第 1 部分　习题解答

第 1 章习题解答

1.1　什么是操作系统？引入操作系统的目的有哪几方面？

答：操作系统是计算机系统中最基础的系统软件。它是这样一些程序模块的集合——管理和控制计算机系统中的硬件及其他软件资源，合理地组织计算机工作流程，以便有效地利用这些资源为用户提供一个具有足够的功能、使用方便、可扩展、安全和可管理的工作环境，从而在计算机与用户之间起到接口的作用。

引入操作系统的目的有以下 3 方面：

(1) 从用户的观点看，计算机是为用户提供服务的，计算机完成的任何工作都是为了满足用户的计算或处理需求。因此，引入操作系统是让计算机为用户提供最好的服务，构建用户和计算机之间的和谐交互环境。这要求计算机有良好的用户界面，使用户无须了解许多有关硬件和系统软件的细节，能够方便、灵活地使用计算机。同时，计算机还能为用户提供可靠、安全的服务管理，以保证用户得到可靠、安全的服务。

(2) 从系统管理人员的观点看，引入操作系统是为了合理地组织计算机工作流程，管理和分配计算机系统硬件及软件资源，使之能为多个用户高效率地共享，同时又能保证用户和计算机系统安全。因此，操作系统是计算机资源的管理者。

(3) 从发展的观点看，引入操作系统是为了给计算机系统的功能扩展提供支撑平台，使之在追加新的服务和功能时更加容易和不影响原有的服务与功能。

1.2　操作系统的基本功能有哪些？各功能对应的物理设备分别有哪些？

答：操作系统的基本功能是管理资源和提供用户接口，即管理和控制计算机系统中的所有硬件和软件资源，合理地组织计算机工作流程，并为用户提供良好的工作环境和友好的接口。资源管理分为处理机管理、存储管理、设备管理和信息管理。其中处理机管理解决对处理机分配调度策略、分配实施和资源回收等问题；存储管理的主要工作是对存储器进行分配、保护、扩充和管理；设备管理对通道、控制器、输入输出设备进行分配和管理；信息管理则是对系统的软件资源进行管理。操作系统还通过用户接口为用户提供使用计算机方便、灵活的手段。

1.3　什么是批处理、分时和实时系统？它们各有什么特点？

答：在批处理系统(batch processing operating system)中，操作员把用户提交的作业分类，把一批作业编成一个作业执行序列，由专门编制的监督程序(monitor)自动依次处理。其主要特征是用户脱机使用计算机、成批处理以及多道程序运行。

分时系统(time sharing operating system)把处理机的运行时间分成很短的时间片，按时间片轮转的方式把处理机分配给各进程使用。其主要特征是交互性、多用户同时性和独

立性。

实时系统(realtime operating system)在被控对象允许时间范围内作出响应。其主要特征是：对实时信息分析处理速度快，要求安全可靠，资源利用率低。

1.4 操作系统的基本类型有哪些？它们与操作系统发展历史有什么联系？

答：操作系统的基本类型有批处理操作系统、分时操作系统、实时操作系统、个人计算机操作系统、网络操作系统、分布式操作系统。

批处理操作系统是一种早期的大型机操作系统，现代操作系统大都具有批处理功能。

分时操作系统一般采用时间片轮转的方式，使一台计算机为多个终端用户服务。当今最流行的多用户分时操作系统 UNIX 起源于 20 世纪 60 年代的 Multics 系统。

实时操作系统是另一类联机的操作系统，其主要特点是提供即时响应和高可靠性。它主要是随着计算机应用于实时控制和实时信息处理领域而发展起来的。

个人计算机上的操作系统是联机的交互式单用户操作系统，它提供的联机交互功能与通用分时系统所提供的很相似。由于是个人专用，因此在多用户和分时所要求的处理机调度、存储保护方面将会简单得多。

计算机网络是通过通信设施将物理上分散的具有自治功能的多个计算机系统互联，实现信息交换、资源共享、可互操作和协作处理的系统。由于网络计算的出现和发展，现代操作系统的主要特征之一就是具有上网功能，因此人们一般不再特指某个操作系统为网络操作系统。

分布式操作系统可以定义为通过通信网络将物理上分布的具有自治功能的数据处理系统或计算机系统互联，实现信息交换和资源共享，协作完成任务。分布式操作系统还处在研究阶段，目前还没有真正实用的系统。20 世纪 90 年代出现的网络计算已使分布式操作系统变得越来越现实，软件构件技术的发展也加快了分布式操作系统的实现。

1.5 多道程序设计和多重处理有什么区别？

答：多道程序设计是作业之间自动调度执行、共享系统资源，并不是真正地同时执行多个作业；而采用多重处理的系统配置多个 CPU，能真正同时执行多道程序。要有效使用多重处理，必须采用多道程序设计技术，而多道程序设计原则上不一定要求多重处理的支持。

1.6 讨论操作系统可以从哪些角度出发？如何把它们统一起来？

答：讨论操作系统可以从 3 个角度出发。①操作系统是计算机资源的管理者；②操作系统为用户提供使用计算机的接口；③用进程管理思维研究操作系统，即围绕进程运行过程来讨论操作系统。

1.7 写出 1.7 节中巡回置换算法的执行结果。

答：1.7 节中的巡回置换算法要求如下。

设 i=1,2,3,4,5,6,7,p[i]= 4,7,3,1,2,5,6。当 k∈[1:n]时，k=p[k]。

(1) 算法如下：

```
local x, k              /* x 和 k 为局部变量 */
begin
    k←1                 /* 初始化 k */
    while k<=7 do
```

```
        x←k
        repeat
        print(x)
        x←p[x]
        until x=k
        k←k+1
    od
end
```

(2) k=1,2,…,7 时,执行结果分别如下:

```
1 4 1
2 7 6 5 2
3 3
4 1 4
5 2 7 6 5
6 5 2 7 6
7 6 5 2 7
```

1.8 计算机操作系统设计与哪些硬件有关?

