**OS Project**

**An Improved Scheduling Algorithm Application**

**Team Members:**

Aishwarya-22BAI1384

Lakshmi-22BAI1438

Chandresh-22BAI1033

**Problem Statement:**

Implementing an improved round robin algorithm in a streamlined scheduling environment to run a hypothetical system in work or leisure mode leading to better performance – lesser average waiting time and context switches – due to lesser CPU load.

**Software Architecture:**

The user is first asked to enter the number of processes, and their arrival and burst time. They’re later asked to input the mode they’d like to work in and for fair comparison to choose the scheduling algorithm to be used. They each are made into function. This function is imported to main function which has the menu.

Additionally, the 5th algorithm acts as an improved round robin algorithm. Instead of taking a static type quantum, a dynamic time quantum is calculated according to the burst of each process.

There are modes to have precise processes running only for those modes, i.e., for work mode only processes related to work will be executed. At the beginning of the program, the mode of working is asked and then the processes burst and arrival time is taken as input. Then the menu is shown between all the different algorithms to show the difference and benefits of each algorithm.

**Algorithms:**

**First-Come, First-Served:**

FCFS is a scheduling algorithm used in operating systems and task management that executes tasks in the order they arrive, with the first task to arrive being the first one to be processed. It lacks flexibility and can lead to poor turnaround times for long-running tasks

**Shortest Job First**:

SJF is a CPU scheduling algorithm that prioritizes tasks based on their burst time, executing the shortest job first to minimize waiting times. It can lead to efficient resource utilization and shorter response times for smaller tasks. However, predicting accurate burst times can be challenging in practice.

**Shortest Time-to-Finish**:

STRF continuously monitors the remaining time needed for each task and schedules the one with the least time, providing excellent responsiveness and efficient task execution in dynamic workloads. However, it may not be suitable for all systems, as it can be prone to frequent context switches, which could incur overhead in certain scenarios, like batch processing. Careful consideration of task characteristics is essential when choosing this scheduling algorithm.

**Round Robin:**

Round Robin (RR) is a widely used CPU scheduling algorithm in operating systems. It allocates a fixed time quantum to each task in a cyclic queue. When a task's time quantum expires, it's moved to the back of the queue, allowing the next task in line to execute.

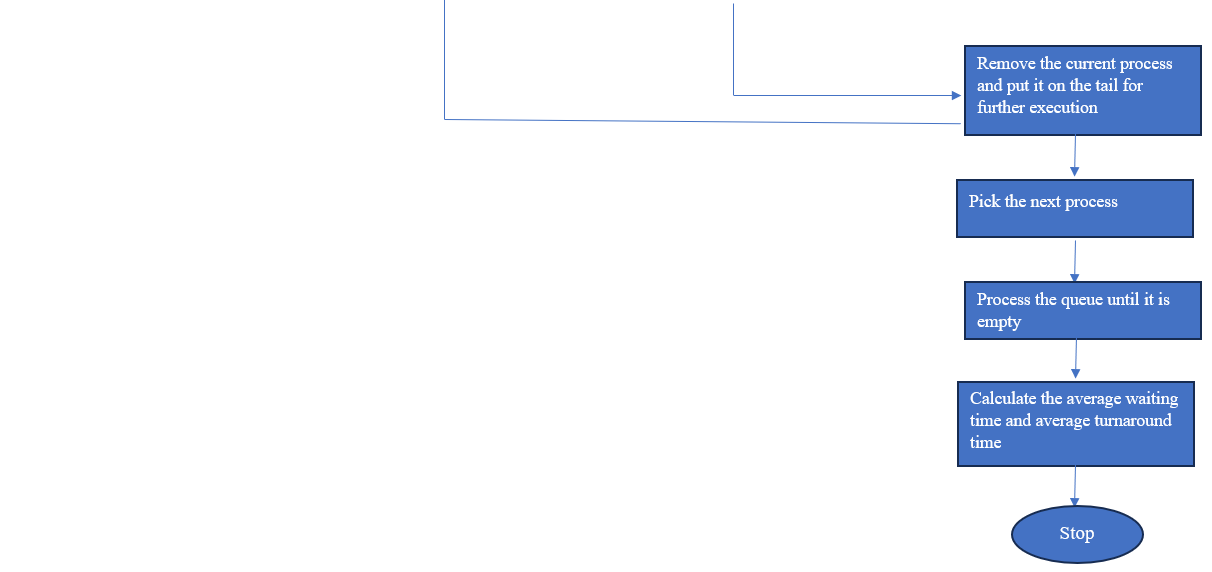
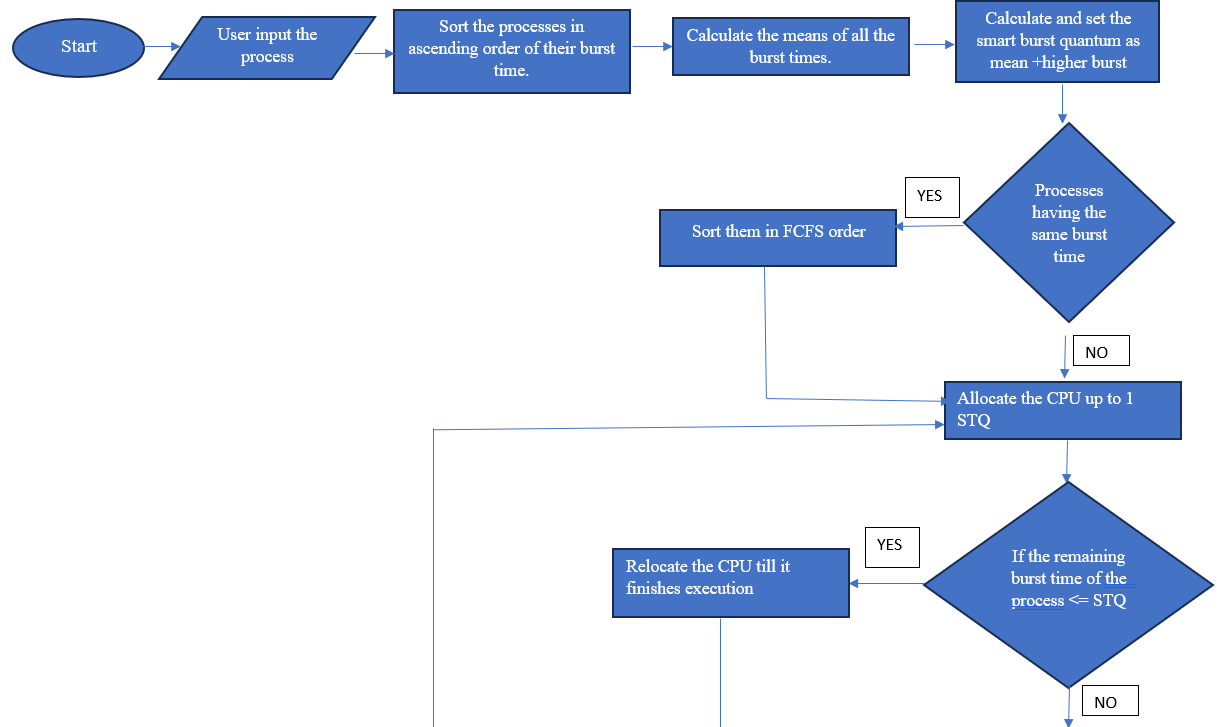
RR provides fairness, as each task gets a chance to run regularly. This prevents one task from monopolizing the CPU. However, RR may introduce overhead due to frequent context switches, particularly when many tasks have short execution times.

The choice of time quantum is a critical factor. A short quantum improves responsiveness but increases context switching, while a long quantum may lead to less responsive multitasking.

Round Robin is commonly used in time-sharing and interactive systems where fairness and responsiveness are crucial, such as in modern operating systems for multitasking desktops and servers. Properly configuring the time quantum is vital to strike a balance between fair allocation and efficient task switching.

**Improved round robin Algorithm:**

The scheduling algorithm used in this project consists of an improved round robin algorithm. Instead of using a static time quantum for the entirety of the processes, a dynamic time quantum is calculated according to the arrival time and burst time of several processes. The value of time quanta is calculated by a formula after the time quanta is specified by the user ensuring that it changes according to the arrival time and burst time. Instead of having too high of a time quanta or too low of a time quanta, this improved algorithm ensures that the time quanta are specific to the different processes and its own arrival and burst times.



