

# 操作系统-复习笔记

曹烨 2021012167 软23 [caoye541@gmail.com](mailto:caoye541@gmail.com)

前言：操作系统是一门好课.....期末占比低，只要完成好大作业就行，期末考试大致复习一下，然后拟合一下作业题和往年题。

本次复习笔记按照课程介绍、知识点大纲式复习、重点算法考点研究、题型速览来进行讲解。

总而言之：Petri！！

## 课程介绍

- 考核方式：大作业 50%、随堂抽测课后作业 10%、期末考试 40%

- 通讯录：

闻立杰 清华大学东配楼11区405 13681026629 [wenlj@tsinghua.edu.cn](mailto:wenlj@tsinghua.edu.cn)

孟诗奥 清华大学东配楼11区401 15937308719 [msa21@mails.tsinghua.edu.cn](mailto:msa21@mails.tsinghua.edu.cn)

马福崑 清华大学东配楼11区401 13051339001 [thss15\\_mafk@163.com](mailto:thss15_mafk@163.com)

- 课程大纲：

- 第一章-操作系统概述

- 第二章-进程管理+Petri网

进程管理、处理器调度与进程间通信、死锁、PN\_CPN

- 第三章-存储管理

- 第四章-设备管理

- 第五章-文件系统

## 知识点大纲式复习

### 第一章：操作系统概述

大概会提及：计算机和计算机系统的基本概念和组成、计算机技术的发展、OS的定义作用组成和分类、一些OS相关的基本概念。

- 电子数字计算机：种能够自行按照已设定的程序进行数据处理的电子设备，是软件与硬件相结合、面向系统、侧重应用的自动化求解工具。
- 计算机系统组成：硬件子系统 CPU（运算、控制、存储）、主存储器、I/O控制系统、外围设备 和软件子系统 系统软件、支撑软件和应用软件

关键的系统软件：**操作系统和语言处理程序**

- 计算机技术的发展：
  - 没有操作系统时期，一次完成一个功能。
  - 串行的单通道批处理。CPU和I/O设备忙闲不均。

- 多通道批处理（现代意义的操作系统）。宏观上并行，微观上串行。等待I/O时可以使用CPU。
- 分时系统。多个用户通过终端分享一台计算机。CTSS、MULTICS、UNIX
- 现代操作系统：受保护指令、系统调用、内存保护、中断、I/O、时钟
- 操作系统（OS）
 

定义：OS是计算机系统最基础的 系统软件，管理软硬件资源、控制程序执行、改善人机界面、合理组织计算机工作流程，为用户使用计算机提供良好的 运行环境。

作用：操作系统是方便 用户管理和控制计算机软硬件资源的系统程序集合，在整个计算机系统中具有承上启下的地位。管理计算机系统的各种资源、用户与机器的接口、大型软件系统。

组成：进程调度子系统、进程通信子系统、内存管理子系统、设备管理子系统、文件管理子系统、网络通信子系统

类型：多道批处理操作系统，脱机控制方式；分时操作系统，交互式控制方式；实时操作系统，即时响应方式

接口：操作接口——系统程序；两类作业级接口——脱机作业控制方式 作业控制语言、联机作业控制方式 操作控制命令；程序接口——操作系统为程序运行扩充的编程接口
- OS的其他深入性概念
  - 驱动程序：最底层的、直接控制和监视各类硬件(或文件)资源的部分。职责是屏蔽或隐藏底层硬件的具体细节，并向其他部分提供一个抽象的、通用的接口
  - 资源共享的方式：独占、并发
  - 资源分配策略：静态分配、动态分配、资源抢占
  - 多道程序设计（指让多个程序同时进入计算机的主存储器运算）
 

特点：CPU与外部设备充分并行、外部设备之间充分并行、发挥CPU的使用效率、提高单位时间的算题量、单道程序的运算时间会增加

利用 进程 来实现，OS能管理各类资源在进程间的使用。实现要点：调用操作系统提供的服务例程 来使用资源，调用程序 来复用CPU，用设备控制器与通道 来使CPU和I/O设备充分并行，用中断 来让正在运行的程序让出CPU
  - OS规定了合理操作计算机的工作流程
  - 系统调用：操作系统实现的完成某种特定功能的过程，为所有运行程序提供访问操作系统的接口
 

实现细节：陷入处理机制 计算机系统中控制和实现系统调用的机制；陷入指令 也称访管指令或异常中断指令，计算机系统为实现系统调用而引起处理器中断的指令；每个系统调用事先规定了编号，并在约定寄存器中规定了传递给内部处理程序的参数

实现要点：处理程序、入口地址表、现场保护区

## 第二章：进程管理

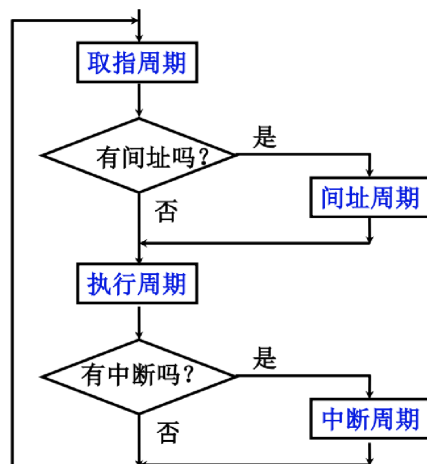
第二章的目录为：2.1 处理器；2.2 中断处理；2.3 进程管理；2.4 线程管理；2.5 处理器调度；2.6 进程间通信；2.7 死锁。

第二章算是操作系统的大头之一（真正的主人还是PETRI），在课上讲了三章来阐释，我们将分节来进行快速回顾

## 2.1 处理器

- 控制与状态寄存器：用于控制处理器的操作，主要被具有特权的操作系统程序使用，以控制程序的执行 程序计数器PC、指令寄存器IR、条件码CC、标志位
- 程序状态字PSW：OS的概念，值记录当前程序运行的动态信息 程序计数器、中断字；也是计算机系统的寄存器
- 通用寄存器（8类，4个数据寄存器、4个指针寄存器及变址寄存器）
- 段寄存器（4个：CS 代码段寄存器、DS 数据段寄存器、SS 堆栈段寄存器、ES 附加段寄存器）
- 指令指针IP：指令指针IP是一个16位专用寄存器。（P指向的是指令地址的段内地址偏移量，又称偏移地址(Offset Address)或有效地址(EA, Effective Address))
- 标志寄存器FR：在FR中有意义的有9位，其中6位是状态位，3位是控制位。
- 机器指令：是计算机系统执行的基本命令，是中央处理器执行的基本单位。由一个或多个字节组成 操作码字段、多个操作数地址、状态字和特征码。完成各种算术逻辑运算、数据传输、控制流跳转。
- 指令执行过程：取指 根据PC从存储器或高速缓冲存储器中取指令到IR、译码 解译IR中的指令来决定其执行行为、执行 连接到CPU部件，执行运算，产生结果并写回，同时在CC里设置运算结论标志；跳转指令操作PC，其他指令递增PC值

