② 例2 程序

```
main()
{    int p1,p2,p3,p4,p5,pp1,pp2;
       printf(“程序开始执行”);
       if ((p1=fork( )== 0）
     {    printf（“进程proc1执行”）；
          exit(1);
          } else
          if ((p2=fork() )== 0） {
               printf（“进程proc2执行”）；
               exit(1);
           }
       pp1=wait(&pp1);
               /* 等待，直到子进程终止 */
       pp2=wait(&pp2);
               /* 等待，直到子进程终止 */
     if ((p3=fork())== 0）{
        printf（“进程proc3执行”）；
        } else
        if（(p4=fork（）)== 0） {
     printf（“进程proc4执行”）；
        } else
        if（(p5=fork（）)== 0） {
           printf（“进程proc5执
        行”）；
           exit(1);
        }
     printf(“整个程序终止“);
     exit(0);
}
```

ii 试回答如下问题

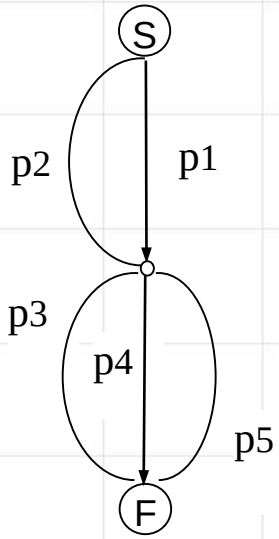
a. 画出描述子进程执行先后次序的进程流图。(各进程分别用其对应的函数名或包含其进程号的符号名标识)。

b. 这个程序执行时最多可能有几个进程同时存在？同时存在的进程数最多时分别是哪几个进程？

c. 程序执行时，“整个程序终止”被输出几次？分别是哪些进程输出的？

iii 问题答案

a.

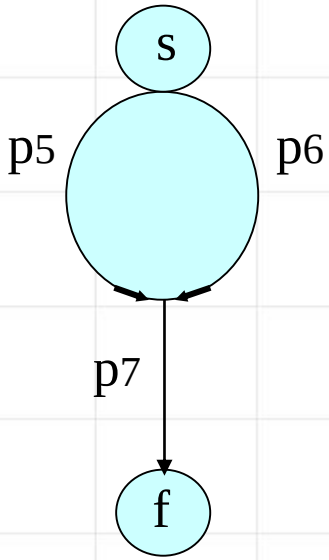


子进程进程执行的流图

b. 最多4个进程同时存在，分别是main、p3、p4、p5 。

c. 3次，main、p3、p4 。

③ 应用实例1



3个合作进程
的进程流图

```
#include  <sys/types.h>
#include  <sys/wait.h>
int  main ()
{
    pid_t  pid;
    int status;
    pid=fork();
    if  (pid==0) {
       p6();  exit();
    } else{
       pid=fork();
       if  (pid==0 )  {
          p5();  exit();
       }
    }
    wait(&status);
    wait(&status);
    p7();
}
```

④ 应用实例2 （自行解决）

```
#include <stdio.h>
#include <pthread.h>
int A;
void subp1()
{
        printf("A      in thread is %d\n",A);
        A = 10;
}
main()
{
        pthread_t  p1;
        int    pid;
            A = 0;
```

```
    pid = fork();
    if (pid==0) {
            printf("A in son process is %d\n",A);
            A=100;
            exit(0);
    }
    wait();
      pthread_create(&p1,NULL,subp1,NULL);
      pthread_join(p1,NULL);
      printf("A     in father process is %d\n",A);
}
```

运行结果？

A in son process is 0
A in thread is 0
A in father process is 10

4. 信号量及其使用方法

- 信号量的本质是一个非负的整数计数器，用来控制对公有变量的访问。
- LINUX信号量函数在通用的信号量数组上操作，而不是在一个单一的二维信号量上操作。

4. 信号量及其使用方法

Linux信号量函数在通用的信号量数组上进行操作，而不是在一个单一的信号量上进行操作。这些系统调用主要包括：semget、semop和semctl。

(1) 信号量的创建

① 功能

创建一个新的信号量或是获得一个已存在的信号量键值。

② 原型

int semget(key_t key, int num_sems, int sem_flags)键值

i 参数key是一个用来允许多个进程访问相同信号量的整数值，它们通过相同的key值来调用semget。

ii 参数num_sems参数是所需要的信号量数目。Semget创建的是一个信号量数组，数组元素的个数即为num_sems。

iii sem_flags参数是一个标记集合，与open函数的标记十分类似。低九位是信号的权限，其作用与文件权限类似。另外，这些标记可以与 IPC_CREAT进行或操作来创建新的信号量。一般用：IPC_CREAT | 0666

