*FIRST COME FIRST SERVE*

**-on the basis of Arrival Time**

#include <stdio.h>

typedef *struct* Process{

*char* pid[10];

*int* AT, BT, WT, TAT, CT;

} Process;

*void* swap(Process \**a*, Process \**b*){

    Process t = \**a*;

    \**a* = \**b*;

    \**b* = t;

}

*void* sort(Process *pro[]*, *int* *n*){

    for (*int* i = *n* - 1; i >= 1; i--){

        for (*int* j = 0; j < i; j++){

            if (*pro*[j].AT > *pro*[j + 1].AT){

                swap(&*pro*[j], &*pro*[j + 1]);

            }

        }

    }

}

*int* main(){

*int* n;

    printf("Enter no of process: ");

    scanf("%d", &n);

    Process pro[n];

    for (*int* i = 0; i < n; i++){

        printf("Enter pid, Arrival time and Burst time : ");

        scanf("%s", pro[i].pid);

        scanf("%d%d", &pro[i].AT, &pro[i].BT);

        pro[i].WT = 0;

        pro[i].TAT = 0;

    }

    sort(pro, n);

    printf("\n-------------------------------------------------\n");

    printf("|\tPID\t|\tAT\t|\tBT\t|\n");

    for (*int* i = 0; i < n; i++){

        printf("-------------------------------------------------\n");

        printf("|\t%s\t|\t%d\t|\t%d\t|\n", pro[i].pid, pro[i].AT, pro[i].BT);

    }

    printf("-------------------------------------------------\n\n");

*int* total\_TAT = 0, total\_WT = 0, clk = pro[0].AT;

    printf("Process scheduling as follows\n");

    printf("\n-------------------------------------------------------------------------------------------------\n");

    printf("|\tPID\t|\tAT\t|\tBT\t|\tCT\t|\tTAT\t|\tWT\t|\n");

    for (*int* i = 0; i < n; i++){

        clk += pro[i].BT;

        pro[i].CT = clk;

        pro[i].TAT = pro[i].CT - pro[i].AT;

        pro[i].WT = pro[i].TAT - pro[i].BT;

        printf("-------------------------------------------------------------------------------------------------\n");

        printf("|\t%s\t|\t", pro[i].pid);

        printf("%d\t|\t%d\t|\t%d\t|\t%d\t|\t%d\t|\n", pro[i].AT, pro[i].BT, pro[i].CT, pro[i].TAT, pro[i].WT);

        total\_WT += pro[i].WT;

        total\_TAT += pro[i].TAT;

    }

    printf("-------------------------------------------------------------------------------------------------\n\n");

    printf("Average waiting time is : %f\n", (*float*)(total\_WT) / n);

    printf("Average turnaround time is : %f\n", (*float*)(total\_TAT) / n);

