# 操作系统实验报告

实验名称： Spooling实验

学院： 计算机学院

学号： 2016210864

姓名： 张翠翠

日期：2018年12月28日星期五

**实验目的**

体会操作系统中Spooling假脱机输入输出的过程，以及缓冲区，输入井、输出井，输出缓冲区之间的关系，并管理程序的作用。

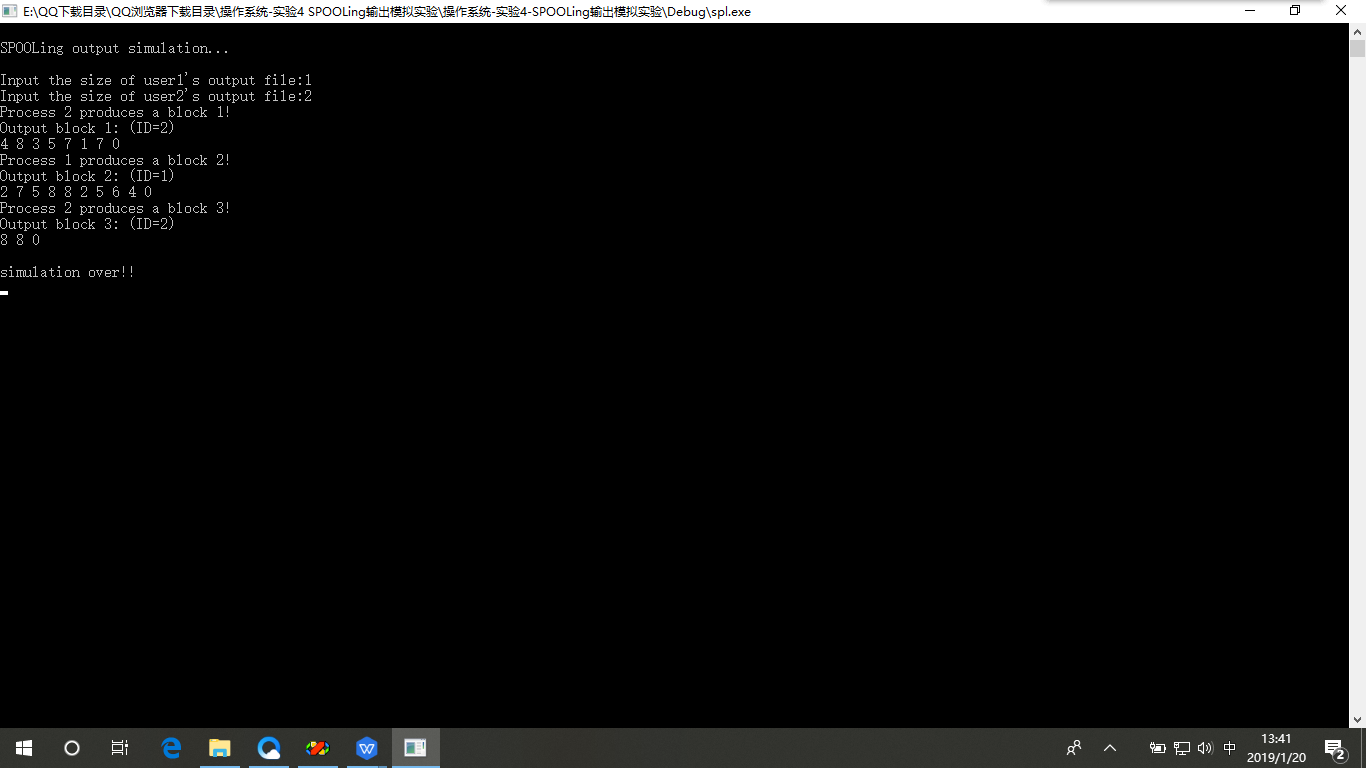
**实验原理**

Spooling技术实质上是利用一个共享设备的一个存储区，并不是真正占有这一设备，用户进程把要完成的任务以及文件的形式存入存储区，在存储区中排队并等待SPOOLING系统调度，只有被SPOOLING系统调度并输入，此项任务才真正完成，通过SPOOLING技术可以使独占设备成为共享设备，由此大大提高了设备的使用率，节约了硬件资源。

Spooling系统主要有以下三部分组成：

1. 输入井和输出井。这是在磁盘上开辟的两个大存储空间。输入井是模拟脱机输入时的磁盘设备，用于暂存I∕O设备输入的数据；输出井是模拟脱机输出时磁盘，用于暂存用户程序的输出数据。   (2)输入缓冲区和输出缓冲区。为了缓和CPU和磁盘之间速度不匹配的矛盾，在内存中要开辟两个缓冲区；输入缓冲区和输出缓冲区。输入缓冲区用于暂存由输入设备送来的数据，以后再传送到输入井。输出缓冲区用于暂存从输出井送来的数据，以后在传送给输出设备。   (3)输入进程SPi和输出进程SP0。这里利用两个进程来模拟脱机I∕O时的外围控制机。其中，进程SPi模拟脱机输入时的外围控制机，将用户要求的数据从输入机通过输入缓冲区再送到输入井，当CPU需要输入数据时，直接从输入井读入内存；进程SP0模拟脱机输出时的外围控制机，把用户要求输出的数据先从内存送到输出井，待输出设备空闲时，再将输出井中的数据经过输出缓冲区送到输出设备上

**实验结果**



**源代码**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <time.h>

#include <iostream.h>

#include <conio.h>

struct PCB

{

long ID;

int status;

long po;

long head;

long count;

long wait[1000];

}PCB[4];

struct ReqBlock

{ long ID;

long len;

long head;

}ReqBlock[128];

struct well

{

long num[1000];

long open;

long closed;

}well[3];

long K[3];

long n;

long r;

long k;

long a;

long b;

long i;

void input()

{ printf("Input the size of user1's output file:");

scanf("%ld",&K[1]);

printf("Input the size of user2's output file:");

scanf("%ld",&K[2]);

}

void init()

{

memset(PCB,0,sizeof(PCB));

PCB[1].ID=1;

PCB[2].ID=2;

PCB[3].ID=3;

PCB[3].status=2;

memset(well,0,sizeof(well));

n=0;

}

void UserServer(long kx)

{ a=rand()%10;

++PCB[kx].po;

PCB[kx].wait[PCB[kx].po]=a;

if (a==0)

{ well[kx].open=well[kx].closed+1;

for (i=1;i<=PCB[kx].po;i++)

well[kx].num[++well[kx].closed]=PCB[kx].wait[i];

PCB[kx].po=0;

PCB[kx].count++;

if (PCB[kx].count==K[kx])

PCB[kx].status=3;

if (PCB[3].status==2)

PCB[3].status=0;

if (well[kx].closed==1000 && PCB[kx].status==0)

PCB[kx].status=1;

n++;

ReqBlock[n].ID=kx;

ReqBlock[n].head=well[kx].open;

ReqBlock[n].len=well[kx].closed-well[kx].open+1;

printf("Process %ld produces a block %ld!\n",kx,n);

}

}

void SpoolingServer()

{ PCB[3].po++;

b=PCB[3].po;

printf("Output block %ld: (ID=%ld)\n",b,ReqBlock[b].ID);

for (i=1;i<=ReqBlock[b].len;i++)

{ printf("%ld ",well[ReqBlock[b].ID].num[i+ReqBlock[b].head-1]);

}

printf("\n");

if (PCB[3].po==n)

{ PCB[3].status=2;

if (PCB[1].status==3 && PCB[2].status==3)

PCB[3].status=3;

}

}

void work()

{ while (PCB[1].status!=3 || PCB[2].status!=3 || PCB[3].status!=3)

{ r=rand()%100+1;

if (r<=45)

k=1;

else if (r<=90)

k=2;

else

k=3;

if (PCB[k].status!=0)

continue;

switch (k)

{

case 1: UserServer(1); break;

case 2: UserServer(2); break;

case 3: SpoolingServer(); break;

}

}

}

void main()

{

printf("\nSPOOLing output simulation...\n\n");

srand(time(NULL));

input();

init();

work();

printf("\nsimulation over!!\n");

getch();

}