# 操作系统实验一：进程管理（含成功运行C语言源代码）

原创

南小山 已于 2022-04-24 16:27:43 修改 16762 收藏 577
分类专栏： 操作系统 文章标签： 操作系统 c语言 c++
于 2020-12-01 11:10:01 首次发布




目录

**（本文知识点较多，如时间较多可以详细看看第3章的知识点；如时间不多可直接点上方目录，直接看第4部分代码实现来理解）**

---

# 操作系统实验一：进程管理

---

## 1.实验目的

1.理解进程的概念，明确进程和程序的区别

2.理解并发执行的实质

3.掌握进程的创建、睡眠、撤销等进程控制方法

---

## 2.实验内容

用C语言编写程序，模拟实现创建新的进程；查看运行进程；换出某个进程；杀死运行进程等功能。

---

## 3.实验准备

以下将分别介绍

①**进程的概念**，以及进程的各类状态（**就绪状态、执行状态、阻塞状态**）；

②**进程控制块PCB 的作用及内容信息**

**③进程的创建与撤销 (□重点)**

**④进程的阻塞与唤醒(□重点)**

---

### 3.1.1进程的含义

进程是程序在一个数据集上的运行过程，是系统资源分配和调度的一个独立单位。一个程序在不同数据集上运行，乃至一个程序在同样数据集上的多次运行都是不同的进程。

### 3.1.2进程的状态

通常情况下，一个进程必须具有**就绪**、**执行**和**阻塞**三种基本状态。

（1）就绪状态

当进程已分配到除处理器（CPU）以外的所有必要资源后，只要再获得处理器就可以立即执行，此时进程的状态称为就绪状态。

在一个系统里，可以有多个进程同时处于就绪状态，通常把这些就绪进程排成一个或多个队列，称为就绪队列。

（2）执行状态

处于就绪状态的进程一旦获得了处理器（分配有处理器资源），就可以运行，进程状态也就处于执行状态。
在**单处理器**系统中，只能有**一个**进程处于执行状态，在**多处理器**系统中，则可能有**多个**进程处于执行状态。

（3）阻塞状态

正在执行的进程因为发生某些事件(如请求输入输出、申请额外空间等)而暂停运行，这种**受阻暂停**的状态称为阻塞状态，也可以称为等待状态。通常将处于阻塞状态的进程排成一个队列，称为阻塞队列。在有些系统中，也会按阻塞原因的不同将处于阻塞状态的进程排成多个队列。

**拓展**

除了进程的3种基本状态外，在很多系统为了更好地描述进程的状态变化，又增加了两种状态。

***I新状态***

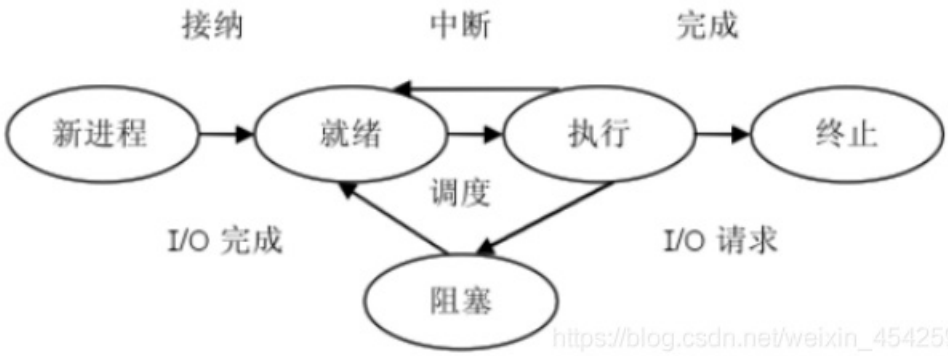
当一个新进程刚刚建立，还未将其放入就绪队列的状态，称为*新状态*。（例如一个，人刚开始接受教育，此时就可以称其处于新状态）

***II终止状态***

当一个进程已经正常结束或异常结束，操作系统已将其从系统队列中移出,但尚未撤消，这时称为*终止状态*。

---

### 3.1.3进程状态之间的转换





---

## 3.2 进程控制块PCB

### 3.2.1进程控制块的作用

进程控制块是构成进程实体的重要组成部分，是操作系统中最重要的记录型数据,在进程控制块PCB中记录了操作系统所需要的、用于描述进程情况及控制进程运行所需要的全部信息。通过PCB，能够使得原来不能独立运行的程序(数据)，成为一个可以独立运行的基本单位，一个能够并发执行的进程。换句话说，在进程的整个生命周期中，操作系统都要通过进程的PCB来对并发执行的进程进行管理和控制，进程控制块是系统对进程控制采用的数据结构，系统是根据进程的PCB而感知进程是否存在。所以，进程控制块是进程存在的**唯一标志**。当系统创建一个新进程时，就要为它建立一个PCB;进程结束时，系统又回收其PCB,进程也随之消亡。

### 3.2.2进程控制块的内容

进程控制块主要包括以下四个方面的内容：

（1）进程标识信息

----进程标识符用于标识一个进程，通常又分外部标识符和内部标识符两种。

（2）说明信息

----说明信息是有关进程状态等一些与进程调度有关的信息，它包括:①进程状态 ②进程优先权 ③与进程调度所需的其他信息 ④阻塞事件

（3）现场信息（处理器状态信息）

----现场信息是用于保留进程存放在处理器中的各种信息。主要由处理器内的各个寄存器的内容组成。**尤其是当执行中的进程暂停时，这些寄存器内的信息将被保存在PCB里，当该进程获得重新执行时，能从上次停止的地方继续执行。**

码字不易，转载请注明原文链接：操作系统实验一：进程管理（含成功运行C语言源代码）_南小山的博客-CSDN博客_操作系统进程管理实验c语言

（4）管理信息（进程控制信息）

进程控制信息主要分四方面：

| | |
|---|---|
| **程序和数据的地址** | 它是指该进程的程序和数据所在的主存和外存地址再次执行时，能够找到程序和数据 |
| **进程同步和通信机制** | 它是指实现进程同步和进程通信时所采用的机制、指针、信号量等 |
| **资源清单** | 该清单中存放有除了CPU以外，进程所需的全部资源和已经分配到的资源 |
| **链接指针** | 它将指向该进程所在队列的下一个进程的PCB的首地址 |

---

### 3.2.3进程控制块（PCB）的组织形式

在一个系统中，通常拥有数十个、数百个乃至数千个PCB，为了能对它们进行有效的管理,就必须通过适当的方式将它们组织起来，日前常用的组织方式有链接方式和索引方式两种。.

**（1)链接方式**

把具有相同状态的PCB，用链接指针链接成队列，如就绪队列、阻塞队列和空闲队列等。就绪队列中的PCB将按照相应的进程调度算法进行排序。而阻塞队列也可以根据阻塞原因的不同，将处于阻塞状态的进程的PCB,排成等待I/O队列、等待主存队列等多个队列。此外，系统主存的PCB区中空闲的空间将排成空闲队列，以方便进行PCB的分配与回收。

**（2)索引方式**

系统根据各个进程的状态，建立不同索引表，例如就绪索引表、阻塞索引表等。并把各个索引表在主存的首地址记录在主存中的专用单元里,也可以称为表指针。在每个索引表的表目中，记录着具有相同状态的各个PCB在表中的地址。

---

### 3.2.4进程控制原语

原语是指具有特定功能的**不可被中断**的过程。它主要用于实现操作系统的一 些专门控制操作。用于进程控制的原语有:

| 原语 | 作用 |
|---|---|
| **创建**原语 | 用于为一个进程分配工作区和建立PCB，该进程为就绪状态 |

| 原语 | 作用 |
| --- | --- |
| **撤销**原语 | 用于一个进程工作完后，收回它的工作区和PCB |
| **阻塞**原语 | 用于进程在运行过程中发生等待事件时，把进程的状态改为等待态 |
| **唤醒**原语 | 用于当进程等待的事件结束时，把进程的状态改为就绪态 |

