

```
1 # Batch Scheduling Algorithms by Chip Henderson
2 # CS7343 - 15 October 2022
3
4 from array import array
5 import numbers
6 import random
7 from statistics import mean
8 import numpy as np
9
10
11 def arrTimeGen(low: int, k: int, size: int) -> array:
12     # Generate arrival times based on uniform distribution
13     arrivalTimeArr = np.random.randint(low, k, size)
14     return arrivalTimeArr
15
16 def cpuTimeGen(d: float, v: float, size: int) -> array:
17     # Generate CPU total times based on Gaussian distribution
18     cpuTimeArr = np.random.normal(d, v, size)
19     return cpuTimeArr
20
21 def activeProcess(thisArray: array, time: int) -> array:
22     # Loop through all processes, set active flags
23     for process in thisArray:
24         if time >= process[0] and process[2] != 1:
25             process[2] = 1
26     return thisArray
27
28 def srt_activeProcess(thisArray: array, time: int) -> array:
29     # Loop through all processes, set active flags specific for SRT
30     for process in thisArray:
31         if time >= process[0] and process[2] != 1:
32             process[2] = 1
33             process[1] = np.random.normal(d, v, size=None)
34     # Create an index array to sort by arrival time, then CPU time
35     indexArray = np.lexsort((thisArray[:,1],thisArray[:,0]))
36     thisArray = thisArray[indexArray]
37     return thisArray
38
39 def fifoQueue(k: int, d: int, v: float, n: int) -> list:
40     ttlist = []
41     t = 0
42
43     # Create an empty array to hold the sim results
44     simArray = np.empty((1,n))
45
46     # Create a zero array to be the flag value
47     zeroArray = np.zeros((1,n),float)
48
49     # Generates random total CPU times
50     cpuTimeArr = cpuTimeGen(d,v,n)
51
52     # Sort the array by arrival time for FIFO
53     arrTimeArr = np.sort(arrTimeGen(0,k,n))
54
55     # Merges the two arrays and transposes the result,
56     # assigning a CPU time to an arrival time
57     simArray = np.vstack((arrTimeArr, cpuTimeArr, zeroArray)).T
58
59     # Loop through each process in the table
```

```
60     for process in simArray:
61         while t < process[0]: # Increments t for inactive processes
62             t += 1
63             simArray = activeProcess(simArray, t)
64
65         i = process[1] # Assign total CPU time to iterator
66         while i > 0:
67             t += 1
68             simArray = activeProcess(simArray, t)
69             i -= 1
70
71         process[2] = 0 # Set flag to inactive
72
73         ttList.append(t - process[0])
74         avg_tt = round(mean(ttList),2)
75
76         # Delete the process from the array once it has completed
77         simArray = np.delete(simArray,0,0)
78
79     return avg_tt
80
81 def sjfQueue(k: int, d: int, v: float, n: int) -> list:
82     ttList = []
83     t = 0
84
85     # Create an empty array to hold the sim results
86     simArray = np.empty((1,n))
87
88     # Create a zero array to be the flag value
89     zeroArray = np.zeros((1,n),float)
90
91     # Get arrays for arrival time and cpu time, both sorted for SJF
92     arrTimeArr = np.sort(arrTimeGen(0,k,n))
93     cpuTimeArr = np.sort(cpuTimeGen(d,v,n))
94
95     # Merges the two arrays and transposes the result,
96     # assigning a CPU time to an arrival time
97     simArray = np.vstack((arrTimeArr, cpuTimeArr, zeroArray)).T
98
99     for process in simArray:
100         while t < process[0]: # Increments t for inactive processes
101             t += 1
102             simArray = activeProcess(simArray, t)
103
104         i = process[1] # Assign total CPU time to iterator
105         while i > 0:
106             t += 1
107             simArray = activeProcess(simArray, t)
108             i -= 1
109
110         process[2] = 0 # Set flag to inactive
111
112         ttList.append(t - process[0])
113         avg_tt = round(mean(ttList),2)
114
115         # Delete the process from the array once it has completed
116         simArray = np.delete(simArray,0,0)
117
118     return avg_tt
119
```

```
120 def srtQueue(k: int, d: int, v: float, n: int) -> list:
121     ttList = []
122     t = 0
123
124     # Create an empty array to hold the sim results
125     simArray = np.empty((1,n))
126
127     # Create a zero array to be CPU time and flag value
128     zeroArray = np.zeros((2,n),float)
129
130     # Sort the array by arrival tie
131     arrTimeArr = np.sort(arrTimeGen(0,k,n))
132
133     # Merges the two arrays and transposes the result,
134     # assigning a CPU time to an arrival time
135     simArray = np.vstack((arrTimeArr, zeroArray)).T
136
137     for process in simArray:
138         while t < process[0]: # Increments t for inactive processes
139             t += 1
140             simArray = srt_activeProcess(simArray, t)
141
142             process = simArray[0] # Update process values
143             i = process[1] # Assign total CPU time to iterator
144             while i > 0:
145                 t += 1
146                 simArray = srt_activeProcess(simArray, t)
147                 i -= 1
148
149             process[2] = 0 # Set flag to inactive
150
151             ttList.append(t - process[0])
152             avg_tt = round(mean(ttList),2)
153
154             # Delete the process from the array once it has completed
155             simArray = np.delete(simArray,0,0)
156
157     return avg_tt
158
159 j = 0
160 k = 1000
161 d = 1
162 v = 0.2 * d
163 n = 100
164 avg_ttList = []
165 while j < 100:
166     avg_ttList.append(fifoQueue(k,d,v,n))
167     j += 1
168 print(avg_ttList)
169
170 j = 0
171 avg_ttList = []
172 while j < 100:
173     avg_ttList.append(sjfQueue(k,d,v,n))
174     j += 1
175 print(avg_ttList)
176
177 j = 0
178 avg_ttList = []
179 while j < 100:
```

```
180     avg_ttList.append(srtQueue(k,d,v,n))
181     j += 1
182 print(avg_ttList)
```