Scheduling Algorithm

นายอิทธิพร แก้วอำไพ 590610681

```python
import random
processCount = int(input("Please enter number of process : "))

percentProcessOne = int(input("Please enter percentage of processOne : "))
percentProcessTwo = int(input("Please enter percentage of processTwo : "))
percentProcessThree = int(input("Please enter percentage of processThree : "))

processOneCount = int(processCount*percentProcessOne/100)
processTwoCount = int(processCount*percentProcessTwo/100)
processThreeCount = int(processCount*percentProcessThree/100)

process = []

#ProcessOne
for i in range(processOneCount):
    process.append(random.randrange(2,9,1))
#ProcessTwo
for i in range(processTwoCount):
    process.append(random.randrange(20,31,1))
#ProcessThree
for i in range(processThreeCount):
    process.append(random.randrange(35,41,1))

#shuffle values in process
random.shuffle(process)
print(*process, sep = ", ")
```

Concept of Code

ส่วนของการรับค่า

เริ่มต้นโปรแกรมด้วยรอรับค่าจำนวน process ที่ต้องการเป็นจำนวนที่ต้องการ หลังจากนั้นจะทำการแบ่งเป็น จะทำการแบ่งเป็น 3 ส่วน คือ process ที่ใช้เวลาน้อย process ที่ใช้เวลาปานกลางและ process ที่ใช้เวลานาน ซึ่งจะเปลี่ยนแปลงไปตามที่เราต้องการกำหนด โดย code ที่อยู่ในส่วนของ #ProcessOne คือ code ที่จะสุ่มค่าระหว่างเลข 2-8 ในส่วนของ #ProcessTwo คือ code ที่จะสุ่มค่าระหว่าง 20-30 และสุดท้าย #ProcessThree คือ code ที่จะสุ่มระหว่าง 35-40 หลังจากสุ่มจำนวนของข้อมูลทั้งหมดแล้วตัว code จะทำการสลับค่าทั้งหมดที่อยู่ใน process

ส่วนของอัลกอริทึม

1. First Come First Serve (FCFS)

เป็นอัลกอริทึ่มที่ process จะทำงานตามลำดับ มาก่อนได้ก่อน สมมติว่ามีงานกำลังทำอยู่แล้วมีอีกงานหนึ่งเข้ามางานนั้นจะต้องรอให้งานแรกเสร็จก่อน ดังนั้นถ้ามีงานต่อๆไปเข้ามางานนั้นก็ต้องรอให้งานก่อนหน้าเสร็จก่อน จึงจะทำได้สามารถเขียนอธิบายอย่างง่ายได้ดังนี้

Process1 -> Process2 -> Process3 -> ... -> ProcessN

```
#First Come First Serve
#############################################################################################
waitTime = [0]
sumWaitTime = 0
for i in range (0 , len(process)):
    waitTime.append(process[i] + waitTime[i])
    sumWaitTime += waitTime[i]
avgWaitTime = sumWaitTime/len(process)
print("Sum FCFS: " + str(sumWaitTime))
print("Avg FCFS: " + str(avgWaitTime))
#############################################################################################
```

2. Shortest Job First (SJF)

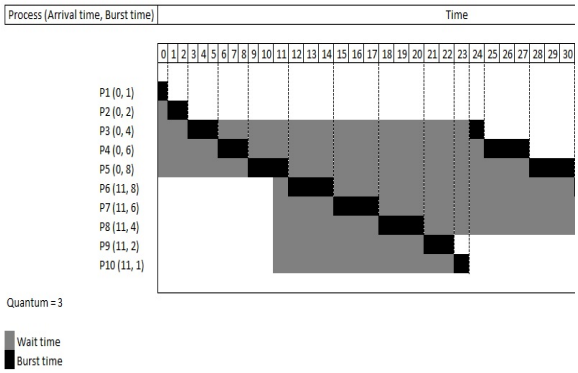
เป็นอัลกอริทึ่มที่ process execution น้อยๆ จะได้เข้าคิวก่อน โดยในที่นี้ยังไม่ได้ใช้ arrive time ในการพิจารณา โดยหลักการทำงานก็คือเมื่อมี process เข้ามาทางอัลกอริทึ่มจะทำการสลับ process ที่มีใช้เวลาน้อยมาทำก่อน

```
#Shortest Job First
#############################################################################################
#applying bubble sort to sort process according to their burst time
SJFProcess = process
for i in range(0,len(SJFProcess)-1):
    for j in range(0,len(SJFProcess)-i-1):
        if(SJFProcess[j]>SJFProcess[j+1]):
            temp=SJFProcess[j]
            SJFProcess[j]=SJFProcess[j+1]
            SJFProcess[j+1]=temp
# print(*SJFProcess ,sep = ", ")
#simply apply FCFS
waitTime = [0]
sumWaitTime = 0
for i in range (0 , len(process)):
    waitTime.append(process[i] + waitTime[i])
    sumWaitTime += waitTime[i]
avgWaitTime = sumWaitTime/len(process)
print("Sum SJF: " + str(sumWaitTime))
print("Avg SJF: " + str(avgWaitTime))
#############################################################################################
```

3. Round Robin (RR)

เป็นอัลกอริทึ่มที่ทุก process จะได้ทำในเวลาที่กำหนดให้ ซึ่งตัวที่กำหนดเวลาให้จะเรียกว่า Quantum time ไม่สน priority และมี Limit ของ Response Time ที่ชัดเจนจะรอแบบมีเวลาจำกัด ทำให้ไม่มี Starvation เกิดขึ้น ซึ่ง Quantum Time จะเป็นตัวกำหนด Wait Time ว่าจะมากหรือจะน้อย

```python
#Round Robin
################################################################################
waitTime = [0]
quantum = 5
waitTime = [0] * len(process)
rem_process = [0] * len(process)
for i in range(len(process)):
    rem_process[i] = process[i]
t = 0
while(1):
    done = True
    for i in range(len(process)):
        if (rem_process[i] > 0) :
            done = False
            if (rem_process[i] > quantum) :
                t += quantum
                rem_process[i] -= quantum
            else:
                t = t + rem_process[i]
                waitTime[i] = t - process[i]
                rem_process[i] = 0
    if (done == True):
        break
sumWaitTime = 0
for i in range(len(process)):
    sumWaitTime += waitTime[i]
avgWaitTime = sumWaitTime/len(process)
print("Sum RR: " + str(sumWaitTime))
print("Avg RR: " + str(avgWaitTime))
################################################################################
```



ผลการทดลอง

สมมติฐานที่ 1 ถ้ามีชุดโปรเซส ที่ต้องการใช้ CPU ความยาวไม่เท่ากันในการทำงาน เข้ามาเพื่อให้จัดคิวจำนวน 60 โปรเซส โดยให้นักศึกษาสร้างชุดโปรเซสเองโดยทำการสุ่มโปรเซสที่ใช้เวลา (2 ถึง 8 milisec) จำนวน 70 %, โปรเซสที่ใช้เวลา (20 ถึง 30 milisec) จำนวน 20 % , โปรเซสที่ใช้เวลา (35 ถึง 40 milisec) จำนวน 10 % และเมื่อได้โปรเซสครบแล้วให้สุ่มลำดับที่เข้ามาในคิว

a. การรันครั้งที่ 1 ได้ผลดังภาพที่ 1.1

```
WIN7PRO@WIN7PRO-PC MINGW64 /e/Github/Oshomework/Scheduling Algorithm (master)
$ py Code.py
Please enter number of process : 60
Please enter percentage of processOne : 70
Please enter percentage of processTwo : 20
Please enter percentage of processThree : 10
DataSet : 5 7 29 5 4 2 7 8 39 7 5 35 35 5 8 23 5 2 30 2 7 6 28 38 5 4 4 20 7 40 2 30 5 30 8 3 3 23 8 3 2 3 3 3 5 8 7 35 4 6 21 2 8 29 8 20 5 8 24 8
Sum FCFS: 22765
Avg FCFS: 379.4166666666667
Sum SJF: 11203
Avg SJF: 186.71666666666667
Sum RR: 16728
Avg RR: 278.8
```