答:计算机操作系统的重要功能之一是对硬件资源进行管理。因此设计计算机操作系统时应考虑下列与计算机硬件资源有关的问题:

(1) CPU 与指令的长度及执行方式。

(2) 内存、缓存和高速缓存等存储装置。

(3) 各类寄存器,包括各种通用寄存器、控制寄存器和状态寄存器等。

(4) 中断机构。

(5) 外部设备与 I/O 控制装置。

(6) 内部总线与外部总线。

(7) 对硬件进行操作的指令集。

1.9 考察一个小的嵌入式系统或物联网系统,写出系统引导和操作系统加载过程。

答:嵌入式系统的引导和操作系统加载过程分为两个阶段。

第一阶段引导程序通常完成以下工作:

(1) 硬件设备初始化。屏蔽所有中断,设置 CPU 速度和时钟频率,RAM 初始化,等等。

(2) 为第二阶段的引导程序加载准备 RAM 空间。初始化中断向量表。

(3) 复制第二阶段的二进制代码到 RAM 空间中。

(4) 设置好堆栈指针,为执行 C 语言代码做好准备。初始化堆栈。

(5) 跳转到第二阶段。

第二阶段引导程序通常完成以下工作:

(1) 其他硬件设备的初始化。

(2) 检测系统内存映象。

(3) 将操作系统内核映像及文件系统映像从 Flash 读取到系统 RAM 中。

(4) 为操作系统内核设置启动参数。

（5）调用操作系统内核。

1.10　以生态圈的观点阐述为何现在不能仅从技术角度评判操作系统。

答：因为人们不再关心操作系统的技术细节和执行效率，而是把注意力放在如何简捷地构建可跨软硬件平台、方便管理同时又能被尽可能多的普通用户使用的操作系统用户生态圈上。

第 2 章习题解答

2.1　什么是作业、作业步？

答：在一次应用业务处理过程中，从输入开始到输出结束，用户要求计算机所做的有关该次业务处理的全部工作称为一个作业。从系统的角度看，作业则是一个比程序更广的概念。作业由程序、数据和作业说明书组成。系统通过作业说明书控制文件形式的程序和数据，使之执行和操作。而且，在批处理系统中，作业是抢占内存的基本单位。也就是说，批处理系统以作业为单位把程序和数据调入内存以便执行。作业由顺序相连的不同作业步组成。

2.2　操作系统为普通用户、管理员用户以及程序开发人员分别提供了什么接口？请分别举个例子。

答：操作系统为普通用户和管理员用户提供的界面由一些操作命令组成，即命令控制界面。其中，每个命令实现和完成用户所要求的特定功能和服务，例如上网、在线处理、办公处理等。

操作系统为程序开发人员提供的唯一界面是系统调用。每一个系统调用都相当于一个能完成特定功能的子程序。程序开发人员通过系统调用可以访问操作系统的底层服务，并使用相关的软硬件资源。

2.3　作业由哪几部分组成？它们各有什么功能？

答：作业由 3 部分组成，包括程序、数据和作业说明书。程序和数据完成用户要求的业务处理工作。系统通过作业说明书控制文件形式的程序和数据，使之执行和操作。

2.4　作业的输入方式有哪几种？它们各有何特点？

答：作业的输入方式有 5 种，分别是联机输入方式、脱机输入方式、直接耦合方式、假脱机系统和网络联机方式，这 5 种输入方式的特点分别如下：

（1）采用联机输入方式时，用户和系统通过交互式会话输入作业。

（2）脱机输入方式利用低档个人计算机作为外围处理机进行输入处理，存储在后援存储器上，然后将此后援存储器连接到高速外围设备上和主机相连，从而在较短的时间内完成作业的输入工作。

（3）直接耦合方式把主机和外围低档机通过一个公用的大容量外存直接耦合起来，从而省去了在脱机输入中依靠人工干预来传递后援存储器的过程。

（4）假脱机（spooling）系统可译为外围设备同时联机操作。在假脱机系统中，多台外围设备通过通道或 DMA 器件与主机和外存连接起来，作业的输入过程由主机中的操作系统控制。

(5) 网络联机方式以上述几种输入方式为基础。当用户通过计算机网络中的某一台设备对计算机网络中的另一台主机进行输入操作时,就构成了网络联机方式。

2.5 试述假脱机系统的工作原理。

答:在假脱机系统中,多台外围设备通过通道或 DMA 器件与主机和外存连接起来,作业的输入输出过程由主机中的操作系统控制。操作系统中的输入程序包含两个独立的过程。一个是读过程,负责从外部设备把信息读入缓冲区,另一个是写过程,负责把缓冲区中的信息送入外存输入井。

在系统输入模块收到作业输入请求后,输入管理模块中的读过程负责将信息从输入装置读入缓冲区。当缓冲区满时,由写过程将信息从缓冲区写到外存输入井中。读过程和写过程反复循环,直到一个作业输入完毕。当读过程读到一个硬件结束标志后,系统再次驱动写过程把最后一批信息写入外存并调用中断处理程序结束该次输入。然后,系统为该作业建立作业控制块(JCB),从而使输入井中的作业进入作业等待队列,等待作业调度程序选中后进入内存。

2.6 操作系统为用户提供了哪些接口?它们的区别是什么?