(2) 信号量的控制

原型

int semctl(int sem_id, int sem_num, int command, ...)

i 参数sem_id，是由semget所获得的信号量标识符。

ii 参数sem_num参数是信号量数组元素的下标，即指定对第几个信号量进行控制。

iii command参数是要执行的动作，有多个不同的ommand值可以用于semctl。常用的两个command值为:

SETVAL: 用于为信号量赋初值，其值通过第四个参数指定。

IPC_RMID：当信号量不再需要时用于删除一个信号量标识。

iv 如果有第四个参数，则是union semun，该联合定义如下：

```
union semun {
        int val;
        struct semid_ds *buf;
        unsigned short *array;
 }
```

(3) 信号量的操作

① 原型

int semop(int sem_id, struct sembuf *sem_ops, size_t num_sem_ops)

i 参数sem_id，是由semget函数所返回的信号量标识符。

ii 参数sem_ops是一个指向结构数组的指针，该结构定义如下：

iii num_sem_ops 操作次数，一般为1

```
struct sembuf {
        short sem_num;  //数组下标
        short sem_op; //操作，-1或+1
        short sem_flg; //0
}
```

② P操作

```
void P(int semid,int index)
{      struct sembuf sem;
       sem.sem_num = index;
       sem.sem_op = -1;
       sem.sem_flg = 0;//：操作标记：0或
       IPC_NOWAIT等
       semop(semid,&sem,1);  //1:表示执行命令的个数
       return;
}
```

③ V操作

```
void P(int semid,int index)
void V(int semid,int index)
  {    struct sembuf sem;
       sem.sem_num = index;
       sem.sem_op =  1;
       sem.sem_flg = 0;
      semop(semid,&sem,1);
      return;
}
```

5. 共享内存

(1) 功能

共享内存允许两个或更多进程访问同一块内存，就如同 malloc() 函数向不同进程返回了指向同一个物理内存区域的指针。当一个进程改变了这块地址中的内容的时候，其它进程都会察觉到这个更改。

(2) 共享内存创建

int shmget(key_t key,int size,int shmflg)

其中：

① key: 键值，多个需要使用此共享内存的进程用相同的key来创建

② shmflg: IPC_CREAT|0666

(3) 共享内存绑定

int shmget(key_t key,int size,int shmflg)

int shmat (int shmid, char *shmaddr, int shmflg)

其中：

① shmid:共享内存句柄，shmget调用的返回值；

② shmaddr:一般用NUL;

③ shmflg: SHM_R|SHM_W

S = (char *)shmat(shmid1,NULL,SHM_R|SHM_W)

一旦绑定，对共享内存的操作即转化为对局部变量S的操作。

(4) 共享内存的释放

系统调用格式：int shmctl(shmid,cmd,buf);

其中：

shmid:共享内存句柄，shmget调用的返回值；

cmd:操作命令shmctl(shmid,IPC_RMID,0)

