操作系统复习课

题型

简答题，计算比较多

- 基本概念解答
- 算法计算

基本概念

- 原子性
    - 原语

        > 一些可被调用的公用小程序，它们各自完成一个规定的操作。
        >
        > **特点**：
        >
        > 1. 处于操作系统底层，最接近硬件。
        > 2. 运行具有原子性，不会被中断。
        > 3. 运行时间都很短，且调用频繁。

    - 信号量操作
    - 进程阻塞、唤醒
- 操作系统基本特征
    - 并发
    - 共享
    - 虚拟
    - 异步
- 操作系统发展阶段
    - 手工操作阶段
    - 批处理阶段
    - 分时操作系统
    - 实时操作系统
    - 网络操作系统和分布式计算机系统
    - 个人计算机操作系统

- 线程/进程

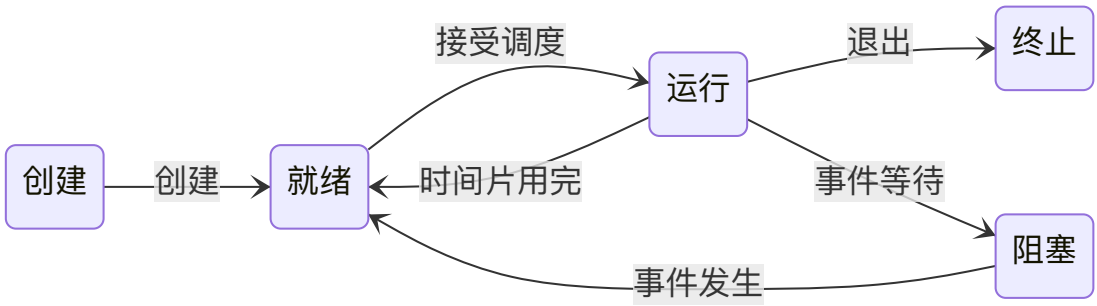
  > 进程是进程实体的运行过程，是系统进行资源分配和调度的一个独立单位。（体现并发性、共享性）
  >
  > 线程是一种轻量级进程，是一个基本的CPU执行单元，也是程序执行流的最小单元。

- 作业调度流程及功能

调度类型	发生时机	功能
高级调度（作业调度）	作业提交时	从后备队列选作业装入内存，创建进程
中级调度（内存调度）	内存紧张时	挂起/激活进程（内存↔外存交换）
低级调度（进程调度）	进程就绪时	从就绪队列选进程分配CPU

进程/线程

1. 基本状态转换
    1. 进程状态：运行态、就绪态、阻塞态、创建态、终止态



> 就绪态和阻塞态区别：就绪态仅缺少CPU，获得CPU即可运行；阻塞态指进程需要其他资源（CPU除外）或等待其他事件。

    2. 进程状态：运行态、就绪态、阻塞态

	调度性	并发性	拥有资源	系统开销
进程	资源分配的基本单位	进程间并发，开销大	拥有资源的基本单位。独立地址空间、文件、I/O设备等	创建/切换开销大（需切换页表等）
线程	CPU调度的基本单位（独立运行）	线程间并发，轻量高效	不拥有系统资源。共享进程资源，仅私有栈/寄存器	创建/切换开销小（共享内存）

2. **进程切换与线程切换的区别**

   操作系统需要完成的功能

   1. 挂起一个进程，将CPU上下文保存到PCB
   2. 将进程PCB移入相应队列（线程不需要）
   3. 选择另一个进程执行，更新PCB
   4. 恢复新进程的CPU上下文
   5. 跳转到新进程PCB中的程序计数器所指向位置执行（线程不需要）

3. **进程同步两种思路方法**

   1. 生产者-消费者：

      - 生产者-计算者-消费者，每一个`-`代表一对生产者-消费者关系

```
sem mutex = 1, empty = n, full = 0;
producer() {
  while(1) {
    生产一个产品
    P(empty);
    P(mutex);
    将产品放入缓冲区
    V(mutex);
    V(full);
  }
}
consumer() {
  while(1) {
    P(full);
    P(mutex);
    将产品放入缓冲区
    V(mutex);
    V(empty);
    消费产品
  }
}
```

