操作系统

第一章 计算机系统概述

操作系统的概念，功能和目标

作为用户和计算机硬件之间的接口

提供的功能：
1.处理机管理
2.存储器管理
3.设备管理
4.文件管理

提供的接口：
1.命令接口（联机命令接口、脱机命令接口/批处理命令接口）
2.程序接口/系统调用
3.GUI（图形用户界面win/ios/andrio）

目标：方便用户使用

操作系统的特征

1.并发、并行

并发：多个事件交替发生（宏观同时发生，违规交替进行）
并行：多个事件同时发生

2.共享

两种资源共享方式：
1.互斥共享方式：一个时间段内只允许一个进程访问该资源
2.同时共享方式：允许一个时间段内由多个进程“同时”对他们进行访问

3.虚拟：

概念：把一个物理上的实体变为多个逻辑上的对应物

空分复用技术
时分复用技术

4.异步：

概念：在多道程序环境下，允许多个程序并发执行，但由于资源有限，进程的执行不是一贯到底的，而是走走停停的，以不可预知的速度向前推进。只有系统拥有并发性，才有可能导致异步性。

操作系统的发展与分类

1.手工操作阶段

纸带机（用户独占全机、人机速度矛盾）

2.批处理阶段

单道批处理系统（外围机——磁带）
多道批处理系统（操作系统开始出现）

3.分时操作系统

轮流处理作业
不能处理紧急任务

4.实时操作系统

优先处理紧急任务
硬实时系统：必须在严格的时间内完成处理
软实时系统：可以偶尔犯错

5.网络操作系统

6.分布式操作系统

7.个人计算机操作系统

操作系统的运行机制与体系结构

1.运行机制

两种指令：特权指令，非特权指令

两种处理机状态：核心态（root），用户态

两种程序：内核程序（运行在内核态），应用程序

2.操作系统内核

时钟管理（实现计时功能）

中断处理

原语（程序运行具有原子性，不可中断）

系统控制的数据结构及处理

3.对系统资源进行管理的功能

进程管理，存储器管理，设备管理

4.操作系统的体系结构

大内核（将操作系统的主要功能模块都作为系统内核，运行在核心态）
微内核（只把最基本的功能保留在内核）

中断和异常

1.中断机制的产生

操作系统介入，开展管理工作

“用户态->核心态”是通过中断实现的。并且中断是唯一途径

2.中断的概念和作用

中断的分类：
内中断（异常）
    陷阱/自陷（trap）
    故障（fault）
    终止（abort）
外中断（CPU处理）

可屏蔽中断
不可屏蔽中断

系统调用

概念：应用程序通过系统调用请求操作系统的服务。保证系统的稳定性和安全性。

按功能进行分类：
(1)设备管理
(2)文件管理
(3)进程控制
(4)进程通信
(5)内存管理

系统调用和库函数的区别：
1.系统调用是操作系统向上层提供的接口
2.有的库函数是对系统调用的进一步封装
3.当今编写的应用程序大多是通过高级语言提供的库函数间接地进行系统调用

用户态转向核心态的例子：
(1)系统调用
(2)发生一次中断
(3)用户程序产生了一个错误状态
(4)用户程序企图执行特权指令

第二章 进程与线程

进程的状态与切换

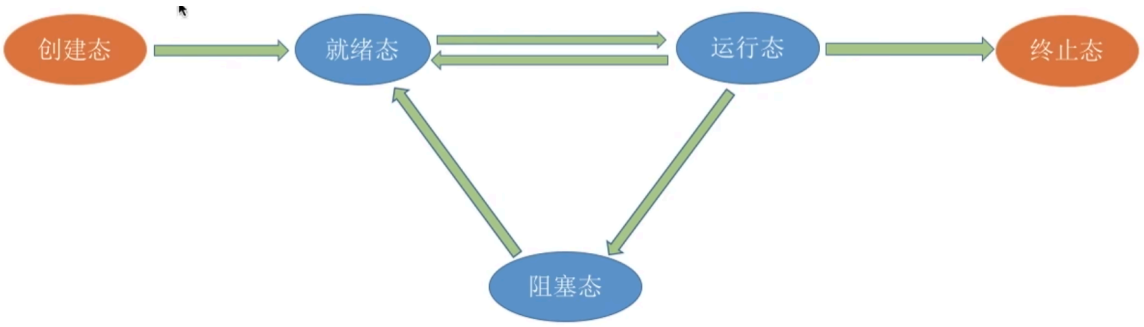
1.进程的特征：

(1)动态性
(2)并发性
(3)独立性
(4)异步性

2.状态：

(1)运行态：占有CPU，并在CPU上运行，单核只能一个进程（双核两个）（CPU√，其它资源√）
(2)就绪态：已经具备运行条件，但是没有空闲的CPU，暂时不能运行（CPUX，其它资源√）
(3)阻塞态：等在某个事件的发生，暂时不能运行（CPUX，其它资源X）
(4)创建态：创建PCB，程序段，数据段
(5)终止态：回收内存，程序段，数据段，撤销PCB

3.进程状态间的转换



(1)创建态->就绪态
(2)就绪态->运行态
(3)运行态->就绪态
(4)运行态->中止态（比如数组越界）
(5)运行态->阻塞态（主动）（等待I/O）
(6)阻塞态->就绪态（被动）

进程控制

1.什么是进程控制?
答：实现各种进程状态转换。

2.如何实现进程控制?
答：用“原语”实现。

3.原语做的事情：
(1)更新PCD中的信息
(2)将PCD插入合适的队列
(3)分配/回收资源

4.进程控制相关的原语：

(1)进程的创建：

创建原语：申请空白PCB、为新进程分配所需资源、初始化PCB、将PCB插入就绪队列

引起进程创建的事件：用户登录、作业调度、提供服务、应用请求

(2)进程的终止：

撤销原语

引起进程中止的事件：正常结束、异常结束、外界干预

(3)进程的阻塞：

阻塞原语：运行态->阻塞态

引起进程阻塞的事件：需要等待系统分配某种资源、需要等待相互合作的其他进程完成工作

(4)进程的唤醒：

唤醒原语：阻塞态->就绪态

引起进程唤醒的事件：等待的事件发生

(5)进程的切换

切换原语

引起进程切换的事件：当前进程事件片到、有更高优先级的进程到达、当前进程主动阻塞、当前进程终止

进程通信

1.共享存储

（分配共享空间，且需要同步互斥工具（P、V操作））

基于数据结构的共享：固定分配（低级）

基于存储区的共享：划分存储区（高级）

2.消息传递

消息：消息头，消息体

直接通信方式（直接挂载消息）

间接通信方式（间接利用信箱发送消息）

3.管道通信pipe

只能半双工通信

互斥（没写满，不能读，反之同理）

线程的概念和多线程模型

1.什么是线程，为什么要引入线程?
答：线程是一个基本的CPU执行单元，也是程序执行流的最小单位，进一步提高了系统的并发度

2.引入线程机制后，有什么变化?