---

### 3.3进程的创建与撤销 *重点

#### 3.3.1进程的创建

一旦操作系统发现了要求创建进程的事件后,便调用进程创建原按下列步骤创建一个新进程。

①为新进程分配惟一的进程标识符， 并从PCB队列中申请一个空闲PCB。

②为新进程的程序和数据，以及用户栈分配相应的主存空间及其他必要分配资源。

③初始化PCB中的相应信息，如标识信息、处理器信息、进程控制信息等。

④如果就绪队列可以接纳新进程，便将新进程加入到就绪队列中。

#### 3.3.2进程的撤销

一旦操作系统发现了要求终止进程的事件后,便调用进程终止原语按下列步骤终止指定的进程。

①根据被终止进程的标识符，从PCB集合中检索该进程的PCB,读出进程状态。

②若该进程处于执行状态，则立即终止该进程的执行。

③若该进程有子孙进程，还要将其子孙进程终止。

④将该进程所占用的资源回收，归还给其父进程或操作系统。

⑤将被终止进程的PCB从所在队列中移出，并撤销该进程的PCB。

---

### 3.4进程的阻塞与唤醒

#### 3.4.1进程的阻塞

一旦操作系统发现了要求阻塞进程的事件后，便调用进程阻塞原语，按下列步骤阻塞指定的进程。

①立即停止执行该进程。

②修改进程控制块中的相关信息。把进程控制块中的运行状态由“执行”状态改为“阻塞”状态，并填入等待的原因，以及进程的各种状态信息。

③把进程控制块插入到阻塞队列。根据阻塞队列的组织方式插入阻塞队列中。

④待调度程序重新调度，运行就绪队列中的其他进程。

#### 3.4.2进程的唤醒

一旦操作系统发现了要求唤醒进程的事件后，便调用进程唤醒原语,按下列步骤唤醒指定的进程。

①从阻塞队列中找到该进程。

②修改该进程控制块的相关内容。把阻塞状态改为就绪状态，删除等待原因等。

③把进程控制块插入到就绪队列中。

④按照就绪队列的组织方式，把被唤醒的进程的进程控制块插入到就绪队列中。

---

## 4.代码实现

**可成功运行代码如下**

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct jincheng_type{   //进程状态定义
 int pid;           //进程
 int youxian;     //进程优先级
 int daxiao;      //进程大小
 int zhuangtai;   //标志进程状态，0为不在内存，1为在内存，3为挂起
 char info[10];   //进程内容
};
struct jincheng_type neicun[20];
int shumu=0,guaqi=0,pid,flag=0;

//创建进程
void create(){
 if(shumu>=20) printf("\n内存已满，请先换出或结束进程\n");   //内存容量大小设置为20
 else{
  int i;
  printf("**当前默认一次性创建5个进程，内存容量20**");
  for(i=0;i<5;i++)  {               //默认一次创建5个进程
  //定位，找到可以还未创建的进程
  if(neicun[i].zhuangtai==1) break;  //如果找到的进程在内存则结束 ，初始设置都不在内存中（main函数中设置状态为0）
  printf("\n请输入新进程pid\n");
  scanf("%d",&(neicun[i].pid));
  for(int j=0;j<i;j++)
   if(neicun[i].pid==neicun[j].pid){
    printf("\n该进程已存在\n");
    return;
   }
  printf("请输入新进程优先级\n");
  scanf("%d",&(neicun[i].youxian));
  printf("请输入新进程大小\n");
  scanf("%d",&(neicun[i].daxiao));
  printf("请输入新进程内容\n");
  scanf("%s",&(neicun[i].info));
  //创建进程，使标记位为1
  neicun[i].zhuangtai=1;
  printf("进程已成功创建！");
  shumu++;
  }
 }
}