      - 多生产者-多消费者

```
sem plate = 1, apple = 0, orange = 0;
dad() {
  while(1) {
    准备一个苹果
    P(plate);
    把苹果放入盘子
    V(apple);
  }
}
mom() {
  while(1) {
    准备一个橘子
```

```
    P(plate);
    把橘子放入盘子
    V(orange);
  }
}
son() {
  while(1) {
    P(orange);
    从盘子取出橘子
    V(plate);
    吃掉橘子
  }
}
daughter() {
  while(1) {
    P(apple);
    从盘子取出苹果
    V(plate);
    吃掉苹果
  }
}
```

2. 读者-写者

```
int cnt = 0;
sem mutex = 1, rw = 1, w = 1;
writer() {
  while(1) {
    P(w);
    P(rw);
    写文件
    V(rw);
    V(w);
  }
}
reader() {
  while(1) {
    P(w);
    P(mutex);
    if (cnt == 0)
      P(rw);
    cnt++;
    V(mutex);
    V(w);
    读文件
    P(mutex);
    cnt--;
    if (cnt == 0)
      V(rw);
    V(mutex);
  }
}
```

3. 信号量分类

    1. 整型信号量

```
wait(S) {
  while (S <= 0);
  S = S - 1;
}
signal(S) {
  S = S + 1;
}
```

`wait()` 进行一次P操作，表示请求一个资源

`signal()` 进行一次V操作，表示释放一个资源

    2. 记录型信号量

```
typedef struct {
  int value;
  struct process *L;
} semaphore;
void wait(semaphore *S){
  S->value--;
  if (S->value < 0) {
    add this process to S->L
    block(S->L);
  }
}
void wait(semaphore *S){
  S->value++;
  if (S->value <= 0) {
    remove s process from S->L
    wakeup(S->L);
  }
}
```

4. **调度算法**

    1. 指标

        - 平均周转时间

$$\text{平均周转时间} = \frac{(\text{作业1周转时间} + \cdots + \text{作业}n\text{周转时间})}{n} \tag{1}$$

        - 带权周转时间

$$\text{带权周转时间} = \frac{\text{作业周转时间}}{\text{作业实际运行时间}} \tag{2}$$

    2. 不同算法（四种）

        - **先来先服务**

            - 每次从后备作业队列中选择最先进入队列的一个或几个作业。

            - 长作业有利，对短作业不利
        - **短作业优先**
            - 每次从后备作业队列中选择估计运行时间最短的一个或几个作业。
            - 对长作业不利，会产生饥饿现象（进程长期不被调度）。
        - **优先级调度**
            - 每次从后备作业队列中选择优先级最高的一个或几个作业。
            - 静态优先级、动态优先级
        - **时间片轮转**
            - 每隔一定时间产生一次中断，将CPU分配给当前首位进程，执行一个时间片。若完不成则被释放出队列并重新入队。
    3. 能给出调度顺序，计算平均周转时间、带权平均周转时间
        - 画甘特图
5. 死锁
    1. 必要条件
        1. 互斥条件
        2. 不可剥夺条件
        3. 请求并保持条件
        4. 循环等待条件
    2. 几种方法（只需破坏必要条件之一即可）
        - 破坏循环等待条件
        - 破坏请求并保持条件
        - 破坏互斥条件
        - 破坏不可剥夺条件
    3. 题目：资源分配（详看作业题）
    4. 银行家算法
    5. 安全状态（安全性算法）

内存

1. 编译和链接
    1. 含义
    2. 解决什么问题

阶段	含义	核心任务
编译	将**高级语言源代码**（如 C/C++）翻译成**机器可识别的目标文件**（`.o` 或 `.obj`）的过程。	语法检查、代码优化、生成机器码
链接	将**多个目标文件**和**库文件**合并为单一可执行文件（如 `.exe` 或 `.out`）的过程。	符号解析、地址重定位、合并代码
装入	将**可执行程序**从磁盘加载到内存中，并为其分配物理地址空间的过程。它是程序执行的必经步骤，发生在链接之后、运行之前。	将程序中的**逻辑地址**（相对地址）转换为内存中的**物理地址**

