实 验 报 告

课程名称：操作系统试验

实 验 二：进程调度

班　　级：02

学生姓名：白文强

学　　号：20191060064

专　　业：计算机科学与技术

指导教师：杨旭涛

学　　期：2021－2022 学年秋季学期

成　　绩：

云南大学信息学院

一、实验目的

1、熟悉进程的定义和描述，熟悉进程控制块；

2、掌握进程的状态定义及其转换过程；

3、掌握进程的基本调度算法，包括先来先服务，轮转法，优先级法，多级反馈轮转法，最短进程优先法，最高响应比优先法等；

二、知识要点

1、进程控制块 PCB；

2、进程的初始化、就绪、执行、等待和终止状态；

3、先来先服务，轮转法，优先级法，多级反馈轮转法，最短进程优先法，最高响应比优先法等调度算法；

三、实验预习（要求做实验前完成）

1、了解 linux 系统中常用命令的使用方法；

2、掌握进程 PCB 控制块的内容和描述；

3、掌握系统状态的转换过程；

4、掌握常用进程调度算法的原理

四、实验内容和试验结果

结合课程所讲授内容以及课件中的试验讲解，完成以下试验。请分别描述程序的流程，附上源代码，并将试验结果截图附后。

1、通过编程模拟实现轮转法进程调度算法。

系统中的每个进程用一个进程控制块 PCB 表示；将多个进程按输入顺序排成就绪队列链表（进程信息从键盘录入）；按进程在链表中的顺序依次调度，每个被调度的进程执行一个时间片，然后回到就绪队列，“已运行时间”加 1；若进程“要求运行时间”==“已运行时间”，则将其状态置为“结束”，并退出队列；运行程序，显示每次调度时被调度运行的进程 id，以及各进程控制块的动态变化过程。

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

typedef struct pcb
{
    int pid;
    char state;
    int total_time;
```

```
    int cputime;
    struct pcb* next;
}*proc;

int proc_num;
proc head, tail;

int init_pcb(){
    int i;
    proc p, tmp;
    printf("please input the number of processes:\n");
    scanf("%d", &proc_num);
    printf("there are %d processes, please input pcb info:\n", proc_num);

    p = (proc)malloc(sizeof(struct pcb));
    printf("process id: ");
    scanf("%d",&p->pid);
    printf("cputime required: ");
    scanf("%d",&p->total_time);
    p->state = 'R';
    p->cputime = 0;
    head=p;

    for(i = proc_num; i > 1; i--){
        tmp = p;
        p = (proc)malloc(sizeof(struct pcb));
        printf("process id:");
        scanf("%d", &p->pid);
        printf("cputime required: ");
        scanf("%d", &p->total_time);
        p->state = 'R';
        p->cputime = 0;
        tmp->next = p;
    }
    tail = p;
    p->next = head;

    return 0;
}

void display(){
    int i;
    proc p = head;
```

```
52.     printf("pid\tcpu_time \treq_time\n");
53.     for(i = 0; i < proc_num; i++){
54.         printf("%d\t%d\t\t%d\n",p->pid, p->cputime,p->total_time);
55.         p = p->next;
56.     }
57. }
58. 
59. void sched(){
60.     int round = 1, i;
61.     proc tmp = tail;
62.     proc p = head;
63.     while(p->total_time > p->cputime){
64.         printf("\nRound %d, Process %d is running\n", round, p->pid);
65.         p->cputime++;
66.         display();
67.         if( p->total_time == p->cputime ){
68.             p->state='E';
69.             proc_num--;
70.             tmp->next = p->next;
71.             if( p == head ) {
72.                 head = p->next;
73.             }
74.             printf("process %d is finished\n", p->pid);
75.         }else{
76.             tmp = p;
77.         }
78.         p = p->next;
79.         round++;
80.     }
81. }
82. 
83. int main(){
84.     init_pcb();
85.     display();
86.     sched();
87.     return 0;
88. }
```

代码运行情况：

