计算机技术基础实验指导书

郝兴伟　巩裕伟　刘毅　蔡晓军　主编

山东大学出版社

计算机技术基础实验指导书

主　编　郝兴伟　巩裕伟　刘　毅　蔡晓军

山东大学出版社

图书在版编目(CIP)数据

计算机技术基础实验指导书/郝兴伟等主编 .—济南:山东大学出版社,2003.8
ISBN 7-5607-2610-0

Ⅰ.计... Ⅱ.郝... Ⅲ.电子计算机—高等学校—教学参考资料 Ⅳ.TP3

中国版本图书馆 CIP 数据核字(2003)第 067579 号

山东大学出版社出版发行
(山东省济南市山大南路 27 号 邮政编码:250100)
山东省新华书店经销
济南新华印刷厂印刷
787×1092 毫米 1/16 12.75 印张 290 千字
2003 年 8 月第 1 版 2003 年 8 月第 1 次印刷
印数:1—5000 册
定价:18.50 元

前　言

随着计算机技术的快速发展和人们计算机应用水平的提高，高校学生对计算机课程提出了更高的要求。在这样的背景下，根据教育部计算机基础教学三个层次的要求，结合高校教学的实际情况，我们开始开设了针对非计算机专业的计算机技术基础课程。课程按照模块组织，不同的学科、专业可以按照自身的特点，选择不同的模块，在具体的讲解深度上也可以按照学生的实际情况进行。

为配合计算机技术基础的教学，我们编写了本实验指导书，该指导书在教材的基础上安排了大量的实验，特别是有关 C, C++, VC++, SQL, Java 等内容提供了详细的实验练习，其中也包含了课程内容的有益补充。

对于操作系统、数据结构以及各种编程工具，我们把实验分成实验环境和实验内容两大部分，实验内容又包括了若干个不同的实验题目，用于练习不同的知识点。对于每个实验，包括实验内容、简要分析、实验步骤、参考解答、提示等几个部分（注：各章节根据不同的情况有所不同）：

一、实验环境：介绍实验所需的环境，便于学生上机操作。

二、实验内容：每个实验的题目、要求和目的。

三、简要分析：对部分较难的实验题目作简单的问题分析，目的是让学生理解问题的解决方法，帮助和引导学生建立正确的解题思路。

四、实验步骤：对每个实验给出简单的实验步骤。

五、参考答案：给出实验步骤和参考程序。使学生在独立编程解决问题的同时，还可以通过参考程序拓展思路，这也是学习计算机软件的重要方法。

六、提示：对实验中的一些重点注意事项和难点问题给出简单的提示。

本书是在《计算机技术基础》（高等教育出版社，郝兴伟主编，2003）的基础上编写的，是《计算机技术基础》一书的辅助教材。在编写过程中我们得到了很多一线教师的帮助，同时就教学内容和实验要求也征求了许多同学的意见，对他们的工作表示忠心的感谢。另外还要特别感谢山东大学网络中心主任、山东大学计算机学院副院长、博士生导师王海洋教授，山东大学教务处副处长龙世立教授，他们在课程的设置及教学研究中，提出了非常好的指导性意见。

由于作者水平有限，加之时间仓促，本书可能存在不足和纰漏，望广大读者批评指正。作者 E-mail 地址：hxw@sdu.edu.cn.

