2024 考研 408

操作系统

复习笔记

【考查目标】

1. 掌握操作系统的基本概念、方法和原理，了解操作系统的结构、功能和服务，理解操作系统所采用的的策略、算法和机制。

2. 能够从计算机系统的角度理解并描述应用程序、操作系统内核和计算机硬件协作完成任务的过程。

3. 能够运用操作系统原理，分析并解决计算机系统中与操作系统相关的问题。

2023 年 10 月 28 日

目录

1 操作系统概述

1.1 操作系统的基本概念

1.2 操作系统的发展历程

> 多道程序系统：
> 1. 资源利用率高，各种资源得到充分利用；
> 2. 系统吐量大，CPU 和其他资源保持“忙碌”状态；
> 3. 用户响应的时间较长，不提供人机交互能力。

1.3 程序运行环境

1. CPU 运行模式：内核模式、用户模式
2. 中断和异常的处理
3. 系统调用
4. 程序的链接与装入
5. 程序运行时内存映像与地址空间

1.4 操作系统结构

分层，模块化，宏内核，微内核，外核

1.5 操作系统引导

1.6 虚拟机

2 进程管理

2.1 进程与线程

1. 进程与线程的基本概念 2. 进程/线程的状态与转换 3. 线程的实现：内核支持的线程，线程库支持的线程 4. 进程与线程的组织与控制 5. 进程间通信：共享内存，消息传递，管道

2.2 CPU 调度与上下文切换

1. 调度的基本概念 2. 调度的目标

> 1. 周转时间。指从作业提交到作业完成所经历的时间：周转时间 = 作业完成时间-作业提交时间。
> 2. 响应时间。指从用户提交请求到系统首次产生响应所用的时间。

3. 调度的实现：调度器/调度程序 (scheduler)，调度的时机与调度方式（抢占式/非抢占式），闲逛进程，内核级线程与用户级线程调度

4. 典型调度算法：先来先服务调度算法；短作业（短进程、短线程）优先调度算法；时间片轮转调度算法；优先级调度算法；高响应比优先调度算法；多级队列调度算法，多级反馈队列调度算法

5. 上下文及其切换机制

2.3 同步与互斥

1. 同步与互斥的基本概念 2. 基本的实现方法：软件方法；硬件方法 3. 锁 4. 信号量 5. 条件变量

> 1. 同步信号量初值为 0，互斥信号量初值为可用资源数 n。
> 2. 条件变量作用类似于信号量，都用于实现进程同步。
> 在同一时刻，管程中只能有一个进程在执行，wait() 操作会使当前进程阻塞。

6. 经典同步问题：生产者-消费者问题；读者-写者问题；哲学家进餐问题

1）生产者-消费者问题

```
semaphore mutex = 1; /* 互斥访问缓冲区 */
semaphore empty = n; /* 缓冲区空的个数 */
semaphore full  = 0; /* 缓冲区满的个数 */
```

```
void Tproduce() { /* 生产者进程 */
  while (1) {
    P(&empty);
    P(&mutex);
    /* 放入缓冲区 */
    V(&mutex);
    V(&full);
  }
}
```

```
void Tconsume() { /* 消费者进程 */
  while (1) {
    P(&full);
    P(&mutex);
    /* 从缓冲区拿出 */
    V(&mutex);
    V(&empty);
  }
}
```

2）读者-写者问题

```
int count = 0; /* 当前读者数量 */
semaphore mutex = 1; /* 互斥访问 count 变量 */
semaphore rw    = 1; /* 互斥访问文件 */
semaphore w     = 1; /* 写优先 */
```

```
void Tread() {
  while (1) {
    P(&w);
    P(&mutex);
    //第一个读线程准备读，阻止写线程写
    if (count == 0) P(&rw);
    count++;
    V(&mutex);
    V(&w);
    /* 读文件 */
    P(&mutex);
    count--;
    //最后一个读线程读完，允许写线程写
    if (count == 0) V(&rw);
    V(&mutex);
  }
}
```

```
void Twrite() {
  while (1) {
    P(&w);
    P(&rw);
    /* 写文件 */
    V(&rw);
    V(&w);
  }
}
```

3）哲学家进餐问题

```
semaphore chops[n] = {1,1,...,1};
semaphore mutex = 1; /* 每一时刻只能一位哲学家拿筷子 */
void Tphilosopher(int i) {
  while (1) {
    P(&mutex); /* 同时拿左边和右边的筷子 */
    P(&chops[i]);
    P(&chops[(i + 1) % n]);
    V(&mutex);
    /* 进餐 */
    V(&chops[i]);
    V(&chops[(i + 1) % n]);
  }
}
```

在读者写者问题的基础上增加如下条件：若写者写完的内容还没有被任何一个读者读取，则新的写进程不能进行写操作，直到有至少一个读进程进行了读操作。

```
int rtime = 0;       // 文件读的次数
int rcount = 0;      // 读者数量
semaphore wlock = 1;         // 写者锁
semaphore rwlock = 1;        // 读写锁
semaphore wwlock = 1;        // 写者写完至少读一次
semaphore mutex_rcount = 1; // 互斥访问 rcount
semaphore mutex_rtime = 1;  // 互斥访问 rtime
```

```
void Twrite() {
  while (1) {
    P(&wwlock);
    P(&rwlock);
    P(&wlock);
    // write
    P(&mutex_rtime);
    rtime = 0; // 写者写完，读次数置0
    V(&mutex_rtime);
    V(&wlock);
    V(&rwlock);
  }
}
```

```
void Tread() {
  while (1) {
    P(&rwlock);
    P(&mutex_rcount);
    rcount++;
    // 第一个读者
    if (rcount == 1) P(&wlock);
    V(&mutex_rcount);
    V(&rwlock);
    // read
    P(&mutex_rtime);
    rtime++;
    // 写者写完第一次读
    if (rtime == 1) V(&wwlock);
    V(&mutex_rtime);
    P(&mutex_rcount);
    rcount--;
    // 最后一个读者
    if (rcount == 0) V(&wlock);
    V(&mutex_rcount);
  }
}
```

2.4 死锁

1. 死锁的基本概念 2. 死锁预防 3. 死锁避免 4. 死锁检测和解除

表 1: 死锁处理策略比较的总结

	资源分配策略	各种可能模式	主要优点	主要缺点
死锁预防	保守， 宁可资源闲置	一次请求所有资源， 资源剥夺， 资源按序分配	适用于突发式处理的进程， 不必进行剥夺	效率低，进程初始化时间长， 剥夺次数过多， 不便灵活申请新资源
死锁避免	在运行时判断 是否可能死锁	寻找可能的 安全允许顺序	不必进行剥夺	必须知道将来的资源需求， 进程不能被长时间阻塞
死锁检测	宽松，只要允许 就分配资源	定期检查死锁 是否已经发生	不延长进程初始化时间， 允许对死锁进行现场处理	通过剥夺解除死锁，造成损失

1. 银行家算法作为一种死锁避免算法，不能判断系统是否处于死锁状态。
2. 死锁检测，通过简化资源分配图可以检测系统是否处于死锁状态。

3 内存管理

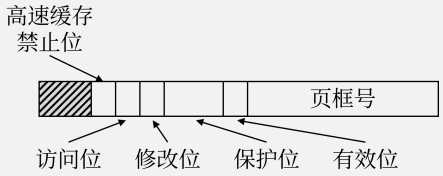
3.1 内存管理基础

1. 内存管理的基本概念：逻辑地址空间与物理地址空间，地址变换，内存共享，内存保护，内存分配与回收
2. 连续分配管理方式
3. 页式管理
4. 段式管理
5. 段页式管理

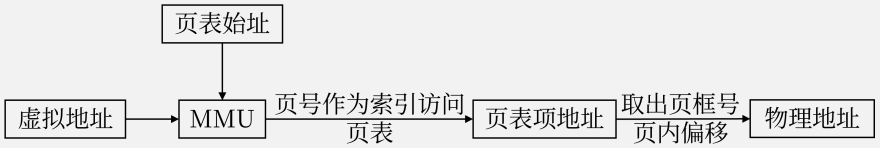
3.2 虚拟内存管理

1. 虚拟内存基本概念
2. 请求页式管理
3. 页框分配
4. 页置换算法
5. 内存映射文件 (Memory-Mapped Files)
6. 虚拟存储器性能的影响因素及改进方式

1. 页表项结构：

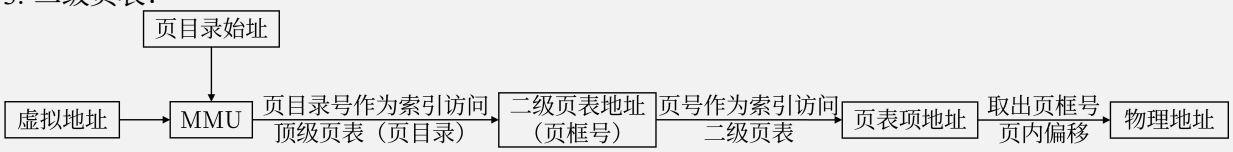


2. 一级页表：



页表项地址 = 页表始址 + 页号 × 页表项长度

3. 二级页表：



页目录项地址 = 页目录始址 + 页目录号 × 页目录项长度

页表项地址 = 二级页表始址（页框号） × 页表长度 + 页号 × 页表项长度

4 文件管理

4.1 文件

1. 文件的基本概念 2. 文件元数据和索引节点 (inode)
3. 文件的操作：建立，删除，打开，关闭，读，写

> 1. open 系统调用：根据文件名搜索文件目录，将文件的 FCB（inode）从外存复制到内存打开文件表的一个表目中，将表目索引（Unix 文件描述符，Windows 文件句柄）返回用户。
> 2. read/write 系统调用：根据表目索引读写文件。

