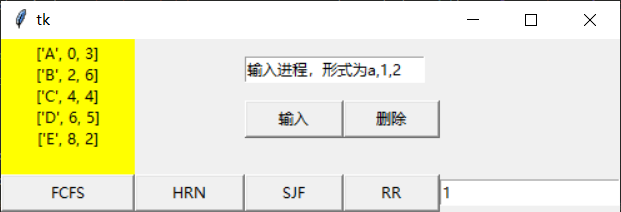
目录

[实验结果 2](#_Toc27283)

[代码 3](#_Toc19763)

# 实验结果

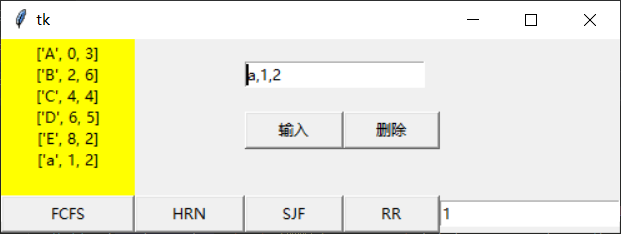
1. 初始界面



（右下角为RR的轮换时间，默认为1）

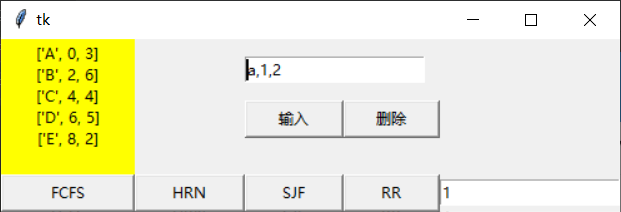
1. 输入新进程信息（a,1,2）

输入a,1,2点击输入



1. 删除最后一个进程

点击删除

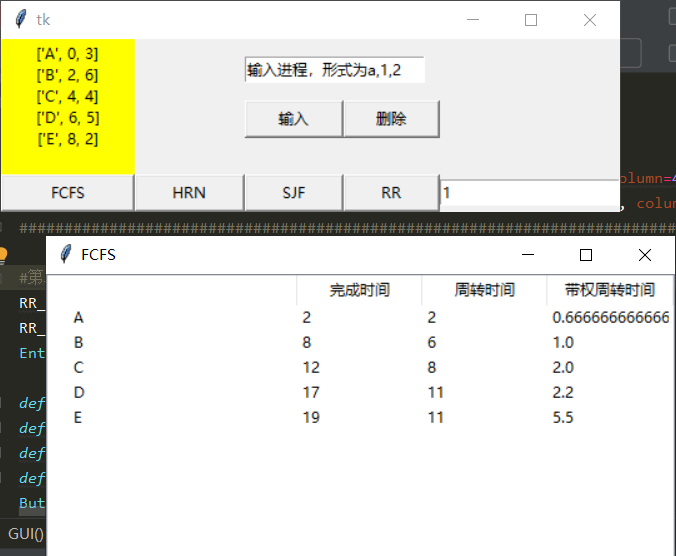


全部删除时：

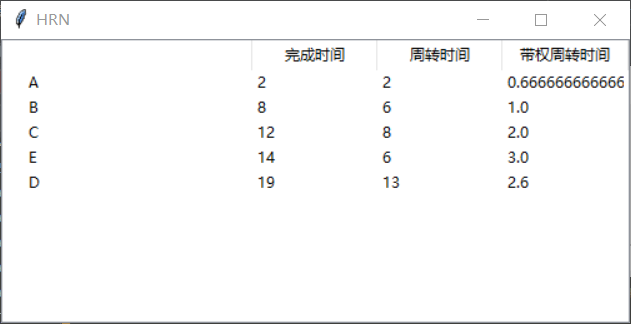


1. 显示算法运行结果

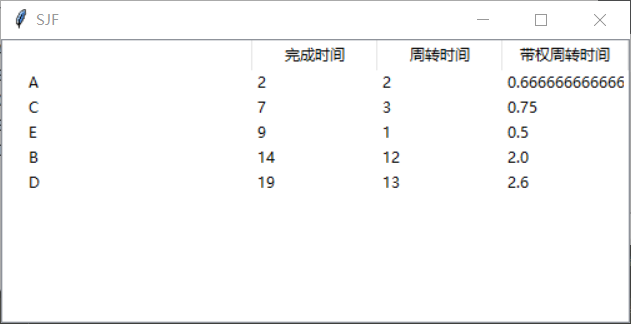
FCFS：



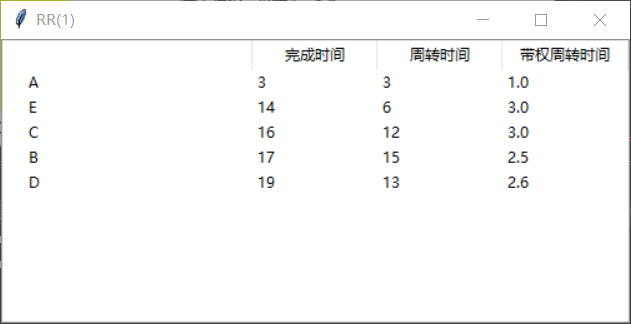
HRN：



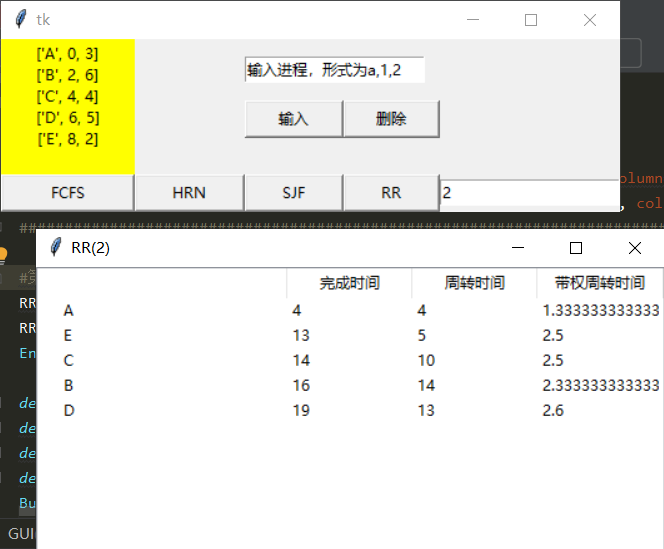
SJF：



RR（轮转时间为1）：



RR（轮转时间为2）：



# 代码

（Python3.7）

*from* tkinter *import* \*  
*from* tkinter *import* ttk  
*class* p: # 进程类  
 *def* \_\_init\_\_(self, *p\_name*, *p\_arrive*, *p\_serve*):  
 self.p\_name = *p\_name* # 进程名  
 self.p\_arrive = *p\_arrive* # 进程到达时间  
 self.p\_need\_serve = *p\_serve* # 剩余进程需要服务时间  
 self.p\_start = -1 # 进程开始时间  
 self.p\_end = -1 # 进程结束时间  
 self.p\_serve = *p\_serve* # 进程需要服务时间  
 *def* calculate\_turnover(self):  
 self.turnover = self.p\_end - self.p\_arrive # 周转时间  
 self.weighted\_turnover = (self.p\_end - self.p\_arrive) / self.p\_serve # 带权周转时间  
 *def* disp(self):  
 self.calculate\_turnover()  
 print('{:^5}|{:^8}|{:^8}|{:^8}|{:^8}|{:^8}|{}'.format('name', 'arrive', 'serve', 'start', 'end', 'turnover', 'weighted-turnover'))  
 print('{:^5}|{:^8}|{:^8}|{:^8}|{:^8}|{:^8}|{}'.format(self.p\_name, self.p\_arrive, self.p\_serve, self.p\_start, self.p\_end,  
 self.turnover, self.weighted\_turnover))  
  
  
*def* GUI(*process*):  
 root\_window = Tk()  
 root\_window.resizable(width=*True*, height=*True*)  
   