取指周期&执行周期



- 特权指令：只能被操作系统内核使用的指令（I/O指令、置PC指令）
- 处理器模式：内核模式（0模式 OS内核）、用户模式（3模式 用户程序等保护级别）

模式切换：中断、异常或系统异常等事件导致用户程序向OS内核切换，触发用户模式→内核模式；OS内核处理完成后，调用中断返回指令（如Intel的iret）触发内核模式→用户模式

## 2.2 中断管理

- 中断的概念：暂时中止CPU上现程序的运行，转去执行相应的事件处理程序，待处理完成后返回原程序被中断处或调度其他程序执行的过程。

OS是中断驱动的，中断是激活操作系统的唯一方式。

- 狭义的中断：处理器外的中断事件，即与当前运行指令无关的中断事件 I/O中断、时钟中断、外部信号中断。

异常：运行指令引起的中断事件 地址异常、算术异常、处理器硬件故障等

系统异常：指执行陷入指令而触发系统调用引起的中断事件 请求设备、请求I/O、创建进程等

- 几类中断源：
  - 硬件故障：由处理器、内存储器、总线等硬件故障引起的中断事件

保护现场、停止设备、停止CPU、报告等待人工

- 程序性：处理器执行机器指令引起的中断事件

算数异常简单处理报告用户、非法指令or存取等指令异常终止进程、虚拟地址异常调整内存重新执行

- 自愿性：处理器执行陷入指令请求OS服务引起的中断事件，又称作系统调用 请求分配外设、请求I/O

陷入OS、保护现场、查入口地址、跳转处理程序

- I/O：来源于外围设备，用于报告I/O状态的中断事件

I/O完成则调整进程状态、释放阻塞、加入就绪进程；I/O出错or异常等待人工

- 外部：由外围设备发出的信号引起的中断事件

- 中断系统：是计算机系统中响应和处理中断的系统，包括硬件子系统 中断响应 和软件子系统 中断处理 两部分

- 中断装置：是计算机系统中发现并响应中断/异常的硬件装置

处理器外的中断：由 中断控制器 发现和响应

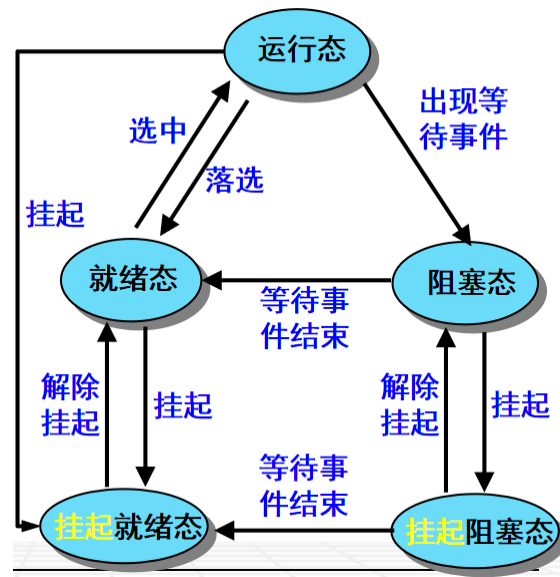
处理器内的异常：由 指令的控制逻辑和实现线路 发现和响应，相应机制称为陷阱

请求OS服务的系统异常：处理器执行陷入指令时直接触发，相应机制称为系统陷阱

- 中断控制器：是CPU中的一个控制部件，包括中断控制逻辑线路和中断寄存器
- 陷阱与系统陷阱：是指令的控制逻辑与实现线路的一部分
- 中断响应过程：1. 发现中断源，触发中断请求 2. 中断当前程序的执行 3. 转向操作系统的中断处理程序
- 中断屏蔽：当检测到中断时，中断装置通过中断屏蔽位决定是否响应已发生的中断（有选择的响应中断）
- 中断优先级：即中断装置响应中断顺序的
- 中断的嵌套处理：当响应中断后在处理中断的过程中还可以再响应其他中断。（有硬件需求，有限定层数）（嵌套可能改变处理次序，可能先响应后处理）
- 多中断的响应及处理原则：中断屏蔽可以使中断装置不响应某些中断；中断优先级决定了中断装置响应中断的次序；中断可以嵌套处理，但嵌套层数应有限制；中断的嵌套处理改变了中断处理的次序

## 2.3 进程管理

- 进程的概念：是具有一定独立功能的程序关于某个数据集合的一次运行活动。是操作系统进行资源分配和CPU调度的一个独立单位
- 进程的具体定义（包含以下实体部分）：(OS管理运行程序的)数据结构P；(运行程序的)内存代码C；(运行程序的)内存数据D；(运行程序的)通用寄存器信息R；(OS控制程序执行的)程序状态字信息PSW
- 进程的三态模型：运行态、阻塞态、就绪态
- 进程挂起：OS无法预期进程的数目与资源需求，计算机系统会在运行过程中出现资源不足的情况 运行资源不足：性能低和死锁。解决方案就是进程挂起：剥夺某些进程的内存及其他资源，调入OS管理的对换区，不再参加进程调度，待适当时候再调入内存、恢复资源、参与运行
- 挂起态与阻塞态有着本质区别，后者占有已申请到的部分资源处于等待其他资源的状态，前者则没有任何资源

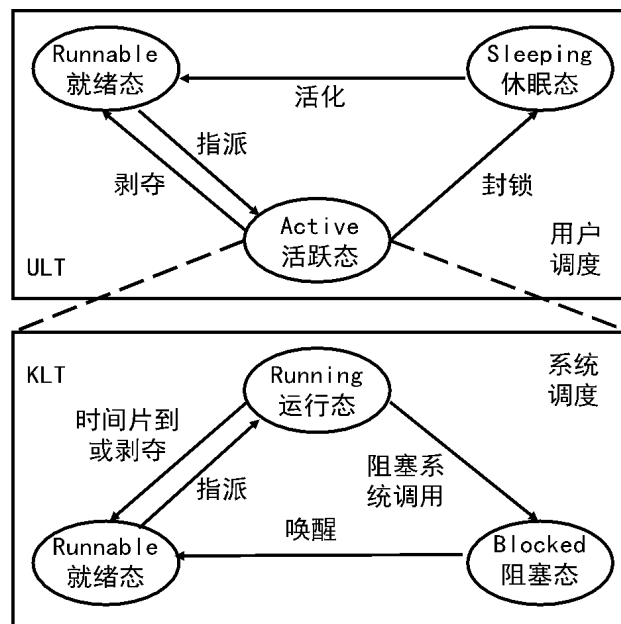


- 进程的特性：动态性，独立性，并发性
- 进程的创建：在一个已经存在的进程（用户进程或系统进程）当中，通过系统调用来创建一个新的进程（Unix:fork；Windows:CreateProcess）
- 进程的数据结构：进程控制块PCB。用于记录和可画进程状态及环境信息的数据结构。标识信息、现场信息、控制信息
- 进程映像：某一时刻进程的内容及其执行状态的集合。进程映像时内存级的物理实体又称为进程的内存映像。
- 进程上下文：进程的执行需要环境支持，包括CPU现场和高速缓存中的执行信息。用户级上下文、寄存器上下文、系统级上下文。刻画了进程的执行情况。
- 队列管理模块：操作系统实现进程管理的核心模块
  - 操作系统建立多个进程队列，包括就绪队列和阻塞队列
  - 按需组织为先进先出队列与优先队列
  - 队列中的进程可以通过PCB中的队列指引采用单/双向索引连接
  - 出队和入队操作
- 原语：是由若干指令构成的完成某种特定功能的程序，执行时有不可分割性。进程使用的是进程控制原语。
- 进程切换：从正在运行的进程中收回处理器，让待运行进程来占有处理器运行。保存被中断进程的上下文、转向进程调度、恢复待运行进程的上下文
- 模式切换：进程切换必须在操作系统内核模式下完成，必须切换模式（又称处理器状态切换）中断装置完成正向模式切换、中断返回指令完成逆向模式切换。