(1)资源分配、调度：进程是资源分配的基本单位，线程是调度的基本单位

(2)并发性：各线程间也能并发，提升了并发度

(3)系统开销：可以只在进程中切换，减小了CPU切换环境的系统开销

3.线程有哪些重要的属性

线程是处理机调度的基本单位
多CPU计算机中，各个线程可占用不同的CPU
每个线程都有一个线程ID、线程控制块（TCB）
线程也有就绪、阻塞、运行三种基本状态
线程几乎不拥有系统资源
同一进程的不同线程间共享进程的资源
由于共享内存地址空间，统一进程中的线程间通信甚至无需系统干预
同一进程中的线程切换，不会引起进程切换
不同进程中的线程切换，会引起进程切换
切换同进程内的线程，系统开销很小
切换进程，系统开销较大

4.线程的实现方式

用户级线程（ULT）：
由应用管理，从用户的视角看能看到的线程

内核级线程（KLT）：
由操作系统管理，从操作系统内核视角看能看到的线程

n个ULT可以映射到m个KLT上（n>=m）

内核级线程才是处理机分配的单位

5.多线程模型

多对一模型

n个ULT映射到1个KLT
优点：开销小，效率高
缺点：容易阻塞，并发度不高

一对一模型

n个ULT映射到n个KLT
优点：并发能力很强
缺点：占用成本高，开销大

多对多模型

n个ULT映射到m个KLT上（n>=m）
中和以上两种优缺点

处理机调度的概念、层次、基本概念

1.通常进程数量大于处理机数量，所以要按照一定的算法选择一个进程，并将处理机分配给它运行，以实现进程的并发执行

2.三个层次

(1)高级调度（作业调度）

辅助外存与内存之间的调度，作业调入时会建立相应的PCB，作业调出时才撤销PCB，调入可由操作系统决定，调出由作业运行结束才调出

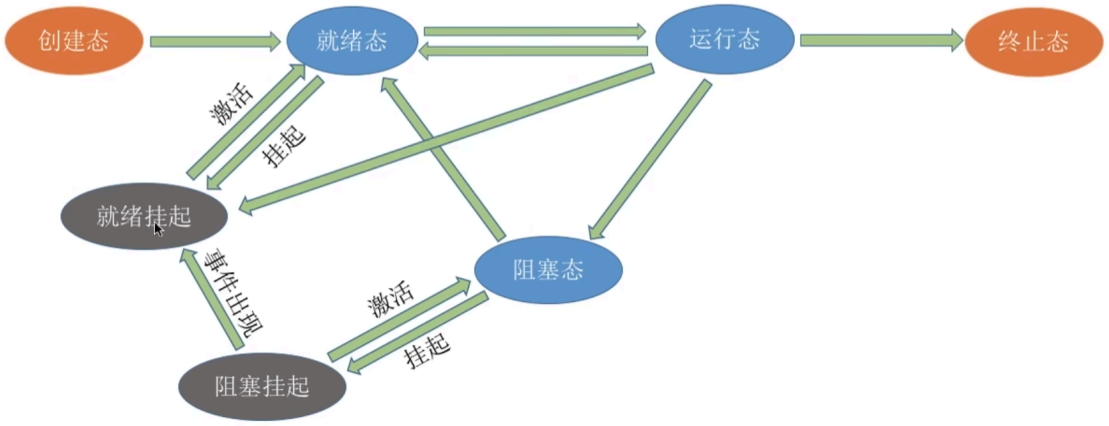
(2)中级调度（内存调度）

将暂时不用的进程放到外存（PCB不外放），提高内存利用率和系统吞吐量，进程状态为挂起状态，形成挂起队列

(3)低级调度（进程调度）

最基本，用算法为进程分配处理机资源，几十ms一次

3.七状态模型



进程调度的时机、切换与过程调度方式

1.时机

什么时候需要进程调度?
(1)主动放弃（进程正常终止、运行过程中发生异常而终止、进程主动请求阻塞）
(2)被动放弃（分给进程的时间片用完、有更紧急的事需要处理、有更高优先级的进程进入就绪队列）

2.什么时候不能进行进程调度?
(1)在处理中断的过程中
(2)在操作系统内核程序临界区中
临界资源：一个时段段内各进程互斥地访问临界资源
临界区：访问临界资源的那段代码a
(3)内核程序临界区会访问就绪队列，导致其上锁
(4)在原子操作过程中（原语）

3.什么事件可能会触发调度程序?
(1)创建新进程
(2)进程退出
(3)运行进程阻塞
(4)I/O中断发送（可能唤醒某些阻塞进程）

4.切换与过程

“狭义的调度”与“进程切换”的区别?
(1)狭义：选择一个进程
(2)广义：狭义+进程切换

进程切换的过程需要做什么?
(1)对原来运行进程各种数据的保存
(2)对新的进程各种数据的恢复

5.方式
非剥夺调度方式（非抢占式）：只允许进程主动放弃处理机
剥夺调度方式（抢占式）：进程被动放弃，可以优先处理紧急任务，适合分时操作系统、实时操作系统

调度器和闲逛进程

1.调度器

就绪进程->排队器->就绪队列->分派器->上下文切换器->CPU

(1)排队器
将所有就绪进程排成一个或多个队列，每有一个进程编程就绪态就插入就绪队列

(2)分派器
根据调度器获得的就绪队列，将CPU分派给新进程

(3)上下文切换器
将当前进程的上下文保存到PCB，装入分派程序的上下文
移出分派程序的上下文，将新选进程的CPU信息装入CPU

2.调度的处理对象
不支持内核级线程的操作系统，调度程序的处理对象是进程
支持内核级线程的操作系统，调度程序的处理对象是内核线程。

3.闲逛进程idle

没有其他就绪进程时，运行闲逛进程

优先级最低

不需要CPU以外的资源，不会被阻塞

可以是0地址指令，占一个完整的指令周期（指令周期末尾例行检查中断）

能耗低

调度算法的评价指标

1、CPU利用率

CPU利用率=CPU忙碌的时间/总时间

2、系统吞吐量

=总共完成了多少道作业/总共画了多少时间

3、周转时间

周转时间（提交作业到完成作业花费的时间）
包括：作业在外存后备队列上等待作业调度（高级调度）的时间、进程在就绪队列上等待进程调度的时间（低级调度）、进程在CPU上执行的时间、进程等待I/O操作完成的时间

平均周转时间（各作业周转时间之和/作业数）

带权周转时间（作业周转时间/作业实际运行的时间）

平均带权周转时间（各作业带权周转时间/作业数）

4、等待时间

进程或作业等待处理机状态时间的和

进程：等待被服务的时间之和

作业：建立后的等待时间+作业在外存后备队列中等待的时间

5、响应时间

从用户提交请求到首次产生响应所用的时间

FCFS、SJF、HRRN调度算法

1、先来先服务（FCFS）

先到达先进行服务

作业-后备队列；进程-就绪队列

非抢占式

公平、算法简单

对长作业有利、对短作业不利、不会饥饿

2、短作业优先（SJF，shortest job first）

SPF，shortest process first短进程优先

SRTN，shortest remaining time next最短剩余时间优先

最短（服务时间最短）的作业优先得到服务，时间相同，先到达的先被服务

非抢占式（SJF）：选最短需要时间的作业先进入运行态

抢占式（SRTN）：有新作业进入就绪队列或有作业完成了，考察队列中的最小需要时间的作业

在所有进程都几乎同时到达时，采用SJP调度算法的平均等待时间、平均周转时间最少

若无红色前提，抢占式的短作业/进程的平均时间最少

优点：“最短的”平均等待时间，平均周转时间

缺点：对短作业有利，对长作业不利，可能产生饥饿现象

3、高响应比优先（HRRN）

要综合考虑作业/进程的等待时间和要求服务的时间

在每次调度时先计算各个作业/进程的响应比，选择响应比最高的作业/进程为其服务

响应比=（等待时间+要求服务时间）/要求服务时间

非抢占式

进程主动放弃CPU时，需要该算法选取就绪队列的作业

不会饥饿

时间片轮转、优先级调度、多级反馈队列（适合交互式系统）

1、时间片轮转算法（RR）

算法思想：公平轮流地位各个进程服务，让每个进程在一定时间间隔内都可以得到响应

算法规则：按照各进程到达就绪队列的顺序，轮流让各个进程执行一个时间片（如100ms）。若进程未在一个时间片内执行完，则剥夺处理机，将进程重新放到就绪队列对位重新排队。

只能用于进程调度

抢占式

优点：响应块，适用于分时操作系统

缺点：由于高频率的进程切换，因此有一定的开销；不区分任务的紧急程度

不会饥饿

2、优先级调度算法

算法思想：根据任务的紧急程度来决定处理顺序

算法规则：每个进程/作业有各自的优先级，调度时选择优先级最高的作业/进程

适用：作业/进程/IO

抢占式/不可抢占均有

静态优先级：不变

动态优先级：可以变

通常：系统进程优先级高于用户进程，前台进程优先级高于后台进程，操作系统更偏好I/O进程

可以从追求公平、提升资源利用率等角度考虑改变优先级

可能会饥饿

3、多级反馈队列调度算法

算法思想：对其它算法调度的这种权衡

算法实现：设置多级就绪队列，各级队列优先级从高到低，时间片从小到大。新进程到达时先进入第一级队列，按照FCFS原则排队等待被分配时间片。若用完时间片进程还未结束，则进程进入下一级队列对位。如果此时已经在最下级的队列，则重新放回最下级队列末尾。啊只有第K级队头的进程为空时，才会为K+1级对头的进程分配时间片，被抢占处理机的进程重新放回原队列队尾。

优点：对各个进程相对公平（FCFS的优点），每个新到达的进程都可以很快就得到响应（RR的优点）；短进程只用较少的时间就可以完成（SPF的优点）；不必实现估计进程的运行时间（避免用户作假）；可灵活地调整对各类进程的偏好程度，比如CPU密集型进程、IO密集型进程

默认抢占式

会饥饿

进程的同步和互斥

1、进程同步

指为了完成某种任务而建立的两个或多个进程，这些进程因为需要在某些位置上协调他们的工作次序而产生的制约关系。进程间的直接制约关系就是源于它们之间的相互合作。

2、进程互斥

把一个时间段内只允许一个进程使用的资源称为临界资源。

对临界资源的互斥访问，可以在逻辑上分为四个部分：

```
do{
  entry section;  //进入区  对访问的资源检查或进行上锁
  critical section; //临界区(段) 访问临界资源的那部分代码
  exit section;   //退出区  负责解锁
  remainder section; //剩余区  其它处理
} while(true)
```

(1)空闲让进。 空的可以直接进去

(2)忙则等待。 繁忙不能进去

(3)有限等待。 不能让进程等待无限长时间

(4)让权等待。 不能进去，不要堵着

进程互斥的软件实现方法

1、单标志法

两个进程在访问完临界区后会把使用临界区的权限教给另一个进程。也就是说每个进程进入临界区的权限只能被另一个进程赋予。

```
int turn =0;
//p0进程
while(turn!=0);
critical section;
turn = 1;
remainder section;
//p1进程
while(turn!=1);
critical section;
turn = 0;
remainder section;
可以实现互斥
```

存在的问题：p1要访问的话，必须p0先访问，违背：空闲让进原则

2、双标志先检查

算法思想:设置一个bool数组flag[]来标记自己是否想要进入临界区的意愿。

```
bool flag[2]={false,false};
//p1进程
while(flag[1]);
flag[0]=true;
critical section;
flag[0]=false;
remainder section;
//p2进程
while(flag[0]);
flag[0]=true;
critical section;
flag[1]=false;
remainder section;
```