3. 静态链接和动态链接的优缺点

类型	优点	缺点
静态链接	执行快（代码完整）；依赖少	浪费磁盘/内存（重复代码）；更新需重编译
动态链接	节省内存（共享库）；易更新	首次执行慢（加载库）；依赖环境（库版本）

2. 可重定位（静态重定位）、内存保护
    - 在CPU中设置一对上、下限寄存器，存放用户进程在主存中的下限和上限地址，每当CPU要访问一个地址时分别比较上下限来判断是否越界。
    - 采用重定位寄存器和界地址寄存器进行越界检查。
3. 内存分配方法

    单一连续分配 => 固定分区分配 => 动态分区分配
    - 连续分配：系统区、用户区
    - 固定分区分配：事先划分好分区，选择适合大小的内存分区放入作业。
    - 动态分区分配：
        - 首次适应法：放入分区时查找第一个满足的分区
        - 临近适应法：从上次查找结束的位置开始循环查找第一个满足的分区
        - 最佳适应法：顺序查找第一个满足大小的最小空闲分区
        - 最坏适应法：顺序查找第一个满足大小的最大空闲分区，在其中划分满足分区的大小
4. **分页系统**
    1. 逻辑地址结构

$31\cdots12$	$11\cdots0$
页号P	页内偏移量W

    2. 页号、页表、页帧、页面大小

3. 管理方式（地址转换）
    1. 逻辑地址 => 物理地址

$$
\begin{aligned}
\text{页号} &= \text{页表始址} + \text{页号}_{\text{逻辑地址}} \\
\text{块号} &= f_{\text{页表}}(\text{页号}) = \text{页表始址} + \text{页号} \times \text{页表项长度} \\
\text{物理地址}_{high} &= \text{块号} \\
\text{物理地址}_{low} &= \text{页内偏移量}
\end{aligned} \tag{3}
$$

    2. 虚拟存储器、TLB
        - TLB访问方案（并行、串行）
        - 虚拟存储器中**页面置换算法**
            - 先进先出
                - 淘汰最早进入内存的页面
            - 最佳置换
                - 淘汰永不使用或最长时间内不再访问的页面
            - 时钟算法
            - LRU
                - 淘汰最近最久未使用的页面
                - **计数器法**：
                    - 为每个页表项维护计数器，访问页时更新为当前时间戳。
                    - 置换时选择**时间戳最小**（最久未使用）的页面。
                - **栈算法**：
                    - 用双向链表维护页面访问顺序：
                        - 访问某页时，将其移到链表**头部**（MRU端）。
                        - 置换时选择链表**尾部**（LRU端）的页面。
            - 计算缺页中断次数、页面置换次数

I/O设备

1. 控制方式（轮询、中断、DMA、通道）
2. I/O设备与程序独立性（设备独立性软件）
3. 缓冲
    1. 从设备将一块数据输入缓冲区的时间为T，操作系统将缓冲区中数据传送到工作区的时间为M，CPU对数据进行处理的时间为C

2. 单缓冲区处理每块数据的平均时间为$Max(C,T)+M$
3. 双缓冲区处理每块数据的平均时间为$Max(C+M,T)$

4. **磁盘**
    1. 访问时间、访问动作
        - 寻道 => 寻道时间
        - 找扇区 => 旋转延迟
        - 读写数据 => 传输时间

部分	计算方式	公式
寻道时间	磁头移动到目标磁道的时间	$T_s=nm+s$，n条磁道，启动磁头臂时间s，每跨越一条磁道需要m
旋转延迟	磁盘旋转到目标扇区的时间	$T_r=\frac{1}{2r}$，转速为r
传输时间	读写数据的时间	$T_t=\frac{b}{rN}$，每次读写字节数b，此盘转速r，N为一个磁道上的字节数
总时间	寻道时间+旋转延迟+传输时间	$T_s+T_r+T_s$