}

*/\**

*4*

*p1 5 12*

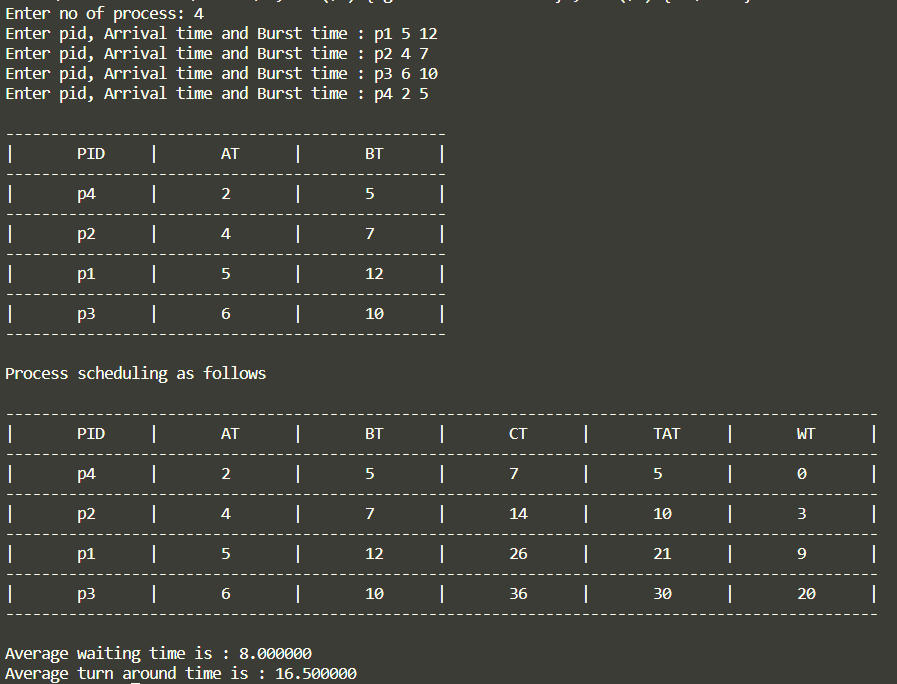
*p2 4 7*

*p3 6 10*

*p4 2 5*

*\*/*

OUTPUT



*ROUND-ROBIN*

**-on the basis of Quanta**

#include <stdio.h>

#include <string.h>

*struct* Process

{

*char* pid[10];

*int* BT;

*int* no\_of\_cycles;

};

*struct* Queue

{

*struct* Process pro[100];

*int* front;

*int* size;

};

*int* isempty(*struct* Queue *q*)

{

    return (*q*.size == 0 ? 1 : 0);

}

*void* enqueue(*struct* Queue \**q*, *struct* Process *p*)

{

    if ((\**q*).size == 100)

    {

        printf("Wait.\n");

        return;

    }

    (\**q*).pro[((\**q*).front + (\**q*).size) % 100] = *p*;

    (\**q*).size++;

*// printf("%d\n",(\*q).size);*

*// printf("Process is in Ready Queue.\n");*

}

*struct* Process dequeue(*struct* Queue \**q*)

{

*struct* Process p;

    if ((\**q*).size == 0)

    {

        printf("Nothing left to execute.\n");

        return p;

    }

    p = (\**q*).pro[(\**q*).front];

    (\**q*).front++;

    (\**q*).size--;

    return p;

}

*int* main()

{

*int* Quanta\_time = 4; *// 4 ms*

*int* n;

    printf("Enter no of process : ");

    scanf("%d", &n);

*struct* Process pro;

*struct* Queue q;

    q.front = 0;

    q.size = 0;

    for (*int* i = 0; i < n; i++)

    {

        printf("Enter PID and its total execution time : ");

        scanf("%s", (pro.pid));

        scanf("%d", &(pro.BT));

        pro.no\_of\_cycles = 0;

        enqueue(&q, pro);

    }

    printf("\n---------------------------------\n");

    printf("|\tPID\t|\tBT\t|\n");

    for (*int* i = 0; i < n; i++)

    {

        printf("---------------------------------\n");

        printf("|\t%s\t|\t%d\t|\n", q.pro[i].pid, q.pro[i].BT);

    }

    printf("---------------------------------\n\n");

    printf("---------ROUND ROBIN Scheduling-----------\n");

    while (!isempty(q))

    {

*struct* Process p = dequeue(&q);

        p.no\_of\_cycles++;

*int* time = (p.BT > Quanta\_time ? Quanta\_time : p.BT);

        printf("(pid : %s) executed for %d ms.\n", p.pid, time);

        if (p.BT > Quanta\_time)

        {

            p.BT -= Quanta\_time;

            enqueue(&q, p);

        }

    }

    printf("------------------------------------------\n\n");

