目录

[第一章 操作系统概论 1](#_Toc87390567)

[第二章 处理器管理 2](#_Toc87390568)

[第三章 同步、通信与死锁 5](#_Toc87390569)

[第四章 存储管理 10](#_Toc87390570)

### 第一章 操作系统概论

1、什么是操作系统？操作系统在计算机系统中的主要作用是什么？

1. 定义

操作系统是一组有效管理和控制计算机硬件和软件资源，合理组织计算机工作流程和为用户有效使用计算机提供良好运行环境的系统软件。

2) 作用

面向底层硬件：

* 管理、控制和调度计算机系统软、硬件资源；
* 对程序的执行进行控制和调度，提高系统效率和资源利用率。

面向上层应用和用户：

* 在裸机基础上形成更易理解和使用的“虚拟机”供应用程序使用；
* 为普通用户提供友善的运行环境，为开发者提供程序接口。

1. 现代操作系统具有哪些基本功能？请简述之。

基本功能：处理器管理、内存管理、设备管理、文件管理。叙述略。注意根据叙述情况打分，如果只有上面四点，没有叙述，不给A。

1. 若内存中有3道程序A、B、C，它们按A、B、C优先次序运行。各程序的计算轨迹为：

A：计算(20)、I/O(30)、计算(10)

B：计算(40)、I/O(20)、计算(10)

C：计算(10)、I/O(30)、计算(20)

如果三道程序都使用相同设备进行I/O(即程序用串行方式使用设备，调度开销忽略不计)。试分别画出单道和多道运行的时间关系图。两种情况下，CPU的平均利用率各为多少?

**答**：分别画出单道和多道运行的时间图

1. 单道运行时间关系图

0 20 40 50 60 80 100 120 140 160 180 190

I/O

CPU

时间

(ms)

A

A

A

B

B

B C

C

C

单道总运行时间为190ms。CPU利用率为(190-80)/190=57.9%

1. 单道运行时间关系图

I/O

CPU

时间

(ms)

A

A

A

B

C

0 20 40 50 60 80 100 120 140

B

B

C

C

B

多道总运行时间为140ms。CPU利用率为(140-30)/140=78.6%

4、叙述系统调用的概念和操作系统提供系统调用的原因。

系统调用也称程序接口，是程序级的接口，即用户程序可以利用系统提供的一组系统调用命令，来调用操作系统内核中的一个（或一组）过程来获得系统服务，完成自己所需要的功能。在系统调用时，处理器将由用户模式切换到内核模式。

操作系统向用户程序提供系统调用的原因是为了对系统进行“保护”。当用户程序需要系统服务，也就是要调用系统内核中的某些程序时，只能从规定的位置进入内核，这样才能保证系统的安全。

### 第二章 处理器管理

1、什么是内核模式和用户模式？处理器为什么要区分内核模式和用户模式两种运行模式？计算机是如何识别内核模式和用户模式的？

内核模式是操作系统程序的运行模式，在这种模式下，程序可以执行特权指令，对资源的访问权限不受限制；相对来说，用户模式下运行的程序只能执行非特权指令，其资源需求也将受到各种限制。

区分内核模式和用户模式的目的是为了确保系统的安全性和可靠性。

处理器的程序状态寄存器(PSW)中有一个模式位(mode bit)，用于记录当前正在处理器上执行的程序可执行于内核模式还是用户模式。

2、为什么要引入进程概念？它与程序有何联系和区别。

答：多道程序系统带来了更为复杂的环境，程序的执行表现出动态性、并发性等新型特点，程序已不足以刻画和反映系统中的情况,为了描述系统内部出现的情况，系统内部各作业的活动规律而引出的进程。

进程(Process)是具有独立功能的程序在某个数据集合上的一次运行活动，是操作系统中最基本的概念。

联系：进程是程序的一次执行过程，没有程序就没有进程。

区别：进程不仅仅是程序，还包含程序在执行过程中所需要的全部资源，如内存、CPU等系统资源，没有资源，程序就无法执行，因此进程是程序执行的载体。

* 进程是程序的执行，是动态概念；程序是一组指令的有序集合，是静态概念；
* 进程存在于内存，其存在是暂时的（有生命周期），而程序存在于外存，不具有生命周期；
* 进程的组成包含程序和数据，此外还有记录进程状态信息的PCB；
* 一个程序可能对应多个进程，一个进程可以包含多个程序，两者无一一对应关系。

\*批改说明：进程引入原因须涉及“多道程序系统带来的复杂性”，内容大致相符即可。

进程和程序的联系与区别包含其中2-3点即可，从宽改。

3、试从调度、并发性、拥有资源和系统开销等四个方面对传统进程和多线程进程进行比较。

调度：在引入线程的操作系统中，把线程作为CPU调度和分派的基本单位，而把进程作为资源分配和保护的独立单位。

并发性：在引入线程的操作系统中，一个进程可以包含多个线程。不仅进程之间可以并发执行，而且在一个进程中的多个线程之间也可以并发执行，因而使操作系统具有更好的并发性。

拥有资源：进程始终是拥有资源的一个独立单位，线程自己不拥有系统资源，但它可以访问其隶属进程的资源。

系统开销：在对进程和线程进行创建、撤销和切换等操作时，进程的开销远远大于线程的开销。

1. 系统有5个进程，其就绪时刻（指在该时刻已经在就绪队列中就绪）、运行时间如下表所示。若采用先来先服务、短作业优先、高响应比优先、时间片轮转调度算法（时间片=1），请分别填写表1~表4，分析在不同算法下各进程的开始时间、结束时间、周转时间、带权周转时间，并计算相关的平均周转时间和平均带权周转时间。

表1 采用先来先服务调度算法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 进程 | 就绪时刻 | 运行时间 | 开始时间 | 结束时间 | 周转时间 | 带权周转时间 |
| P1 | 0 | 3 | 0 | 3 | 3 | 1.0 |
| P2 | 2 | 6 | 3 | 9 | 7 | 1.17 |
| P3 | 4 | 4 | 9 | 13 | 9 | 2.25 |
| P4 | 6 | 5 | 13 | 18 | 12 | 2.4 |
| P5 | 8 | 2 | 18 | 20 | 12 | 6.0 |
| 平均 |  | | | | 8.6 | 2.56 |

表2 采用短作业优先调度算法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 进程 | 就绪时刻 | 运行时间 | 开始时间 | 结束时间 | 周转时间 | 带权周转时间 |
| P1 | 0 | 3 | 0 | 3 | 3 | 1.0 |
| P2 | 2 | 6 | 3 | 9 | 7 | 1.17 |
| P5 | 8 | 2 | 9 | 11 | 3 | 1.5 |
| P3 | 4 | 4 | 11 | 15 | 11 | 2.75 |
| P4 | 6 | 5 | 15 | 20 | 14 | 2.8 |
| 平均 |  | | | | 7.6 | 1.84 |

表3 采用高响应比优先调度算法

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 进程 | 就绪时刻 | 运行时间 | 开始时间 | 结束时间 | 周转时间 | 带权周转时间 |
| P1 | 0 | 3 | 0 | 3 | 3 | 1.0 |
| P2 | 2 | 6 | 3 | 9 | 7 | 1.17 |
| P3 | 4 | 4 | 9 | 13 | 9 | 2.25 |
| P5 | 8 | 2 | 13 | 15 | 7 | 3.5 |
| P4 | 6 | 5 | 15 | 20 | 14 | 2.8 |
| 平均 |  | | | | 8.0 | 2.14 |

表4 采用时间片轮转调度算法（时间片=1）

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 进程 | 就绪时刻 | 运行时间 | 开始时间 | 结束时间 | 周转时间 | 带权周转时间 |
| P1 | 0 | 3 | 0 | 4 | 4 | 1.33 |
| P2 | 2 | 6 | 2 | 18 | 16 | 2.67 |
| P3 | 4 | 4 | 5 | 17 | 13 | 3.25 |
| P4 | 6 | 5 | 7 | 20 | 14 | 2.8 |
| P5 | 8 | 2 | 10 | 15 | 7 | 3.5 |
| 平均 |  | | | | 10.8 | 2.71 |

1. 某系统中有5个进程，每个进程的运行时间、优先级（优先级越大越优先运行）和到达时刻如表5所示，给出采用不可抢占优先级调度和抢占式优先级调度算法时，各进程的执行情况，包括开始时间、结束时间、周转时间，平均周转时间。

表5 五个进程的相关数据

|  |  |  |  |
| --- | --- | --- | --- |
| 进程 | 到达时刻 | 运行时间 | 优先级 |
| P1 | 0 | 10 | 4 |
| P2 | 1 | 1 | 6 |
| P3 | 2 | 2 | 2 |
| P4 | 3 | 1 | 3 |
| P5 | 4 | 5 | 6 |

