OS复习

1.2 操作系统简史

1. 人工操作阶段. 人工穿孔纸带，独享资源
2. 单道批处理系统. 一次性输入多个用户作业，自动执行
   单道程序设计，CPU与I/O不能并行
3. 多道批处理系统 多个程序交替执行，吞吐量大
   周转周期长(排队)，无交互能力
4. 分时操作系统 划分CPU时间片，"同时"运行；时钟中断
   "公平性" 支持多用户、交互性、独立性(每个用户"独享")
5. 多用户多任务操作系统
   实时操作系统（银行、生产过程控制）
   嵌入式操作系统、分布式、网络.

1.3 操作系统的功能与特征

1. 对硬件资源抽象 处理器-进程 存储器-地址空间
   磁盘-文件 IO-统一接口
2. 功能
   (1) 处理机管理： 进制控制、同步、通信、调度
   (2) 存储器： 分配、回收、保护
   (3) 设备管理： IO分配、驱动、中断
   (4) 文件系统： 存储、目录、保护
   (5) 接口： 系统调用 syscall.
3. 特征； 并发、共享(互斥)、虚拟、异步(推进速度不可预知)
   (并行：多个CPU)

4. 操作系统结构

(1) 宏内核：特权集过级控制所有OS操作

性能好，复杂度高(安全与可靠性)

(2) 微内核：把一部分功能移出内核，靠消息通信

↓最小化 内核只有：中断、IPC、基本调度...

·可移植性好，但速度慢，安全、扩展性好.

(3) 混合内核架构： eg. Windows, MacOS.

(4) 外核架构：尽量暴露硬件能力.

1.4 计算机硬件简介

1. 体系结构

(1) 冯诺依曼结构： CPU、存储器(内存、寄存器)、输入输出(I/O)

↳局限：内存墙(内存跟不上CPU速度)、数据指令不去分、串行

(2) 流水线结构：乱序执行、预测执行

↓旁路攻击(缓存)

① Meltdown攻击.

·利用权限检查延后 → 读到非法数据

·数据作为索引去读Cache

·看哪里访问速度快 → 获取数据

solution：内核与应用使用不同页表(寻址不到了) ⇒但影响TLB

② Spectre攻击：

·故意让CPU预测分支错误

2. 存储层次结构

(1) 寄存器 - Cache - 内存 - 磁盘.

(2) 计算、访存与IO速度失配.

3. 指令集 ISA

(1) 两种架构：RISC(MIPS)：指令定长，寻指单一，处理快，指令少
CISC：指令变长，多种寻址，处理复杂，写代码简单

(2) 两种状态：内核态(特权)、用户态(非特权)
⇓
由中断/异常/系统调用触发，返回看CP0上寄存器

4. 寄存器与控制状态

(1) 32个通用寄存器

(2) CP0寄存器(控制与状态)：SR：CPU状态、中断
Cause：中断/异常原因

5. 中断与异常 (中断是异常的一种：异步异常)

(1) 中断：为支持CPU与设备并行，多由外部触发 ⇒返回至下一条指令
异常：CPU执行命令本身出现问题，内部触发
⇓
分类：陷阱：可恢复错误(Syscall)，去下一条指令
故障：修好可在该条重来(eg. 缺页)
终止：硬件错误.

(2) 处理：① 保存现场
② 查找异常向量(编号⇒异常向量⇒handler函数)
③ 进入handler处理
④ 恢复现场.

6. 系统调用：异常的一种.

2. 启动

1. 计算机的启动过程

(1) 启动状态单一，但系统功能复杂

(2) Bootloader程序：定位内核、加载到存储器、设置环境并执行

① 位于ROM，因为RAM启动时未知.

② 初始化硬件、建立内存映射表.

③ 两个阶段：Stage1　从ROM加载，初始化硬件 ← 汇编

Stage2　把内核读到RAM，指令寄存器到内核入口 ← C语言

↑ 本来在磁盘

(3) 初始化过程

①处理器核初始化：处理器复位、TLB、Cache初始化(设为无效)

②总线接口初始化　③设备探测与驱动加载.

