上 海 大 学

2022-2023 秋季学期

《操作系统 A（08695030）》课程考核报告

学　号：20123101

姓　名：李昀哲

课程考核评分表

序号	题号	分值	成绩
1	题目 1	20	
2	题目 2	20	
3	题目 3	20	
4	题目 4	20	
5	题目 5	20	
考核成绩		100	
评阅人签名			

计算机工程与科学学院

2022 年 11 月

一、注意事项：

1、课程考核必须由考生独立完成。考核报告提交结束后将进行查重处理，对有抄袭现象的材料， 考核成绩作 0 分处理!！！

2、考核报告在 11 月 23 日中午 12：00 之前提交到超星平台上任课老师课程作业的相关目录，逾期没有提交的同学作缺考处理。

3、考核报告用 PDF 文件格式，文件名格式为：学号-姓名.PDF,例如：19120000-张三.PDF。

二、考核题目

题目 1 （20 分）

在操作系统发展史上，微内核是一个十分重要的操作系统体系结构，总的来说微内核并不是一个成功的体系结构，在主流的操作系统中，没有采用或者抛弃了微内核结构，但是微内核结构仍然吸引学术界的关注， 也存在一些基于微内核构建的操作系统。请你根据自己的理解回答下列问题：

（1） 什么是微内核？（2 分）微内核中包括哪些基本功能。（4 分）

（2） 和宏内核相比，微内核有哪些优点？（5 分）存在哪些问题？（2 分）

（3） 请查询资料介绍两个微内核（2 分）和两个宏内核（2 分）的实例。

（4） 如果由你设计实现一个操作系统内核，你是采用微内核还是宏内核？给出你的理由。（3 分）

题目 2 （20 分）

请你根据自己的理解回答下列问题：

（1） 请简答核心态、用户态、特权指令、普通指令四个概念（每个 2 分共 8 分），以及四者的相互关系（2 分）？

（2） 请简答系统功能调用和库函数的概念（每个 4 分共 8 分）以及二者的区别（4 分）。

题目 3：（20 分）

请你根据自己的理解回答下列问题：

（1） 处理机调度的多级反馈队列进程调度算法：（a）描述该算法的设计思想（2 分）；（b）总结该算法的必要组成部分（3 分）；（c）分析该算法针对不同类型进程的调度能力、并分别举例说明调度过程（5 分）。

（2） 死锁：（a）描述死锁的概念（2 分）；（b）描述死锁产生的必要条件（2 分）；（c）分析死锁安全状态和安全序列之间的关系（3 分）；（d）分析银行家算法实现的数据结构、算法过程和算法适用条件（3 分）。

题目 4：（20 分）

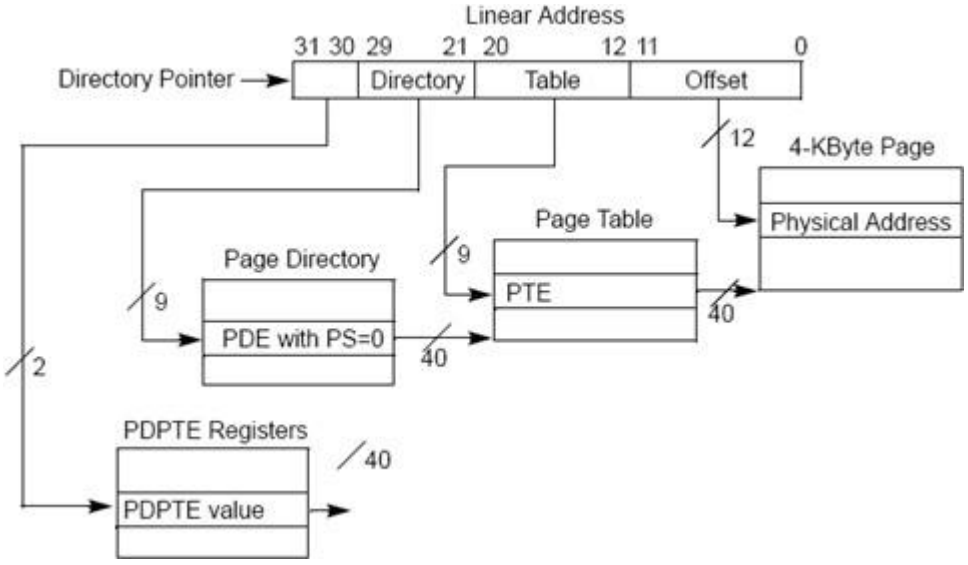
（1） 下图是奔腾处理器在 PAE 模式下的地址映射示意图，对照该图请回答：

a) 计算虚拟地址空间的大小（2 分）。

b) 计算物理地址空间的大小（3 分）。

c) 论述地址转换过程（3 分）。

d) 在虚拟空间远小于物理空间的情况下，如何充分利用存储资源（2 分）。



（2） 一个个人电脑销售商到上海大学推销他的产品，声称他们公司的电脑性能非常高，原因是他们的磁盘驱动程序使用了电梯算法。你被打动并购买了一台该销售商的电脑。回家后你写了一个程序随机地读分布在磁盘上的 10000 个磁盘块。令你惊讶的是：你的测试的结果与先到先服务方式得出的结果竟然完全一样。请问：

a) 什么是电梯算法（2 分），该算法有何优点（2 分）？

b) 判断商人是否在撒谎（2 分）？

c) 对该测试结果给出一个合理的解释（4 分）。

题目 5：（20 分）

请你根据自己的理解回答下列问题：

（1） 简述下列系统功能调用的功能：fork()、exec()、wait()、exit()、getpid()。（5 分，每个 1 分）

（2） 写出 C 语言 main 函数的完整格式（2 分），并回答 main 函数入口参数和系统功能调用 exec()的关系（1 分），回答 main 函数的返回值和系统功能调用 wait ()的关系（1 分），回答 main 函数入口参数与返回值和 shell 语言中内置变量的关系。（1 分）