    2. 调度算法
        1. 先来先服务
            - 根据进程请求访问磁盘的先后顺序进行处理
        2. 最短寻道时间/距离
            - 每次选择调度里当前磁头最近的磁道，使每次寻道时间最短
        3. 扫描调度
            - 在最短寻道时间算法基础上规定：只有磁头移动到最外侧磁道时才能向内移动，移动到最内侧磁道时才能向外移动
        4. 循环扫描
            - 在扫描调度算法基础上规定：磁头单向移动来提供服务，返回时直接快速移动至起始端而不服务任何请求
        5. 访问队列、访问磁道、移动距离

文件系统

1. 文件系统结构
    1. 分配表FAT
    2. **索引节点**

        - 多级节点：十三个地址项（十个直接地址块，一个一级间址，一个二级间址，一个三级间址）
        - 计算文件占用空间、访问数据访问盘块（索引块分配问题）
2. 文件打开、关闭、删除过程

> **打开**：
>
> 1. 检索目录，找到文件目录中的文件。将其从磁盘inode复制到活动inode中。
> 2. 将参数mode所给出的打开方式与活动inode中在创建文件时所记录的文件访问权限相比较，若合法则本次打开操作成功。
> 3. 打开合法时为文件分配用户打开文件表表项和系统打开文件表表项，并为后者设置初值，通过指针建立表项与inode之间的联系，再将文件描述符fd返回给调用者。
>
> **关闭**：
>
> 1. 根据 fd找到对应的 用户打开文件表项，将其标记为可用。减少关联的系统打开文件表项的引用计数。
> 2. 若系统文件表项的引用计数归零则刷新缓冲区（将内核缓存中未写入磁盘的数据同步，并释放文件表项）。
> 3. 减少活动 inode的引用计数，若 inode 引用计数归零且无其他关联（如硬链接）：
>   - 标记 inode 为可释放（但暂不删除磁盘数据）。
>   - 将内存中的 inode 移出活动表（可被后续重用）。
>   - 成功返回 0，失败返回错误码。
>
> **删除**：
>
> 1. 验证用户是否有删除权限（目录写权限 + 文件权限）。若文件正在被其他进程打开，则仅删除目录项，实际数据待所有进程关闭后释放。
> 2. 从父目录中移除文件名对应的目录项，断开文件名与 inode 的链接。
> 3. 减少 inode 的硬链接计数。
>   - 若硬链接计数归零 **且** 无进程打开该文件：
>   - 释放 inode 和数据块占用的磁盘空间。
>   - 标记 inode 为空闲（可重用）。
>   - 若文件仍被打开：
>   - 延迟释放磁盘空间，直到最后一个进程关闭文件（内存 inode 暂存删除标记）。

系统调用

1. 系统调用和库函数调用的区别

> 库函数是语言或应用程序的一部分，可以运行在用户空间中。
>
> 系统调用是操作系统的一部分，运行在内核空间中，并且很多库函数会使用系统调用来实现其功能。
>
> 未使用系统调用的库函数的执行效率比使用系统调用的库函数高——因为使用系统调用时，需要上下文的切换和系统状态的切换（用户态切换到内核态及切回用户态的开销）

2. 系统调用执行过程

> 1. 将系统调用号和所需参数压入堆栈，调用实际的调用指令，先执行陷入指令，将CPU状态从用户态切换至内核态，并保护中断现场。
> 2. 分析系统调用类型，转入相应系统调用处理子程序。
> 3. 子程序执行结束后，恢复中断现场，并返回被中断进程或新进程继续运行。（回到用户态）

一些操作系统数据结构

PCB（进程控制块）

1. 进程描述信息
2. 进程控制和管理信息
3. 资源分配清单
4. 处理机相关信息

进程描述信息	进程控制和管理信息	资源分配清单	处理机相关信息
进程标识符	进程当前状态	代码段指针	通用寄存器值
用户标识符	进程优先级	数据段指针	地址寄存器值
	代码运行入口地址	堆栈段指针	控制寄存器值
	程序的外存地址	文件描述符	标志寄存器值
	进入内存时间	键盘	状态字
	CPU占用时间	鼠标	
	信号量使用		

FCB（文件控制块）

FCB
文件名
类型
文件权限（读、写）
文件大小
文件数据块指针