进程调度

1. 调度/分派结构

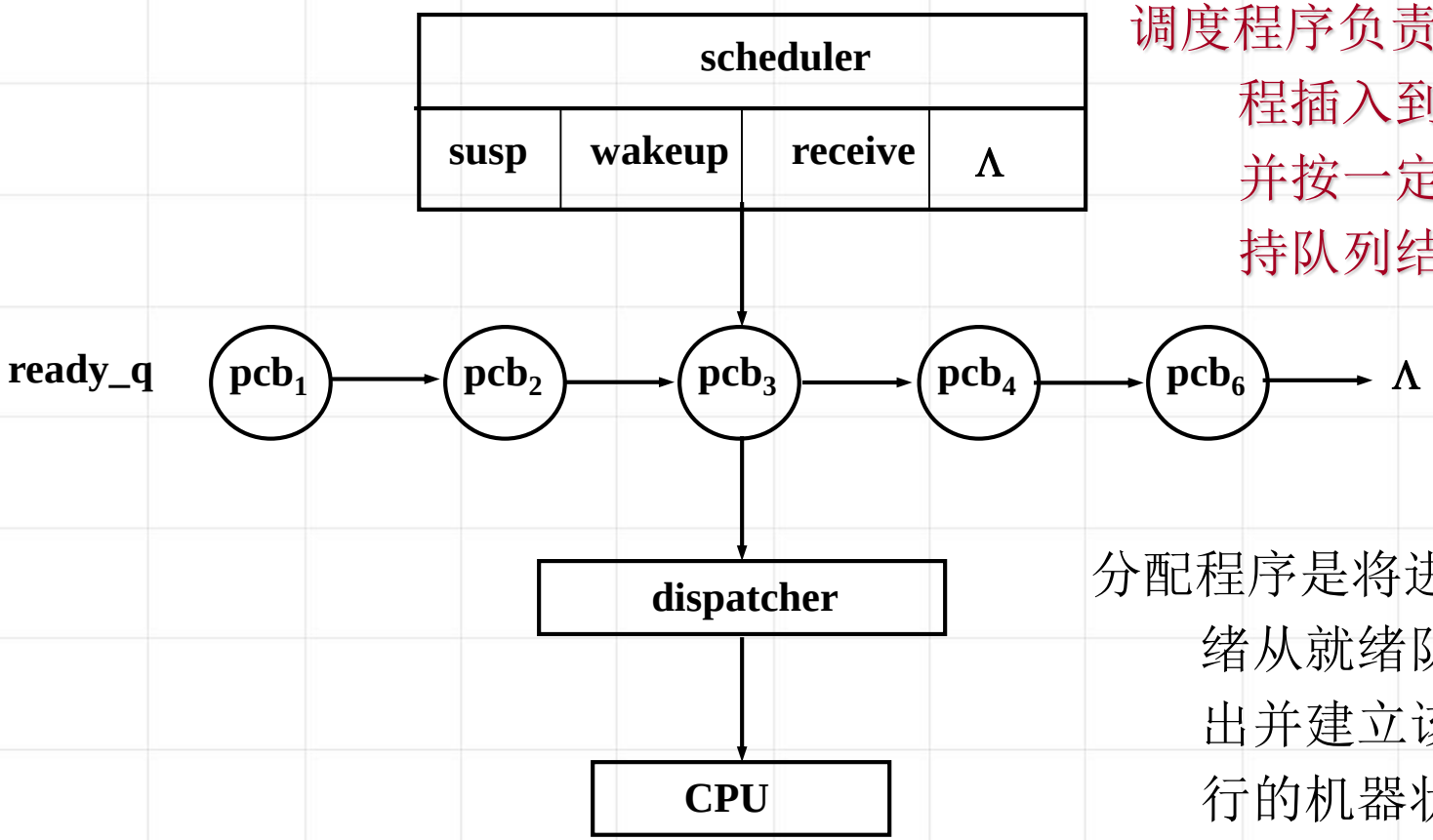
(1) 调度

在众多处于就绪状态的进程中，按一定的原则选择一个进程。

(2) 分派

当处理机空闲时，移出就绪队列中第一个进程，并赋予它使用处理机的权利。

(3) 调度分派结构图



调度/分派结构示意图

调度程序负责将一个进程插入到就绪队列并按一定的原则保持队列结构

分配程序是将进程从就绪从就绪队列中移出并建立该进程执行的机器状态

2. 进程调度的功能

(1) 记录进程的有关情况

(2) 决定调度策略

① 优先调度

就绪队列按进程优先级高低排序

② 先来先服务

就绪队列按进程来到的先后次序排序

(3) 实施处理机的分配和回收

3. 进程调度的方式

(1) 什么是调度方式

当一进程正在处理机上执行时，若有某个更为“重要而紧迫”的进程需要运行，系统如何分配处理机。

(2) 非剥夺方式

当“重要而紧迫”的进程来到时，让正在执行的进程继续执行，直到该进程完成或发生某事件而进入“完成”或“阻塞”状态时，才把处理机分配给“重要而紧迫”的进程。