```
bwq@ubuntu:~/桌面/C$ ./process_schedu
please input the number of processes:
3
there are 3 processes, please input pcb info:
process id: 1
cputime required: 2
process id:2
cputime required: 3
process id:3
cputime required: 2
pid     cpu_time        req_time
1       0               2
2       0               3
3       0               2

Round 1, Process 1 is running
pid     cpu_time        req_time
1       1               2
2       0               3
3       0               2

Round 2, Process 2 is running
pid     cpu_time        req_time
1       1               2
2       1               3
3       0               2

Round 3, Process 3 is running
pid     cpu_time        req_time
1       1               2
2       1               3
3       1               2

Round 4, Process 1 is running
pid     cpu_time        req_time
1       2               2
2       1               3
3       1               2
process 1 is finished

Round 5, Process 2 is running
```

```
pid     cpu_time        req_time
2       2               3
3       1               2

Round 6, Process 3 is running
pid     cpu_time        req_time
2       2               3
3       2               2
process 3 is finished

Round 7, Process 2 is running
pid     cpu_time        req_time
2       3               3
process 2 is finished
```

由运行结果可以看出，使用轮转调度算法，所有进程轮流使用 CPU，当前进程的时间片到了之后，若还有其他进程，CPU 将会被调度给其他进程使用，每个进程每次使用一个时间片。

2、参考第一题的描述，通过编程模拟实现动态优先级轮转调度算法。

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

typedef struct pcb
{
    int pid;         //进程 id
    char state;      //进程状态
    int total_time; //总共运行时间
    int cputime;     //已运行时间
     int priority;    //优先级
     struct pcb* next;
}*proc;

int proc_num;
proc head, tail;

int init_pcb(){
    int i;
    proc p, tmp;
    printf("please input the number of processes:\n");
    scanf("%d", &proc_num);
    printf("there are %d processes, please input pcb info:\n", proc_num);

    p = (proc)malloc(sizeof(struct pcb));
    printf("process id: ");
    scanf("%d", &p->pid);
    printf("cputime required: ");
    scanf("%d", &p->total_time);
    printf("process priority:");
    scanf("%d", &p->priority);

    p->state = 'R';
    p->cputime = 0;
    head=p;

    for(i = proc_num; i > 1; i--){
        tmp = p;
        p = (proc)malloc(sizeof(struct pcb));
        printf("process id:");
        scanf("%d", &p->pid);
        printf("cputime required: ");
        scanf("%d", &p->total_time);
```

```
43.         printf("process priority:");
44.         scanf("%d", &p->priority);
45.         p->state = 'R';
46.         p->cputime = 0;
47.         tmp->next = p;
48.     }
49.     tail = p;
50.     p->next = head;
51.
52.     return 0;
53. }
54.
55. void display(){
56.     int i;
57.     proc p = head;
58.     printf("pid\tcpu_time\treq_time\tpriority\n");
59.     for(i = 0; i < proc_num; i++){
60.         printf("%d\t%d\t\t%d\t\t%d\n",p->pid, p->cputime,p->total_tim
    e,p->priority);
61.         p = p->next;
62.     }
63. }
64.
65. void sched(){
66.     int round = 1, i;
67.     proc tmp = tail;
68.     proc ready_to_run;
69.     proc p;
70.     while( proc_num != 0 ){
71.         // seek a should to do process
72.         ready_to_run = head;
73.         p = head->next;
74.         while(p != head){
75.             if( p->total_time > p->cputime && p->priority > ready_to_
    run->priority && p->state=='R'){
76.                 ready_to_run = p;
77.             }
78.             p = p->next;
79.         }
80.
81.         printf("\nRound %d, Process %d is running\n", round, ready_to
    _run->pid);
82.         ready_to_run->cputime++;
83.         ready_to_run->priority--;
```

```
84.
85.         display();
86.         if( ready_to_run->cputime == ready_to_run->total_time ){
87.             ready_to_run->state = 'E';
88.             proc_num--;
89.             if( ready_to_run == head ){
90.                 head = ready_to_run->next;
91.             }
92.             while( tmp->next != ready_to_run ){
93.                 tmp = tmp->next;
94.             }
95.             tmp->next = ready_to_run->next;
96.
97.             printf("process %d is finished\n", ready_to_run->pid);
98.         }
99.         round++;
100.      }
101.  }
102.
103.  int main(){
104.      init_pcb();
105.      display();
106.      sched();
107.      return 0;
108.  }
```

```
bwq@ubuntu:~/桌面/C$ ./process_schedu2
please input the number of processes:
3
there are 3 processes, please input pcb info:
process id: 1
cputime required: 2
process priority:5
process id:2
cputime required: 2
process priority:1
process id:3
cputime required: 5
process priority:4
pid     cpu_time        req_time        priority
1       0               2               5
2       0               2               1
3       0               5               4

Round 1, Process 1 is running
pid     cpu_time        req_time        priority
1       1               2               4
2       0               2               1
3       0               5               4

Round 2, Process 1 is running
pid     cpu_time        req_time        priority
1       2               2               3
2       0               2               1
3       0               5               4
process 1 is finished

Round 3, Process 3 is running
pid     cpu_time        req_time        priority
2       0               2               1
3       1               5               3

Round 4, Process 3 is running
pid     cpu_time        req_time        priority
2       0               2               1
3       2               5               2
```

```
Round 5, Process 3 is running
pid     cpu_time        req_time        priority
2       0               2               1
3       3               5               1

Round 6, Process 2 is running
pid     cpu_time        req_time        priority
2       1               2               0
3       3               5               1

Round 7, Process 3 is running
pid     cpu_time        req_time        priority
2       1               2               0
3       4               5               0

Round 8, Process 2 is running
pid     cpu_time        req_time        priority
2       2               2               -1
3       4               5               0
process 2 is finished

Round 9, Process 3 is running
pid     cpu_time        req_time        priority
3       5               5               -1
process 3 is finished
```

