Exercises

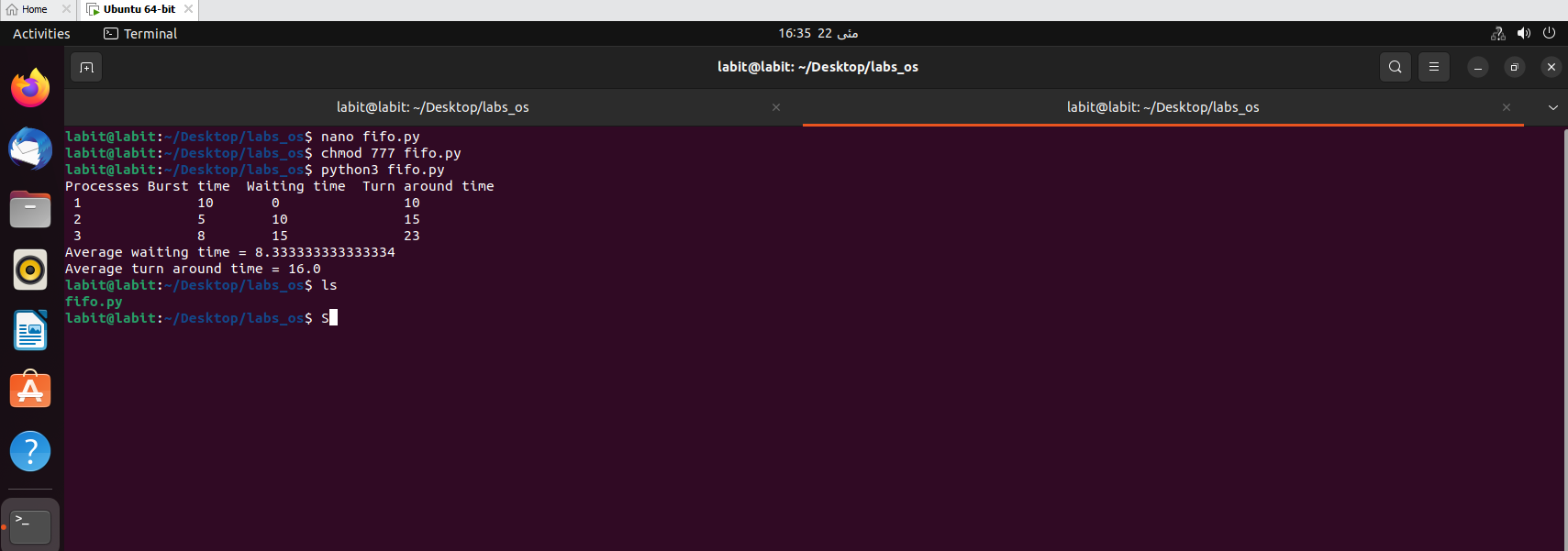
1. Write a Python program to implement and simulate the FCFS Algorithm.

Ans:

fifo.py:

|  |
| --- |
| def findWaitingTime(processes, n, bt, wt):  wt[0] = 0  for i in range(1, n ):  wt[i] = bt[i - 1] + wt[i - 1]  def findTurnAroundTime(processes, n,  bt, wt, tat):  for i in range(n):  tat[i] = bt[i] + wt[i]  def findavgTime( processes, n, bt):  wt = [0] \* n  tat = [0] \* n  total\_wt = 0  total\_tat = 0  findWaitingTime(processes, n, bt, wt)  findTurnAroundTime(processes, n,  bt, wt, tat)  print( "Processes Burst time " +  " Waiting time " +  " Turn around time")  for i in range(n):  total\_wt = total\_wt + wt[i]  total\_tat = total\_tat + tat[i]  print(" " + str(i + 1) + "\t\t" +  str(bt[i]) + "\t " +  str(wt[i]) + "\t\t " +  str(tat[i]))  print( "Average waiting time = "+  str(total\_wt / n))  print("Average turn around time = "+  str(total\_tat / n))  if \_\_name\_\_ =="\_\_main\_\_":  processes = [ 1, 2, 3]  n = len(processes)  burst\_time = [10, 5, 8]  findavgTime(processes, n, burst\_time) |

OUTPUT:



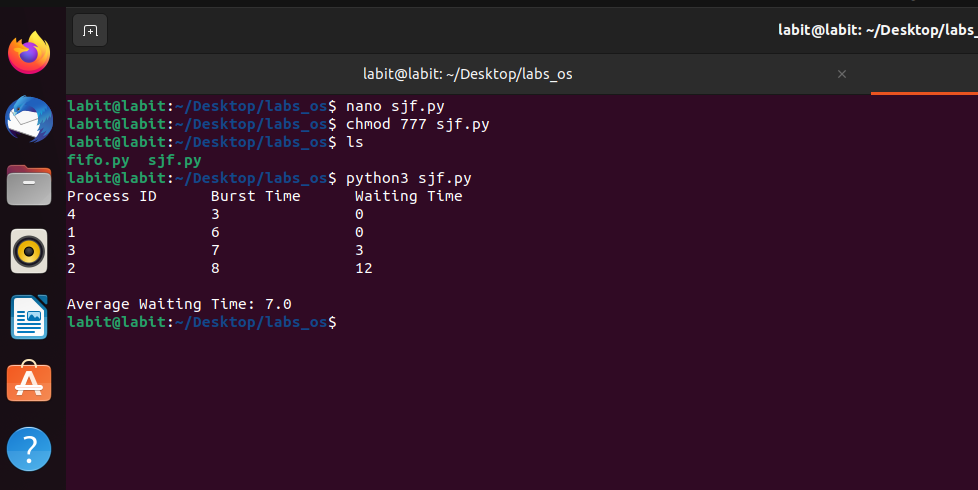
2. Write a Python program to implement and simulate the SJF Algorithm.