2. MIPS的启动

低 kuseg：用户态、需转换、有Cache、2GB

kseg0：内核态、直接映射、有Cache、512MB

kseg1：内核态、直接映射、非Cache、512MB

高 kseg2：内核态、直接映射、Cache、1GB

由 启动最开始不能用TLB/Cache ⇒ kseg1

3. PC启动过程

(1) 对比 { U-Boot：多种架构，嵌入式系统
BIOS：x86架构，桌面/服务器，大

(2) 启动过程：

① CPU从预定位置加载BIOS → 包含硬件设置与控制

② BIOS进行硬件自检

③ 读取启动顺序，选择由哪个设备引导电脑.

④ 读取MBR(硬盘0磁头0磁道1扇区) → 主引导记录(512Byte)

↓

启动代码 + 分区表 + 幻数 → 最多四个主分区 or 3+1扩展

⑤ 启动Bootloader. 内核加载器

↳Linux上 { LILO：放在MBR上
GNU GRUB：交互界面 + 网络引导

内核加载：

⑥ 系统读取内存内核映像 → 解压缩

⑦ start_kernel 真正初始化核心数据结构、子系统

用户态启动

⑧ 根据用户层init设定运行等级（常用5）

⑨ 运行rc.sysinit，启动模块、服务脚本 → 用户登录

3.1 内存管理基础

1. 存储管理目标.

(1) 帕金森定律：无论空间多大，程序都能耗尽

(2) 用户：独立使用　　管理：同时为多个用户服务

2. 存储器硬件.

- RAM 随机访问
  - SRAM 静态　晶体管 → 快、贵、大
  - DRAM 动态　电容 → 慢、便宜、小、刷新
- ROM 只读
- Flash Memory
- Disk、Tape.

Memory Wall：CPU与内存速度差距↑

3. 存储管理的功能.

(1) 历史：
- 单道：直接物理内存 ↓
- 多道：分时复用物理内存，各占一部分，⇐ 隔离性差

(2) 要求：①进程中不是最终物理地址 ②运行前不可预测

逻辑地址(0开始) —重定位→ 物理地址

- 静态重定位：装入前确定地址（软件）
- 动态 ... ：硬件/MMU，运行时转换

(3) 动态重定位：base、limit寄存器　　base + va(?< limit)

4. 程序链接与装载.

(1) ~~程序~~：
程序：
- .bss　未初始化~~的~~的全局变量
- .data　数据段：已初始化全局变量
- .text　代码段；
- 栈：局部变量/函数参数 ⇒ 从上到下（低addr
- 堆：动态分配 malloc、free ⇒ 从下到上

(2) 目标文件(程序二进制接口) → ELF 可执行可链接 格式
ABI

① Section节：链接用
- .text 代码
- .data 已初始化全局/静态变量
- .rodata 只读数据(字符串常量)
- .bss 未初始化全局/静态变量
- .symtab 符号表.

② Segment节：执行用　合并多个section → 一个segment

③ ELF文件头：节头表、段头表

(3) 程序编译

.c —预处理 #define ...→ .i —编译 to汇编→ .s —汇编 ELF目标文件→ .o

—链接→ 可执行文件 —装载运行→

① 过程分为符号解析与重定位
- 全局符号(自己的)
- 引用 external
- 静态 static ..

- 分类
  - 静态：装载前链接：装载执行快，应用占用大.
  - 动态：
    - 装载时链接：内存中同一模块共享，但一起于难
    - 运行时链接：避免某些模块没被真正调用

② 装载：
- 分类：绝对装载、可重定位装载、动态运行时装载
- 过程：fork复制进程 → execve替换程序

※ 一个segment大小 小于 内存中大小 (有的没初始化)

5. 存储器分配方法.

(1) 单道程序内存管理：写死物理地址，静态编译(运行前)

(2) 多道程序：存储器分区.

① 固定式分区：固定把内存划死为几个分区.

