**Practical-8**

**AIM:-**

**Implementation of Process Scheduling Algorithm:**

**a. FCFS**

**b. Round Robing**

**c. SJF**

**d. Priority Scheduling**

|  |  |
| --- | --- |
| **Sr. No.** | **Code** |
| **8.1** | **FCFS**  **Code:-**  #include <iostream>  #include <vector>  using namespace std;  void FCFS(vector<int> arrivalTime, vector<int> burstTime)  {      int n = arrivalTime.size();      int completionTime[n], waitingTime[n], turnaroundTime[n];      // Calculate completion, waiting, and turnaround times      completionTime[0] = burstTime[0];      waitingTime[0] = 0;      turnaroundTime[0] = completionTime[0] - arrivalTime[0];      for (int i = 1; i < n; i++)      {          completionTime[i] = completionTime[i - 1] + burstTime[i];          waitingTime[i] = completionTime[i - 1] - arrivalTime[i];          turnaroundTime[i] = completionTime[i] - arrivalTime[i];      }      // Display results      cout << "Process\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time\n";      for (int i = 0; i < n; i++)      {          cout << i + 1 << "\t" << arrivalTime[i] << "\t\t" << burstTime[i] << "\t\t" << completionTime[i] << "\t\t"               << waitingTime[i] << "\t\t" << turnaroundTime[i] << "\n";      }  }  int main()  {      vector<int> arrivalTime = {0, 1, 2, 3};      vector<int> burstTime = {5, 3, 8, 6};      cout << "FCFS Scheduling:\n";      FCFS(arrivalTime, burstTime);      return 0;  }  **Output:-** |
| **8.2** | **Round Robin**  **Code:-**  #include <iostream>  #include <vector>  #include <queue>  using namespace std;  struct Process  {      int processID;      int burstTime;  };  void roundRobin(vector<int> arrivalTime, vector<int> burstTime, int quantum)  {      int n = arrivalTime.size();      queue<Process> readyQueue;      vector<int> remainingBurstTime(burstTime.begin(), burstTime.end());      vector<int> completionTime(n, 0); // Initialize completion time to 0 for all processes      int currentTime = 0;      int completed = 0;      int timeQuantum = quantum;      int waitingTime[n], turnaroundTime[n];      // Enqueue the first process      readyQueue.push({0, burstTime[0]});      // Simulate the Round Robin scheduling      while (completed < n)      {          Process currentProcess = readyQueue.front();          readyQueue.pop();          if (remainingBurstTime[currentProcess.processID] <= timeQuantum)          {              currentTime += remainingBurstTime[currentProcess.processID];              timeQuantum = remainingBurstTime[currentProcess.processID];              remainingBurstTime[currentProcess.processID] = 0;              completed++;              // Set completion time for this process              completionTime[currentProcess.processID] = currentTime;          }          else          {              currentTime += timeQuantum;              remainingBurstTime[currentProcess.processID] -= timeQuantum;          }          // Enqueue the processes that arrive in the meantime          for (int i = 0; i < n; i++)          {              if (arrivalTime[i] <= currentTime && remainingBurstTime[i] > 0)              {                  readyQueue.push({i, remainingBurstTime[i]});              }          }          // Enqueue the current process again if it's not completed          if (remainingBurstTime[currentProcess.processID] > 0)          {              readyQueue.push(currentProcess);          }          // Calculate waiting and turnaround times          waitingTime[currentProcess.processID] = currentTime - burstTime[currentProcess.processID];          turnaroundTime[currentProcess.processID] = waitingTime[currentProcess.processID] + burstTime[currentProcess.processID];      }      // Display results      cout << "Process\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time\n";      for (int i = 0; i < n; i++)      {          cout << i + 1 << "\t" << arrivalTime[i] << "\t\t" << burstTime[i] << "\t\t" << completionTime[i] << "\t\t"               << waitingTime[i] << "\t\t" << turnaroundTime[i] << "\n";      }  }  int main()  {      vector<int> arrivalTime = {0, 1, 2, 3};      vector<int> burstTime = {5, 3, 8, 6};      int quantum = 2;      cout << "Round Robin Scheduling:\n";      roundRobin(arrivalTime, burstTime, quantum);      return 0;  }  **Output:-** |
