2020春

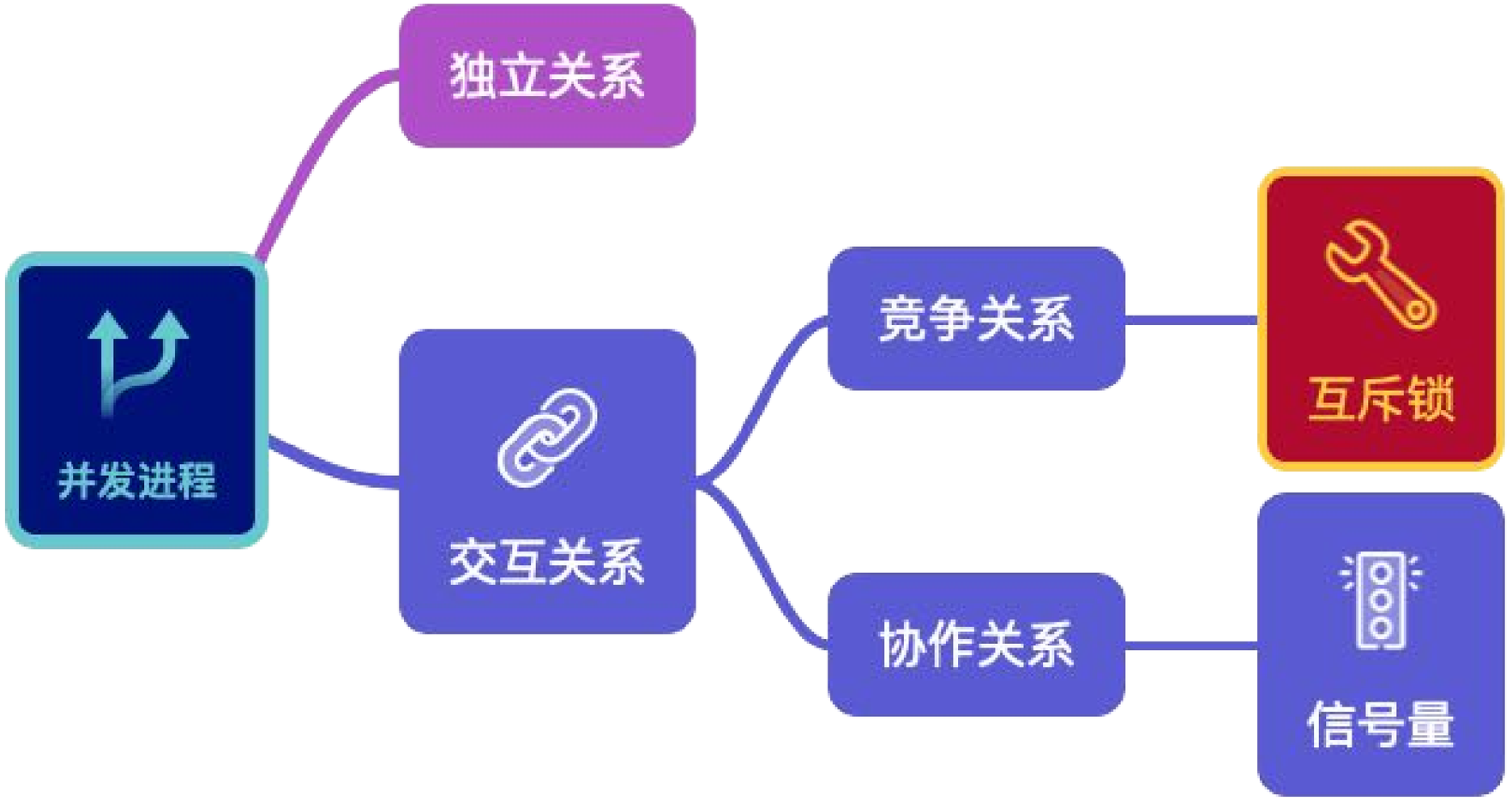
LINUX OPERATING SYSTEM

YANG

LINUX操作系统（双语）

双语课→课件内容中英混排

REVIEW L07



Lecture 8

Mutex Locks

本讲内容

- 临界区问题
- 喂养金鱼
- 互斥锁

临界区问题

CRITICAL-SECTION PROBLEM

- Each concurrent process has a segment of code, called a critical section, in which the process may be changing common variables, updating a table, writing a file, and so on.
- The important feature of the system is that, when one process is executing in its critical section, no other process is allowed to execute in its critical section. That is, NO two processes are executing in their critical sections at the same time.
- The critical-section problem is to design a protocol that the processes can use to cooperate.

