**Institute of Computer Technology**

**B. Tech Computer Science and Engineering**

**Sub: Operating Systems**

**Practical 2: CPU Scheduling Algorithms**

**CPU Scheduling Algorithm**

**CODE:**

def readdata():

file = open('data', 'r')

data = []

for line in file.read().split('\n'):

l = line.split(' ')

dict = {

'PID': l[0].strip(),

'AT': int(l[1].strip()),

'BT': int(l[2].strip()),

'CT': 0,

'TAT': 0,

'WT': 0,

}

data.append(dict)

return data

def readdata\_with\_priority():

file = open('data\_with\_priority', 'r')

data = []

for line in file.read().split('\n'):

l = line.split(' ')

dict = {

'PID': l[0].strip(),

'AT': int(l[1].strip()),

'BT': int(l[2].strip()),

'CT': 0,

'TAT': 0,

'WT': 0,

'PRIORITY': int(l[3].strip())

}

data.append(dict)

return data

def fcfscompletiontime(data):

data = sorted(data, key=lambda x: x['AT'])

lastct = min([x['AT'] for x in data])

for d in data:

lastct = lastct + d['BT']

d['CT'] = lastct

return data

def sjfcompletiontime(data):

queue = []

data = sorted(data, key=lambda x: (x['AT'], x['BT']))

loopend = sum([x['BT'] for x in data]) + min([x['AT'] for x in data])

completed = []

time = 0

while time != loopend + 1:

for x in data.copy():

if x['AT'] <= time:

queue.append(x)

data.remove(x)

queue = sorted(queue, key=lambda x: x['BT'])

if len(queue) != 0:

d = queue.pop(0)

time = time + d['BT']

d['CT'] = time

completed.append(d)

else:

time = time + 1

return completed

def sjfcompletiontime\_premptive(data):

queue = []

data = sorted(data, key=lambda x: (x['AT'], x['BT']))

completed = []

time = 0

for d in data: d['RT'] = d['BT']

n = len(data)

pcounter = 0

while pcounter != n:

for x in data.copy():

if x['AT'] <= time:

queue.append(x)

data.remove(x)

queue = sorted(queue, key=lambda x: x['BT'])

if len(queue) != 0:

d = queue.pop(0)

d['RT'] = d['RT'] - 1

if(d['RT'] == 0):

completed.append(d)

d['CT'] = time+1

pcounter+=1

else:

queue.append(d)

time = time + 1

else:

time = time + 1

for d in completed: del d['RT']

return completed

def roundrobincompletiontime(data,tq):

data = sorted(data, key=lambda x: x['AT'])

queue = []

completed = []

time = 0

n = len(data)

temp = {}

flag = False

pcounter = 0

for d in data: d['RT'] = d['BT']

while (n != pcounter):

for x in data.copy():

if x['AT'] <= time:

queue.append(x)

data.remove(x)

if flag:

queue.append(temp)

flag = False

if len(queue) != 0:

d = queue.pop(0)

if d['RT'] <= tq:

time += d['RT']

d['CT'] = time

d['RT'] = 0

completed.append(d)

pcounter += 1

else:

time += tq

d['RT'] -= tq

temp = d

flag = True

# print(d)

else:

time += 1

for d in completed: del d['RT']

return completed

def prioritycompletiontime(data):

queue = []

data = sorted(data, key=lambda x: (x['PRIORITY'], x['AT'], x['PID']),reverse = True)

completed = []

time = 0

n = len(data)

pcounter = 0

while pcounter != n:

for x in data.copy():

if x['AT'] <= time:

queue.append(x)

data.remove(x)

queue = sorted(queue, key=lambda x: (x['PRIORITY'], x['AT'], x['PID']),reverse = True)

if len(queue) != 0:

d = queue.pop(0)

time += d['BT']

d['CT'] = time

completed.append(d)

pcounter+=1

else:

time = time + 1

return sorted(completed,key= lambda x: x['PID'])

def turnaroundtime(data):

for d in data:

d['TAT'] = d['CT'] - d['AT']

def waitingtime(data):

for d in data:

d['WT'] = d['TAT'] - d['BT']

def displaytable(data):

print('PID \t','AT \t','BT \t','CT \t','TAT \t','WT')

wt = 0

tat = 0

for d in data:

print(' %s \t %2d \t %2d \t %2d \t %2d \t %2d' % tuple(d.values()))

wt = wt + d['WT']

tat = tat + d['TAT']

wt = wt / len(data)

tat = tat / len(data)

print()

print('Average Waiting Time : {:.2f}'.format(wt))

print('Average Turn Around Time : {:.2f} '.format(tat))

def display\_with\_priority(data):

print('PID \t', 'AT \t', 'BT \t', 'CT \t', 'TAT \t', 'WT \t', 'PRTY')

wt = 0

tat = 0

for d in data:

print(' %s \t %2d \t %2d \t %2d \t %2d \t %2d \t %2d' % tuple(d.values()))

wt = wt + d['WT']

tat = tat + d['TAT']

wt = wt / len(data)

tat = tat / len(data)

print()

print('Average Waiting Time : {:.2f}'.format(wt))

print('Average Turn Around Time : {:.2f} '.format(tat))

def showgraph(data,title):

import numpy as np

import matplotlib.pyplot as plt

avgwt = 0

for d in data: avgwt+=d['WT']

avgct = 0

for d in data: avgct += d['CT']

avgtat = 0

for d in data: avgtat += d['TAT']

height = [round(avgwt/len(data),2),round(avgtat/len(data),2),round(avgct/len(data),2)]

print(height)

bars = ('WT', 'TAT', 'CT')

y\_pos = np.arange(len(bars))

plt.bar(y\_pos, height)

plt.title(title)

plt.xticks(y\_pos, bars)

plt.show()

1. **FCFS Scheduling Algorithm**

**CODE:**

import cpuscheduling as cs

data = cs.readdata()

data = cs.fcfscompletiontime(data)

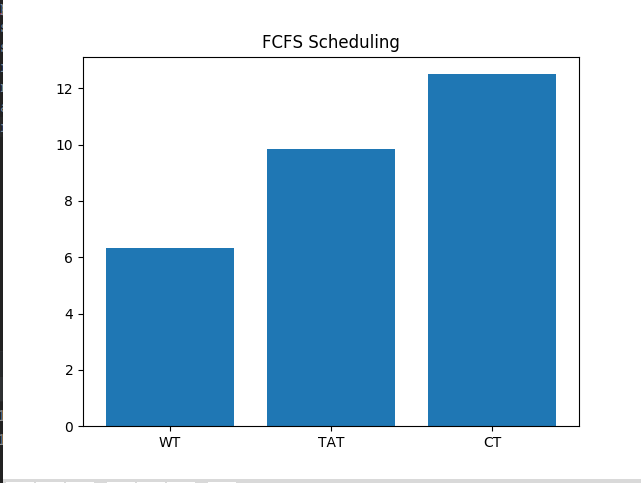
cs.turnaroundtime(data)

cs.waitingtime(data)

cs.displaytable(data)

cs.showgraph(data,'FCFS Scheduling')

**OUTPUT:**

****

1. **SJF Scheduling Algorithm**

**CODE:**

import cpuscheduling as cs

data = cs.readdata()

data = cs.sjfcompletiontime(data)

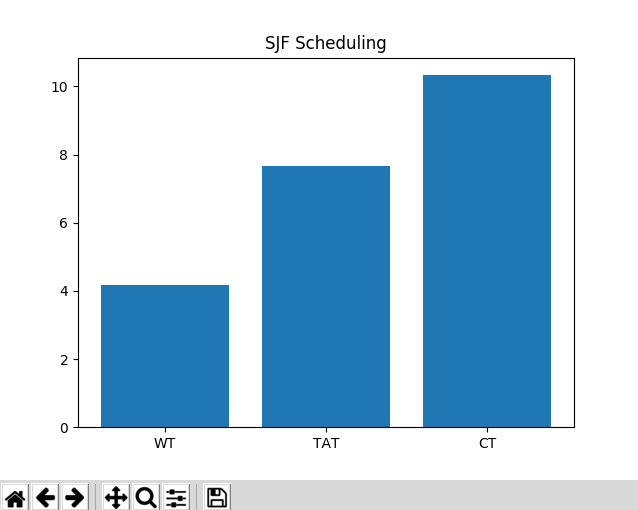
cs.turnaroundtime(data)

cs.waitingtime(data)

cs.displaytable(data)

cs.showgraph(data,'SJF Scheduling')

**OUTPUT:**

****

1. **ROUND-ROBIN Scheduling Algorithm**

**CODE:**

import cpuscheduling as cs

data = cs.readdata()

data = cs.roundrobincompletiontime(data)

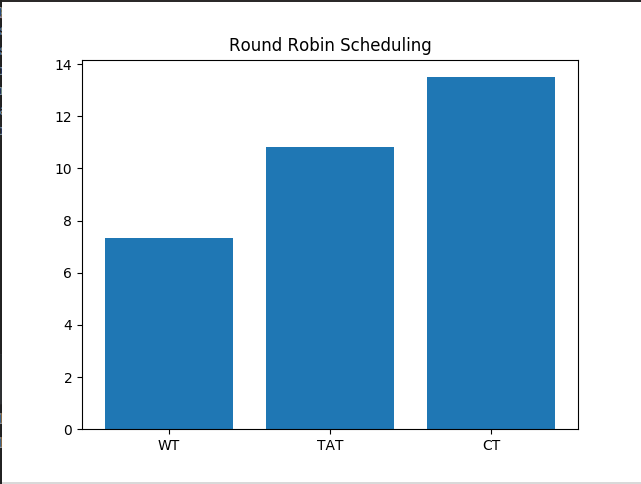
cs.turnaroundtime(data)

cs.waitingtime(data)

cs.displaytable(data)

cs.showgraph(data,'Round-Robin Scheduling')

**OUTPUT:**

****

1. **PRIORITY Scheduling Algorithm**

**CODE:**

import cpuscheduling as cs

data = cs.readdata()

data = cs.prioritycompletiontime (data)

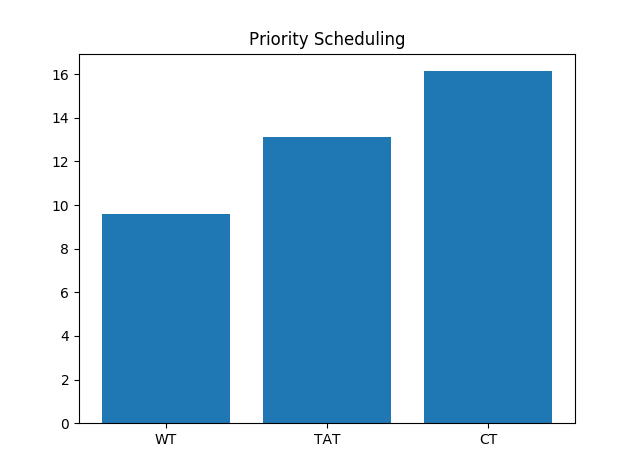
cs.turnaroundtime(data)

cs.waitingtime(data)

cs.displaytable(data)

cs.showgraph(data,'Priority Scheduling')

**OUTPUT:**

****

* **PRE LAB QUESTIONS**

1. **Define Operating system?**

An operating system (OS), in its most general sense, is software that allows a user to run other applications on a computing device. ... The operating system manages a computer's hardware resources, including: Input devices such as a keyboard and mouse. Output devices such as display monitors, printers and scanners.

1. **What are the different types of operating systems?**

* Batch Operating system
* Time Sharing Operating system
* Distributed Operating System
* Network Operating system
* Real-Time Operating system

1. **Define a process?**

a series of actions or steps taken in order to achieve a particular end.

1. **What is CPU Scheduling?**

**CPU scheduling** is a process which allows one process to use the **CPU** while the execution of another process is on hold(in waiting state) due to unavailability of any resource like I/O etc, thereby making full use of **CPU**. The aim of **CPU scheduling**is to make the system efficient, fast and fair.

1. **Define arrival time, burst time, waiting time and turn around time?**

**Arrival time :** Time when the process enter into the ready queue.

**Burst Time :** Time required to complete its task in CPU.

**Waiting Time :** Time Required to wait for cpu after arrival in ready queue.

**Turn Around Time :** Time difference of completion time and its arrival time

* **PRE LAB QUESTIONS**
* **What is the advantage of round robin CPU scheduling algorithm?**

A big advantage of round robin scheduling over **non**-preemptive schedulers is that it dramatically improves average response times. By limiting each task to a certain amount of time, the operating system can ensure that it can cycle through all ready tasks, giving each one a chance to run.

* **Which CPU scheduling algorithm is for real-time operating system?**

**Rate Monotonic**(RM) or **earliest Deadline First**(**EDF**) is generally used.

* **In general, which CPU scheduling algorithm works with highest waiting time?**

FCFS Scheduling Algorithm

* **Is it possible to use optimal CPU scheduling algorithm in practice?**
* **What is the real difficulty with the SJF CPU scheduling algorithm?**

We allocate CPU to shortest job so the job with more burst keeps on waiting. It is known as **starvation**. A particular process keeps on waiting, but it is not allocated the CPU. To prevent this, use **aging**.

**ASSIGNMENTS QUESTIONS:**

**Write a C program to simulate pre-emptive SJF CPU scheduling algorithm?**

**CODE:**

**Code for cpuscheduling is given on beginning of pratical**

import cpuscheduling as cs

data = cs.readdata()

cs.sjfcompletiontime\_premptive(data)

cs.turnaroundtime(data)

cs.waitingtime(data)

cs.displaytable(data)

cs.showgraph(data,"SJF Scheduling (Premptive)")