不可抢占（平均周转时间 12.6ms）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 进程 | 到达时刻 | 运行时间 | 优先级 | 结束时间 | 周转时间 |
| P1 | 0 | 10 | 4 | 10 | 10 |
| P2 | 1 | 1 | 6 | 11 | 10 |
| P3 | 2 | 2 | 2 | 19 | 17 |
| P4 | 3 | 1 | 3 | 17 | 14 |
| P5 | 4 | 5 | 6 | 16 | 12 |

可抢占（平均周转时间 10.6ms）

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 进程 | 到达时刻 | 运行时间 | 优先级 | 结束时间 | 周转时间 |
| P1 | 0 | 10 | 4 | 16 | 16 |
| P2 | 1 | 1 | 6 | 2 | 1 |
| P3 | 2 | 2 | 2 | 19 | 17 |
| P4 | 3 | 1 | 3 | 17 | 14 |
| P5 | 4 | 5 | 6 | 9 | 5 |

### 第三章 同步、通信与死锁

1、某银行提供一个服务窗口和10个供顾客等待的座位。顾客到达银行时，若有空座位，则到取号机上领取一个号，等待叫号。取号机每次仅允许一位顾客使用。当营业员空闲时，通过叫号机选取一位顾客，并为其服务。顾客和营业员的活动过程描述如下：

cobegin

{

   顾客进程i

   {   从取号机获得一个号码；

       等待叫号；

       获得服务；

   }

   营业员进程

   { while(TRUE)

     {  叫号；

        为顾客服务；

      }

    }

}  coend

请添加必要的信号量和P、V操作，实现上述过程中的互斥与同步。要求写出完整的过程，说明信号量的含义并赋初值。

解：

semaphore mutex=1; //互斥使用取号机信号量

semaphore empty=10; //空座位的数量信号量

semaphore full=0; //已占座位的数量信号量

semaphore service=0; //等待叫号信号量

cobegin

{ 顾客进程i

{ P(empty); //递减空座位数量

P(mutex); //进入临界区

从取号机获得一个号；

V(mutex); //离开临界区

V(full); // 递增已占座位数

P(service); //等待叫号

}

营业员进程

{ while(true)

{ P(full); //递减已占座位数

V(empty); //递增空座位数量

V(service); //叫号

为顾客服务;

}

}

}coend

int waiting=0;

semaphore cu=0, s=1; //同步信号量，初值代表资源个数

semaphore mutex=1；//共享变量waiting的互斥访问信号量

cobegin

{ 顾客进程i

{ P(mutex); //进入临界区

if(waiting<10)

{ waiting++；

从取号机获得一个号；

V(cu); //递增等待顾客数

V(mutex); //离开临界区

P(s); //请求服务员叫号，如果服务员忙，则顾客等待

}

else

V(mutex); //离开临界区

}

营业员进程

{ while(true)

{ P(cu); //递减顾客数，如果顾客为0，则营业员等待

P(mutex);

waiting--;

V(s); //叫号，准备为顾客服务

V(mutex);

为顾客服务;

}

}

}coend

2、（有界缓冲区问题）三个进程P1、P2、P3互斥使用一个包含N（N>0）个单元的缓冲区，P1每次用produce( )生成一个正整数并用put( )送入缓冲区的某一个空单元中；P2每次用getodd( )从该缓冲区中取出一个奇数并用countodd( )统计奇数个数； P3每次用geteven( )从该缓冲区中取出一个偶数并用counteven(  )统计偶数个数。

请用信号量机制实现这三个进程的同步与互斥活动，并说明所定义信号量的含义。要求用伪代码描述。

解：

semaphore mutex=1; //缓冲区操作互斥信号量

semaphore odd=0,even=0; //奇数、偶数进程的同步信号量

semaphore empty=N; //空缓冲区单元个数信号量

int number;

main( )

{ cobegin

{ 进程P1:

{ while (true)

{ number=produce();

P(empty); //递减空缓冲区的单元个数

P(mutex); //互斥访问缓冲区

put( );

V(mutex); //恢复访问缓冲区

if(number%2==0)

V(even); //允许取偶数

Else V(odd); //允许取奇数

}

}

进程P2

{ while (true)

{ P(odd); //判断是否有奇数可取

P(mutex); //互斥访问缓冲区

getodd( );

V(mutex); //恢复访问缓冲区

V(empty); //递增空缓冲区单元个数

countodd( );

}

}

进程P3

{ while (true)

{ P(even); //判断是否有偶数可取

P(mutex); //互斥访问缓冲区

geteven( );

V(mutex); //恢复访问缓冲区

V(empty); //递增空缓冲区单元个数

counteven( );

}

}

} coend

}

3、简述进程之间的基本通信方式？

进程通信根据交换信息量的多少分为高级通信和低级通信。

低级通信一般只传送一个或几个字节的信息，以达到控制进程执行速度的效果（如P、V操作）。

高级通信机制用于进程间高效地传送大量的数据，大致归为如下三类：

共享内存：在内存中分配一片空间作为共享存储区。需要进行通信的进程把它附加到自己的地址空间中，不需要则把它取消。

管道文件：用于连接读、写进程以实现通信的共享文件。

消息传递：以消息为单位在进程间进行数据交换。

4、什么是死锁？产生死锁的原因是什么?

所谓死锁是指多个进程因竞争系统资源或相互通信而处于永久等待状态，若无外力作用，这些进程都将无法向前推进。

产生死锁的原因：一是由于多进程共享的资源不足而引起的竞争资源；二是由于进程在运行过程中具有异步性特征，进程推进顺序不当。

5、现有5个进程P1、P2、P3、P4、P5，共享R1、R2、R3这3类资源，进程对资源的需求量和T0时刻分配情况如下图所示，若系统还剩余资源数如下：Available=(2，3，3)，请按银行家算法回答下列问题。

R1 R2 R3

R1 R2 R3

2 1 2

4 0 2

1 0 5

2 0 4

3 1 4

5 5 9

5 3 6

4 0 11

4 2 5

4 2 4

P1

P2

P3

P4

P5

Allocation=

Max=

（1）T0时刻系统处于安全状态吗？若是，请给出安全序列。

T0时刻的资源分配表如下：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 进程 | Allocation | | | Need | | | Available | | |
| A | B | C | A | B | C | A | B | C |
| P1 | 2 | 1 | 2 | 3 | 4 | 7 | 2 | 3 | 3 |
| P2 | 4 | 0 | 2 | 1 | 3 | 4 |  |  |  |
| P3 | 1 | 0 | 5 | 3 | 0 | 6 |  |  |  |
| P4 | 2 | 0 | 4 | 2 | 2 | 1 |  |  |  |
| P5 | 3 | 1 | 4 | 1 | 1 | 0 |  |  |  |

利用银行家算法对T0时刻的资源分配情况进行分析，可得T0时刻的安全性检测表，如下：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 进程 | Work | | | Need | | | Allocation | | | Work+Allocation | | | Finish |
| A | B | C | A | B | C | A | B | C | A | B | C |
| P5 | 2 | 3 | 3 | 1 | 1 | 0 | 3 | 1 | 4 | 5 | 4 | 7 | true |
| P4 | 5 | 4 | 7 | 2 | 2 | 1 | 2 | 0 | 4 | 7 | 4 | 11 | true |
| P3 | 7 | 4 | 11 | 3 | 0 | 6 | 1 | 0 | 5 | 8 | 4 | 16 | true |
| P2 | 8 | 4 | 16 | 1 | 3 | 4 | 4 | 0 | 2 | 12 | 4 | 18 | true |
| P1 | 12 | 4 | 18 | 3 | 4 | 7 | 2 | 1 | 2 | 14 | 5 | 20 | true |

可见，T0时刻存在一个安全序列{P5,P4,P3,P2,P1}，故T0时刻是安全的。

1. 若T0时刻进程P2提出资源请求Request2(0, 3, 4)，系统能分配资源给它吗？给出原因。

因请求资源数(0, 3, 4)大于剩余资源数(2, 3,3)，所以不能分配。

1. 在(2)的基础上，若进程P4请求资源Request4(2, 0, 1)，系统能分配资源给它吗？给出原因。

按银行家算法进行检查：

* + 1. P4请求资源数(2, 0, 1) ≤P4资源需求量(2, 2,1)
    2. P4请求资源数(2, 0, 1) ≤剩余资源数(2,3,3)
    3. 试分配并修改相应数据结构，由此得到的资源分配表如下：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 进程 | Allocation | | | Need | | | Available | | |
| A | B | C | A | B | C | A | B | C |
| P1 | 2 | 1 | 2 | 3 | 4 | 7 | 0 | 3 | 2 |
| P2 | 4 | 0 | 2 | 1 | 3 | 4 |  |  |  |
| P3 | 1 | 0 | 5 | 3 | 0 | 6 |  |  |  |
| P4 | 4 | 0 | 5 | 0 | 2 | 0 |  |  |  |
| P5 | 3 | 1 | 4 | 1 | 1 | 0 |  |  |  |

* + 1. 再利用安全性算法检查系统是否安全，可得到P4请求资源后的安全性检测表，如下：

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 进程 | Work | | | Need | | | Allocation | | | Work+Allocation | | | Finish |
| A | B | C | A | B | C | A | B | C | A | B | C |
| P4 | 0 | 3 | 2 | 0 | 2 | 0 | 4 | 0 | 5 | 4 | 3 | 7 | true |
| P5 | 4 | 3 | 7 | 1 | 1 | 0 | 3 | 1 | 4 | 7 | 4 | 11 | true |
| P3 | 7 | 4 | 11 | 3 | 0 | 6 | 1 | 0 | 5 | 8 | 4 | 16 | true |
| P2 | 8 | 4 | 16 | 1 | 3 | 4 | 4 | 0 | 2 | 12 | 4 | 18 | true |
| P1 | 12 | 4 | 18 | 3 | 4 | 7 | 2 | 1 | 2 | 14 | 5 | 20 | true |

可见，此时存在一个安全序列{P4,P5,P3,P2,P1}，故该状态是安全的，可以立即将P4所申请的资源分配给它。

### 第四章 存储管理

1、 何谓地址转换？哪些方法可以实现地址转换？

地址转换就是将程序的逻辑地址转换成物理地址，又称为地址重定位，包括静态重定位和动态重定位两种方法。

【静态重定位：在装入一个程序时，把程序中的指令地址和数据地址全部转换成物理地址。这种转换工作是在程序执行前集中完成的，在程序执行过程中无需再进行地址转换。

动态重定位：在装入一个程序时，不进行地址转换，而是直接把程序装到分配的区域中。在程序执行过程中，每当执行一条指令时，由硬件的地址转换机构转换成物理地址。

静态重定位的特点是在一个程序装入内存时必须分配其要求的全部内存空间，如果没有足够的空闲内存，就不能装入该程序。此外，程序一旦进入内存后，在整个运行过程中，不能在内存中移动，也不能再申请内存空间。

动态重定位的特点是可以将程序分配到不连续的存储区中。在程序运行之前可以只装如它的部分代码即可投入运行，然后在程序运行期间，根据需要动态申请分配内存，便于程序段/页的共享，可以向用户提供一个比存储空间大得多的地址空间。但动态重定位需要附加的硬件支持，且实现存储管理的软件算法比较复杂。】

2、 试比较分页式存储管理和分段式存储管理。

相似之处：两种存储方式中，程序均可在内存中非连续存储，通过地址转换机构将逻辑地址映射到物理内存中。

主要区别：

1. 页是信息的物理单位，与源程序的逻辑结构无关；段是信息的逻辑单位，由源程序的逻辑结构及含义来决定，每一段在逻辑上是一组相对完整的信息。
2. 页的大小是固定的，由操作系统决定，对用户是不可见的。页面只能从页大小的整数倍开始，页与页之间可以连续，也可以不连续。而段长由用户根据需要来确定，是用户可见的，每个段不一定等长，段起始地址可从任何内存地址开始，段与段之间可以连续，也可以不连续。
3. 分页的程序地址空间是一维的，引入的目的是实现离散分配，以消减内存的外部碎片，提高内存的利用率。在分段存储管理中，源程序经链接装配后仍保持二维地址结构，引入的目的是满足用户模块化程序设计的需要。
4. 分页存储管理不易实现共享和动态连接，而分段存储管理则很容易实现。

3、（教材P250 第14题)设有一页式存储管理系统，向用户提供的逻辑地址空间最大为16页，每页2048字节，内存总共有8个存储块。试问逻辑地址至少应为多少位？内存空间有多大？

**答：**逻辑地址211 ×24 ，故为15位。内存大小为23×211=214B=16KB。

4、（教材P250 第15题）在一分页存储管理系统中，逻辑地址长度为16位，页面大小为4096字节，现有一逻辑地址为2F6AH，且第0、1、2页依次存在物理块10、12、14号中，问相应的物理地址为多少？

**答：**因为逻辑地址长度为16位，而页面大小为4096字节，所以，前面的4位表示页号。把2F6AH转换成二进制为：0010 1111 0110 1010，可知页号为2。故放在14号物理块中，写成十六进制为：EF6AH。