- 分配方式：
  - 单一队列：一个队列对所有分区
  - 多队列：按大小，一个队列对一个分区
- 易于实现，开销小；内部碎片、并发有限

② 可变式分区　根据作业分配内存. → 开销大

- 记录方法：
  - 位图表示法：0空闲1占用，容错差
  - 链表表示法：分配单元链起来，扫描慢，相邻
- 顺序分配算法：
  - (i) 首次适应(First Fit)　每次从头找，第一个大小够的
    缺点：产生小分区
  - (ii) 下次适应(Next Fit)　从上一次结束开始找 → 更均衡
  - (iii) 最佳适应(Best Fit)　排序(小→大)，找大小刚好满足
    产生碎片
  - (iv) 最坏适应(Worst Fit)　排序(大→小)，直接给最大的
    后面大作业不够
- 评价指标：
  - 内存利用率：外部、内部碎片
  - 分配速度.

(3) 伙伴系统：内存划为 2^k 的块，可以一分为二

分配：选最合适，太大则分裂

回收：伙伴空闲合并

(4) 紧凑技术： 移动进程，把碎片空闲块合成一整个.

(5) 内存扩充？ 两种

覆盖： 程序事先分为几段，只给一部分内存，然后一段一段执行

交换： 把不用的部分先放到辅存.

3.2 页式内存管理

1. 程序、进程、作业

程序：静态文件

进程：程序执行实体、资源分配 ⟲ 多对多

作业：用户提给系统的一组任务 ⟲ 一对多 (程序、数据...)

2. 分页基本思想

(1) 页：逻辑地址空间

页框：物理内存

} 大小相同 (4KB) ⇒ 页表转换

(2) 逻辑地址：页号 + 页内位移

物理地址：块号 + 块内位移

页表项： 块号 + 有效位 ...

3. 地址变换

虚拟地址 → MMU通过页号查页框号 → 物理地址

4. 多级页表： 页目录号 - 页表号 - 页内偏移

5. TLB： 减少访存次数

Hit、Miss → 硬件/软件处理 → 硬件不灵活，软件慢

ASID → 切进程要刷TLB

6. 有效内存访问时间

TLB命中 → 查一次内存

TLB不命中 → 查两次内存.

7. (1) 反置页表：按物理页面号组织页表 → 与物理内存大小有关（适合大地址）

- 所有进程一张表：块号 → (pid, 页号)
- 查询：PID + 逻辑地址 ⟨ 逻辑页号 p / 页内偏移 d

(PID, p) —哈希→ 定位表项 → 序号即块号

(2) 页共享：多个进程映射同一物理页 → 但有些不想共享的也共享了

页保护：页表设置保护位.

3.3 段式内存管理

1. 优势：方便编程，信息共享、保护、动态链接

2. 管理方法：

(1) 地址：(二维) 段号 + 段内偏移

(2) 段表项：段基址 + 段长 + 权限 (索引：段号)

3. 缺点：复杂，且内存分配会产生碎片.

4. 段页式管理：先分段，再分页 → 访存3次

- 每个进程一张段表，每个段一张页表
- 地址：段号 + 段内页号 + 页内地址

3.4 虚拟内存管理

1. 虚存技术：← 局部性原理
   对内存分时复用，内存与磁盘swap.

2. 请求分页存储管理
   (1) MMU：访问页表，发现缺页中断 → 由内核处理装入
   (2) 页表扩展：驻留位：在物理存储/磁盘
   修改位：在内存中是否修改过
   引用位
   保护锁位
   (3) 按需调页
   预调页：预测不久将要被访问的页面
   (4) 流程：现场保护 → 找到虚拟页 → 权限检查
   → 新页面调入 → (旧页表面写回) → 更新页表 → 恢复

3. 页面置换算法
   (1) OPT最优：以后最长时间才访问的页
   (2) FIFO 先进先出：队列先进先出，淘汰最早访问 ← 性能差
   ↓ ↑ 有Belady现象：系统页数变多 → 反而命中率下降
   (3) Second Chance：若被访问过则放队列头 (到头时再放)
   ↓
   (4) Clock (NRU)：改为循环队列，设置访问标置 ← 复杂度↓
   (5) LRU 最近最少使用：被访问直接放栈顶，栈底是未使用的
   ↓ 简化 ↳ 硬件开销大
   (6) AGING 老化算法：一个8位寄存器，周用到则首位放1，然后>>

4.

(1) 工作集：进程近期活跃页面集合

驻留集：实际分配给进程的内存页框集合

↳ 过小会导致高缺页与抖动

(2) 抖动问题：内存进程数↑ → 利用率↓

驻留集↓ → 缺页↑ → 开销↑ 利用率↓

solution：局部置换 / 挂起进程

3.5 页表自映射.

1. 进程用户态访问页表：

∵ 页目录、页表都是物理页 → 映射到虚拟空间

(赋一个虚地址)

2. 4GB空间 ⟶ 1024个4MB的块

指定其中一个4MB的块(第k块)放页表(1024个页) → 页表基址 → 虚拟基址

第k页为页目录映射 ⟶ 映射至页目录

怎么做：页目录[k] = 页目录物理地址 (内核态)

⇒ 10位页号(第k块)下地址 ⟶ 对应所有页表

3. 地址推导 → 使用：查某地址的PTE、PDE：base + A

页目录索引A	页表索引B	页内偏移
10位	10位	12位

① 页表起始地址(base)：后22位为0，记前10位为A ⇒ 第A个页表

② 页目录起始：$\underbrace{A}_{10}\ \underbrace{A}_{10}\ \underbrace{0}_{12}$ ⇒ 第A页

③ 自映射PDE：$\underbrace{A}_{10}\ \underbrace{A}_{10}\ \underbrace{A}_{10}\ \underbrace{00}_{2}$

4.1 进程与线程

1. 两种执行方式

顺序执行：封闭性、可再现性

并发执行：间断性、非封闭性(互相影响)、不可再现

↓

是否发生竞争？ Bernstein条件：两进程S_1、S_2的写子集与对面读与写子集交集都为空。(只有写+写没事

2. 进程概念

(1) 进程是程序的一次计算活动，资源分配与保护的基本单位

↳ 包含代码、数据、堆栈、寄存器上下文 ...

(2) 三种状态：就绪 - 执行 - 阻塞

(3) 进程控制块 PCB：标识符、状态、现场保护区、优先级

↳ 多个链表/队列

3. 进程控制

(1) 创建：fork / exec　　撤销：kill

fork中子进程中返回0，父进程返回子进程id.

(2) 子进程结束时用exit向父进程返回，释放资源

父进程用wait等待子进程，并接收值、回收进程

⇓

(3) 僵尸进程：已经exit，等待父进程回收

孤儿进程：父进程先退出了 → 挂到init下面

4. 线程概念

(1) 进程间数据难共享(IPC) → 拆成多个并行控制流(线程)

