

**课 程 设 计 报 告**

( 2022--2023年度第二学期)

**课程名称: 操作系统课程设计**

**课设题目:** **SPOOLing假脱机输入输出技术模拟**

**院 系:** **控制与计算机工程学院**

**班 级:** **XXX**

**学 号:** **XXX**

**姓 名:**  **XXX**

**指导教师:** XXX

**设计周数:**   **一周**

**成 绩:**

**2023年6月23日**

目录

[一、 需求分析 3](#_Toc138173371)

[1. 背景介绍 3](#_Toc138173372)

[2. 功能需求 3](#_Toc138173373)

[3. 界面需求 3](#_Toc138173374)

[4. 类图 4](#_Toc138173375)

[二、 整体功能及设计 4](#_Toc138173376)

[1. 用例图 4](#_Toc138173377)

[(1) 进程调度流程图 5](#_Toc138173378)

[(2) 用户进程流程图 6](#_Toc138173379)

[(3) SPOOLing输出进程流程图 7](#_Toc138173380)

[(4) 用户进程状态转换图 7](#_Toc138173381)

[(5) SPOOLing输出进程状态转换图 8](#_Toc138173382)

[三、 编程实现 8](#_Toc138173383)

[1. 用户进程 8](#_Toc138173384)

[2. SPOOLing输出进程 9](#_Toc138173385)

[3. 随机调度 11](#_Toc138173386)

[四、 使用说明 14](#_Toc138173387)

[1. 初始界面 14](#_Toc138173388)

[2. 出错处理 14](#_Toc138173389)

[3. 开始运行 15](#_Toc138173390)

[4. 完成作业 17](#_Toc138173391)

[五、 结果分析 18](#_Toc138173392)

[1. 运行效率分析 18](#_Toc138173393)

[2. 进程状态分析 18](#_Toc138173394)

[3. 调度信息分析 19](#_Toc138173395)

[4. 输出文件分析 20](#_Toc138173396)

[5. 结论 20](#_Toc138173397)

# 需求分析

## 背景介绍

在计算机系统中，经常需要将大量数据进行输入/输出操作，由于外设设备的速度远远低于计算机主存储器的速度，因此，存在大量的输入/输出等待时间。为了提高计算机系统的效率，可采用输入/输出缓冲和缓冲技术，其中，SPOOLing技术是一种常用的技术之一。

Spooling技术意为Simultaneous Peripheral Operations On-line（联机同时操作外设）。Spooling技术的主要思想是将输入/输出请求缓存在磁盘上，以避免对主存储器造成过大的压力。该技术主要应用于操作系统的文件管理、输出管理和打印管理等方面。

Spooling技术按照先来先服务（FIFO）的方式排队等待设备处理。这些请求按照顺序排队，第一个请求排在队列的头部，每个后续请求排在它前面的请求之后。

Spooling技术的优点是可以提高计算机系统的效率。同时，Spooling技术还可以避免在不同用户之间的I/O竞争问题，并且可以支持多任务并发处理，提高系统的效率和响应速度。

Spooling技术的缺点是需要大量的磁盘空间来存储缓存的请求，而且需要保证磁盘输入/输出速度与计算机处理速度相当，否则会导致系统性能下降。此外，如果磁盘出现故障，会导致请求丢失，影响系统的稳定性。