答:操作系统为用户提供两个接口。一个接口是系统为用户提供的各种命令,用户利用这些操作命令组织和控制作业的执行或管理计算机系统。另一个接口是系统调用,编程人员使用系统调用请求操作系统提供服务,例如申请和释放外设等资源、控制程序的执行速度等。

2.7 作业控制的方式分为脱机和联机两种,这两种方式的区别是什么?

答:脱机控制操作系统按作业说明书的规定控制作业的执行,不直接干预作业的执行,由系统中的命令解释程序对作业说明书中的命令逐条解释执行。联机控制操作系统向用户提供一组联机命令,用户在终端输入命令来控制作业的执行过程,需要人与计算机交互配合执行。

2.8 列举一些 Linux 系统调用,并描述系统调用执行过程中从应用程序到内核的执行流程。

答:Linux系统调用从功能上大致可分为如下 6 类。

(1) 设备管理的系统调用,例如申请设备、释放设备、设备 I/O 和重定向、设备属性获取及设置、逻辑上连接和释放设备。

(2) 文件系统操作的系统调用,例如建立文件、删除文件、打开文件、关闭文件、读写文件、获得和设置文件属性。

(3) 进程控制的系统调用,例如终止或异常终止进程、载入和执行进程、创建和撤销进程、获取和设置进程属性。

(4) 存储管理的系统调用,例如申请和释放内存。

(5) 管理用的系统调用,例如获取和设置日期及时间、获取和设置系统数据。

(6) 通信的系统调用,例如建立和断开通信连接、发送和接收消息、传送状态信息、连接和断开远程设备。

2.9 简述在命令行中执行一条 Linux 命令的流程，如命令的解析、命令的查找、命令的执行等。

答：在 Linux 中，执行一条命令的流程如下。

（1）判断路径。判断用户是否按相对路径或绝对路径的方式输入命令，如果是，直接执行。

（2）检查别名。检查用户输入的命令是否为别名命令。Linux 可以通过 alias 命令给现有命令定义别名，即用自定义命令名称来替换原本的命令名称。

（3）判断用户输入的是内部命令还是外部命令。内部命令指的是解释器内部的命令，会被直接执行；外部命令通常是用户输入的命令，执行外部命令时跳转到步骤（4）。

（4）查找外部命令的可执行文件。当用户执行的是外部命令时，系统会在指定的多个路径中查找命令的可执行文件，而定义这些路径的变量被称为 PATH 环境变量，用来表示执行命令的可执行文件存放的位置。

（5）执行命令。

2.10 作业控制方式有哪几种？调查你周围的计算机的作业控制方式。

答：作业控制的主要方式有两种，即脱机方式和联机方式。

（1）脱机控制方式利用作业控制语言编写表示用户控制意图的作业控制程序，也就是作业说明书。作业控制语言的语句就是作业控制命令。不同的批处理系统提供不同的作业控制语言。

（2）联机控制方式不要求用户编写作业说明书，系统只为用户提供一组键盘或其他操作方式的命令。用户使用操作系统提供的操作命令和系统会话，交互地控制程序执行和管理计算机系统。

2.11 什么是系统调用？系统调用与一般用户程序有什么区别？系统调用与库函数和实用程序又有什么区别？

答：系统调用是操作系统提供给编程人员的唯一接口。编程人员利用系统调用，在源程序一级动态请求和释放系统资源，调用系统中已有的系统功能完成与计算机硬件部分相关的工作以及控制程序的执行速度等。因此，系统调用像一个黑盒子那样，对用户屏蔽了操作系统的具体动作，而只提供有关的功能。与一般用户程序、库函数和实用程序相比，系统调用程序是在核心态执行，调用它们时需要一个类似于硬件中断处理的中断处理机制提供系统服务。

2.12 简述系统调用的实现过程。

答：用户在程序中使用系统调用，给出系统调用名和函数后，即产生一条相应的陷入指令，通过陷入处理机制调用服务，引起处理机中断，然后保护处理机现场，取得系统调用功能号并寻找子程序入口，通过入口地址表调用系统子程序，然后返回用户程序继续执行。

2.13 为什么说分时系统没有作业的概念？

答：在分时系统中，每个用户得到的时间片有限，用户的程序和数据信息直接输入内存工作区中，和其他程序一起抢占系统资源并投入执行，而不必进入外存输入井等待作业调度程序选择。因此，分时系统没有作业控制表，也没有作业调度程序。

2.14 Linux 操作系统为用户提供了哪些接口？试举例说明。

答：Linux 系统为用户提供两种接口。一种是面向普通用户输入的操作命令的接口，即 shell；另一种是面向编程用户的接口，即系统调用。常见的 shell 命令有 login、logout、vi、emacs、cp、rm、ls、cc、link、adduser、chown、dbx、date 等，常见的系统调用有 ioctl、read、write、open、close、creat、execl、flock、stat、mount、fork、wait、exit、socket 等。

2.15 在装有 Linux 系统的计算机上查看有关 shell 的基本命令，并编写一个简单的 shell 程序，完成一个已有数据文件的复制和打印任务。

答：假设需要将文件 src.txt 复制为 dst.txt 并打印出来，shell 程序如下。

```
#!/bin/bash
#copy file
cat src.txt>dst.txt
#print file
cat src.txt>/dev/lp
```