//进程运行状态检测
void run(){
 printf("运行进程信息如下：");
 for(int i=0;i<20;i++){
  if(neicun[i].zhuangtai==1){
   //如果进程正在运行，则输出此运行进程的各个属性值
   printf("\n  pid %d ",neicun[i].pid);
```

```
   printf(" \n  pid=%d  ",neicun[i].pid);
   printf(" youxian=%d  ",neicun[i].youxian);
   printf(" daxiao=%d  ",neicun[i].daxiao);
   printf(" zhuanbgtai=%d  ",neicun[i].zhuangtai);
   printf(" info=%s  ",neicun[i].info);
   flag=1;
  }
 }
 if(!flag)
 printf("\n当前没有运行进程! \n");
}

//进程换出
void huanchu(){
 if(!shumu){
  printf("当前没有运行进程! \n");
  return;
 }
 printf("\n请输入换出进程的ID值");
 scanf("%d",&pid);
 for(int i=0;i<20;i++){
  //定位，找到要换出的进程，根据其状态进行相应处理
  if(pid==neicun[i].pid) {
   if(neicun[i].zhuangtai==1){
    neicun[i].zhuangtai==2;
    guaqi++;
    printf("\n已经成功换出进程\n");
   }
   else if(neicun[i].zhuangtai==0) printf("\n要换出的进程不存在\n");
   else printf("\n要换出的进程已被挂起\n");
   flag=1;
   break;
  }
 }
 //找不到，则说明进程不存在
 if(flag==0) printf("\n要换出的进程不存在\n");
}

//结束（杀死）进程
void kill(){
 if(!shumu){
  printf("当前没有运行进程! \n");
  return;
 }
 printf("\n输入杀死进程的ID值");
 scanf("%d",&pid);
 for(int i=0;i<20;i++){
  //定位，找到所要杀死的进程，根据其状态做出相应处理
  if(pid==neicun[i].pid){
   neicun[i].zhuangtai = 0;
   shumu--;
   printf("\n已经成功杀死进程\n");
  }
  else if(neicun[i].zhuangtai==0) printf ("\n要杀死的进程不存在\n");
  else printf("\n要杀死的进程已被挂起\n");
  flag=1;
  break;
 }
 //找不到，则说明进程不存在
 if(!flag) printf("\n 要杀死的进程不存在\n");
```

```
}

//唤醒进程
void huanxing(){
 if (!shumu) {
  printf("当前没有运行进程\n");
  return;
 }
 if(!guaqi){
  printf("\n当前没有挂起进程\n");
  return;
 }
 printf("\n输入进程pid:\n");
 scanf ("%d",&pid);
 for (int i=0; i<20;i++) {
  //定位，找到所要杀死的进程，根据其状态做相应处理
  if (pid==neicun[i].pid) {
   flag=false;
   if(neicun[i].zhuangtai==2){
    neicun[i].zhuangtai=1;
    guaqi--;
    printf ("\n已经成功唤醒进程\n");
   }
   else if(neicun[i].zhuangtai==0) printf("\n要唤醒的进程不存在\n");
   else printf("\n要唤醒的进程已被挂起\n");
   break;
  }
 }
 //找不到，则说明进程不存在
 if(flag) printf("\n要唤醒的进程不存在\n");
}

//主函数
int main()
{
 int n = 1;
 int num;
 //一开始所有进程都不在内存中
 for(int i=0;i<20;i++)
  neicun[i].zhuangtai = 0;
 while(n){
  printf("\n****************************************");
  printf("\n*  进程演示系统      *");
  printf("\n****************************************");
  printf("\n*1.创建新的进程  2.查看运行进程   *");
  printf("\n*3.换出某个进程  4.杀死运行进程   *");
  printf("\n*5.唤醒某个进程  6.退出系统   *");
  printf("\n****************************************");
  printf("\n请选择（1~6）\n");
  scanf("%d",&num);
  switch(num){
   case 1: create();break;
   case 2: run();break;
   case 3: huanchu(); break;
   case 4: kill();break;
   case 5: huanxing(); break;
   case 6: printf("已退出系统");exit(0);
   default: printf("请检查输入数值是否在系统功能中1~6");n=0;
  }
```