总之，Spooling技术是一项重要的操作系统技术。为了深入了解SPOOLing技术的具体工作原理，特此设计该SPOOLing输出模拟系统。

## 功能需求

系统需要提供输出井的管理功能，包括输出井的分配和释放，以及输出井的状态的监控和控制等。

系统需要提供请求输出块的管理功能，包括请求输出块的分配和释放，以及请求输出块的状态的监控和控制等。

系统需要支持多任务并发处理，能够处理多个用户的请求，并保证请求的公平性和优先级。

系统需要提供用户接口，用户能够通过图形界面接口向系统提交文件数量，并能够查看文件的输出状态和输出结果。

系统需要支持异常处理，能够处理各种异常情况，如输入文件数错误等错误，避免系统由于输入错误而崩溃。

系统需要实现SPOOLing输出算法，模拟SPOOLing输出服务程序、输出用户进程、SPOOLing输出进程的工作方式。

## 界面需求

系统需要提供简单易用的用户界面，方便用户进行操作和管理。

系统界面应当布局合理，结构清晰，使用户能清楚的观察SPOOLing输出模拟系统的工作状态。

用户界面需要具有友好的操作性能和视觉效果，能够提高用户体验，利用不同颜色标识数据状态，加强界面可视化效果。

## 类图

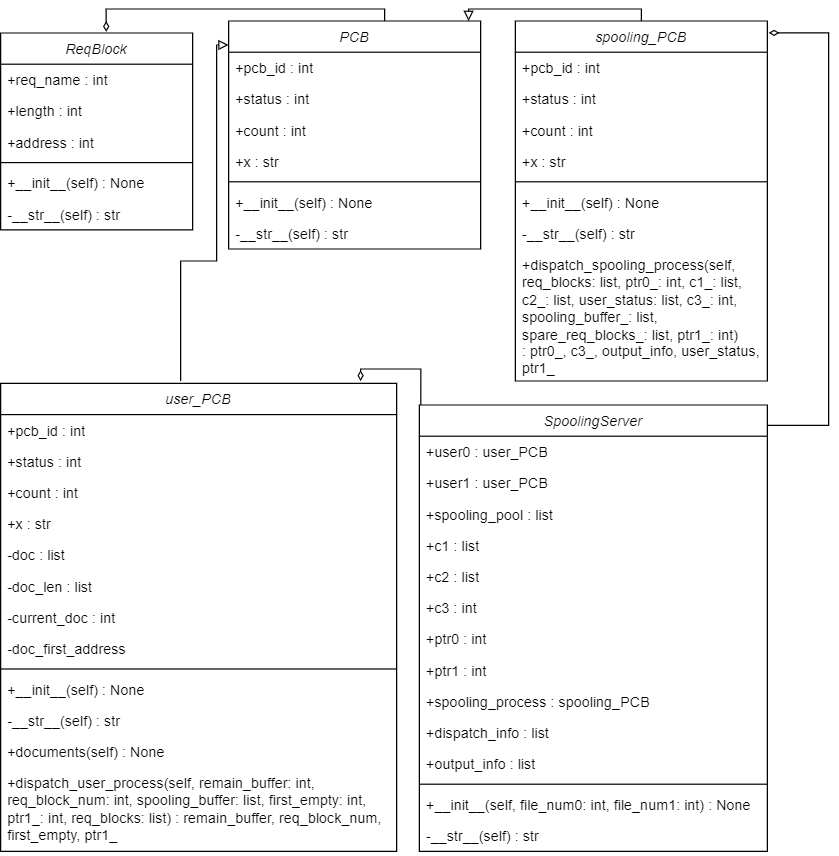


图1-1 SPOOLing输出模拟类图

# 整体功能及设计

## 用例图

整体功能共分为四个部分（如图2-1所示）：SPOOLing服务程序、SPOOLing输出进程和两个用户进程。

SPOOLing服务程序负责进行SPOOLing输出进程和两个用户进程的进程调度，以及输出井、输出请求块等资源的管理。

用户进程负责将文件输出到输出井，并更新输出井相关参数的状态，同时在将文件输出到输出井后，申请输出请求块，并在申请输出请求块后更新输出请求块相关的状态。

SPOOLing输出进程负责根据输出请求块的内容将输出井中的数据输出到打印机或显示器上，并释放输出井空间和输出请求块。

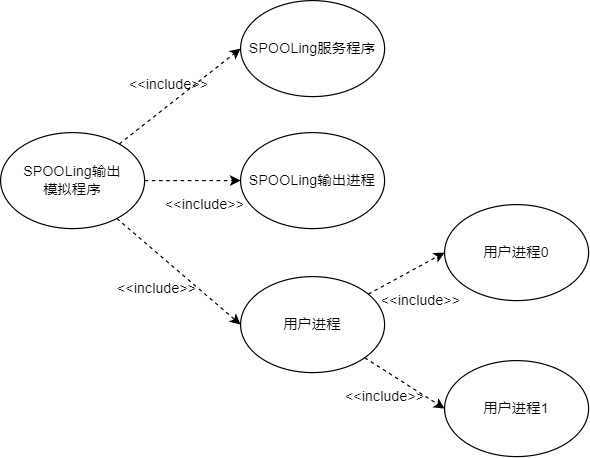


图2-1 用例图

### 进程调度流程图

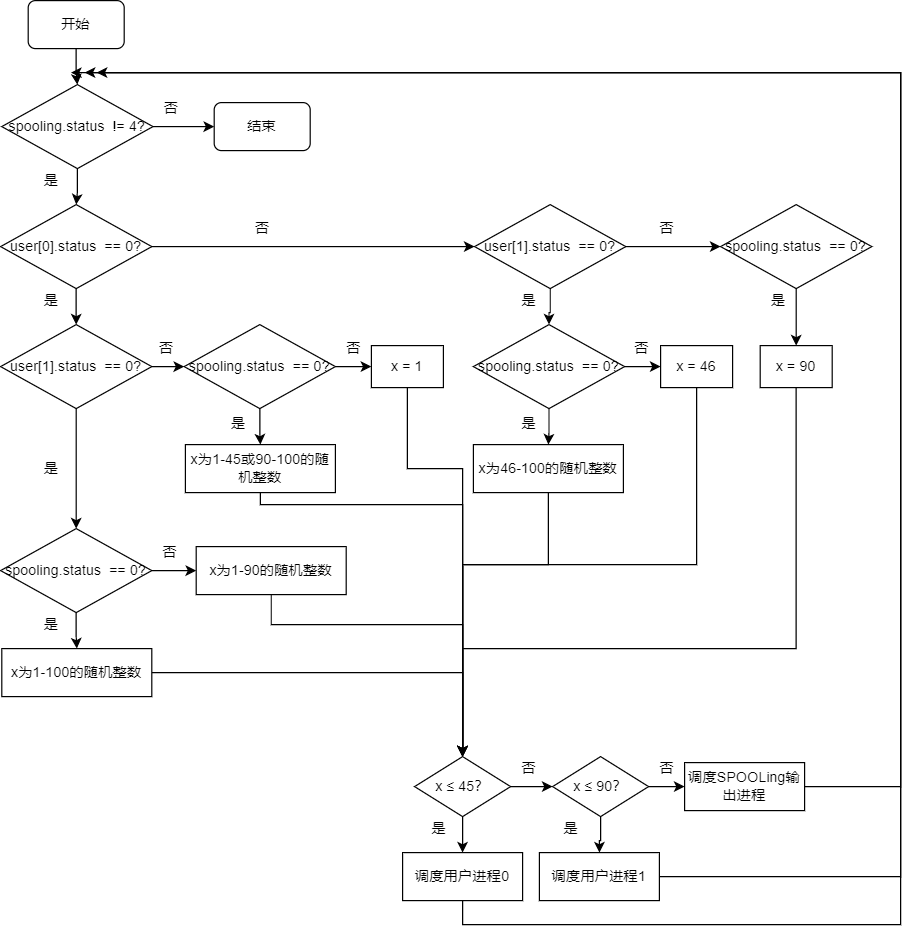


图2-2 进程调度流程图

### 用户进程流程图

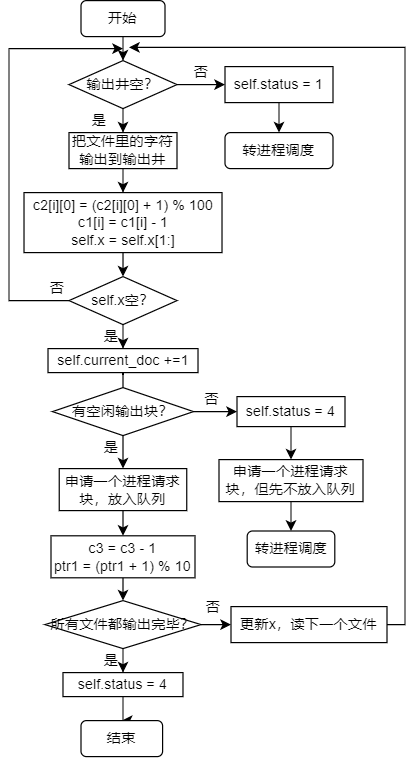


图2-3 用户进程流程图

### SPOOLing输出进程流程图

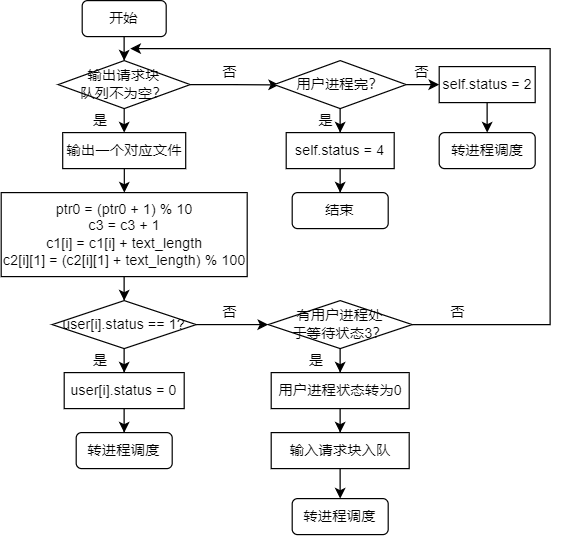


图2-4 SPOOLing输出进程流程图

### 用户进程状态转换图

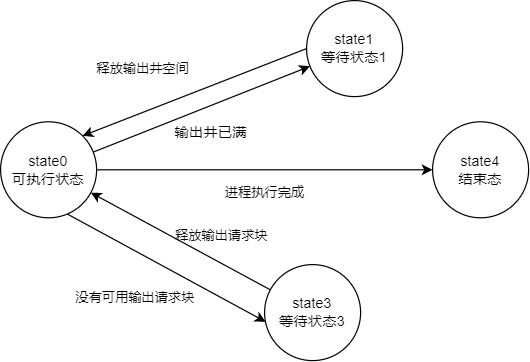


图2-5 用户进程状态转换图

### SPOOLing输出进程状态转换图

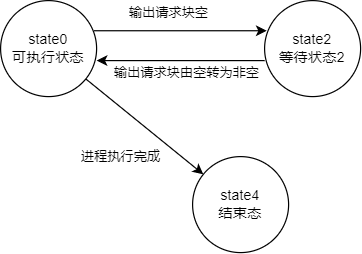


图2-6 SPOOLing输出进程状态转换图

# 编程实现

## 用户进程

def dispatch\_user\_process(self, remain\_buffer: int, req\_block\_num: int, spooling\_buffer: list, first\_empty: int,  
 ptr1\_: int, req\_blocks: list):  
 *"""  
 调度用户进程  
 :param remain\_buffer:输出井剩余容量 c1[i]  
 :param req\_block\_num:空闲输出请求块数 c3  
 :param spooling\_buffer:输出井 spooling\_pool[i]  
 :param first\_empty:第一个可用空缓冲指针 c2[i][0]  
 :param ptr1\_:空闲请求输出块指针 ptr1  
 :param req\_blocks:输出请求块req\_blocks  
 :return:c1[i], c3, c2[i][0], ptr1\_  
 """* while True:  
 while True:  
 if remain\_buffer == 0: *# 输出井满* self.status = 1  
 return remain\_buffer, req\_block\_num, first\_empty, ptr1\_  
 else: *# 把文件里的字符输出到输出井* spooling\_buffer[first\_empty] = self.x[0]  
 if len(self.doc\_first\_address) == self.current\_doc:  
 self.doc\_first\_address.append(first\_empty)  
 first\_empty = (first\_empty + 1) % 100  
 remain\_buffer -= 1  
 self.x = self.x[1:]  
 if len(self.x) == 0:  
 break  
  