（3） 请写出一个你上机时使用了上述五个系统功能调用的程序，简述程序的功能，简述你亲自调试该程序时遇到了哪些问题。（5 分）请不要抄袭，你写的程序不可能和别人的是一样的！

（4） 写一个不少于 200 字的课程总结，包括课程体会，以及对课程教学的建议。（5 分）请不要抄袭，你的课程体会不可能和别人的是一样的！

<题目部分结束，总计 5 道题目>

三、《操作系统A（08695030）》课程考核报告

注意：

1）题目论述要详细，但不要冗长，以下报告篇幅可以自行增加；

2）需要画图的部分可以用画图工具，也可以使用纸张手绘拍照插入进来；

3）报告正文使用宋体、五号字、1.25 倍行距排版。

题目 1：

1. **什么是微内核？微内核中包含哪些功能？**

   a) 微内核是一种操作系统的体系结构，只在内核中保留最基本的核心功能，使内核“足够小”（near-minimum）；

   绝大部分操作系统的功能移到微内核外的一组服务器进程中实现，运行于用户态，客户进程与服务器进程之间通过微内核提供的“消息传递”机制实现信息交互，因此是基于“客户/服务器”模式的；

   微内核中应用“机制与策略分离”原理，将机制（实现某一功能的具体执行机构，往往依赖于硬件）放在微内核中，将策略（机制基础上实现功能的某种算法）置于微内核外；

   采用面向对象技术。

   b) 微内核通常具有的功能包括：进程（线程）管理、低级存储器管理、中断和陷入处理。

   但需要注意的是，一般都采用a)中提到的“机制与策略分离”原理，仅仅将功能的机制部分以及与硬件紧密相关的部分放入微内核，如属于调度功能机制的“优先级队列”、页表机制、地址变换机制等。

2. **和宏内核相比，微内核有哪些优点？存在哪些问题？**

优点：微内核的建立基于模块化和层次化结构，采用了客户/服务器模式和面向对象程序设计技术，相较于宏内核将内核的所有功能（包括文件系统、内存管理、设备驱动、进程调度等等）实现在一起，作为一个a.out文件运行于核心态，微内核在以下方面具有优势：

1）更高的可扩展性。微内核大部分功能由相对独立的若干服务器进程实现，对于功能的增、删、改，只需对服务器进行操作，更加灵活。而对于宏内核的修改，往往是牵一发而动全身的；**2）更高的安全性和可靠性。**微内核提供了规范且精简的应用程序接口（API），所有服务器都运行在用户态，通过消息传递机制通信，一个服务器出现的错误，不会影响内核和其他服务器。而宏内核模块间没有很强的隔离机制，一个bug可能导致整个内核的错误；**3）可移植性强。**微内核中与特定CPU、I/O设备有关的代码均放在内核的硬件隐藏层中，各种服务器均与硬件平台无关，将操作系统移植到另一个硬件平台的代价较小；**4）提供了对分布式系统的支持。**客户和服务器、服务器和服务器均通过消息传递机制通信，使其较好地支持分布式系统；**5）融入了面向对象技术**，通过“封装”、“继承”、“多态”等特性，提高了系统的正确性、可靠性，减少开发系统的开销。

问题：当然，相较于宏内核作为一个整体运行，内核中模块交互采用函数调用实现的高性能优势，微

内核的模块间采用进程通信的方式就存在一定的问题：

1）客户/服务器模式和消息传递机制虽带来了许多如较强的可扩展性、可移植性等，但无法避免的降低了微内核操作系统的运行效率，在完成一次客户对操作系统的服务请求时，需要利用消息实现多次交互和多次用户/核心模式的切换，模式切换本就是个非常昂贵的开销，多次的切换会产生更大的、极大降低效率的开销。若将频繁使用的系统服务移回内核，又会使得微内核容量明显增大；**2）**宏内核往往会有先发优势，新的微内核有时不得不复用或重新实现一部分宏内核已有的功能来提供兼容性。

3. 请查询资料介绍两个微内核和两个宏内核的实例。

微内核:

a) **QNX:** QNX是一种商用、类Unix的微内核实时操作系统，面向嵌入式系统。其满足实时性要求的特点，使得它广泛应用于交通、能源、医疗、航空航天领域，尤其是车载领域，QNX在车用市场的占有率达到75%，各大汽车品牌的汽车电子平台几乎都采用了基于QNX技术的系统。当然在未来10年智能汽车高速发展的时期，QNX是否能保持其龙头地位，还不得而知。凭借其目前在车载市场的统治地位，QNX也被认为是最成功、甚至是唯一成功的微内核操作系统。

b) **Exokernel：**外核是一种比较极端但想法很好的设计方式，由MIT研究人员开发，在1995年左右停止了更新，目前也并未投入商业系统。它的设计动机是：传统的内核往往都对硬件资源作了抽象（abstraction），硬件资源对于应用程序来说是不可见的，应用程序只能和抽象模型进行交互；外核不提供抽象，“只有应用才知道最适合的抽象”，因此能让应用程序直接请求一块特定的物理空间，系统本身只保证被请求的资源当前是空闲的，应用程序就可以直接存取，将服务应用的功能作为库提供给用户，称为库OS（LibOS），来实现更高的性能。

