2 | Implementation of priority Scheduling Algorithm

Aim:

To schedule snapshot of processed queued according to priority scheduling.

Priority:

- process that has higher priority is processed first.
- priority can be preemptive or non-preemptive.
- when two processes have same priority, FCFS is used to break the tie.
- can result in starvation, since low priority processes may not be processed.

Algorithm:

- Define an array of structure process with members pid, btime, pri, wtime & ttime.
- Get length of the ready queue.
- obtain btime and pri for each process.
- sort the processes according to their pri in ascending order.

* If two process have same pri then FCFS is used to resolve the tie.

- The wtime for first process is 0.
- Compute wtime and ttime for each process as

a. $wtime_{i+1} = wtime_i + btime_i$

b. $ttime_i = wtime_i + btime_i$

- Compute average waiting time avwt and average turn around time atur.
- Display the btime, pri, ttime and wtime for each process.
- Display GANTT chart for the above scheduling
- Display avwt and atur.
- Stop

IMPLEMENTATION OF PRIORITY CPU SCHEDULING ALGORITHM

PROGRAM:

```
#include <stdio.h>
struct process
{
        int pid;
        int btime;
        int pri;
        int wtime;
        int ttime;
} p[10], temp;
int main()
{
        int i,j,k,n,ttur,twat;
        float awat,atur;
        printf("Enter no. of process : ");
        scanf("%d", &n);
        for(i=0; i<n; i++)
        {
                printf("Burst time for process P%d (in ms) : ", (i+1));
                scanf("%d", &p[i].btime);
                printf("Priority for process P%d : ", (i+1));
                scanf("%d", &p[i].pri);
                p[i].pid = i+1;
        }
```

```
        for(i=0; i<n-1; i++)
        {
                for(j=i+1; j<n; j++)
                {
                        if((p[i].pri > p[j].pri) || (p[i].pri == p[j].pri && p[i].pid > p[j].pid))
                        {
                                temp = p[i];
                                p[i] = p[j];
                                p[j] = temp;
                        }
                }
        }
        p[0].wtime = 0;
        for(i=0; i<n; i++)
        {
                p[i+1].wtime = p[i].wtime + p[i].btime;
                p[i].ttime = p[i].wtime + p[i].btime;
        }
        ttur = twat = 0;
        for(i=0; i<n; i++)
        {
                ttur += p[i].ttime;
                twat += p[i].wtime;
        }
        awat = (float)twat
        atur = (float)ttur
        printf("\n\t Priority Scheduling\n\n");
```

```
        for(i=0; i<38; i++)
                printf("-");
        printf("\nProcess B-Time Priority T-Time W-Time\n");
        for(i=0, i<38; i++)
                printf("-");
        for (i=0; i<n; i++)
        printf("\n P%4d\t%4d\t%3d\t%4d\t%4d"
                                        ,p[i].pid,p[i].btime,p[i].pri,p[i].ttime,p[i] wtime);
    printf("\n");
    for(i=0; i<38; i++)
            printf("-");
   printf("\n\nAverage waiting time: %5.2fms", awat);
   printf("\nAverage turn around time : %5.2fms\n", atur);
   printf("\n\nGANTT Chart\n");
  printf("-");
  for(i=0; i<(p[n-1].ttime + 2*n); i++)
          printf("-");
printf("\n|");
for(i=0; i<n; i++)
{
        k = p[i].btime/2;
        for(j=0; j<k; j++)
                printf(" ");
       printf("P%d",p[i].pid);
       for(j=k+1; j<p[i].btime; j++)
               printf(" ");
     printf("|");
```

```
    }
    printf("\n-");
    for(i=0; i<(p[n-1].ttime + 2*n); i++)
        printf("-");
    printf("\n0");
    for(i=0; i<n; i++)
    {
        for(j=0; j<p[i].btime; j++)
            printf(" ");
        printf("%2d",p[i].ttime);
    }
}
```

Result

Thus waiting time and turnaround time for processes based on priority scheduling was computed and the average waiting time was executed.