Ans:

sjf.py

|  |
| --- |
| class Process:  def \_\_init\_\_(self, pid, burst\_time):  self.pid = pid  self.burst\_time = burst\_time  def sjf\_scheduling(processes):  processes.sort(key=lambda x: x.burst\_time) # Sort processes by burst time  total\_processes = len(processes)  current\_time = 0  waiting\_time = 0    print("Process ID\tBurst Time\tWaiting Time")  for p in processes:  print(f"{p.pid}\t\t{p.burst\_time}\t\t{waiting\_time}")  waiting\_time += current\_time  current\_time += p.burst\_time  average\_waiting\_time = waiting\_time / total\_processes  print("\nAverage Waiting Time:", average\_waiting\_time)  if \_\_name\_\_ == "\_\_main\_\_":  processes = [Process(1, 6), Process(2, 8), Process(3, 7), Process(4, 3)]  sjf\_scheduling(processes) |

OUTPUT:



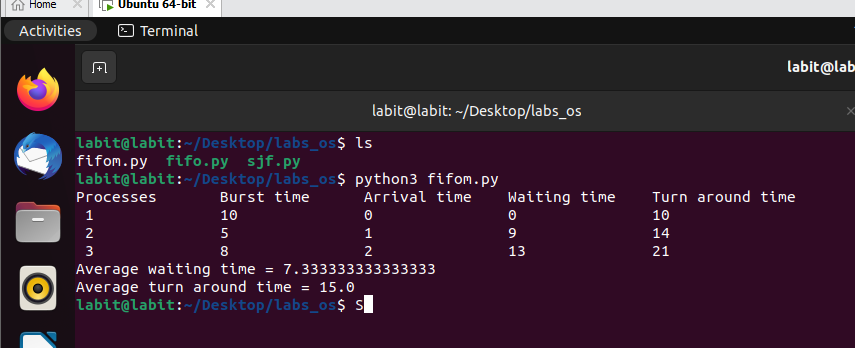
3. Modify both algorithms for the different arrival time.

Ans:

Fifom.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):  current\_time = 0  for i in range(n):  # Wait until the process arrives  if current\_time < processes[i].arrival\_time:  current\_time = processes[i].arrival\_time  # Waiting time = time when process starts - arrival time  wt[i] = current\_time - processes[i].arrival\_time  # Update current time after process execution  current\_time += processes[i].burst\_time  def findTurnAroundTime(processes, n, wt, tat):  for i in range(n):  # Turnaround time = burst time + waiting time  tat[i] = processes[i].burst\_time + wt[i]  def findavgTime(processes, n):  wt = [0] \* n  tat = [0] \* n  total\_wt = 0  total\_tat = 0    # Sort processes by arrival time to simulate FCFS  processes.sort(key=lambda x: x.arrival\_time)    findWaitingTime(processes, n, wt)  findTurnAroundTime(processes, n, wt, tat)    print("Processes\tBurst time\tArrival time\tWaiting time\tTurn around time")  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"Average waiting time = {total\_wt / n}")  print(f"Average turn around time = {total\_tat / n}")  if \_\_name\_\_ == "\_\_main\_\_":  processes = [  Process(1, 10, 0),  Process(2, 5, 1),  Process(3, 8, 2)  ]  n = len(processes)  findavgTime(processes, n) |

OUTPUT:



Sjfm.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 sjf\_scheduling(processes):  # Sort processes by arrival time initially to process them in order of arrival  processes.sort(key=lambda x: x.arrival\_time)  total\_processes = len(processes)  current\_time = 0  waiting\_time = 0  completed = []  ready\_queue = []  index = 0  print("Process ID\tBurst Time\tArrival Time\tWaiting Time")    while len(completed) < total\_processes:  # Add all processes that have arrived by current\_time to the ready queue  while index < total\_processes and processes[index].arrival\_time <= current\_time:  ready\_queue.append(processes[index])  index += 1    if ready\_queue:  # Sort ready queue by burst time to pick the shortest job  ready\_queue.sort(key=lambda x: x.burst\_time)  current\_process = ready\_queue.pop(0) # Pick the process with shortest burst time    # Calculate waiting time for the current process  wait\_time = current\_time - current\_process.arrival\_time  if wait\_time < 0:  wait\_time = 0 # Ensure waiting time is not negative  waiting\_time += wait\_time    # Print process details  print(f"{current\_process.pid}\t\t{current\_process.burst\_time}\t\t{current\_process.arrival\_time}\t\t{wait\_time}")    # Update current time  current\_time += current\_process.burst\_time  completed.append(current\_process)  else:  # If no process is ready, advance time to the next process's arrival  if index < total\_processes:  current\_time = processes[index].arrival\_time    average\_waiting\_time = waiting\_time / total\_processes  print("\nAverage Waiting Time:", average\_waiting\_time)  if \_\_name\_\_ == "\_\_main\_\_":  # Example processes with PID, burst time, and arrival time  processes = [  Process(1, 6, 0),  Process(2, 8, 1),  Process(3, 7, 2),  Process(4, 3, 3)  ]  sjf\_scheduling(processes) |

OUTPUT:

