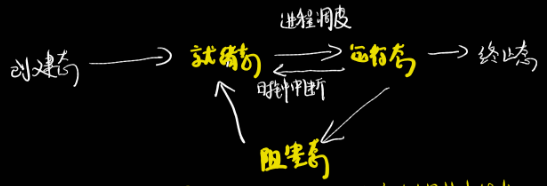
进程的状态

1. 创建态 分配资源 初始化
2. 就绪态 等待运行 拥有所有需要的资源 CPU×
3. 运行态 正在CPU中运行 拥有所有需要的资源 CPU√
4. 阻塞态 等待某事件而被阻塞
5. 终止态 回收内存资源



每个箭头都是不可逆的　黄色的是基本状态

PCB (Process Controller Block)

PID 唯一标识
STATE 状态

进程的组织方式

链接方式
执行指针 → PCB2
就绪队列指针 → PCB5 → PCB1
等待资源阻塞队列 → PCB6 → PCB7

索引方式
了解即可

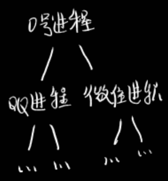
进程控制

使用 原语 实现
原语的执行具有一气呵成的特性，不可被中断
为什么原语这么强？
通过 **关闭中断** 和 **开启中断** 来实现
这两个指令只允许内核程序调用

原语

进程的创建

创建原语 {
- 申请空白PCB
- 为新进程申请资源
- 初始化PCB
- 将PCB插入就绪队列



撤销原语 {
- 从PCB集合中找到该PCB
- 剥夺CPU，分配给其他PCB
- 终止子进程
- 归还资源
- 删除PCB

↓
正常
异常
干预

阻塞和唤醒　阻塞原语　运行 → 阻塞

唤醒原语　等待 → 就绪态

切换原语　保存现场 → 其他进程
进程的寄存器
中的当前状态

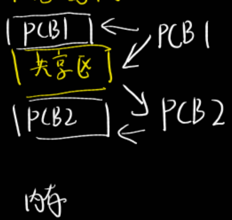
所有原语要做的事 {
- 状态
- → 在在队列
- 分配/回收资源

进程间的通信

需要操作系统的支持
进程**不可以**访问其他进程的内存空间

共享
消息
管道

1. 共享存储



在linux中　shm_open 申请
mmap　共享内存映射到自己地址空间

基于数据结构　只有限制　低级通信
基于存储区　高级通信

为避免数据覆盖，访问共享区是**互斥**的
如 P/V 操作

2、消息传递

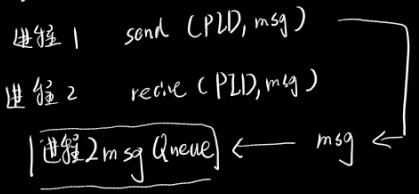
用格式化的消息传递，通过 **发送/接收** 原语

消息头 → { send PID / recive PID / length }

消息体

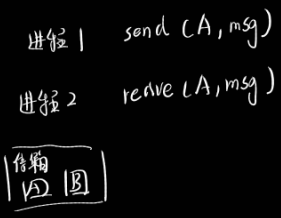
直接通信方式

将消息放入目标进程的消息队列中

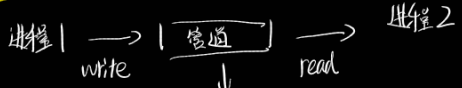


间接通信

以信箱为中介



3、管道通信



PIP文件，大小固定的内存缓冲区

如需要全双工通信则需两个管道

管道写满时 阻塞写

管道读空时 阻塞读

PIP是一个Queue，只能先进先出，半双工通信

共享存储 相比于管道通信更为自由