实验 4　进程调度的模拟

实验内容：

熟悉进程调度的各种算法，并对模拟程序给出数据和流程的详细分析，之后画出流程图，**最后参考模拟程序写出时间片轮转调度算法的程序**。

实验目的：

通过本实验，加深对进程调度算法原理和过程的理解。

实验要求：

(1) 对调度算法进行详细分析，在仔细分析的基础上，完全理解主要数据结构和过程的作用，给出主要数据结构的说明及画出主要模块的流程图。

(2) **根据提示信息，把函数写完整，使成为一个可运行程序。**

(3) 反复运行程序，观察程序执行的结果，验证分析的正确性，然后给出一次执行的最后运行结果，并由结果计算出周转时间和带权周转时间。

1 数据结构与算法、流程图

1.1 数据结构与算法

时间片轮转算法是一种常见的进程调度算法，它旨在公平地分配 CPU 时间片给就绪状态的进程。下面是对该算法的详细分析：

1.1.1 数据结构

ProcStruct 结构体：用于表示进程的相关信息，包括进程 ID、状态、运行序列、位置、开始时间、结束时间、CPU 时间、IO 时间和下一个进程指针等字段。

全局变量：RunPoint 表示当前运行的进程指针，WaitPoint 表示阻塞进程指针，ReadyPoint 表示就绪进程指针，ClockNumber 表示时钟计数器，ProcNumber 表示进程数量，FinishedProc 表示已完成的进程数量。

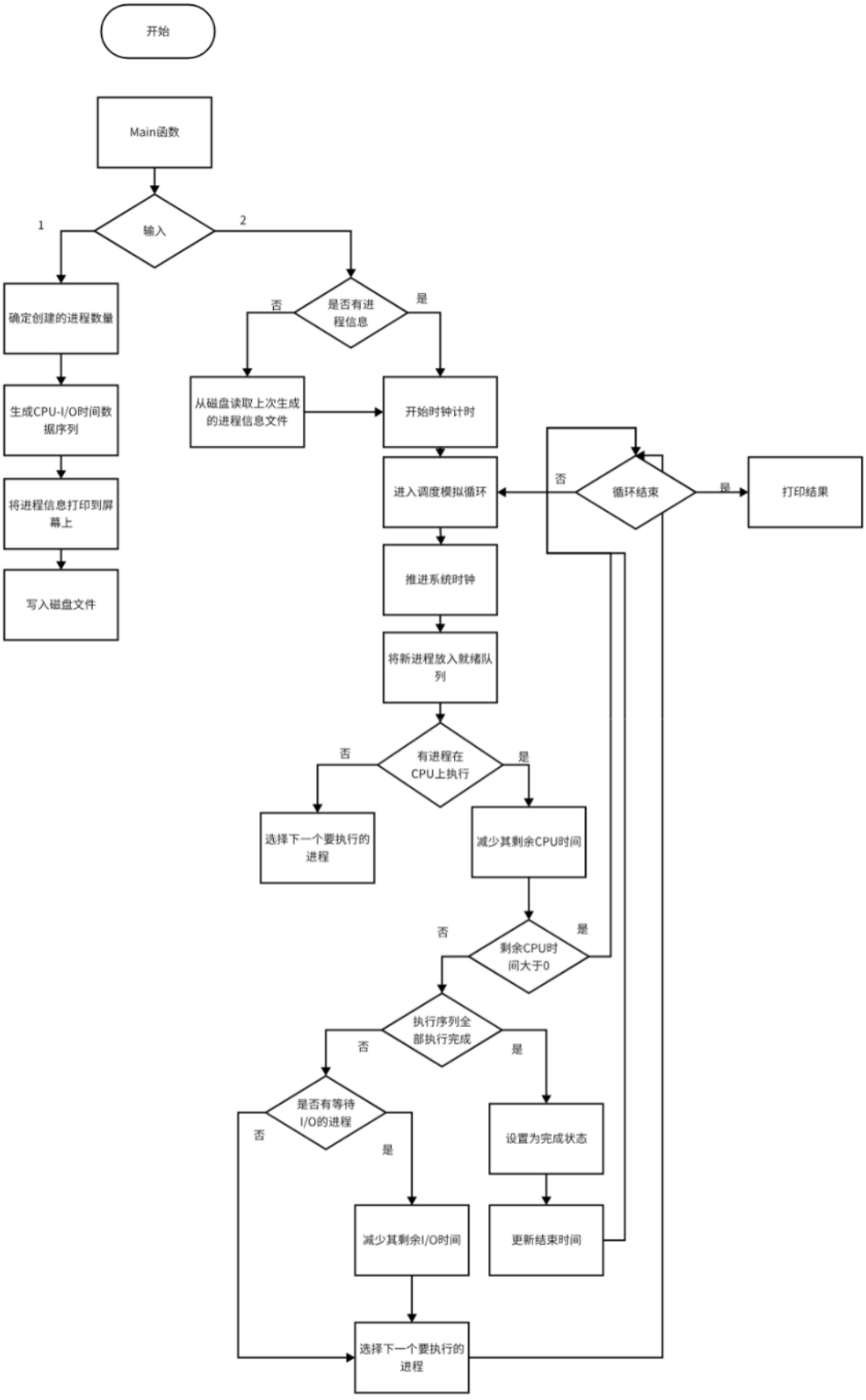
1.1.2 调度算法

在每个时钟周期内，根据当前时钟计数器的值，将到达开始时间的进程添加到就绪队列中。

CPU 调度：如果没有进程在 CPU 上运行，则选择下一个要运行的进程。如果当前运行的进程的 CPU 时间片还未用完，则继续运行。如果 CPU 时间片用完，则根据其运行序列决定进程的状态：如果序列中的所有任务都已完成，则将进程设置为已完成状态；否则，将进程设置为阻塞状态，并将其添加到阻塞队列中。

IO 调度：如果没有进程在阻塞状态，则返回。否则，减少阻塞进程的 IO 时间片。如果 IO 时间片用完，则将进程从阻塞队列中移除，并将其设置为就绪状态，添加到就绪队列的末尾。

1.2 流程图


开始
Main函数
输入
1
2
确定创建的进程数量
生成CPU-I/O时间数据序列
将进程信息打印到屏幕上
写入磁盘文件
是否有进程信息
否
是
从磁盘读取上次生成的进程信息文件
开始时钟计时
进入调度模拟循环
循环结束
否
是
打印结果
推进系统时钟
将新进程放入就绪队列
有进程在CPU上执行
否
是
选择下一个要执行的进程
减少其剩余CPU时间
剩余CPU时间大于0
否
是
执行序列全部执行完成
否
是
是否有等待I/O的进程
否
是
设置为完成状态
减少其剩余I/O时间
更新结束时间
选择下一个要执行的进程

2 时间片轮转算法程序

```
struct ProcStruct {
    int p_pid; // 进程的标识号
    char p_state; // 进程的状态，C--运行 R--就绪 W--组塞 B--后备 F--完成
    int p_rserial[10]; // 模拟的进程执行的 CPU 和 I/O 时间数据序列，间隔存储，第 0 项存储随后序列的长度（项数），以便知晓啥时该进程执行结束
    int p_pos; // 当前进程运行到的位置，用来记忆执行到序列中的哪项
    int p_starttime; // 进程建立时间
    int p_endtime; // 进程运行结束时间
    int p_cputime; // 当前运行时间段进程剩余的需要运行时间
    int p_iotime; // 当前 I/O 时间段进程剩余的 I/O 时间
    int p_next; // 进程控制块的链接指针，指向该进程所在队列的下一个进程 id
} proc[10];

int RunPoint; // 运行进程指针，-1 时为没有运行进程
int WaitPoint; // 阻塞队列指针，-1 时为没有阻塞进程
int ReadyPoint; // 就绪队列指针，-1 时为没有就绪进程
long ClockNumber; // 系统时钟
int ProcNumber; // 系统中模拟产生的进程总数
int FinishedProc; // 系统中目前已执行完毕的进程总数

void Create_ProcInfo(); //创建进程
void DisData(); //显示进程初始值
void Scheduler_FF(); //进程调度函数
void Read_Process_Info(); //从磁盘读取进程初始值
void NewReadyProc(); // 判别新进程是否到达
void Cpu_Sched(); // CPU 调度
void IO_Sched(); // IO 调度
void Display_ProcInfo(); //显示当前状态
void NextRunProcess();//寻找下一个运行程序
void Statistic(); //统计

int main() {
    while (true) {
        RunPoint = -1; // 运行进程指针，-1 时为没有运行进程
        WaitPoint = -1; // 阻塞队列指针，-1 时为没有阻塞进程
        ReadyPoint = -1; // 就绪队列指针，-1 时为没有就绪进程
        ClockNumber = 0; // 系统时钟
        ProcNumber = 0; // 当前系统中的进程总数
        printf("**********************************\n");
        printf("1: 建立进程调度数据序列 \n");
        printf("2: 读进程信息，执行调度算法\n");
        printf("**********************************\n");
```

```
            printf("Enter your choice (1 ~ 2): ");
            char ch; cin >> ch;
            cout << endl;
            if (ch == '1') Create_ProcInfo();
            else if (ch == '2') Scheduler_FF();
            else cout << "invalid input" << endl;
        }
}

void Create_ProcInfo() {
        srand(GetTickCount()); //初始化种子
        ProcNumber = rand() % 5 + 5; //随机产生进程数量
        for (int i = 0; i < ProcNumber; i++) { //  生成进程的 CPU--I/O 时间数据序列
            proc[i].p_pid = ((float)rand() / 32767) * 1000; //  初始化随机的进程 ID 号
            proc[i].p_state = 'B'; //  初始都为后备状态，可用其他表示符
            //生成 CPU 和 I/O 时间序列
            int listLength = proc[i].p_rserial[0] = rand() % 7 + 1; //序列长度
            for (int j = 1; j <= listLength; j++)
                proc[i].p_rserial[j] = rand() % 3 + 1;
            proc[i].p_pos = 1;
            proc[i].p_starttime = rand() % 5 + 1;
            proc[i].p_endtime = -1;
            proc[i].p_cputime = proc[i].p_rserial[1];
            proc[i].p_iotime = -1;
            proc[i].p_next = -1;
        }
        printf("建立了%d 个进程数据序列\n\n", ProcNumber);
        DisData();
}

void DisData() {
        ofstream outFile;
        outFile.open("./Process_Info.txt", ios::out);
        for (int i = 0; i < ProcNumber; i++) {
            outFile << format("ID={}(len={},startTime={}):", proc[i].p_pid, proc[i].p_rserial[0],
proc[i].p_starttime);
            cout << format("ID={}(len={},startTime={}):", proc[i].p_pid, proc[i].p_rserial[0],
proc[i].p_starttime);
            for (int j = 1; j <= proc[i].p_rserial[0]; j++) {
                outFile << proc[i].p_rserial[j] << " ";
                cout << proc[i].p_rserial[j] << " ";
            }
            outFile << endl;
            cout << endl;
```

```
    }
    cout << endl;
    outFile.close();
}

void Scheduler_FF() {
    if (ProcNumber == 0) Read_Process_Info(); //磁盘读取上次的进程信息
    NewReadyProc();
    Display_ProcInfo();
    while (FinishedProc < ProcNumber) {
        ClockNumber++; // 时钟前进 1 个单位
        NewReadyProc(); // 判别新进程是否到达
        Cpu_Sched(); // CPU 调度
        IO_Sched(); // IO 调度
        Display_ProcInfo(); //显示当前状态
        Sleep(0);
    }
    Statistic();
}

void NewReadyProc() {
    for (int i = 0; i < ProcNumber; i++) {
        if (proc[i].p_starttime == ClockNumber) {// 进程进入时间达到系统时间，
ClockNumber 是当前的系统时间
            proc[i].p_state = 'R'; // 进程状态修改为就绪
            proc[i].p_next = -1; // 该进行即将要挂在队列末尾，它肯定是尾巴，后面没人
的，所以先设置 next=-1
            if (ReadyPoint == -1) // 如果当前就绪队列无进程
                ReadyPoint = i;
            else { // 如果就绪队列有进程，放入队列尾
                int n = ReadyPoint;
                while (proc[n].p_next != -1)
                    n = proc[n].p_next; //找到原来队伍中的尾巴
                proc[n].p_next = i; //挂在这个尾巴后面
            }
        }
    }
}

void Cpu_Sched() {
    int n;
    if (RunPoint == -1) { // 没有进程在 CPU 上执行
        NextRunProcess();
        return;
```

```
    }
    proc[RunPoint].p_cputime--; // 进程 CPU 执行时间减少 1 个时钟单位
    if (proc[RunPoint].p_cputime > 0) return; // 还需要 CPU 时间，下次继续，这次就返回了
    if (proc[RunPoint].p_rserial[0] == proc[RunPoint].p_pos) {//进程全部序列执行完成
        FinishedProc++;
        proc[RunPoint].p_state = 'F';
        proc[RunPoint].p_endtime = ClockNumber;
        RunPoint = -1; //无进程执行
        NextRunProcess(); //找分派程序去，接着做下一个
    } else { //进行 IO 操作，进入阻塞队列
        proc[RunPoint].p_pos++;
        proc[RunPoint].p_state = 'W';
        proc[RunPoint].p_iotime = proc[RunPoint].p_rserial[proc[RunPoint].p_pos];
        proc[RunPoint].p_next = -1; //标记下，就自己一个进程，没带尾巴一起来；否则，当 p_next 不为-1 时，后面的那一串都是被阻塞者
        n = WaitPoint;
        //是阻塞队列第一个 I/O 进程
        if (n == -1) {
            WaitPoint = RunPoint;
            proc[WaitPoint].p_iotime++; //为了避免一个时钟周期内两次减少，做的处理
        } else {
            do { //放入阻塞队列尾
                if (proc[n].p_next == -1) {
                    proc[n].p_next = RunPoint;
                    break;
                }
                n = proc[n].p_next;
            } while (n != -1);
        }
        RunPoint = -1;
        NextRunProcess();
    }
    return;
}


void IO_Sched() {
    if (WaitPoint == -1) return;// 没有等待 I/O 的进程，直接返回
    proc[WaitPoint].p_iotime--; // 进行 1 个时钟的 I/O 时间
    if (proc[WaitPoint].p_iotime > 0) return;// 还没有完成本次 I/O
    else {
        //进程全部序列执行完成
        if (proc[WaitPoint].p_rserial[0] == proc[WaitPoint].p_pos) {
```

```
                FinishedProc++;
                proc[WaitPoint].p_state = 'F';
                proc[WaitPoint].p_endtime = ClockNumber;
                //阻塞队列更新
                WaitPoint = proc[WaitPoint].p_next;
            } else {
                int i = WaitPoint;
                //进程本身变化
                proc[i].p_state = 'R';
                proc[i].p_pos++;
                proc[i].p_cputime = proc[i].p_rserial[proc[i].p_pos];
                //阻塞队列更新
                WaitPoint = proc[i].p_next;
                proc[i].p_next = -1;
                //插到就绪队列队尾
                if (ReadyPoint == -1) ReadyPoint = i;// 如果当前就绪队列无进程
                else { // 如果就绪队列有进程，放入队列尾
                    int n = ReadyPoint;
                    while (proc[n].p_next != -1)
                        n = proc[n].p_next; //找到原来队伍中的尾巴
                    proc[n].p_next = i; //挂在这个尾巴后面
                }
            }
        }
    }

    void NextRunProcess() {
        int n = ReadyPoint;
        if (n == -1) return;
        if (proc[n].p_starttime == ClockNumber) return;
        RunPoint = ReadyPoint, ReadyPoint = proc[n].p_next;
        proc[n].p_state = 'C';
    }

    void Display_ProcInfo() {
        cout << format("当前系统模拟{}个进程的运行　　时钟：{}", ProcNumber,
    ClockNumber);
        cout << format("就绪指针={}，运行指针={}，阻塞指针={}\n\n", ReadyPoint, RunPoint,
    WaitPoint);
        cout << "......Running Process......\n";
        if (RunPoint != -1) {
            cout << format("NO.{} ID:{},总CPU时间={},剩余CPU时间={},serial:", RunPoint,
    proc[RunPoint].p_pid,
                    proc[RunPoint].p_rserial[proc[RunPoint].p_pos], proc[RunPoint].p_cputime);
```

```
        for (int j = 1; j <= proc[RunPoint].p_rserial[0]; j++)
            cout << format("{} ", proc[RunPoint].p_rserial[j]);
        cout << endl;
    } else cout << "No Process Running !\n";
    int n = ReadyPoint;
    cout << "\n.....Ready Process.....\n";
    while (n != -1) { // 显示就绪进程信息
        cout << format("NO.{} ID:{},第{}个/总时间={},serial:",
            n, proc[n].p_pid, proc[n].p_pos, proc[n].p_rserial[proc[n].p_pos]);
        for (int j = 1; j <= proc[n].p_rserial[0]; j++)
            cout << format("{} ", proc[n].p_rserial[j]);
        n = proc[n].p_next;
        cout << endl;
    }
    n = WaitPoint;
    cout << "\nWaiting Process ..... \n";
    while (n != -1) { // 显示阻塞进程信息
        cout << format("NO.{} ID:{},I/O 执行到序列中的第{}个，总 I/O 时间={}，剩余 I/O
时间={},serial:",
            n, proc[n].p_pid, proc[n].p_pos, proc[n].p_rserial[proc[n].p_pos],
            proc[n].p_iotime);
        for (int j = 1; j <= proc[n].p_rserial[0]; j++)
            cout << format("{} ", proc[n].p_rserial[j]);
        n = proc[n].p_next;
        cout << endl;
    }
    cout << "\n==================== 后备进程 ====================\n";
    for (int i = 0; i < ProcNumber; i++)
        if (proc[i].p_state == 'B') {
            cout << format("NO.{} ID:{},serial:", i, proc[i].p_pid);
            for (int j = 1; j <= proc[i].p_rserial[0]; j++)
                cout << format("{} ", proc[i].p_rserial[j]);
            cout << endl;
        }
    cout << "\n================ 已经完成的进程 ================\n";
    for (int i = 0; i < ProcNumber; i++)
        if (proc[i].p_state == 'F') {
            cout << format("NO.{} ID:{},EndTime={},serial:", i, proc[i].p_pid,
                proc[i].p_endtime);
            for (int j = 1; j <= proc[i].p_rserial[0]; j++)
                cout << format("{} ", proc[i].p_rserial[j]);
            cout << endl;
        }
    cout << endl;
```

```
}

void Statistic() {
	cout << "统计\n";
	for (int i = 0; i < ProcNumber; i++)
		if (proc[i].p_state == 'F') {
			int runTime = 0;
			cout << format("ID:{},StartTime={},EndTime={},serial:", proc[i].p_pid,
proc[i].p_starttime, proc[i].p_endtime);
			for (int j = 1; j <= proc[i].p_rserial[0]; j++) {
				cout << format("{} ", proc[i].p_rserial[j]);
				runTime += proc[i].p_rserial[j] * (j % 2);
			}
			cout << format("(周转：{}, 带权周转：{})", proc[i].p_endtime -
proc[i].p_starttime,
				(proc[i].p_endtime - proc[i].p_starttime) * 1.0 / runTime);
			cout << format("\n");
		}
	cout << format("\n");
}

void Read_Process_Info() {
	ifstream inFile;
	inFile.open("./Process_Info.txt", ios::in);
	char data[500];
	while (true) {
		inFile.getline(data, 500);
		proc[ProcNumber].p_pid = proc[ProcNumber].p_rserial[0] =
proc[ProcNumber].p_starttime = 0;
		int i = 3;
		if (data[0] == '\0') break;
		while (data[i] != '(')
			proc[ProcNumber].p_pid = data[i] - '0' + proc[ProcNumber].p_pid * 10, i++;
		i += 5;
		while (data[i] != ',')
			proc[ProcNumber].p_rserial[0] = data[i] - '0' + proc[ProcNumber].p_rserial[0] *
10, i++;
		i += 11;
		while (data[i] != ')')
			proc[ProcNumber].p_starttime = data[i] - '0' + proc[ProcNumber].p_starttime *
10, i++;
		i += 2;
		for (int j = 1; j <= proc[ProcNumber].p_rserial[0]; j++) {
			int number = 0;
```

```
                for (;; i++) {
                    if (data[i] == ' ') {
                        i++;
                        break;
                    } else number = data[i] - '0' + number * 10;
                }
                proc[ProcNumber].p_rserial[j] = number;
            }
            proc[ProcNumber].p_state = 'B';
            proc[ProcNumber].p_pos = 1;
            proc[ProcNumber].p_endtime = proc[ProcNumber].p_iotime =
proc[ProcNumber].p_next = -1;
            proc[ProcNumber].p_cputime = proc[ProcNumber].p_rserial[1];
            ProcNumber++;
        }
        cout << format("从文件读取了{}个进程数据序列\n", ProcNumber);
        for (int i = 0; i < ProcNumber; i++) {
            cout << format("ID={}(len={},startTime={}):", proc[i].p_pid, proc[i].p_rserial[0],
                proc[i].p_starttime);
            for (int j = 1; j <= proc[i].p_rserial[0]; j++)
                cout << format("{} ", proc[i].p_rserial[j]);
            cout << endl;
        }
        cout << endl;
}
```