 self.current\_doc += 1  
  
 if req\_block\_num == 0: *# 没有空闲输出请求块* self.status = 3  
 self.spare\_req\_block.req\_name = self.pcb\_id  
 self.spare\_req\_block.length = self.doc\_len[self.current\_doc - 1]  
 self.spare\_req\_block.address = self.doc\_first\_address[self.current\_doc - 1]  
 if self.current\_doc == self.count: *# 所有文件都输出完毕* self.status = 4  
 return remain\_buffer, req\_block\_num, first\_empty, ptr1\_  
 else:  
 self.x = self.doc[self.current\_doc] *# 更新x，读取下一个文件* return remain\_buffer, req\_block\_num, first\_empty, ptr1\_  
  
 req\_blocks[ptr1\_].req\_name = self.pcb\_id  
 req\_blocks[ptr1\_].length = self.doc\_len[self.current\_doc - 1]  
 req\_blocks[ptr1\_].address = self.doc\_first\_address[self.current\_doc - 1]  
 req\_block\_num -= 1  
 ptr1\_ = (ptr1\_ + 1) % 10  
 if self.current\_doc == self.count: *# 所有文件都输出完毕* self.status = 4  
 return remain\_buffer, req\_block\_num, first\_empty, ptr1\_  
 else:  
 self.x = self.doc[self.current\_doc] *# 更新x，读取下一个文件*

## SPOOLing输出进程

def dispatch\_spooling\_output\_process(self, req\_blocks: list, ptr0\_: int, c1\_: list, c2\_: list, user\_status: list,  
 c3\_: int, spooling\_buffer\_: list, spare\_req\_blocks\_: list, ptr1\_: int):  
 *"""  
 输出井进程  
 :param req\_blocks:输出请求块req\_blocks  
 :param ptr0\_:要输出的第一个请求输出块指针  
 :param c1\_:输出井剩余容量 [c1[0], c1[1]]  
 :param c2\_:输出井使用情况 [第一个可用空缓冲指针, 第一个满缓冲指针]  
 :param user\_status:[用户进程0状态, 用户进程1状态]  
 :param c3\_:输出请求块数  
 :param spooling\_buffer\_:输出井[spooling\_pool0, spooling\_pool1]  
 :param spare\_req\_blocks\_:备用输出请求块[spare\_req\_block0, spare\_req\_block1]  
 :param ptr1\_:空闲请求输出块指针  
 :return:ptr0\_, c3\_, output\_info, user\_status, ptr1\_  
 """*  
output\_info = []  
 while True:  
 if c3\_ == 10: *# 输出请求块满* if user\_status[0] == 4 and user\_status[1] == 4:  
 self.status = 4  
 else:  
 self.status = 2  
 return ptr0\_, c3\_, output\_info, user\_status, ptr1\_  
  
 user\_process\_id = req\_blocks[ptr0\_].req\_name  
 text\_length = req\_blocks[ptr0\_].length  
 text\_address = req\_blocks[ptr0\_].address  
 text\_end = (text\_address + text\_length) % 100  
 if text\_address < text\_end:  
 text = spooling\_buffer\_[user\_process\_id][text\_address:text\_end]  
 else:  
 text = spooling\_buffer\_[user\_process\_id][text\_address:] + spooling\_buffer\_[user\_process\_id][:text\_end]  
 texts = ""  
 for i in range(len(text)):  
 texts += text[i]  
 output\_info.append("用户进程%d输出文件：%s" % (user\_process\_id, texts))  
 ptr0\_ = (ptr0\_ + 1) % 10  
 c3\_ += 1  
 c1\_[user\_process\_id] += text\_length  
 c2\_[user\_process\_id][1] = (c2\_[user\_process\_id][1] + text\_length) % 100  
 if user\_status[user\_process\_id] == 1:  
 user\_status[user\_process\_id] = 0  
 return ptr0\_, c3\_, output\_info, user\_status, ptr1\_  
 if user\_status[0] == 3 or user\_status[1] == 3:  
 if (user\_status[0] == 3 and user\_status[1] != 3) or (user\_status[0] == 3 and user\_status[1] == 3):  
 user\_status[0] = 0  
 req\_blocks[ptr1\_].req\_name = 0  
 req\_blocks[ptr1\_].length = spare\_req\_blocks\_[0].length  
 req\_blocks[ptr1\_].address = spare\_req\_blocks\_[0].address  
 ptr1\_ = (ptr1\_ + 1) % 10  
 c3\_ -= 1  
 elif user\_status[0] != 3 and user\_status[1] == 3:  
 user\_status[1] = 0  
 req\_blocks[ptr1\_].req\_name = 1  
 req\_blocks[ptr1\_].length = spare\_req\_blocks\_[1].length  
 req\_blocks[ptr1\_].address = spare\_req\_blocks\_[1].address  
 ptr1\_ = (ptr1\_ + 1) % 10  
 c3\_ -= 1  
 return ptr0\_, c3\_, output\_info, user\_status, ptr1\_

## 随机调度

def random\_dispatch(self, first\_time\_append: list):if self.user0.status == 0 and self.user1.status == 0 and self.spooling\_process.status == 0:  
 x = random.randint(1, 100)  
 elif self.user0.status != 0 and self.user1.status == 0 and self.spooling\_process.status == 0:  
 x = random.randint(46, 100)  
 elif self.user0.status == 0 and self.user1.status != 0 and self.spooling\_process.status == 0:  
 x = random.randint(46, 100)  
 if 46 <= x <= 90:  
 x -= 45  
 elif self.user0.status == 0 and self.user1.status == 0 and self.spooling\_process.status != 0:  
 x = random.randint(1, 90)  
 elif self.user0.status == 0 and self.user1.status != 0 and self.spooling\_process.status != 0:  
 x = 1  
 elif self.user0.status != 0 and self.user1.status == 0 and self.spooling\_process.status != 0:  
 x = 46  
 elif self.user0.status != 0 and self.user1.status != 0 and self.spooling\_process.status == 0:  
 x = 91  
 else:  
 x = random.randint(1, 100)  
  
 if x <= 45:  
 if self.user0.status == 0: *# 执行用户进程0* self.c1[0], self.c3, self.c2[0][0], self.ptr1 = self.user0.dispatch\_user\_process(self.c1[0], self.c3, self.spooling\_pool[0], self.c2[0][0], self.ptr1, self.req\_blocks)  
 if self.spooling\_process.status != 0:  
 self.spooling\_process.status = 0  
 self.dispatch\_info.append("执行用户进程0，输出字符到输出井，处于执行状态0\n")  
 if self.user0.status == 1: *# 输出井0满* self.dispatch\_info.append("用户进程0输出井0满，处于等待状态1\n")  
 if self.user0.status == 3: *# 没有空闲输出请求块* self.dispatch\_info.append("用户进程0没有空闲输出请求块，处于等待状态3\n")  
 if self.user0.status == 4: *# 执行结束* if first\_time\_append[0]:  
 self.dispatch\_info.append("用户进程0执行结束，处于终止状态4\n")  
 first\_time\_append[0] = False  
 elif x <= 90:  
 if self.user1.status == 0:  
 self.c1[1], self.c3, self.c2[1][0], self.ptr1 = self.user1.dispatch\_user\_process(self.c1[1], self.c3, self.spooling\_pool[1], self.c2[1][0],  
self.ptr1, self.req\_blocks)  
 if self.spooling\_process.status != 0:  
 self.spooling\_process.status = 0  
 self.dispatch\_info.append("执行用户进程1，输出字符到输出井，处于执行状态0\n")  
 if self.user1.status == 1:  
 self.dispatch\_info.append("用户进程1输出井1满，处于等待状态1\n")  
 if self.user1.status == 3:  
 self.dispatch\_info.append("用户进程1没有空闲输出请求块，处于等待状态3\n")  
 if self.user1.status == 4:  
 if first\_time\_append[1]:  
 self.dispatch\_info.append("用户进程1执行结束，处于终止状态4\n")  
 first\_time\_append[1] = False  
 elif x <= 100:  
 if self.spooling\_process.status == 0:  
 self.ptr0, self.c3, opif, us, self.ptr1 = self.spooling\_process.dispatch\_spooling\_output\_process(  
 self.req\_blocks,  
 self.ptr0,  
 self.c1,  
 self.c2,  
 [self.user0.status, self.user1.status],  
 self.c3,  
 self.spooling\_pool,  
 [self.user0.spare\_req\_block, self.user1.spare\_req\_block],  
 self.ptr1)  
 self.user0.status = us[0]  
 self.user1.status = us[1]  
 for i in range(len(opif)):  
 self.output\_info.append(opif[i])  
 self.dispatch\_info.append("SPOOLing执行进程处于执行状态0\n")  
 if self.spooling\_process.status == 2:  
 self.dispatch\_info.append("请求输出块为空，SPOOLing执行进程处于等待状态2\n")  
 if self.spooling\_process.status == 4:  
 self.dispatch\_info.append("SPOOLing执行进程已经完成，处于终止状态4\n")  
 else:  
 pass

