**操作系统作业**

20211120138 薛凡豪 网络空间安全

# 第一章

1.

设计操作系统的主要目标是提供一个方便、高效、可靠和安全的计算机系统环境，以便用户和应用程序可以方便地使用和管理硬件资源，同时保证系统的稳定性和安全性。具体来说，设计操作系统的目标包括以下两个方面：

(1.资源：操作系统需要管理计算机系统上的各种资源，如内存、CPU、输入输出设备等，以提高它们的利用率和效率，并保证它们的分配和释放是公平和合理的。

(2.用户：操作系统需要提供一个简单易用的用户界面，使用户能够方便地完成各种操作，如存储和管理文件、运行程序等。

2.

(1.进程管理：操作系统负责管理计算机系统中正在运行的进程，为它们分配所需的系统资源（如 CPU 时间、内存空间等），同时控制它们的执行顺序和优先级，以实现系统的高效稳定。

(2.内存管理：操作系统需要管理计算机系统中的内存，并进行分配和回收，以满足不同应用程序的内存需求，保证系统的运行稳定。

(3.文件管理：操作系统需要管理存储设备上的文件和目录，提供统一的文件系统接口，使得用户能够方便地存储、访问和管理文件。

(4.设备管理：操作系统需要管理计算机系统中各个设备，控制它们的访问和操作，并为用户和应用程序提供各种设备驱动程序和硬件接口。

(5.安全性：操作系统需要提供安全保护机制，以保护用户的数据和系统的安全，包括访问控制、密码保护、病毒检查等。

3.

多道批处理系统形成和发展的主要动力是提高计算机资源利用率和生产效率。

在早期计算机系统中，每次只能执行一个作业，而且需要由程序员手工为计算机编写程序和输入数据，任务完成后还需要手工更换程序和数据，这使得计算机的使用和管理非常低效。

为了提高计算机的使用效率和生产率，多道批处理系统应运而生。多道批处理系统可以将多个作业同时装载到内存中，在共享硬件资源的情况下，通过时间片轮转等调度算法，让多个作业交替执行，从而大大提高计算机的资源利用率和生产效率，减少了程序员手工操作的重复工作，加速了计算机的数据处理和产生效果。

5.

推动分时操作系统形成和发展的主要动力是对计算机系统用户的需求。早期的计算机系统是单用户的，只能由一个用户独占地使用，其他用户需要通过批处理方式提交作业并等待计算机处理完成后才能得到结果。这样的方式不仅效率低下，而且限制了计算机系统的使用范围。

为了解决这个问题，分时操作系统应运而生，它可以让多个用户在同一计算机系统上进行交互式操作，并共享计算机资源，从而大大提高了计算机系统的使用效率和灵活性。

除此之外，随着计算机技术的不断发展和计算机应用的普及，对计算机系统的使用需求也越来越多样化。分时操作系统可以满足不同用户的需求，提供更为灵活的计算机资源调度和管理方式，进一步促进了计算机技术的发展和应用。

7.

1. 整体结构

2. 层次式结构

3. 微内核式结构

UNIX的结构特点：

1. 短小精悍的内核，强大的系统功能
2. 良好的可移植性
3. 树形结构的文件系统
4. 统一看待外设和文件
5. I/O 重定向和管道
6. 良好的用户接口
7. 开放式的操作系统
8. 强有力的软件开发环境
9. 丰富的实用程序

UNIX 系统可分成三层。最内层是 UNIX 系统的核心，它提供进程管理、存储管理、 设备管理和文件管理等各项功能。中间一层是 Shell 和实用程序层，它在内核和用户程序 之间起到中间纽带作用，并提供各种实用程序和库函数服务。最外层是应用层，它由用户 程序构成。UNIX 系统建立在具体机器硬件的基础之上，构造了分层的虚似机，为用户提 供独立于具体机器硬件的各种服务。

从系统构成的角度我们可将 UNIX 系统大致看成三级：用户级、核心级、硬件级。用 户级包括 Shell、各种实用程序、标准库函数等。核心级即 UNIX 内核，它主要完成进程、 存储、文件和设备的管理工作。

# 第二章

第二章、处理机管理

1.判断题：

1-5:××√√√ 6:×

2.选择题：

1-5:CACDA 6:D

3. 填空题：

（1）cpu

（2）进程 线程

（3）（4）（5）（6）涉及到UNIX

4.多道程序设计：同时把多个作业放入主存，并且允许交替执行和共享资源

优点：提高系统吞吐量、提高资源和设备的利用率、提高cpu利用率

5.程序并发执行失去封闭性的主要原因是：并发执行时会共享资源，从而失去其封闭性

6.**进程**：

进程是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配的基本单位，是操作系统结构的基础。

控制进程运行需要保护：即PCB的内容，包括断点地址、程序状态字、寄存器、堆栈等等。

7.作业和进程：

（1）进程是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配的基本单位。作业是用户向计算机提交的任务实体

（2）一个作业可以由一个或多个进程组成，至少有一个进程组成

（3）作业一般只在批处理系统出现，进程几乎在所有的多道程序系统中都存在

8.进程和程序：

（1）进程是程序的一次执行过程，是动态的，而程序是静态的。

（2）进程是一个能独立运行，独立分配资源的基本单位

（3）多个进程可以同时在内存中并发执行，而程序的并发执行具有不可再现性。

（4）程序和进程不具有一一对应性，同一个程序执行多次可以对应多个进程，一个程序也可以由多个进程组成

9.

（1）创建进程：系统需要为新进程创建PCB、分配资源

（2）撤销进程：释放进程占用的空间和资源，并且删除进程对应的PCB

（3）A->B：进程切换，需要将B的上下文信息载入，包括B的PCB以及寄存器状态等

11.处理机调度：

共有三级调度：

1. 高级调度：又叫作业调度，存在于批处理系统，按照某种算法将作业从外存调入内存
2. 中级调度：让暂时不运行的进程调出到外存等待，当需要运行并且内存空间充足时重新调入，可以提高内存利用率和系统吞吐量。
3. 低级调度：又叫进程调度，按照算法从就绪队列选择进程使其获得CPU

13.进程阻塞和唤醒：

**三种操作：**

（1）完成某种操作：当进程启动I/O操作时进入阻塞，中断处理唤醒

（2）请求系统服务：

当执行的进程请求系统提供服务无法满足时，进入阻塞；释放服务的进程可以唤醒阻塞的进程

1. 无工作：当进程没有工作时，进入阻塞，有进程发出请求时唤醒。

**唤醒原语**：实现阻塞态->就绪态

步骤（1）：找的阻塞进程的PCB

步骤（2）：将PCB插入相应事件的等待队列

步骤（3）：将PCB插入就绪队列，等待调度

19.采用多级反馈队列更有利于资源利用率

原因：以I/O为主的进程，不可以使用时间片调度算法，其会使CPU利用率降低。

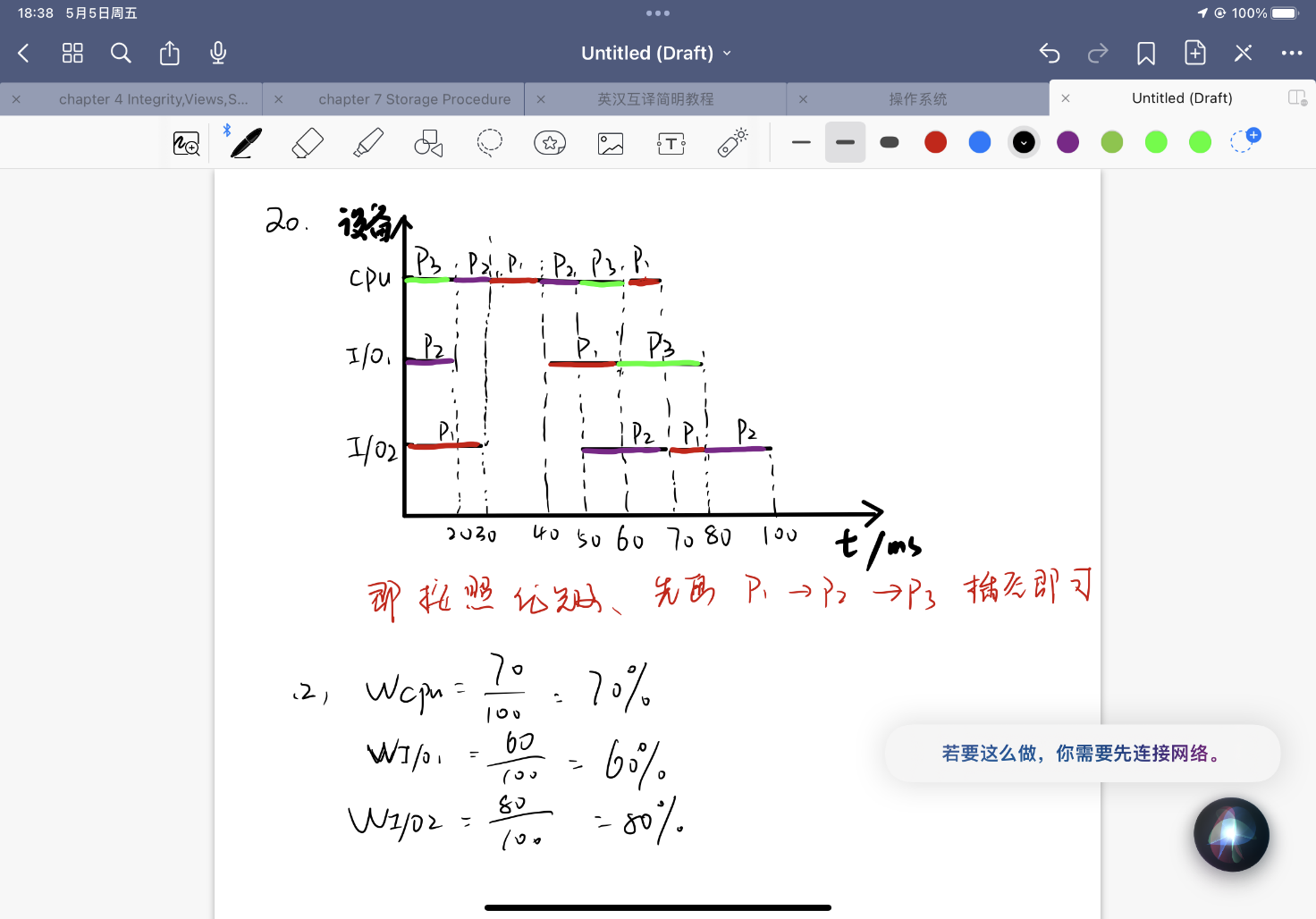
以计算为主的进程，使用优先调度算法会导致周转时间变长

而先来先服务算法无法保证周转时间，一般来讲周转时间是较长的，资源利用率也得不到保障

而多级反馈队列照顾了紧急的进程同时兼顾了短进程和长进程，是比较好的调度算法。

20.

见下图：



**22.**

不对。

分析过程：

执行顺序：1、3、4、2

T1=2.0h, T3=8+2+0.1-9=1.1h T4=10.1+0.2-9:50=28min=0.47h

T2=10:18+0.5h-8:50=1h58min=1.97h(单位换算)

T=(2+1.1+0.47+1.97)/4=1.385h

W=(2/2+1.1/0.1+0.47/0.2+1.97/0.5)/4=4.5725h

**23.**

**（1）先来先服务：顺序：0.1.2.3.4**

T0=20

T1=20+15=35

T2=35+35=70

T3=70+25=95

T4=95+40=135

**周转时间T=**（20+35+70+95+135）/5 = 71

**带权周转时间W=**（20/20+35/15+70/35+95/25+135/40）/5 = 2.50

**（2）最短周期优先算法：顺序：1.0.3.2.4**

T1=15

T0=15+20=35

T3=25+35=60

T2=35+60=95

T4=40+95=135

**周转时间T**=（15+35+60+95+135）/5 = 68

**带权周转时间W**=（15/15+35/20+60/25+95/35+135/40）/5 = 2.25

**（4）最高优先级算法：顺序：1.3.0.2.4.**

T1=15

T3=25+15=40

T0=20+40=60

T2=35+60=95

T4=40+95=135

**周转时间T** = （15+40+60+95+135）/5=69

**带权周转时间W**= （15/15+40/25+60/20+95/35+135/40）/5=2.92

24.