3 程序运行结果和分析

3.1 程序运行演示

3.1.1 建立进程调度序列

```
**********************************
1: 建立进程调度数据序列
2: 读进程信息, 执行调度算法
**********************************
Enter your choice (1 ~ 2): 1

建立了5个进程数据序列

ID=857(len=7,startTime=3):2 1 2 1 3 2 1
ID=878(len=1,startTime=1):3
ID=646(len=7,startTime=5):2 2 3 2 1 3 1
ID=416(len=7,startTime=3):3 3 3 2 1 1 1
ID=569(len=6,startTime=5):2 2 1 1 1 1
```

3.1.2 读进程信息，执行调度算法

```
**********************************
1: 建立进程调度数据序列
2: 读进程信息，执行调度算法
**********************************
Enter your choice (1 ~ 2): 2

从文件读取了5个进程数据序列
ID=857(len=7,startTime=3):2 1 2 1 3 2 1
ID=878(len=1,startTime=1):3
ID=646(len=7,startTime=5):2 2 3 2 1 3 1
ID=416(len=7,startTime=3):3 3 3 2 1 1 1
ID=569(len=6,startTime=5):2 2 1 1 1 1

当前系统模拟5个进程的运行    时钟：0就绪指针=-1，运行指针=-1，阻塞指针=-1

......Running Process......
No Process Running !

......Ready Process......

Waiting Process ......

=================== 后备进程 ====================
NO.0 ID:857,serial:2 1 2 1 3 2 1
NO.1 ID:878,serial:3
NO.2 ID:646,serial:2 2 3 2 1 3 1
NO.3 ID:416,serial:3 3 3 2 1 1 1
NO.4 ID:569,serial:2 2 1 1 1 1

================ 已经完成的进程 =================
```

```
当前系统模拟5个进程的运行    时钟：1就绪指针=1，运行指针=-1，阻塞指针=-1

......Running Process......
No Process Running !

......Ready Process......
NO.1 ID:878,第1个/总时间=3,serial:3

Waiting Process ......

=================== 后备进程 ====================
NO.0 ID:857,serial:2 1 2 1 3 2 1
NO.2 ID:646,serial:2 2 3 2 1 3 1
NO.3 ID:416,serial:3 3 3 2 1 1 1
NO.4 ID:569,serial:2 2 1 1 1 1

================ 已经完成的进程 =================
```

```
当前系统模拟5个进程的运行    时钟：15就绪指针=3, 运行指针=0, 阻塞指针=4

......Running Process......
NO.0 ID:857,总CPU时间=2,剩余CPU时间=1,serial:2 1 2 1 3 2 1

......Ready Process......
NO.3 ID:416,第3个/总时间=3,serial:3 3 3 2 1 1 1
NO.2 ID:646,第3个/总时间=3,serial:2 2 3 2 1 3 1

Waiting Process ......
NO.4 ID:569,I/O执行到序列中的第2个, 总I/O时间=2, 剩余I/O时间=2,serial:2 2 1 1 1 1

=================== 后备进程 ====================

================ 已经完成的进程 =================
NO.1 ID:878,EndTime=5,serial:3
```

```
当前系统模拟5个进程的运行    时钟：34就绪指针=-1, 运行指针=-1, 阻塞指针=-1

......Running Process......
No Process Running !

......Ready Process......

Waiting Process ......

=================== 后备进程 ====================

================ 已经完成的进程 =================
NO.0 ID:857,EndTime=30,serial:2 1 2 1 3 2 1
NO.1 ID:878,EndTime=5,serial:3
NO.2 ID:646,EndTime=34,serial:2 2 3 2 1 3 1
NO.3 ID:416,EndTime=31,serial:3 3 3 2 1 1 1
NO.4 ID:569,EndTime=33,serial:2 2 1 1 1 1

统计
ID:857,StartTime=3,EndTime=30,serial:2 1 2 1 3 2 1 (周转：27,带权周转：3.375)
ID:878,StartTime=1,EndTime=5,serial:3 (周转：4,带权周转：1.3333333333333333)
ID:646,StartTime=5,EndTime=34,serial:2 2 3 2 1 3 1 (周转：29,带权周转：4.142857142857143)
ID:416,StartTime=3,EndTime=31,serial:3 3 3 2 1 1 1 (周转：28,带权周转：3.5)
ID:569,StartTime=5,EndTime=33,serial:2 2 1 1 1 1 (周转：28,带权周转：7)
```

3.2 最后一次结果的分析

```
ID:857,StartTime=3,EndTime=30,serial:2 1 2 1 3 2 1 (周转：27,带权周转：3.375)
ID:878,StartTime=1,EndTime=5,serial:3 (周转：4,带权周转：1.3333333333333333)
ID:646,StartTime=5,EndTime=34,serial:2 2 3 2 1 3 1 (周转：29,带权周转：4.142857142857143)
ID:416,StartTime=3,EndTime=31,serial:3 3 3 2 1 1 1 (周转：28,带权周转：3.5)
ID:569,StartTime=5,EndTime=33,serial:2 2 1 1 1 1 (周转：28,带权周转：7)
```

以 ID857 为例，周转时间为 完成时间 - 到达时间 =27，带权周转时间为 周转时间 / 运行时间 =3.375