## 2.4 线程管理

- 单线程在并发程序设计上的问题：进程切换开销大、进程通信开销大、限制进程并发的粒度、降低了并行计算的效率
- 线程的提出：把进程的两项功能，即独立分配资源与被调度分派执行分离开来。进程作为系统资源分配和保护的独立单位，不需要频繁地切换；线程作为系统调度和分派的基本单位，能轻装运行，会被频繁地调度和切换。
- 在多线程环境中，进程是操作系统中进行保护和资源分配的独立单位。用来容纳进程映像的虚拟地址空间对进程、文件和设备的存取保护机制
- 线程是进程的一条执行路径，是调度的基本单位，同一个进程中所有进程共享进程获得的主存空间和资源。

- 线程状态：运行、就绪、阻塞。（挂起为资源相关，属于进程）
- 线程调度：1. OS感知线程环境下：处理器调度对象是线程、进程没用三种状态只有挂起；2. OS不感知线程环境下：调度对象仍是进程，用户空间中的用户调度程序调度线程。
- 并发多线程优点：快速线程切换、减少（系统）管理开销、（线程）通信易于实现、并行程度提高、节省内存空间
- 内核级线程KLT：线程管理的所有工作由OS内核来做，OS直接调度KLT 需要模式切换，系统开销较大
- 用户级线程ULT：用户空间运行的线程库，提供多线程应用程序的开发和运行支撑环境，内核未意识到线程的存在 仅有一个ULLT能执行、一个阻塞将引起整个进程阻塞
- Jacketing技术：把阻塞式系统调用改造成非阻塞式；由jacketing程序来检查资源使用情况，从而决定是否执行进程切换或传递控制权给另一个线程
- 多线程实现的混合策略：单应用的多个ULT可映射成一些KLT，通过调整KLT数目，可以达到较好的并行效果
- 混合策略下的三态：KLT三态，系统调度负责；ULT三态，用户调度负责；活跃态ULT代表绑定KLT的三态；活跃态ULT运行时可激活用户调度；非阻塞系统调用可使用Jacketing启动用户调度，调整活跃态ULT



## 2.5 处理器调度

- 处理器调度层次
  - 高级调度 长程调度、作业调度：决定是否加入到执行的进程池中
  - 中级调度 平衡负载调度：决定主存的可用进程集合；提供内存利用率和作业吞吐量
  - 低级调度 短程调度、进程调度：决定哪个可用进程占用处理器执行；按照某种原则把处理器分配给就绪态进程或内核级线程
- 处理器调度算法的相关计算式
  - 周转时间:  $T_i = E_i - S_i$
  - 平均周转时间:  $T = \frac{1}{N} \sum_{i=1}^N T_i$
  - 平均带权周转时间:  $W = \frac{1}{N} \sum_{i=1}^N \frac{T_i}{r_i}$
- 调度算法
  - 先来先服务算法FCFS
  - 短作业有限SJF（属于优先级算法）CPU运行时间：可以得到最小平均周转时间



- 时间片轮转调度算法RR：时钟中断中把现在执行的进程扔到队尾。（公平性、活动性）（时间片q太大退化为FCFS，太小系统开销大。）（一半设置为20-50ms）
- 优先级算法（分为可抢占、不可抢占）（分为静态、动态）
  - 静态：创建进程的时候就确定进程优先级（缺点：高优先级一直占用CPU）
  - 动态：创建进程时赋给进程的优先级，在进程运行过程中可以动态改变；每执行一个时间片就降低优先级，等待时间延长就提高优先级
  - 优先级队列法：按照不同优先级分组，各级别之间使用优先级，统一级别之间使用时间片轮转。但是会出现优先级反转。
  - 多级队列算法：先按照性质和类型（系统、用户、批处理）将就绪队列分为子队列，不同队列可以采用不同的调度算法。
- 多级反馈队列（RR和优先级算法综合发展）：优先级越高的队列时间片越短，如果用完了时间片还没进行完，则降级，如果提前用完，则升级。只有高优先级队列为空时才调度低优先级的。照顾短作业/IO繁忙/新进程。CPU繁忙进程采用最大时间片来执行，减少调度次数。

## 2.6 进程间通信

- 并发程序设计：把一个具体问题求解设计成若干个可同时执行的程序模块的方法。【并发性、共享性、交往性】
- 进程互斥：并发进程之间因相互争夺独占性资源而产生的竞争制约关系
  - 进程同步：并发进程之间为完成共同任务基于某个条件来协调执行先后关系而产生的协作制约关系
- 临界区：指并发进程中与互斥共享变量相关的程序段。（临界资源-互斥共享变量所代表的资源）
  - 临界区管理的三个要求：1. 一次至多允许一个进程停留在相关的临界区内；2. 一个进程不能无限止地停留在临界区内；3. 一个进程不能无限止地等待进入临界区（临界区的指令长度须短小精悍，这样才能保证系统效率）
- 记录型信号量：一种带数值的软资源；每个信号量建立一个阻塞进程队列；每个信号量相关一个整数值 正值表示资源可用资源个数或者可复用次数；零值表示无资源可用且无进程阻塞；负值的绝对值表示阻塞队列中进程个数
- P原语（尝试减少、申请资源）：申请一个空闲资源(把信号量减1)，若成功，则退出；若失败，则该进程被阻塞
  - N原语（尝试增加、释放资源）：释放一个被占用资源(把信号量加1)，若发现有被阻塞的进程，则选择一个唤醒之。
- 管程：管程试图抽象相关并发进程对共享变量的访问，提供一个友好的并发程序设计开发环境；由若干公共变量及说明和所有访问这些变量的进程所组成；管程的局部变量只能由该管程的过程存取；进程只能互斥地调用管程中的过程
  - 管程的条件变量：条件变量、同步原语wait、同步原语signal
- 低级通信：只能传递状态与整数值（信号量 面对用户、信号 面对os）
  - 高级通信：传递任意数量数据。（共享内存、管道、信息传递）
    - 共享内存：把所有共享数据放在共享内存区域，任何想要访问该共享数据的进程都必须在本进程地址空间开辟一块内存区域，来映射存放共享数据的物理内存。
      - （三份缓冲区：共享区 存放共享变量、写入区、读取区）（读和写都需要先把共享区做一个映射，映射到本地划出来的空间，在本地访问）
    - 消息传递：向中间件发送消息，订阅后获得消息。分为直接通信（利用套接字）和间接通信（利用消息队列）两种模式。

