Assignment -05

**KeshavPorwal23I4143**

**CPU Scheduling Algorithms**

This document outlines the implementation of three common CPU scheduling algorithms: First Come First Serve (FCFS), Shortest Job First (SJF), and Round Robin Scheduling. The C programs for each algorithm are provided along with sample outputs.

**1. First Come First Serve (FCFS) Scheduling**

**Code:**

#include <stdio.h>

void findWaitingTime(int processes[], int n, int bt[], int wt[]) {

wt[0] = 0;

for (int i = 1; i < n; i++) {

wt[i] = bt[i - 1] + wt[i - 1];

}

}

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {

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

tat[i] = bt[i] + wt[i];

}

}

**KeshavPorwal23I4143**

void findAvgTime(int processes[], int n, int bt[]) {

int wt[n], tat[n], total\_wt = 0, total\_tat = 0;

findWaitingTime(processes, n, bt, wt);

findTurnAroundTime(processes, n, bt, wt, tat);

printf("Processes\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

total\_wt += wt[i];

total\_tat += tat[i];

printf("%d\t\t%d\t\t%d\t\t%d\n", processes[i], bt[i], wt[i], tat[i]);

}

printf("\nAverage waiting time = %.2f", (float)total\_wt / n);

printf("\nAverage turnaround time = %.2f\n", (float)total\_tat / n);

}

int main() {

int processes[] = {1, 2, 3};

int n = sizeof processes / sizeof processes[0];

int burst\_time[] = {24, 3, 3};

findAvgTime(processes, n, burst\_time);

return 0;

}

**Sample Output:**

**KeshavPorwal23I4143**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**2. Shortest Job First (SJF) Scheduling**

**Code:**

#include <stdio.h>

#include <stdbool.h>

void findWaitingTime(int processes[], int n, int bt[], int wt[]) {

int completed = 0, t = 0, min\_bt = 0;

bool is\_completed[n];

for (int i = 0; i < n; i++) is\_completed[i] = false;

while (completed != n) {

min\_bt = 1000; **KeshavPorwal23I4143**

int shortest = -1;

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

if (!is\_completed[i] && bt[i] < min\_bt) {

min\_bt = bt[i];

shortest = i;

}

}

t += bt[shortest];

wt[shortest] = t - bt[shortest];

is\_completed[shortest] = true;

completed++;

}

} **KeshavPorwal23I4143**

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {

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

tat[i] = bt[i] + wt[i];

}

}

void findAvgTime(int processes[], int n, int bt[]) {

int wt[n], tat[n];

findWaitingTime(processes, n, bt, wt);

findTurnAroundTime(processes, n, bt, wt, tat);

printf("Processes\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

printf("%d\t\t%d\t\t%d\t\t%d\n", processes[i], bt[i], wt[i], tat[i]);

}

} **KeshavPorwal23I4143**

int main() {

int processes[] = {1, 2, 3};

int n = sizeof processes / sizeof processes[0];

int burst\_time[] = {6, 8, 7};

findAvgTime(processes, n, burst\_time);

return 0;

}

A screen shot of a computer

AI-generated content may be incorrect.

**3. Round Robin Scheduling**

**Code:**

#include <stdio.h>

void findWaitingTime(int processes[], int n, int bt[], int wt[], int quantum) {

int rem\_bt[n];

for (int i = 0; i < n; i++) rem\_bt[i] = bt[i];

int t = 0;

while (1) {

int done = 1; **KeshavPorwal23I4143**

**KeshavPorwal23I4143**

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

if (rem\_bt[i] > 0) {

done = 0;

if (rem\_bt[i] > quantum) {

t += quantum;

rem\_bt[i] -= quantum;

} else {

t += rem\_bt[i];

wt[i] = t - bt[i];

rem\_bt[i] = 0;

}

}

}

if (done) break;

}

}

void findTurnAroundTime(int processes[], int n, int bt[], int wt[], int tat[]) {

for (int i = 0; i < n; i++) tat[i] = bt[i] + wt[i];

}

void findAvgTime(int processes[], int n, int bt[], int quantum) {

int wt[n], tat[n];

findWaitingTime(processes, n, bt, wt, quantum);

findTurnAroundTime(processes, n, bt, wt, tat);

printf("Processes\tBurst Time\tWaiting Time\tTurnaround Time\n");

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

printf("%d\t\t%d\t\t%d\t\t%d\n", processes[i], bt[i], wt[i], tat[i]);

}

}

int main() { **KeshavPorwal23I4143**

int processes[] = {1, 2, 3};

int n = sizeof processes / sizeof processes[0];

int burst\_time[] = {10, 5, 8};

int quantum = 2;

findAvgTime(processes, n, burst\_time, quantum);

return 0;

}

**Sample Output:**

A screen shot of a computer

AI-generated content may be incorrect.

**KeshavPorwal23I4143**