```
  }
  flag = 0;//恢复标记
 }
 return 0;
}
```

### 4.1代码分解介绍

**源程序中一共构造了5个函数方法来实现进程的创建、（运行进程的）显示、换出、结束（杀死）与唤醒。**

#### 4.1.1进程的创建

```
//创建进程
void create(){
 if(shumu>=20) printf("\n内存已满，请先换出或结束进程\n");   //内存容量大小设置为20
 else{
  int i;
  printf("**当前默认一次性创建5个进程，内存容量20**");
  for(i=0;i<5;i++)  {               //默认一次创建5个进程
  //定位，找到可以还未创建的进程
  if(neicun[i].zhuangtai==1) break;  //如果找到的进程在内存则结束 ，初始设置都不在内存中（main函数中设置状态为0）
  printf("\n请输入新进程pid\n");
  scanf("%d",&(neicun[i].pid));
  for(int j=0;j<i;j++)
   if(neicun[i].pid==neicun[j].pid){
    printf("\n该进程已存在\n");
    return;
   }
  printf("请输入新进程优先级\n");
  scanf("%d",&(neicun[i].youxian));
  printf("请输入新进程大小\n");
  scanf("%d",&(neicun[i].daxiao));
  printf("请输入新进程内容\n");
  scanf("%s",&(neicun[i].info));
  //创建进程，使标记位为1
  neicun[i].zhuangtai=1;
  printf("进程已成功创建！");
  shumu++;
  }
 }
}
```

（默认设置内存容量大小为20，一次性创建5个进程；进程创建时首先检测进程状态，若为1则表明该进程此前已在内存中了，不可再创建）

#### 4.1.2进程的显示（运行状态检测）

```
//进程运行状态检测
void run(){
 printf("运行进程信息如下：");
 for(int i=0;i<20;i++){
  if(neicun[i].zhuangtai==1){
   //如果进程正在运行，则输出此运行进程的各个属性值
   printf("\n  pid=%d ",neicun[i].pid);
   printf(" youxian=%d  ",neicun[i].youxian);
   printf(" daxiao=%d  ",neicun[i].daxiao);
   printf(" zhuanbgtai=%d  ",neicun[i].zhuangtai);
   printf(" info=%s  ",neicun[i].info);
   flag=1;
  }
 }
 if(!flag)
 printf("\n当前没有运行进程！\n");
}
```

### 4.1.3进程的换出

```
//进程换出
void huanchu(){
 if(!shumu){
  printf("当前没有运行进程！\n");
  return;
 }
 printf("\n请输入换出进程的ID值");
 scanf("%d",&pid);
 for(int i=0;i<20;i++){
  //定位，找到要换出的进程，根据其状态进行相应处理
  if(pid==neicun[i].pid) {
   if(neicun[i].zhuangtai==1){
    neicun[i].zhuangtai==2;
    guaqi++;
    printf("\n已经成功换出进程\n");
   }
   else if(neicun[i].zhuangtai==0) printf("\n要换出的进程不存在\n");
   else printf("\n要换出的进程已被挂起\n");
   flag=1;
   break;
  }
 }
 //找不到，则说明进程不存在
 if(flag==0) printf("\n要换出的进程不存在\n");
}
```

### 4.1.4进程的结束（杀死）

```
//结束（杀死）进程
void kill(){
 if(!shumu){
  printf("当前没有运行进程! \n");
  return;
 }
 printf("\n输入杀死进程的ID值");
 scanf("%d",&pid);
 for(int i=0;i<20;i++){
  //定位，找到所要杀死的进程，根据其状态做出相应处理
  if(pid==neicun[i].pid){
   neicun[i].zhuangtai = 0;
   shumu--;
   printf("\n已经成功杀死进程\n");
  }
  else if(neicun[i].zhuangtai==0) printf ("\n要杀死的进程不存在\n");
  else printf("\n要杀死的进程已被挂起\n");
  flag=1;
  break;
 }
 //找不到，则说明进程不存在
 if(!flag) printf("\n 要杀死的进程不存在\n");

