**EX.NO: DATE :**

**CS22411 – OPERATING SYSTEM LABORATORY**

**IMPLEMENTATION OF CPU SCHEDULING ALGORITHMS**

1. **ROUND ROBIN ALGORITHM:**

**AIM:**

To implement the Round Robin CPU Scheduling Algorithm for Process Scheduling using C.

# ALGORITHM:

* 1. Start.
  2. Get the number of processes, the process id, the burst time and the time quantum.
  3. Copy the burst time into a temp array.
  4. Check the burst time and the time quantum and decide on the following actions.
     1. If the burst time is greater than the time quantum:
  5. For finishing time, add the time quantum.
  6. For burst time, subtract the time quantum.
     1. B) If the burst time is 0: reduce the number of processes.
     2. If the burst time is lesser than time quantum:
  7. For finishing time, add the time quantum.
  8. For burst time, make it 0
  9. Reduce the number of processes.
  10. Print the calculated values, the total and average times.
  11. Stop.

# SOURCE CODE:

#include <stdio.h> #include <stdlib.h>

int main ()

{

int n, i, j, x, t;

printf ("\n ROUND ROBIN CPU SCHEDULING ALGORITHM: ");

printf ("\n\n Enter the number of processess: "); scanf ("%d", &n);

x = n;

int final = 0;

int pid[n], bt[n], temp[n], ft[n]; int tot\_tat=0, tot\_wt=0;

double avg\_tat, avg\_wt;

printf ("\n Enter the processes id: ");

for(i=0; i<n; i++) scanf("%d",&pid[i]);

printf ("\n Enter the burst time: "); for (i=0; i<n; i++) {

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

ft[i] = 0;

}

printf ("\n Enter the Time Quantum: "); scanf ("%d", &t);

while(n>0) { for(i=0;i<x;i++) { if (bt[i] >= t) { ft[i] = final + t; final = final + t; bt[i] = bt[i]- t;

if(bt[i] == 0)

n--;

}

else if (bt[i] > 0) { ft[i] = final + bt[i]; final = final + bt[i]; bt[i] = 0;

n--;

}

}

}

printf ("\n The Process Specifications: \n"); printf ("\n PID \t BT \t FT \t WT \t TAT"); for (i=0; i<x; i++) {

printf ("\n %d \t %d \t %d \t %d \t %d", pid[i], temp[i], ft[i], ft[i]-temp[i], ft[i]);

tot\_tat += ft[i];

tot\_wt += ft[i]-temp[i];

}

printf ("\n\n The Total Waiting Time: %d", tot\_wt); printf ("\n The Total Turn Around Time: %d", tot\_tat); avg\_tat = (float)tot\_tat/x;

avg\_wt = (float)tot\_wt/x;

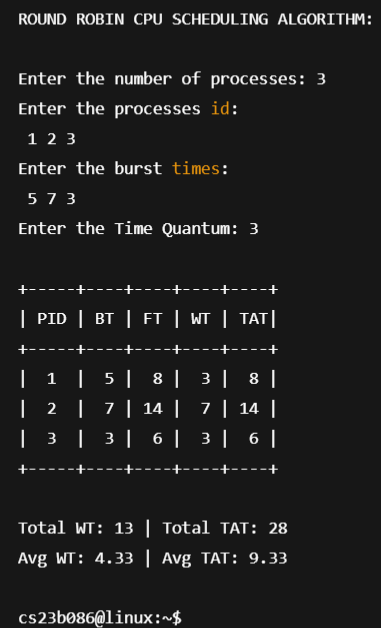
printf ("\n\n The Average Turn Around Time: %.2f", avg\_tat); printf ("\n The Average Waiting Time: %.2f", avg\_wt);

return 0;

}

# OUTPUT:

# 

****

# RESULT:

The Round Robin CPU Scheduling Algorithm has been implemented successfully.

# SHORTEST JOB FIRST (SJF) ALGORITHM:

**AIM:**

To implement the SJF CPU Scheduling Algorithm for Process Scheduling using C.

# ALGORITHM:

* 1. Start.
  2. Get the number of processes, the process id, and the burst time.
  3. Copy the burst time into a temp array.
  4. Sort all the arrays based on the burst time.
  5. Calculate the finishing time for each process following the FCFS technique.
  6. Print the specifications.
  7. Stop.

# SOURCE CODE:

#include <stdio.h> #include <stdlib.h>

void swap (int \*x, int \*y)

{

int temp; temp = \*x;

\*x = \*y;

\*y = temp;

}

int main ()

{

int n,i,j;

printf ("\n SHORTEST JOB FIRST CPU SCHEDULING ALGORITHM:");

printf ("\n\n Enter the number of processess: "); scanf ("%d", &n);

int pid[n], bt[n], tat[n], wt[n], ft[n]; int tot\_tat=0, tot\_wt=0;

double avg\_tat, avg\_wt;

printf ("\n Enter the Process ID: "); for(i=0; i<n; i++) scanf("%d",&pid[i]);

printf ("\n Enter the burst time: "); for (i=0; i<n; i++)

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

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

{

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

{

if (bt[i] > bt[j])

{

swap(&bt[i], &bt[j]);

swap(&pid[i], &pid[j]);

}

}

}

int x = pid[0];

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

{

if (pid[i] == x)

{

ft[i] = bt[i]; wt[i] = 0; tat[i] = ft[i];

}

else

{

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

wt[i] = ft[i-1];

tat[i] = ft[i];

}

tot\_tat += tat[i]; tot\_wt += wt[i];

}

printf ("\n");

printf ("\n The Process Specifications: \n"); printf ("\n PID \t BT \t FT \t WT \t TAT"); for (i=0; i<n; i++)

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

printf ("\n");

printf ("\n The Total Waiting Time: %d", tot\_wt); printf ("\n The Total Turn Around Time: %d", tot\_tat);

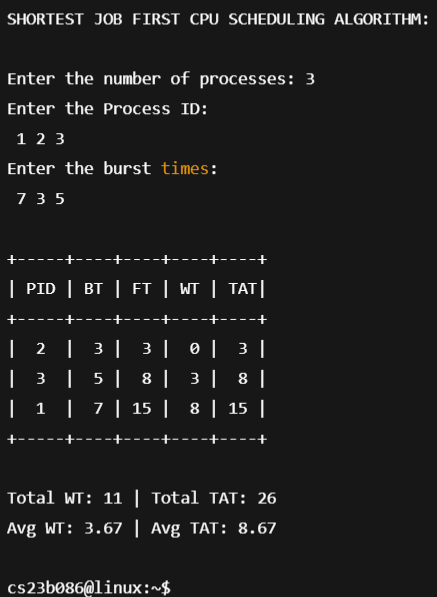
avg\_tat = (float)tot\_tat/n; avg\_wt = (float)tot\_wt/n; printf ("\n");

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

return 0;

}

# OUTPUT:

****

# RESULT:

The Shortest Job First CPU Scheduling Algorithm has been implemented successfully.

# PRIORITY BASED ALGORITHM:

**AIM:**

To implement the Priority Based CPU Scheduling Algorithm for Process Scheduling using C.

# ALGORITHM:

* 1. Start.
  2. Get the number of processes, the process id, the burst time and the priority.
  3. Copy the burst time into a temp array.
  4. Sort all the arrays based on the given priority.
  5. Calculate the finishing time for each process following the FCFS technique.
  6. Print the specifications.
  7. Stop.

# SOURCE CODE:

#include <stdio.h> #include <stdlib.h>

void swap (int \*x, int \*y)

{

int temp; temp = \*x;

\*x = \*y;

\*y = temp;

}

int main ()

{

int n,i,j;

printf ("\n\n PRIORITY BASED CPU SCHEDULING ALGORITHM: ");

printf ("\n\n Enter the number of processess: "); scanf ("%d", &n);

int pid[n], prio[n], bt[n], tat[n], wt[n], ft[n]; int tot\_tat=0, tot\_wt=0;

double avg\_tat, avg\_wt;

printf ("\n Enter the Process ID: "); for(i=0; i<n; i++) scanf("%d",&pid[i]);

printf ("\n Enter the burst time: "); for (i=0; i<n; i++)

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

printf ("\n Enter the Priority: "); for (i=0; i<n; i++)

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

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

{

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

{

if (prio[i] > prio[j])

{

swap (&prio[i], &prio[j]);

swap (&bt[i], &bt[j]);

swap (&pid[i], &pid[j]);

}

}

}

int x = pid[0];

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

{

if (pid[i] == x)

{

ft[i] = bt[i]; wt[i] = 0; tat[i] = ft[i];

}

else

{

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

wt[i] = ft[i-1];

tat[i] = ft[i];

}

tot\_tat += tat[i]; tot\_wt += wt[i];

}

printf ("\n The Process Specifications: \n"); printf ("\n PID \t BT \t FT \t WT \t TAT"); for (i=0; i<n; i++)

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

printf ("\n The Total Waiting Time: %d", tot\_wt); printf ("\n The Total Turn Around Time: %d", tot\_tat);

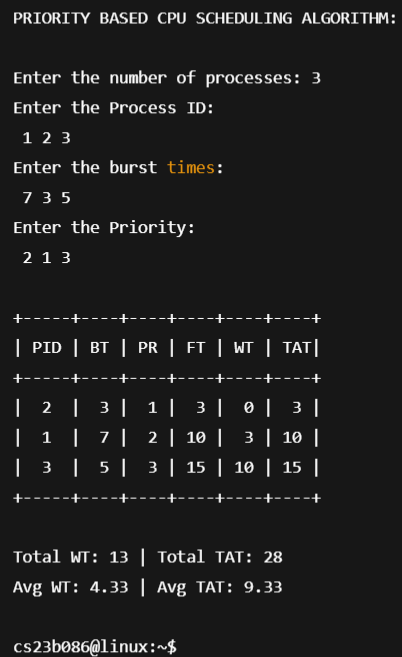
avg\_tat = (float)tot\_tat/n; avg\_wt = (float)tot\_wt/n;

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

return 0;

}

# OUTPUT:



# RESULT:

The Priority Based CPU Scheduling Algorithm has been implemented successfully.

# FIRST COME FIRST SERVE (FCFS) ALGORITHM: AIM:

To implement the First Come First Serve CPU Scheduling Algorithm for Process Scheduling using C.

# ALGORITHM:

* 1. Start.
  2. Get the number of processes, the process id and the burst time.
  3. Sort all the arrays based on the given priority.
  4. As the FCFS technique, finish the first process and proceed to another.
  5. Add the burst time to the FT and increment it completely.
  6. Calculate TAT and WT as FT and FT-BT.
  7. Print the specifications.
  8. Stop.

# SOURCE CODE:

#include <stdio.h> #include <stdlib.h>

int main ()

{

int n,i;

printf ("\n FIRST COME FIRST SERVE CPU SCHEDULING ALGORITHM:");

printf ("\n\n Enter the number of processess: "); scanf ("%d", &n);

int pid[n], bt[n], tat[n], wt[n], ft[n]; int tot\_tat=0, tot\_wt=0;

double avg\_tat, avg\_wt;

printf ("\n Enter the Process ID: "); for(i=0; i<n; i++) scanf("%d",&pid[i]);

printf ("\n Enter the burst time: "); for (i=0; i<n; i++)

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

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

{

if (pid[i] == 1)

{

ft[i] = bt[i]; wt[i] = 0; tat[i] = ft[i];

}

else

{

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

wt[i] = ft[i-1];

tat[i] = ft[i];

}

tot\_tat += tat[i]; tot\_wt += wt[i];

}

printf ("\n The Process Specifications: \n"); printf ("\n PID \t BT \t FT \t WT \t TAT"); for (i=0; i<n; i++)

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

printf ("\n The Total Waiting Time: %d", tot\_wt); printf ("\n The Total Turn Around Time: %d", tot\_tat);

avg\_tat = (float)tot\_tat/n; avg\_wt = (float)tot\_wt/n;

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

return 0;

}

# OUTPUT:

# 

# RESULT:

The First Come First Serve CPU Scheduling Algorithm has been implemented successfully.