**（1）最短周期优先：**  
顺序：1.3.4.2

T1=2h T3=4-3+0.1=1.1h T4=4.3-3.5=0.8h T2=4.8-2.5=2.3

**周转时间T** = （2+1.1+0.8+2.3）/4=1.55h

**带权周转时间W**=（2/2+1.1/0.1+0.8/0.2+2.3/0.5）/4=5.15h

**（2）先来先服务：**  
顺序：1.2.3.4

T1=2h T2=4.5-2.5=2h T3=4.6-3=1.6h T4=4.8-3.5=1.3h

**周转时间T**=（2+2+1.6+1.3）/4=1.725h

**带权周转时间W**=（2/2+2/0.5+1.6/0.1+1.3/0.2）/4=6.875h

# 第三章

1.

（1）错误，CPU正在执行一个 Down 操作时，如果有中断到来，那么这个中断处理进程可以被放在等待队列上。当 Down 操作完成后，CPU再回到原进程或者执行等待队列中的进程。如果中断处理进程可以抢占CPU执行Down操作，则会导致死锁，并且由于Down 操作是临界区的代码，因此此时不会发生中断抢占的情况，因为中断无法抢占进程的临界区。

（2）正确，临界区是共享资源的一段程序代码，同一时间只能被一个进程访问。在这个临界区内，进程需要互斥，以避免产生数据异常和硬件资源冲突的情况。

（3）错误，多个进程可以同时进入临界区，只要它们不是同时访问共享资源。

（4）错误，进程 A 与进程 C 可以同时运行，如果它们不共享同一个资源。只有当它们之间有一个共享的资源时，才需要互斥。

（5）错误，系统处于不安全状态只是一个可能导致死锁的原因，但不一定导致死锁的发生。只有在所有进程都无法继续执行时，才会出现死锁问题。

（6）错误，系统不是死锁状态并不意味着它是安全状态。

2.

(1) D、低级进程通信原语。Down、Up 操作是进程或线程通过操作信号量来进行互斥访问临界资源的一种低级进程通信原语。

(2) A、命令解释程序。命令解释程序（shell）是在用户态执行的，用于读取用户输入的命令并将其解释和执行。

(3)B、 M、N 分别为 1、0。信号量初值为 3，当前值为 1，表示该资源目前只有 1 个可用的资源，有 2 个进程在使用该资源。但是没有等待的。

(4) C、每个进程最多需要 3 台打印机，所以每个进程占用 个资源时， 有2k+1<=8，则K<=3.5,所以k=4时会发生死锁

3.

（1）[-9,1]

（2）竞争条件和死锁

（3）消息队列、读指针和写指针

（4）间接通信

（5）环路图

（6）死锁避免、死锁预防、死锁预防

4.

Bernstein 条件是判断程序能否并发执行的条件。当P1的读集和P2的写集、P1的写集和P2的读集、P1的写集和P2的写集交集都为空时，满足条件

可能会失去封闭性和可再现性

5.

信号量是一种特殊的变量，用于控制对共享资源的访问，是一种用于进程间同步和互斥的方法。

信号量定义了一个整型变量和两个原语操作：P 和 V。P用于获取资源（阻塞进程），每次P操作将信号量减1，表示已经使用了一部分资源，同时检查资源数量是否大于或等于0；如果不是，则阻塞等待。V用于释放资源，每次V操作将信号量加1，表示已经释放了一部分资源，同时检查是否有其他进程在等待资源，如果有，则唤醒其中一个进程。

通过这两个操作，可以实现进程间的通信。例如，通过信号量可实现进程对缓存区和消息队列的读写操作。在缓存区中，进程可以利用信号量实现生产者-消费者的同步操作；在消息队列中，进程可以利用信号量实现进程间消息的发送和接收。在进程间通信时，每个进程需要先通过P操作来获得互斥访问的权限，然后再进行临界资源的访问。访问完成后，该进程需要通过V操作释放互斥访问权限，以便其他进程也能够访问该资源。

6.