宏内核:

a) **GNU/Linux：**一个基于Linux内核的开源类Unix操作系统，第一版在1991年发布。Linux实际上就是一个完整的可执行文件，且拥有最高权限来运行。最初是为基于Intel x86架构的个人计算机开发，后被移植到其他平台，包括嵌入式系统。基本思想是：一切都是文件；每个文件都有确定的用途。目前有很多不同的发行版，如基于社区的debian等，和基于商业开发的Red Hat等。

b) **UNIX：**UNIX是一个多任务、多用户计算机操作系统，支持多种处理器架构，属于分时操作系统，由Ken Thompson等人开发。结构上分为kernel和shell，kernel部分承担系统内部的各模块功能如进程控制和文件子系统等；shell部分包括用户界面、系统实用程序等，用户通过shell使用计算机。

4. 如果由你设计实现一个操作系统内核，你是采用微内核还是宏内核？给出理由

我会选择宏内核。对于操作系统内核的设计，硬件条件是必须纳入考虑范围的。在理论上，不可否认微内核分布式解耦的思路非常好，但对于实际设计和投入使用，微内核会造成资源的浪费，正如Linus说的“微内核的想法在现在的硬件条件下根本就是在浪费资源”。同时，宏内核的系统调用和相对较少的模式切换，使得宏内核运行效率较高，不会存在系统服务之间大量复制的问题也不需要花费高昂的代价实现进程间通信。随着硬件技术的发展，或许在不远的将来能够实现高效的微内核架构，但就目前而言，宏内核是较优的选择。

题目 2：

1. **简答核心态、用户态、特权指令、普通指令的概念，以及四者的相互关系。**

核心态和用户态是处理机的两种状态，目的是在指令级实现安全与保护；

核心态：处理机较高权限的执行状态，该状态可以执行所有指令、访问所有的寄存器；

用户态：处理机较低权限的执行状态，该状态只能执行普通指令、访问指定的寄存器；

特权指令：不允许用户直接使用的指令，它对内存空间的访问基本不受限制，包括如I/O指令、存取用于内存保护的寄存器等；

普通指令：允许用户直接使用的指令，它不能直接访问系统中的软硬件资源，仅限于访问用户的地址空间。

关系：CPU处于“核心态”时，执行了某一条特权指令，修改PSW标志位，主动让出CPU使用权，使CPU进入“用户态”；CPU处于“用户态”时，只能执行普通指令，当发生中断或异常时，运行于“用户态”的CPU会进入“核心态”，执行相应的特权指令处理中断，如图1所示。

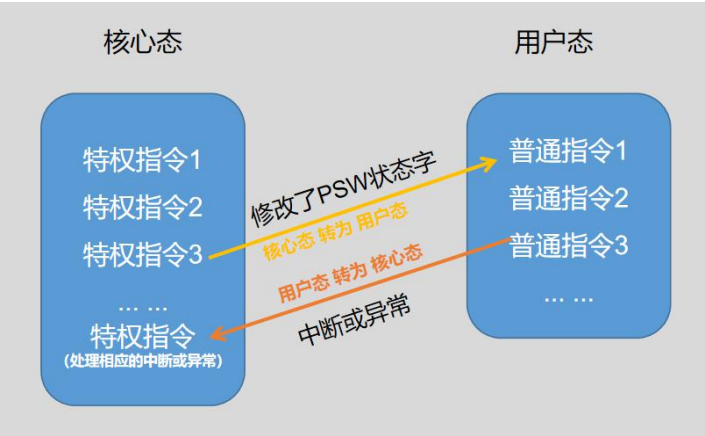


图1 核心态用户态及两种指令的关系

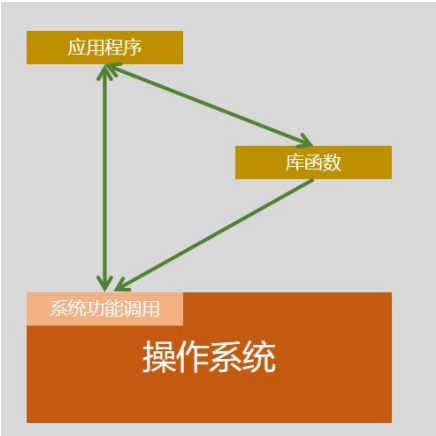


图2 库函数与系统调用

2. **简答系统功能调用和库函数的概念以及二者的区别。**

系统功能调用：操作系统提供给应用程序（编程人员）使用的接口，应用程序可以通过系统功能调用来请求获得操作系统内核的服务。

库函数：使用编程语言将函数封装入库，供用户使用的一种方式。它们具有明确的功能、入口调用参数和返回值。

区别：系统功能调用发生在内核空间，需要用户态到内核态的切换，造成较大的开销，各个操作系统的系统功能调用是不同的，一般没有跨操作系统的可移植性；库函数的可移植性好（如C库），且属于过程调用，调用开销小，有时也会将系统功能调用封装为库函数以隐藏系统功能调用的一些细节，使编程人员更便捷地使用。需要注意的是，库函数并不全都涉及系统调用，系统功能调用是比库函数更底层的接口。二者的关系如图2所示。

题目 3：

1. 处理机调度的多级反馈队列进程调度算法：

（a）描述该算法设计思想

多级反馈队列算法综合考虑了优先级调度算法、多队列调度算法、时间片轮转调度算法，对它们的设计思想进行权衡以避免各自的局限性。其思想主要是：不比事先知道各种进程所需的执行时间，还可以较好地满足各种类型进程的需要。

（b）总结该算法必要组成部分

1. **设置多个就绪队列，并为每个队列赋予不同的优先级**。第1级、第2级……优先级逐个降低；
2. **每个队列中进程运行时间片的大小不同**。队列优先级越高，其中进程的时间片越小；
3. **每个队列都采用先来先服务（FCFS）算法**。新进程进入内存后，首先放入第1级队列末尾，按FCFS原则等待调度；
4. **按队列优先级调度**。首先调度最高优先级队列中的进程，仅当第1级至i-1级队列均空闲时，第i级队列中的进程才会被调度。

（c）分析该算法针对不同类型进程的调度能力，举例说明调度过程.