(2) 线程：CPU调度基本单位，进程一部分，共享进程资源

↳ 有独立寄存器、栈
切换更轻量、性能↑

进程 一对多 线程

5. 线程实现方式

(1) 用户级线程：上下文切换快，应用决定，但容易阻塞进程

(2) 内核级线程：（多线程内核）内核可独立调度，但开销大

(3) 混合线程：映射（用户-内核）

分为 Many-to-One，One-to-One，Many-to-Many

4.2 进程调度

1. CPU调度：处理就绪队列

(1) 时机：阻塞 / 时间片用完 / 进程退出 / 高优先级到达

切换过程：页表（页目录）切换、上下文切换（保存原来的）

↳ 缓存失效

(2) 调度目标

用户侧：
- 周转时间：作业从提交到完全结束（批处理系统）
- 响应时间：用户请求到首次响应时间（分时系统）
- 优先级、公平性

系统：吞吐量、CPU利用率、资源利用率

算法：少饥饿、低开销

(3) 调度类型：
- 高级调度：作业提交
- 中级：哪些进程进内存
- 低级：谁拿CPU ⇒ 非抢占式 / 抢占式

2、批处理系统

(1) 指标：　周转时间：完成 - 提交　　服务时间：真实用多久

带权周转时间：　周转时间/服务时间

平均周转时间：作业周转时间和/作业数

(2) FCFS 先来先服务　　维护就绪队列，每次取队头

不利于短作业，不利于I/O多的作业　⇒利用率低

(3) SJF (Short join first) 短作业优先

对预计时间短的作业优先，不抢占．⇒长作业饥饿

(4) SRTF (Short Rest Time First) 最短剩余作业优先

SJF抢占式　　⇒长作业更加饥饿

(5) HRRF (Highest Response Ratio First) 最高响应比优先

不抢占，计算 $RP = 1 + \frac{\text{已等待时间}}{\text{要求运行时间}}$ → 长等待优先 / → 短优先

3、交互式系统

(1) Round Robin 时间片流转

过3个时间片就按 FCFS 切换 → 过长对短都不行

响应时间 = 进程数目 × 时间片长度

(2) Priority Scheduling 优先级算法

按照优先级划队列 { 静态 / 动态

(3) 多级队列算法 MLQ (Multiple Level Queue)

就绪队列 → 多个子队列 → 每个队列有调度策略

(4) MLFQ (反馈) 多级反馈队列算法.

无法预知作业时间 ⇒ 动态优先级

先假设为短任务 → 高优先级
一个时间片没结束 → 降优先级 (主动释放不管)
⇒ 优先级 ~ 时间片大小

4. 公平共享调度

(1) 彩票调度: 每个进程有不同数量彩票

每次生成随机数 → 决定调谁 运气/开销大

(2) 步幅调度 → 任务执行一次时增加的虚拟时间 ⇒ 确定性公平

5. 实时系统:

必须规定时间完成 { 硬实时 必须
软实时 大部分

(1) 周期实时任务: 周期 P 执行时间 e 使用率 $U=\frac{e}{P}$

• 可调度: $\sum \frac{e_i}{P_i} \le 1$ → 所有周期使用率和 ≤ 1

↳ 总使用率

(2) 静态表调度: 事先固定死调度顺序

(3) RMS 单调速率调度: 令优先级与周期成反比、可抢占 (静态)

↳ 该算法可调度条件: 使用率 $U=\sum \frac{e_i}{P_i} \le n(\sqrt[n]{2}-1) \to \ln 2$

静态最优算法, 开销低

(4) EDF 最早的截止期优先 = 动态优先级 可抢占 ∈ ddl近

理论上限 100%

(5) 优先级倒置 (高P被低P阻塞)

eg. A>B>C A需C的锁，A也被B抢了

① 条件：三种优先级、AC竞争资源、C的资源不可抢占

② 解决：(i)优先级置顶：占用资源进程提优先级相同→A还等C

(ii)优先级继承：优先级提升只到开锁之后

6、多处理机调度

- 非对称式多处理系统 AMP：主从机 (主负责分配)
- 对称式 SMP：机地位相同
  - 静态分配：每个一个队
  - 动态分配：公共队列

↓

自调度、成组调度、专用处理机调度

↓

进程成组

7. Linux调度

(1) O(n)调度：每次遍历就绪队列 → max priority

(2) O(1)调度：优先 → 多级队列

(3) CFS调度：O(log n) 完全公平 → 红黑树

4.3 进程同步

1. 同步与互斥问题

(1) 进程三大特征：并发，共享，不确定性

(2) 临界资源：一次仅允许一个进程访问的资源

临界区：访问临界资源的代码段

竞争条件：多进程并发访问同一数据且结果与访问顺序有关

(3) 设计准则：
- 空闲让进
- 忙则等待
- 有限等待：防饥饿
- 让权等待：让cpu资源

(4)
- 互斥：资源仅允许一个访问者进行访问
- 同步：互斥 + 实现有序访问

2. 软件互斥方法

(1) ① 用turn = P/Q ⇒ 必须交替进行 × 空闲让进(progress)

② 用Occupied = T/F ⇒ 检查+修改非原子 → 无法互斥

③ 先声明我想进，再判断对方想不想 ⇒ 非原子，可能都进不去 × 空闲让进

↓

④ 若对方想进，让权等待 ⇒ 非原子可能仍无法互斥

(2) Dekker算法 pturn / qturn：表示谁想进 turn：竞争时谁优先 = P/Q

```
P:  pturn = true
    while(qturn){
        if(turn==Q){
            pturn = false
            while(turn == Q)
            pturn = true }}
```

若两人都想进，改意愿 → 让权等待 但看turn

临界区 ← turn=1, pturn=false

缺点：忙等

(3) Peterson算法　　ptrun、qturn表示意愿，turn=P/Q表示谦让步

```
P:  ptrun = true;
    turn = P;
    while(turn=P && qturn)
    临界区
    ptrun = false
```

→ turn必然为其中一个值"互斥"

→ 克服①的交替

(4) Bakery面包店算法：用于多个进程互斥

- 进程~~抽号~~取号(顺序号) → 从小到大顺序调(相等看pid)
- ~~**~~每个进程进入临界区前：
  - ① 抓号 ← 两端用choosing是否在抓记录
  - ② 遍历所有进程 → 若在抓号：等
    若(抓了号)且(号码更小(=时有pid)
    等待
  - ③ 临界区
  - ④ 扔掉自己的号

3. 硬件互斥方法

(1) 开关中断　　关中断 → 临界区 → 开中断　　代价高、效率不行

关掉中断：无法切进程

(2) test and set (TS)指令 ：原子指令

写值到某个内存并传回旧值. bool TS(bool &x)

```
bool s=true
process Pi(){
    while(!TS(s)); → 旧值是否为True(互斥性)
临界区 ←
    s=true; → 用完开锁
}
```

↳ true为开

(3) swap指令，原子交换a与b的值

请求指针 k 、锁指针 use ：什么时候换回"开"→进临界区

但以上都基于忙等 → 应变为阻塞

(4) CAS (compare and swap)

CAS(V,A,B){若V=A → V=B
返回V原值 }

4. 信号量.

(1) 定义：(s,q)，s为资源数量（可用→负数为排队数），q为等待队列(阻塞)

强信号量：阻塞队列FIFO
弱信号量：不规定顺序

(2) P/wait 申请资源：信号量-1，若<0挂链表上(阻塞)
V/signal 释放资源：信号量+1，若≤0则唤醒一个

(3) 实现信号量(原子性) 软件/硬件

(4) 实现互斥/同步：s初始值不同（Semaphore）
互斥 → 初始值1/>1
同步 → 初始值0

同步：若A→B：A结束时V，B开始前P

汇合：a_1 b_2 / b_1 a_2 ($a_1>b_2$, $b_1>a_2$)

$a_1 \to V(B) \to P(A) \to a_2$

$b_1 \to V(A) \to P(B) \to b_2$

(5) 屏障：一堆A-D都完事后，才能进下一阶段

有n个进程：count=0；已到达数
mutex = S(1) 保护count(++)
barrier = S(0)

最后一个人打开屏障：
mutex.wait()
count++
mutex.signal
if count==n: barrier.signal

第n个释放一个 → barrier.wait
前n-1个卡在这 → barrier.signal
每放一个就放一个

(5) 类型题求解：分析并发操作 → 同步：前面人放锁，后面等
→ 互斥：按资源量拿信号量

(6) 信号量集：同时申多个资源 ← 必须一起发放

SP(S, t, d)：申d个S资源，S量小于t时不分配

(7) PV容易出现死锁

5、管程

(1) 操作系统把共享变量 + 操作函数 + 互斥机制，封装一起（定义）

- 组成：共享变量，操作函数，初始化代码、(条件变量)

(2) 条件变量：用于进程在管程内部等待某个条件

wait(c) → 因c不满足而等待，放弃互斥
put(c) → 唤醒是一个c上等待进程

- 与信号量区别：没有值，只有等待队列

(3) ~~Hoare管程~~

管程本来里面只能一个进程 ⟶ 但有了条件变量，可能多个：

eg. P唤醒Q → 有多个活动进程，怎么运行？

① Hoare管程：P等待Q执行 ↙ 入口等待队列（没进来）
↘ 紧急等待队列（在里面）→ 高P

6、进程间通信 IPC

(1) 进程同步是一种进程通信（改信号量）→ 低级通信（状态/整数值）

- 并发进程的交互需满足：同步（含互斥）、通信

(2) 管道：先进先出传送数据（一个特殊共享文件）⇒ 半双工（单向）
（间接通信）

无名管道：只能父子进程创建

有名管道 FIFO　靠路径打开，可非父子

(3) 消息队列　内核维护队列（一条一条）　结构化消息 ⟨ send / receive

(4) 共享内存　内-物理内存映射到多个进程 ⟸ 需同步机制

(5) 信号量

(6) 套接字 socket → 通信端点 ⟨ 双向、网络通信 / 开销大，复杂

(7) 信号：通知异步事件发生，轻量
用户、内核、进程都能生成信号请求

7、经典进程同步问题　（用户行为分析 - 行为关系⇒）

(1) 生产者-消费者问题

共享缓冲区有N个（产品架），生产者写入，消费者换出
任何时刻只能一个进程操作。

解：semaphore：mutex=1 互斥访问，empty=N，full=0

生产者：	消费：
P(empty)	P(full)
P(mutex)	P(mutex)
放	拿
V(mutex)	V(mutex)
V(full)	V(empty)

(2) 读者-写者问题

写者只能有一个，读者可以无限多，读写不能同时

解 Semaphore：int readcount=0，rmutex=1（互斥），fmutex=1（Data互斥）　queue=1（公平排队+控制写）

写者：
P(queue)
P(fmutex)
写 ← V(queue)
V(fmutex)

读者：P(queue)；P(rmutex)；readcount++；
if(readcount==1){P(fmutex)} V(rmutex)，V(queue)
读
P(rmutex)；readcount--；if(rc...==0){V(fmutex)}
V(rmutex)；

(3) 哲学家进餐问题

解法 {
- 最多只允许4个人尝试
- 奇数人先拿左，再拿右；偶相反
- 必须同时拿两个筷子

4.4 死锁

1. 死锁的四个必要条件
   - 互斥条件：资源只能一个人用
   - 请求与保持：拿着一个不放，再要别的
   - 不剥夺条件：无法抢占
   - 环路等待条件
2. 死锁预防
   (1) 破坏互斥：允许同时访问
   (2) 打破占有且申请：全部空闲才分配
   (3) 打破不可剥夺：抢占性
   (4) 打破循环等待：按编号顺序分配
3. 死锁避免
   (1) 安全序列：按某序列能安全运行完
   (定义) 安全状态：存在安全序列 ⟶ 运行时判断

(2) 银行家算法.

[1] 假设有n个进程，m类资源

Available [j]：j类还剩几个

减 Max [i] [j] ：进程i最多要几个j

减 Allocation [i] [j]：进程i已有几个j

Need [i] [j] ：进程i还要几个j

辅助变量：Work [i]：系统可给i用于运行的资源（假设推演）

Finish [i]：是否完成

[2] 安全检查步骤：

① 找一个finish[i]=false 且 need[i]<=work的进程Pi

② 若找到：让Pi假装执行完并释放

work = work + Allocation[i]

finish[i]= true

③ 重复1~2

④ 若全部finish=true → 安全，否则不安全

[3] 资源请求算法：进程请求资源 Request[i]

① 不能超自身需求 Req <= need

② 系统当前资源必须够多 Req <= Avail

③ 假装分配 → 再调安全检查

4. 死锁检测

(1) 资源分配图：○：进程(P) □：资源(Q)→其中小圆为一类一个资源

(2) 资源分配图(RAG)算法

① 化简RAG：若某进程req资源 → 若资源请求量均配都够用

→ req变分配 → 进程若无req，变为孤立

②死锁定理：死锁 → RAG不可完全化简

5、死锁解除

(1)撤销进程

(2)剥夺资源

5 IO管理

1. 设备接口类型

- 字符设备：键盘　　速率低、不可随机读写
- 块：磁盘驱动器　　快
- 网络：　　数据包

资源分配分类：独占、共享、虚设备(模拟共享)