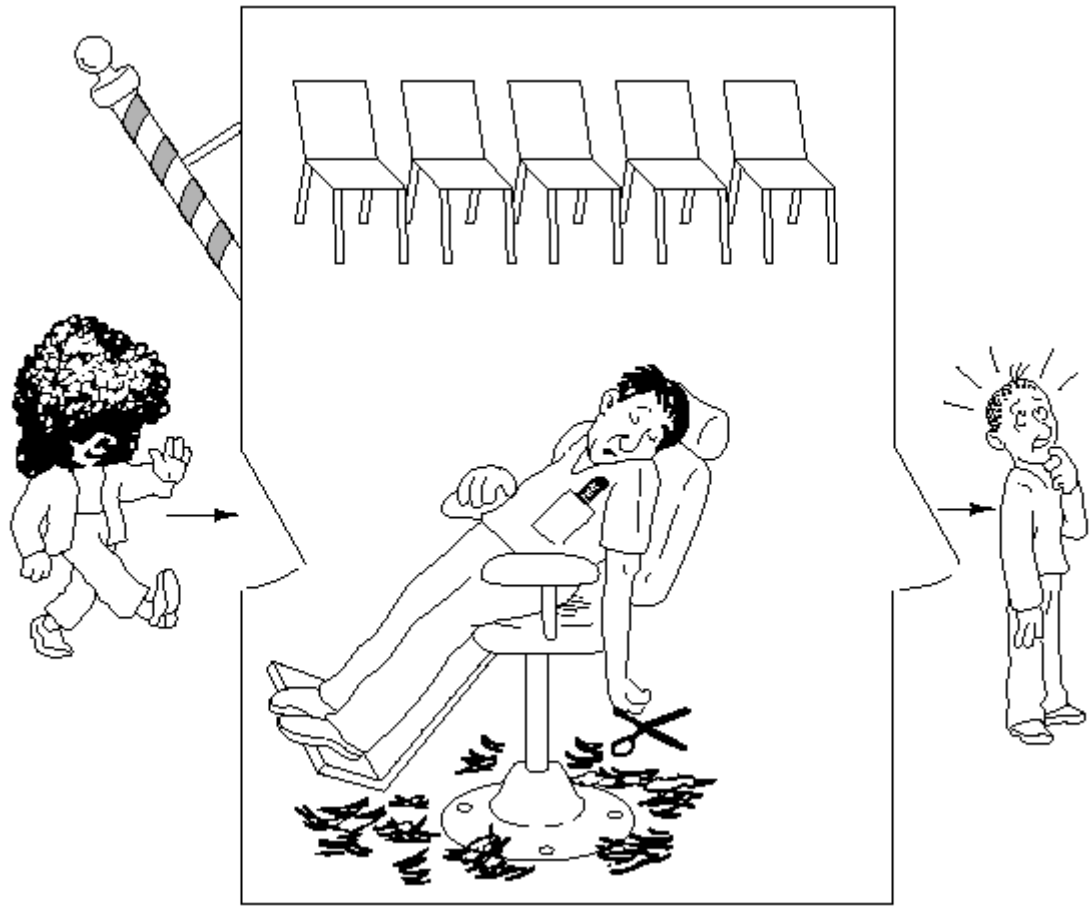
- Outro problema clássico de IPC
- Uma barbearia possui um barbeiro, uma cadeira de cortar cabelo, e n cadeiras para fregueses em espera se sentarem.
- Se não existem fregueses na barbearia, o barbeiro senta-se na cadeira de cortar cabelo e tira um cochilo.

Problemas Clássicos de IPC
Barbeiro Dorminhoco

- Quando um freguês chega na barbearia, ele deve acordar o barbeiro dorminhoco.
- Se mais fregueses chegarem enquanto o barbeiro estiver cortando o cabelo de um freguês, eles devem esperar sentados em uma das cadeiras disponíveis na sala de espera.
- Caso não haja cadeiras disponíveis, o cliente vai embora da barbearia.

Problemas Clássicos de IPC

Barbeiro Dorminhoco



Problemas Clássicos de IPC
Barbeiro Dorminhoco

```
#define CHAIRS 5 / * # chairs for waiting customers * /

typedef int semaphore; / * use your imagination * /

semaphore customers = 0; / * # of customers waiting for service * /
semaphore barbers = 0; / * # of barbers waiting for customers * /
semaphore mutex = 1; / * for mutual exclusion * /
int waiting = 0; / * customers are waiting (not being cut) * /
```

Problemas Clássicos de IPC
Barbeiro Dorminhoco

```
void barber(void)
{
     while (TRUE) {
          down(customers); / * go to sleep if # of customers is 0 * /
          down(mutex); / * acquire access to 'waiting' * /
          waiting = waiting - 1; / * decrement count of waiting customers * /
          up(barbers); / * one barber is now ready to cut hair * /
          up(mutex); / * release 'waiting' * /
          cut_hair(); / * cut hair (outside critical region) * /
     }
}
```

Problemas Clássicos de IPC
Barbeiro Dorminhoco

```
void customer(void)
{
     down(mutex); / * enter critical region * /
     if (waiting < CHAIRS) { / * if there are no free chairs, leave * /
          waiting = waiting + 1; / * increment count of waiting customers *
          up(customers); / * wake up barber if necessary * /
          up(mutex); / * release access to 'waiting' * /
          down(barbers); / * go to sleep if # of free barbers is 0 * /
          get_haircut(); / * be seated and be serviced * /
     } else {
          up(mutex); / * shop is full; do not wait * /
       }
}
```