- 管道：一个进程的输出直接连接到另一个进程的输入。（本质是伪文件、内核缓冲区）

## 2.7 死锁

- 死锁：一组进程处于死锁状态是指：每一个进程都在等待被另一个进程所占有的、不能抢占的资源。
- 死锁产生：竞争资源产生死锁、PV操作不当产生死锁、同类资源分配不当引起死锁、对临时性资源使用不加限制引起死锁
- 解决死锁的三个思路：死锁防止、死锁避免、死锁检测和恢复
- 死锁产生的四个条件：互斥条件、持有和等待条件、不可剥夺条件、循环等待条件
- 死锁的防止（思路是破坏四个条件之一即可）
  - 把独占型资源改造成共享性资源，使资源可被同时访问而不是互斥使用。（针对 互斥条件）
  - 采用剥夺式调度方法可以破坏第三个条件
  - 静态分配：进程在执行中不再申请资源，因而不会出现持有了某些资源再等待另一些资源的情况，即破坏了第二个条件
  - 在层次分配策略下，资源被分成多个层次（破坏了第四个条件）
- 死锁的避免：掌握并发进程中与每个进程有关的资源申请情况，避免死锁的发生
- 安全状态的两个条件：本身不存在死锁问题；存在着某种调度顺序，使得即使在最坏的情况下，每个进程仍都能够顺利的运行结束。
- 死锁的检测：此方法对资源分配不加限制，但系统定时运行一个死锁检测程序，判断系统内是否已出现死锁，若检测到死锁则设法加以解除
- 检测方法：设置两张表格来记录进程使用资源的情况。**等待资源表**记录每个被阻塞进程等待的资源；**占用资源表**记录每个进程占有的资源。  
检测等待占用关系 $W(P_i, P_j)$ ，列出所有 $W$ ，判断是否出现了循环等待问题。
- 资源分配图：进程、资源、进程占用资源、进程请求资源未果被阻塞  
如没用环路、不会有死锁；有环路，每种资源实例数目，1必然死锁，n可能死锁  
(多资源可以进行化简)
- 死锁检测后的方法：可以采用重新启动进程执行的办法，恢复工作应包含重启一个或全部进程，以及从哪一点开始重启动

---

## 特殊章节0：Petri Net

Petri在OS课上的重要性，无需多言

### 0.0 概述术语

- 并发、顺序、冲突、同步、异步
- 网模型论 (SNT) {模型部分}  
有向网（模型基础）：结构特征、结构性质  
网系统（资源+规则）：动态性质、分析方法、层次结构
- 通用网论 (GNT) {理论部分}{借用SNT描述自然现象，具有普适性}  
Synchrony（同步论，研究事件间的同步及定量描述，即同步距离）  
Concurrency（并发公理，准确认识与定义并发现象，无传递性）




Enlogy (网逻辑, 关注系统中无发生权的事件, 来推导系统性质)


Net Topology (网拓扑, 无向网, 用于沟通离散与连续的工具)


Information Flow Net (信息流网, 可逆信息元件, 无信息丢失)

## 0.1 有向网

- 定义: 三元组  $N = (S, T; F)$  称为有向网, 如果  
 $S \cup T \neq \emptyset \wedge S \cap T = \emptyset \wedge F \subseteq S \times T \cup T \times S \wedge \text{dom}(F) \cup \text{cod}(F) = S \cup T$

$S$  元素:  place 场所 (类似仓库, 存储同类东西, 无固定坐标)

$T$  元素:  transition 变迁 (变化含质变与量变, 不是直接搬家)

$F$  关系:  flow relation 流关系 (资源在网上的流动)

- 有向网规则: 不能有重复弧 (基础概念定义金律: 没有必要包含的, 就有必要不包含)  
有向网定义不包含: 连通性、单纯性、简单性、有限性

- $X = S \cup T$  为有向网的节点集合

$\cdot x = \{y | (y, x) \in F\}$ :  $x$  的前集     $x \cdot = \{y | (x, y) \in F\}$ :  $x$  的后集

## 0.2 Petri网系统

- Petri网定义 (根据上下文决定): 1. 有向网; 2. 以有向网为基础的网系统; 3. 网系统: 模型 (SNT) + 理论 (GNT); 4. 模型 + 理论 + 应用。

- Petri网系统:

- EN系统: 描述个体活动, 有初始状态
- CE系统: 描述自然活动, 有当前状态
- P/T系统: 描述同类个体
- 高级网系统: 描述不同类个体 (Pr/T系统)
- C\_net非线性系统

- EN系统**:  $\sum = (B, E; F, c_{in})$  【条件:  $(B, E; F)$  为有向网, 且  $c_{in} \in 2^B$ 】

$2^B$ :  $B$  的幂集合, 即  $B$  的所有子集的集合

$B$  的库所: 两种状态-有托肯或无托肯, 称为条件

$E$  中的变迁称为事件, 只与条件有关

$c_{in}$  由真的条件组成 (有托肯的库所), 称为  $\sum$  的初始情态

- 丛:  $B$  的子集称为  $\sum$  的丛

- 变迁规则的定义:

$e \in E$ , 在丛  $c$  有发生权, 记为:  $c[e >$ , 如果  $\cdot e \subseteq c \wedge e \cdot \cap c = \emptyset$

$c[e >$ ,  $c' = (c - \cdot e) \cup e \cdot$  称为  $c$  的后继丛, 记为  $c[e > c'$ , 即  $e$  在  $c$  发生, 并将  $c$  变成  $c'$

- 同步与异步

同步 (局部): 两组 (个) 事件反复发生呈现的规律 如鼓掌、走路、四季系统

同步 (全局): 任何两组事件都 (加权) 同步

异步 (全局): 没有全局控制, 局部确定

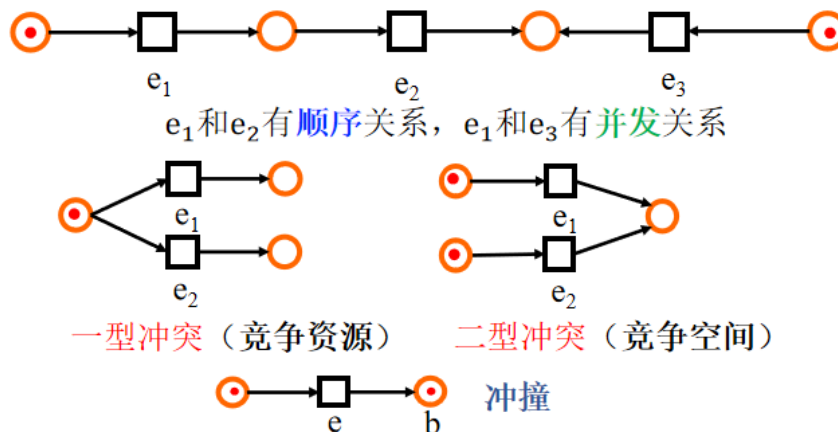
Petri网是异步并发系统（局部确定，所以并发）

- 可达情态集：

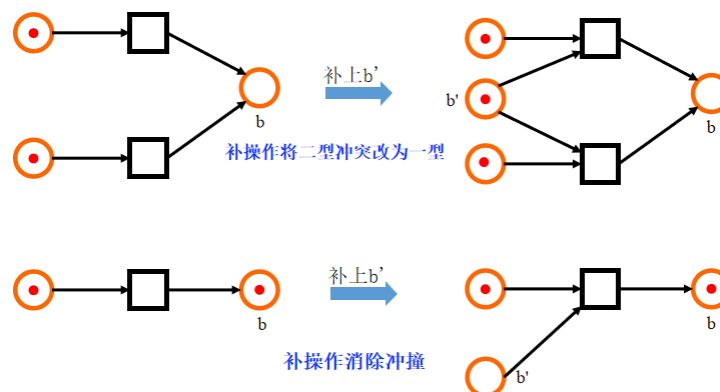
从 $c_{in}$ （初始情态）开始，经有限多个事件发生所产生的所有后继丛构成的集合称为EN系统  
 $\Sigma = (B, E; F, c_{in})$ 的可达集，用 $C_\Sigma$ 表示。 $C_\Sigma$ 中的丛称为 $\Sigma$ 的情态。

- 基本现象

顺序关系、并发关系、冲突关系、冲撞关系



- 结构互补条件： $b_1$ 和 $b_2$ 称为结构互补条件，如果一方前集后集为另一方后集前集，即  
 $\cdot b_1 = b_2 \quad \wedge \quad b_1 = \cdot b_2$
- 条件补操作：将 $b$ 的结构互补条件 $b'$ 添加到网上；如果 $b$ 中有token， $b'$ 中无token，否则 $b'$ 添一token



- 条件/事件系统 (C/E系统)：** 区别是C为完全可达集，用来描述自然现象。
- 库所/变迁系统 (P/T系统)：**  $\Sigma = (S, T; F, K, W, M_0)$   
 $K$ ：S元素的容量函数  
 $W$ ：F上的权函数，表示资源消耗或产生的量  
 $M_0$ ：初始标识
- 标识（资源分布）： $M : S \rightarrow \{0, 1, 2, \dots\}$ 称为 $\Sigma$ 的表示，如果 $\forall s \in S : M(s) \leq K(s)$
- 发生权： $t$ 在 $M$ 有发生权，记作 $M[t >$
- 后继：若 $M[t >$ ，则 $t$ 可以发生， $M'$ 为 $M$ 的后继标识，后继关系记作 $M[t > M'$
- 外延： $\cdot t \cup t \cdot$ 称为变迁 $t$ 的外延
- 局部确定原理：每个变迁都有它的外延；变迁的发生只依赖其外延也只改变其外延
- 库所的物理意义：可观察的同类资源；同类资源在系统中作用相同
- EN $\in$ P/T，在EN中 $K=1, W=1, M_0$ 是 $C_{in}$ 的另一种表现形式
- 并发步：在同一标识可以并发的一个或多个变迁，称为该标识上的一个并发步

- 可达标识集： $[M_0]$ 称为 $\Sigma$ 的可达标识集 命题  $M \in \vdash[M_0]$ ，则 $[M] \subseteq \vdash[M_0]$
- 变迁序列： $\sigma = \tau_1\tau_2 \dots \tau_n$ 称为 $\Sigma$ 的变迁序列， $M_n$ 称为 $\sigma$ 的后继标识，记为 $M_0 \vdash[\sigma]M_n$ 。
- P/T系统的性质：活性、有界性、公平性

## 第三章：存储管理

大概内容为：3.1 存储管理基础、3.2 单连续分区存储管理、3.3 页式存储管理、3.4 段式存储管理、3.5 段页式存储管理

### 3.1 存储管理基础

- 逻辑地址：即用户变成所使用的地址空间 相对地址、虚地址 从0开始编号
- 段式程序设计：把一个程序设计成多个段 代码段、数据段、堆栈段等；用户可以自己应用段覆盖技术 扩充内存空间使用量 这一技术是程序设计技术，不是OS存储管理的功能
- 物理地址：程序执行所用的地址空间 绝对地址、实地址
- 主存储器的复用：按照分区复用、按照页框复用
- 存储管理的基本模式：单连续存储管理 一维逻辑地址空间的程序占用一个主存固定分区或可变分区；段式存储管理 段式二维逻辑地址空间的程序占用多个主存可变分区；页式存储管理 一维逻辑地址空间的程序占用多个主存页框区；段页式存储管理 段式二维逻辑地址空间的程序占用多个主存页框区
- 存储管理的功能：
  - 地址转换：地址重定位，即把逻辑地址转成物理地址 静态重定位：在程序装入内存时进行地址转换， 动态重定位：在CPU执行程序时进行地址转换
  - 主存储器空间的分配与回收
  - 主存储器空间的共享（多个进程共享主存储器资源、多个进程共享主存储器的某些区域）
  - 存储保护（为避免主存中的多个进程相互干扰，必须对主存中的程序和数据进行保护；这一功能需要软硬件协同完成）
  - 主存储器空间的扩充
- 虚拟存储器思想的提出：主存容量限制带来诸多不便；用户编程行为分析；因此可以考虑部分调入进程内容
- 虚拟存储器的基本思想：存储管理把进程全部信息放在辅存中，执行时先将其中的一部分装入主存，以后根据执行行为随用随调入；如果主存中没有足够的空闲空间，存储管理需要根据执行行为把主存中暂时不用的信息调出到辅存上去
- 虚拟存储器的实现思路：(辅存)虚拟地址空间 容纳进程装入；(主存)实际地址空间 承载进程执行
- 高速缓存存储器（Cache）：Cache是介于CPU和主存储器间的高速小容量存储器  
由高速存储器、联想存储器、地址转换部件、替换逻辑等组成  
分级：
  - L1 Cache：分为数据缓存和指令缓存；内置；其成本最高，对CPU的性能影响最大；通常在32-256KB之间
  - L2 Cache：分内置和外置两种，后者性能低一些；通常在512KB-8MB之间
  - L3 Cache：多为外置，在游戏和服务器领域有效；但对很多应用来说，总线改善比设置L3更加有利于提升系统性能
- 存储管理与硬件支撑：

- 鉴于程序执行与数据访问的局部性原理，存储管理软件使用Cache可以大幅度提升程序执行效率
- 动态重定位、存储保护等，若无硬件支撑在效率上是毫无意义的，即无实现价值
- 无虚拟地址中断，虚拟存储器无法实现
- 无页面替换等硬件支撑机制，虚拟存储器在效率上是毫无意义的

### 3.2 单连续分区存储管理

- 单连续分区存储管理：每个进程占用一个物理上完全连续的存储空间(区域) 单用户连续分区存储管理、固定分区存储管理、可变分区存储管理
- 单用户连续分区存储管理
  - 主存区域划分为系统区与用户区
  - 设置一个栅栏寄存器以分隔两个区域，硬件用它在执行时进行存储保护
  - 一般采用静态重定位进行地址转换
  - 硬件实现代价低
  - 适用于单用户单任务操作系统，如DOS
- 主存分配表（固定分区存储管理） 支持多个分区、分区数量固定、分区大小固定、可用静态重定位、硬件实现代价低、早期OS采用

分区号	起始地址	长度	占用标志
1	8K	8K	0
2	16K	16K	Job1
3	32K	16K	0
4	48K	16K	0
5	64K	32K	Job2
6	96K	32K	0

- 可变分区存储管理：按照进程的实际内存需求动态划分分区，并允许分区个数可变
  - 按进程的内存需求来动态划分分区
  - 创建一个进程时，根据进程所需主存量查看主存中是否有足够的空闲空间 若有，则按需要量分割一个分区；若无，则令该进程等待主存资源
  - 由于分区大小按照进程实际需要量来确定，因此分区个数是随机变化的

#### ❖ 已分配区表与未分配区表，采用链表

起址	长度	标志
4k	6k	J1
46k	6k	J2
		空
		空
⋮	⋮	⋮

(a)已分配区情况表

起址	长度	标志
10k	36k	未分配
52k	76k	未分配
		空
		空
⋮	⋮	⋮

(b)未分配区情况表

- 可变分区方式的内存分配：最先适应(first-fit)分配算法；下次适应(next-fit)分配算法；最佳适应(best-fit)分配算法；最坏适应(worst-fit)分配算法
- 地址转换与存储保护：硬件实现机制与动态重定位。
- 可变分区方式的内存碎片：
  - 固定分区方式会产生内存内碎片
  - 可变分区方式会随着进程的内存分配产生一些小的不可用的内存分区，称为内存外碎片
  - 最佳适应分配算法最容易产生外碎片
  - 任何适应分配算法都不能避免产生外碎片
- 移动技术：用动态重定位，来用移动分区来解决内存外碎片

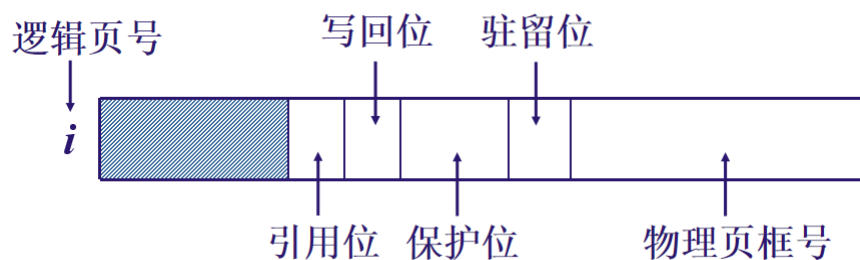
### 3.3 页式存储管理

- 基本原理：
 

分页存储器将主存划分成多个大小相等的页框；受页框尺寸限制，程序的逻辑地址也自然分成页；不同的页可以放在不同页框中，不需要连续；页表用于维系进程的主存完整性

页式存储管理的逻辑地址由两部分组成 页号和页内偏移量；页式存储管理的物理地址由两部分组成 页框号和页内偏移量；地址转换可以通过查页表完成

页的共享：页式存储管理能够实现多个进程共享程序和数据 数据共享：不同进程可以使用不同页号共享数据页；程序共享：不同进程必须使用相同页号共享代码页
- 页表存储管理的地址转换：给定逻辑地址，计算逻辑页号和页内偏移量；查找页表，找到相应的物理页框号；计算最终的物理地址
- 逻辑地址划分：逻辑页号和页内偏移量。这种划分由系统自动完成。由于页面的大小一般为2的整数次幂，因此，地址的高位部分即为页号，低位部分即为页内偏移量。 $(PW) \text{ 逻辑页号} = \text{虚地址} / \text{页大小}$   $\text{页内偏移} = \text{虚地址} \% \text{页大小}$
- 转换代价：页表放在主存，每次转换必须访问两次主存；降低了存取速度；解决方案是利用Cache存放部分页表
- 页式存储管理的优缺点：优点-没有外碎片、一个程序不必连续存放、便于管理；缺点-一次性全部装入内存、为每个进程维护一张页表
- 局部性原理（共享物理空间，前提是程序执行遵从局部性原理）：程序在执行过程中的一个较短时期，所执行的指令地址和指令的操作数地址，分别局限于一定区域。
- 页表项的格式与内容：
  - 驻留位（有效位）：表示该页是否在内存。若该位为1，表示该页位于内存中，即该页表项有效，可以使用；若该位为0，表示该页当前在外存中，此时若访问该页表项，将导致缺页中断；
  - 保护位：表示允许对该页做何种类型的访问，如只读、可读写、可执行等；
  - 写回位（修改位）：表明此页在内存中是否被修改过。当系统回收该物理页面时，根据此位来决定是否把它的内容写回外存；
  - 引用位（访问位）：如果该页面被访问过，则设置此位。用于页面置换算法。



- 缺页处理：1.保护CPU现场2. 分析中断原因3. 转入缺页中断处理程序进行处理4. 恢复CPU现场，继续执行
- 页面调度：选择淘汰页的工作，目标：尽可能减少页面的换进换出次数。
- 缺页中断率：  $f = F / A$  （P运行中成功访问次数为S，不成功访问次数为F，总访问次数  $A = S + F$ ）  
影响因素：页框数、页面的大小、程序编制方式
- **页面调度/置换算法**
  - 最佳Opt页面调度算法 只可模拟、不可实现  
理想的调度算法是：当要调入新页面时，首先淘汰以后不再访问的页，然后选择距现在最长时间后才访问的页
  - 最近较久未用LRU页面调度算法 实现代价大  
淘汰最近一段时间较久未被访问的那一页，即那些刚被使用过的页面，可能马上还要被使用到
  - 最少使用LFU页面调度算法  
淘汰最近一段时间内访问次数较少的页面，对Opt算法的模拟性比LRU更好 基于时间间隔中断，并给每一页设置一个计数器；时间间隔中断发生后，所有计数器清0；每访问页1次就给计数器加1；选择计数值最小的页面淘汰
  - 先进先出FIFO页面调度算法  
总是淘汰最先调入主存的那一页，或者说主存驻留时间最长的那一页(常驻的除外)
  - 时钟Clock页面调度算法 FIFO + 跳过刚被访问的页面  
采用循环队列机制构造页面队列，形成了一个类似于钟表面的环形表；队列指针则相当于钟表面上的表针，指向可能要淘汰的页面；使用页引用标志位
  - 二次机会EClock页面置换算法 FIFO + 跳过已经读|写的页面  
需要用到页表项当中的访问位(R)和修改位(M)。如果一个页面被读取，硬件自动将其R位置为1，如果一个页面被修改，则将M位置1。
- 工作集：一个进程当前正在使用的逻辑页面集合，可以用一个二元函数  $W(t, \Delta)$  来表示 是当前的执行时刻； $\Delta$  称为工作集窗口，即一个定长的页面访问窗口； $W(t, \Delta)$  = 在当前时刻  $t$  之前的  $\Delta$  窗口当中的所有逻辑页面所组成的集合
- 驻留集：驻留集是指在当前时刻，进程实际驻留在内存当中的逻辑页面集合。（驻留集取决于系统分配给进程的物理页面数目，以及所采用的页面置换算法）
- 缺页率：缺页率表示“缺页次数/内存访问次数”(比率)或“缺页的平均时间间隔的倒数”。  
影响因素：页面置换算法、分配给进程的物理页面数目、页面本身的大小、程序的编制方法
- 反置页表：页号；进程标志符；标志位；哈希链
- 二级页表：由于不是所有的虚拟地址都会用到，所以不必把所有的页表项都保存在内存当中。缺省情形下，Windows采用了一种二级页表结构来实现逻辑地址到物理地址的映射。
- 一个32位的逻辑地址被划分为三个部分：页面目录索引（10位）、页表索引（10位）、页内字节索引（即页内偏移地址，12位）。