}

*/\**

*4*

*p1 20*

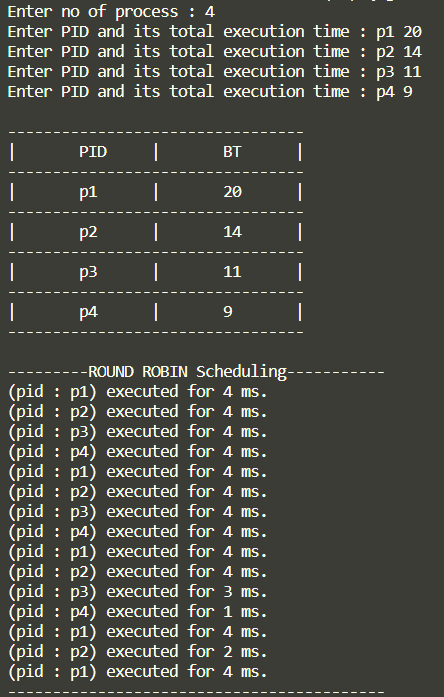
*p2 14*

*p3 11*

*p4 9*

*\*/*

OUTPUT



*MULTI-QUEUE-FEEDBACK*

**-Queues with time quanta 8ms,16ms and 3rd one is FCFS**

#include <stdio.h>

#include <string.h>

*struct* Process{

*char* pid[10];

*int* BT;

*int* no\_of\_cycles;

};

*struct* Queue{

*struct* Process pro[100];

*int* front;

*int* size;

};

*int* isempty(*struct* Queue *q*){

    return (*q*.size == 0 ? 1 : 0);

}

*void* enqueue(*struct* Queue \**q*, *struct* Process *p*){

    if ((\**q*).size == 100){

        printf("Wait.\n");

        return;

    }

    (\**q*).pro[((\**q*).front + (\**q*).size) % 100] = *p*;

    (\**q*).size++;

}

*struct* Process dequeue(*struct* Queue \**q*){

*struct* Process p;

    if ((\**q*).size == 0){

        printf("Nothing left to execute.\n");

        return p;

    }

    p = (\**q*).pro[(\**q*).front];

    (\**q*).front++;

    (\**q*).size--;

    return p;

}

*int* main(){

*int* Quanta\_time1 = 8; *// 8 ms*

*int* Quanta\_time2 = 16; *// 16 ms*

*int* n;

    printf("Enter no of process : ");

    scanf("%d", &n);

*struct* Process pro;

*struct* Queue q8, q16, fcfs;

    q8.front = 0;q8.size = 0;

    q16.front = 0;q16.size = 0;

    fcfs.front = 0;fcfs.size = 0;

    for (*int* i = 0; i < n; i++){

        printf("Enter PID and its total execution time : ");

        scanf("%s", (pro.pid));

        scanf("%d", &(pro.BT));

        pro.no\_of\_cycles = 0;

        enqueue(&q8, pro);

    }

    printf("\n---------------------------------\n");

    printf("|\tPID\t|\tBT\t|\n");

    for (*int* i = 0; i < n; i++){

        printf("---------------------------------\n");

        printf("|\t%s\t|\t%d\t|\n", q8.pro[i].pid, q8.pro[i].BT);

    }

    printf("---------------------------------\n\n");

    printf("---------MULTI - LEVEL - QUEUE - SCHEDULING-----------\n");

    printf("\nQueue having Quanta time = 8 ms \n");

    while (!isempty(q8)){

*struct* Process p = dequeue(&q8);

        p.no\_of\_cycles++;

*int* time = (p.BT > Quanta\_time1 ? Quanta\_time1 : p.BT);

        printf("(pid : %s) executed for %d ms.\n", p.pid, time);

        if (p.BT > Quanta\_time1)

        {

            p.BT -= Quanta\_time1;

            enqueue(&q16, p);

        }

    }

    printf("\nQueue having Quanta time = 16 ms \n");

    while (!isempty(q16)){

*struct* Process p = dequeue(&q16);

        p.no\_of\_cycles++;

*int* time = (p.BT > Quanta\_time2 ? Quanta\_time2 : p.BT);

        printf("(pid : %s) executed for %d ms.\n", p.pid, time);

        if (p.BT > Quanta\_time2)

        {

            p.BT -= Quanta\_time2;

            enqueue(&fcfs, p);

        }

    }

    printf("\nQueue having Quanta time = infinity ms (FCFS) \n");

    while (!isempty(fcfs)){

*struct* Process p = dequeue(&fcfs);

        p.no\_of\_cycles++;

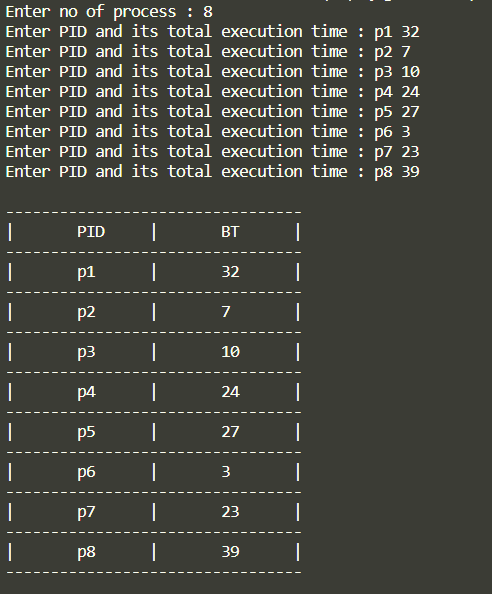
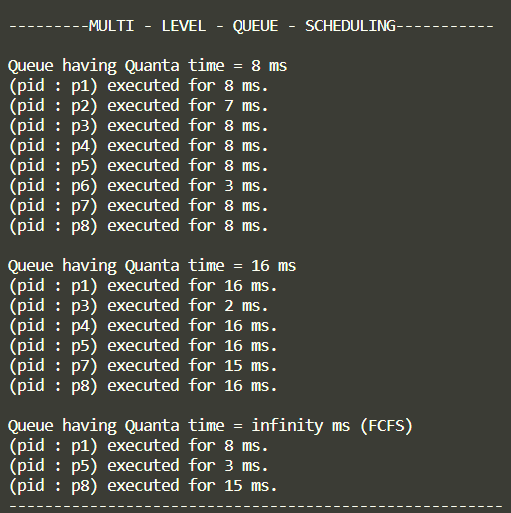
        printf("(pid : %s) executed for %d ms.\n", p.pid, p.BT);

    }

    printf("-------------------------------------------------------");

}

OUTPUT

*SHORTEST-JOB-FIRST*

**-on the basis of burst time**

#include <stdio.h>

typedef *struct* Process

{

*char* pid[10];

*int* BT, WT, TAT, AT, CT;

} Process;

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

{

    Process t = \**a*;

    \**a* = \**b*;

    \**b* = t;

}

*void* sort(Process *pro[]*, *int* *n*)

{

    for (*int* i = *n* - 1; i >= 1; i--)

    {

        for (*int* j = 0; j < i; j++)

        {

            if (*pro*[j].BT > *pro*[j + 1].BT)

                swap(&*pro*[j], &*pro*[j + 1]);

        }

    }

}

*int* main()

{

*int* n;

    printf("Enter no of process: ");

    scanf("%d", &n);

    Process pro[n];

    for (*int* i = 0; i < n; i++)

    {

        printf("Enter pid,Burst time : ");

        scanf("%s", pro[i].pid);

        scanf("%d", &pro[i].BT);

        pro[i].WT = 0;

        pro[i].TAT = 0;

        pro[i].AT = 0;

    }

    sort(pro, n);

    printf("\n---------------------------------\n");

    printf("|\tPID\t|\tBT\t|\n");

    for (*int* i = 0; i < n; i++)

    {

        printf("---------------------------------\n");

        printf("|\t%s\t|\t%d\t|\n", pro[i].pid, pro[i].BT);

    }

    printf("---------------------------------\n\n");

