北 京 林 业 大 学

2019 学年—2020 学年第 2 学期操作系统实验报告书

专业：计算机科学与技术(创新实验班)　班级：计创 18

姓　　名：连月菡　　学　　号：181002222

实验地点：家　　任课教师：李巨虎

实验题目：实验 1 进程调度

实验环境：Visual Studio 2019 Community

一、 实验目的

1）掌握处理机调度及其实现；
2）掌握进程状态及其状态转换；
3）掌握进程控制块 PCB 及其作用。

二、 实验内容

进程调度模拟程序：假设有 10 个进程需要在 CPU 上执行，分别用：

- ✧ 先进先出调度算法；0
- ✧ 基于优先数的调度算法；1
- ✧ 最短执行时间调度算法 2

确定这 10 个进程在 CPU 上的执行过程。要求每次进程调度时在屏幕上显示：

- ➢ 当前执行进程；
- ➢ 就绪队列；
- ➢ 等待队列

三、 实验步骤

1） 创建 10 个进程的 PCB，每个 PCB 包括：进程名、进程状态、优先级（1~10）、需要在处理机上执行的时间（1~50ms）、队列指针等；

```
2）  for (i = 0; i < 10; i++) {
3）      PCB p;
4）      p.name = i + 1;
5）      p.prior = rand() % 10;  //进程名为 0-9
6）      p.state = rand() % 2;   //状态
7）      p.time = rand() % 50+1;
8）    }
```

2） 初始化 10 个 PCB（产生随机数 0 或 1，分别表示进程处于就绪态或

等待态)；rand()%2，rand()%10+1，rand()%50+1；
3） 根据调度算法选择一个就绪进程在 CPU 上执行；

```
FCFS(wait, ok);  //先进先出算法
```

4） 在进程执行过程中，产生随机数 0 或 1，该随机数为 1 时，将等待队列中的第一个 PCB 加入就绪队列的对尾；

```
9）    if (p.state)
10）         EnQueue(ok, p);
11）     else
12）         EnQueue(wait, p);
13）
14）     cout << "进程名" << "\t" << p.name << "\t";
15）     cout << "优先级" << p.prior << "\t";
16）     if (p.state)
17）         cout << "状态为 1" << "    ";
18）     else
19）         cout << "状态为 0" << "    ";
20）     cout<<"所需时间" << p.time << "ms" << endl << endl;
21） }
```

5） 在进程执行过程中，产生一个随机数，表示执行进程能在处理机上执行的时间，如果随机时间(1~20ms)大于总需要的时间，则执行完成。如果小于，则从总时间中减去执行时间。

6） 如果执行进程没有执行完成。则产生随机数 0 或 1，当该随机数为 0 时，将执行进程加入就绪队列对尾；否则，将执行进程加入等待队列对尾；
7） 一直到就绪队列为空，程序执行结束。

```
1.  while (QueueEmpty(wait) && QueueEmpty(ok)) {
2.          cout << endl << "第" << i << "次: ";
3.          int j = rand() % 3;
4.          if (j == 0) {
5.              DeQueue(ok, e);
6.              cout << "当前执行进程" << e.name << "出队" << endl;
7.          }
8.          else if (j == 1) {
9.              DeQueue(ok, e);
10.             EnQueue(ok, e);
11.             cout << "当前执行进程" << e.name << "进队" << endl;
12.         }
13.         else if (j == 2) {
14.             DeQueue(ok, e);
15.             EnQueue(wait, e);
```

```
            DeQueue(wait, f);
            EnQueue(ok, f);
            cout << "当前执行进程" << e.name << "进队" << endl;
        }

        Print(wait, ok);

        i++;
        cout << endl;
    }
```

```
进程名  1       优先级1 状态为1   所需时间35ms

进程名  2       优先级0 状态为1   所需时间25ms

进程名  3       优先级8 状态为0   所需时间13ms

进程名  4       优先级4 状态为1   所需时间46ms

进程名  5       优先级1 状态为1   所需时间12ms

进程名  6       优先级1 状态为1   所需时间43ms

进程名  7       优先级7 状态为0   所需时间42ms

进程名  8       优先级4 状态为0   所需时间4ms

进程名  9       优先级2 状态为0   所需时间22ms

进程名  10      优先级6 状态为0   所需时间46ms


就绪队列: 1   2   4   5   6
等待队列: 3   7   8   9   10
第1次: 当前执行进程1进队
就绪队列: 2   4   5   6   3
等待队列: 7   8   9   10   1

第2次: 当前执行进程2出队
就绪队列: 4   5   6   3
等待队列: 7   8   9   10   1

第3次: 当前执行进程4进队
就绪队列: 5   6   3   7
等待队列: 8   9   10   1   4

第4次: 当前执行进程5出队
就绪队列: 6   3   7
等待队列: 8   9   10   1   4

第5次: 当前执行进程6出队
就绪队列: 3   7
等待队列: 8   9   10   1   4

第6次: 当前执行进程3进队
就绪队列: 7   3
等待队列: 8   9   10   1   4

第7次: 当前执行进程7进队
就绪队列: 3   8
等待队列: 9   10   1   4   7

第8次: 当前执行进程3进队
就绪队列: 8   3
等待队列: 9   10   1   4   7

第9次: 当前执行进程8进队
就绪队列: 3   8
等待队列: 9   10   1   4   7

第10次: 当前执行进程3出队
就绪队列: 8
等待队列: 9   10   1   4   7

第11次: 当前执行进程8进队
就绪队列: 9
等待队列: 10   1   4   7   8

第12次: 当前执行进程9出队

执行完毕
等待队列: 10   1   4   7   8
结束了
```

四、实验心得

通过这次实验, FIFO 算法总是把处理机分配给最先进入就绪队列的进程，一个进程一旦分得处理机，便一直执行下去，直到该进程完成或阻塞时，才释放处理机。
由输出结果可见，FIFO 算法服务质量不佳，容易引起作业用户不满，常作为一种辅助调度算法。
借助 C++编程实现 FIFO 算法, 使得我对进程调度的方式有了更深刻的理解。