#include <stdio.h> #include <stdlib.h> #include <stdbool.h> #include <limits.h> void fcfs();

void sjf();

void priority(); void srtf();

void roundrobin();

int findmax(int a, int b)

{

return a > b ? a : b;

}

int findmin(int a, int b)

{

return a < b ? a : b;

}

void swap(int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

void main()

{

while (1)

{

printf("\n\n\t\t\tCPU SCHEDULING ALGORITHMS."); int choice;

printf("\n1.Non-Premitive\t2.Premitive\t3.Exit\n"); scanf("%d", &choice);

switch (choice)

{

case 1:

printf("1.FCFS\t2.SJF\t3.PRIORTY\n"); scanf("%d", &choice);

switch (choice)

{

case 1:

fcfs(); break;

case 2:

sjf(); break;

case 3:

priority(); break;

default:

printf("Invalid input");

}

break; case 2:

{

printf("\n1.SRTF\t2.Round Robin\n"); scanf("%d", &choice);

switch (choice)

{

case 1:

srtf(); break;

case 2:

roundrobin(); break;

default:

printf("Invalid input");

}

break;

}

case 3:

exit(0); default:

printf("Invalid input");

}

}

}

void priority()

{

int n, avg\_wt = 0;

printf("Enter Number of Processes: "); scanf("%d", &n);

// b is array for burst time, p for priority and index for process id int b[n], p[n], index[n];

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

{

printf("Enter Burst Time and Priority Value for Process %d: ", i + 1); scanf("%d %d", &b[i], &p[i]);

index[i] = i + 1;

}

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

{

int a = p[i], m = i;

// Finding out highest priority element and placing it at its desired

position

for (int j = i; j < n; j++)

{

if (p[j] > a)

{

a = p[j]; m = j;

}

}

// Swapping processes swap(&p[i], &p[m]);

swap(&b[i], &b[m]);

swap(&index[i], &index[m]);

}

// T stores the starting time of process int t = 0;

// Printing scheduled process printf("Order of process Execution is\n"); for (int i = 0; i < n; i++)

{

printf("P%d is executed from %d to %d\n", index[i], t, t + b[i]); t += b[i];

}

printf("\n");

printf("Process Id Burst Time Wait Time TurnAround Time\n"); int wait\_time = 0;

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

{

printf("P%d %d %d %d\n", index[i], b[i], wait\_time, wait\_time + b[i]);

avg\_wt += wait\_time; wait\_time += b[i];

}

printf("Average waiting time is: %g\n", (avg\_wt \* 1.0) / n);

}

void sjf()

{

int i, n, p[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, min, k = 1, btime = 0;

int bt[10], temp, j, at[10], wt[10], tt[10], ta = 0, sum = 0; float wavg = 0, tavg = 0, tsum = 0, wsum = 0;

printf(" -------Shortest Job First Scheduling ( NP ) \n");

printf("\nEnter the No. of processes :"); scanf("%d", &n);

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

{

printf("\tEnter the burst time of %d process :", i + 1); scanf(" %d", &bt[i]);

printf("\tEnter the arrival time of %d process :", i + 1); scanf(" %d", &at[i]);

}

/\*Sorting According to Arrival Time\*/

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;

}

}

}

/\*Arranging the table according to Burst time, Execution time and Arrival Time

Arrival time <= Execution time

\*/

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];

}

wavg = (wsum / n);

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

{

ta = ta + bt[i]; tt[i] = ta - at[i]; tsum = tsum + tt[i];

}

tavg = (tsum / n);

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"); printf("\n RESULT:-");

printf("\nProcess\t Burst\t Arrival\t Waiting\t Turn-around"); for (i = 0; i < n; i++)

{

printf("\n p%d\t %d\t %d\t\t%d\t\t%d", p[i], bt[i], at[i], wt[i],

tt[i]);

}

printf("\n\nAVERAGE WAITING TIME : %.2f", wavg); printf("\nAVERAGE TURN AROUND TIME : %.2f", tavg);

}

void fcfs()

{

int n, bt[20], wt[20], tat[20], avwt = 0, avtat = 0, i, j; printf("Enter total number of processes(maximum 20):"); scanf("%d", &n);

printf("Enter Process Burst Timen"); for (i = 0; i < n; i++)

{

printf("P[%d]:", i + 1);

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

}

wt[0] = 0;

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

{

wt[i] = 0;

for (j = 0; j < i; j++) wt[i] += bt[j];

}

printf("ProcessttBurst TimetWaiting TimetTurnaround Completion Time\n");

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

{

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

avtat += tat[i];

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

}

avwt /= i; avtat /= i;

printf("\n\nAverage Waiting Time:%d\n", avwt); printf("\nAverage Turnaround Time:%d", avtat);

}

void srtf()

{

struct process\_struct

{

int pid; int at; int bt;

int ct, wt, tat, rt, start\_time;

} ps[100];

int n;

float bt\_remaining[100];

bool is\_completed[100] = {false}, is\_first\_process = true; int current\_time = 0;

int completed = 0;

;

float sum\_tat = 0, sum\_wt = 0, sum\_rt = 0, total\_idle\_time = 0, prev = 0; int max\_completion\_time, min\_arrival\_time;

printf("Enter total number of processes: "); scanf("%d", &n);

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

{

printf("\nEnter Process %d Arrival Time: ", i); scanf("%d", &ps[i].at);

ps[i].pid = i;

}

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

{

printf("\nEnter Process %d Burst Time: ", i); scanf("%d", &ps[i].bt);

bt\_remaining[i] = ps[i].bt;

}

while (completed != n)

{

// find process with min. burst time in ready queue at current time int min\_index = -1;

int minimum = INT\_MAX;

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

{

if (ps[i].at <= current\_time && is\_completed[i] == false)

{

if (bt\_remaining[i] < minimum)

{

minimum = bt\_remaining[i]; min\_index = i;

}

if (bt\_remaining[i] == minimum)

{

if (ps[i].at < ps[min\_index].at)

{

minimum = bt\_remaining[i]; min\_index = i;

}

}

}

}

if (min\_index == -1)

{

current\_time++;

}

else

{

if (bt\_remaining[min\_index] == ps[min\_index].bt)

{

ps[min\_index].start\_time = current\_time; total\_idle\_time += (is\_first\_process == true) ? 0 :

(ps[min\_index].start\_time - prev);

is\_first\_process = false;

}

bt\_remaining[min\_index] -= 1; current\_time++;

prev = current\_time;

if (bt\_remaining[min\_index] == 0)

{

ps[min\_index].ct = current\_time;

ps[min\_index].tat = ps[min\_index].ct - ps[min\_index].at; ps[min\_index].wt = ps[min\_index].tat - ps[min\_index].bt; ps[min\_index].rt = ps[min\_index].start\_time - ps[min\_index].at;

sum\_tat += ps[min\_index].tat; sum\_wt += ps[min\_index].wt; sum\_rt += ps[min\_index].rt; completed++; is\_completed[min\_index] = true;

// total\_idle\_time += (is\_first\_process==true) ? 0 : (ps[min\_index].start\_time - prev);

// prev= ps[min\_index].ct; // or current\_time;

}

}

}

// Calculate Length of Process completion cycle max\_completion\_time = INT\_MIN;

min\_arrival\_time = INT\_MAX; for (int i = 0; i < n; i++)

{

max\_completion\_time = findmax(max\_completion\_time, ps[i].ct); min\_arrival\_time = findmin(min\_arrival\_time, ps[i].at);

}

// Output

printf("\nProcess No.\tAT\tCPU Burst Time\tCT\tTAT\tWT\tRT\n"); for (int i = 0; i < n; i++)

printf("%d\t\t%d\t%d\t\t%d\t%d\t%d\t%d\n", ps[i].pid, ps[i].at, ps[i].bt, ps[i].ct, ps[i].tat, ps[i].wt, ps[i].rt);

printf("\n");

printf("\nAverage Turn Around time = %f ", (float)sum\_tat / n); printf("\nAverage Waiting Time = %f ", (float)sum\_wt / n); printf("\nAverage Response Tim e= %f ", (float)sum\_rt / n);

}

void roundrobin()

{

int n, i, qt, count = 0, temp, sq = 0, bt[10], wt[10], tat[10], rem\_bt[10]; float awt = 0, atat = 0;

printf("ENTER NUMBER OF PROCESS : ");

scanf("%d", &n);

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

{

printf("ENTER BURST TIME OF PROCESS :%d : ", i + 1);

scanf("%d", &bt[i]); rem\_bt[i] = bt[i];

}

printf("ENTER QUANTUM TIME : ");

scanf("%d", &qt); while (1)

{

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

{

temp = qt;

if (rem\_bt[i] == 0)

{

count++; continue;

}

if (rem\_bt[i] > qt)

rem\_bt[i] = rem\_bt[i] - qt; else if (rem\_bt[i] >= 0)

{

temp = rem\_bt[i]; rem\_bt[i] = 0;

}

sq = sq + temp; tat[i] = sq;

}

if (n == count) break;

}

printf("\nPROCESS\tBURST TIME\tTURN AROUND TIME\tWAITING TIME\n");

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

{

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

atat = atat + tat[i];

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

}

awt = awt / n; atat = atat / n;

printf("\nAVERAGE WAITING TIME=%f\n", awt); printf("AVERAGE TURN AROUND TIME=%f", atat);

}