}
```

### 4.1.5进程的唤醒

```
//唤醒进程
void huanxing(){
 if (!shumu) {
  printf("当前没有运行进程\n");
  return;
 }
 if(!guaqi){
  printf("\n当前没有挂起进程\n");
  return;
 }
 printf("\n输入进程pid:\n");
 scanf ("%d",&pid);
 for (int i=0; i<20;i++) {
  //定位，找到所要杀死的进程，根据其状态做相应处理
  if (pid==neicun[i].pid) {
   flag=false;
   if(neicun[i].zhuangtai==2){
    neicun[i].zhuangtai=1;
    guaqi--;
    printf ("\n已经成功唤醒进程\n");
   }
   else if(neicun[i].zhuangtai==0) printf("\n要唤醒的进程不存在\n");
   else printf("\n要唤醒的进程已被挂起\n");
   break;
  }
 }
 //找不到，则说明进程不存在
 if(flag) printf("\n要唤醒的进程不存在\n");
}
```

### 4.1.6 mian（）函数

```
//主函数
int main()
{
 int n = 1;
 int num;
 //一开始所有进程都不在内存中
 for(int i=0;i<20;i++)
  neicun[i].zhuangtai = 0;
 while(n){
  printf("\n*****************************************");
  printf("\n*  进程演示系统      *");
  printf("\n*****************************************");
  printf("\n*1.创建新的进程  2.查看运行进程   *");
  printf("\n*3.换出某个进程  4.杀死运行进程   *");
  printf("\n*5.唤醒某个进程  6.退出系统   *");
  printf("\n*****************************************");
  printf("\n请选择（1~6）\n");
  scanf("%d",&num);
  switch(num){
   case 1: create();break;
   case 2: run();break;
   case 3: huanchu(); break;
   case 4: kill();break;
   case 5: huanxing(); break;
   case 6: printf("已退出系统");exit(0);
   default: printf("请检查输入数值是否在系统功能中1~6");n=0;
  }
  flag = 0;//恢复标记
 }
 return 0;
}
```

---

## 5.运行结果截图


（图1--初始运行状态界面）

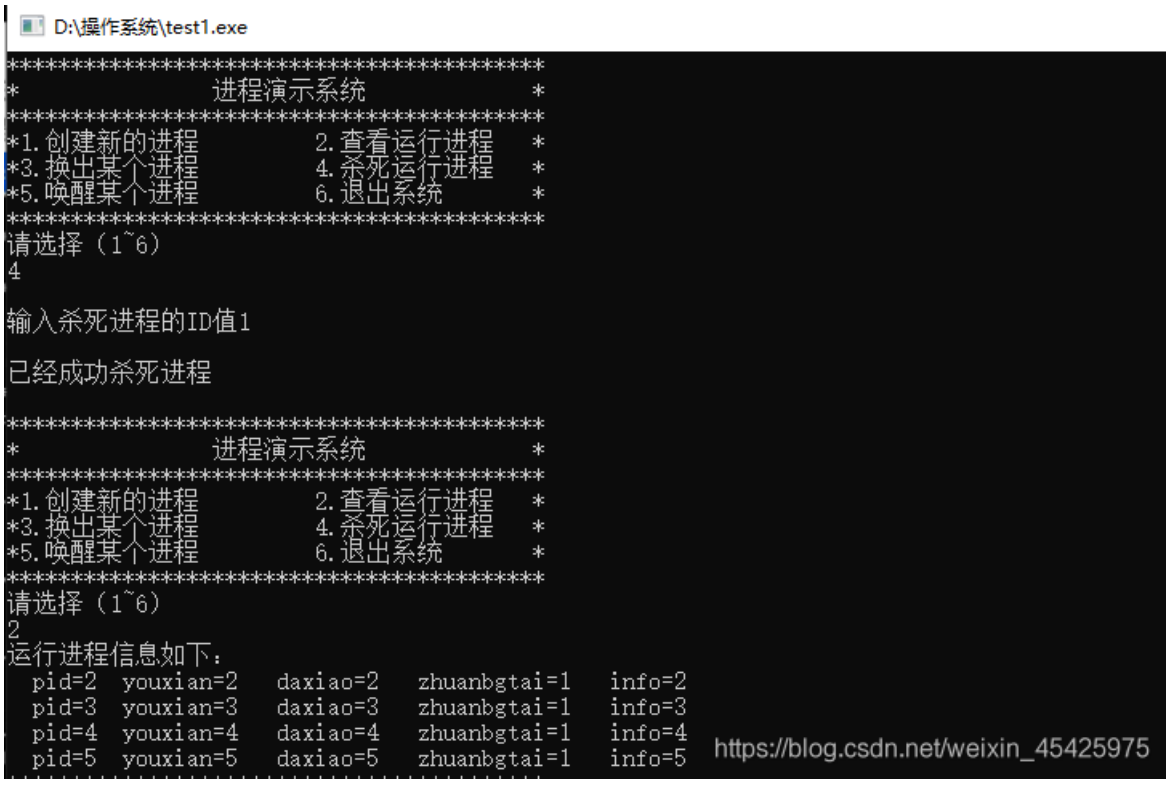

（图2--选择功能1 创建新进程）

源程序一次性创建5个进程，内存容量为20。


（图3--依次输入创建进程的pid、优先级、大小、内容）

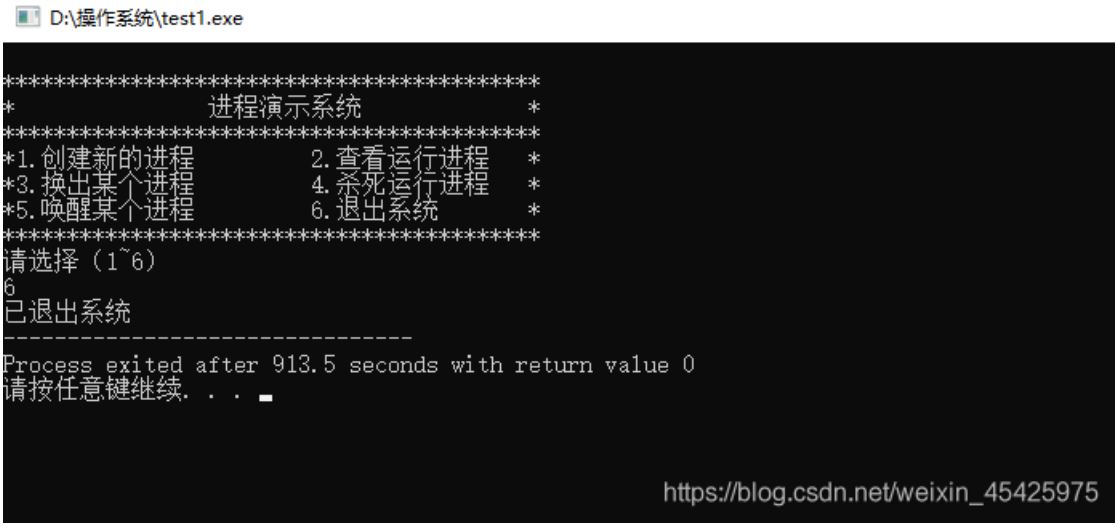

（图4--选择功能2 显示当前运行的进程）


（图5--换出进程 输入要换出的进程pid，换出后提示进程已换出）




（图6--杀死进程 输入要结束的进程pid，之后执行操作杀死此进程，再次查看运行进程时可以看到进程1已不在运行进程列）


（图7--唤醒进程 输入要唤醒的进程pid，之后该进程将被挂起）




（图8--退出系统 输入6，选择退出系统功，系统结束运行）

---

本文为课程实验记录，参考学校实验教材书籍，重在学习交流。

码字不易，走过路过点个赞□吧□(❁´◡`❁)