1: #include <stdio.h>

2: #define MAX 100

3: // Structure to represent a process 5: typedef struct {

7: int id; // Process ID 9: int bt; // Burst Time

11: int at; // Arrival Time 13: int wt; // Waiting Time

15: int tat; // Turnaround Time 17: } Process;

21: // Function to calculate waiting time for SJF

23: void calculateWaitingTimeSJF(Process proc[], int n) { 25: int total\_wt = 0;

27: proc[0].wt = 0; // Waiting time for the first process is 0 31: for (int i = 1; i < n; i++) {

33: proc[i].wt = 0; // Initialize waiting time 35: for (int j = 0; j < i; j++) {

37: proc[i].wt += proc[j].bt; // Summing burst times for previous processes 38:

39: }

41: total\_wt += proc[i].wt; // Accumulating total waiting time 43: }

45: }

49: // Function to calculate turnaround time

51: void calculateTurnaroundTime(Process proc[], int n) { 53: for (int i = 0; i < n; i++) {

55: proc[i].tat = proc[i].bt + proc[i].wt; // TAT = BT + WT 57: }

59: }

63: // Function to sort processes based on Burst Time for SJF 65: void sortProcessesSJF(Process proc[], int n) {

67: for (int i = 0; i < n - 1; i++) { 69: for (int j = i + 1; j < n; j++) { 71: if (proc[i].bt > proc[j].bt) {

73: Process temp = proc[i];

75: proc[i] = proc[j];

77: proc[j] = temp;

79: }

81: }

83: }

85: }

89: // Function for Round Robin Scheduling

91: void calculateWaitingTimeRR(Process proc[], int n, int quantum) { 93: int remaining\_bt[MAX];

95: for (int i = 0; i < n; i++) {

97: remaining\_bt[i] = proc[i].bt; // Copy burst times 99: }

103: int t = 0; // Current time 105: while (1) {

107: int done = 1; // Check if all processes are done 109: for (int i = 0; i < n; i++) {

111: if (remaining\_bt[i] > 0) {

113: done = 0; // If a process is remaining

115: if (remaining\_bt[i] > quantum) {

117: t += quantum; // Increment time by quantum

119: remaining\_bt[i] -= quantum; // Decrease remaining burst time 121: } else {:

123: t = t + remaining\_bt[i]; // Increment time by remaining time

125: proc[i].wt = t - proc[i].bt; // Waiting time

127: remaining\_bt[i] = 0; // Mark process as done

129: }

131: }

133: }

135: if (done == 1) break; // All processes are done 137: }

139: }

143: // Function to display the processes and their details in a table format 145: void displayProcesses(Process proc[], int n) {

147: printf("\n \n"); 149: printf("| P | BT | AT | WT | TAT |\n"); 151: printf(" \n"); 153: for (int i = 0; i < n; i++) {

155: printf("| %-2d | %-3d | %-3d | %-3d | %-4d |\n",

157: proc[i].id, proc[i].bt, proc[i].at, proc[i].wt, proc[i].tat); 159: }

161: printf(" \n");

163: }

167: int main() {

169: Process proc[MAX]; 171: int n, choice;

175: // Taking number of processes as input 177: printf("Enter the number of processes: "); 179: scanf("%d", &n);

183: // Taking Burst Time and Arrival Time input for each process 185: for (int i = 0; i < n; i++) {

187: printf("Enter Burst Time and Arrival Time for P%d: ", i + 1); 189: scanf("%d %d", &proc[i].bt, &proc[i].at);

191: proc[i].id = i + 1; // Process ID assignment 193: proc[i].wt = 0; // Initialize Waiting Time 194:

195: }

199: while (1) {

201: printf("\nChoose the scheduling algorithm:\n"); 203: printf("1. Shortest Job First (Preemptive)\n"); 205: printf("2. Round Robin\n");

207: printf("3. Exit\n");

209: printf("Enter your choice: "); 211: scanf("%d", &choice);

215: switch (choice) {

217: case 1: // Shortest Job First

219: sortProcessesSJF(proc, n);

221: calculateWaitingTimeSJF(proc, n);

223: calculateTurnaroundTime(proc, n);

225: displayProcesses(proc, n);

227: break;

231: case 2: // Round Robin

233: {

235: int quantum;

237: printf("Enter Time Quantum for Round Robin: "); 239: scanf("%d", &quantum);

241: calculateWaitingTimeRR(proc, n, quantum);

243: calculateTurnaroundTime(proc, n);

245: displayProcesses(proc, n);

247: }

248:

249: break;

253: case 3: // Exit

255: return 0;

259: default:

261: printf("Invalid choice! Please try again.\n"); 263: }

265: }

267: return 0;

269: }