4. 文件的保护 5. 文件的逻辑结构 6. 文件的物理结构

> 1. 连续分配：文件目录项包含**起始地址**和**文件长度**。支持**顺序访问**和**直接访问**。只适用于长度固定的文件。
> 2. 链接分配：
> 1）隐式链接：文件目录项含有文件第一块和最后一块的指针，除最后一个磁盘块外，每个盘块都含有一个指向文件下一个盘块的指针。**只适合顺序访问**。
> 2）显示链接：**FAT（文件分配表）**：磁盘启动时读入内存，一个磁盘一张表，每个表项存放盘块号的下一盘块号的指针。支持**直接访问**，但需要占用较大的内存空间。
> FAT 不仅记录了文件各块之间的先后链接关系，同时还**标记了空闲的磁盘块**，操作系统也可以通过 FAT 对文件存储空间进行管理。
> 3. 索引分配：一个文件一个索引块，第 i 个条目指向文件的第 i 块。支持**直接访问**。

4.2 目录

1. 目录的基本概念 2. 树形目录 3. 目录的操作 4. 硬链接和软链接

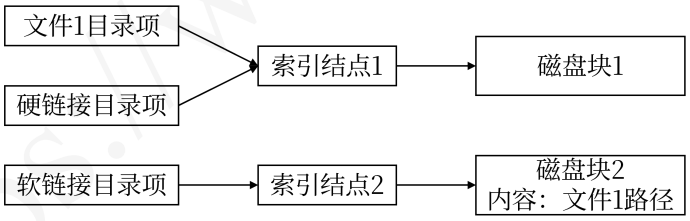


图 1: 文件软硬链接的区别

4.3 文件系统

1. 文件系统的全局结构 (layout)：文件系统在外存中的结构，文件系统在内存中的结构

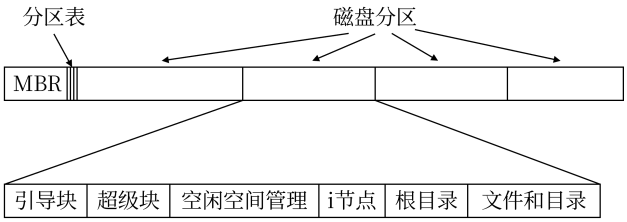


图 2: 文件系统布局

2. 外存空闲空间管理办法 3. 虚拟文件系统 4. 文件系统挂载 (mounting)

5 输入输出 (I/O) 管理

5.1 I/O 管理基础

1. 设备：设备的基本概念，设备的分类，I/O 接口，I/O 端口
2. I/O 控制方式：轮询方式，中断方式，DMA 方式
3. I/O 软件层次结构：中断处理程序，驱动程序，设备独立软件，用户层 I/O 软件

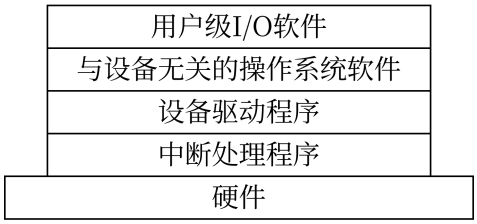


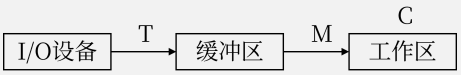
图 3: I/O 软件层次结构

4. 输入/输出应用程序接口：字符设备接口，块设备接口，网络设备接口，阻塞/非阻塞 I/O

5.2 设备独立软件

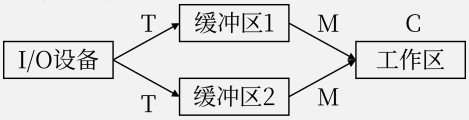
1. 缓冲区管理

1. 单缓冲:



设初始状态为工作区满，缓冲区空，单缓冲区处理每块数据用时 $\max(C,T)+M$.

2. 双缓冲:



设初始状态为工作区空，缓冲区 1 空，缓冲区 2 满，双缓冲区处理每块数据用时 $\max(C+M,T)$.

2. 设备分配与回收 3. 假脱机技术 (SPOOLing) 4. 设备驱动程序接口

5.3 外存管理

1. 磁盘：磁盘结构，格式化，分区，磁盘调度方法

1. 磁盘结构:
柱面号 = ⌊簇号/每个柱面的簇数⌋
磁头号 = ⌊(簇号%每个柱面的簇数)/每个磁道的簇数⌋
扇区号 = 扇区地址%每个磁道的扇区数
2. 磁盘格式化:
(1) **物理格式化**: 分成扇区，格式化后的容量比格式化前小。
(2) **逻辑格式化**: 设置一个引导块、空闲存储管理（空闲列表或位图）、根目录和一个空文件系统。还要将一个代码设置在分区表项中，表明在分区中使用的是哪个文件系统。

2. 固态硬盘：读写性能特性，磨损均衡