(3) 剥夺方式

当“重要而紧迫”的进程来到时，便暂停正在执行的进程，立即把处理机分配给优先级更高的进程。

4. 进程调度算法

(1) 进程优先数调度算法

① 什么是进程优先数调度算法

预先确定各进程的优先数，系统把处理机的使用权赋予就绪队列中具备最高优先权 (优先数和一定的优先级相对应) 的就绪进程。

② 优先数的分类及确定

i 静态优先数

在进程被创建时确定，且一经确定后在整个进程运行期间不再改变。

ii 静态优先数的确定

- 优先数根据进程所需使用的资源来计算
- 优先数基于程序运行时间的估计
- 优先数基于进程的类型 ■

iii 动态优先数

进程优先数在进程运行期间可以改变。

iv 动态优先数的确定

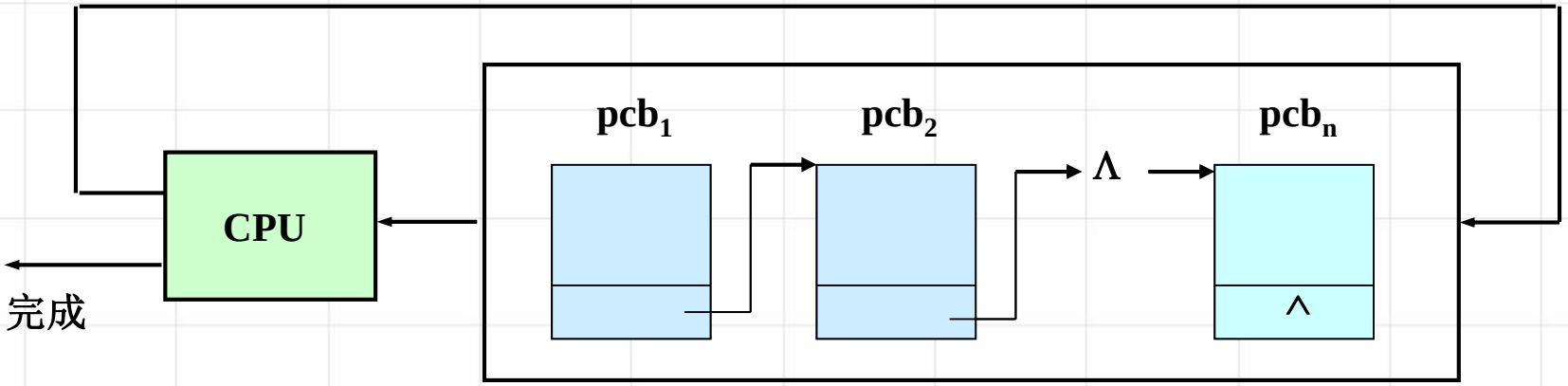
- 进程使用CPU超过一定数值时，降低优先数
- 进程I/O操作后，增加优先数
- 进程等待时间超过一定数值时，提高优先数

(2) 循环轮转调度算法

① 什么是循环轮转调度算法

当CPU空闲时，选取就绪队列首元素，赋予一个时间片，当时间片用完时，该进程转为就绪态并进入就绪队列末端。

该队列排序的原则是什么？



简单循环轮转调度算法示意图

② 简单循环轮转调度算法

就绪队列中的所有进程以等速度向前进展。

$q = t/n$

t 为用户所能接受的响应时间，n为进入系统的进程数目。

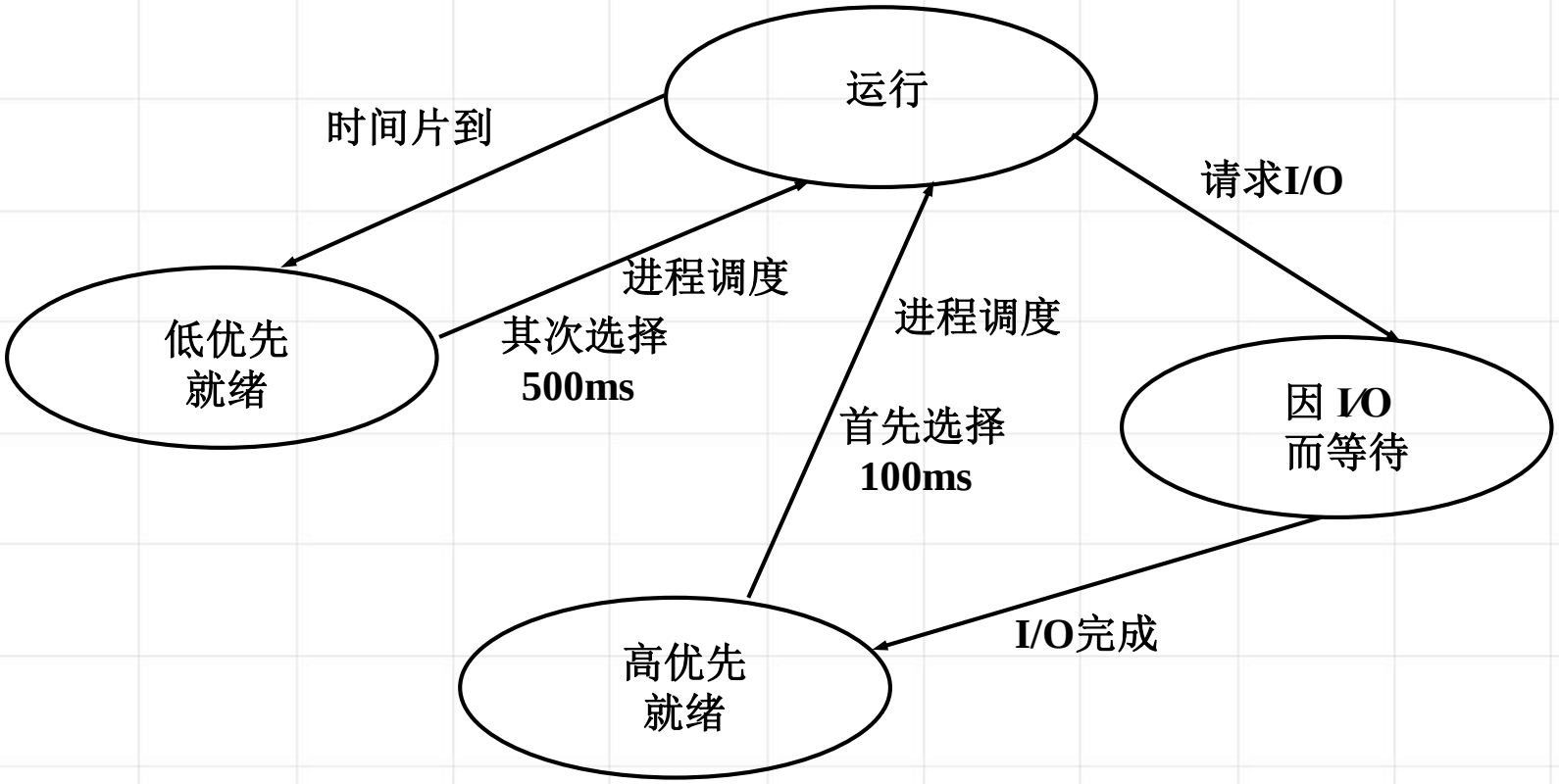
q 值的影响?

③ 循环轮转调度算法的发展

- 可变时间片轮转调度
- 多重时间片循环调度

5. 调度用的进程状态变迁图

(1) 一个调度用的进程状态变迁图的实例



调度用的进程状态变迁图

(2) 调度用进程状态变迁图实例的分析

① 进程状态

- 运行状态
- 低优先就绪状态
- 高优先就绪状态
- 因I/O而等待状态

② 队列结构

- 低优先就绪队列
- 高优先就绪队列
- 因I/O而等待队列

③ 进程调度算法

优先调度与时间片调度相结合的调度算法

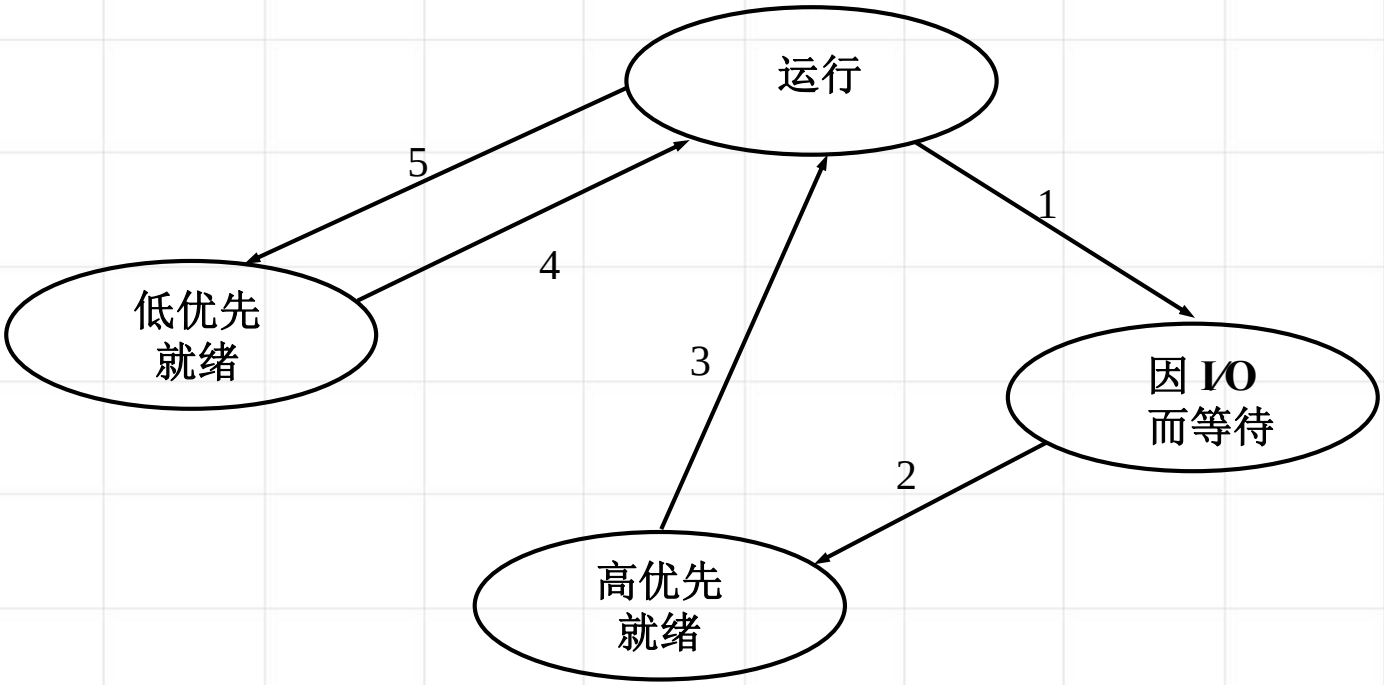
i 当CPU空闲时，若高优先就绪队列非空，则从高优先就绪队列中选择一个进程运行，分配时间片为100ms。

ii 当CPU空闲时，若高优先就绪队列为空，则从低优先就绪队列中选择一个进程运行，分配时间片为500ms。

④ 调度效果

优先照顾I/O量大的进程；适当照顾计算量大的进程。

(3) 较复杂进程状态变迁的讨论



进程状态变迁图

变迁1 → 变迁3

变迁1 → 变迁4

变迁2 → 变迁3

第4章 进程及进程管理
小结

- 进程概念
  - 进程引入
    - 程序的顺序执行　定义　特点
    - 程序的并发执行　定义　特点
  - 进程定义
    - 定义
    - 进程与程序的区别
  - 进程状态
    - 三个基本状态、状态变迁图
    - 不同操作系统类型的进程状态变迁图
  - 进程描述
    - PCB的定义与作用
    - 进程的组成
  - 线程定义

- 进程控制
  - 进程控制原语
    - 基本进程控制原语
    - 进程控制原语的执行与进程状态的变化
  - 进程创建、进程撤销原语的功能
  - 进程等待、进程唤醒原语的功能
- 进程的相互制约关系
  - 进程互斥
    - 临界资源
    - 互斥
    - 临界区

- 进程同步
  - 进程同步的概念
  - 进程同步的例
- 进程同步机构
  - 锁、上锁原语、 开锁原语
  - 信号灯及P、V操作
- 进程同步与互斥的实现
  - 用信号灯的P、V操作实现进程互斥
  - 两类同步问题的解答
    - 合作进程的执行次序
    - 共享缓冲区的合作进程的同步
  - 生产者——消费者问题及解答

- 操作系统的并发控制机制
  - 创建进程、创建线程及其使用
  - 等待进程、线程的终止及其使用
  - 信号量与使用方法
  - 共享内存与使用方法
- 进程调度
  - 进程调度的功能
  - 调度方式　非剥夺方式　剥夺方式
  - 常用的进程调度算法
  - 调度用的进程状态变迁图的分析