ภาพที่ 1.1

b. การรันครั้งที่ 2 ได้ผลดังภาพที่ 1.2

```
WIN7PRO@WIN7PRO-PC MINGW64 /e/Github/Oshomework/Scheduling Algorithm (master)
$ py Code.py
Please enter number of process : 60
Please enter percentage of processOne : 70
Please enter percentage of processTwo : 20
Please enter percentage of processThree : 10
DataSet : 7 2 4 3 6 2 2 7 39 6 5 3 22 24 7 24 20 5 40 5 36 5 5 6 37 4 6 6 8 27 2 30 2 4 30 7 3 4 7 7 4 22 26 36 8 3 35 5 3 4 8 5 22 3 5 20 6 5 26 8
Sum FCFS: 20905
Avg FCFS: 348.4166666666667
Sum SJF: 10826
Avg SJF: 180.43333333333334
Sum RR: 16161
Avg RR: 269.35
```

ภาพที่ 1.2

c. การรันครั้งที่ 3 ได้ผลดังภาพที่ 1.3

```
WIN7PRO@WIN7PRO-PC MINGW64 /e/Github/Oshomework/Scheduling Algorithm (master)
$ py Code.py
Please enter number of process : 60
Please enter percentage of processOne : 70
Please enter percentage of processTwo : 20
Please enter percentage of processThree : 10
DataSet : 7 4 36 7 30 4 7 27 26 7 37 2 2 21 24 2 8 8 35 2 37 8 3 4 5 3 7 2 5 2 8 8 6 26 4 7 5 24 2 2 27 4 8 20 40 7 20 3 3 6 3 4 22 40 4 5 25 8 2 7
Sum FCFS: 22528
Avg FCFS: 375.46666666666664
Sum SJF: 10571
Avg SJF: 176.18333333333334
Sum RR: 16076
Avg RR: 267.93333333333334
```

ภาพที่ 1.3

สมมติฐานที่ 2 ถ้ามีชุดโปรเซส ที่ต้องการใช้ CPU ความยาวไม่เท่ากันในการทำงาน เข้ามาเพื่อให้จัดคิวจำนวน 40 โปรเซส โดยให้นักศึกษาสร้างชุดโปรเซสเองโดยทำการสุ่มโปรเซสที่ใช้เวลา (2 ถึง 8 milisec) จำนวน 50 %, โปรเซสที่ใช้เวลา (20 ถึง 30 milisec) จำนวน 30 % , โปรเซสที่ใช้เวลา (35 ถึง 40 milisec) จำนวน 20 % และเมื่อได้โปรเซสครบแล้วให้สุ่มลำดับที่เข้ามาในคิว

a. การรันครั้งที่ 1 ได้ผลดังภาพที่ 2.1

```
WIN7PRO@WIN7PRO-PC MINGW64 /e/Github/Oshomework/Scheduling Algorithm (master)
$ py Code.py
Please enter number of process : 40
Please enter percentage of processOne : 50
Please enter percentage of processTwo : 30
Please enter percentage of processThree : 20
DataSet : 6 4 21 20 4 26 38 7 28 21 27 8 6 5 2 6 6 5 2 8 22 36 36 29 40 2 7 38 24 39 8 38 40 21 20 7 7 23 5 5
Sum FCFS: 12779
Avg FCFS: 319.475
Sum SJF: 7761
Avg SJF: 194.025
Sum RR: 13361
Avg RR: 334.025
```

ภาพที่ 2.1

b. การรันครั้งที่ 2 ได้ผลดังภาพที่ 2.2

```
WIN7PRO@WIN7PRO-PC MINGW64 /e/Github/Oshomework/Scheduling Algorithm (master)
$ py Code.py
Please enter number of process : 40
Please enter percentage of processOne : 50
Please enter percentage of processTwo : 30
Please enter percentage of processThree : 20
DataSet : 3 6 26 5 28 21 7 7 21 36 2 36 5 8 39 8 7 6 5 25 26 37 7 3 8 21 38 2 21 4 40 7 4 20 30 40 27 35 30 8
Sum FCFS: 12426
Avg FCFS: 310.65
Sum SJF: 7955
Avg SJF: 198.875
Sum RR: 13735
Avg RR: 343.375
```