# 使用说明

## 初始界面

点击运行跳转到初始界面（如图4-1所示）。

界面有1个输入模块，输入对应的文件数后，点击“确定”按钮，开始SPOOLing输出模拟程序。

7个输出模块，分别是：输出井0和输出井1模块、输出井剩余空间和使用情况模块、输出请求块指示模块、进程状态模块、用户进程0文件和用户进程1文件模块、输出文件模块和调度信息模块均为空。

其中：

1. 输出井0和输出井1模块：用于显示输出井的内容，初始为空。
2. 用户进程0文件和用户进程1文件模块：用于展示根据输入用户0文件数和用户1文件数随机生成的用户进程文件。
3. 输出请求块指示模块：用于指示输出请求块状态，初始值如图4-1所示。
4. 进程状态模块：用于指示进程状态，用户进程0和用户进程1处于执行状态，用绿色标识。SPOOLing输出进程由于输出请求块为空，处于等待状态2，用橘色标识。
5. 输出井剩余空间和使用情况模块：用于标识输出井的使用情况，初始值如图4-1所示。
6. 输出文件模块：用于显示输出的文件，初始为空。
7. 调度信息模块：用于记录调度信息，初始为空。

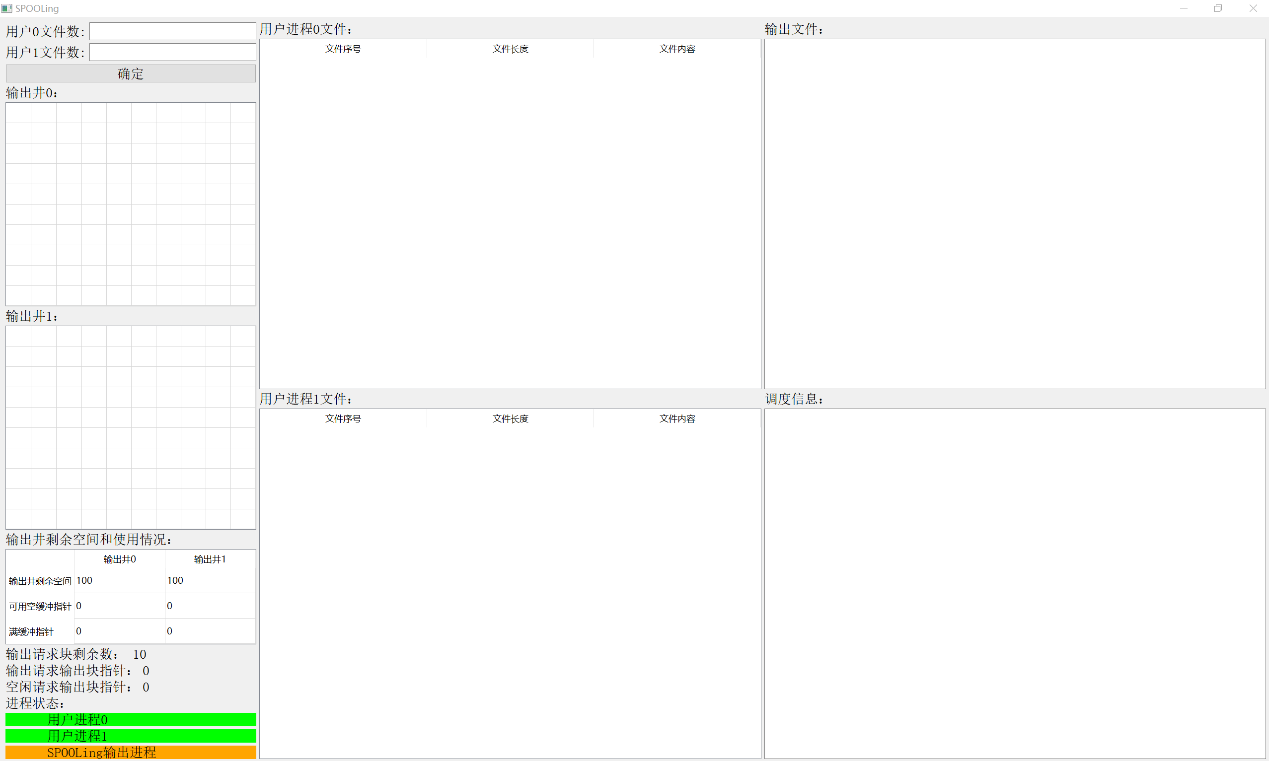


图3-1 初始界面

## 出错处理

在输入错误的文件数或者未输入文件数时，会弹窗提示输入正确的文件数，防止程序由于输入错误导致崩溃。

当出现该弹窗后，需要点击“OK”按钮或者“×”按钮关闭弹窗，输入正确的文件数以正常运行程序。

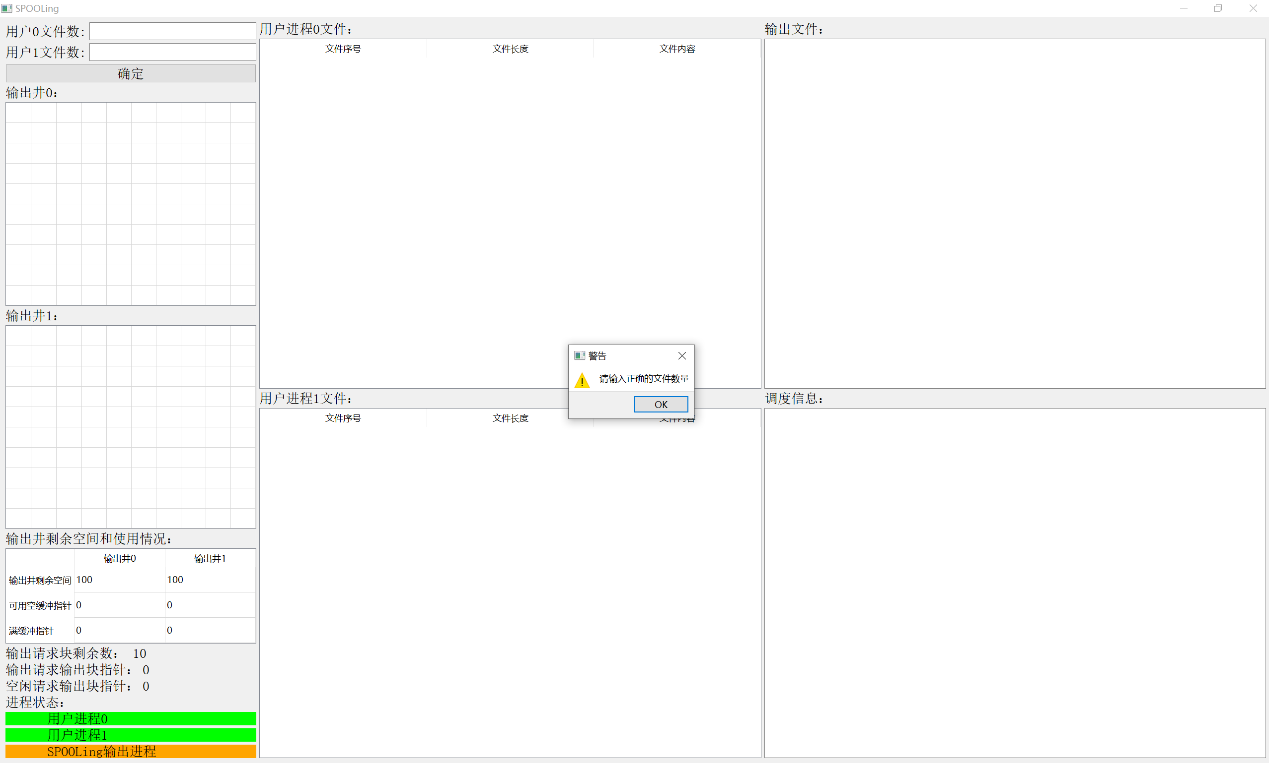


图3-2 出错处理

## 开始运行

程序运行时，首先随机生成文件信息，显示在用户进程0文件和用户进程1文件模块。

同时输出井0和输出井1模块会动态显示输出井的内容，当前有效数据由青色标识，其余数据用白色标识。

输出井模块下方的输出井剩余空间和使用情况模块、输出请求块指示模块会根据输出井和输出请求块状态实时更新，本轮调度更新的数据会用红色标识，未发生变化的数据会用黑色显示。

进程状态模块用颜色标识，运行过程中：

1. 绿色（见图3-3）用于标识执行状态，即status=0，表示进程处于就绪状态或者执行状态；
2. 红色（见图3-4）用于标识等待状态1，即status=1，表示用户进程对应的输出井满；
3. 橘色（见图3-1）用于标识等待状态2，即status=2，表示请求输出块为空；
4. 黄色（见图3-5）用于标识等待状态3，即status=3，标识请求输出块为满；
5. 青色（见图3-6）用于标识完成状态，即status=4，表示进程已经完成。

