# Bansilal Ramnath Agarwal Charitable Trust’s

Vishwakarma Institute of Technology, Pune-37

*(Anautonomous Institute of Savitribai Phule Pune University)*



**Department of Multidisciplinary Engineering**

|  |  |
| --- | --- |
| **Division** | **CS-A** |
| **Batch** | **B1** |
| **Roll no.** | **90** |
| **Name** | **Aditya Shrinivas Kurapati** |

1. **FCFS :-**

#include <stdio.h>

struct ps {

char p\_name[20];

int at, bt, ct, tat, wt, flag;

};

struct queue {

int front, rear;

struct ps a[100];

};

int at[] = {3, 7, 2, 1, 0};

int bt[] = {5, 3, 1, 2, 3};

int n = 5;

int main() {

struct ps p[n];

float avg\_tat, avg\_wt;

int done = 0;

int time = 0;

int ps = 0;

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

sprintf(p[i].p\_name, "P%d", i + 1);

p[i].at = at[i];

p[i].bt = bt[i];

p[i].flag = 0;

}

while (!done) {

done = 1;

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

if (p[i].flag == 0) {

done = 0;

if (time >= p[i].at) {

p[i].flag = 1;

p[i].ct = time + p[i].bt;

time = p[i].ct;

} else {

time = p[i].at;

p[i].flag = 1;

p[i].ct = time + p[i].bt;

time = p[i].ct;

}

}

}

}

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

p[i].tat = p[i].ct - p[i].at;

p[i].wt = p[i].tat - p[i].bt;

}

// Calculate average turnaround time and average waiting time

float total\_tat = 0, total\_wt = 0;

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

total\_tat += p[i].tat;

total\_wt += p[i].wt;

}

avg\_tat = total\_tat / n;

avg\_wt = total\_wt / n;

printf("PS\tAT\tBT\tCT\tTAT\tWT\n");

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

printf("%s\t%d\t%d\t%d\t%d\t%d\n", p[i].p\_name, p[i].at, p[i].bt, p[i].ct,

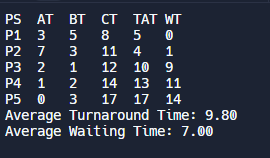
p[i].tat, p[i].wt);

}

printf("Average Turnaround Time: %.2f\n", avg\_tat);

printf("Average Waiting Time: %.2f\n", avg\_wt);

return 0;

}  


1. **SJF Non-Preemptive :-**

#include <stdio.h>

struct ps {

char p\_name[20];

int at, bt, ct, tat, wt, flag;

};

struct queue {

int front, rear;

struct ps a[100];

};

int at[] = {3, 7, 2, 1, 0};

int bt[] = {5, 3, 1, 2, 3};

int n = 5;

void swap(struct ps \*xp, struct ps \*yp) {

struct ps temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void sort(struct ps p[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (p[j].bt > p[j + 1].bt) {

swap(&p[j], &p[j + 1]);

}

}

}

}

int main() {

struct ps p[n];

float avg\_tat, avg\_wt;

int time = 0;

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

sprintf(p[i].p\_name, "P%d", i + 1);

p[i].at = at[i];

p[i].bt = bt[i];

p[i].flag = 0;

}

sort(p, n);

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

if (time < p[i].at) { // Process arrives after current time

time = p[i].at;

}

p[i].ct = time + p[i].bt;

time = p[i].ct;

}

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

p[i].tat = p[i].ct - p[i].at;

p[i].wt = p[i].tat - p[i].bt;

if (p[i].tat < 0) {

p[i].tat = 0; // Turnaround time cannot be negative

}

if (p[i].wt < 0) {

p[i].wt = 0; // Waiting time cannot be negative

}

}

// Calculate average turnaround time and average waiting time

float total\_tat = 0, total\_wt = 0;

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

total\_tat += p[i].tat;

total\_wt += p[i].wt;

}

avg\_tat = total\_tat / n;

avg\_wt = total\_wt / n;

printf("PS\tAT\tBT\tCT\tTAT\tWT\n");

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

printf("%s\t%d\t%d\t%d\t%d\t%d\n", p[i].p\_name, p[i].at, p[i].bt, p[i].ct,

p[i].tat, p[i].wt);

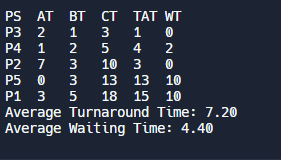
}

printf("Average Turnaround Time: %.2f\n", avg\_tat);

printf("Average Waiting Time: %.2f\n", avg\_wt);

return 0;

}



1. **SJF Preemptive :-**

#include <limits.h>

#include <stdio.h>

#define MAX 5

struct ps {

int name;

int arrival\_time;

int burst\_time;

int completion\_time;

int turnaround\_time;

int waiting\_time;

};

int main() {

struct ps ps[MAX];

int n = 5; // Number of processes

int at[] = {3, 7, 2, 1, 0};

int bt[] = {5, 3, 1, 2, 3};

int i, total\_burst\_time = 0;

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

ps[i].name = i + 1;

ps[i].arrival\_time = at[i];

ps[i].burst\_time = bt[i];

total\_burst\_time += ps[i].burst\_time;

ps[i].completion\_time = -1;

ps[i].turnaround\_time = -1;

ps[i].waiting\_time = -1;

}

int current\_time = 0;

int completed\_ps = 0;

while (completed\_ps < n) {

int smallest\_burst\_index = -1;

int smallest\_burst = INT\_MAX;

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

if (ps[i].arrival\_time <= current\_time && ps[i].completion\_time == -1 &&

ps[i].burst\_time < smallest\_burst) {

smallest\_burst = ps[i].burst\_time;

smallest\_burst\_index = i;

}

}

if (smallest\_burst\_index != -1) {

current\_time++;

ps[smallest\_burst\_index].burst\_time--;

if (ps[smallest\_burst\_index].burst\_time == 0) {

ps[smallest\_burst\_index].completion\_time = current\_time;

completed\_ps++;

}

} else {

current\_time++;

}

}

// Calculating turnaround time and waiting time

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

ps[i].turnaround\_time = ps[i].completion\_time - ps[i].arrival\_time;

ps[i].waiting\_time = ps[i].turnaround\_time - bt[i];

}

// Displaying process information

printf("\nProcess\tAT\tBT\tCT\tTAT\tWT\n");

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

printf("%d\t%d\t%d\t%d\t%d\t%d\n", ps[i].name, ps[i].arrival\_time, bt[i],

ps[i].completion\_time, ps[i].turnaround\_time, ps[i].waiting\_time);

}

// Calculating averages

float avg\_turnaround\_time = 0, avg\_waiting\_time = 0;

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

avg\_turnaround\_time += ps[i].turnaround\_time;

avg\_waiting\_time += ps[i].waiting\_time;

}

avg\_turnaround\_time /= n;

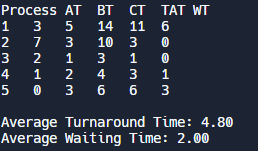
avg\_waiting\_time /= n;

printf("\nAverage Turnaround Time: %.2f\n", avg\_turnaround\_time);

printf("Average Waiting Time: %.2f\n", avg\_waiting\_time);

return 0;

}



1. **Priorty Preemtive And Non-Preemptive:-**

#include <limits.h>

#include <stdio.h>

#define MAX 5

struct Process {

int name;

int arrival\_time;

int burst\_time;

int priority;

int completion\_time;

int turnaround\_time;

int waiting\_time;

};

void swap(struct Process \*a, struct Process \*b) {

struct Process temp = \*a;

\*a = \*b;

\*b = temp;

}

void sort\_by\_arrival\_time(struct Process processes[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (processes[j].arrival\_time > processes[j + 1].arrival\_time) {

swap(&processes[j], &processes[j + 1]);

}

}

}

}

void sort\_by\_priority(struct Process processes[], int n) {

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (processes[j].priority > processes[j + 1].priority) {

swap(&processes[j], &processes[j + 1]);

}

}

}

}

void calculate\_completion\_time(struct Process processes[], int n) {

int current\_time = 0;

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

if (current\_time < processes[i].arrival\_time) {

current\_time = processes[i].arrival\_time;

}

current\_time += processes[i].burst\_time;

processes[i].completion\_time = current\_time;

}

}

void calculate\_turnaround\_time(struct Process processes[], int n) {

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

processes[i].turnaround\_time =

processes[i].completion\_time - processes[i].arrival\_time;

}

}

void calculate\_waiting\_time(struct Process processes[], int n) {

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

processes[i].waiting\_time =

processes[i].turnaround\_time - processes[i].burst\_time;

}

}

void display\_process\_info(struct Process processes[], int n) {

printf("\nProcess\tAT\tBT\tPri\tCT\tTAT\tWT\n");

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

printf("%d\t%d\t%d\t%d\t%d\t%d\t%d\n", processes[i].name,

processes[i].arrival\_time, processes[i].burst\_time,

processes[i].priority, processes[i].completion\_time,

processes[i].turnaround\_time, processes[i].waiting\_time);

}

}

int main() {

struct Process processes[MAX];

int n = MAX;

int at[] = {3, 7, 2, 1, 0};

int bt[] = {5, 3, 1, 2, 3};

int priority[] = {2, 1, 3, 4, 5};

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

processes[i].name = i + 1;

processes[i].arrival\_time = at[i];

processes[i].burst\_time = bt[i];

processes[i].priority = priority[i];

processes[i].completion\_time = -1;

processes[i].turnaround\_time = -1;

processes[i].waiting\_time = -1;

}

// Sort processes by arrival time

sort\_by\_arrival\_time(processes, n);

// Implement Priority Scheduling (Preemptive)

int current\_time = 0;

int completed\_processes = 0;

while (completed\_processes < n) {

int highest\_priority\_index = -1;

int highest\_priority = INT\_MAX;

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

if (processes[i].arrival\_time <= current\_time &&

processes[i].burst\_time > 0) {

if (processes[i].priority < highest\_priority) {

highest\_priority = processes[i].priority;

highest\_priority\_index = i;

}

}

}

if (highest\_priority\_index != -1) {

processes[highest\_priority\_index].burst\_time--;

current\_time++;

if (processes[highest\_priority\_index].burst\_time == 0) {

processes[highest\_priority\_index].completion\_time = current\_time;

completed\_processes++;

}

} else {

current\_time++;

}

}

// Calculate turnaround time and waiting time for preemptive priority

// scheduling

calculate\_turnaround\_time(processes, n);

calculate\_waiting\_time(processes, n);

// Display process information for preemptive priority scheduling

printf("\nPreemptive Priority Scheduling (PPS):\n");

display\_process\_info(processes, n);

// Sort processes by priority for non-preemptive priority scheduling

sort\_by\_priority(processes, n);

// Calculate completion time for non-preemptive priority scheduling

calculate\_completion\_time(processes, n);

// Calculate turnaround time and waiting time for non-preemptive priority

// scheduling

calculate\_turnaround\_time(processes, n);

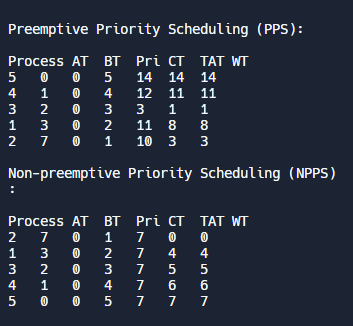
calculate\_waiting\_time(processes, n);

// Display process information for non-preemptive priority scheduling

printf("\nNon-preemptive Priority Scheduling (NPPS):\n");

display\_process\_info(processes, n);

return 0;

}  
  


1. **Round Robin :-**

#include <stdio.h>

struct ps {

char p\_name[20];

int at, bt, ct, tat, wt, flag, rt;

};

struct queue {

int front, rear;

struct ps a[100];

};

int at[] = {0, 1, 2, 3, 4};

int bt[] = {5, 3, 1, 2, 3};

int n = 5;

int main() {

float avg\_tat, avg\_wt;

int TQ;

struct ps p[n];

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

sprintf(p[i].p\_name, "P%d", i + 1);

p[i].at = at[i];

p[i].bt = bt[i];

p[i].flag = 0;

p[i].rt = bt[i];

}

printf("Enter the time quantum : ");

scanf("%d", &TQ);

int time = 0;

int ps = 0;

struct queue q;

q.front = q.rear = -1;

while (1) {

int done = 1; // Flag to check if all processes are done

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

if (p[i].flag == 0) {

done = 0; // At least one process is not done

if (p[i].rt > 0) {

if (p[i].rt <= TQ) {

time += p[i].rt;

p[i].rt = 0;

p[i].ct = time;

p[i].tat = p[i].ct - p[i].at;

p[i].wt = p[i].tat - p[i].bt;

p[i].flag = 1;

} else {

time += TQ;

p[i].rt -= TQ;

}

}

if (p[i].rt == 0 && q.rear != -1) {

struct ps temp = q.a[q.front];

q.front++;

if (q.front > q.rear) {

q.front = q.rear = -1;

}

q.a[q.rear + 1] = temp;

q.rear++;

}

}

}

if (done)

break; // All processes are done

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

if (p[i].flag == 0 && p[i].at <= time) {

if (q.rear == -1) {

q.front = q.rear = 0;

q.a[q.rear] = p[i];

} else {

q.rear++;

q.a[q.rear] = p[i];

}

ps++;

}

}

}

// Calculate average turnaround time and average waiting time

float total\_tat = 0, total\_wt = 0;

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

total\_tat += p[i].tat;

total\_wt += p[i].wt;

}

avg\_tat = total\_tat / n;

avg\_wt = total\_wt / n;

printf("PS\tAT\tBT\tCT\tTAT\tWT\tRT\n");

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

printf("%s\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].p\_name, p[i].at, p[i].bt,

p[i].ct, p[i].tat, p[i].wt, p[i].bt);

}

printf("Average Turnaround Time: %.2f\n", avg\_tat);

printf("Average Waiting Time: %.2f\n", avg\_wt);

return 0;

}

