Exercises

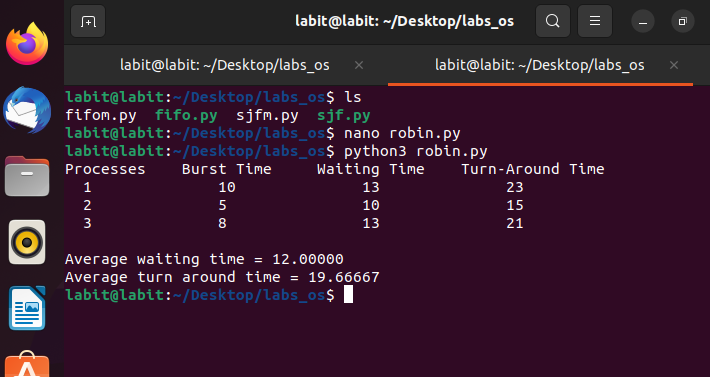
1. Write a Python program to implement Round Robin Algorithm.

Ans:

robin.py:

|  |
| --- |
| def findWaitingTime(processes, n, bt,  wt, quantum):  rem\_bt = [0] \* n  for i in range(n):  rem\_bt[i] = bt[i]  t = 0 # Current time  while(1):  done = True  for i in range(n):  if (rem\_bt[i] > 0):  done = False # There is a pending process  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)  # Display processes along with all details  print("Processes 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(" ", 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))  if \_\_name\_\_ == "\_\_main\_\_":  proc = [1, 2, 3]  n = 3  burst\_time = [10, 5, 8]  quantum = 2  findavgTime(proc, n, burst\_time, quantum) |

OUTPUT:



2. Modify the algorithm with different arrival, burst time and quantum time then observe the output.

Ans:

robinm.py

|  |
| --- |
| class Process:  def \_\_init\_\_(self, pid, burst\_time, arrival\_time):  self.pid = pid  self.burst\_time = burst\_time  self.arrival\_time = arrival\_time  def findWaitingTime(processes, n, wt, quantum):  rem\_bt = [p.burst\_time for p in processes] # Remaining burst time  t = 0 # Current time  ready\_queue = [] # Queue of process indices that have arrived  arrived = [False] \* n # Track which processes have been added to queue  index = 0 # To iterate through processes sorted by arrival time    # Sort processes by arrival time to process arrivals in order  processes.sort(key=lambda x: x.arrival\_time)    while True:  # Add processes that have arrived by current time to ready queue  while index < n and processes[index].arrival\_time <= t:  ready\_queue.append(index)  arrived[index] = True  index += 1    if not ready\_queue:  # If no process is ready, advance time to next arrival  if index < n:  t = processes[index].arrival\_time  else:  break # All processes completed    # Process the first process in the ready queue  i = ready\_queue.pop(0)  if rem\_bt[i] > quantum:  t += quantum  rem\_bt[i] -= quantum  # Re-add process to queue if it still has remaining burst time  ready\_queue.append(i)  else:  t += rem\_bt[i]  wt[i] = t - processes[i].burst\_time - processes[i].arrival\_time  rem\_bt[i] = 0    # Check if all processes are done  if all(bt == 0 for bt in rem\_bt):  break  def findTurnAroundTime(processes, n, wt, tat):  for i in range(n):  tat[i] = processes[i].burst\_time + wt[i]  def findavgTime(processes, n, quantum):  wt = [0] \* n  tat = [0] \* n  findWaitingTime(processes, n, wt, quantum)  findTurnAroundTime(processes, n, wt, tat)  print("Processes\tBurst Time\tArrival Time\tWaiting Time\tTurn-Around Time")  total\_wt = 0  total\_tat = 0  for i in range(n):  total\_wt += wt[i]  total\_tat += tat[i]  print(f" {processes[i].pid}\t\t{processes[i].burst\_time}\t\t{processes[i].arrival\_time}\t\t{wt[i]}\t\t{tat[i]}")  print(f"\nAverage waiting time = {total\_wt / n:.5f}")  print(f"Average turn around time = {total\_tat / n:.5f}")  if \_\_name\_\_ == "\_\_main\_\_":  processes = [  Process(1, 10, 0),  Process(2, 5, 1),  Process(3, 8, 2)  ]  n = len(processes)  quantum = 2  findavgTime(processes, n, quantum) |

OUTPUT:

