**Sample coding:**

from multiprocessing import Process

import os

def info (title):

print (title)

print ('module name:',\_\_name\_\_)

print ('parent process:', os.getppid())

print ('process id:', os.getpid())

def f(name):

info('function f')

print('hello',name)

if \_\_name\_\_ == '\_\_main\_\_':

info('main line')

p = Process(target = f, args = ('bob' ,))

p.start()

p.join()

print("child process :", p.name)

print ("child process ID :" , p.pid)

**Sample output:**

main line

module name: \_\_main\_\_

parent process: 2176

process id: 7120

child process : Process-1

child process ID : 8444

**Sample coding:**

n = int(input('Enter no of processes: '))

bt = [0] \* (n + 1)

at = [0] \* (n + 1)

abt = [0] \* (n + 1)

for i in range(n):

abt[i] = int(input('Enter the burst time for process {} : '.format(i + 1)))

at[i] = int(input('Enter the arrival time for process {} : '.format(i + 1)))

bt[i] = [abt[i], at[i], i]

bt.pop(-1)

print(abt)

print(bt)

sumbt = 0

i = 0

ll = []

for i in range(0, sum(abt)):

l = [j for j in bt if j[1] <= i]

l.sort(key=lambda x : x[0])

print(l,l[0][2])

bt[bt.index(l[0])][0] -= 1

for k in bt:

if k[0] == 0:

t = bt.pop(bt.index(k))

ll.append([k, i + 1])

print(ll)

ct = [0] \* (n + 1)

tat = [0] \* (n + 1)

wt = [0] \* (n + 1)

for i in ll:

ct[i[0][2]] = i[1]

for i in range(len(ct)):

tat[i] = ct[i] - at[i]

wt[i] = tat[i] - abt[i]

ct.pop(-1)

wt.pop(-1)

tat.pop(-1)

abt.pop(-1)

at.pop(-1)

print('PNo\tBT\tAT\tCT\tTAT\tWT')

for i in range(len(ct)):

print("{}\t{}\t{}\t{}\t{}\t{}\n".format(i+1,abt[i], at[i], ct[i], tat[i], wt[i]))

print('Average Waiting Time = ', sum(wt)/len(wt))

print('Average Turnaround Time = ', sum(tat)/len(tat))

**Sample output:**

Enter no of processes: 4

Enter the burst time for process 1 : 8

Enter the arrival time for process 1 : 0

Enter the burst time for process 2 : 4

Enter the arrival time for process 2 : 1

Enter the burst time for process 3 : 9

Enter the arrival time for process 3 : 2

Enter the burst time for process 4 : 5

Enter the arrival time for process 4 : 3

[8, 4, 9, 5, 0]

[[8, 0, 0], [4, 1, 1], [9, 2, 2], [5, 3, 3]]

[[8, 0, 0]] 0

[[4, 1, 1], [7, 0, 0]] 1

[[3, 1, 1], [7, 0, 0], [9, 2, 2]] 1

[[2, 1, 1], [5, 3, 3], [7, 0, 0], [9, 2, 2]] 1

[[1, 1, 1], [5, 3, 3], [7, 0, 0], [9, 2, 2]] 1

[[5, 3, 3], [7, 0, 0], [9, 2, 2]] 3

[[4, 3, 3], [7, 0, 0], [9, 2, 2]] 3

[[3, 3, 3], [7, 0, 0], [9, 2, 2]] 3

[[2, 3, 3], [7, 0, 0], [9, 2, 2]] 3

[[1, 3, 3], [7, 0, 0], [9, 2, 2]] 3

[[7, 0, 0], [9, 2, 2]] 0

[[6, 0, 0], [9, 2, 2]] 0

[[5, 0, 0], [9, 2, 2]] 0

[[4, 0, 0], [9, 2, 2]] 0

[[3, 0, 0], [9, 2, 2]] 0

[[2, 0, 0], [9, 2, 2]] 0

[[1, 0, 0], [9, 2, 2]] 0

[[9, 2, 2]] 2

[[8, 2, 2]] 2

[[7, 2, 2]] 2

[[6, 2, 2]] 2

[[5, 2, 2]] 2

[[4, 2, 2]] 2

[[3, 2, 2]] 2

[[2, 2, 2]] 2

[[1, 2, 2]] 2

P. NO:

[[[0, 1, 1], 5], [[0, 3, 3], 10], [[0, 0, 0], 17], [[0, 2, 2], 26]]

PNo BT AT CT TAT WT

1 8 0 17 17 9

2 4 1 5 4 0

3 9 2 26 24 15

4 5 3 10 7 2

Average Waiting Time = 6.5

Average Turnaround Time = 13.0

**Sample coding:**

print("FIRST COME FIRST SERVE SCHEDULLING")

n= int(input("Enter number of processes : "))

d = dict()

for i in range(n):

key = "P"+str(i+1)

a = int(input("Enter arrival time of process"+str(i+1)+": "))

b = int(input("Enter burst time of process"+str(i+1)+": "))

l = []

l.append(a)

l.append(b)

d[key] = l

d = sorted(d.items(), key=lambda item: item[1][0])

ET = []

for i in range(len(d)):

if(i==0):

ET.append(d[i][1][1])

else:

ET.append(ET[i-1] + d[i][1][1])

TAT = []

for i in range(len(d)):

TAT.append(ET[i] - d[i][1][0])

WT = []

for i in range(len(d)):

WT.append(TAT[i] - d[i][1][1])

avg\_WT = 0

for i in WT:

avg\_WT +=i

avg\_WT = (avg\_WT/n)

print("Process | Arrival | Burst | Exit | Turn Around | Wait |")

for i in range(n):

print(" ",d[i][0]," | ",d[i][1][0]," | ",d[i][1][1]," | ",ET[i]," | ",TAT[i]," | ",WT[i]," | ")

print("Average Waiting Time: ",avg\_WT)

P. NO:

**Sample output:**

FIRST COME FIRST SERVE SCHEDULLING

Enter number of processes : 4

Enter arrival time of process1: 1

Enter burst time of process1: 5

Enter arrival time of process2: 0

Enter burst time of process2: 4

Enter arrival time of process3: 3

Enter burst time of process3: 3

Enter arrival time of process4: 2

Enter burst time of process4: 5

Process | Arrival | Burst | Exit | Turn Around | Wait |

P2 | 0 | 4 | 4 | 4 | 0 |

P1 | 1 | 5 | 9 | 8 | 3 |

P4 | 2 | 5 | 14 | 12 | 7 |

P3 | 3 | 3 | 17 | 14 | 11 |

Average Waiting Time: 5.25

**Sample coding:**

1. **Round Robin**

def findwaitingTime(Processes, n, bt,wt, quantum):

rem\_bt = [0] \* n

for i in range(n):

rem\_bt[i] = bt[i]

t = 0

while(1):

done = True

for i in range(n):

if (rem\_bt[i] > 0):

done = False

if (rem\_bt[i] > quantum):

t += quantum

rem\_bt[i] -= quantum

else:

t = t + rem\_bt[i]

wt[i] = t - bt[i]

rem\_bt[i] = 0

if (done == True):

break

def findTurnAroundTime(Processes, n, bt, wt, tat):

for i in range(n):

tat[i] = bt[i] + wt[i]

def findavgTime(Processes, n, bt, quantum):

wt = [0] \* n

tat = [0] \* n

findwaitingTime(Processes, n, bt,wt, quantum)

findTurnAroundTime(Processes, n, bt,wt, tat)

print("Processes Burst Time Waiting","Time Turn-Around Time")

total\_wt = 0

total\_tat = 0

for i in range(n):

total\_wt = 0

total\_tat = 0

for i in range(n):

total\_wt = total\_wt + wt[i]

total\_tat = total\_tat + tat[i]

print("", i + 1,"\t\t",bt[i],"\t\t",wt[i],"\t\t",tat[i])

print("\nAverage waiting time = %.5f "%(total\_wt /n))

print("Average turn around time = %.5f "%(total\_tat /n))

P. NO:

if \_\_name\_\_ == "\_\_main\_\_":

proc = [1,2,3]

n = 3

burst\_time = [24, 3, 3]

quantum = 4;

findavgTime(proc, n, burst\_time, quantum)

1. **Priority scheduling:**

def findWaitingTime(processes, n, wt):

wt[0] = 0

for i in range(1, n):

wt[i] = processes[i - 1][1] + wt[i - 1]

def findTurnAroundTime(processes, n, wt, tat):

for i in range(n):

tat[i] = processes[i][1] + wt[i]

def findavgTime(processes, n):

wt = [0] \* n

tat = [0] \* n

findWaitingTime(processes, n, wt)

findTurnAroundTime(processes, n, wt, tat)

print("\nProcesses Burst Time Waiting",

"Time Turn-Around Time")

total\_wt = 0

total\_tat = 0

for i in range(n):

total\_wt = total\_wt + wt[i]

total\_tat = total\_tat + tat[i]

print(" ", processes[i][0], "\t\t",

processes[i][1], "\t\t",

wt[i], "\t\t", tat[i])

print("\nAverage waiting time = %.5f "%(total\_wt /n))

print("Average turn around time = ", total\_tat / n)

def priorityScheduling(proc, n):

proc = sorted(proc, key = lambda proc:proc[2],

reverse = True);

print("Order in which processes gets executed")

for i in proc:

print(i[0], end = " ")

findavgTime(proc, n)

P. NO:

if \_\_name\_\_ =="\_\_main\_\_":

proc = [[1, 10, 1],

[2, 5, 0],

[3, 8, 1]]

n = 3

priorityScheduling(proc, n)

**Sample output:**

**a)**

Processes Burst Time Waiting Time Turn-Around Time

1 24 6 30

2 3 27 30

3 3 27 30

Average waiting time = 20.00000

Average turn around time = 30.00000

1 24 6 30

2 3 27 30

3 3 27 30

Average waiting time = 20.00000

Average turn around time = 30.00000

1 24 6 30

2 3 27 30

3 3 27 30

Average waiting time = 20.00000

Average turn around time = 30.00000

**b)**

Order in which processes gets executed

1 3 2

Processes Burst Time Waiting Time Turn-Around Time

1 10 0 10

3 8 10 18

2 5 18 23

Average waiting time = 9.33333

Average turn around time = 17.0

**Sample coding:**

import threading as thread

import random

global x

x = 0

lock = thread.Lock()

def Reader():

global x

print('Reader is Reading!')

lock.acquire()

print('Shared Data:', x)

lock.release()

print()

def Writer():

global x

print('Writer is Writing!')

lock.acquire()

x += 1

print('Writer is Releasing the lock!')

lock.release()

print()

if \_\_name\_\_ == '\_\_main\_\_':

for i in range(0, 10):

randomNumber = random.randint(0, 100)

if(randomNumber > 50):

Thread1 = thread.Thread(target = Reader)

Thread1.start()

else:

Thread2 = thread.Thread(target = Writer)

Thread2.start()

Thread1.join()

Thread2.join()

P. NO:

**Sample output:**

Reader is Reading!Reader is Reading!Writer is Writing!Reader is Reading!Reader is Reading!Writer is Writing!Writer is Writing!Writer is Writing!Reader is Reading!Reader is Reading!

Shared Data: 0

Shared Data:

0

Writer is Releasing the lock!

Shared Data: 1

Shared Data: 1

Writer is Releasing the lock!

Writer is Releasing the lock!

Writer is Releasing the lock!

Shared Data: 4

Shared Data:

4

**Sample coding:**

P = 5

R = 3

def calculateNeed(need, maxm, allot):

for i in range(P):

for j in range(R):

need[i][j] = maxm[i][j] - allot[i][j]

def isSafe(processes, avail, maxm, allot):

need = []

for i in range(P):

l = []

for j in range(R):

l.append(0)

need.append(l)

calculateNeed(need, maxm, allot)

finish = [0] \* P

safeSeq = [0] \* P

work = [0] \* R

for i in range(R):

work[i] = avail[i]

count = 0

while (count < P):

found = False

for p in range(P):

if (finish[p] == 0):

for j in range(R):

if (need[p][j] > work[j]):

break

if (j == R - 1):

for k in range(R):

work[k] += allot[p][k]

safeSeq[count] = p

count += 1

finish[p] = 1

found = True

if (found == False):

print("System is not in safe state")

return False

print("System is in safe state.",

"\nSafe sequence is: ", end = " ")

print(\*safeSeq)

P. NO:

return True

if \_\_name\_\_ =="\_\_main\_\_":

processes = [0, 1, 2, 3, 4]

avail = [3, 3, 2]

maxm = [[7, 5, 3], [3, 2, 2],

[9, 0, 2], [2, 2, 2],

[4, 3, 3]]

allot = [[0, 1, 0], [2, 0, 0],

[3, 0, 2], [2, 1, 1],

[0, 0, 2]]

isSafe(processes, avail, maxm, allot)

**Sample output:**

System is in safe state.

Safe sequence is: 1 3 4 0 2

**Sample coding:**

from queue import Queue

def pagefaults(pages, n, capacity):

s = set()

indexes = Queue()

page\_faults = 0

for i in range(n):

if (len(s) < capacity):

if (pages[i] not in s):

s.add(pages[i])

page\_faults += 1

indexes.put(pages[i])

else:

if (pages[i] not in s):

val = indexes.queue[0]

indexes.get()

s.remove(val)

s.add(pages[i])

indexes.put(pages[i])

page\_faults += 1

print(s,end= " ")

print("page fault count",page\_faults)

return page\_faults

if \_\_name\_\_ == '\_\_main\_\_':

pages = [ 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4]

n = len(pages)

capacity = 3

print("Total page fault count",pagefaults(pages, n, capacity))

**Sample output:**

{3} page fault count 1

{2, 3} page fault count 2

{1, 2, 3} page fault count 3

{0, 1, 2} page fault count 4

{0, 1, 3} page fault count 5

{0, 2, 3} page fault count 6

{2, 3, 4} page fault count 7

{2, 3, 4} page fault count 7

{2, 3, 4} page fault count 7

{1, 2, 4} page fault count 8

{0, 1, 4} page fault count 9

{0, 1, 4} page fault count 9

Total page fault count 9

**Sample coding:**

import collections

class SimpleLRUCache:

def \_\_init\_\_(self, size):

self.size = size

self.lru\_cache = collections.OrderedDict()

def get(self, key):

try:

value = self.lru\_cache.pop(key)

self.lru\_cache[key] = value

return value

except KeyError:

return -1

def put(self, key, value):

try:

self.lru\_cache.pop(key)

except KeyError:

if len(self.lru\_cache) >= self.size:

self.lru\_cache.popitem(last=False)

self.lru\_cache[key] = value

def show\_entries(self):

print(self.lru\_cache)

cache = SimpleLRUCache(3)

cache.put("1","1")

cache.put("2","2")

cache.put("3","3")

cache.get("1")

cache.get("3")

cache.put("4","4") # This will replace 2

cache.show\_entries() # shows 1,3,4

cache.put("5","5") # This will replace 1

cache.show\_entries() # shows 3,4,5

**Sample output:**

OrderedDict([('1', '1'), ('3', '3'), ('4', '4')])

OrderedDict([('3', '3'), ('4', '4'), ('5', '5')])

**Sample coding:**

1. **First fit:**

def firstFit(blockSize, m, processSize, n):

allocation = [-1] \* n

for i in range(n):

for j in range(m):

if blockSize[j] >= processSize[i]:

allocation[i] = j

blockSize[j] -= processSize[i]

break

print(" Process No. Process Size Block no.")

for i in range(n):

print(" ", i + 1, " ", processSize[i],

" ", end = " ")

if allocation[i] != -1:

print(allocation[i] + 1)

else:

print("Not Allocated")

if \_\_name\_\_ == '\_\_main\_\_':

blockSize = [100, 500, 200, 300, 600]

processSize = [212, 417, 112, 426]

m = len(blockSize)

n = len(processSize)

firstFit(blockSize, m, processSize, n)

1. **Sample output:**

Process No. Process Size Block no.

1 212 2

2 417 5

3 112 2

4 426 Not Allocated

P. NO:

**Sample coding:**

1. **Best fit:**

def bestFit(blockSize, m, processSize, n):

allocation = [-1] \* n

for i in range(n):

bestIdx = -1

for j in range(m):

if blockSize[j] >= processSize[i]:

if bestIdx == -1:

bestIdx = j

elif blockSize[bestIdx] > blockSize[j]:

bestIdx = j

if bestIdx != -1:

allocation[i] = bestIdx

blockSize[bestIdx] -= processSize[i]

print("Process No. Process Size Block no.")

for i in range(n):

print(i + 1, " ", processSize[i],

end = " ")

if allocation[i] != -1:

print(allocation[i] + 1)

else:

print("Not Allocated")

if \_\_name\_\_ == '\_\_main\_\_':

blockSize = [100, 500, 200, 300, 600]

processSize = [212, 417, 112, 426]

m = len(blockSize)

n = len(processSize)

bestFit(blockSize, m, processSize, n)

1. **Sample output:**

Process No. Process Size Block no.

Process No. Process Size Block no.

Process No. Process Size Block no.

Process No. Process Size Block no.

1 212 4

2 417 2

3 112 3

4 426 5

**Sample coding:**

1. **Worst fit:**

def worstFit(blockSize, m, processSize, n):

allocation = [-1] \* n

for i in range(n):

wstIdx = -1

for j in range(m):

if blockSize[j] >= processSize[i]:

if wstIdx == -1:

wstIdx = j

elif blockSize[wstIdx] < blockSize[j]:

wstIdx = j

if wstIdx != -1:

allocation[i] = wstIdx

blockSize[wstIdx] -= processSize[i]

print("Process No. Process Size Block no.")

for i in range(n):

print(i + 1, " ",

processSize[i], end = " ")

if allocation[i] != -1:

print(allocation[i] + 1)

else:

print("Not Allocated")

if \_\_name\_\_ == '\_\_main\_\_':

blockSize = [100, 500, 200, 300, 600]

processSize = [212, 417, 112, 426]

m = len(blockSize)

n = len(processSize)

worstFit(blockSize, m, processSize, n)

1. **Sample output:**

Process No. Process Size Block no.

1 212 5

2 417 2

3 112 5

4 426 Not Allocated

**Sample coding:**

from multiprocessing import Process, Value, Array

def f(n, a):

n.value = 3.1415927

for i in range(len(a)):

a[i] = -a[i]

if \_\_name\_\_ == '\_\_main\_\_':

num = Value('d', 0.0)

arr = Array('i', range(10))

arr1 = Array('i' ,range(1,20,2))

print("\t\tIPC using shared memory")

p1 = Process(target = f, args = (num, arr))

p1.start()

p1.join()

p2 = Process(target = f, args = (num, arr1))

p2.start()

p2.join()

print(num.value)

print(arr[:])

print(arr1[:])

**Sample output:**

IPC using shared memory

3.1415927

[0, -1, -2, -3, -4, -5, -6, -7, -8, -9]

[-1, -3, -5, -7, -9, -11, -13, -15, -17, -19]