| **8.3** | **SJF**  **Coode:-**  #include <iostream>  #include <vector>  #include <algorithm>  using namespace std;  struct Process  {      int processID;      int arrivalTime;      int burstTime;  };  bool compareBurstTime(const Process &a, const Process &b)  {      return a.burstTime < b.burstTime;  }  void SJF(vector<int> arrivalTime, vector<int> burstTime)  {      int n = arrivalTime.size();      vector<Process> processes(n);      for (int i = 0; i < n; i++)      {          processes[i].processID = i;          processes[i].arrivalTime = arrivalTime[i];          processes[i].burstTime = burstTime[i];      }      sort(processes.begin(), processes.end(), compareBurstTime);      int completionTime[n], waitingTime[n], turnaroundTime[n];      int currentTime = processes[0].arrivalTime;      for (int i = 0; i < n; i++)      {          currentTime = max(currentTime, processes[i].arrivalTime);          completionTime[processes[i].processID] = currentTime + processes[i].burstTime;          waitingTime[processes[i].processID] = currentTime - processes[i].arrivalTime;          turnaroundTime[processes[i].processID] = waitingTime[processes[i].processID] + processes[i].burstTime;          currentTime = completionTime[processes[i].processID];      }      // Display results      cout << "Process\tArrival Time\tBurst Time\tCompletion Time\tWaiting Time\tTurnaround Time\n";      for (int i = 0; i < n; i++)      {          cout << processes[i].processID + 1 << "\t" << processes[i].arrivalTime << "\t\t" << processes[i].burstTime               << "\t\t" << completionTime[processes[i].processID] << "\t\t" << waitingTime[processes[i].processID]               << "\t\t" << turnaroundTime[processes[i].processID] << "\n";      }  }  int main()  {      vector<int> arrivalTime = {0, 1, 2, 3};      vector<int> burstTime = {5, 3, 8, 6};      cout << "SJF Scheduling:\n";      SJF(arrivalTime, burstTime);      return 0;  }  **Output:-** |
| **8.4** | **Prioriy Scheduling**  **Code:-**  #include <iostream>  #include <vector>  #include <algorithm>  using namespace std;  struct Process  {      int processID;      int arrivalTime;      int burstTime;      int priority;  };  bool comparePriority(const Process &a, const Process &b)  {      return a.priority < b.priority;  }  void priorityScheduling(vector<int> arrivalTime, vector<int> burstTime, vector<int> priority)  {      int n = arrivalTime.size();      vector<Process> processes(n);      for (int i = 0; i < n; i++)      {          processes[i].processID = i;          processes[i].arrivalTime = arrivalTime[i];          processes[i].burstTime = burstTime[i];          processes[i].priority = priority[i];      }      sort(processes.begin(), processes.end(), comparePriority);      int completionTime[n], waitingTime[n], turnaroundTime[n];      int currentTime = processes[0].arrivalTime;      for (int i = 0; i < n; i++)      {          currentTime = max(currentTime, processes[i].arrivalTime);          completionTime[processes[i].processID] = currentTime + processes[i].burstTime;          waitingTime[processes[i].processID] = currentTime - processes[i].arrivalTime;          turnaroundTime[processes[i].processID] = waitingTime[processes[i].processID] + processes[i].burstTime;          currentTime = completionTime[processes[i].processID];      }      // Display results      cout << "Process\tArrival Time\tBurst Time\tPriority\tCompletion Time\tWaiting Time\tTurnaround Time\n";      for (int i = 0; i < n; i++)      {          cout << processes[i].processID + 1 << "\t" << processes[i].arrivalTime << "\t\t" << processes[i].burstTime               << "\t\t" << processes[i].priority << "\t\t" << completionTime[processes[i].processID] << "\t\t"               << waitingTime[processes[i].processID] << "\t\t" << turnaroundTime[processes[i].processID] << "\n";      }  }  int main()  {      vector<int> arrivalTime = {0, 1, 2, 3};      vector<int> burstTime = {5, 3, 8, 6};      vector<int> priority = {2, 1, 3, 4};      cout << "Priority Scheduling:\n";      priorityScheduling(arrivalTime, burstTime, priority);      return 0;  }  **Output:-** |