西安交通大学
软件学院

# 操作系统原理

## Operating System Principle

田丽华

# 6-2 临界区

# Access to critical areas
# 临界区的访问过程

- 对于临界资源，多个进程必须互斥的对它进行访问。
- **临界区(critical section)：**进程中访问**临界资源**的一段代码。
- **实现进程对临界资源的互斥访问—各进程互斥的进入自己的临界区**
- 假定一个系统有n个进程{P0,P1,……,Pn-1},每个进程有一个代码段称为临界区,在该区中进程可能修改共享变量\更新一个表\写一个文件等.

  **当一个进程在临界区中执行时,其他进程都不能进入临界区**

# Access to critical areas
# 临界区的访问过程

临界区的执行在时间上是互斥的,进程必须请求允许进入临界区

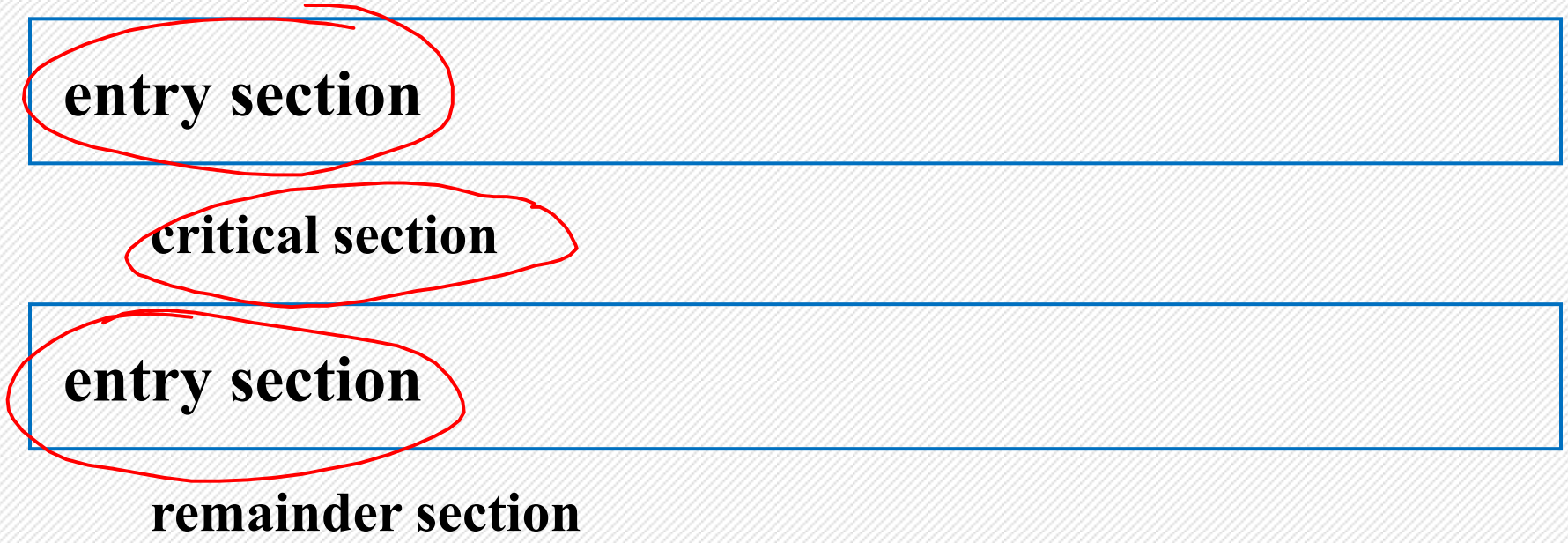
**进入区(entry section)**：在进入临界区之前，检查可否进入临界区的一段代码。如果可以进入临界区，通常设置相应“正在访问临界区”标志。

**退出区(exit section)**：用于将"正在访问临界区"标志清除。

**剩余区(remainder section)**：代码中的其余部分。

# Access to critical areas

# 临界区的访问过程



# Solution to Critical-Section Problem
# 临界区问题的解决方案

## Mutual Exclusion 互斥

If process Pi is executing in its critical section, then no other processes can be executing in their critical sections（假定进程Pi在其临界区内执行，其他任何进程将被排斥在自己的临界区之外）.

# Solution to Critical-Section Problem
# 临界区问题的解决方案

## Progress 有空让进

If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely（临界区虽没有进程执行，但有些进程需要进入临界区，不能无限期地延长下一个要进入临界区进程的等待时间）.

# Solution to Critical-Section Problem
# 临界区问题的解决方案

## Bounded Waiting 有限等待

. A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted（在一个进程提出进入临界区的请求和该请求得到答复的时间内，其他进程进入临界区的次数必须是有限的）.

如何实现进程间的互斥？

- 轮流？
- 申请？

# 两进程互斥的软件方法

## 算法1：

设立一个两进程公用的整型变量 **turn**：描述允许进入临界区的进程标识

有两个进程 **Pi**，**Pj**，如果 **turn==i**，那么进程**Pi**允许在其临界区执行

Pi

```
while (turn != i);
    critical section
turn = j;
    remainder section
```

Pj

```
while (turn != j;)
    critical section
turn = i;
    remainder section
```

- 在进入区循环检查是否允许本进程进入：
- **turn为i时，进程Pi可进入；**
- 在退出区修改允许进入进程标识：
- **进程Pi退出时，改turn为进程Pj的标识j；**

# Algorithm 1

## 缺点：

强制轮流进入临界区，没有考虑进程的实际需要。

容易造成资源利用不充分：在进程1让出临界区之后，进程2使用临界区之前，进程1不可能再次使用临界区；

# 两进程互斥的软件方法

算法1只记住了哪个进程能进入临界区，没有保存进程的状态

设立一个标志数组**flag[]**：描述进程是否准备进入临界区，初值均为**FALSE**,先申请后检查。可防止两个进程同时进入临界区。

```
flag[i] = TRUE;          〈b〉
while (flag[j]);         〈a〉
  critical section
flag[i] = FALSE;
  remainder section
```

```
flag[j]=TRUE;
while(flag[i]);
  critical section
flag[j]=FALSE;
  remainder section
```

# Algorithm 2

## 优点：

不用交替进入，可连续使用；

## 缺点：

两进程可能都进入不了临界区

Pi和Pj在切换自己flag之后和检查对方flag之前有一段时间，如果都切换flag，都检查不通过。

# Algorithm 3

**turn=j**;描述可进入的进程（同时修改标志时）

在进入区先修改后检查，并检查并发修改的先后：

- 检查对方**flag**，如果不在临界区则自己进入 - - 空闲则入
- 否则再检查**turn**：保存的是较晚的一次赋值，则较晚的进程等待，较早的进程进入 - - 先到先入，后到等待

```
flag[i]=TRUE;turn=j
while(flag[j]&&turn==j);
    critical section
flag[i]=FALSE;
    remainder section
```

```
flag[j]=TRUE;turn=i
while(flag[i]&&turn==i);
    critical section
flag[j]=FALSE;
    remainder section
```