 #第一层,1个label，1个empty##########################################################################  
 process\_label\_str = StringVar()  
 process\_str = ''  
 *for* i *in* range(len(*process*)):  
 *if* i != len(process\_str) - 1:  
 process\_str = process\_str + str(*process*[i]) + '\n'  
 *else*:  
 process\_str = process\_str + str(*process*[i])  
 process\_label\_str.set(process\_str)  
 show\_process\_label = Label(root\_window, textvariable=process\_label\_str, bg='yellow')  
 show\_process\_label.grid(row=0, column=0, columnspan=2, rowspan=2)  
  
 new\_process\_entry\_str = StringVar()  
 new\_process\_entry\_str.set('a,1,2')  
 Entry(root\_window, textvariable=new\_process\_entry\_str).grid(row=0, column=4, columnspan=3)  
  
 *def* get\_process():  
 *nonlocal* process  
 new\_process = new\_process\_entry\_str.get()  
 new\_process\_list = new\_process.split(',')  
 process.append([new\_process\_list[0], int(new\_process\_list[1]), int(new\_process\_list[2])])  
 process\_str = ''  
 *for* i *in* range(len(process)):  
 *if* i != len(process\_str) - 1:  
 process\_str = process\_str + str(process[i]) + '\n'  
 *else*:  
 process\_str = process\_str + str(process[i])  
 process\_label\_str.set(process\_str)  
 *def* delete\_process():  
 *del* process[len(process) - 1]  
 process\_str = ''  
 *if* len(process) == 0:  
 process\_str = '无进程'  
 *else*:  
 *for* i *in* range(len(process)):  
 *if* i != len(process\_str) - 1:  
 process\_str = process\_str + str(process[i]) + '\n'  
 *else*:  
 process\_str = process\_str + str(process[i])  
 process\_label\_str.set(process\_str)  
  
  
 Button(root\_window, text='输入', command=get\_process).grid(row=1, column=3, columnspan=2)  
 Button(root\_window, text='删除', command=delete\_process).grid(row=1, column=5, columnspan=2)  
 ##############################################################################################  
  
 #第二层四个按钮################################################################################  
 RR\_time\_str = StringVar()  
 RR\_time\_str.set('1')  
 Entry(root\_window, textvariable=RR\_time\_str).grid(row=2, column=8)  
  
 *def* show\_FCFS():  
 process\_end = FCFS(process.copy())  
 sub\_window = Toplevel()  
 sub\_window.title('FCFS')  
  
 tree = ttk.Treeview(sub\_window) # #创建表格对象  
 tree["columns"] = ("完成时间", "周转时间", "带权周转时间") # #定义列  
 tree.column("完成时间", width=100)  
 tree.column("周转时间", width=100)  
 tree.column("带权周转时间", width=100)  
 tree.heading("完成时间", text="完成时间")  
 tree.heading("周转时间", text="周转时间")  
 tree.heading("带权周转时间", text="带权周转时间")  
 *for* i *in* range(len(process\_end)):  
 p = process\_end[i]  
 tuple\_p = (p.p\_end, p.turnover, p.weighted\_turnover)  
 tree.insert("", i, text=p.p\_name, values=tuple\_p)  
 tree.pack()  
 *def* show\_HRN():  
 process\_end = HRN(process.copy())  
 sub\_window = Toplevel()  
 sub\_window.title('HRN')  
  
 tree = ttk.Treeview(sub\_window) # #创建表格对象  
 tree["columns"] = ("完成时间", "周转时间", "带权周转时间") # #定义列  
 tree.column("完成时间", width=100)  
 tree.column("周转时间", width=100)  
 tree.column("带权周转时间", width=100)  
 tree.heading("完成时间", text="完成时间")  
 tree.heading("周转时间", text="周转时间")  
 tree.heading("带权周转时间", text="带权周转时间")  
 *for* i *in* range(len(process\_end)):  
 p = process\_end[i]  
 tuple\_p = (p.p\_end, p.turnover, p.weighted\_turnover)  
 tree.insert("", i, text=p.p\_name, values=tuple\_p)  
 tree.pack()  
 *def* show\_SJF():  
 process\_end = SJF(process.copy())  
 sub\_window = Toplevel()  
 sub\_window.title('SJF')  
  