主要问题：由于进程是并发进行的，可能会违背忙则等待的原则

3、双标志后检查

算法思想:设置一个bool数组flag[]来标记自己是否想要进入临界区的意愿,不过是先上锁后检查

```
bool flag[2]={false,false};
//p1进程
flag[0]=true;
while(flag[1]);
critical section;
flag[0]=false;
remainder section;
//p2进程
flag[0]=true;
while(flag[0]);
critical section;
flag[1]=false;
remainder section;
```

主要问题：由于进程是并发进行的，可能会两个同时上锁，都进不去，违反空闲让进和有限等待原则

会饥饿

4、Peterson 算法

主动让对方先使用处理器

```
bool flag[2]={false,false};
int turn=0;
//p1进程
flag[0]=true;
turn=1;
while(flag[1]&&turn==1);
critical section;
flag[0]=false;
remainder section;
//p2进程
flag[1]=true;
turn=0;
while(flag[0]&&turn==0);
critical section;
flag[1]=false;
remainder section;
```

遵循空闲让进、忙则等待、有限等待三个原则

进程互斥的硬件实现方法

硬件实现方法也称**元方法**

1、中断屏蔽方法

关中断（不允许进程中断）
临界区
开中断

简单、高效（算法）

多处理机，可能会同时访问临界资源

使用OS内核进程

限制了交替执行程序能力，处理机效率降低

2、TestAndSet（TSL指令）

TSL是用硬件实现的，上锁、检查一气呵成

不满足让权等待，会盲等

C语言描述逻辑：

```
//true表示已经上锁
bool TestAndSet(bool *lock){
  bool old;
  old=*lock;
  *lock=true;
  return old;
}
//以下是使用TSL指令实现互斥的算法逻辑
while(TestAndSet (&lock));//上锁并检查临界区代码段
lock=false; //解锁
```

3、Swap指令

别称：Exchange指令、XCHG指令

Swap指令是用硬件实现的

```
//true表示已经上锁
void Swap(bool *a,bool *b){
  bool temp;
  temp=*a;
  *a=*b;
  *b=temp;
}

//以下是使用Swap指令实现互斥的算法逻辑
bool old=true;
while(old=true)
  Swap(&lock,&old);
临界区代码段
lock=false; //解锁
//剩余代码段
```

简单

适用多处理机

不能让权等待

信号量机制

信号量：

信号量是一种变量，表示系统中某种资源的数量

一对原语：

wait(S)原语和signal(S)原语，分别简称P(S)、V(S)

前者是请求资源（加锁），后者释放资源（解锁）

1、整形信号量

用一个整数表示系统资源的变量，用来表示系统中某种资源的数量

```
int S=1;
void wait(int S){ //wait原语，相当于：进入区
  while(S<=0); //如果资源数不够，就意志循环等待
  S=S-1;    //如果资源数够，则占用一个资源
}

void signal(int S){//signal原语，相当于“退出区”
  S=S+1;    //使用完资源后，在退出区释放资源
}
```

可能会出现盲等

2、记录型信号量

记录型数据结构表示的信号量

```
//记录型信号量的定义
typedef struct{
  int value;
  struct process *L;
} semaphore;
//某进程需要使用资源时，通过wait原语申请
void wait (semaphore S){
  S.value--;
  if(S.value<0){
    block (S.L);//将该进程加入到消息队列中
 }
}
//进程使用完资源后，通过signal原语释放
void signal (semaphore S){
  S.value++;
  if(S.valie<=0){
    wakeup(S.L);
 }
}
```

除非特别说明，否则默认S为记录型信号量

用信号量机制实现进程互斥、同步、前驱关系

1、实现进程互斥

设置互斥信号量mutex，初值为1

对不同的临界资源需要设置不同的互斥信号量

PV必须成对出现

2、实现进程同步

保证一前一后的操作顺序

设置同步信号量S，初始为0

在“前操作”之后执行V（S）

在“后操作”之后执行（V）

3、实现进程的前驱关系

(1)要为每一对前驱关系各设置一个同步变量（一个箭头一个变量）

(2)在“前操作”之后对相应的同步变量执行V操作（前驱完成后释放资源）

(3)在“后操作”之前对相应的同步变量执行P操作（后继开始前获得资源）

经典同步互斥问题

生产者-消费者问题

只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待

只有缓冲区不空时，消费者才能从中取出产品，否则必须等待

缓冲区是临界资源，各个进程互斥访问

实现互斥的P操作要放在实现同步的P操作之后，不然会发生死锁

V操作不会导致进程发生阻塞的状态，所以可以交换

使用操作不要放在临界区，不然并发度会降低

多生产者-多消费者模型

在生产-消费者问题中，如果缓冲区大小为1，那么有可能不需要设置互斥信号量就可以实现互斥访问缓冲区

分析同步问题是，应该从“事件”的角度来考虑

吸烟者问题

解决“可以让生产多个产品的单生产者”问题提供一个思路；

若一个生产者要生产多种产品（或者说会引发多种前驱事件），那么各个V操作应该放在各自对应的“事件”发生之后的位置

读者-写者问题

1、允许多个读者同时对文件执行读操作

2、只允许一个写者往文件中写信息

3、任一写者在完成写操作之前不允许其他读者或写者工作

4、写者执行写操作前，应让已有的读者和写者全部退出

```
semaphore rw=1;//用于实现对文件的互斥访问。表示当前是否有进程在访问共享文件
int count=0;//记录当前有几个读进程在访问文件
semaphore mutex=1;//用于保证对count变量的互斥访问
semaphore w=1; //用于实现“写优先”

writer(){
  while(1){
    P（w）;
    P(rw); //写之前“加锁”
    写文件。。。
    V（rw);//写之后“解锁”
  V(w);
```

```
 }
}

reader(){
  while(1){
    P(w);
   P(mutex); //各读进程互斥访问count
    if(count==0)
      P(rw); //第一个读进程的读进程数+1
    count++; //访问文件的读进程数+1
    V(mutex);
    V(w);
    读文件...
    P(mutex); //各读进程互斥访问count
    count--; //访问文件的读进程数-1
    if(count==0)
      V(rw); //最后一个读进程负责“解锁”
    V(mutex);
 }
}
```

哲学家进餐问题

五个人，必须拿左右的筷子才能吃饭

避免死锁发生

解决方案：

1、可以对哲学家进程施加一些限制条件，比如最多允许四个哲学家同时进餐，这样可以保证至少有一个哲学家是可以拿到左右两只筷子的。

2、要求奇数号哲学家先拿左边的筷子，然后再拿右边的筷子，而偶数号哲学家刚好相反。用这种方法可以保证如果相邻的两个奇偶号哲学家都想吃饭，那么只会有其中一个可以拿起第一只筷子，另一个会直接阻塞。这就避免了占有一只后再等待另一只的情况。

3、仅当一个哲学家左右两只筷子都可用时才允许他抓起筷子。

```
semaphore chopstick[5]={1,1,1,1,1};
semaphore mutex = 1; //互斥地取筷子
Pi(){     //i号哲学家的进程
  while(1){
    P(mutex);
    p(chopstick[i]);   //拿右
    p(chopstick[(i+1)%5]);//拿左
    V(mutex);
    吃饭...
    V(chopstick[i]);
    V(chopstick[(i+1)%5]);
    思考...
 }
}
```

管程

1、为什么要引入管程

PV操作容易出错、困难

2、管程的定义和基本特征

定义：

管程的名称
局部于管程的共享数据结构说明
对该数据结构进程操作的一组过程
对局部于管程的共享数据设置初始值的语句

基本特征：

局部于管程数据结构只能被局部于管程的过程所访问
一个进程只有通过调用管程内的过程才能进入管程访问共享数据
每次仅允许一个进程在管程内执行某个内部过程
心得：相当于C++的类，管程是数据放在private中，函数放在public中

拓展1：用管程解决生产者消费者问题

```
monitor producerconsumer
  condition full,empty;
  int count = 0;
  void insert(Item item){
    if(count == N)
      wait(full);
    count++;
    insert_item (item);
    if(count == 1)
      signal(empty);
 }
  Item remove(){
    if(count == 0)
      wait(empty);
    count--;
    if(count == N-1)
      signal(full);
    return remove_item();
 }
  end monitor;

//使用
producer(){
  while(1){
    item = 生产一个产品;
    producerconsumer.insert(item);
 }
}

consumer(){
  while(1){
    item = producerconsumer.remove();
    消费产品 item;
```

```
  }
}
```

拓展2：Java中类似于管程的机制

java中用synchronized来描述一个函数,这个函数同一时间只能被一个线程调用

死锁

死锁的概念

1、什么是死锁

各进程互相等待对方手里的资源，导致各进程都阻塞，无法向前推进的现象。

2、进程死锁、饥饿、死循环的区别

死锁：

定义：各进程互相等待对方手里的资源，导致各进程都阻塞，无法向前推进的现象。

区别：至少两个或两个的进程同时发生死锁

饥饿：

定义：由于长期得不到想要的资源，某进程无法向前推进的现象。

区别：可能只有一个进程发生饥饿

死循环：

定义：某进程执行过程中一直跳不出某个循环的现象。

区别：死循环是程序员的问题

3、死锁产生的必要条件

(1)互斥条件：多个进程争夺资源发生死锁
(2)不剥夺条件：进程获得的资源不能由其它进程强行抢夺
(3)请求和保持条件：某个进程有了资源，还在请求资源
(4)循环等待条件：存在资源的循环等待链

4、什么时候会发生死锁

(1)对系统资源的竞争
(2)进程推进顺序非法
(3)信号量的使用不当也会造成死锁

5、死锁的处理策略

(1)预防死锁
(2)避免死锁
(3)死锁的检测和解除

死锁的处理策略——预防死锁

1、不允许死锁发生

静态策略：预防死锁
(1)破坏互斥条件（有些不能破坏）
　　把互斥的资源改造为共享资源
(2)破坏不剥夺条件（复杂，造成之前工作失效，降低系统开销，会全部放弃、导致饥饿）
　　方案1：当请求得不到满足的时候，立即释放手里的资源
　　方案2：由系统介入，强行帮助剥夺
(3)破坏请求和保持条件（资源利用率极低，可能会导致某些进程饥饿）
　　采用静态分配方法，一次性全部申请，如果申请不到，不要允许
(4)破坏循环等待条件（不方便增加新的设备，实际使用与递增顺序不一致，会导致资源的浪费，必须按规定次序申请资源）
　　顺序资源分配法：对资源编号，进程按编号递增顺序请求资源
动态检测：避免死锁

2、允许死锁发生

死锁的检测和解除

死锁的处理策略——避免死锁

1、什么是安全序列?
进行后面的某些情况，不会使系统发生死锁

2、什么是系统的不安全状态，与死锁有何联系?
如果系统处于安全状态，就一定不会发生死锁。如果系统进入不安全状态，就可能发生死锁（处于不安全状态未必就是发生了死锁，但发生死锁时一定时在不安全状态）

3、如何避免系统进入不安全状态——**银行家算法**
初始分配完成后，优先全部分配给最少的，并且拿回资源
　　步骤：

> 1、检查此次申请是否超过了之前声明的最大需求数
>
> 2、检查此时系统剩余的可用资源是否还能满足这次请求
>
> 3、试探着分配，更改各数据结构
>
> 4、用安全性算法检查此次所分配是否会导致系统进入不安全状态

安全性算法:

把可用资源分配给可分配的，找出一个合法序列

相关概念:

Available为一开始的剩余可用资源(n)

Allocation为当前已经分配的资源(n*m)

Request为进程请求资源(n*m)

Max为资源的最大需求(n*m)

Need为进程请求的资源，Need=Max-Allocation(n*m)

Work为实时的剩余可用资源

死锁的处理策略——检测和解除

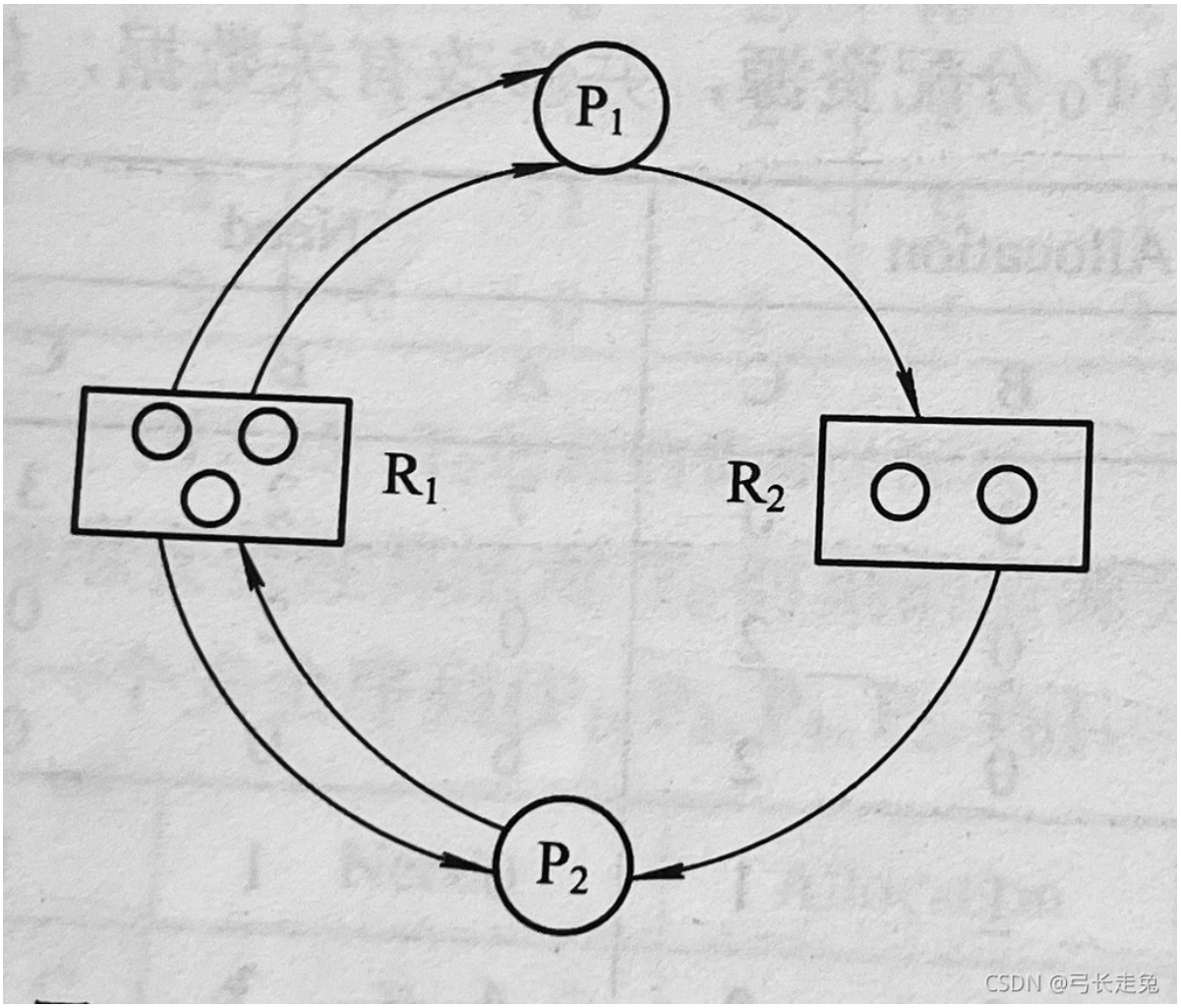
1、死锁的检测

(1)用某种数据结构来保存资源的请求和分配信息

(2)提供一种算法，利用上述信息来检测系统是否已进入死锁状态

2、死锁定理

画出资源分配图



圆为进程，方为某种资源，方里面的圆为资源数量

指向进程的箭头表示已经分配的资源（资源->进程)

从进程指出的箭头表示进程请求的资源（进程->资源）

空闲资源=资源总数-资源出度

if资源入度<空闲资源，那么该入度所对应的进程可以完成

if所有进程都可以完成，那么不会发生死锁

3、死锁的解除

(1)资源剥夺法：挂起某些死锁进程，并抢占它的资源，将这些资源分配给其他的死锁进程。

(2)撤销进程法：强制撤销部分，甚至全部死锁进程，并剥夺这些进程的资源。

(3)进程回退法：让一个或多个死锁进程回退到足以避免死锁的地步。

第三章 内存管理

内存的基础知识

1、什么是内存

存储单元：每个地址对应一个存储单元

内存地址：

2、进程运行的基本原理

指令的工作原理：

逻辑地址vs物理地址：逻辑地址就是相对地址

从写程序到程序运行：编辑-编译-链接-装入

三种链接方式：静态链接（在程序运行前，先将各目标模块及它们所需的库函数连接成一个完整的可执行文件）、装入时动态链接（将各目标模块装入内存时，边装入边链接的链接方式）、运行时动态链接（在程序执行中需要该模块时，才对它进行链接，其优点时便于修改和更新。）

三种装入方式：绝对装入（在编译的时候就知道程序放在内存的哪个位置）、可重定位装入/静态重定位（装入时将逻辑地址转表为物理地址）、动态运行时装入/动态重定位（把地址转化推迟到程序真正要执行时才进行）

内存管理的概念

内存管理的主要功能：

1、内存空间的分配与回收

2、内存空间的扩充

内存的虚拟性

3、地址转换

逻辑地址和物理地址转换

4、存储保护

设置上下限寄存器
采用重定位寄存器（基址寄存器）和界地址寄存器（限长寄存器）

内存管理单元MMU，Memory Management Unit用来将进程使用的逻辑地址转化为物理地址

覆盖与交换

内存空间的扩充

覆盖技术：将程序分为多个段，内存分为”固定区“和”覆盖区“，需要常驻的放在”固定区“，调入后就不再调出，不常用的段放在”覆盖区“，其他段放在外存，需要用到时调入内存，用不到时掉出内存

交换技术：内存空间紧张时，系统将内存中某些进程暂时换出外存，把外存中某些已具备运行条件的进程换入内存（PCB会常驻内存，不会被换出）

虚拟存储技术：

连续分配管理方式

连续分配方式

1、单一连续分配：内存被分配为系统区和用户区，系统区在低地址，用户区是一个用户独享，无外部碎片，有内部碎片。

1、固定分区分配：将用户区分割为若干固定分区给各道程序，分割策略有**分区大小相等**和**分区大小不相等**，可以建议一个分区说明表来管理各个分区，有内部碎片。

3、动态分区分配：可变分区分配，不会预先划分内存分区，而是在进程装入内存时，根据进程的大小动态地建立分区，并使分区的大小正好适合进程的需要，无内部碎片，有外部碎片。

内部碎片：分配给某进程的内存区域中，有些部分没有用上造成空间浪费

外部碎片：是指内存中的某些空闲分区由于太小而难以利用（如果有外部碎片，可以采用紧凑技术，紧凑技术需要重定位寄存器的支持）

动态分区分配算法

1、首次适应算法（First Fit)

算法思想：每次从低地址开始查找，找到第一个能满足大小的空闲分区

缺点：每次查找都需要经过小碎片区

2、最佳适应算法（Best Fit）

算法思想：为了保证“大进程”到来时能有连续的大片区域，可以尽可能留下大片的空闲区，优先使用更小的空闲区。

空闲分区按容量递增次序链接，分配内存时顺序查找空闲分区链

缺点：产生更多外部碎片

缺点：会留下小碎片

3、最坏适应算法（Worst Fit）

算法思想：和最佳适应算法相反，按容量递减次序排列，每次尽可能用大的分区

缺点：很快就导致没有可用的大内存块，大空间容易被用完。

4、领近适应算法（Next Fit）

也叫循环首次适应算法

算法思想：每次从上次查找结束的位置开始检索

缺点：大空间容易被用完

基本分页存储管理的基本概念

分页管理不会产生外部碎片

1、基本概念

把主存空间划分为大小相等且固定的块，块相对较小，作为主存的基本单位。进程也以块划分，执行时以块为单位申请主存中的块空间。注意申请的空间不一定相邻。

连续分配：为用户进程分配连续的内存空间

非连续分配：为用户进程分配分散的内存空间

页/页面（Page）：进程中的块

页帧（Page Frame）：内存中的块

块/盘块（Block）：外存中的块

2、地址结构

0-11为为页内偏移量/页内地址W，12-31为页号P

31 12	11 0
页号 P	页内偏移量 W

每页4KB

最多允许2^20页

3、页表

每个进程有一张页表

便于进程在内存中找到每个页面对应的块，存放页号和块号的对应关系

由页号和块号组成

基本地址变换机构

地址变换机构的任务是将逻辑地址转化为物理地址

页表寄存器（PTR），存放页表在内存中的**起始地址F**和**页表长度M**，进程未执行时，页表的起始地址和页表的长度放在进程控制块（PCB）中，当进程被调度时，操作系统内核会把它们放在页表寄存器中。

设页面大小L，逻辑地址A到物理地址E的变换过程：
1.页号P=A/L；页内偏移量W=A%L
2.比较页号P和页表长度M，若P>=M，产生越界中断
3.页号P对应的页表项地址=页表始址F+页号P * 页表项长度，取出其内容b，即为物理块号
4.计算E=b * L + W，用得到的物理地址E去访问内存

具有快表的基本地址变换结构

TLB（Translation Lookaside Buffer）相联存储器，是一种访问速度比内存快很多的**高速缓冲存储器**，用来存放当前访问的若干页表项，以加速地址变换的过程。与此对应，内存中的页表常称为慢表。

1、局部性原理

时间局部性：访问某个变量后，在不久的将来还会被访问

空间局部性：程序访问了某个存储单元，不久之后，其附近的存储单元也很有可能被访问

2、引入快表后

如果快表命中，只需访问一次内存
快表未命中，需要访问两次内存

两级页表

1、单级页表存在什么问题？如何解决？

所有页表项必须连续存放，页表过大时需要很大的连续空间

在一段时间内并非所有页面都用得到，因此没必要让整个页表常驻内存

2、两级页表的原理、逻辑地址结构

将长长的页表再分页

逻辑地址结构：（一级页号、二级页号、页内偏移量）

31 22	21 12	11 0
一级页号	二级页号	页内偏移量

别称：页目录表、外层页表、顶级页表

3、如何实现地址变换？

按照地址结构将逻辑地址拆分成三部分

从PCB中读出页目录表始址，根据一级页号查**页目录表**，找到下一级页表在内存中的存放位置

根据二级页号查表，找到最终想访问的内存块号

结合页内偏移量得到物理地址

4、两级页表问题需要注意的几个细节

多级页表中，各级页表的大小不能超过一个页面。若两级页表不够，可以分更多级

多级页表的访问次数（假设没有快表结构）——N级页表访问一个逻辑地址需要N+1次访存

基本分段存储管理方式

1、什么是分段？

进程的地址空间：按照程序自身的逻辑关系划分为若干个段，每段有段名，每段从0开始编址

段号的位数决定了每个进程最多可以分几个段

段内地址位数决定了每个段的最大长度是多少

2、什么是段表

段表：段映射表

每个程序被分段后，用段表记录该程序在内存中存放的位置

段表：段号 段长 基址

段号	段长	基址
0	7K	80K
1	3K	120K
2	6K	40K

3、如何实现地址变换

逻辑地址结构：

31 16	15 0
段号	段内地址

4、分段、分页管理的对比

页：信息的物理单位，实现离散分配，提高内存利用率，地址是一维的，访存两次

段：信息的逻辑单位，对系统可见，地址是二维的，访存两次

分段比分页更容易实现信息的共享和保护（不能被修改的代码称为纯代码和可重入代码，不属于临界资源）

段页式的管理方式

1、分页、分段管理方式最大的优缺点

分页：利用率高，碎片少，不方便进行信息共享和保护

分段：方便信息共享和保护，如果段长大，容易产生外部碎片

2、分段+分页的结合——段页式管理方式

先分段再分页

段表结构：

段号	页表长度	页表存放块号
0	2	1
1	1	...
2	2	...

逻辑地址结构：段号+页号+页内偏移量

31 16	15 12	11 0
段号	页号	页内偏移量

地址结构是二维的

3、段表、页表

4、如何实现地址变换

虚拟内存的基本概念

1、传统存储管理方式的特征、缺点

之前讲的

一次性：作业必须全部装入内存后才能开始运行，并发性下降

驻留性：一旦作业被装入内存，就会一直驻留在内存

2、局部性原理

时间局部性
空间局部性
高速缓存技术

3、虚拟内存的定义和特征

虚拟内存最大容量是计算机地址结构确定的

虚拟内存的实际容量=min(内存和外存容量之和，CPU寻址范围)

eg：某计算机地址结构为32位，按字节编址，内存大小为512MB，外存大小为2DB.