从程序运行结果中可以看到，采用动态优先级调度算法，CPU 会被调度给当前优先级最高的进程使用，当前进程执行完毕后，其优先级会减 1。采用动态优先级的方法，让优先级较低的进程也有机会进入 CPU 执行，不会出现优先级低的进程一直等待的情况。

3、参考第一题的描述，通过编程模拟实现最高响应比优先进程调度算法。

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

typedef struct pcb
{
    int pid;        //进程 id
    char state;     //进程状态
    int arrive_time;//到达时间
    int total_time; //总共运行时间
     int cputime;    //已运行时间
     struct pcb* next;
}*proc;

int time = 0;
int proc_num;
proc head, tail;

int init_pcb(){
    int i;
    proc p, tmp;
    printf("please input the number of processes:\n");
    scanf("%d", &proc_num);
    printf("there are %d processes, please input pcb info:\n", proc_num);

    p = (proc)malloc(sizeof(struct pcb));
    printf("process id: ");
    scanf("%d", &p->pid);
    printf("arrive time: ");
    scanf("%d", &p->arrive_time);
    printf("cputime required: ");
    scanf("%d", &p->total_time);

    p->state = 'R';
    p->cputime = 0;
    head=p;

    for(i = proc_num; i > 1; i--){
        tmp = p;
        p = (proc)malloc(sizeof(struct pcb));
        printf("process id:");
        scanf("%d", &p->pid);
```

```
42.         printf("arrive time: ");
43.         scanf("%d", &p->arrive_time);
44.         printf("cputime required: ");
45.         scanf("%d", &p->total_time);
46.         p->state = 'R';
47.         p->cputime = 0;
48.         tmp->next = p;
49.     }
50.     tail = p;
51.     p->next = head;
52.
53.     return 0;
54. }
55.
56. void display(){
57.     int i;
58.     proc p = head;
59.     printf("pid\tarrive_time\tcpu_time \treq_time\n");
60.     for(i = 0; i < proc_num; i++){
61.         printf("%d\t%d\t%d\t\t%d\n",p->pid, p->arrive_time,p->cputime
    ,p->total_time);
62.         p = p->next;
63.     }
64. }
65.
66. void sched(){
67.     int round = 1, i;
68.     proc tmp = tail;
69.     proc ready_to_run;
70.     proc p;
71.     while( proc_num != 0 ){
72.         // seek a should to do process
73.         ready_to_run = head;
74.         double ready_response_scale = (double)(time - ready_to_run->a
    rrive_time + ready_to_run->total_time)/ready_to_run->total_time;
75.
76.         p = ready_to_run->next;
77.         for(i = 1; i < proc_num; i++){
78.             if(p->arrive_time < time){
79.                 //p already arrived;
80.                 double response_scale = (double)(time - p->arrive_tim
    e + p->total_time)/p->total_time;
81.                 if(response_scale > ready_response_scale){
82.                     ready_to_run = p;
```

```
                    ready_response_scale = response_scale;
                }
            }
            p = p->next;
        }


        tmp = ready_to_run->next;
        while(tmp->next != ready_to_run){
            tmp = tmp->next;
        }

        printf("\nRound %d, Process %d is running\n", round, ready_to
_run->pid);
        ready_to_run->cputime++;

        display();
        if( ready_to_run->cputime == ready_to_run->total_time ){
              time+=ready_to_run->total_time;
              ready_to_run->state = 'E';
              proc_num--;
              tmp->next = ready_to_run->next;
              if( ready_to_run == head ){
                  head = ready_to_run->next;
              }
              printf("process %d is finished\n", ready_to_run->pid);

          }
          round++;
      }
  }

  int main(){
      init_pcb();
      display();
      sched();
      return 0;
  }
```

程序运行截图：

```
bwq@ubuntu:~/桌面/C$ ./process_schedu3
please input the number of processes:
5
there are 5 processes, please input pcb info:
process id: 1
arrive time: 0
cputime required: 3
process id:2
arrive time: 2
cputime required: 6
process id:3
arrive time: 4
cputime required: 4
process id:5
arrive time: 6
cputime required: 5
process id:7
arrive time: 8
cputime required: 2
pid     arrive_time     cpu_time        req_time
1       0               0               3
2       2               0               6
3       4               0               4
5       6               0               5
7       8               0               2

Round 1, Process 1 is running
pid     arrive_time     cpu_time        req_time
1       0               1               3
2       2               0               6
3       4               0               4
5       6               0               5
7       8               0               2

Round 2, Process 1 is running
pid     arrive_time     cpu_time        req_time
1       0               2               3
2       2               0               6
3       4               0               4
5       6               0               5
7       8               0               2
```

```
Round 3, Process 1 is running
pid     arrive_time     cpu_time        req_time
1       0               3               3
2       2               0               6
3       4               0               4
5       6               0               5
7       8               0               2
process 1 is finished

Round 4, Process 2 is running
pid     arrive_time     cpu_time        req_time
2       2               1               6
3       4               0               4
5       6               0               5
7       8               0               2
```

```
Round 5, Process 2 is running
pid     arrive_time     cpu_time        req_time
2       2               2               6
3       4               0               4
5       6               0               5
7       8               0               2

Round 6, Process 2 is running
pid     arrive_time     cpu_time        req_time
2       2               3               6
3       4               0               4
5       6               0               5
7       8               0               2

Round 7, Process 2 is running
pid     arrive_time     cpu_time        req_time
2       2               4               6
3       4               0               4
5       6               0               5
7       8               0               2
```

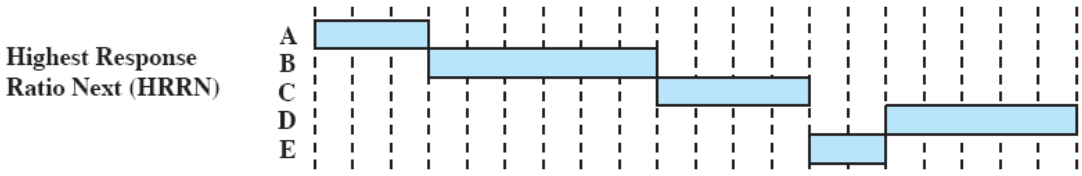
```
Round 8, Process 2 is running
pid     arrive_time     cpu_time        req_time
2       2               5               6
3       4               0               4
5       6               0               5
7       8               0               2

Round 9, Process 2 is running
pid     arrive_time     cpu_time        req_time
2       2               6               6
3       4               0               4
5       6               0               5
7       8               0               2
process 2 is finished

Round 10, Process 3 is running
pid     arrive_time     cpu_time        req_time
3       4               1               4
5       6               0               5
7       8               0               2

Round 11, Process 3 is running
pid     arrive_time     cpu_time        req_time
3       4               2               4
5       6               0               5
7       8               0               2

Round 12, Process 3 is running
pid     arrive_time     cpu_time        req_time
3       4               3               4
5       6               0               5
7       8               0               2

Round 13, Process 3 is running
pid     arrive_time     cpu_time        req_time
3       4               4               4
5       6               0               5
7       8               0               2
process 3 is finished
```

```
Round 14, Process 7 is running
pid     arrive_time     cpu_time        req_time
5       6               0               5
7       8               1               2

Round 15, Process 7 is running
pid     arrive_time     cpu_time        req_time
5       6               0               5
7       8               2               2
process 7 is finished

Round 16, Process 5 is running
pid     arrive_time     cpu_time        req_time
5       6               1               5

Round 17, Process 5 is running
pid     arrive_time     cpu_time        req_time
5       6               2               5

Round 18, Process 5 is running
pid     arrive_time     cpu_time        req_time
5       6               3               5

Round 19, Process 5 is running
pid     arrive_time     cpu_time        req_time
5       6               4               5

Round 20, Process 5 is running
pid     arrive_time     cpu_time        req_time
5       6               5               5
process 5 is finished_
```

上面运行的程序模拟了 5 个进程最高响应比调度执行过程：

Process	Arrival Time	Service Time
A	0	3
B	2	6
C	4	4
D	6	5
E	8	2

调度过程如下图：



采用最高响应比优先算法，使等待时间较长的作业也能获得执行的机会。

五、问题讨论

1、如何理解进程执行的顺序性、并发性和并行性的概念。

顺序性：对于一个进程来说，它所有的指令都是按顺序执行的，对于多个进程来说，它们之间是按照一定的顺序依次执行的。

并发性：对于一个进程来说，它所有的指令时按顺序执行的，对于多个进程来说，他们是交叉执行的。

并行性：两个或多个事件在同一时间点同时执行，这取决于 CPU 有多少个核心可以同时执行。

2、描述产生死锁的原因，以及解决方法。

产生死锁的原因：

（1）系统资源不足。

（2）进程运行推进的顺序不合适。

（3）资源分配不当等。

如果系统资源充足，进程的资源请求都能够得到满足，死锁出现的可能性就很低，否则就会因争夺有限的资源而陷入死锁。其次，进程运行推进顺序与速度不同，也可能产生死锁。

解决死锁的办法：

产生死锁有四个必要条件：

（1）互斥条件：一个资源每次只能被一个进程使用。

（2）请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。

（3）不剥夺条件:进程已获得的资源，在末使用完之前，不能强行剥夺。

（4）循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

只要上述条件之一不满足，就不会发生死锁。通常采用剥夺资源和终止进程的方式来让进程释放临界资源，以达到让某个进程可以执行完毕进而释放资源解除死锁的目的。