 tree = ttk.Treeview(sub\_window) # #创建表格对象  
 tree["columns"] = ("完成时间", "周转时间", "带权周转时间") # #定义列  
 tree.column("完成时间", width=100)  
 tree.column("周转时间", width=100)  
 tree.column("带权周转时间", width=100)  
 tree.heading("完成时间", text="完成时间")  
 tree.heading("周转时间", text="周转时间")  
 tree.heading("带权周转时间", text="带权周转时间")  
 *for* i *in* range(len(process\_end)):  
 p = process\_end[i]  
 tuple\_p = (p.p\_end, p.turnover, p.weighted\_turnover)  
 tree.insert("", i, text=p.p\_name, values=tuple\_p)  
 tree.pack()  
 *def* show\_RR():  
 RR\_time = int(RR\_time\_str.get())  
 process\_end = RR(process.copy(), RR\_time)  
 sub\_window = Toplevel()  
 sub\_window.title('RR(' + str(RR\_time) + ')')  
  
 tree = ttk.Treeview(sub\_window) # #创建表格对象  
 tree["columns"] = ("完成时间", "周转时间", "带权周转时间") # #定义列  
 tree.column("完成时间", width=100)  
 tree.column("周转时间", width=100)  
 tree.column("带权周转时间", width=100)  
 tree.heading("完成时间", text="完成时间")  
 tree.heading("周转时间", text="周转时间")  
 tree.heading("带权周转时间", text="带权周转时间")  
 *for* i *in* range(len(process\_end)):  
 p = process\_end[i]  
 tuple\_p = (p.p\_end, p.turnover, p.weighted\_turnover)  
 tree.insert("", i, text=p.p\_name, values=tuple\_p)  
 tree.pack()  
 Button(root\_window, text='FCFS', command=show\_FCFS).grid(row=2, column=0, columnspan=2)  
 Button(root\_window, text='HRN', command=show\_HRN).grid(row=2, column=2, columnspan=2)  
 Button(root\_window, text='SJF', command=show\_SJF).grid(row=2, column=4, columnspan=2)  
 Button(root\_window, text='RR', command=show\_RR).grid(row=2, column=6, columnspan=2)  
  
  
 root\_window.mainloop()  
  
  
*def* FCFS(*process*):  
 process = sorted(*process*, key=*lambda x*: x[1])  
 process\_stack = [] # 进程栈，栈底为正在运行的进程  
 process\_end = [] # 完成进程列表  
 time = -1 # 时间  
 *while*(1):  
 time += 1  
  
 *if*(len(*process*) != 0 *and* time == *process*[0][1]): # 来进程就进入进程栈  
 new\_p = p(*process*[0][0], *process*[0][1], *process*[0][2]) # 新建进程  
 process\_stack.append(new\_p) # 加入进程栈  
 *del process*[0] # 在输入进程中删除  
  
 *if*(len(process\_stack) != 0): # 进程栈不空就要运行进程栈栈底的程序  
 now\_p = process\_stack[0] # 获取栈底进程  
 *if*(now\_p.p\_need\_serve == now\_p.p\_serve): # 如果需要服务时间等于服务时间，说明刚开始运行，记录开始时间  
 now\_p.p\_start = time  
 now\_p.p\_need\_serve -= 1 # 运行，需要服务时间减1  
  
 *if*(now\_p.p\_need\_serve == 0): # 如果需要服务时间为0  
 now\_p.p\_end = time # 记录结束时间  
 now\_p.calculate\_turnover() # 计算周转时间  
 *del* process\_stack[0] # 从进程栈中删除  
 process\_end.append(now\_p) # 加入完成栈  
 *else*:  
 *break* print('FCFS:')  
 show\_process\_tima(process\_end)  
 *return* process\_end  
  