则虚拟内存的最大容量为 2^32B=4GB

虚拟内存的实际容量=min(2^32B,512MB+2GB)=2GB+512MB

多次性：无需在作业运行时一次性全部装入内存，而是允许被分成多次调用内存

对换性：在作业运行时无需一直常驻内存，而是允许在作业运行过程中，将作业换入换出

虚拟性：从逻辑上扩充了内存的容量，使用户看到的内存容量，远大于实际的容量

4、如何实现虚拟内存技术

在程序执行过程中，当所访问的信息不再内存时，由操作系统负责将所需信息从外存调入内存，然后继续执行程序。

若内存空间不够，由操作系统负责将内存中暂时用不到的信息换出到外存。

请求分页管理方式

1、页表机制

请求分页存储的页表：

页号	内存块号	状态位	访问字段	修改位	外存地址
0	无	0	0	0	x

状态位P：是否已经调入内存

访问字段A：一段时间内的访问次数，或最近多长时间未访问

修改位M：调入内存后是否修改过

外存地址：在外存中的地址

2、缺页中断机构

内中断，可被修复

3、地址变换机构

页面置换算法

1、最佳置换算法（OPT）

（Optimal Page Replacement Algorithm）

每次选择淘汰的页面是以后永不使用或者在最长时间内不再被访问的页面，这样可以保证最低的缺页率。

实际上不知道后面的序列

2、先进先出置换算法（FIFO）

每次选择淘汰的页面是最早进入内存的页面

Belady异常，当分配的内存块增大时，缺页次数反而增加

3、最近最久未使用置换算法（LRU）

Least Recently Used

每次淘汰最近最久未使用的页面

4、时钟置换算法（最近未用算法，CLOCK）（NRU）

简单的：最多经历两轮扫描，初始为1，扫一下为0，再扫一下被踢

替换指针指向被替换页的下一页（初始为0）

选择淘汰时访问位为1就置0并后移指针继续检查，访问位为0就换出。

5、改进型的时钟置换算法

优先淘汰没有被修改过的，因为没有修改过的不用进行IO操作

访问位A修改位M

00->01（改）->00->01

页面分配策略

1、驻留集

指请求分页存储管理中给进程分配的物理块的集合

2、页面分配、置换策略

固定分配局部替换：驻留集大小不可改变

可变分配全局替换：可以将操作系统保留的空闲物理块分配给缺页进程

可变分配局部替换：只能选进程自己的物理块置换

3、调入页面的时机

预调页策略：一次调用若干个相邻页面，运行前调入

请求调页策略：运行时缺页再调入

4、从何处调页

对换区：快，采用连续分配方式

文件区：慢，采用离散分配方式

5、抖动（颠簸）现象

刚刚换出的又要换入，刚刚换入的又要换出，物理块不够

（频繁的页面调度）

6、工作集

指在某段时间间隔里，进程要访问页面的集合

可以由时间t和工作集窗口大小Δ确定

例如访问次序如下：1,4,2,3,5,3,2(t),2,1
如果Δ=5，那么窗口则为t之前访问的5个页面，即{2,3,5,3,2}
去重之后则是工作集{2,3,5}

一般来说驻留集要大于工作集

第四章 文件管理

文件的逻辑结构

1、无结构文件

文件由一系列二进制文件流组成

2、有结构文件（记录式文件）

顺序文件：文件中的记录一个接一个顺序排列，定长或变长，可以顺序存储或者链式存储

按照是否与关键字顺序有关，可以分为串结构和顺序结构

链式：无法随机存取

顺序存储：

可变长：无法随机存取
定长：可以随机存取，采用串结构，无法快速找到关键字；采用顺序结构，可以快速查找关键字
索引文件：索引表本身是定长的顺序文件

索引顺序文件：多级索引表嵌套查找

文件的物理结构

1、顺序结构

目录项存储：start、length

2、链接分配

（1）隐式连接：

目录项存储：start、end

指针在盘块中

（2）显式连接

整个文件系统仅有一张，文件分配表FAT，指针存在FAT里面

FAT通常在内存里面

3、索引分配

目录项存储：index（索引块的块号/索引指针）

文件目录

1、文件控制块（FCB）

搜索、创建文件、删除文件、显示目录、修改目录

2、目录结构

单级目录结构
两级目录结构
　　主文件目录（MFD）+用户文件目录（UFD）

多级目录结构（树形目录结构）
　　当代操作系统采用方法、不便于文件共享

无环图目录结构
可以共享
3、索引节点（对文件控制块

压缩文件名和信息

文件的物理结构（文件分配方式）

1、对非空闲磁盘块的管理

(1)连续分配：连续分配方式要求每个文件在磁盘上占有一组连续的块，对文件的拓展不方便，有很多磁盘碎片

(2)链接分配

隐式分配：采用链接分配方式的文件，只支持顺序访问，不支持随机访问，方便拓展
显示分配：文件分配表显式记录下一块物理块的位置，方便拓展，支持随机访问，文件表会占内存空间

(3)索引分配

索引分配允许文件离散地分配在各个磁盘块中，系统会为每个文件建立一张索引表，索引表记录了文件的各个逻辑块对应的物理块

支持随机访问

链接方案
多层索引
混合索引

文件保护

1、口令保护

2、加密保护

保密性强，不需要在系统中存储“密码”

编码/译码，需要花费一定时间

文件系统的层次结构

用户/应用接口

用户接口

文件目录系统

存取控制模块

逻辑文件系统与文件信息缓冲区

物理文件系统

辅助分配模块　　设备管理模块

第五章 IO管理

磁盘

1、磁盘地址：柱面号.盘面号.扇区号

磁盘调度算法

1、FCFS先来先服务

2、STTF最短寻道时间优先

3、SCAN/LOOK电梯调度

每次寻找同方向最小

4、C-SCAN/C-LOOK循环电梯调度

每次寻找同方向最小，转向后直接到最远

启动相关

1、操作系统启动

ROM中的引导程序->磁盘引导程序->分区引导程序->操作系统初始化程序

2、磁盘启动

磁盘物理格式化(划分扇区)->磁盘分区->磁盘逻辑格式化(初始化文件系统)->安装操作系统