Name: V.P.Abishek **Roll no:** 205001002

Ex 4: Implementation of CPU Scheduling Policies: Priority and Round Robin

AIM:

To develop a menu driven C program to implement the CPU scheduling algorithm Round robin and priority

ALGORITHM:

1) Round robin

```
1. Start
2. Accept number of process and process information .
3. Copy the burst time of the process into a temporary array and initialize time to
zero . Initialize ‘t’ to 0, ‘comp’ to 0, ‘quanta’ to 5 and ‘chk’ to 0.
4. While comp is equal to number of processes
        ● If the current process’ temporary burst time is greater than zero and the
        current process has arrival time less than or equal to ‘t’ and burst time is
        less than or equal to quarter ,then update the burst time to zero and chk
        to 1.
        ● Else if the current process has arrival time less than or equal to ‘t’ and its
        burst time is greater than 0 , update the burst time to burst time
        subtracted by quarter
        ● If the current process’ burst time is 0 and chk is 1,
        * Increase ‘comp’ value by 1
        * Calculate the waiting time and turnaround time of that process
        ● Increase ‘t’ count by 1.
        ● End while
5. Calculate and print the average waiting time and average turnaround time and
```

print the gantt chart.

6. Stop

2) Priority Preemptive

1. Start

2. Accept number of process and process

3. Copy burst time and the process’ priority into temporary arrays and make use of these values.

4. Initialize ‘comp’ to 0 (to count the number of completed process), ‘t’ to 0

5. While comp is not equal to the number of processes

    ● Assign min to a high value

    ● Run a for loop from 0 to number of processes

    If priority if current process is less than ‘min’ abd current process’ arrival time is less than or equal to ‘t’ and burst time is greater than 0 assign ind to current index , min to process’ priority

    * Update burst time of ‘ind’ process

    * If Burst time of ‘ind’ process is zero , then increase the ‘comp’ count and calculate the current process’ waiting time.

    * Increase the count of time ‘t’ by one

    ● End while

6. Print the gantt chart for the processes

7. Print the details of process and calculate the average turnaround time and waiting time and print them

8. Stop

CODE:

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

```c
#include <stdlib.h>
#include <string.h>

#define MAX 1000

typedef struct process {
  char pid[5];
  int burst_time;
  int arrival_time;
  int remaining_time;
  int waiting_time;
  int turnaround_time;
  int priority;
  int waiting;
} process;

char chart[MAX][5];
int time;

int min(int a,int b){
return a<b ?a:b;
}

void gantt_chart(int limit) {
   printf("\nGANTT CHART:\n\n");
   printf("-");
   int cur = 0, count = 1;
   for (int i = 1; i <= time; i ++ ) {
   if (count < limit && !strcmp(chart[i], chart[i - 1])) {
  }
   count ++;
   }
   else {
   printf("-");
  for (int j = 0; j < count; j ++) {
    printf("--");
   }
   count = 1;
   printf("--");
   }
   }
   printf("\n|");
   cur = 0, count = 1;
   for (int i = 1; i <= time; i ++ ) {
   if (count < limit && !strcmp(chart[i], chart[i - 1])) {
   count ++;
   }
   else {
   for (int j = 0; j < count; j ++) {
   printf(" ");
   }
   printf("%s", chart[i - 1]);
   for (int j = 1; j < count; j ++) {
   printf(" ");
   }
   count = 1;
   printf(" |");
   }
   }
```

```
cur = 0, count = 1;
printf("\n-");
for (int i = 1; i <= time; i ++ ) {
  if (count < limit && !strcmp(chart[i], chart[i - 1])) {
count ++;
}
else {
printf("-");
for (int j = 0; j < count; j ++) {
printf("--");
}
count = 1;
printf("--");
}
}
cur = 0, count = 1;
printf("\n0");
for (int i = 1; i <= time; i ++ ) {
if (count < limit && !strcmp(chart[i], chart[i - 1])) {
count ++;
}
else {
 printf(" ");
 for (int j = 0; j < count; j ++) {
  printf("  ");
 }
 cur += count;
 count = 1;
 printf("%2d", cur);
```

```
  }
 }

 printf("\n");
}
void p_table(process task[], int n) {
  double avg_of_waiting_time, avg_of_turnaround_time;
  int sum_of_waiting_time = 0, sum_of_turnaround_time = 0;
  int i;
  printf("\nTABLE:\n");
  printf("--------------------------------------------------------------------------------\n");
  puts("| PID | Arrival Time | Burst Time | Waiting TimeTurnaround Time | Priority |");
  printf("--------------------------------------------------------------------------------\n");
  for (i = 1; i <= n; i++) {
  printf("| %s | %2d | %2d | %2d| %2d | %2d |\n", task[i].pid,
 task[i].arrival_time, task[i].burst_time,
 task[i].waiting_time,
 task[i].turnaround_time,task[i].priority);
    sum_of_waiting_time += task[i].waiting_time;
  sum_of_turnaround_time += task[i].turnaround_time;
  }
  avg_of_waiting_time = 1.0l * sum_of_waiting_time / n;
  avg_of_turnaround_time = 1.0l * sum_of_turnaround_time /
 n;
  printf("--------------------------------------------------------------------------------\n");
  printf("| Average: | %7.4f |%7.4f | |\n", avg_of_waiting_time,
 avg_of_turnaround_time);
  printf("--------------------------------------------------------------------------------\n");
}
```

```
void rr_table(process task[], int n) {
double avg_of_waiting_time, avg_of_turnaround_time;
int sum_of_waiting_time = 0, sum_of_turnaround_time = 0;
int i;
printf("\nTABLE:\n");
printf("------------------------------------------------------------------\n");
puts("| PID | Arrival Time | Burst Time | Waiting Time |Turnaround Time |");
printf("------------------------------------------------------------------\n");
for (i = 1; i <= n; i++) {
printf("| %s | %2d | %2d | %2d| %2d |\n", task[i].pid, task[i].arrival_time,
task[i].burst_time, task[i].waiting_time,
task[i].turnaround_time);
sum_of_waiting_time += task[i].waiting_time;
sum_of_turnaround_time += task[i].turnaround_time;
}
avg_of_waiting_time = 1.0l * sum_of_waiting_time / n;
avg_of_turnaround_time = 1.0l * sum_of_turnaround_time /
n;
printf("------------------------------------------------------------------\n");
printf("| Average: | %7.4f |%7.4f |\n", avg_of_waiting_time, avg_of_turnaround_time);
printf("------------------------------------------------------------------\n");
}

void priority_preemp() {
int number_of_processes,no_of_processes_completed=0;
process task[MAX];
int current_time = 0;
  printf("Enter Number of Processes: ");
```

```
scanf("%d", & number_of_processes);

printf("\n");

for (int i = 1; i <= number_of_processes; i++) {

printf("Enter Process ID: ");

scanf("%s", task[i].pid);

printf("Enter Arrival Time: ");

scanf("%d", & task[i].arrival_time);

printf("Enter Burst Time: ");

scanf("%d", & task[i].burst_time);

printf("Enter Priority: ");

scanf("%d", & task[i].priority);

task[i].waiting = 1;

task[i].remaining_time = task[i].burst_time;

printf("\n");

}

int current_process = 0;

task[current_process].priority=MAX;

while (no_of_processes_completed!=number_of_processes)

{

current_process = 0;

int no_of_process_waiting = 0;

for (int i = 1; i <= number_of_processes; i++) {

if (task[i].waiting && task[i].arrival_time <=

current_time && task[i].priority <

task[current_process].priority){

current_process = i;

}

if (task[i].waiting)

no_of_process_waiting += 1;
}
if (!no_of_process_waiting) {
time = current_time;
break;
}
if (current_process == 0) {
char s[5] = " ";
strcpy(chart[current_time], s);
current_time += 1;
continue;
}
if (task[current_process].waiting &&
task[current_process].arrival_time <= current_time) {
task[current_process].remaining_time -= 1;
strcpy(chart[current_time],
task[current_process].pid);
if (!task[current_process].remaining_time) {
task[current_process].waiting = 0;
no_of_processes_completed++;
task[current_process].turnaround_time = current_time
+ 1 - task[current_process].arrival_time;
task[current_process].waiting_time =
task[current_process].turnaround_time -
task[current_process].burst_time;
}
}
current_time += 1;
}
p_table(task, number_of_processes);
```

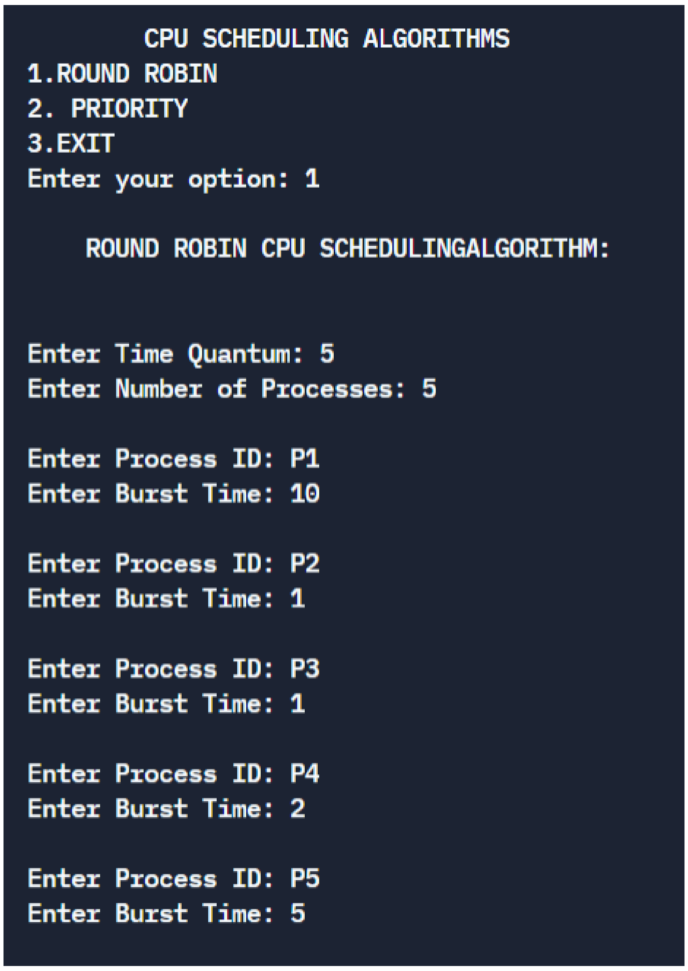
```
gantt_chart(MAX);
}

void round_robin() {
int number_of_processes;
int quantum;
int current_time = 0;
process task[MAX];
printf("\nEnter Time Quantum: ");
scanf("%d", &quantum);
printf("Enter Number of Processes: ");
scanf("%d", & number_of_processes);
printf("\n");
for (int i = 1; i <= number_of_processes; i++) {
printf("Enter Process ID: ");
scanf("%s", task[i].pid);
printf("Enter Burst Time: ");
scanf("%d", & task[i].burst_time);
task[i].arrival_time = 0;
task[i].waiting = 1;
task[i].remaining_time = task[i].burst_time;
printf("\n");
}
time = 0;
while (1)
{
int no_of_process_waiting = 0;
for (int i = 1; i <= number_of_processes; i ++) {
if (task[i].waiting) {
```

```
no_of_process_waiting += 1;

int mintime = min(quantum, task[i].remaining_time);

for (int j = 0; j < mintime; j ++) {

strcpy(chart[j + current_time], task[i].pid);

}

current_time += mintime;

task[i].remaining_time -= mintime;

if (!task[i].remaining_time) {

no_of_process_waiting -= 1;

task[i].waiting = 0;

task[i].waiting_time = current_time -

task[i].burst_time;

task[i].turnaround_time = task[i].waiting_time +

task[i].burst_time;

}

}

}

if (!no_of_process_waiting) {

time = current_time;

break;

}

}

rr_table(task, number_of_processes);

gantt_chart(quantum);

}


int main(void) {

  int choice;
```

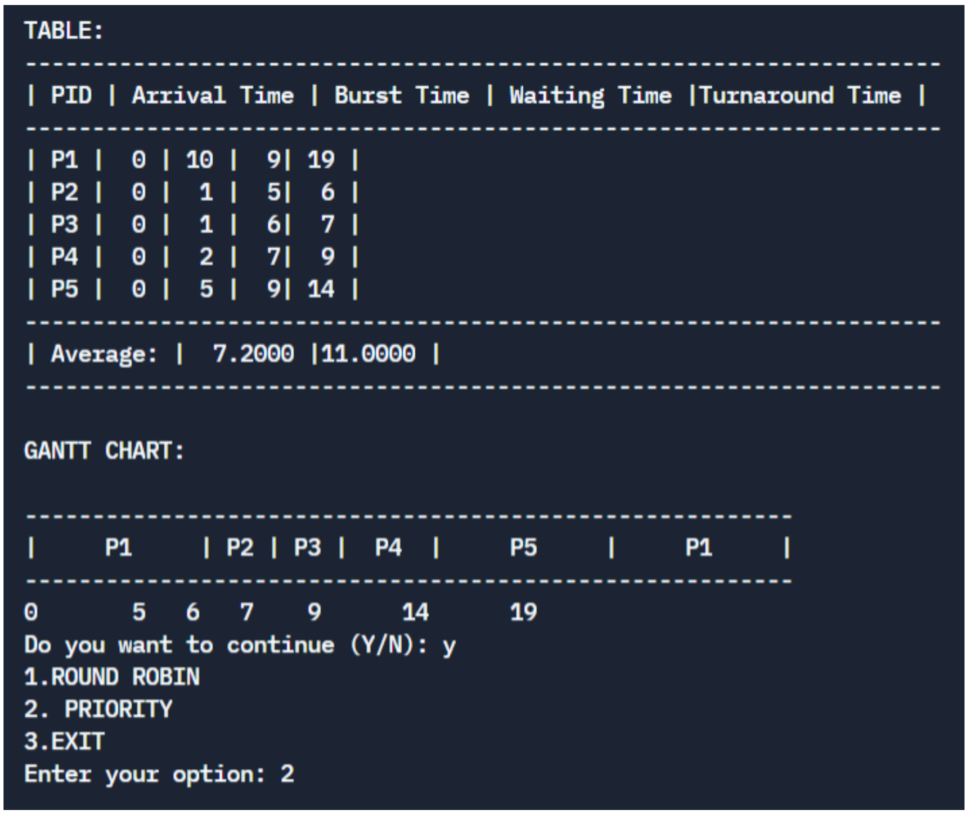
```
printf("\t\tCPU SCHEDULING ALGORITHMS\t\t\n");

int cont = 1;

while(cont == 1) {
 for (int i = 0; i < 1000; i++) strcpy(chart[i], " ");
 printf("1.ROUND ROBIN\n");
 printf("2. PRIORITY\n");
 printf("3.EXIT\n");
 printf("Enter your option: ");
 scanf("%d", &choice);

 switch(choice) {
  case 1:
   printf("\n\tROUND ROBIN CPU SCHEDULINGALGORITHM:\n\n");
   round_robin();
   break;
  case 2:
   printf("\n\tPRIORITY PREEMPTIVE CPU SCHEDULING ALGORITHM:\n\n");
   priority_preemp();
   break;
  case 3:
   printf("Processes terminated!\n");
   return 0;
   break;
  default:
   printf("Enter a valid choice!\n");
   break;
 }
```

```
    char ch;

    printf("Do you want to continue (Y/N): ");

    scanf("\n%c", &ch);

    if(ch == 'N' || ch == 'n') {

      cont = 0;

    }

  }


  return 0;

}
```

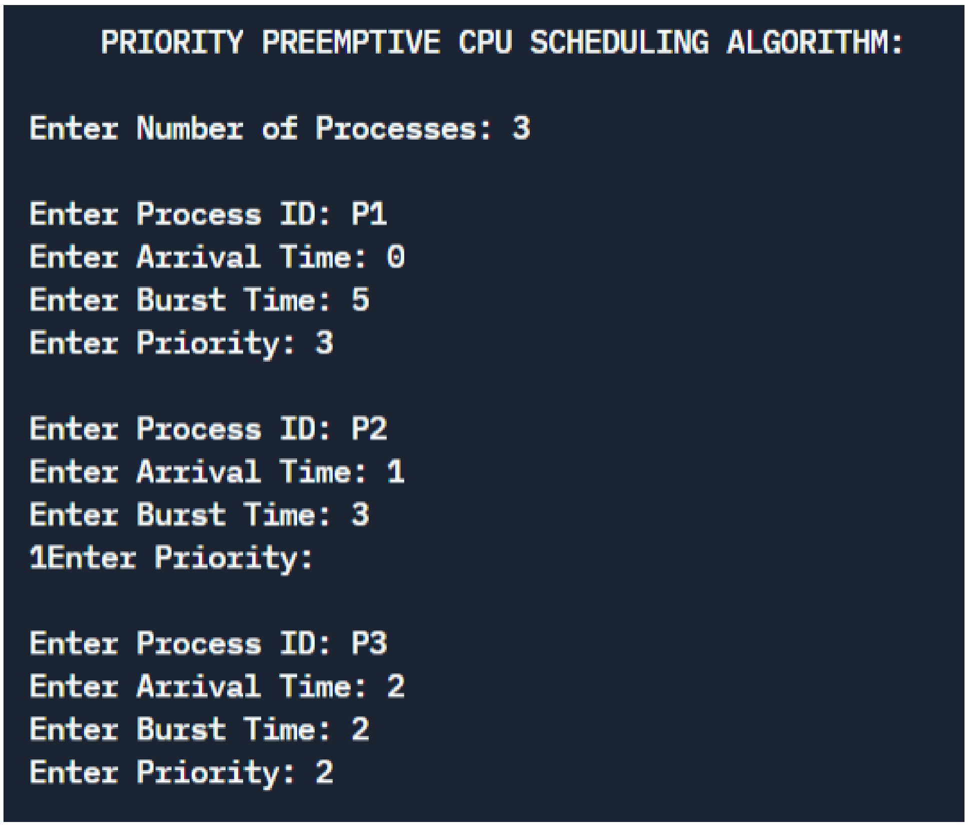
SAMPLE I/O:



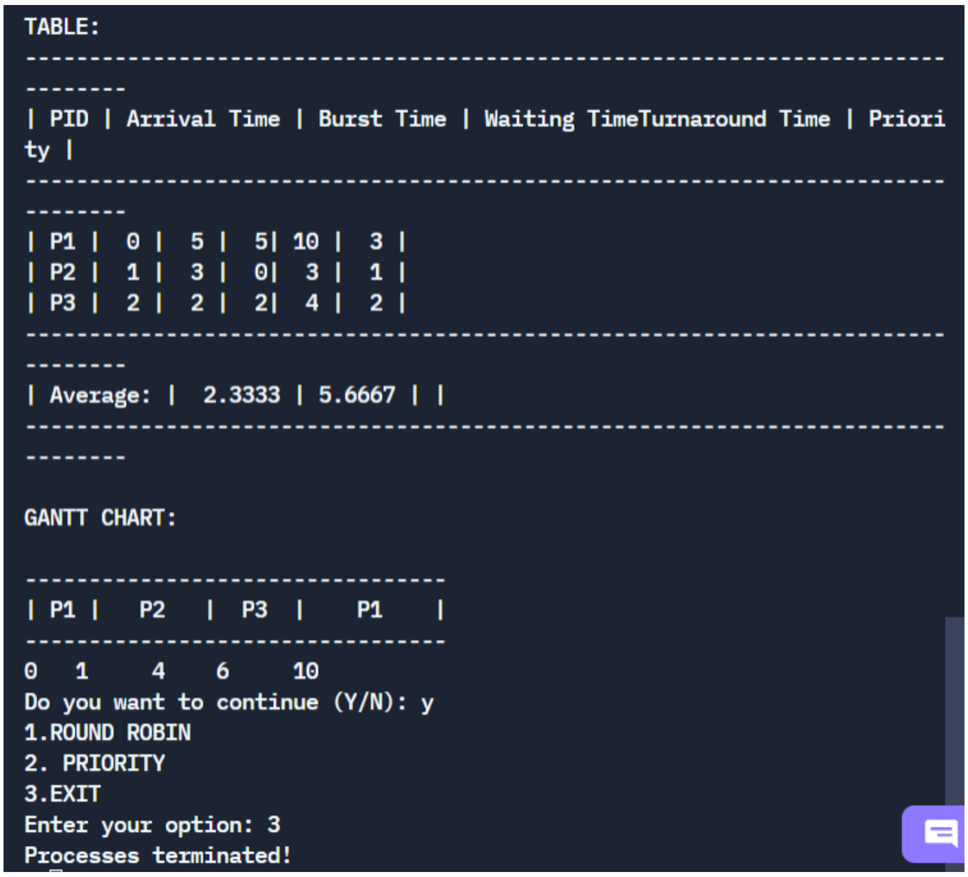
```
        CPU SCHEDULING ALGORITHMS
1.ROUND ROBIN
2. PRIORITY
3.EXIT
Enter your option: 1

    ROUND ROBIN CPU SCHEDULINGALGORITHM:


Enter Time Quantum: 5
Enter Number of Processes: 5

Enter Process ID: P1
Enter Burst Time: 10

Enter Process ID: P2
Enter Burst Time: 1

Enter Process ID: P3
Enter Burst Time: 1

Enter Process ID: P4
Enter Burst Time: 2

Enter Process ID: P5
Enter Burst Time: 5
```



```
TABLE:
--------------------------------------------------------------------
| PID | Arrival Time | Burst Time | Waiting Time |Turnaround Time |
--------------------------------------------------------------------
| P1 |  0 | 10 |  9| 19 |
| P2 |  0 |  1 |  5|  6 |
| P3 |  0 |  1 |  6|  7 |
| P4 |  0 |  2 |  7|  9 |
| P5 |  0 |  5 |  9| 14 |
--------------------------------------------------------------------
| Average: |  7.2000 |11.0000 |
--------------------------------------------------------------------

GANTT CHART:

---------------------------------------------------------
|     P1     | P2 | P3 |  P4  |      P5     |      P1     |
---------------------------------------------------------
0       5   6    7    9      14      19
Do you want to continue (Y/N): y
1.ROUND ROBIN
2. PRIORITY
3.EXIT
Enter your option: 2
```



```
     PRIORITY PREEMPTIVE CPU SCHEDULING ALGORITHM:

 Enter Number of Processes: 3

 Enter Process ID: P1
 Enter Arrival Time: 0
 Enter Burst Time: 5
 Enter Priority: 3

 Enter Process ID: P2
 Enter Arrival Time: 1
 Enter Burst Time: 3
 1Enter Priority:

 Enter Process ID: P3
 Enter Arrival Time: 2
 Enter Burst Time: 2
 Enter Priority: 2
```



```
TABLE:
----------------------------------------------------------------------
--------
| PID | Arrival Time | Burst Time | Waiting TimeTurnaround Time | Priori
ty |
----------------------------------------------------------------------
--------
| P1 |  0 |  5 |  5| 10 |  3 |
| P2 |  1 |  3 |  0|  3 |  1 |
| P3 |  2 |  2 |  2|  4 |  2 |
----------------------------------------------------------------------
--------
| Average: |  2.3333 | 5.6667 | |
----------------------------------------------------------------------
--------

GANTT CHART:

---------------------------------
| P1 |   P2   |  P3  |    P1    |
---------------------------------
0   1     4    6     10
Do you want to continue (Y/N): y
1.ROUND ROBIN
2. PRIORITY
3.EXIT
Enter your option: 3
Processes terminated!
```

RESULT:

The required program is written and executed successfully.

LEARNING OUTCOME:

1. Learnt to implement different CPU scheduling policies such as Round robin and Preemptive priority in C language

2. Learnt to print Gantt chart for preemptive CPU scheduling algorithms