ภาพที่ 2.2

c. การรันครั้งที่ 3 ได้ผลดังภาพที่ 2.3

```
WIN7PRO@WIN7PRO-PC MINGW64 /e/Github/Oshomework/Scheduling Algorithm (master)
$ py Code.py
Please enter number of process :40
Please enter percentage of processOne : 50
Please enter percentage of processTwo : 30
Please enter percentage of processThree : 20
DataSet : 27 8 26 39 7 8 25 5 37 38 3 8 2 21 6 24 36 21 3 37 27 38 6 5 3 38 4 2 8 5 8 39 3 5 29 30 21 7 20 20
Sum FCFS: 14209
Avg FCFS: 355.225
Sum SJF: 7718
Avg SJF: 192.95
Sum RR: 13083
Avg RR: 327.075
```

ภาพที่ 2.3

สมมติฐานที่ 3 ถ้ามีชุดโปรเซส ที่ต้องการใช้ CPU ความยาวไม่เท่ากันในการทำงาน เข้ามาเพื่อให้จัดคิวจำนวน 20 โปรเซส โดยให้นักศึกษาสร้างชุดโปรเซสเองโดยทำการสุ่มโปรเซสที่ใช้เวลา (2 ถึง 8 milisec) จำนวน 40 %, โปรเซสที่ใช้เวลา (20 ถึง 30 milisec) จำนวน 40 % , โปรเซสที่ใช้เวลา (35 ถึง 40 milisec) จำนวน 20 % และเมื่อได้โปรเซสครบแล้วให้สุ่มลำดับที่เข้ามาในคิว

a. การรันครั้งที่ 1 ได้ผลดังภาพที่ 3.1

```
WIN7PRO@WIN7PRO-PC MINGW64 /e/Github/Oshomework/Scheduling Algorithm (master)
$ py Code.py
Please enter number of process : 20
Please enter percentage of processOne : 40
Please enter percentage of processTwo : 40
Please enter percentage of processThree : 20
DataSet : 37 29 8 24 4 5 39 3 30 6 22 38 22 5 23 3 6 22 23 37
Sum FCFS: 3706
Avg FCFS: 185.3
Sum SJF: 2229
Avg SJF: 111.45
Sum RR: 3879
Avg RR: 193.95
```

ภาพที่ 3.1

b. การรันครั้งที่ 2 ได้ผลดังภาพที่ 3.2

```
WIN7PRO@WIN7PRO-PC MINGW64 /e/Github/Oshomework/Scheduling Algorithm (master)
$ py Code.py
Please enter number of process : 20
Please enter percentage of processOne : 40
Please enter percentage of processTwo : 40
Please enter percentage of processThree : 20
DataSet : 35 26 7 4 3 5 29 3 20 39 7 25 26 25 4 5 24 35 24 39
Sum FCFS: 3255
Avg FCFS: 162.75
Sum SJF: 2230
Avg SJF: 111.5
Sum RR: 3760
Avg RR: 188.0
```

ภาพที่ 3.2

c. การรันครั้งที่ 3 ได้ผลดังภาพที่ 3.3

```
WIN7PRO@WIN7PRO-PC MINGW64 /e/Github/Oshomework/Scheduling Algorithm (master)
$ py Code.py
Please enter number of process : 20
Please enter percentage of processOne : 40
Please enter percentage of processTwo : 40
Please enter percentage of processThree : 20
DataSet : 3 27 20 6 38 2 3 36 2 4 7 26 30 29 35 22 28 29 39 6
Sum FCFS: 3209
Avg FCFS: 160.45
Sum SJF: 2219
Avg SJF: 110.95
Sum RR: 3949
Avg RR: 197.45
```

