Vivekanand Education Society's
Institute of Technology
Approved by AICTE & Affiliated to University of Mumbai

Artificial Intelligence and Data Science Department.
OS / Even Sem 2021-22 / Experiment 6.

YASH SARANG.
47 / D6AD.
EXPERIMENT - 6.
CPU Scheduling.

Aim: Write a program to demonstrate any three CPU scheduling algorithms like FCFS, SJF (preemptive and non preemptive), Round Robin, priority(preemptive and non preemptive) .

Theory:

EXPERIMENT NO. 6

AIM: Write a program to demonstrate any 3 CPU scheduling algorithms like FCFS, SJF. (preemptive & nonpreemptive), Round Robin.

Theory: CPU Scheduling.

1) FCFS Scheduling.

First come first serve (FCFS) scheduling algorithm simply schedule the jobs according to their arrival time. The job which comes first in the ready queue will get the CPU first.

The lesser the arrival time of the job, the sooner the job will get the CPU.

FCFS may cause the problem of starvation if the burst time of the first process is longest among the jobs.

Average waiting time is higher as compared to other scheduling algorithms.

3) Round Robin

It is a starvation free algorithm as all the processes get a fair share of the CPU. A process with least burst time is executed for the fixed time quantum & if two process have the same burst time remaining, then the arrival time is considered & the one process arrived first is executed first.

Thus all the processes get a fair share of the CPU.

Disadvantage is - It has a fairly large waiting & response time.

The Gantt chart is complex & comparitively difficult to make.

2) SJF Scheduling

i) Non-preemptive.

Shortest Job first (SJF) is a scheduling algorithm that selects the waiting process with the smallest execution time to execute next.

It has a minimum average waiting time among all scheduling algorithms.

It's a greedy algorithm.

ii) Pre-emptive

SRTF - Shortest Remaining Time First.

In SRTF, the process with the smallest amount of time remaining until completion is selected to execute.

If every time period is same, the shortest process is selected & thus executed.

Snippets:

1. FCFS:

```
def WaitingTime(processes, n, bt, wt):
  wt[0] = 0
  for i in range(1, n ):
    wt[i] = bt[i - 1] + wt[i - 1]
def TurnAroundTime(processes, n, bt, wt, tat):
  for i in range(n):
    tat[i] = bt[i] + wt[i]
def AverageTime( processes, n, bt):
  wt = [0] * n
  tat = [0] * n
  total_wt = 0
  total_tat = 0
  WaitingTime(processes, n, bt, wt)
  TurnAroundTime(processes, n,
          bt, wt, tat)
  print( "Process  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 = [2, 4, 5]
  AverageTime(processes, n, burst_time)
```

Output:

```
Process  Burst time  Waiting time  Turn around time
 1              2          0               2
 2              4          2               6
 3              5          6               11
Average waiting time = 2.6666666666666665
Average turn around time = 6.333333333333333
```

---

2. SJF (Non-Pre emptive)

```
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 = 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))
# Driver code
if __name__ =="__main__":
    proc = [1, 2, 3]
    n = 3
    burst_time = [2, 4, 5]
    quantum = 2;
    findavgTime(proc, n, burst_time, quantum)
```

Output:

```
Processes    Burst Time     Waiting Time    Turn-Around Time
  1              2              0               2
  2              4              4               8
  3              5              6               11

Average waiting time = 3.33333
Average turn around time = 7.00000
```

3. SRTF (Pre-emptive)

```
def findWaitingTime(processes, n, wt):
  rt = [0] * n
  for i in range(n):
    rt[i] = processes[i][1]
  complete = 0
  t = 0
  minm = 999999999
  short = 0
  check = False
  while (complete != n):
    for j in range(n):
      if ((processes[j][2] <= t) and
        (rt[j] < minm) and rt[j] > 0):
        minm = rt[j]
        short = j
        check = True
    if (check == False):
      t += 1
      continue
    rt[short] -= 1
    minm = rt[short]
    if (minm == 0):
      minm = 999999999
    if (rt[short] == 0):
      complete += 1
      check = False
      fint = t + 1
      wt[short] = (fint - proc[short][1] -
                   proc[short][2])
      if (wt[short] < 0):
        wt[short] = 0
    t += 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("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(" ", 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)
# Driver code
if __name__ =="__main__":
    proc = [[1, 2, 1], [2, 4, 1],
        [3, 5, 1]]
    n = 3
    findavgTime(proc, n)
```

Output:

```
Processes Burst Time     Waiting Time    Turn-Around Time
  1              2               0               2
  2              4               2               6
  3              5               6               11

Average waiting time = 2.66667
Average turn around time =  6.333333333333333
```

---

4. 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 = 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))
# Driver code
if __name__ =="__main__":
  proc = [1, 2, 3]
  n = 3
  burst_time = [2, 4, 5]
  quantum = 2;
  findavgTime(proc, n, burst_time, quantum)
```

Output:

```
Processes Burst Time     Waiting Time Turn-Around Time
  1              2              0              2
  2              4              4              8
  3              5              6              11

Average waiting time = 3.33333
Average turn around time = 7.00000
```

Conclusion :

With the above example, we can see that the least waiting time (average) is for FCFS & SRTF (preemptive) & also the same for the turn-around time.