  
*def* RR(*process*, *RR\_time*):  
 # 对于时间片到了，前一个进程完成了，同时新来了一个进程的情况  
 # 处理过程是，1.将前一个进程移除 2.将新来的进程加入进程栈 3.由于Pass=1不将交换栈底换到栈顶，直接运行栈底进程  
 process = sorted(*process*, key=*lambda x*: x[1])  
  
 process\_stack = [] # 进程栈，栈底为正在运行的进程  
 process\_end = [] # 完成进程列表  
 time = -1 # 时间  
 Pass = 0 # 如果等于1，意为跳过时间片，原因是上一个时间片完成进程完成了，剩下的时间留给下个进程  
 *while*(1):  
 time += 1  
  
 *if*(len(*process*) != 0 *and* time == *process*[0][1]): # 来进程就进入进程栈  
 new\_p = p(*process*[0][0], *process*[0][1], *process*[0][2]) # 新建进程  
 process\_stack.append(new\_p) # 加入进程栈  
 *del process*[0] # 在输入进程中删除  
  
  
 *if*(len(process\_stack) != 0): # 进程栈不空就要运行进程栈栈底的程序  
  
 *if* (time != 0 *and* time % *RR\_time* == 0 *and* Pass==0): # 到了新的时间片  
 temp\_p = process\_stack[0]  
 *del* process\_stack[0]  
 process\_stack.append(temp\_p)  
  
 *if*(time % *RR\_time* == 0): # Pass修改为0  
 Pass = 0  
  
 now\_p = process\_stack[0] # 获取栈底进程  
 # print(time, ' ', now\_p.p\_name)  
 # for i in range(len(process\_stack)):  
 # print('-'\*3, process\_stack[i].p\_name)  
 *if*(now\_p.p\_need\_serve == now\_p.p\_serve): # 如果需要服务时间等于服务时间，说明刚开始运行，记录开始时间  
 now\_p.p\_start = time  
 now\_p.p\_need\_serve -= 1 # 运行，需要服务时间减1  
  
 *if*(now\_p.p\_need\_serve == 0): # 如果需要服务时间为0  
 now\_p.p\_end = time # 记录结束时间  
 now\_p.calculate\_turnover() # 计算周转时间  
 *del* process\_stack[0] # 从进程栈中删除  
 process\_end.append(now\_p) # 加入完成栈  
 Pass = 1 # 意为跳过时间片，原因是上一个时间片完成进程完成了，剩下的时间留给下个进程  
 *else*:  
 *break* print('RR:')  
 show\_process\_tima(process\_end)  
 *return* process\_end  
  
  
*def* SJF(*process*):  
 *from* operator *import* attrgetter  
 process = sorted(*process*, key=*lambda x*: x[1])  
 process\_stack = [] # 进程栈，栈底为正在运行的进程  
 process\_end = [] # 完成进程列表  
 time = -1 # 时间  
 *while*(1):  
 time += 1  
  
 *if*(len(*process*) != 0 *and* time == *process*[0][1]): # 来进程就进入进程栈  
 new\_p = p(*process*[0][0], *process*[0][1], *process*[0][2]) # 新建进程  
 process\_stack.append(new\_p) # 加入进程栈  
 *del process*[0] # 在输入进程中删除  
 process\_stack = sorted(process\_stack, key=attrgetter('p\_need\_serve', 'p\_arrive')) # 每次运行需要服务时间最短的进程  
  
  
 *if*(len(process\_stack) != 0): # 进程栈不空就要运行进程栈栈底的程序  
 now\_p = process\_stack[0] # 获取栈底进程  
 # print(time, ' ', now\_p.p\_name)  
 *if*(now\_p.p\_need\_serve == now\_p.p\_serve): # 如果需要服务时间等于服务时间，说明刚开始运行，记录开始时间  
 now\_p.p\_start = time  
 now\_p.p\_need\_serve -= 1 # 运行，需要服务时间减1  
  