假设有两个进程P1和P2，在它们之间有一个临界区需要进行互斥访问。如果Down操作和Up操作没有被实现为原语，那么在相同时间内可能会有两个进程同时试图进入临界区，这会导致数据的不一致和出错。

例如，如果P1执行的Down操作被中断，导致未成功获取互斥访问的权限，而此时P2又执行了Down操作成功获得互斥访问权限，就会导致两个进程同时访问临界区。这时，就会发生数据不一致和出错的问题。因此，Down和Up操作必须被实现为原语，以确保临界区的互斥访问。

8.

同步是指多个进程或线程按照一定的顺序协同工作，保持有序性。也就是说，不同的进程或线程共同完成某个任务时，它们之间需要有一定的时间协调和信息交互，以保证整个任务能够按预期完成。同步是保持进程间顺序的方法。

而互斥是指多个进程或线程之间共享一些资源，但同时只有一个进程或线程能够访问这些资源。也就是说，如果两个进程同时访问一个共享资源，就会引起冲突和数据不一致的问题，需要通过一定的方法来解决这些问题。互斥是保护共享资源，防止并发访问的方法。

同步和互斥的联系在于，它们都是实现进程或线程之间协作的方法。如果进程或线程之间需要共享资源，那么就需要采用互斥的方法来避免冲突和数据不一致的问题。而在共享资源的过程中，进程或线程还需要按照一定的顺序协作，以完成某个任务，这时就需要采用同步的方法来保持进程或线程之间的有序性。

10.

管道和管程都是用于进程间通信的工具，但它们在实现上有一些不同之处。

管道是一种通信方式。它将一个进程的输出和另一个进程的输入直接相连，其中一个进程所输出的数据会作为另一个进程的输入数据。它可以实现在不同进程间进行数据传输，被广泛应用于 Shell 脚本等领域中的一些 I/O 操作。在管道中，数据流动的方向是单向的，而且管道只有一个读端口和一个写端口，不支持多个进程同时写入和读取。

管程是一种数据结构，它提供了一种同步机制，可以保护共享资源的访问，防止并发访问引起的错误。管程在进程间提供了操作共享数据的原语，使得进程可以互相通信并协同工作。管程中封装了访问共享资源的代码，使得共享资源的访问可以在管程维护的临界区中被互斥地进行，进程可以在不发生冲突的情况下访问共享资源。管程还提供了条件变量，可以让进程在满足特定条件时等待或唤醒。

总的来说，管道和管程都是进程间通信的机制，但它们适用于不同的场景。管道适合用来进行进程之间单向的数据传输，而管程适合用于实现对共享资源的管理和保护。

12.

使用三个信号量 empty、full 和 mutex 来实现缓冲区的管理和互斥访问。其中 empty 表示当前缓冲区中的空闲位置数量，full 表示当前缓冲区中已有的数据数量，mutex 用于实现对缓冲区的互斥访问。

#define N 8 // 缓冲区长度

#define M 3 // 计算进程数量

#define K 2 // 打印进程数量

Semaphore empty = N; // 缓冲区剩余空间

Semaphore full = 0; // 已经有的数据个数

Semaphore mutex = 1; // 互斥信号量

int buffer[N]; // 缓冲区

int in = 0, out = 0; // 缓冲区下标

void producer() {

int result; // 计算结果

while (true) {

result = calculate();

// 等待空余的缓冲区

Down(&empty);

// 获取缓冲区的互斥访问权

Down(&mutex);

// 将数据放入缓冲区

buffer[in] = result;

in = (in + 1) % N;

// 释放缓冲区的互斥访问权

Up(&mutex);

// 唤醒等待已经有数据的进程

Up(&full);

}

}

void consumer() {

int result; // 取出的数据

while (true) {

// 等待已经有数据的缓冲区

Down(&full);

// 获取缓冲区的互斥访问权

Down(&mutex);

// 从缓冲区取出数据

result = buffer[out];

out = (out + 1) % N;

// 释放缓冲区的互斥访问权

Up(&mutex);

// 唤醒等待空余缓冲区的进程

Up(&empty);

// 打印取出的数据

print(result);

}

}

20.

死锁产生的四个必要条件如下：

1. 互斥条件：至少有一个资源被进程排他地占用，不能被多个进程同时使用。

2. 不可抢占条件：资源不能被操作系统抢占，它只能在进程使用完之后由进程主动释放。

3. 占有且等待条件：进程已经占有了至少一个资源，并且在等待获取其他资源。

4. 循环等待条件：存在一个进程资源请求的循环链，链中每个进程已经占有了下一个进程所需要的资源。

为了避免死锁，操作系统可以采用以下方法：

1. 破除互斥条件：放宽互斥条件，允许多个进程访问同一个资源。

2. 破除不可抢占条件：操作系统在进程请求资源时可以将占用资源时间过长的进程抢占，释放资源。

3. 破除占有且等待条件：要求进程一开始就请求所有所需资源。或者允许进程释放已经占有的资源，并且在未获得所需资源之前不再获取其他资源。

4. 破除循环等待条件：对资源进行按序号分配，规定进程只能按照一定顺序申请资源，并且不允许占用多个资源。

21.

死锁指的是在多个进程中，每个进程持有某些资源，并且同时等待其他进程释放其所持有的资源。这种情况下，所有进程都无法向前推进，形成了一种僵局。死锁问题的出现，会导致系统资源浪费、运行效率下降，需要通过合理的资源分配和调度来避免死锁的产生。

饿死指的是在资源调度中，某些进程由于始终无法满足其资源需求而无法得到调度运行，这种情况下进程长期处于等待状态，无法被及时执行。在该情况下，某些进程在系统中因一直得不到资源而处于一种长期饥饿和无法推进的状态，这种现象称为饿死。饿死可能是由于进程优先级低导致的，也可能与系统中的资源分配算法不合理有关。避免饿死需要合理的资源分配算法，以及考虑进程优先级等因素。

23.

（1）需要满足：n(k-1)+1<=m

解方程得：k<=(m-1)/n+1,最后对k向下取整

（2）带入上式

N=3,m=2时，k<=4/3,故k<=1

N=3,m=3时，k<=5/3，故k<=1

N=3,m=4时，k<=2,故k<=2

26.

银行家算法需要使用如下数据结构：

1. 可利用资源数组（available）：记录当前系统中每种资源可用的数量。

2. 最大需求矩阵（max）：记录每个进程对每种资源的最大需求。

3. 已分配资源矩阵（allocation）：记录当前已经分配给每个进程的每种资源数量。

4. 需求矩阵（need）：记录每个进程仍需要的每种资源数量。

银行家算法的运行过程：

1. 当进程申请资源时，检查申请的资源量是否小于等于需要的最大资源量和系统现在可用的资源量，如果成立，分配资源给该进程；否则不分配，等待其他进程释放资源。

2. 当进程释放资源时，系统将释放的资源加回到 available 数组中。

3. 当进程申请资源导致系统不能满足时，它将等待直到有足够的资源可用，但是如果等待时间过长，可以通过通知其他进程释放资源来打破死锁。

银行家算法对于避免死锁是非常有效的，它可以检测到潜在的死锁情况，并且能够防止应用程序进入不安全的状态。

然而，在实际应用中，银行家算法也存在一些缺点。例如，它需要预先知道每个进程所需的最大资源量，这在一些情况下可能是难以确定的。此外，当资源分配存在过多限制时，银行家算法可能会导致一定的资源浪费和系统效率降低的问题。

28.

见下图:

