**SET1**

**FCFS**

n=int(input("Enter no.of processes:"))

p=[]

et=[]

wt=[]

tat=[]

for i in range(n):

at=int(input("Enter arrival time for p{}:".format(i+1)))

bt=int(input("Enter burst time for p{}:".format(i+1)))

p.append([at,bt])

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

for i in range(len(p)):

et.append(p[i][1] if i==0 else et[i-1]+p[i][1])

tat.append(et[i]-p[i][0])

wt.append(tat[i]-p[i][1])

print(sum(wt)/n)

**Demonstrate Bankers Algorithm for Deadlock Detection.**

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

m = int(input("Enter the number of resources: "))

alloc = []

max = []

avail = []

print("Enter the allocation matrix:")

for i in range(n):

row = list(map(int, input().split()))

alloc.append(row)

print("Enter the maximum matrix:")

for i in range(n):

row = list(map(int, input().split()))

max.append(row)

print("Enter the available resources:")

avail = list(map(int, input().split()))

f = [0] \* n

ans = []

need = [[max[i][j] - alloc[i][j] for j in range(m)] for i in range(n)]

while True:

for i in range(n):

if f[i] == 0 and all(need[i][j] <= avail[j] for j in range(m)):

ans.append(i)

avail = [avail[j] + alloc[i][j] for j in range(m)]

f[i] = 1

if len(ans) == n:

print("System is in a safe state.")

print("Safe sequence:", "P" + " -> P".join(map(str, ans)))

break

if not any(f[i] == 0 and all(need[i][j] <= avail[j] for j in range(m)) for i in range(n)):

print("System is in an unsafe state. Deadlock detected.")

break

**SET2**

**Demonstrate Producer and Consumer problem using Semaphores.**

import threading

import time

mutex = threading.Lock()

full = 0

empty = 10

x = 0

def producer():

global mutex, full, empty, x

mutex.acquire()

if full < 10:

full += 1

empty -= 1

x += 1

print("Producer produces item", x)

else:

print("Buffer is full!")

mutex.release()

def consumer():

global mutex, full, empty, x

mutex.acquire()

if full > 0:

full -= 1

empty += 1

print("Consumer consumes item", x)

x -= 1

else:

print("Buffer is empty!")

mutex.release()

# Driver Code

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

while True:

print("\n1. Press 1 for Producer")

print("2. Press 2 for Consumer")

print("3. Press 3 for Exit")

n = int(input("Enter your choice: "))

if n == 1:

producer()

elif n == 2:

consumer()

elif n == 3:

break

else:

print("Invalid choice!")

time.sleep(1)

**ROUND ROBIN(NON-PREEPMTIVE)**

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

bt = [int(input(f"Enter burst time for process {i + 1}: ")) for i in range(n)]

q = int(input("Enter the time quantum: "))

wt = [0] \* n

tat = [0] \* n

while True:

done = True

for i in range(n):

if bt[i] > 0:

done = False

time = min(bt[i], q)

bt[i] -= time

wt[i] += time

if done:

break

tat = [bt[i] + wt[i] for i in range(n)]

print("Average waiting time =", sum(wt) / n)

print("Average turnaround time =", sum(tat) / n)

**SET3**

**SJF PREEPMTIVE :**

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

p = []

wt = [0] \* n

tat = [0] \* n

rt = [0] \* n

complete = 0

t = 0

total\_wt = 0

total\_tat = 0

for i in range(n):

at = int(input("Enter arrival time for process {}: ".format(i + 1)))

bt = int(input("Enter burst time for process {}: ".format(i + 1)))

p.append([i + 1, bt, at])

rt[i] = bt

while complete != n:

ep = [(rt[i], i) for i in range(n) if p[i][2] <= t and rt[i] > 0]

if ep:

s = min(ep, key=lambda x: x[0])[1]

rt[s] -= 1

t += 1

if rt[s] == 0:

complete += 1

wt[s] = t - p[s][2] - p[s][1]

wt[s] = max(0, wt[s])

tat[s] = p[s][1] + wt[s]

total\_wt += wt[s]

total\_tat += tat[s]

else:

t += 1

avg\_wt = total\_wt / n

avg\_tat = total\_tat / n

print("\nAverage Waiting Time =",total\_wt / n)

print("Average Turnaround Time =",total\_tat / n)

**Demonstrate Bankers Algorithm for Deadlock Avoidance.**

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

m = int(input("Enter the number of resources: "))

alloc = []

max = []

avail = []

print("Enter the allocation matrix:")

for i in range(n):

row = list(map(int, input().split()))

alloc.append(row)

print("Enter the maximum matrix:")

for i in range(n):

row = list(map(int, input().split()))

max.append(row)

print("Enter the available resources:")

avail = list(map(int, input().split()))

f = [0] \* n

ans = []

need = [[max[i][j] - alloc[i][j] for j in range(m)] for i in range(n)]

for \_ in range(n):

for i in range(n):

if f[i] == 0 and all(need[i][j] <= avail[j] for j in range(m)):

ans.append(i)

for y in range(m):

avail[y] += alloc[i][y]

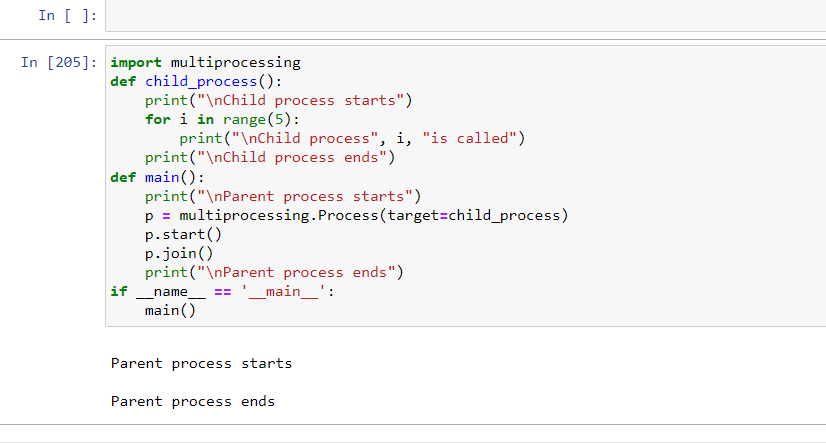
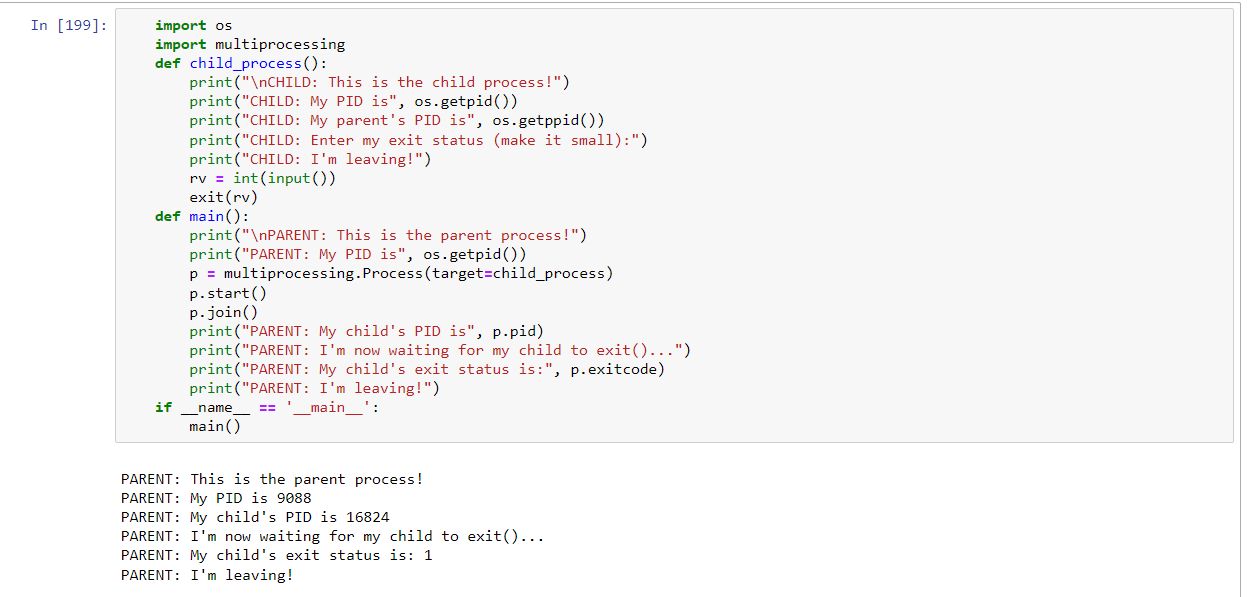
f[i] = 1

print("Following is the SAFE Sequence:")

print("P", " -> P".join(map(str, ans)), sep="")

**SET4**

**SYSTEM CALLS**



**Priority preemptive**

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

proc = []

for i in range(n):

arrival\_time = int(input(f"Enter arrival time for process {i+1}: "))

burst\_time = int(input(f"Enter burst time for process {i+1}: "))

priority = int(input(f"Enter priority for process {i+1}: "))

proc.append([arrival\_time, burst\_time, priority, i + 1])

proc.sort()

wt = [0] \* n

tat = [0] \* n

wavg = 0

tavg = 0

service = 0

for i in range(n):

service += proc[i][1]

tat[i] = service - proc[i][0]

wt[i] = tat[i] - proc[i][1]

wavg += wt[i]

tavg += tat[i]

print("Average waiting time: ", wavg / n)

print("Average turnaround time: ", tavg / n)

**SET5**

**SJF NON PREEMPTIVE**

n = int(input("Enter the value of n: "))

p = []

for i in range(n):

pr = input("Enter pid: ")

bt = int(input("Enter burst time: "))

p.append([pr, bt])

p.sort(key=lambda x: x[1])

et = []

wtsum = 0

for i in range(n):

et.append(sum([p[j][1] for j in range(i + 1)]))

wtsum += et[i] if i == 0 else et[i - 1]

print("Sorted order of list is:", p)

print("Average waiting time:", wtsum / n)

**Demonstrate Producer and Consumer problem using Semaphores.**

import threading

import time

mutex = threading.Lock()

full = 0

empty = 10

x = 0

def producer():

global mutex, full, empty, x

mutex.acquire()

if full < 10:

full += 1

empty -= 1

x += 1

print("Producer produces item", x)

else:

print("Buffer is full!")

mutex.release()

def consumer():

global mutex, full, empty, x

mutex.acquire()

if full > 0:

full -= 1

empty += 1

print("Consumer consumes item", x)

x -= 1

else:

print("Buffer is empty!")

mutex.release()

# Driver Code

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

while True:

print("\n1. Press 1 for Producer")

print("2. Press 2 for Consumer")

print("3. Press 3 for Exit")

n = int(input("Enter your choice: "))

if n == 1:

producer()

elif n == 2:

consumer()

elif n == 3:

break

else:

print("Invalid choice!")

time.sleep(1)

**SET6**

**PRIORITY NON PREEMPTIVE**

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

p = []

for i in range(n):

p1 = {}

p1["name"] = input(f"Enter the name of process {i+1}: ")

p1["priority"] = int(input(f"Enter the priority of process {i+1}: "))

p1["bt"] = int(input(f"Enter the burst time of process {i+1}: "))

p.append(p1)

p = sorted(p, key=lambda x: x["priority"])

wt = [0]

tat = [p[0]["bt"]]

for i in range(1, n):

wt.append(wt[i-1] + p[i-1]["bt"])

tat.append(wt[i] + p[i]["bt"])

print("\nAverage Waiting Time:", sum(wt) / n)

print("Average Turnaround Time:", sum(tat) / n)

**Demonstrate Bankers Algorithm for Deadlock Avoidance.**

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

m = int(input("Enter the number of resources: "))

alloc = []

max = []

avail = []

print("Enter the allocation matrix:")

for i in range(n):

row = list(map(int, input().split()))

alloc.append(row)

print("Enter the maximum matrix:")

for i in range(n):

row = list(map(int, input().split()))

max.append(row)

print("Enter the available resources:")

avail = list(map(int, input().split()))

f = [0] \* n

ans = []

need = [[max[i][j] - alloc[i][j] for j in range(m)] for i in range(n)]

for \_ in range(n):

for i in range(n):

if f[i] == 0 and all(need[i][j] <= avail[j] for j in range(m)):

ans.append(i)

for y in range(m):

avail[y] += alloc[i][y]

f[i] = 1

print("Following is the SAFE Sequence:")

print("P", " -> P".join(map(str, ans)), sep="")