终端型用户作业：终端型用户提交的作业大多属于交互型作业，通常较小，系统只要能使这些作业在第1级队列规定的时间片内完成，就可以满足终端型用户的需求，举例如图所示（队列设置如图3所示，后续图例仅展示调度情况）；

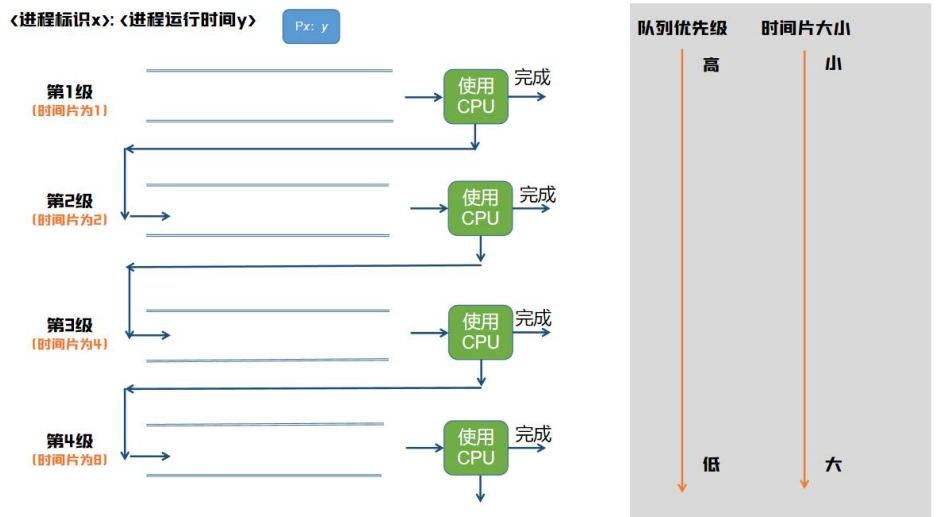


图3 多级反馈队列示意图

终端型用户作业的调度举例，各进程如表1所示，调度过程如图4所示

表1 终端型用户作业调度各进程

进程	到达时间	运行时间
P1	0	1
P2	1	1
P3	3	1

表2 短批处理作业调度各进程

进程	到达时间	运行时间
P1	0	3
P2	1	2
P3	3	1

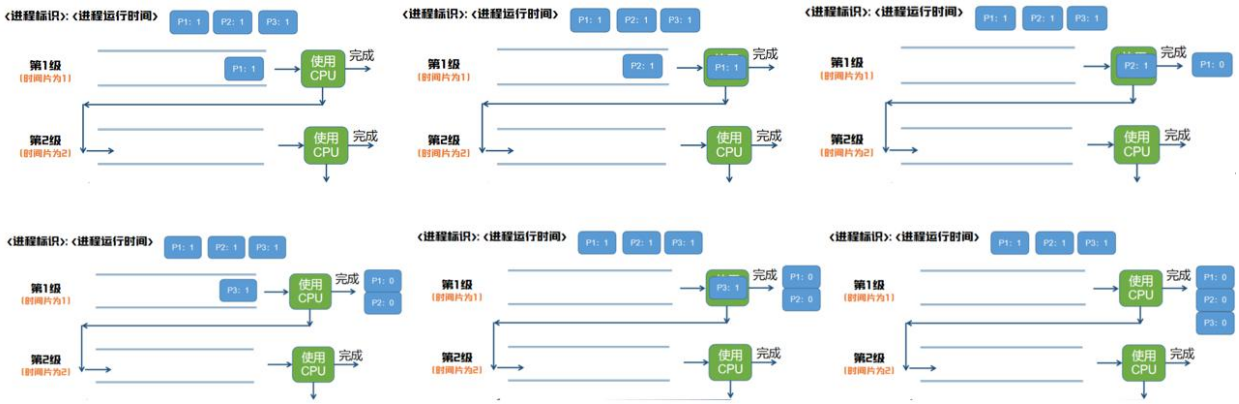


图4 终端型用户作业调度过程

短批处理作业用户：对于短批处理作业，如果可在第1级队列中执行完成，就可以获得与终端型作业一样的响应时间，对于稍长的短作业，也只需要在第2级和第3级队列中各执行一时间片就可以完成，周转时间较短。各进程如表2所示，调度过程如图5所示**（图中红字可能因文档格式转化后失真，红字为”处理机被更高优先级进程抢占，P1进程回到队尾”）**