### 3.4 段式存储管理

- 页式存储管理（和分区存储管理）只有一个逻辑地址空间，即一维的线性连续空间。
- 段表的具体实现
  - 段表保存在内存当中；
  - 设置一个段表基地址寄存器，用来指向内存当中段表的起始地址；
  - 设置一个段表长度寄存器，用来指示段表的大小，即程序当中的段的个数；

段号+对应内存分区的起始地址+段长度
- 优缺点：
  - 程序通过分段来划分多个模块，每个模块可以分别编写和编译，可以针对不同类型的段采取不同的保护，可以按段为单位来进行共享
  - 一个程序不必连续存放，没有内碎片
  - 缺点：程序必须全部装入内存、外碎片等。
- 虚拟存储管理的段表扩充：特征位：00(不在内存)、01(在内存)、11(共享段) 存取权限：00(可执行)、01(可读)、11(可写) 扩充位：0(固定长)、1(可扩充) 标志位：00(未修改)、01(已修改)、11(不可移动)

### 3.5 段页式存储管理

- 基本思想：先把程序按逻辑划分为段，然后在段内等分为页

段号	段内地址	
	页号	页内地址

内存划分：按照**段式**存储管理方案  
内存分配：以**页面**为单位进行分配

- 具体实现：
  - 具体实现段表：记录每个段的页表起始地址和段长，而不是该段所在内存分区的起始地址
  - 页表：记录逻辑页面号与物理页框号之间的对应关系。(每一段有一个，一个程序可有多页表)
  - 需要的硬件支持：段表基地址寄存器（STBR）和段表长度寄存器（STLR）

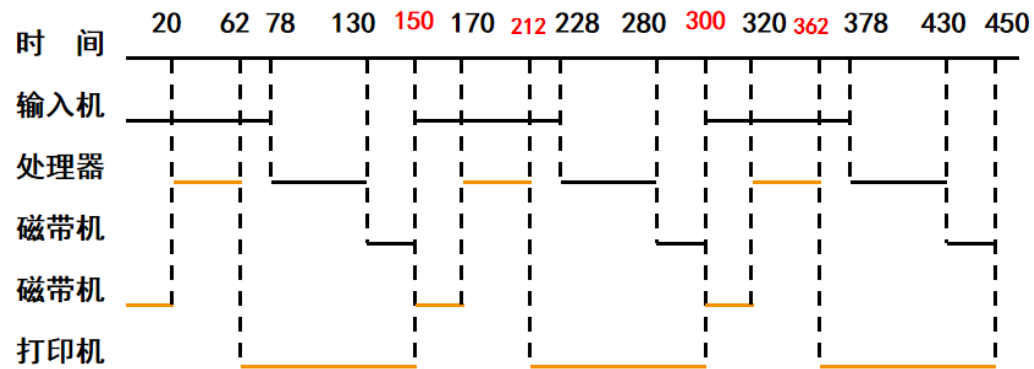
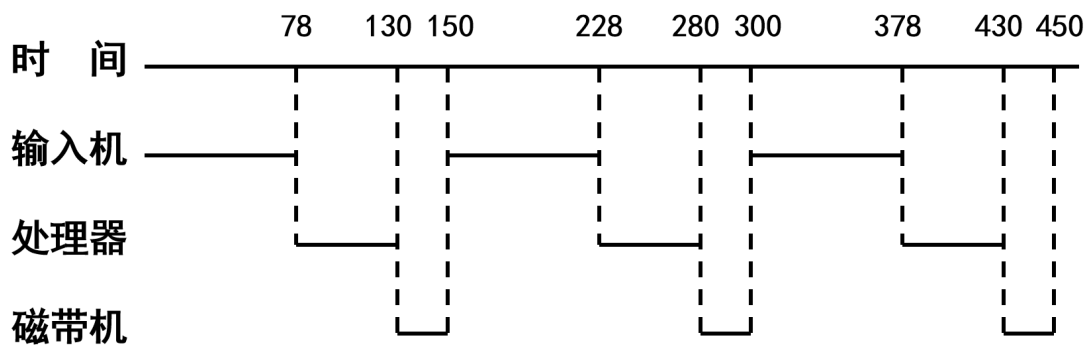
---

## 题型速览

这里将PPT、作业、参考题列出来分析。仅供过拟合用。

### From PPT

#### T1（PPT1 p68）（处理器利用率计算）



第一个进程（单序程序工作）：利用率 =  $\frac{(130-78)+(280-228)+(430-378)}{450} \times 100\% = 34.67\%$

第二个进程（双序程序工作）：

利用率 =  $\frac{(62-20)+(212-170)+(362-320)+(130-78)+(280-228)+(430-378)}{450} \times 100\% = 62.67\%$

## T2（PPT3 p101）（哲学家就餐问题）

```

1 //算法一：不完全正确可能死锁
2 #define N 5 // 哲学家个数
3 void philosopher(int i) // 哲学家编号：0-4
4 {
5     while (TRUE)
6     {
7         think(); // 哲学家在思考
8         take_fork(i); // 去拿左边的叉子
9         take_fork((i + 1) % N); // 去拿右边的叉子
10        eat(); // 吃面条中...
11        put_fork(i); // 放下左边的叉子
12        put_fork((i + 1) % N); // 放下右边的叉子
13    }
14 }

```

```

1 //算法二：改进了拿叉子的过程但是仍然不正确
2 while(1)          // 去拿两把叉子
3 {
4     take_fork(i);    // 去拿左边的叉子
5     if(fork((i + 1) % N)) { // 右边叉子还在吗
6         take_fork((i + 1) % N); // 去拿右边的叉子
7         break;        // 两把叉子均到手
8     }
9     else {           // 右边叉子已不在
10        put_fork(i);    // 放下左边的叉子
11        wait_some_time( ); // 等待一会儿
12    }
13 }

```

```

1 //算法三：等待时间随机变化（可行但非万全之策）
2 while(1)          // 去拿两把叉子
3 {
4     take_fork(i);    // 去拿左边的叉子
5     if(fork((i + 1) % N)) { // 右边叉子还在吗
6         take_fork((i + 1) % N); // 去拿右边的叉子
7         break;        // 两把叉子均到手
8     }
9     else {           // 右边叉子已不在
10        put_fork(i);    // 放下左边的叉子
11        wait_random_time( ); // 等待随机长时间
12    }
13 }

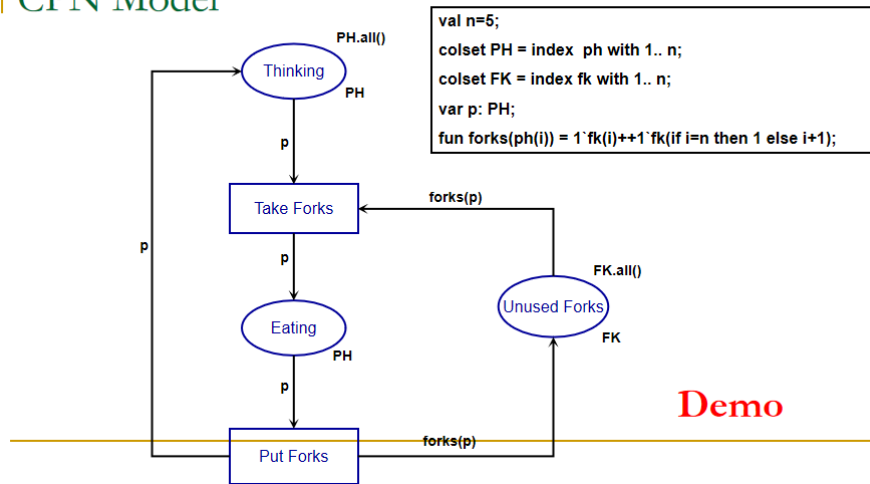
```

```

1 //算法四：互斥访问。正确但是每次只允许一个人进餐
2 semaphore mutex;    // 互斥信号量，初值1
3 void philosopher(int i) // 哲学家编号i: 0-4
4 {
5     while(TRUE){
6         think( );    // 哲学家在思考
7         P(mutex);    // 进入临界区
8         take_fork(i);    // 去拿左边的叉子
9         take_fork((i + 1) % N); // 去拿右边的叉子
10        eat( );        // 吃面条中...
11        put_fork(i);    // 放下左边的叉子
12        put_fork((i + 1) % N); // 放下右边的叉子
13        V(mutex);    // 退出临界区
14    }
15 }

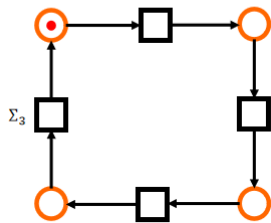
```

## CPN Model



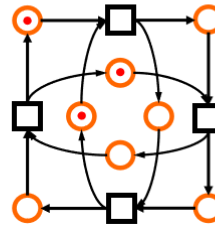
### T3 (PPT5 P61) (EN系统, 四季系统)

$\Sigma_3$ 是四季变化合格的EN系统吗?



- 条件（状态）可观察吗?
- 事件（季节改变）可观察吗?

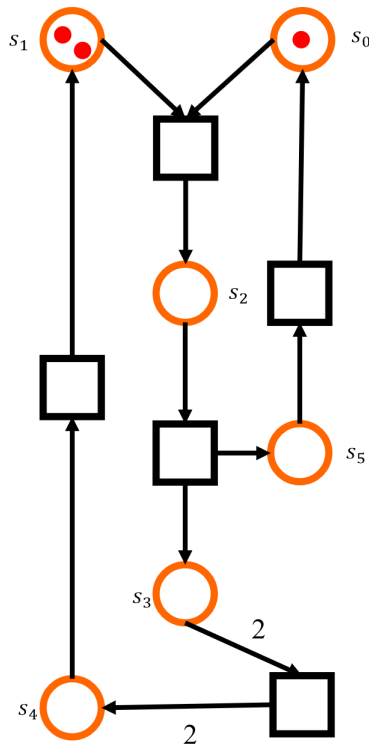
合格的四季系统什么样?

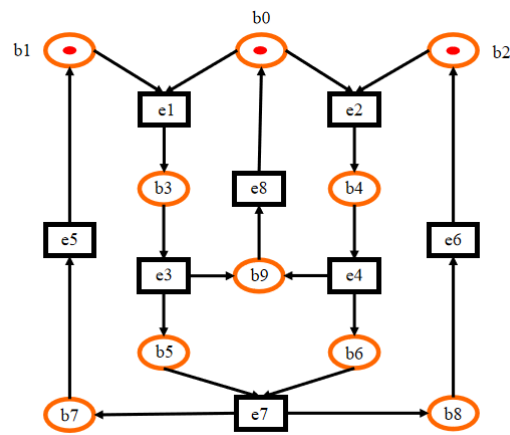


S完备化  
伴随库所  
并发论!

- 本季为主，上一季的影子，下一季的味道
- 满足并发（出现）公理的最小系统
- 有向网上的完备化操作+同步距离计算

### T4 (PPT5 P89) (教堂问题之EN、PT图)





## T5 (PPT6 P81) (页面调度置换)

请求分页管理系统中，假设某进程的页表内容如下表所示。

页号	页框(Page Frame)号	有效位
0	101H	1
1	—	0
2	254H	1

页面大小为4KB，一次内存的访问时间是100ns，一次快表（TLB）的访问时间是10ns，处理一次缺页的平均时间为 $10^8$ ns（已含更新TLB和页表的时间），进程的驻留集大小固定为2，采用最近最少使用置换算法（LFU）和局部淘汰策略。假设①TLB初始为空；②地址转换时先访问TLB，若TLB未命中，再访问页表（忽略访问页表之后的TLB更新时间）；③有效位为0表示页面不在内存，产生缺页中断，缺页中断处理后，返回到产生缺页中断的指令处重新执行。设有虚地址访问序列2362H、1565H、25A5H，请问：

(1) 依次访问上述三个虚地址，各需多久？(2) 基于上述访问序列，则虚地址1565H的物理地址是多少？

(1) 页面大小为4KB，即212，则得到页内偏移占虚地址的低12位

2362H: P=2, 访问快表10ns, 因初始为空, 访问页表100ns得到页框号, 合成物理地址后访问主存100ns, 共计: 10ns+100ns+100ns=210ns

1565H: P=1, 访问快表10ns, 落空, 访问页表100ns落空, 进行缺页中断处理 $10^8$ ns, 合成物理地址后访问主存100ns, 共计: 10ns+100ns+ $10^8$ ns+100ns

25A5H: P=2, 访问快表, 因第一次访问已将该页号放入快表, 因此花费10ns便可合成物理地址, 访问主存100ns, 共计: 10ns+100ns=110ns

(2)

当访问虚地址1565H时，产生缺页中断，合法驻留集为2，必须从页表中淘汰一个页面，根据题目的置换算法，应淘汰0号页面，因此1565H的对应页框号为101H。由此可得1565H的物理地址为101565H。