编　者

2003 年春于山东大学

目 录

第1章　操作系统

1.1　操作系统实验环境

操作系统实验要求在 linux 系统环境中进行，linux 系统版本不限。

1.2　实验题

1.2.1　操作系统接口实验

1. 实验目的

练习最基本的 shells 文件操作命令，熟悉操作系统接口经典服务方式。体验经典方式的服务效率。

2. 实验内容

练习 linux 中的系统注册命令，在成功注册进入系统后：打开一终端命令行窗体，利用文本文件编辑命令 vi 在你注册目录中建立以下名为 pctrl.c 的 C 语言程序：

```
/*************************************
main.c  -  description
-------
begin                    : 一 4 月  4 21:01:11 CST 2003
copyright                : (C) 2003 by 张鸿烈
Function                 : linux 系统中进程树的建立、控制、和父子进程同步
*************************************/
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]) //进程的启动入口
{
   int pid;  //存放进程号
   int status;  //存放子进程状态
```

```
    char * args[] = {"/bin/ls","-l",NULL};//子进程要执行的命令
    //父进程开始工作
    printf("Hello1, world! \n");
    pid = fork();//建立子进程系统调用
    if(pid == -1){//父进程创建失败,立即退出
      printf("Create Process fail! \n");
        exit(1);
    }
    else if(pid > 0){//父进程代码段
     wait(&status); //父进程等待子进程终止,在此睡眠
     //子进程终止后唤醒父进程,父进程报告自己的进程号
     printf("Parent PID = %d\n",getpid()); //取进程号系统调用
     exit(0);//父进程终止退出
    }
   else {  //子进程代码段
       //子进程报告自己的进程号等待用户敲回车键后执行新的命令
     printf("Child PID = %d\nPress Enter to Running %s\n",getpid(),args[0]);
  getchar();
    //执行新进程系统调用
     execve(args[0],args,NULL);//子进程开始执行 args 中准备的命令
     exit(0);//子进程终止退出
    }
    return 0;
}
```

然后完成以下操作:

① 使用 mkfs 命令格式化一张 linux 系统软盘上,并将其安装到系统上。

② 进入安装的软盘目录在软盘上使用 mkdir 命令建立子目录。

③ 将以上建立的 C 程序文件使用 cp 命令复制到在软盘上建立的子目录中。

④ 使用 grep 命令找出并记录 C 程序中含有"int"的行。

⑤ 使用 df 命令报告并记录软盘文件系统的统计信息,使用 ls 命令报告并记录软盘文件的统计信息。

⑥ 将建立在注册目录中的文件使用 rm 命令删除,并将软盘从系统中卸载。

3. 实验要求

记录每步实验中出现的结果,分析结果,如果有错误则排除错误;如果正确则说明该结果反映了哪些操作系统接口的功能,将怎样利用这些功能,将实验 1 中取得的程序信息、文件信息、文件系统信息进行总结分析。分析和比较 shell 命令方式的系统接口和可视化系统接口的优劣。将分析结果写成实验报告。

1.2.2　进程控制实验

1. 实验目的

通过进程控制有关的系统调用的编程和调试，加深对于进程概念的理解，体验进程创建和撤销的过程和进程运行的特征，掌握进程控制的方法。

2. 实验内容

(1) 在成功注册进入系统后，打开一终端命令行窗体，利用实验 1 建立的 pctrl.c 程序(请注意阅读一下程序中的注释，该程序反映了操作系统中进程树的一些创建和进程控制功能)在命令行输入 GNU 的 C 编译命令：

```
$ gcc pctrl.c -o pctrl
$
```

(2) 如果有错，请用 vi 按出错提示重新修改程序。如果没有报错，则说明编译成功，已经生成了 pctrl.c 的可执行程序 pctrl。

(3) 在命令行输入：

```
$ ./pctrl
Hello1, world!
Child PID = 2852
Press Enter to Running /bin/ls
```

注意：Child PID 每次运行是不同的。

根据 pctrl.c 的程序，说明程序此时的输出反映了操作系统中那些进程控制的功能。请记录下以上输出中的 Child PID 号。

(4) 按回车键，将会看到：

```
total 64
-rwxrwxr-x          1 student student        12167 Mar 29 15:18 pctrl
-rwxr-xr-x          1 student student         1142 Mar 29 15:18 pctrl.c
Parent PID = 2851
$
```

请根据 pctrl.c 的程序，进一步说明程序此时的输出反映了操作系统中那些进程控制的功能。并记录下以上程序输出中的 Parent PID 号。

(5) 再次使用 vi 将 pctrl.c 程序中 char *args[]={"/bin/ls","-l",NULL}; 一行改为 char *args[]={"/bin/ps","-l",NULL}; 重复本节(1)～(3)步，当显示：

```
Hello1, world!
Child PID = 3034
Press Enter to Running /bin/ps
```

注意,请先不要按回车键,而是先打开另一命令终端命令行窗口在其中输入命令:

```
$ ps -H aux
USER    PID %CPU %MEM    VSZ   RSS    TTY      STAT START   TIME COMMAND
......
student  1626  0.0  0.5   4596 1344 pts/1    S    12:51   0:00 bash
student  3036  0.0  0.2   2636 724 pts/1     R    16:02   0:00 ps -H aux
student  1915  0.0  0.5   4592 1452 pts/0    S    13:07   0:00 bash
student  3034  0.0  0.1   1312 264 pts/0     S    16:02   0:00 ./pctrl
student  3035  0.0  0.1   1316 284 pts/0     S    16:02   0:00 ./pctrl
....
```

这是一个显示进程信息的命令,将会看到你的 pctrl 程序在当前系统中生成的两个进程列表信息。以上列表各列信息的含义如表 1.1 所示。

表 1.1 进程列表信息

符号	含义	符号	含义
USER	代表进程拥有者	TTY	所在终端号
PID	进程号	STAT	进程状态(R 执行,S 睡眠)
%CPU	进程自上次切换已来占用 CPU 时间的百分比	START	开始时间
%MEM	进程占内存总量的百分比	TIME	执行时间
VSZ	虚存的占有量	COMMAND	命令名
RSS	实存的占有量		

请记录您当前两个./pctrl 进程的状态。

(6) 返回启动程序的窗口按回车键,这次子进程应该执行新的命令,它也是一条 ps 命令但有不同的选项,将会看到:

```
F S   UID   PID  PPID  C PRI  NI ADDR SZ  WCHAN  TTY      TIME CMD
000 S 500  1915  1568  0  76   0 -    1148 wait4  pts/0    00:00:00 bash
000 S 500  3034  1915  0  76   0 -    328  wait4  pts/0    00:00:00 pctrl
000 R 500  3036  3185  1  75   0 -    772  -      pts/0    00:00:00 ps
Parent PID = 3035
$
```

请将(5)和(6)输出中 pctrl 的有关信息对比一下,它们说明了什么?

(7) 返回第(5)步,这次你在另一窗口中发命令:

```
$ kill <你记录的 Parent PID 或 Child PID>
```

将会看到什么情况?记录并说明这些情况又反映了操作系统中那些进程控制的功能。

(8) 请将 pctrl.c 程序中的 wait(&status);一行前面加"//"注释掉,再重复(1)~(6)步记录并分析一下新的情况,说明进程的并发发生了什么变化?

3. 实验要求

根据实验中观察和记录的信息结合 pctrl.c 程序,说明它们反映出操作系统教材中进程

及处理机管理一节讲解的进程的哪些特征和功能？在真实的操作系统中它是怎样实现和反映出教材中讲解的进程的生命期、进程的实体和进程状态控制的。你对于进程概念和并发概念有哪些新的理解和认识？根据以上实验结果和分析写出实验报告。

1.2.3　进程同步实验

1. 实验目的

加深对进程同步的理解，体验进程同步机制的功能，练习进程同步工具的使用方法。体验经典进程同步问题的解决方法。

2. 实验内容

(1) 共享内存(SYSTEM-V-UNIX-IPC 的第一种)是 OS 内核为并发进程间交换数据而提供的一块内存区(段)。如果段的权限设置恰当，每个要访问该段内存的进程都可以把它映射到自己私有的地址空间中。如果一进程更新了段中数据，那么其他进程立即会看到这一更新。进程创建的段也可由另一进程读写。

linux 中可用命令 ipcs －m 观察共享内存情况。

```
$ ipcs -m

------ Shared Memory Segments --------
key        shmid      owner      perms      bytes      nattch     status
0x00000000 327682     student    600        393216     2          dest
0x00000000 360451     student    600        196608     2          dest
0x00000000 393220     student    600        196608     2          dest
```

其中：key 为共享内存关键值。shmid 为共享内存标识；owner 为共享内存所由者(本例为 student)；perm 为共享内存使用权限(本例为 student 可读可写)；byte 为共享内存字节数；nattch 为共享内存使用计数；status 为共享内存状态。

上例说明系统当前中已建立了一些共享内存。

(2) 信号灯(SYSTEM_V_UNIX_IPC 的第二种)是 OS 内核控制并发进程间共享资源的一种进程同步机制。linux 中可用命令 ipcs －s 观察信号灯的情况。

```
$ ipcs -s

------ Semaphore Arrays --------
key        semid      owner      perms      nsems      status
```

其中 semid 为信号灯的标识号；nsems 为信号灯的个数，其他字段意义同以上共享内存所述。

上例说明当前系统中还没有建立信号灯。

(3) 为了实验教材中生产者/消费者问题的同步问题，在你注册目录中编写以下程序，建立一个共享内存作为该问题中的有界缓冲。

```
/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
main.c - description
- - - - - - - - - - -
begin          : 一 4 月  4 21:01:11 CST 2003
copyright      : (C) 2003 by 张鸿烈
Function       : 建立一段共享内存 作为有界数组
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
#include <sys/sem.h>

#define BUFSZ 4 //缓冲区长度

int main(int argc, char * argv[])
{
   int shmid;  //共享内存标识

   // shmget 为获取共享内存函数;其中参数的意义为:
   // IPC_PRIVATE    共享内存中默认的 key
   // BUFSZ    共享内存长度
   // 0666    共享内存权限的 8 进制数(均为可读可写)

   if((shmid = shmget(IPC_PRIVATE,BUFSZ,0666)) <0){
     perror("shmget");//没有取得报错返回
     exit(1);
   }

   // 成功
}
```

(4) 使用以下 GNU 的 C 命令编译、执行程序:

```
$ gcc shmem.c -o shmem
$ ./shmem
```

(5) 查看新生成的共享内存:

```
$ ipcs -m
```

```
------Shared Memory Segments --------
key         shmid      owner      perms      bytes      nattch     status
....

0x00000000 458758     student    600        393216     2          dest
0x00000000 3538967    student    666        4          0
```

可以看到新建的共享内存的长度为 4 字节，其标识 shmid 号为 3538967。记下这个数字。

(6) 再建立用于同步的信号灯

```
/**************************
main.c - description
-----
begin                : 一 4 月  4 21:01:11 CST 2003
copyright            : (C) 2003 by 张鸿烈
Function             : 建立同步信号灯
**************************/
#ifdef HAVE_CONFIG_H
#include <config.h>
#endif
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
#include <sys/sem.h>
int main(int argc, char *argv[])
{
   int full; //生产者同步信号灯标识
   int empty; //消费者同步信号灯标识
   int mutex; //互斥用信号灯标识
   int nsems=1; //创建的信号灯集个数
   int flag = 0666; //信号灯使用权限(均为可读可写)
   // semget 创建信号灯函数,其参数的含义如下:
   // IPC_PRIVATE 信号灯的键值 key
   // nsems 信号灯集个数
   // flag 信号灯使用权限
   full = semget(IPC_PRIVATE, nsems, flag);
   if(full<0){ //信号灯创建不成功,报错退出
      perror("full error\n");
      return EXIT_FAILURE;
```

```
    }
    // 信号灯创建成功,报告标识号
    printf("Full semaphore semid= %d\n",full);
    empty = semget(IPC_PRIVATE,nsems,flag);
    if(empty<0){ //信号灯创建不成功,报错退出
      perror("empty error\n");
      return EXIT_FAILURE;
    }
    // 信号灯创建成功,报告标识号
    printf("Empty semaphore semid= %d\n",empty);
    mutex = semget(IPC_PRIVATE,nsems,flag);
    if(mutex<0){ //信号灯创建不成功,报错退出
      perror("mutex error\n");
      return EXIT_FAILURE;
    }
    // 信号灯创建成功,报告标识号
    printf("Mutex semaphore semid= %d\n",mutex);
    return EXIT_SUCCESS;
}
```

(7) 将以上程序存盘,使用以下命令编译、执行程序:

```
$ gcc sema.c -o sema
$ ./sema
Full semaphore semid= 0
Empty semaphore semid= 32769
Mutex semaphore semid= 65538
```

(8) 查看新生成的信号灯,注意记下每个信号灯的 semid:

```
$ ipcs -s

------ Semaphore Arrays --------
key        semid     owner      perms      nsems     status
0x00000000 0         student    666        1
0x00000000 32769     student    666        1
0x00000000 65538     student    666        1
```

(9) 建立生产者进程程序:

```
/**********************************
main.c - description
-----------
begin                : 一 4月  4 21:01:11 CST 2003
copyright            : (C) 2003 by 张鸿烈
Function             : 建立生产者进程
```

```
**************************************************************/

#ifdef HAVE_CONFIG_H
#include <config.h>
#endif

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>
#include <sys/sem.h>

#define BUFSZ 4 //缓冲区长度

// 以下 P(int semid)中的语句是 UNIX SYSTEM V IPC 中信号灯的 wait()操作的一种实现,
// 由于 UNIX SYSTEM V IPC 的信号灯功能很多,在此不便详述。

int P(int semid) // semid 是信号灯标识
{
    struct sembuf buf;
    int nsems = 1; // 信号灯数为 1 个

    buf.sem_num = 0; // 信号灯初值为 0
    buf.sem_op = -1; // 对信号灯值减 1
    buf.sem_flg = SEM_UNDO;

    // semop ()函数按给定的初值完成信号灯操作
    if((semop(semid, &buf, nsems)) <0) {
        printf("wait error \n");
        return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}

// 以下 V(int semid)中的语句是 UNIX SYSTEM V IPC 中信号灯的 signal()操作的一种实现,
// 由于 UNIX SYSTEM V IPC 的信号灯功能很多,在此不便详述。

int V(int semid) //semid 是信号灯标识
{
    struct sembuf buf;
    int nsems = 1; // 信号灯数为 1 个
```

```
    buf.sem_num = 0; // 信号灯初值为 0
    buf.sem_op = 1; // 对信号灯值加 1
    buf.sem_flg = SEM_UNDO;

    // semop ()函数按给定的初值完成信号灯操作
    if((semop(semid, &buf, nsems)) <0) {
      printf("signal error ");
      return EXIT_FAILURE;
    }
    return EXIT_SUCCESS;
}

// 生产者进程主程序
int main(int argc, char * argv[])
{
    int shmid; // 共享内存标识,可从它获取到共享内存区
    int fullid; // 同步生产者信号灯标识
    int emptyid; // 同步消费者信号灯标识
    int mutexid; // 互斥使用读写指针标识

    char * shmbuf; // 共享内存区指针,它是有界缓冲区的首地址
    int in=0; // 共享内存区写指针偏量
    // 如果命令行输入没给出共享内存标识、信号灯标识,提示命令用法并退出
    if(argc ! = 5){
      puts("USAGE: prod <shmid> <fullid> <emptyid> <mutex>");
      exit(EXIT_FAILURE);
    }
    shmid = atoi(argv[1]); // 共享内存标识由命令行第一参数取得
      // 通过 shmat(shmid,0,0)函数获得共享内存区指针,放在 shmbuf 中
    if((shmbuf=shmat(shmid,0,0)) < (char * ) 0){
      perror("shmat1 \ n");
      exit(1);
    }
      fullid=atoi(argv[2]); // 生产者信号灯标识,由命令行第二参数取得
    emptyid=atoi(argv[3]);// 消费者信号灯标识,由命令行第三参数取得
    mutexid=atoi(argv[4]);//互斥使用读写指针标识,由命令行第四参数取得
    V(fullid);      //置生产者信号灯初值=4
    V(fullid);
    V(fullid);
    V(fullid);
    V(mutexid);      //置互斥信号灯初值=1
 Dwhile(1){  //生产者循环不息的生产产品
```