输出文件模块会实时输出用户需要输出的文件，以文本框的形式显示在窗口中。

调度信息会实时输出并保存用户进程和SPOOLing输出进程调度的信息，用以当时查看和调度后检查。

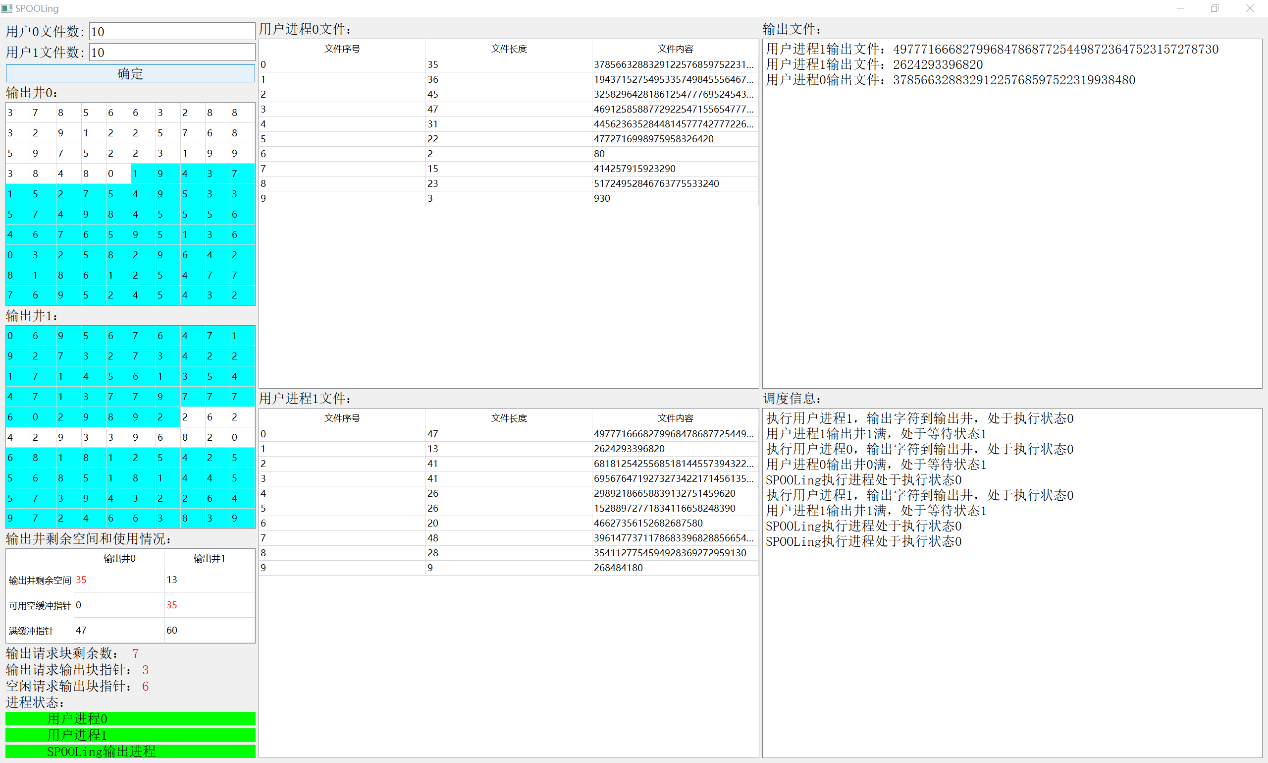


图3-3 正常运行

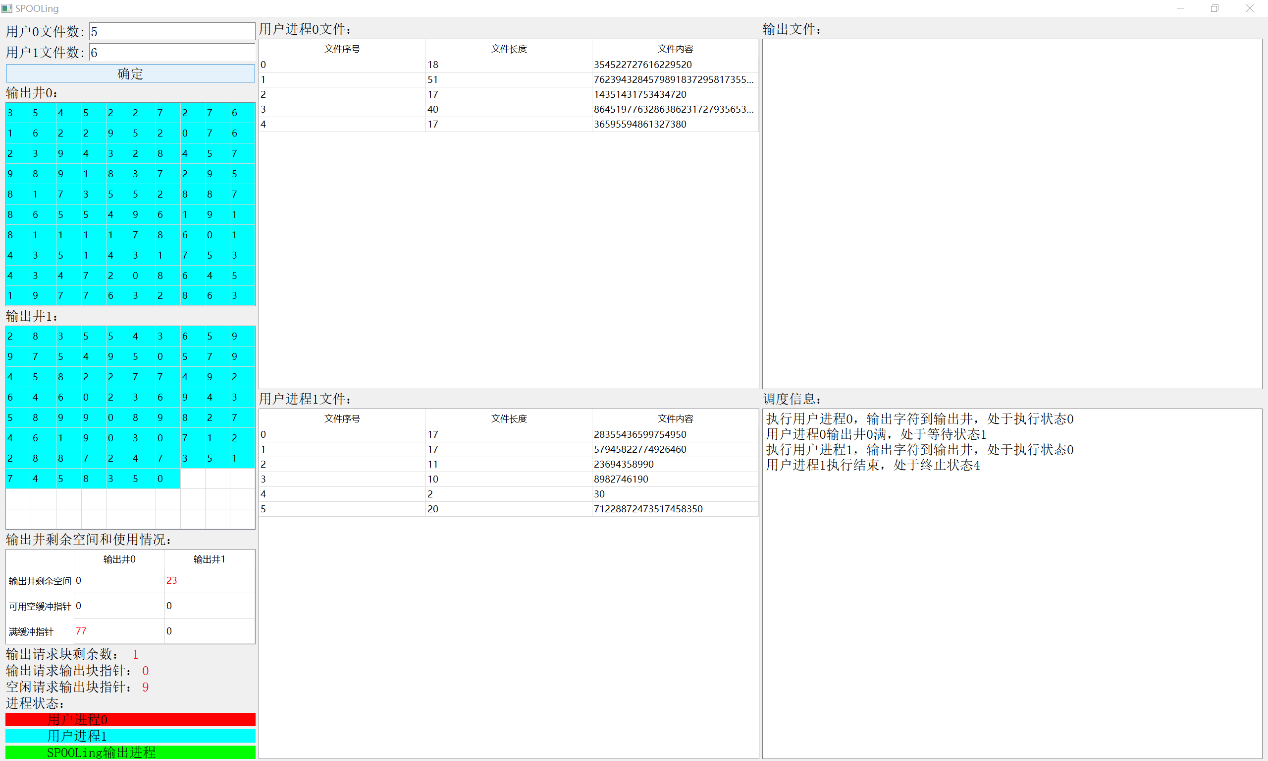


图3-4 用户进程0由于输出井已满而阻塞

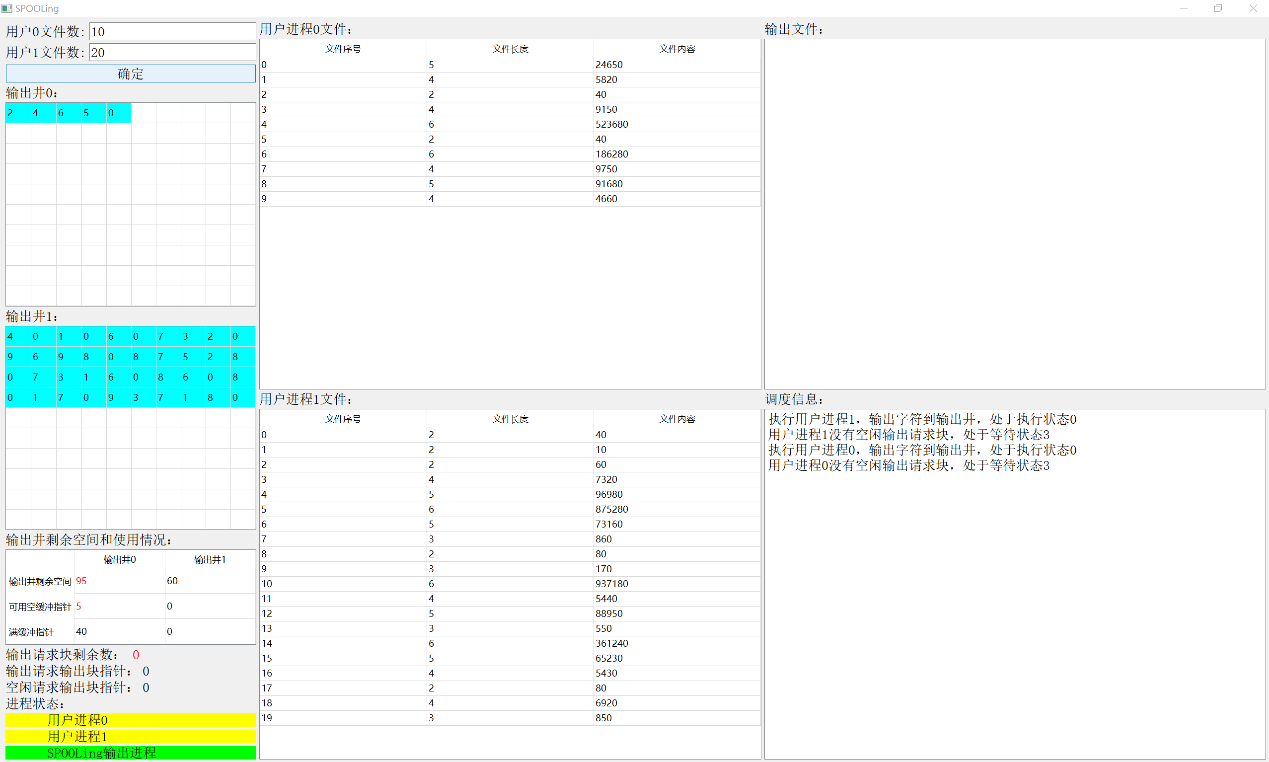


图3-5 用户进程0和用户进程1由于输出请求块已满而阻塞

## 完成作业

作业完成时，系统会进行弹窗，提示作业已经完成，通过点击“OK”按钮或者“×”按钮关闭弹窗。