**Working:**

**Case 1:** Zero Arrival Time

|  |  |  |
| --- | --- | --- |
| Processes | Arrival Time | Burst Time |
| P1 | 0 | 5 |
| P2 | 0 | 12 |
| P3 | 0 | 20 |
| P4 | 0 | 26 |
| P5 | 0 | 34 |

Gantt Representation for Round Robin

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P5 | P2 | P3 | P4 | P5 | P4 | P5 | P5 |

0 5 15 25 35 45 47 57 67 77 83 93 97 Time(ms)

Gantt Representation for Improved Round Robin

Average burst time = (5 + 12 + 20 + 26 + 34) / 5 = 19

Smart time quantum = (19 + 34) / 2 = 26

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 |  |  |  |  |

0 5 Time(ms)

Average burst time = (12 + 20 + 26 + 34) / 4 = 23

Smart time quantum = (23 + 34) / 2 = 26

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P2 |  |  |  |

0 5 17 Time(ms)

Average burst time = (20 + 26 + 34) / 3 = 26

Smart time quantum = (26 + 34) / 2 = 30

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P2 | P3 |  |  |

0 5 17 37 Time(ms)

Average burst time = (26 + 34) / 2 = 30

Smart time quantum = (30 + 34) / 2 = 32

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 |  |

0 5 17 37 63 Time(ms)

Average burst time = 34

Smart time quantum = (34 + 34) / 2 = 34

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P5 |

0 5 17 37 63 97 Time(ms)

|  |  |  |
| --- | --- | --- |
| Algorithm | Average Waiting Time(ms) | No. of context switches |
| Round robin | 38.5 | 12 |
| Improved round robin | 24.5 | 5 |

**Case 2:** Non Zero Arrival Time

|  |  |  |
| --- | --- | --- |
| Processes | Arrival Time | Burst Time |
| P1 | 0 | 7 |
| P2 | 4 | 18 |
| P3 | 10 | 27 |
| P4 | 15 | 30 |
| P5 | 17 | 36 |

Gantt Representation of Round Robin

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P5 | P2 | P3 | P4 | P5 | P3 | P4 | P5 | P5 |

0 7 17 27 37 47 55 65 75 85 92 102 112 118 Time(ms)

Gantt Representation of Improved Round Robin

Average burst time = 7

Smart time quantum = 7

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| P1 |  |  |  |  |  |

0 7 Time(ms)

Average burst time = 18

Smart time quantum = 18

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P2 |  |  |  |

0 7 25 Time(ms)

Average burst time = (27 + 30 + 36) / 3 = 31

Smart time quantum = (31 + 36) = 33

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P2 | P3 |  |  |

0 7 25 52 Time(ms)

Average burst time = (30 + 36) / 2 = 33

Smart time quantum = (33 + 36) / 2 = 34

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 |  |

0 7 25 52 82 Time(ms)

Average burst time = 36

Smart time quantum = 36

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| P1 | P2 | P3 | P4 | P5 |

0 7 25 52 82 118 Time(ms)

|  |  |  |
| --- | --- | --- |
| Algorithm | Average Waiting Time(ms) | No. of context switches |
| Round robin | 51.2 | 12 |
| Improved round robin | 33.2 | 5 |

**Code:**

#include <stdio.h>

#include <math.h>

#include <string.h>

#include<time.h>

int wt[100], bt[100], at[100], tat[100], n, p[100];//multiple lists containing processes

float awt[5], atat[5];

void input() {

printf("Enter Number of processes:");//taking input of the processes

scanf("%d", &n);

int i;

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

p[i] = i + 1;

}

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

printf("Enter Burst Time of process %d:", i + 1);//taking input of burst time

scanf("%d", &bt[i]);

printf("Enter Arrival Time of process %d:", i + 1);//taking input of arrival time

scanf("%d", &at[i]);

}

for (i = 0; i < 5; i++) {

awt[i] = 0.0;

atat[i] = 0.0;

}

}

void changeArrival() {// function to change the time quantum based on processes

int a = at[0];

int i;

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

if (at[i] < a)

a = at[i];

}

if (a != 0) {

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

at[i] = at[i] - a;

}

}

void fcfs() {//fcfs scheduling algorithm

wt[0] = 0;

atat[0] = tat[0] = bt[0];

int btt = bt[0];

int i;

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

wt[i] = btt - at[i];

btt += bt[i];

awt[0] += wt[i];

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

atat[0] += tat[i];

}

atat[0] /= n;

awt[0] /= n;

printf("SR.\tA.T.\tB.T.\tW.T.\tT.A.T.\n");

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

printf("%3d\t%3d\t%3d\t%3d\t%4d\n", i + 1, at[i], bt[i], wt[i], tat[i]);

}

}

void innovative(int mode) {//new algorithm based on round robin and dynamic time quanta

int bt1[n], i, j, temp, tq;

int b[n];

float twt, ttat;

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

bt1[i] = bt[i];

}

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

b[i] = bt[i];

}

int num = n;

int time = 0;

int max;

int sum, t, a, ap;

ap = 0;

while (num > 0) {

a = 0;

max = 0;

sum = 0;

t = 0;

// Sorting in ascending order

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

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

if (b[i] < b[j] && at[i] <= time) {

temp = p[j];

p[j] = p[i];

p[i] = temp;

temp = at[j];

at[j] = at[i];

at[i] = temp;

temp = b[j];

b[j] = b[i];

b[i] = temp;

temp = bt1[j];

bt1[j] = bt1[i];

bt1[i] = temp;

}

}

}

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

if (at[i] <= time && b[i] != 0) {

a++;

if (b[i] > max) {

max = b[i];

}

sum += b[i];

}

}

if (a != ap) {

tq = sqrt((sum / a) \* max);

ap = a;

}

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

if (at[i] <= time && b[i] != 0) {

if (b[i] < tq) {

t += b[i];

b[i] = 0;

} else {

t += tq;

b[i] -= tq;

}

if (b[i] == 0) {

wt[i] = (time + t) - bt1[i];

tat[i] = time + t;

num--;

}

}

}

time += t;

}

printf("Processes\tWaitingtime\tTurnAroundTime\n");

for (j = 1; j <= n; j++) {

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

if (j == p[i] && mode==1 ) {

printf("process of work mode %d\t%d\t\t%d\n", p[i], wt[i], tat[i]);}

else if(j==p[i] && mode==2)

{

printf("process of leisure mode %d\t%d\t\t%d\n",p[i], wt[i], tat[i]);

}

}

}

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

twt += wt[i];

}

awt[4] = twt / n;

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

ttat += tat[i];

}

atat[4] = ttat / n;

}

void rr() {//round robin process

int i, total = 0, x, counter = 0, time\_quantum;

int wait\_time = 0, turnaround\_time = 0, temp[100];

x = n;

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

temp[i] = bt[i];

}

printf("\nEnter Time Quantum:\t");

scanf("%d", &time\_quantum);

printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n");

for (total = 0, i = 0; x != 0;) {

if (temp[i] <= time\_quantum && temp[i] > 0) {

total = total + temp[i];

temp[i] = 0;

counter = 1;

} else if (temp[i] > 0) {

temp[i] = temp[i] - time\_quantum;

total = total + time\_quantum;

}

if (temp[i] == 0 && counter == 1) {

x--;

printf("Process[%d]\t\t%d\t\t %d\t\t\t %d\n", i + 1, bt[i], total - at[i], total - at[i] - bt[i]);

wait\_time = wait\_time + total - at[i] - bt[i];

turnaround\_time = turnaround\_time + total - at[i];

counter = 0;

}

if (i == n - 1) {

i = 0;

} else if (at[i + 1] <= total) {

i++;

} else {

i = 0;

}

}

awt[2] = wait\_time \* 1.0 / n;

atat[2] = turnaround\_time \* 1.0 / n;

}

void srtf() {//shortest job first algorithm

int i, x[10], b[10], count = 0, time, smallest;

double avg = 0, tt = 0, end;

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

x[i] = bt[i];

}

bt[9] = 9999;

for (time = 0; count != n; time++) {

smallest = 9;

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

if (at[i] <= time && bt[i] < b[smallest] && bt[i] > 0) {

smallest = i;

}

}

bt[smallest]--;

if (bt[smallest] == 0) {

count++;

end = time + 1;

avg = avg + end - at[smallest] - x[smallest];

tt = tt + end - at[smallest];

}

}

awt[3] = avg / n;

atat[3] = tt / n;

}

void display(int c) {//printing the process and its burst and arrival time and its run time

int i;

printf("Average Waiting Time: %f\nAverage Turn Around Time: %f", awt[c - 1], atat[c - 1]);

}

void sjf() {

float wavg = 0, tavg = 0, tsum = 0, wsum = 0;

int i, j, temp, sum = 0, ta = 0;

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

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

if (at[i] < at[j]) {

temp = p[j];

p[j] = p[i];

p[i] = temp;

temp = at[j];

at[j] = at[i];

at[i] = temp;

temp = bt[j];

bt[j] = bt[i];

bt[i] = temp;

}

}

}

int btime = 0, min, k = 1;

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

btime = btime + bt[j];

min = bt[k];

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

if (btime >= at[i] && bt[i] < min) {

temp = p[k];

p[k] = p[i];

p[i] = temp;

temp = at[k];

at[k] = at[i];

at[i] = temp;

temp = bt[k];

bt[k] = bt[i];

bt[i] = temp;

}

}

k++;

}

wt[0] = 0;

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

sum = sum + bt[i - 1];

wt[i] = sum - at[i];

wsum = wsum + wt[i];

}

awt[1] = (wsum / n);

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

ta = ta + bt[i];

tat[i] = ta - at[i];

tsum = tsum + tat[i];

}

atat[1] = (tsum / n);

printf("SR.\tA.T.\tB.T.\tW.T.\tT.A.T.\n");

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

printf("%3d\t%3d\t%3d\t%3d\t%4d\n", i + 1, at[i], bt[i], wt[i], tat[i]);

}

}

int main() {

printf("Welcome to CPU Scheduling:\n\n");

input();

int c, choice;

changeArrival();

int mode;

printf("Enter working mode 1.Work 2.Leisure:");

scanf("%d",&mode);

printf("Choice\tAlgorithm used\n1\tFCFS Algorithm\n2\tSJF Algorithm\n3\tRound robin\n4\tSRTF Algorithm\n5\t Improved algorithm\n");

do {

printf("Enter your choice from the above table");

scanf("%d", &c);

switch (c) {

case 1:

fcfs();

break;

case 2:

sjf();

break;

case 3:

rr();

break;

case 4:

srtf();

break;

case 5:

innovative(mode);

break;

default:

printf("Please enter choice from 1 to 5 only\n");

break;

}

display(c);

printf("\n\nEnter 1 to continue 0 to stop");

scanf("%d", &choice);

} while (choice == 1);

int var;

scanf("%d",&var);

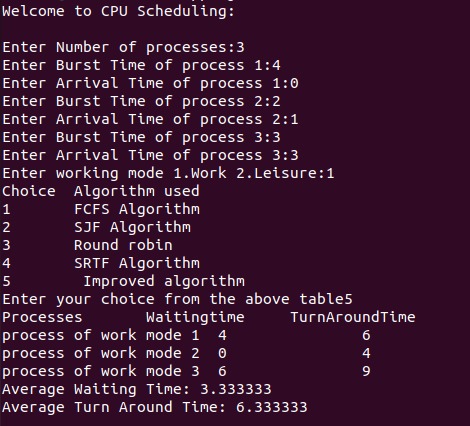
int a[5][2], i;

return 0;

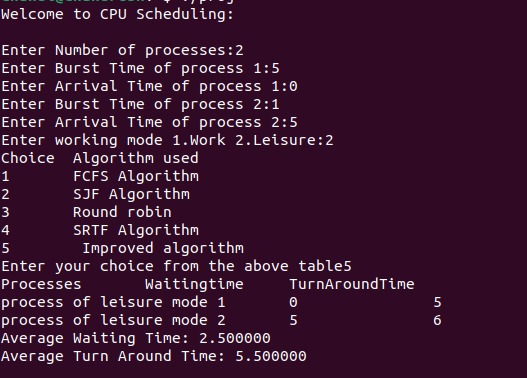
}

**Output:**

Processes running in Work mode:



Processes running in Leisure mode:



**Reference:**

Mishra MK. An improved round robin CPU scheduling algorithm. Journal of Global Research in computer science. 2012 Jul 11;3(6):64-9.