2.16 用 Linux 文件读写的相关系统调用，编写一个复制文件的程序。

答：假设该程序的执行格式为 copy src dst，程序的代码如下。

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#define BUFSIZE 8192
int main(char **argv, int argc)
{
    if(argc!=3)
    {
        print("\n usage: copy src dst\n");
        return-1;
    }
    int src,dst;
    char buf[BUFSIZE];
    int n;
    src=open(argv[1], O_RDONLY);
    dst=open(argv[2], O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR | S_IXUSR);
    while((n=read(src,buf,BUFSIZE))>0)
    {
        if(write(dst,buf,n)!=n)
            print("write error!")
    }
    if(n<0)
        print("read error!");
    close(src);
    close(dst);
    exit(0);
}
```

2.17　用 Windows 的动态链接库编写复制文件的程序。

答：（1）程序如下。

```
dlltest.cpp
#include "windows.h"
BOOL APIENTRY DllMain(HANDLE hModule,
                      DWORD ul_reason_for_call,
                      LPVOID lpReserved)
{
    return TRUE;
}
extern "C" _declspec(dllexport) int MyCopyFile(LPCSTR src,LPCSTR tar)
{
    if(CopyFile(src,tar,FALSE)==TRUE)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

（2）测试程序如下。

```
#include "windows.h"
extern "C" _declspec (dllimport) int MyCopyFile(LPCSTR,LPCSTR);
int main(int argc, char * argv[])
{
    MyCopyFile("C:\\1.txt","C:\\2.txt");
    return 0;
}
```

2.18　打开一个手机 App，写出使用该 App 后的交互感受。

答：略。

第 3 章习题解答

3.1　有人说，一个进程是由伪处理机执行的一个程序。这话对吗？为什么？

答：对。因为伪处理机的概念只有在执行时才存在，它表示多个进程在单处理机上并发执行的一个调度单位。因此，尽管进程是动态概念，是程序的执行过程，但是，在多个进程并行执行时，仍然只有一个进程占据处理机执行，而其他并发进程则处于就绪或等待状态。这些并发进程就相当于由伪处理机执行的程序。

3.2　单 CPU 是否可以实现程序的并发执行以及并行执行？

答：并发执行在同一时刻只能有一条指令执行，但多个进程指令被快速轮换执行，在宏

观上具有多个进程同时执行的效果。并行执行在同一时刻，有多条指令在多个处理器上同时执行。单 CPU 可以实现程序的并发执行，但是不能实现程序的并行执行。

3.3 “程序的并发执行将导致最终结果失去封闭性”对所有的程序都成立吗？试举例说明。

答：并非对所有的程序均成立。例如，在下面的程序中，x 是内部变量，不可能被外部程序访问，因此这段程序的运行不会受外部环境影响。

```
begin
    local x
    x:=10
    print(x)
end
```

3.4 在 UNIX System Ⅴ中，系统程序所对应的正文段未被考虑成进程上下文的一部分，为什么？

答：因为系统程序的代码被用户程序共享，如果每个进程在保存进程上下文时都将系统程序代码放到其进程上下文中，则非常浪费资源，所以系统程序的代码不放在进程上下文中，而是统一放在核心程序所处的内存中。

3.5 进程控制块包含了很多结构。从进程、内存及文件等角度对这些结构进行分类。

答：进程相关的结构有进程名（进程标识号）、家族关系、进程当前状态、进程优先级、进程开始地址、各种计时信息、通信信息、程序计数器、调度参数、父进程、进程组。

内存相关的结构有占用内存大小及其管理用数据结构指针、交换区外存地址、交换程序模块长度、共享程序段长度及起始地址、缓冲区地址、缓冲区长度、正文段指针、数据段指针、堆栈段指针。

文件相关的结构有根目录、工作目录、文件描述符、用户标识号（用户 ID）、组标识号（组 ID）。

3.6 Linux 系统 init 进程的 PID 是多少？它在系统运行过程中起什么作用？

答：init 进程的 PID 是 1。init 进程是 Linux 系统内核启动的第一个用户进程，是其他所有用户进程的父进程。init 进程有两个主要的功能。

（1）管理孤立进程。当一个进程的父进程早于其终止，则该进程成为孤立进程，这时 init 进程就成为孤立进程的父进程。

（2）实现不同的运行级别。init 进程可以设置为不同的级别，运行不同的系统服务。

3.7 由于进程的创建开销较大，一般操作系统会使用 copy-on-write 技术提升创建进程的效率。简述该技术的实现原理。

答：当有多个调用者同时请求同一资源时，它们会获取指向同一资源的相同的指针，直到某个调用者试图修改资源的内容时，系统才会真正复制一份专用副本给该调用者，而其他调用者见到的最初的资源仍然保持不变。这个过程对其他的调用者都是透明的。这种技术主要的优点是，如果调用者没有修改该资源，就不会有副本被创建，因此多个调用者只在读取操作时可以共享同一资源。

3.8 什么是临界区？试举一个临界区的例子。

答：临界区是指不允许多个并发进程交叉执行的一段程序。它是由于不同并发进程的程序段共享公用数据或公用数据变量而引起的，所以它又被称为访问公用数据的程序。以下是一个临界区的例子。

```
getspace:
begin
    local g
    g=stack[top]
    top=top-1
end
release(ad):
begin
    top=top+1
    stack[top]=ad
end
```

3.9 原子操作与临界区的区别是什么？

答：原子操作是由处理器保证的不可分割的操作。对于简单的原子操作，处理器会提供单条指令以保证原子性；对于复杂的原子操作，通过添加自旋锁（spinlock）来实现执行过程中不被打断。

临界区是不允许多个并发进程交叉执行的一段程序。临界区的实现方式是：在临界区设置和检查访问标志。进程访问临界区时，通过访问标志判断临界区是否有其他进程在访问。如果有，则等待并循环检查；如果没有，则进入临界区。进程离开临界区时重置访问标志。