关闭弹窗后，可选择重新输入用户文件数点击“确定”按钮，来进行新一轮的SPOOLing输出模拟，也可以选择点击右上角的“×”按钮关闭程序，完成模拟。

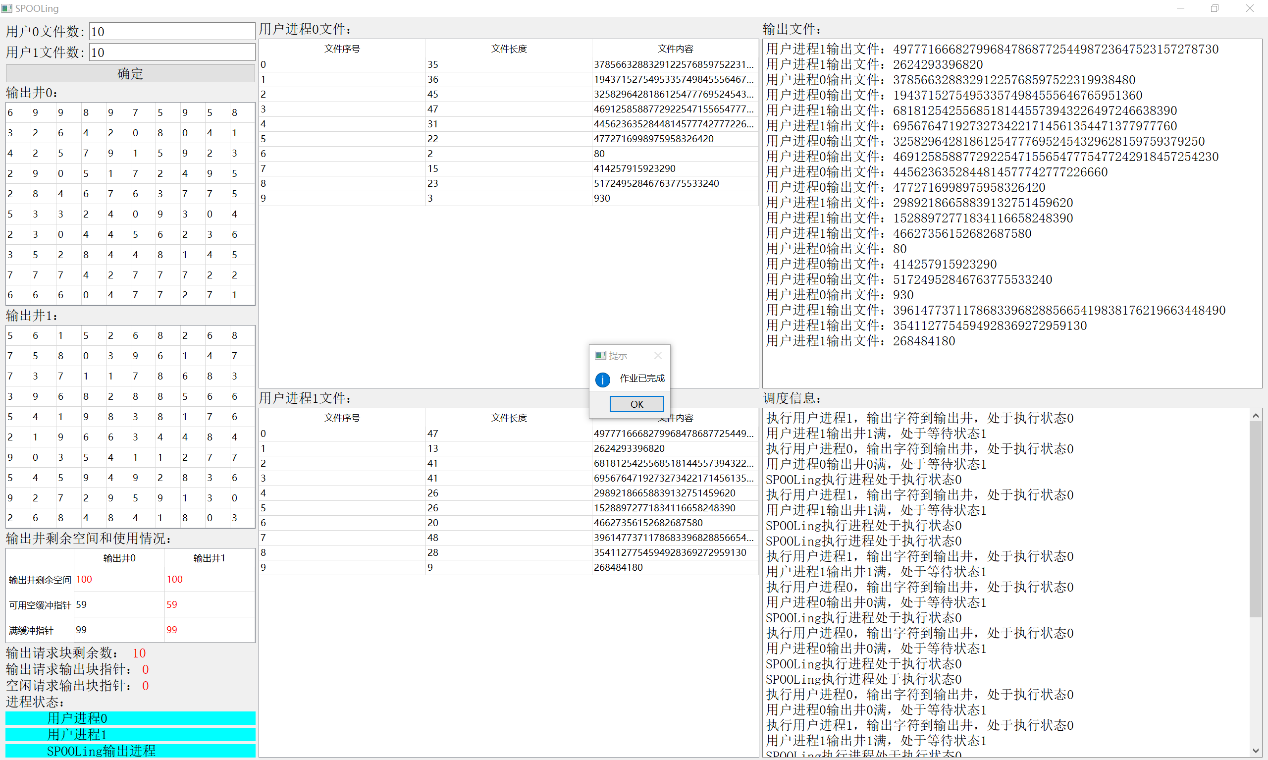


图3-6 所有进程都已经完成，本次模拟结束

# 结果分析

## 运行效率分析

在运行过程中，每进行一次进程调度，对界面进行一次刷新，在刷新后等待一秒，以便用户观察更新后的状态。当输入的文件数较小、随机生成的文件长度较小时，SPOOLing输出模拟系统的调度次数较少，模拟时间较短，而随着文件数的增和和随机生成的文件长度增大时，SPOOLing输出模拟系统的模拟时间会随着调度次数的增加而显著增加。

## 进程状态分析

当设置的文件长度较短且用户输入的文件数量适中时，较容易出现用户进程和SPOOLing输出进程均处于就绪或执行状态的情况。

当设置文件长度较接近100时（如图5-1所示），用户进程较容易处于由于输出井已满而导致的等待状态1。

当设置的文件长度远小于100且用户输入的文件数较大时（如图5-2所示），用户进程较容易处于由于缺少输出请求块而导致的等待状态3。

由于随机调度算法本身的原因，用户进程0和用户进程1被调度的概率远大于SPOOLing输出进程，所以SPOOLing输出进程很少出现由于输出请求块为空而导致的等待状态2。

注意：

1. 当用户输入的文件数量之和小于等于10时，不会发生缺少输出请求块的情况，即用户进程0和用户进程1不可能处于等待状态3。
2. 在上述条件下，如果每个用户进程设置的随机生成的文件长度也足够小，即小于100/文件数量时，不会发生输出井满的情况，即用户进程0和用户进程1不可能处于等待状态1。

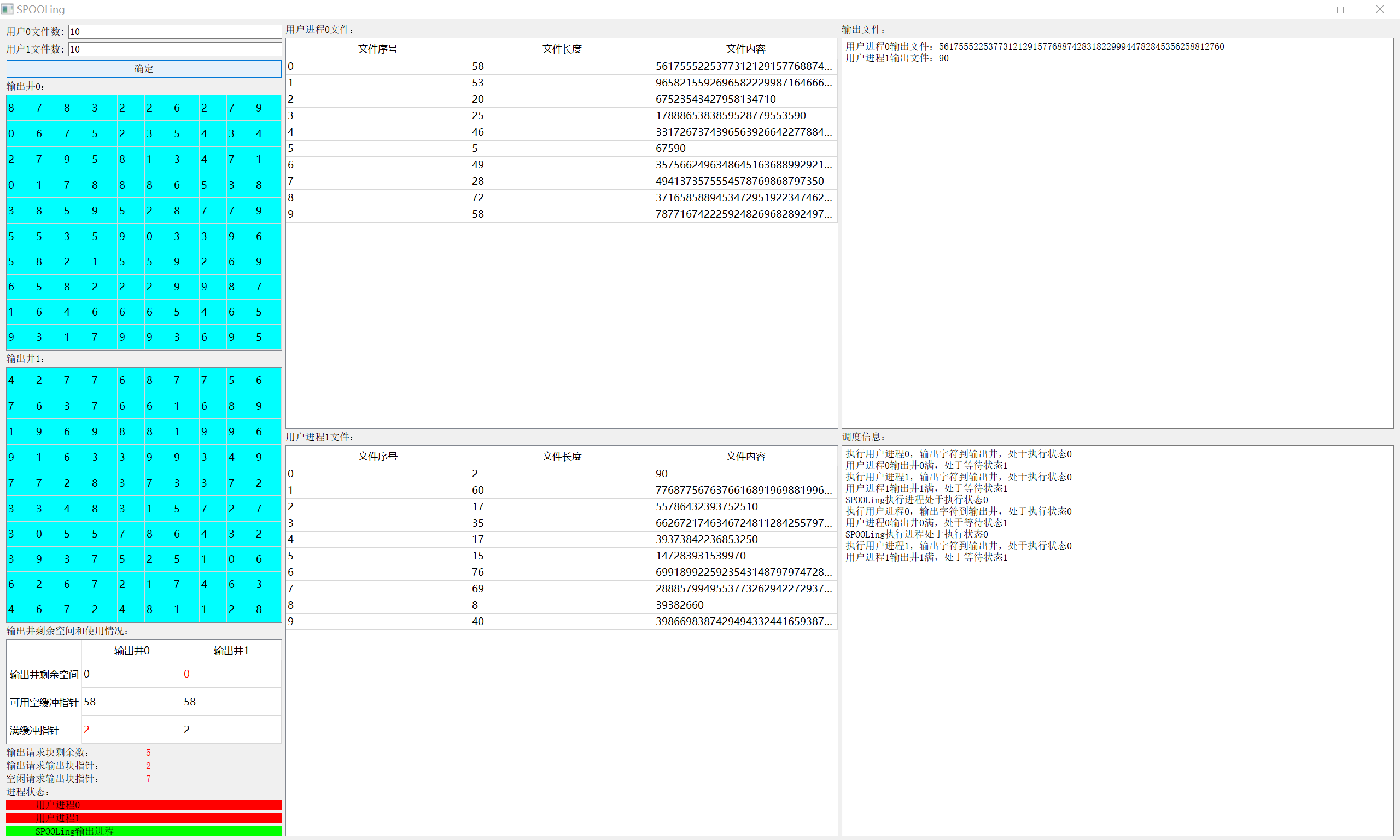


图5-1 文件长度较接近100时

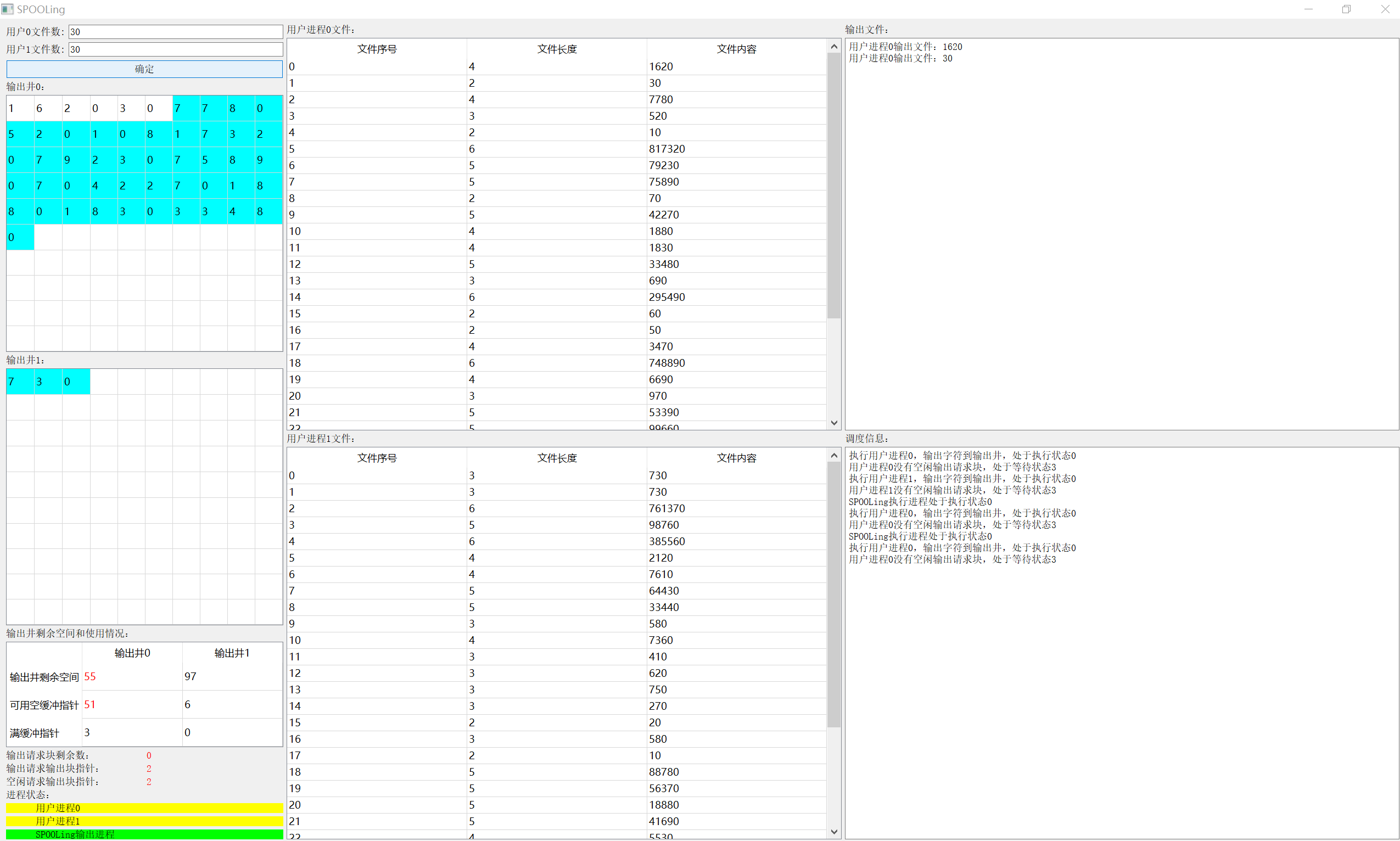


图5-2 设置的文件长度远小于100且用户输入的文件数较大时

## 调度信息分析

文件长度较小时，调度用户进程0、调度用户进程1和调度SPOOLing输出进程的概率在初始情况下分别为45：45：10，而在调度过程中，随着时间的推移，由于用户进程在初始过程中被调度的概率较大，有某一个用户进程会较先处于完成状态，从而导致调度SPOOLing输出进程的概率较大，大于开始时的调度概率（如图5-3所示）。

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0没有空闲输出请求块，处于等待状态3

执行用户进程1，输出字符到输出井，处于执行状态0

用户进程1没有空闲输出请求块，处于等待状态3

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0没有空闲输出请求块，处于等待状态3

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0没有空闲输出请求块，处于等待状态3

SPOOLing执行进程处于执行状态0

SPOOLing执行进程处于执行状态0

SPOOLing执行进程处于执行状态0

请求输出块为空，SPOOLing执行进程处于等待状态2

执行用户进程1，输出字符到输出井，处于执行状态0

用户进程1没有空闲输出请求块，处于等待状态3

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0没有空闲输出请求块，处于等待状态3

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0没有空闲输出请求块，处于等待状态3

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0没有空闲输出请求块，处于等待状态3

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0没有空闲输出请求块，处于等待状态3

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0没有空闲输出请求块，处于等待状态3

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0没有空闲输出请求块，处于等待状态3

SPOOLing执行进程处于执行状态0

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0没有空闲输出请求块，处于等待状态3

执行用户进程1，输出字符到输出井，处于执行状态0

用户进程1没有空闲输出请求块，处于等待状态3

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0没有空闲输出请求块，处于等待状态3

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0没有空闲输出请求块，处于等待状态3

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0没有空闲输出请求块，处于等待状态3

SPOOLing执行进程处于执行状态0

SPOOLing执行进程处于执行状态0

执行用户进程1，输出字符到输出井，处于执行状态0

用户进程1没有空闲输出请求块，处于等待状态3

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0没有空闲输出请求块，处于等待状态3