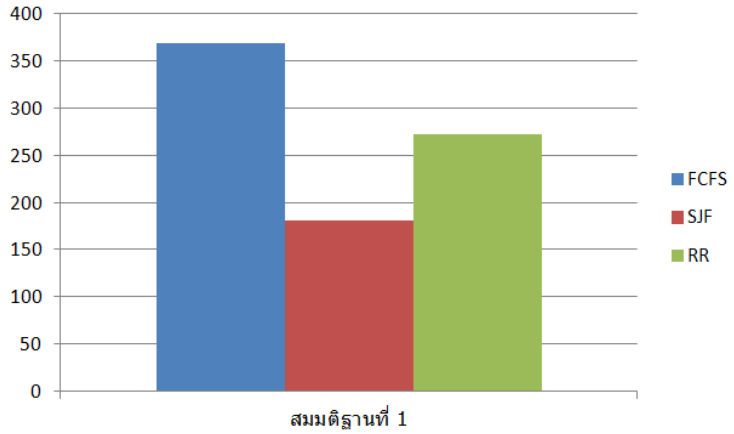
ภาพที่ 3.3

ผลการทดลอง

จากการทดลองสมมติฐาน 3 แบบ สามารถนำไปแสดงเป็นกราฟได้ดังนี้

สมมติฐานที่ 1

สมมติฐานที่ 1								
Algorithm	Waiting Time of Algorithm (ms.)							
	1		2		3		Average	
	Total	Avg	Total	Avg	Total	Avg	Total	Avg
FCFS	22765	379.416	20905	348.416	22528	375.466	66198	22066
SJF	11203	186.716	10826	180.433	10571	176.183	32600	10866.67
RR	16728	278.8	16161	269.35	16076	267.933	48965	16321.67

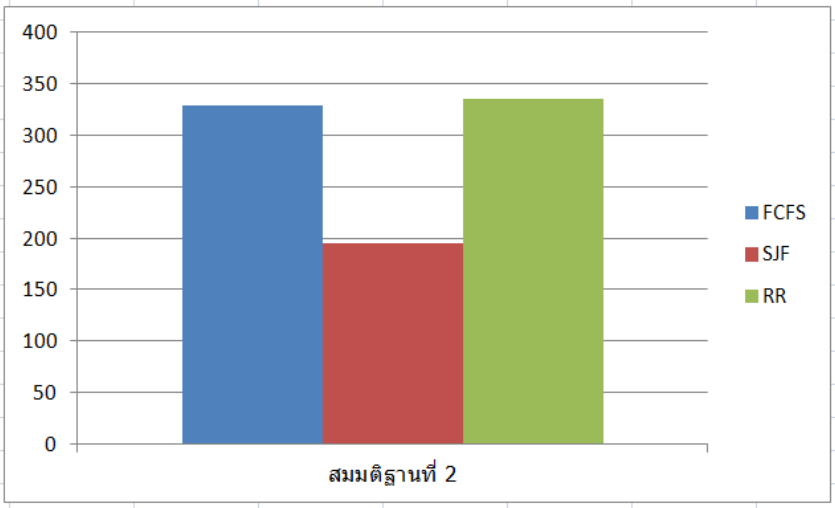


สมมติฐานที่ 1

จากสมมติฐานที่ 1 ถ้ามีชุดโปรเซส ที่ต้องการใช้ CPU ความยาวไม่เท่ากันในการทำงาน เข้ามาเพื่อให้จัดคิวจำนวน 60 โปรเซส โดยให้นักศึกษาสร้างชุดโปรเซสเองโดยทำการสุ่มโปรเซสที่ใช้เวลา (2 ถึง 8 milisec) จำนวน 70 %, โปรเซสที่ใช้เวลา (20 ถึง 30 milisec) จำนวน 20 % , โปรเซสที่ใช้เวลา (35 ถึง 40 milisec) จำนวน 10 % และเมื่อได้โปรเซสครบแล้วให้สุ่มลำดับที่เข้ามาในคิว พบว่า **SJF ใช้เวลาน้อยที่สุด ตามมาด้วย RR และ FCFS ใช้เวลามากที่สุด**

สมมติฐานที่ 2

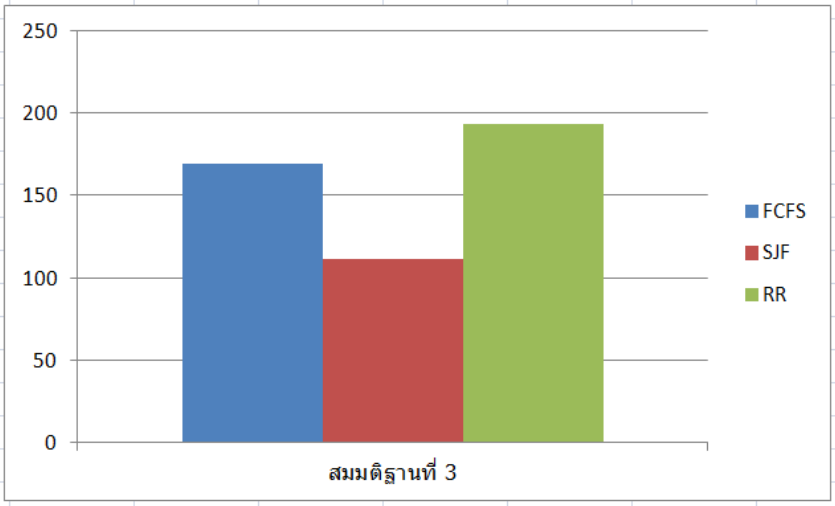
สมมติฐานที่ 2								
Algorithm	Waiting Time of Algorithm (ms.)							
	1		2		3		Average	
	Total	Avg	Total	Avg	Total	Avg	Total	Avg
FCFS	12779	319.475	12426	310.65	14209	355.225	39414	328.45
SJF	7761	194.025	7955	198.875	7718	192.95	23434	195.2833
RR	13361	334.025	13735	343.375	13083	327.075	40179	334.825



จากสมมติฐานที่ 2 ถ้ามีชุดโปรเซส ที่ต้องการใช้ CPU ความยาวไม่เท่ากันในการทำงาน เข้ามาเพื่อให้จัดคิวจำนวน 40 โปรเซส โดยให้นักศึกษาสร้างชุดโปรเซสเองโดยทำการสุ่มโปรเซสที่ใช้เวลา (2 ถึง 8 milisec) จำนวน 50 %, โปรเซสที่ใช้เวลา (20 ถึง 30 milisec) จำนวน 30 % , โปรเซสที่ใช้เวลา (35 ถึง 40 milisec) จำนวน 20 % และเมื่อได้โปรเซสครบแล้วให้สุ่มลำดับที่เข้ามาในคิว พบว่า **SJF ใช้เวลาน้อยที่สุด FCFS ใช้เวลารองลงมา และ RR ใช้เวลามากที่สุด**

สมมติฐานที่ 3

สมมติฐานที่ 3								
Algorithm	Waiting Time of Algorithm (ms.)							
	1		2		3		Average	
	Total	Avg	Total	Avg	Total	Avg	Total	Avg
FCFS	3706	185.3	3255	162.75	3209	160.45	10170	169.5
SJF	2229	111.45	2230	111.5	2219	110.95	6678	111.3
RR	3879	193.95	3760	188	3949	197.45	11588	193.1333



จากสมมติฐานที่ 3 ถ้ามีชุดโปรเซส ที่ต้องการใช้ CPU ความยาวไม่เท่ากันในการทำงาน เข้ามาเพื่อให้จัดคิวจำนวน 20 โปรเซส โดยให้นักศึกษาสร้างชุดโปรเซสเองโดยทำการสุ่มโปรเซสที่ใช้เวลา (2 ถึง 8 milisec) จำนวน 40 %, โปรเซสที่ใช้เวลา (20 ถึง 30 milisec) จำนวน 40 % , โปรเซสที่ใช้เวลา (35 ถึง 40 milisec) จำนวน 20 % และเมื่อได้โปรเซสครบแล้วให้สุ่มลำดับที่เข้ามาในคิว พบว่า **SJF ใช้เวลาน้อยที่สุด FCFS ใช้เวลารองลงมา และ RR ใช้เวลามากที่สุด**