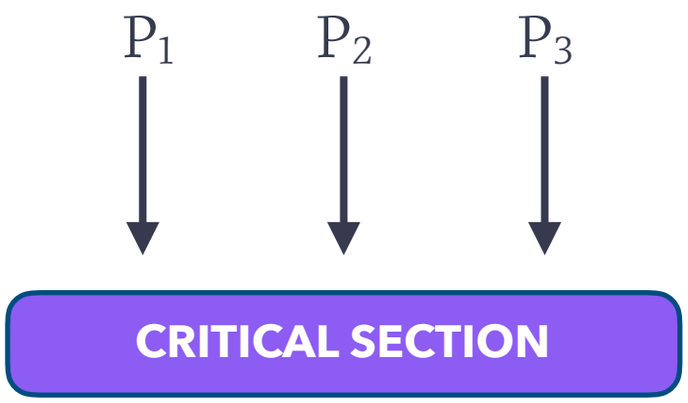
进程进出临界区协议

```
do {
        entry section
                critical section
        exit section
                remainder section
} while (true);
```

- 进入临界区前在entry section要请求许可；
- 离开临界区后在exit section要归还许可。

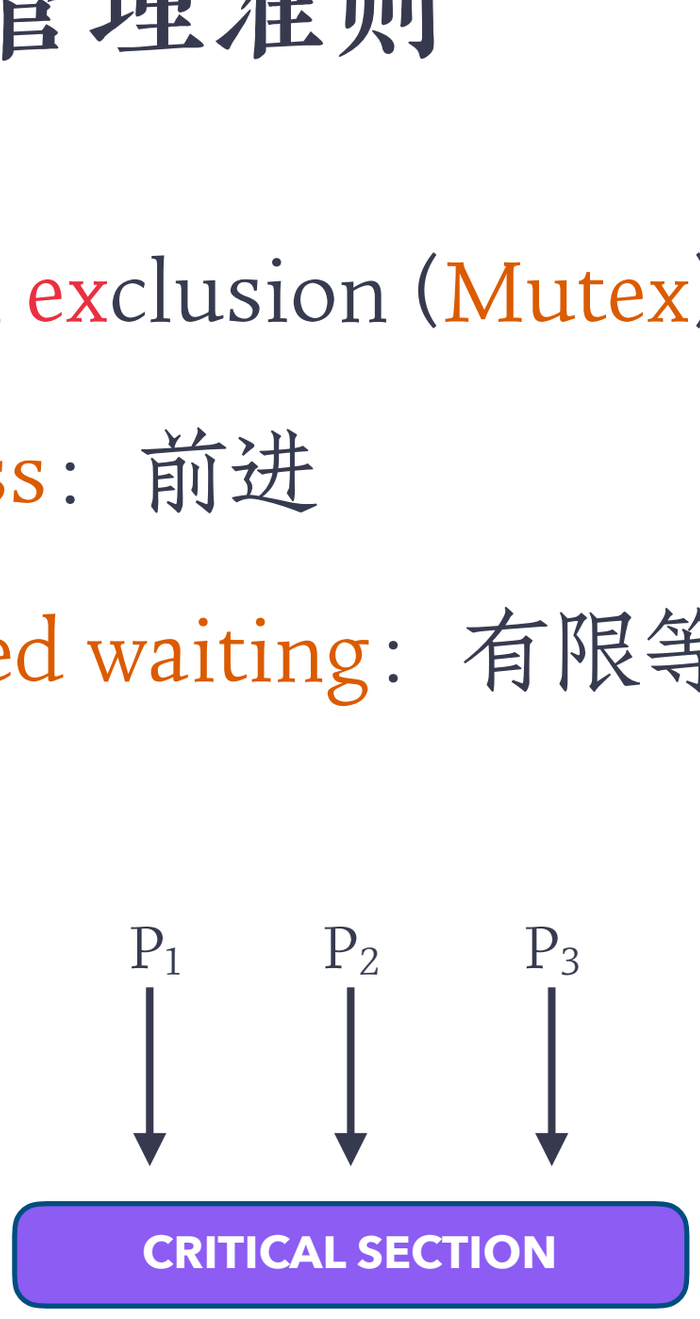
临界区管理准则

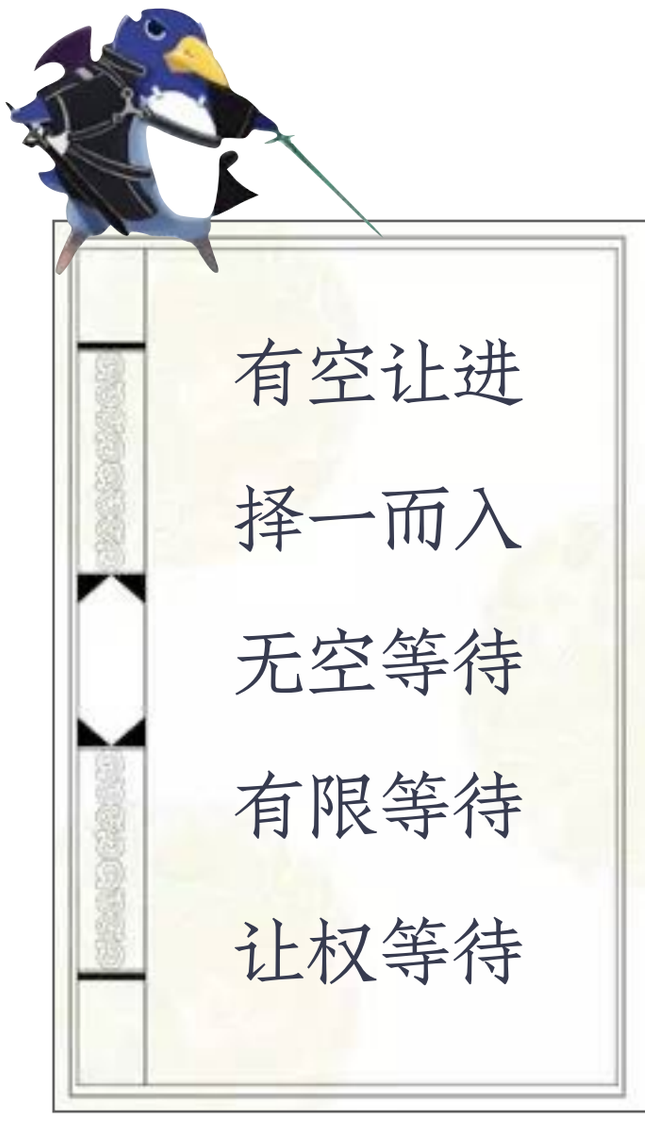
- Mutual exclusion (Mutex)：互斥
- Progress：前进
- Bounded waiting：有限等待



临界区管理准则

- Mutual exclusion (Mutex)：互斥
- Progress：前进
- Bounded waiting：有限等待



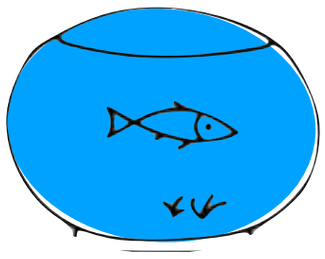


喂养金鱼

金鱼生存法则

- 每天喂一次，且仅一次
- 今天一人喂过，另一人就不能再喂
- 今天一人没有喂过，另一人就必须喂

Alice



Tom

DAY 1

```
Alice

if (no feed){

 feed fish

}
```

```
Tom

if (no feed){

 feed fish

}
```

- [ ] **DEAD**
- [ ] **LIVE**

DAY 2

```
 Alice
if (no note){
 leave a note
 if (no feed){
   feed fish
 }
 remove note
}
```

```
 Tom
if (no note){
 leave a note
 if (no feed){
   feed fish
 }
 remove note
}
```

- [ ] **DEAD**
- [ ] **LIVE**

DAY 3

```
Alice

leave noteAlice

if (no noteTom){

  if (no feed){

     feed fish

  }

  remove noteAlice

}
```

```
Tom

leave noteTom

if (no noteAlice){

  if (no feed){

     feed fish

  }

  remove noteTom

}
```

- [ ] **DEAD**
- [ ] **LIVE**

DAY 4

```
Alice
leave noteAlice
while(noteTom){
   do nothing
}
if (no feed){
   feed fish
}
remove noteAlice
```

```
Tom
leave noteTom


if (no NoteAlice){
   if (no feed){
      feed fish
   }
}
remove noteTom
```

- [ ] **DEAD**
- [ ] **LIVE**

软件解决临界区管理理

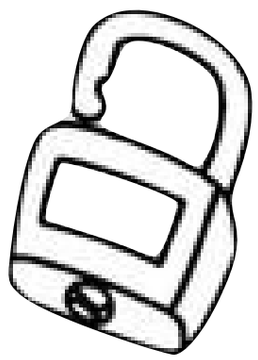
- 实现需要较高的编程技巧
- 两个进程的实现代码是不对称的，当处理超过2个进程的时候，代码的复杂度会变得更大
- 两个著名的软件方案
  - Peterson
  - Dekker

TAKE A BREAK

互斥锁

MUTEX LOCKS

- Operating-systems designers build software tools to solve the critical-section problem. The simplest of these tools is the mutex lock.
  - A process must acquire the lock before entering a critical section;
  - It must release the lock when it exits the critical section.



锁的基本操作

```
Alice                Tom

lock()               lock()

if (no feed){        if (no feed){
feed fish            feed fish
}                    }
unlock()             unlock()
```

- 上锁
  - 等待锁至打开状态
  - 获得锁并锁上
- 解锁
- 原子操作

原子操作

- Atomic operations mean the operation can NOT be interrupted while it's running.
- 原子操作（原语）是操作系统重要的组成部分，下面2条硬件指令都是原子操作，它们可以被用来实现对临界区的管理（也就是“锁”）。
  - test_and_set()
  - compare_and_swap()

锁的实现

```
bool available = true;//unlocked
lock(){
   while(!ts(&available))
        do nothing;
}
```

```
unlock(){
   available = true;
}
```

```
bool ts(bool* target){
    bool result = *target;
    *target = false;
    return result;
}
```

忙式等待（BUSY WAITING）

- 忙式等待是指占用CPU执行空循环实现等待
- 这种类型的互斥锁也被称为“自旋锁”（spin lock）
  - 缺点：浪费CPU周期，可以将进程插入等待队列以让出CPU的使用权；
  - 优点：进程在等待时没有上下文切换，对于使用锁时间不长的进程，自旋锁还是可以接受的；在多处理器系统中，自旋锁的优势更加明显。

下期预告

- 本期视频会分成理论和实践2个部分
- 最近这段时间B站审核时间超过6小时，请大家耐心等待
- 下次直播时间：3月4日 上午9:30
- 课程内容
  - Lecture 9 Semaphores

Lecture 8

The End

实践3 互斥锁

解决订票终端的临界区管理

```c
int ticketAmount = 2; //Global Variable

void* ticketAgent(void* arg){
    int t = ticketAmount;
    if (t > 0)
    {
        printf("One ticket sold!\n");
        t--;
    }else{
        printf("Ticket sold out!!\n");
    }
    ticketAmount = t;
    pthread_exit(0);
}
```

a. pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER; //创建一个锁

b. pthread_mutex_lock(&lock); //上锁

c. pthread_mutex_unlock(&lock); //开锁