3.10 并发进程间的制约有哪两种？引起制约的原因是什么？

答：并发进程所受的制约有两种，分别是直接制约和间接制约。直接制约是由并发进程共享对方的私有资源引起的，间接制约是由竞争公有资源引起的。

3.11 什么是进程间的互斥？

答：进程间的互斥是指一组并发进程中的多个程序段因共享某一公有资源而导致它们必须以一个不允许交叉执行的单位执行，即不允许两个或多个共享该资源的并发进程同时进入临界区。

3.12 简述 P、V 原语法和加锁法实现进程间互斥的区别。

答：互斥的加锁实现原理如下。当某个进程进入临界区之后，它将锁上临界区，直到它退出临界区时为止。并发进程在申请进入临界区时，首先测试该临界区是否是上锁的。如果该临界区已被锁住，则并发进程要等到该临界区开锁之后才有可能获得临界区。

但是加锁法存在如下弊端：①循环测试锁定位将损耗较多的 CPU 计算时间；②产生不公平现象。

为此，P、V 原语法采用信号量管理相应临界区的公有资源，信号量的数值仅能由 P、V 原语操作改变，而 P、V 原语执行期间不允许中断发生。其过程是：当某个进程正在临界区内执行时，其他进程如果执行了 P 原语，则该进程并不像执行 lock 时那样因进不了临界区

而返回 lock 的起点，等以后重新执行测试，而是在等待队列中等待由其他进程执行 V 原语操作释放资源后再进入临界区，这时 P 原语才算真正结束。若有多个进程执行 P 原语操作而进入等待状态，一旦有进程执行 V 原语操作释放资源，则等待进程中的一个进入临界区，其余的进程继续等待。

总之，加锁法是采用反复测试 lock 实现互斥的，存在 CPU 浪费和不公平现象；而 P、V 原语使用了信号量，克服了加锁法的弊端。

3.13　在生产者-消费者问题中，设其缓冲部分由 *m* 个长度相等的有界缓冲区组成，每次传输数据长度等于有界缓冲区长度，生产者和消费者可对缓冲区同时操作。重新描述发送过程 deposit(data)和接收过程 remove(data)。

答：设第 i 块缓冲区的公用信号量为 mutex[i]，保证生产者进程和消费者进程对同一块缓冲区操作的互斥，初始值为 1。设信号量 avail 为生产者进程的私有信号量，初始值为 m；信号量 full 为消费者进程的私有信号量，初始值为 0。发送过程和接收过程如下。

```
deposit(data):
begin
    P(avail)
    选择空缓冲区 i
    P(mutex[i])
    送数据入缓冲区 i
    V(full)
    V(mutex[i])
end
remove(data):
begin
    P(full)
    选择满缓冲区 i
    P(mutex[i])
    取缓冲区 i 中的数据
    V(avail)
    V(mutex[i])
end
```

3.14　进程 PA、PB 通过两个先进先出缓冲区队列连接，如图 1.3.1 所示。每个缓冲区的长度等于传送消息长度。进程 PA、PB 之间的通信满足如下条件：

（1）至少有一个空缓冲区存在时，发送进程才能发送一个消息。

（2）当缓冲区队列中至少存在一个非空缓冲区时，接收进程才能接收一个消息。

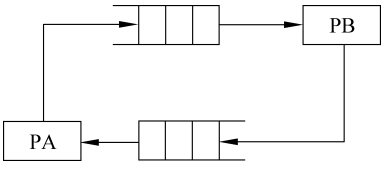

图 1.3.1　进程 PA、PB 通过两个先进先出缓冲区队列连接

试描述发送过程 send(i,m)和接收过程 receive(i,m)。这里,i 代表缓冲队列,m 代表消息。

答：定义数组 buf[0]、buf[1]、bufempty[0]和 buffull[1]是 PA 的私有信息量,buffull[0]、bufempty[1]是 PB 的私有信息量。

初始时：

bufempty[0]=bufempty[1]=n(n 为缓冲区队列中的缓冲区个数)

buffull[0]=buffull[1]=0

发送和接收过程如下：

```
send(i,m):
begin
    local x
    P(bufempty[i])
    按先进先出方式选择一个空缓冲区
    buf[i](x)
    buf[i](x)=m
    buf[i](x)置满标记
    V(buffull[i])
end
receive(i,m):
begin
    local x
    P(buffull[i])
    按先进先出方式选择一个装满数据的缓冲区
    m=buf[i](x)
    buf[i](x)置空标记
    V(bufempty[i])
end
```

PA 调用 send(0,m)和 receive(1,m)。

PB 调用 send(1,m)和 receive(0,m)。

3.15 进程间通信的方式有哪些？它们各有什么优缺点？

答：进程间通信可分为 4 种方式：主从方式、会话方式、消息或邮箱机制以及共享存储区方式。

(1) 主从方式通信系统的主要特点是：①主进程可自由地使用从进程的资源或数据；②从进程的动作受主进程的控制；③主进程和从进程的关系是固定的。

(2) 在会话方式中,通信双方可分别称为使用进程和服务进程。其中,使用进程调用服务进程提供的服务。它们具有如下特点：①使用进程在使用服务进程提供的服务之前,必须得到服务进程的许可；②服务进程根据使用进程的要求提供服务,但对服务的控制由服务进程自身完成；③使用进程和服务进程在进行通信时有固定连接关系。