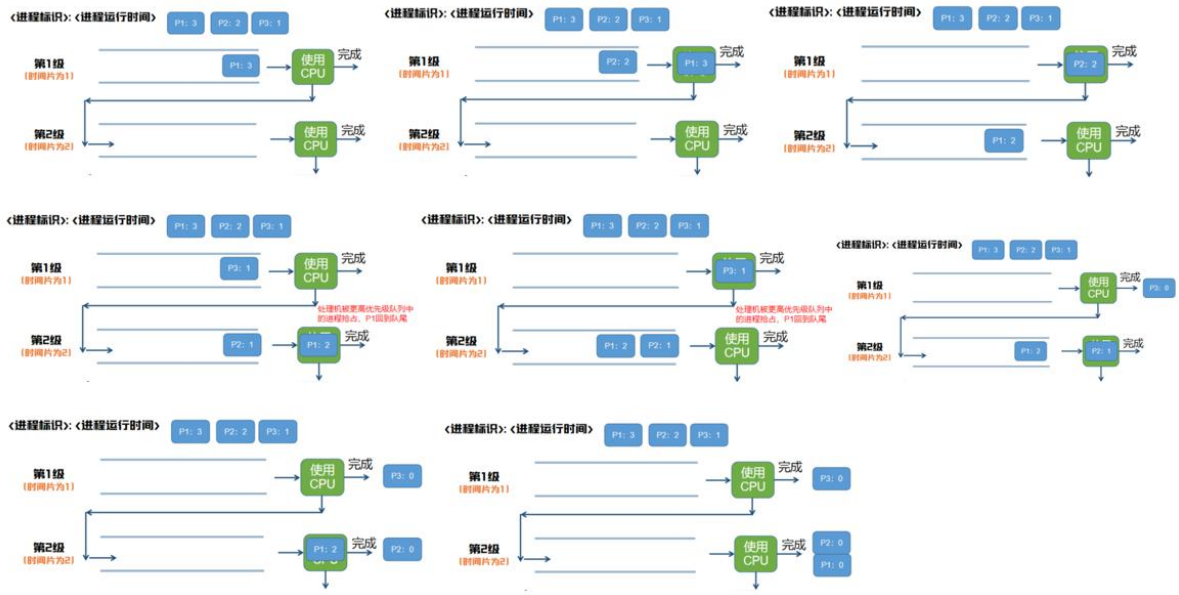


图5 短批处理作业调度过程

长批处理作业用户：对于长作业，它将依次在第1，2，…，n级队列中运行相应的时间片大小，然后再按轮转（RR）方式运行，用户不必担心因作业长期得不到处理而产生的“饥饿”问题。各进程如表3所示，调度过程如图6所示（**图中红字可能因文档格式转化后失真，红字为”处理机被更高优先级进程抢占，P1进程回到队尾”）**

表3 长批处理作业调度各进程

进程	到达时间	运行时间
P1	0	10
P2	1	8
P3	3	3

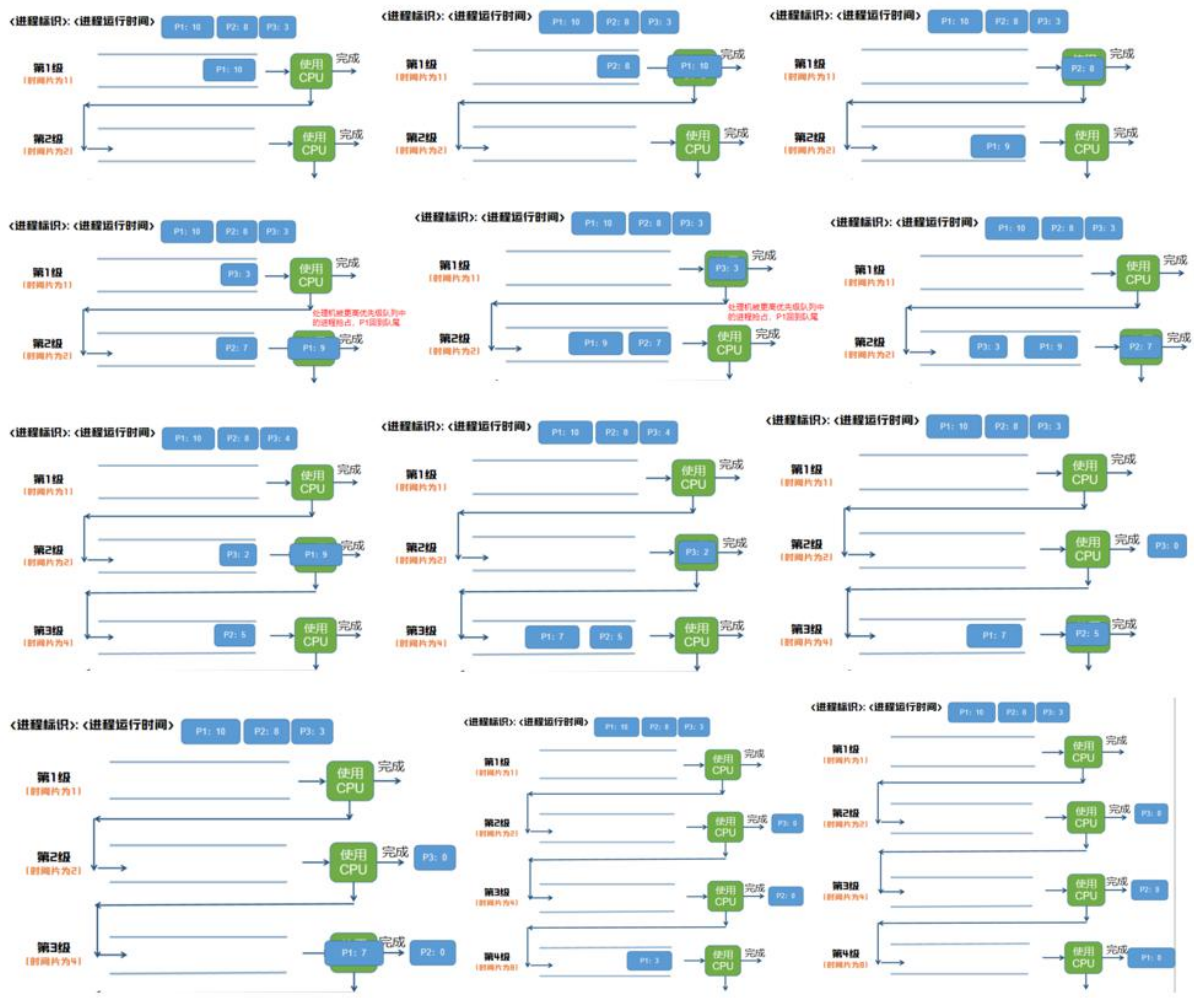


图6 长批组处理作业调度过程

2. 死锁：

(a)描述死锁概念

死锁是指多个进程因竞争资源而造成的一种互相等待的现象，无外力作用，这些进程就无法向前推进。如果一组进程中的每一个进程都在等待仅由该组进程中的其他进程才能引发的事件，这组进程就是死锁的。

(b)描述死锁产生的必要条件

互斥条件。进程对所分配到的资源进行排他性使用，即在一段时间内，某资源只能被一个进程占用。如果此时还有其他进程请求该资源，则请求进程只能等待占有该资源的进程使用完毕后释放；

请求和保持条件。进程已经保持了至少一个资源，但又提出了新的资源请求，而该资源已被其它进程占有，此时请求进程被阻塞，但对自己已经获得的资源保持不放；

不可抢占条件。进程已获得的资源在未使用完之前不能被抢占，只能在进程使用完时由自己释放；

循环等待条件。在发生死锁时，必然存在一个“进程—资源”的循环链，即进程集合{P_0,P_1,P_2,…,P_n}中的P_0正在等待一个P_1占用的资源，P_1正在等待P_2占用的资源，……，P_n正在等待已被P_0占用的资源；

(c)分析死锁安全状态和安全序列之间的关系

“安全状态”是指系统能按某种进程推进顺序（P_1,P_2,…,P_n）为每个进程P_i分配其所需资源，直至满足每个进程对资源的最大需求，使每个进程都可顺利地完成；序列（P_1,P_2,…,P_n）称为“安全序列”。如果一个系统无法找到这样一个安全序列，则该系统就处于不安全状态，就有进入死锁状态的可能，反之，只要系统处于安全状态，就能避免死锁。

(d)分析银行家算法实现的数据结构、算法过程和算法适用条件

实现银行家算法的数据结构如表所示，其中Max, Allocation, Need三个矩阵满足如下关系：

$$Need[i,j] = Max[i,j] - Allocation[i,j]$$

i, j用于表示进程i，进程j。

表4 实现银行家算法的数据结构及描述

数据结构	描述
可利用资源向量 Available	一个含有m个元素的**数组**，其中每一个元素代表一类可利用的资源数目，初值是系统中所配置的该类全部可用资源的数目，其数值随该类资源的分配和回收动态改变
最大需求矩阵 Max	一个$n \times m$的**矩阵**，定义了系统中n个进程中的每一个进程对m类资源的最大需求
分配矩阵 Allocation	一个$n \times m$的**矩阵**，定义了系统中每一类资源当前已分配给每一进程的资源数
需求矩阵 Need	一个$n \times m$的**矩阵**，表示每一个进程尚需的各类资源数

算法过程的描述如下，流程图如图所示：

设$Request_i$是进程P_i的请求向量，如果$Request_i[j] = k$，表示进程P_i需要k个R_j类型的资源。当P_i发出资源请求后，按以下步骤进行：

(1) 若 $Request_i[j] \leq Need[i,j]$，执行(2)，否则认为出错（所需的资源数超过宣布的最大值）；

(2) 若 $Request_i[j] \leq Available[j]$，执行(3)，否则进程$P_i$等待（尚无足够资源）；

(3) 系统尝试把资源分配给进程P_i，并修改如下数据结构中的数值：

$Available[j] = Available[j] - Request[j]$

$Allocation[i,j] = Allocation[i,j] - Request_i[j]$

$Need[i,j] = Need[i,j] - Request_i[j]$

(4) 执行安全性算法，检查此次资源分配后系统是否处于安全状态；

若安全，才正式分配；否则，此次试分配作废，进程P_i等待。

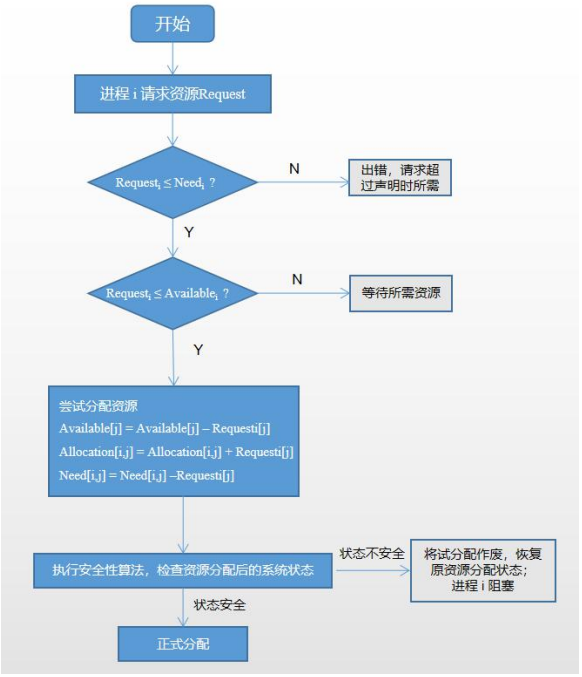


图7 银行家算法流程图

适用条件：根据算法的特点，有三大实际应用场景中的局限使得银行家算法很难适用：

1. 进程所需的最大资源数目进程自己都很难提前知道；
2. 进程的数量是不固定的、动态变化的；
3. 可用的资源可能会突然消失。

以上三点这使得银行家算法很少真正的在如今的操作系统中使用来避免死锁。

题目 4：

1. (a)虚拟地址空间大小

位30-31两位选择PDPTE，位21-29和位12-20分别选择页目录和页表，因此计算得到

$$2^2 * 2^9 * 2^9 = 2^{20}\ 页$$

$$2^{20} * 2^{12} = 4\ GB$$

共2^{20}页，每页4kB，因此虚拟地址空间大小是4GB。

(b) 物理地址空间大小

PAE模式、PS位为“零”的情况下，基址为40位，再根据偏移得到物理地址，偏移由12位表示，因此理论上的物理地址空间大小为：

$$2^{40} * 2^{12} = 2^{52}\ B$$

但在PAE模式下地址线的数量可能存在上限，物理地址空间大小可能无法达到2^{52}B。

(c) 地址转换过程

虚拟内存中的每一个虚拟页会映射到一个物理页，映射通过PDPTE寄存器、页目录、页表实现，可以看成是一个多级页表。PDPTE寄存器中存放PDPTE值、页目录存放页目录条目（Page-Directory Entry）、页表存放页表条目（Page Table Entry）。Linear Adddress列出了对应位的意义，如下：

位31,30:页目录指针表索引；

位21-29：页目录表索引；

位12-20：页表索引；

位0-11：页内偏移；

当进程访问某个虚拟地址中的数据时，首先获取地址目录指针指向的Linear Address，分页地址变换机构会将有效地址分为寄存器号、页目录号、页号和页内地址，再根据“号”去寄存器、页目录、页表中索引，再根据页内偏移确定物理地址。

(d) 当虚拟地址空间远小于物理地址空间时，如何充分利用存储资源

虚拟地址空间大小远小于物理地址空间大小往往是以地址变换为目的的，出现在多用户或多任务系统中，物理地址空间较大的情况下，单个任务只需要较小的地址空间，指令中地址字段的长度就可以缩短，以此充分利用存储资源。每个进程都有个虚空间，物理空间整个计算机只有一个，创建的进程如果足够多，物理空间就会很大，因此创建不了如此多的进程。

2.

(a) 什么是电梯算法，该算法有何优点

电梯算法也叫扫描算法，是一种磁盘调度算法，主要思想就是和电梯一样，会首先处理同一个方向的请求，直到同一个方向再无请求后才改变方向。当磁头自里向外移动时，电梯算法考虑的下一个访问对象应是其欲访问的磁道既在当前磁道之外侧，又是距离最近的。这样自里向外访问，直到再无更外的磁道需要访问。

优点是由于访问磁盘的时间主要是寻道时间，该算法的平均寻道长度较短，因此性能通常较好。对请求的处理相对公平，保证了每个请求都会得到处理，不会出现某个请求长期得不到满足的情况。

(b) 判断商人是否在撒谎

商人没有撒谎。对于一个算法的评判需要考虑其实际的应用场景，题中的“我”和商人的测试场景是无法确保一致的。磁盘读写的最小单位是扇区，是真实存在的，控制读写的磁盘臂和磁头是电梯算法实际控制的；而磁盘块则是一个虚拟的概念，程序对于磁盘块的读并不能和磁盘读写扇区画等号；随机的测试序列也可能出现“单调”的情况，导致FCFS和电梯算法得到相同的结果。因此题中“我”的测试是不严谨的，且仅凭借一次的测试结果判定商人撒谎，也是不合理的。

(c) 对该测试结果给出一个合理的解释

根据(b)中所述，题中“我”的测试是不严谨的，算法的目的是在多个读写请求中选择一个服务，算法测试应充分考虑各种不同的请求序列，在某些请求序列下，先到先服务算法和电梯算法是可能出现完全相同的情况的，如请求的提出先后和电梯算法的扫描过程完全一致、随机序列恰好是“单调”的等；除此以外程序对于磁盘块的读并非完全随机，可能也遵循了某种伪随机的规则，使得序列“单调”。因此导致该测试结果的主要矛盾就在请求序列，“单调”序列的产生具体是因为软硬件差异导致，还是因恰好随机产生的“单调”就无法下定论了。在众多的可能性中出现测试结果的一致性，并非不可能。

题目 5：

1. **简述下列系统功能调用的功能：fork()、exec()、wait()、exit()、getpid()。**

   fork(): 通过复制“调用进程（the calling process）”创建一个新的进程。新的进程称为子进程（child process），调用的进程称为父进程（parent process）。它们运行在不同的内存空间。在fork时，两个内存空间具有相同的内容，其中一个进程执行的对内存的写操作、文件映射等不会影响另一个进程。

   exec(): exec()是个函数族（family of functions），用以将当前的进程图像替换为一个新的进程图像。函数族中包括了execl(), execlp(), execle(), execv(), execvp(), execvpe()。其中l,v,e,p分别有不同的含义，如v表示传递的参数需要是个向量；e表示传递了环境变量参数等；

   wait()：用以等待当前进程子进程的“状态改变”，获取子进程状态改变的信息，如果成功将会把被终止子进程的进程号返回，如果出错，将会返回-1；

   exit()：exit()函数可以使普通的进程（normal procss）终止，状态值将会返回给父进程；

   getpid()：getpid()用以返回“被调用进程（calling process）”的进程号（process ID）；

2. **写出C语言main函数的完整格式；回答main函数入口参数和系统功能调用exec()的关系；回答main函数返回值和系统功能调用wait()的关系；回答main函数入口参数与返回值和shell语言中内置变量的关系**

   (a) main函数完整格式

```
int main (int argc, char * argv[], char ** envp[])
{
    return 0;
}
```

   (b) main函数入口参数和系统功能调用exec()的关系

   如上述exec()的简述，exec()函数族被调用时，会将参数传递给main函数的入口参数；

   (c) main函数返回值和系统功能调用wait()的关系

   main函数的返回值将返回给父进程，父进程通过wait()系统调用获取子进程状态的改变；

   (d) main函数入口参数与返回值和shell语言中内置变量的关系

   shell语言中的内置变量#，*，0的值分别是main函数入口参数的argc（参数个数）、全部参数值的字符串（包括程序名称）、第一个参数值（即程序名称）；内置变量“？”是上一个执行完成的程序的main函数的返回值。

3. **写出一个用到上述系统调用的程序，简述程序功能，简述调试中遇到的问题**

   在实验中出现了不少的问题，在不断出错的过程中，加深了对操作系统和对计算机学习的理解。在用管道实现进程通信的实验中，使用到了fork()和exec()，将父进程的输出通过管道作为子进程的输入，仿照课上所讲，实现了“ls|grep 1”命令的功能，程序可以实现列出当前目录下含“1”的文件，体会到了我们写的“软件”都是流水线上的小程序。程序如下：

```
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>

int main(int argc, char * argv[], char ** envp[])
{
    int p[2];
    pipe(p); // 相当于打开了两个文件，p[0]read端，p[1]write端；
    pid_t pid = fork();

    if (pid == 0){        // child
        // printf("%d", getpid());    // 调试用
        close(0);                     // 关闭标准输入，很重要！！
        dup(p[0]);
        close(p[0]);                  // 关闭pipe的read
        close(p[1]);                  // 关闭pipe的write
        char * arglist[5] = {"grep", "1"}; // 在当前目录下建一个p1.c，就可用grep查到带1的文件
        execvp(arglist[0], arglist);
    }
    else              // parent
    {
        close(1);     //关闭标准输出，很重要！！
        dup(p[1]);
        close(p[0]); // 和上面一样
        close(p[1]);
        execl("/bin/ls", "ls", NULL);
    }
        return 0;
}
```

遇到的问题：

1. 使用管道进行进程通信时，一个进程的输出无法作为另一个进程的输入，且不明白为什么要close(0)和close(1)，后发现需要先关闭标准输入输出，将管道的read端和write端复制到标准输入输出即可完成；

2. 尝试使用execvp()时，参数的使用没有多加思考，直接和execl()用了一样的参数形式，即：

   execvp("/usr/bin/ls", "grep", "1")

   毫无疑问，报错了；仔细阅读exec()系统调用的手册后，发现首先"v"表示参数列表为向量形式，其次"p"表示第一个参数可以直接是文件名，系统会在环境变量路径下查找，因此代码进行改动：

   ```
   char * arglist[5] = {"grep", "1"}; // 参数作为一个向量
   execvp(arglist[0], arglist);       // 符合要求的execvp调用方法
   ```

成功的调试结果如图所示：

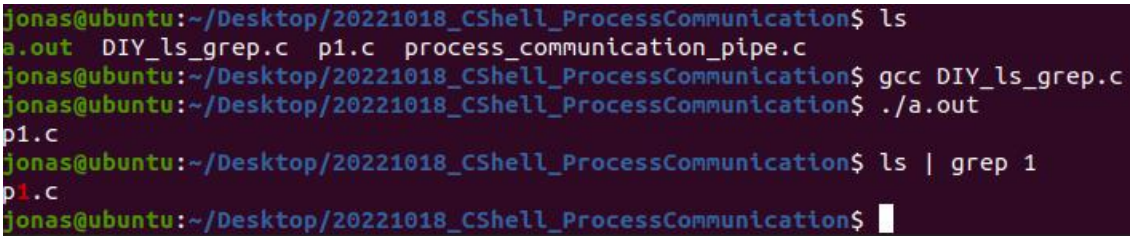


图8 程序实现“ ls|grep 1”的调试成功结果

4. 课程体会

对于操作系统的初学者，比如我，来说，通过自己动手做实验，以及和操作系统有较深理解的前辈沟通、讲述自己的理解，从中看到理解层次上的差距，才能更好地理解操作系统。刘老师安排的面试验收，或许就起到了这样的作用，“自己用专业的词讲清楚，才是真的学懂了”。

一个学期的学习说实话，自认为对操作系统只是对各种概念、原理有了粗浅的认识，还需要未来更深入的学习。不过在实验中，收获还是很大的，此前因为项目需要在arm架构下编程，因此接触了linux，但当时（或者说在本学期以前）对于那些命令，比如ls,ln,gcc,g++,make等，都只是机械地“背下来”，不知其所以然，也闹过很多笑话，但经过这学期，对过去的那些不解，终于得到了理解，非常有成就感，也能更好地和别人阐述它们的意思。

对操作系统的学习不可能仅限于课堂之上，使用计算机的过程中，冥冥之中都用到了操作系统所学，“身为计算机专业的学生，起码得知道点别人不知道的”，未来的学习中会在当前基础之上深入理解，也希望能和刘老师在未来有更多交流。教学建议上或许可以加入更多英文原版书中的内容，会产生对知识点新的启发，比如对“银行家算法实际中几乎没有投入使用”那一段英文的阐述，就让我对这个算法，有了更深层次、不局限于书本的认识。

〈报告结束〉