 *if*(now\_p.p\_need\_serve == 0): # 如果需要服务时间为0  
 now\_p.p\_end = time # 记录结束时间  
 now\_p.calculate\_turnover() # 计算周转时间  
 *del* process\_stack[0] # 从进程栈中删除  
 process\_end.append(now\_p) # 加入完成栈  
 *else*:  
 *break* print('SJF:')  
 show\_process\_tima(process\_end)  
 *return* process\_end  
  
  
*def* HRN(*process*):  
 process = sorted(*process*, key=*lambda x*: x[1])  
 process\_stack = [] # 进程栈，栈底为正在运行的进程  
 process\_end = [] # 完成进程列表  
 time = -1 # 时间  
 *while*(1):  
 time += 1  
  
 *if*(len(*process*) != 0 *and* time == *process*[0][1]): # 来进程就进入进程栈  
 new\_p = p(*process*[0][0], *process*[0][1], *process*[0][2]) # 新建进程  
 process\_stack.append(new\_p) # 加入进程栈  
 *del process*[0] # 在输入进程中删除  
  
 *if*(len(process\_stack) != 0): # 进程栈不空就要运行进程栈栈底的程序  
 process\_stack = sorted(process\_stack, key=*lambda x*:(1 + (time - x.p\_arrive)/x.p\_need\_serve), reverse=*True*)  
 now\_p = process\_stack[0] # 获取栈底进程  
 # print('-'\*30, time, ' ', now\_p.p\_name)  
 # for i in range(len(process\_stack)):  
 # print(process\_stack[i].p\_name, time, ' ', process\_stack[i].p\_arrive, ' ', process\_stack[i].p\_need\_serve)  
 *if*(now\_p.p\_need\_serve == now\_p.p\_serve): # 如果需要服务时间等于服务时间，说明刚开始运行，记录开始时间  
 now\_p.p\_start = time  
 now\_p.p\_need\_serve -= 1 # 运行，需要服务时间减1  
  
 *if*(now\_p.p\_need\_serve == 0): # 如果需要服务时间为0  
 now\_p.p\_end = time # 记录结束时间  
 now\_p.calculate\_turnover() # 计算周转时间  
 *del* process\_stack[0] # 从进程栈中删除  
 process\_end.append(now\_p) # 加入完成栈  
 *else*:  
 *break* print('HRN:')  
 show\_process\_tima(process\_end)  
 *return* process\_end  
  
  
*def* show\_process\_tima(*process\_end*):  
 # print('{:^5}|{:^8}|{:^8}|{:^8}|{:^8}|{:^8}|{}'.format('name', 'arrive', 'serve', 'start', 'end', 'turnover',  
 # 'weighted-turnover'))  
 # for i in range(len(process\_end)):  
 # p = process\_end[i]  
 # print('{:^5}|{:^8}|{:^8}|{:^8}|{:^8}|{:^8}|{}'.format(p.p\_name, p.p\_arrive, p.p\_serve, p.p\_start,  
 # p.p\_end,  
 # p.turnover, p.weighted\_turnover))  
 print('{:^8}|{:^6}|{:^8}|{}'.format('进程名', '完成时间', '周转时间',  
 '带权周转时间'))  
 *for* i *in* range(len(*process\_end*)):  
 p = *process\_end*[i]  
 print('{:^10}|{:^10}|{:^10}|{}'.format(p.p\_name, p.p\_end,  
 p.turnover, p.weighted\_turnover))  
  
  
*if* \_\_name\_\_ == '\_\_main\_\_':  
 # 进程  
 process=[['A', 0, 3],  
 ['B', 2, 6],  
 ['C', 4, 4],  
 ['D', 6, 5],  
 ['E', 8, 2]]  
  
  
  
 # 测试进程类  
 # new\_p = p(process[0][0], process[0][1], process[0][2])  
 # new\_p.disp()  
  
 # 测试算法  
 FCFS\_end = FCFS(process.copy())  
 RR\_end = RR(process.copy(), 1)  
 SJF\_end = SJF(process.copy())  
 HRN\_end = HRN(process.copy())  
  
 # 界面  
 GUI(process)