(3) 消息或邮箱机制的特点是：①只要存在空缓冲区或邮箱,发送进程就可以发送消息；②与会话方式不同,发送进程和接收进程之间无直接连接关系,接收进程可能在收到某

个发送进程发来的消息之后，又转去接收另一个发送进程发来的消息；③发送进程和接收进程之间存在缓冲区或邮箱，用来存放被传送的消息。

(4) 共享存储区方式的特点是：不要求数据移动，两个需要交换信息的进程通过对同一共享数据区的操作来达到互相通信的目的。这个共享数据区是每个通信进程的组成部分。

3.16 每个进程有自己的进程地址空间。当两个进程利用消息缓冲机制进行通信时，操作系统如何将消息从发送进程地址空间缓冲区复制到接收进程地址空间缓冲区？

答：发送进程和接收进程采用消息缓冲机制进行数据传送时，发送进程在发送消息前，先在自己的地址空间设置一个发送区，把要发送的消息填入其中，然后再用发送过程将其发送出去；接收进程则在接收消息之前，在自己的地址空间设置相应的接收区，然后用接收过程接收消息。

3.17 编写一个程序，使用系统调用 fork 生成 3 个子进程，并使用系统调用 pipe 创建一条管道，使得这 3 个子进程和父进程共用同一管道进行通信。

答：

```
main()
{
    int r,p1,p2,p3,fd[2];                          /* fd[2]为管道文件读写标识 */
    char buf[50],s[5];
    pipe(fd);                                      /* 创建管道 */
    while((p1=fork())==-1);                        /* 创建子进程 1 */
    if(p1==0)                                      /* 在子进程 1 中执行 */
    {
        lockf(fd[1],1,0);                          /* 锁定写过程 */
        sprintf(buf,"child process P1 is sending message!\n");
        printf("child process P1!\n");
        write(fd[1],buf,50);                       /* 将 buf 中的数据写入管道 */
        sleep(5);                                  /* 睡眠,等待父进程读出 */
        lockf(fd[1],0,0);                          /* 解锁 */
        exit(0);
    }
    else
    {
        while((p2=fork())==-1);                    /* 创建子进程 2 */
        if(p2==0)                                  /* 在子进程 2 中执行 */
        {
            lockf(fd[1],1,0);                      /* 锁定写过程 */
            sprintf(buf, "child process P2 is sending message!\n");
            /* 数据写入 buf */
            printf("child process P2!\n");
            write(fd[1],buf,50);                   /* 将 buf 中的数据写入管道 */
            sleep(5);                              /* 同步等待父进程读 */
            lockf(fd[1],0,0);                      /* 解锁 */
```

```
            exit(0);                                /*释放进程资源*/
        }
        else
        {
            while((p3=fork())==-1);                 /*创建子进程3*/
            if(p3==0)                               /*在子进程3中执行*/
            {
                lock(fd[1],1,0);
                sprintf(buf,"child process P3 is sending message!\n");
                printf("child process P3!\n");
                write(fd[1],buf,50);
                sleep(5);
                lockf(fd[1],0,0);
                exit(0);
            }
            wait(0);                                /*父进程等待子进程先执行*/
            if(r=read(fd[0],s,50)==-1)              /*读管道中的内容到s中*/
                printf("can't read pipe\n");
            else
                printf("%s\n",s);
            wait(0);                                /*等待另一个子进程执行*/
            if(r=read(fd[0],s,50)==-1)
                printf("can't read pipe\n");
            else
                printf("%s\n",s);
            wait(0);                                /*等待最后一个子进程执行*/
            if(r=read(fd[0],s,50)==-1)
                printf("can't read pipe\n");
            else
                printf("%s\n",s);
            exit(0);
        }
    }
}
```

3.18 产生死锁有哪几个必要条件？为什么？

答：产生死锁的必要条件有以下4个。

（1）互斥。并发进程要求和占有的资源不能同时被两个或多个进程使用或操作，一个进程对它需要的资源进行排他性控制。

（2）不可剥夺。进程获得的资源在未使用完毕之前不能被其他进程强行剥夺，而只能由获得该资源的进程自己释放。

（3）部分分配。进程每次申请它所需要的一部分资源，在等待新资源的同时，继续占用已获得的资源。

（4）环路。存在一种进程循环链，链中每一个进程已获得的资源同时被下一个进程

请求。

3.19 设有5位哲学家，共享一张放有5把椅子的圆桌，每人分得一把椅子。但是，桌子上总共只有5支筷子，在两人之间各放一支。哲学家在肚子饥饿时才试图分两次从两边拾起筷子吃饭。

条件如下：

（1）只有拿到两支筷子时，哲学家才能吃饭。

（2）如果筷子已在他人手上，则哲学家必须等待到他人吃完之后才能拿到筷子。

（3）哲学家在自己未拿到两支筷子吃饭之前，决不放下自己手中的筷子。

解答以下问题：

（1）描述一个保证不会出现相邻的两位哲学家同时要求吃饭的通信算法。

（2）描述一个既没有相邻的两位哲学家同时吃饭，又没有人饿死（永远拿不到筷子）的算法。

（3）在什么情况下，5位哲学家全部吃不上饭？

答：（1）设信号量c[0]～c[4]，初始值均为1，分别表示第i号筷子被拿（i=0,1,2,3,4）。

```
send(i)：第 i 个哲学家要吃饭
begin
    P(c[i]);
    P(c[i+1 mod 5]);
    eat;
    V(c[i+1 mod 5]);
    V(c[i]);
end
```

该过程能保证相邻的两位哲学家不同时吃饭，但会出现5位哲学家一人拿一支筷子，谁也吃不上饭的死锁情况。

（2）解决的思路如下：让奇数号的哲学家先取右手边的筷子，让偶数号的哲学家先取左手边的筷子。这样，任何一个哲学家拿到一支筷子以后，就已经阻止了他邻座的一位哲学家吃饭的企图，除非某位哲学家一直吃下去，否则不会有人饿死。

```
send(i):
begin
    if i mod 2==0 then
    {
        P(c[i]);
        P(c[i+1] mod 5);
        eat;
        V(c[i]);
        V(c[i+1 mod 5]);
    }
    else
    {
        P(c[i+1 mod 5]);
```

```
            P(c[i]);
            eat;
            V(c[i+1 mod 5]);
            V(c[i]);
        }
    end
```

(3) 见(1)的解答。

3.20 银行家算法是经典的死锁避免算法，请简述该算法。

答：银行家算法的基本思想是，进程在进入系统时，必须申明在运行过程中可能需要每种资源类型的最大单元数目，其数目不应超过系统拥有的资源总量。当进程请求一组资源时，系统必须首先确定是否有足够的资源分配给该进程。若有，再进一步计算在将这些资源分配给该进程后是否会使系统处于不安全状态。如果不会，才将资源分配给它，否则让该进程等待。

3.21 什么是线程？简述线程与进程的区别。

答：线程是在进程内用于调度和占有处理机的基本单位，它由线程控制表、存储线程上下文的用户栈以及核心栈组成。线程可分为用户级线程、核心级线程以及混合型线程等类型。其中，用户级线程在用户态下执行，CPU 调度算法和各线程优先级都由用户设置，与操作系统内核无关；核心级线程的调度算法及线程优先级的控制权归操作系统内核所有。混合型线程的控制权则归用户和操作系统内核二者所有。

线程与进程的主要区别如下：

(1) 进程是资源管理的基本单位，它拥有自己的地址空间和各种资源，例如内存空间、外部设备等；线程只是处理机调度的基本单位，它只和其他线程一起共享进程资源，但自己没有任何资源。

(2) 以进程为单位进行处理机切换和调度时，由于涉及资源转移以及现场保护等问题，将导致处理机切换时间变长，资源利用率降低。以线程为单位进行处理机切换和调度时，由于不发生资源变化，特别是地址空间的变化，处理机切换的时间较短，从而处理机效率较高。

(3) 对用户来说，多线程可减少用户的等待时间，提高系统的响应速度。例如，当一个进程需要对两个不同的服务器进行远程过程调用时，对于无线程系统的操作系统来说，需要顺序等待两个不同调用返回结果后才能继续执行，且在等待中容易发生进程调度；对于多线程系统来说，则可以在同一进程中使用不同的线程同时进行远程过程调用，从而缩短进程的等待时间。

(4) 线程和进程一样，都有自己的状态，也有相应的同步机制。不过，由于线程没有单独的数据和程序空间，因此，线程不能像进程那样将数据与程序交换到外存，从而线程没有挂起状态。

(5) 进程的调度、同步等控制大多由操作系统内核完成；而线程的控制既可以由操作系统内核进行，也可以由用户进行。

3.22 使用库函数 clone()与 create_thread()在 Linux 环境下创建两种不同执行模式的线程程序。

答：Linux 系统支持用户级线程和核心级线程两种执行模式，其库函数分别为 pthread_create()和 clone()。创建用户级线程和核心级线程的程序示例如下。

(1) 用户级线程编程示例：

```
#include<pthread.h>
void *ptest(void *arg)
{
    printf("This is the new thread!\n");
    return(NULL);
}
main()
{
    pthread_t tid;
    printf("This is the parent process!\n");
    pthread_create(&tid,NULL,ptest,NULL);/*创建线程*/
    sleep(1);
    return;
}
```

该程序通过调用 pthread_create()创建一个用户级线程，其指针为 tid，过程名为 ptest。

(2) 核心级线程编程示例：

```
#include<signal.h>
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<linux/unistd.h>
#define STACKSIZE 16384
/*在退出时发送的信号掩码*/
#define CSIGNAL 0x000000ff
/*若进程间共享虚拟地址则置位*/
#define CLONE_VM 0x00000100
/*若进程间共享信息则置位*/
#define CLONE_FS 0x00000200
/*若进程间共享文件则置位*/
#define CLONE_FILES 0x00000400
/*若进程间共享信号句柄则置位*/
#define CLONE_SIGHAND 0x00000800
int show_same_vm;
void cloned_process_start_here(void *data)
{
    printf("child:\t got argument %d as fd\n", (int) data);
    show_same_vm=5;
    printf("child: \t vm=%d \n", show_same_vm);
```

```
    close((int)data);
}
int main()
{
    int fd, pid;
    fd=open (" /dev/null", O_RDWR);
    if(fd<0)
    {
        perror("/dev/null");
        exit(1);
    }
    printf("mother: \t vm=%d\n",fd);
    show_same_vm=10;
    printf("mother: \t vm=%d\n", show_same_vm);
    pid=clone(cloned_process_start_here, (void *) fd);
    if(pid<0)
    {
        perror ("start_thread");
        exit(1);
    }
    sleep(1);
    printf("mother: \t vm=%d\n", show_same_vm);
    if(write(fd, "c", 1)<0)
        printf("mother: \t child closed our file descriptor\n");
}

#include<signal.h>
#include<stdio.h>
#include<stdlib.h>
#include<fcntl.h>
#include<linux/unistd.h>
#define STACKSIZE 16384
/*在退出时发送的信号掩码*/
#define CSIGNAL 0x000000ff
/*若进程间共享虚拟地址则置位*/
#define CLONE_VM 0x00000100
/*若进程间共享信息则置位*/
#define CLONE_FS 0x00000200
/*若进程间共享文件则置位*/
#define CLONE_FILES 0x00000400
/*若进程间共享信号句柄则置位*/
#define CLONE_SIGHAND 0x00000800
int clone(void (*fn) (void *), void *data)          /*创建核心线程*/
{
    long retval,errno;
```

```
    void **newstack;
    /*
     * 为子进程分配新的栈空间
     */
    newstack=(void **) malloc (STACKSIZE);
    if(!newstack)
        return-1;
    /*
     * 为子进程函数设置栈,把(void *)参数压入栈中
     */
    newstack=(void **) (STACKSIZE+(char *) newstack);
    *newstack=data;

    _asm_volatile_(
        "int $ 0x80\n\t"              /* Linux/i386 系统调用 */
        "testl %0, %0\n\t"            /* 检查返回值 */
        "jne lf\n\t"                  /* 如果是父进程则跳转 */
        "call * %3\n\t"               /* 开始执行子线程函数 */
        "movl %2, %0\n\t"
        "int $ 0x80\n"                /* 退出系统调用: 退出子线程 */
        "l: \t"
        : "=a" (retval)
        : "0" (_NR_clone), "i" (_NR_exit),
        "r" (fn),
        "b" (CLONE_VM | CLONE_FS | CLONE_FILES |
        CLONE_SIGHAND | CLONE_SIGCHLD),
        "c" (newstack));
    if(retval<0)
    {
        errno=-retval;
        retval=-1;
    }
    return retval;
}
```

