Ex. No: 6a
Date: 15/2/25

FIRST COME FIRST SERVE

AIM:

To implement First-come First- serve (FCFS) scheduling technique

ALGORITHM**:**

1. Get the number of processes from the user.
2. Read the process name and burst time.
3. Calculate the total process time.
4. Calculate the total waiting time and total turnaround time for each process
5. Display the process name & burst time for each process.
6. Display the total waiting time, average waiting time, turnaround time.

PROGRAM:

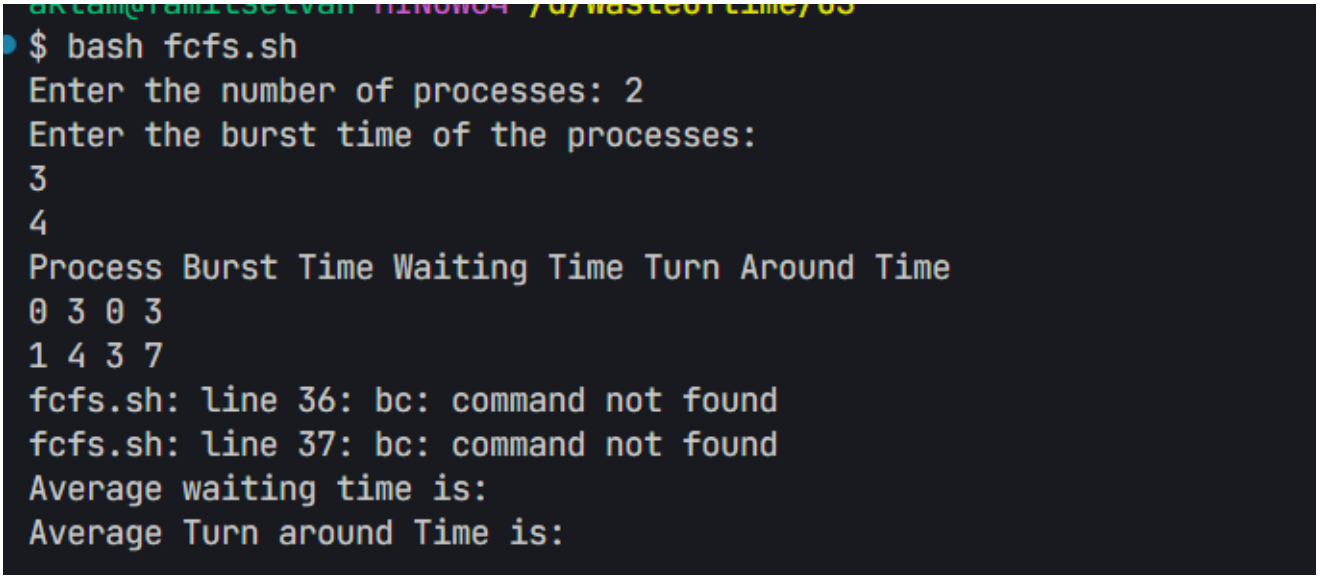
```
#include <stdio.h>

int main() {
int pid[15], bt[15], wt[15], n; float twt = 0, ttat = 0;
printf("Enter the number of processes: "); scanf("%d", &n);

printf("Enter process ID of all the processes:\n"); for (int i = 0; i < n; i++) {
scanf("%d", &pid[i]);
}
printf("Enter burst time of all the processes:\n"); for (int i = 0; i < n; i++) {
scanf("%d", &bt[i]);
}
```

2116231801161

```
wt[0] = 0;
// Calculate waiting time for all other processes for (int i = 1; i < n; i++) {
wt[i] = wt[i - 1] + bt[i - 1];
}

printf("\nProcess ID\tBurst Time\tWaiting Time\tTurnaround Time\n");
```

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

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

printf("\nAverage waiting time = %.2f\n", twt / n); printf("Average turnaround time = %.2f\n", ttat / n);
return 0;
}
```

OUTPUT:



RESULT:

The Program of first come first serve is successfully implemented.

Ex. No: 6b
Date: 15/2/25

SHORTEST JOB FIRST

AIM:

To implement the Shortest Job First (SJF) scheduling technique

ALGORITHM:

1. Declare the structure and its elements.
2. Get a number of processes as input from the user.
3. Read the process name, arrival time and burst time
4. Initialize waiting time, turnaround time & flag of read processes to zero.
5. Sort based on the burst time of all processes in ascending order.
6. Calculate the waiting time and turnaround time for each process.
7. Calculate the average waiting time and average turnaround time.
8. Display the results.

PROGRAM:

```
#include <stdio.h>

int main() {

int A[100][4]; // A[i][0]=PID, A[i][1]=BT, A[i][2]=WT, A[i][3]=TAT

int i, j, n, total = 0, index, temp; float avg_wt, avg_tat;


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

printf("Enter Burst Time:\n"); for (i = 0; i < n; i++) { printf("P%d: ", i + 1); scanf("%d", &A[i][1]);

A[i][0] = i + 1; // Assign process ID

}
```

2116231801161

```
for (i = 0; i < n; i++) { index = i;
for (j = i + 1; j < n; j++) { if (A[j][1] < A[index][1]) index = j;
}


temp = A[i][1];
```

```
A[i][1] = A[index][1]; A[index][1] = temp;



temp = A[i][0]; A[i][0] = A[index][0]; A[index][0] = temp;
}

A[0][2] = 0;
for (i = 1; i < n; i++) { A[i][2] = 0;
for (j = 0; j < i; j++) { A[i][2] += A[j][1];
}
total += A[i][2];
}
avg_wt = (float) total / n;

total = 0; printf("\nProcess\tBT\tWT\tTAT\n"); for (i = 0; i < n; i++) {
A[i][3] = A[i][1] + A[i][2]; // TAT = BT + WT
total += A[i][3];
printf("P%d\t%d\t%d\t%d\n", A[i][0], A[i][1], A[i][2], A[i][3]);
}
avg_tat = (float) total / n;

printf("\nAverage Waiting Time = %.2f", avg_wt); printf("\nAverage Turnaround Time = %.2f\n", avg_tat);

return 0;
}
```

```
$ bash sjf.sh
Enter the number of processes: 2
Enter the burst time of the processes:
1
2
Process Burst Time Waiting Time Turn Around Time
1 1 0 1
2 2 1 3
```

RESULT:

The Program Shortest Job First is successfully implemented.

2116231801161

Ex. No: 6c

Date: 16/2/25

PRIORITY SCHEDULING

AIM:

To implement a priority scheduling technique

ALGORITHM:

1. Get the number of processes from the user.
2. Read the process name, burst time and priority of the process.
3. Sort based on burst time of all processes in ascending order based on priority
4. Calculate the total waiting time and total turnaround time for each process
5. Display the process name & burst time for each process.
6. Display the total waiting time, average waiting time, turnaround time.

PROGRAM:

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

void swap(int *a, int *b) { int temp = *a;

*a = *b;

*b = temp;

}

int main() { int n;

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

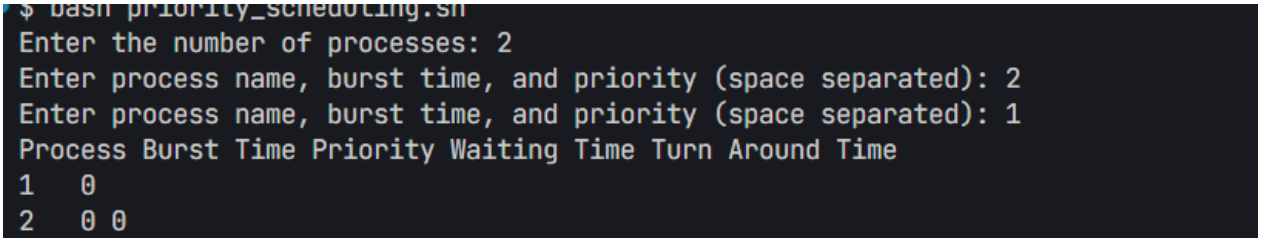
int *burst = (int*)malloc(n * sizeof(int)); int *priority = (int*)malloc(n * sizeof(int)); int *pid = (int*)malloc(n * sizeof(int));

int total_wait = 0, total_turnaround = 0;

```
for (int i = 0; i < n; i++) {
printf("Enter Burst Time and Priority for Process %d: ", i + 1); scanf("%d %d", &burst[i], &priority[i]);
pid[i] = i + 1;
}
for (int i = 0; i < n - 1; i++) { for (int j = i + 1; j < n; j++) {
if (priority[j] > priority[i]) { swap(&priority[i], &priority[j]); swap(&burst[i], &burst[j]);
swap(&pid[i], &pid[j]);
}
}
}
```

```
int wait_time = 0;

printf("\nProcess  Burst Time Wait Time Turnaround Time\n");

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

int turnaround_time = wait_time + burst[i]; total_wait += wait_time;

total_turnaround += turnaround_time;


printf("P%d       %d       %d       %d\n", pid[i], burst[i], wait_time, turnaround_time);

wait_time += burst[i];

}

printf("\nAverage Waiting Time: %.2f\n", (float)total_wait / n); printf("Average Turnaround Time: %.2f\n",
(float)total_turnaround / n);


free(burst); free(priority); free(pid);


return 0;

}
```

OUTPUT:



RESULT:

The Program of Priority scheduling is successfully implemented.

2116231801161

2116231801161

Ex. No: 6d

Date: 16/2/25

ROUND ROBIN SCHEDULING

AIM:

To implement the round-robin (RR) scheduling technique

ALGORITHM:

1. Declare the structure and its elements.
2. Get a number of processes and Time quantum as input from the user.
3. Read the process name, arrival time and burst time
4. Create an array rem_bt[] to keep track of the remaining burst time of processes which is initially copy of bt[] (burst times array)
5. Create another array wt[] to store waiting times of processes. Initialize this array as 0.
6. Initialize time : t = 0
7. Keep traversing all processes while all processes are not done. Do the following for i'th process if it is not done yet.

a- If rem_bt[i] > quantum

(i) t = t + quantum

(ii)bt_rem[i] -= quantum;

b- Else // Last cycle for this process

(i) t = t + bt_rem[i];

(ii)wt[i] = t - bt[i]

(iii) bt_rem[i] = 0; // This process is over

8. Calculate the waiting time and turnaround time for each process.
9. Calculate the average waiting time and average turnaround time.
10. Display the results.

PROGRAM:

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

int main() {
int n, time_quantum;
printf("Enter number of processes: "); scanf("%d", &n);
int *arrival = (int*)malloc(n * sizeof(int)); int *burst = (int*)malloc(n * sizeof(int));
int *remaining = (int*)malloc(n * sizeof(int));
int wait_time = 0, turnaround_time = 0, total = 0, x = n;

for (int i = 0; i < n; i++) {
printf("Enter arrival time and burst time for process %d: ", i + 1); scanf("%d %d", &arrival[i], &burst[i]);
remaining[i] = burst[i];
}
```

```
printf("Enter time quantum: ");

scanf("%d", &time_quantum);printf("\nProcess\tBurst\tTurnaround\tWaiting\n");

for (int i = 0; x != 0;) { if (remaining[i] > 0) {

if (remaining[i] <= time_quantum) { total += remaining[i]; remaining[i] = 0;

x--;

printf("P%d\t%d\t%d\t\t%d\n", i + 1, burst[i], total - arrival[i], total - arrival[i] - burst[i]); wait_time += total - arrival[i] - burst[i];

turnaround_time += total - arrival[i];

} else {

remaining[i] -= time_quantum; total += time_quantum;

}

}


i = (i + 1) % n;

}


printf("\nAverage Waiting Time: %.2f", (float)wait_time / n); printf("\nAverage Turnaround Time: %.2f\n", (float)turnaround_time / n);


free(arrival); free(burst); free(remaining);


return 0;

}
```

OUTPUT:

```
$ bash round_robin.sh
Enter the number of processes: 2
Enter process name and burst time (space separated): 1
Enter process name and burst time (space separated): 1
Enter Time Quantum: 2
round_robin.sh: line 31: [: -gt: unary operator expected
round_robin.sh: line 31: [: -gt: unary operator expected
Process Burst Time Waiting Time Turn Around Time
1  0 0
1  0 0
round_robin.sh: line 62: bc: command not found
round_robin.sh: line 63: bc: command not found
Average waiting time is:
Average Turn Around Time is:
```

RESULT:

The Program of Round Robin Scheduling is successfully implemented.