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0没有空闲输出请求块，处于等待状态3

SPOOLing执行进程处于执行状态0

SPOOLing执行进程处于执行状态0

SPOOLing执行进程处于执行状态0

请求输出块为空，SPOOLing执行进程处于等待状态2

执行用户进程1，输出字符到输出井，处于执行状态0

用户进程1没有空闲输出请求块，处于等待状态3

SPOOLing执行进程处于执行状态0

SPOOLing执行进程处于执行状态0

请求输出块为空，SPOOLing执行进程处于等待状态2

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0执行结束，处于终止状态4

执行用户进程1，输出字符到输出井，处于执行状态0

用户进程1执行结束，处于终止状态4

SPOOLing执行进程处于执行状态0

SPOOLing执行进程已经完成，处于终止状态4

图5-3 调度信息1

在文件长度较大时，较容易在消耗较少的输出请求块的状态下，发生输出井已满的情况。此时需要调度SPOOLing输出进程以释放输出井空间，而调度SPOOLing输出进程后，输出井的空间被释放，由于调度算法本身的原因，调度用户进程的概率较大。综合以上原因，会出现用户进程和SPOOLing输出进程交替调度的情况（如图5-4所示）。

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0输出井0满，处于等待状态1

执行用户进程1，输出字符到输出井，处于执行状态0

用户进程1输出井1满，处于等待状态1

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0输出井0满，处于等待状态1

SPOOLing执行进程处于执行状态0

执行用户进程1，输出字符到输出井，处于执行状态0

用户进程1输出井1满，处于等待状态1

SPOOLing执行进程处于执行状态0

执行用户进程1，输出字符到输出井，处于执行状态0

用户进程1输出井1满，处于等待状态1

SPOOLing执行进程处于执行状态0

执行用户进程1，输出字符到输出井，处于执行状态0

用户进程1输出井1满，处于等待状态1

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0输出井0满，处于等待状态1

SPOOLing执行进程处于执行状态0

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0输出井0满，处于等待状态1

执行用户进程1，输出字符到输出井，处于执行状态0

用户进程1输出井1满，处于等待状态1

SPOOLing执行进程处于执行状态0

执行用户进程1，输出字符到输出井，处于执行状态0

用户进程1输出井1满，处于等待状态1

SPOOLing执行进程处于执行状态0

执行用户进程1，输出字符到输出井，处于执行状态0

用户进程1输出井1满，处于等待状态1

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0输出井0满，处于等待状态1

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0输出井0满，处于等待状态1

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0输出井0满，处于等待状态1

SPOOLing执行进程处于执行状态0

执行用户进程1，输出字符到输出井，处于执行状态0

用户进程1输出井1满，处于等待状态1

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0输出井0满，处于等待状态1

SPOOLing执行进程处于执行状态0

执行用户进程0，输出字符到输出井，处于执行状态0

用户进程0执行结束，处于终止状态4

SPOOLing执行进程处于执行状态0

执行用户进程1，输出字符到输出井，处于执行状态0

用户进程1执行结束，处于终止状态4

SPOOLing执行进程处于执行状态0

SPOOLing执行进程已经完成，处于终止状态4

图5-4 调度信息2

## 输出文件分析

由于调度用户进程0和调度用户进程1的概率相等，所以输出文件（如图5-5所示）的时候，优先输出用户进程0和用户进程1的文件的概率也是相等的。

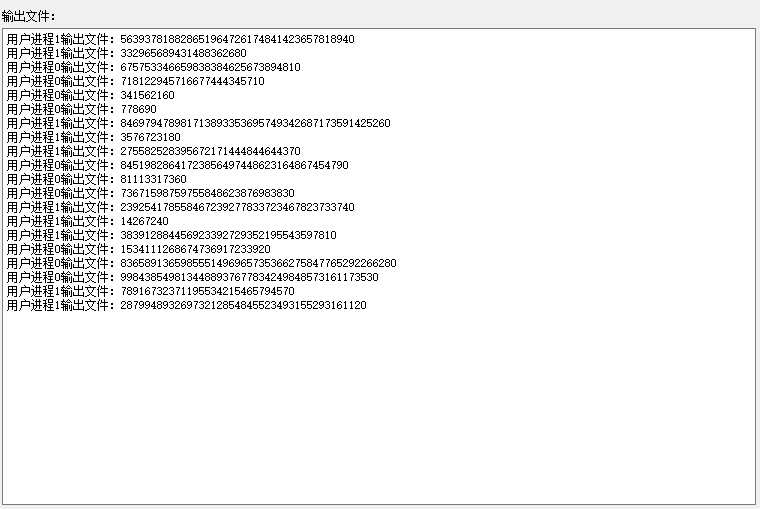


图5-5 输出文件

## 结论

SPOOLing模拟程序的所有结果均按照顺序正确输出，准确无误。