สรุปผลการทดลอง

จากการทดลองทั้งหมดสรุปได้ว่า algorithm ของ Shortest Job First นั้นมีเวลาเฉลี่ยน้อยที่สุด ในส่วนของ First Come First Serve เป็นวิธีที่ธรรมดามากที่สุดแต่จะทำให้บางครั้ง Process สั้นๆต้องรอเป็นเวลานาน และในส่วนของ Round Robin นั้นการกำหนด Quantime จะเป็นตัวกำหนดเวลาที่ต้องรอ ซึ่งควรปรับตามความเหมาะสม

จากการสังเกตของผู้ทดลองพบว่า Shortest Job First เป็นการพัฒนามากจาก First Come First Serve ทำให้เกิดเวลารอที่น้อยที่สุด และยังพบว่า Round Robin ถูกพัฒนามาเพื่อลดการเกิดปัญหา starvation ซึ่งจะพบบ่อยใน FCFS และ SJF

ภาคผนวก

Code.py

```python
import random
processCount = int(input("Please enter number of process : "))

percentProcessOne = int(input("Please enter percentage of processOne : "))
percentProcessTwo = int(input("Please enter percentage of processTwo : "))
percentProcessThree = int(input("Please enter percentage of processThree : "))

processOneCount = int(processCount*percentProcessOne/100)
processTwoCount = int(processCount*percentProcessTwo/100)
processThreeCount = int(processCount*percentProcessThree/100)

process = []

#ProcessOne
for i in range(processOneCount):
    process.append(random.randrange(2,9,1))
#ProcessTwo
for i in range(processTwoCount):
    process.append(random.randrange(20,31,1))
#ProcessThree
for i in range(processThreeCount):
    process.append(random.randrange(35,41,1))

#shuffle values in process
random.shuffle(process)
print("DataSet :" , *process, sep = " ")

#First Come First Serve
##########################################################################################
waitTime = [0]
sumWaitTime = 0
for i in range (0 , len(process)):
    waitTime.append(process[i] + waitTime[i])
    sumWaitTime += waitTime[i]
avgWaitTime = sumWaitTime/len(process)
print("Sum FCFS: " + str(sumWaitTime))
print("Avg FCFS: " + str(avgWaitTime))
##########################################################################################
```

```python
#Shortest Job First
##########################################################################################
#applying bubble sort to sort process according to their burst time
SJFProcess = process
for i in range(0,len(SJFProcess)-1):
    for j in range(0,len(SJFProcess)-i-1):
        if(SJFProcess[j]>SJFProcess[j+1]):
            temp=SJFProcess[j]
            SJFProcess[j]=SJFProcess[j+1]
            SJFProcess[j+1]=temp
# print(*SJFProcess ,sep = ", ")
#simply apply FCFS
waitTime = [0]
sumWaitTime = 0
for i in range (0 , len(process)):
    waitTime.append(process[i] + waitTime[i])
    sumWaitTime += waitTime[i]
avgWaitTime = sumWaitTime/len(process)
print("Sum SJF: " + str(sumWaitTime))
print("Avg SJF: " + str(avgWaitTime))
##########################################################################################

```

```
#Round Robin
################################################################################################
waitTime = [0]
quantum = 5
waitTime = [0] * len(process)
rem_process = [0] * len(process)
for i in range(len(process)):
    rem_process[i] = process[i]
t = 0
while(1):
    done = True
    for i in range(len(process)):
        if (rem_process[i] > 0) :
            done = False
            if (rem_process[i] > quantum) :
                t += quantum
                rem_process[i] -= quantum
            else:
                t = t + rem_process[i]
                waitTime[i] = t - process[i]
                rem_process[i] = 0
    if (done == True):
        break
sumWaitTime = 0
for i in range(len(process)):
    sumWaitTime += waitTime[i]
avgWaitTime = sumWaitTime/len(process)
print("Sum RR: " + str(sumWaitTime))
print("Avg RR: " + str(avgWaitTime))
################################################################################################
```