第 4 章习题解答

4.1 什么是分级调度？分时系统中有作业调度的概念吗？如果没有，为什么？

答：处理机调度问题实际上也是处理机的分配问题。显然只有必需的资源都已得到满足的进程才能享有竞争处理机的资格。这时它们处于内存就绪状态。这些必需的资源包括内存、外设及有关数据结构等。从而，在进程有资格竞争处理机之前，作业调度程序必须先调用存储管理和外设管理程序，并按一定的选择顺序和策略从输入井中选择几个处于后备状态的作业，为它们分配资源和创建进程，使它们获得竞争处理机的资格。另外，由于处于执行状态下的作业一般包括多个进程，而在单机系统中，每一时刻只能有一个进程占有处理

机，这样，在外存中，除了处于后备状态的作业外，还存在处于就绪状态而等待得到内存的作业。需要有一定的方法和策略为这部分作业分配空间。因此处理机调度需要分级。

一般来说，处理机调度可分为 4 级：

（1）作业调度，又称宏观调度或高级调度。

（2）交换调度，又称中级调度。其主要任务是按照给定的原则和策略将处于外存交换区中的就绪状态、等待状态或内存等待状态的进程交换到外存交换区。交换调度主要涉及内存管理与扩充，因此在有些书中也把它归入内存管理部分。

（3）进程调度，又称微观调度或低级调度。其主要任务是按照某种策略和方法选择一个处于就绪状态的进程占用处理机。在确立了占用处理机的进程之后，系统必须进行进程上下文切换，以建立与占用处理机的进程相适应的执行环境。

（4）线程调度。即进程中相关堆栈和控制表等的调度。

在分时系统中，一般不存在作业调度，而只有交换调度、进程调度和线程调度。这是因为在分时系统中，为了缩短响应时间，作业不是建立在外存中，而是直接建立在内存中。在分时系统中，一旦用户和系统的交互开始，用户马上要进行控制。因此，分时系统中没有作业提交状态和后备状态。分时系统的输入信息经过终端缓冲区为系统直接接收，或立即处理，或经交换调度暂存到外存中。

4.2　试比较作业、进程和程序的区别。

答：程序描述计算机要完成独立功能并在时间上按严格次序前后相继的计算机操作序列集合，是一个静态的概念。进程是并发执行的程序在执行过程中分配和管理资源的基本单位。作业可被看作用户向计算机提交任务的任务实体，例如一次计算、一个控制过程等。而进程则是计算机为了完成用户任务实体而设置的执行实体，是系统分配资源的基本单位。显然，计算机要完成一个任务实体，必须有一个以上的执行实体。也就是说，一个作业总是由一个以上的进程组成的。

4.3　作业的状态与进程的状态有什么区别与联系？

答：进程有初始状态、执行状态、等待状态、就绪状态和终止状态 5 种基本状态。当操作系统完成了创建进程的必要操作后，进程处于初始状态。进程已获得 CPU，其程序正在执行，则进程处于执行状态。进程已分配到除处理机以外的所有必要资源后，只要再获得处理机，便可立即执行，进程这时处于就绪状态。正在执行的进程由于发生某事件而暂时无法继续执行时，便放弃处理机，进程将处于等待状态。当一个进程执行结束，或出现了无法克服的错误，或被操作系统所终结，或被其他有终止权的进程所终结，它将进入终止状态。

作业有提交、收容、执行和完成 4 种基本状态。作业在从输入设备进入外部存储设备的过程中处于提交状态。作业的全部信息已进入输入井，在它还未被调度执行之前，该作业处于收容状态。作业从收容状态被选中到内存投入运行，则处于执行状态。当作业运行完毕，但它占用的资源尚未全部被系统回收时，该作业处于完成状态。

4.4　简述作业调度的主要功能。

答：作业调度的主要功能是按一定的原则对外存输入井中的大量后备作业进行选择，为选出的作业分配内存、输入输出设备等必要的资源，并建立相应的进程，使该作业的相关进程获得竞争处理机的资格。另外，当作业执行完毕时，作业调度还负责回收系统资源。