*int* total\_TAT = 0, total\_WT = 0, clk = pro[0].AT;

    printf("Process scheduling as follows\n");

    printf("\n---------------------------------------------------------------------------------\n");

    printf("|\tPID\t|\tBT\t|\tCT\t|\tTAT\t|\tWT\t|\n");

    for (*int* i = 0; i < n; i++)

    {

        clk += pro[i].BT;

        pro[i].CT = clk;

        pro[i].TAT = pro[i].CT - pro[i].AT;

        pro[i].WT = pro[i].TAT - pro[i].BT;

        printf("---------------------------------------------------------------------------------\n");

        printf("|\t%s\t|\t", pro[i].pid);

        printf("%d\t|\t%d\t|\t%d\t|\t%d\t|\n", pro[i].BT, pro[i].CT, pro[i].TAT, pro[i].WT);

        total\_WT += pro[i].WT;

        total\_TAT += pro[i].TAT;

    }

    printf("---------------------------------------------------------------------------------\n\n");

    printf("Average waiting time is : %f\n", (*float*)(total\_WT) / n);

    printf("Average turnaround time is : %f\n", (*float*)(total\_TAT) / n);

}

*/\**

*3*

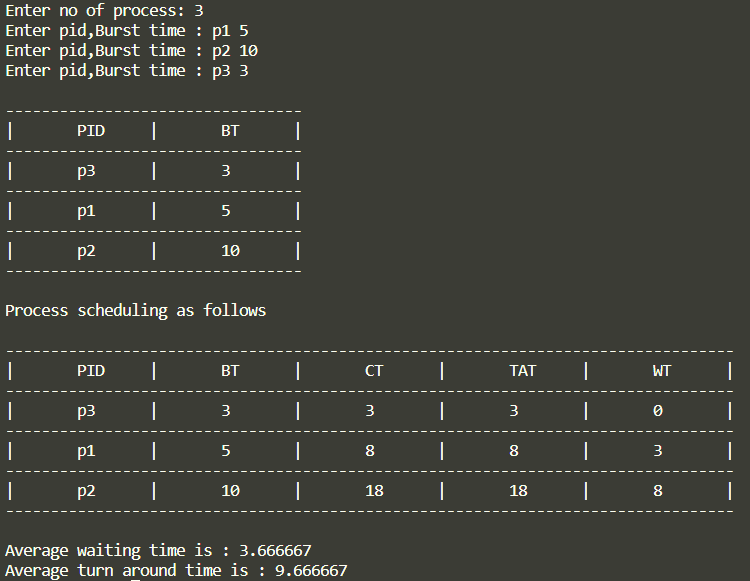
*p1 5*

*p2 10*

*p3 3*

*\*/*

OUTPUT



*SRTF(Pre-emptive SJF)*

**-Pre-empting the processes based on remaining time**

#include <stdio.h>

typedef *struct* Process

{

*char* pid[10];

*int* AT, BT, WT, TAT, REM\_TIME, CT;

} Process;

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

{

    Process t = \**a*;

    \**a* = \**b*;

    \**b* = t;

}

*void* sort(Process *pro[]*, *int* *n*)

{

    for (*int* i = *n* - 1; i >= 1; i--)

    {

        for (*int* j = 0; j < i; j++)

        {

            if (*pro*[j].AT > *pro*[j + 1].AT)

                swap(&*pro*[j], &*pro*[j + 1]);

        }

    }

}

*void* min\_heapify(Process *pro[]*, *int* *i*, *int* *n*)

{

*int* min = *i*;

*int* left = 2 \* *i*;

*int* right = 2 \* *i* + 1;

    if (left <= *n* && *pro*[left].BT < *pro*[min].BT)

        min = left;

    if (right <= *n* && *pro*[right].BT < *pro*[min].BT)

        min = right;

    if (min != *i*)

    {

        swap(&*pro*[min], &*pro*[*i*]);

        min\_heapify(*pro*, min, *n*);

    }

}

*void* build\_min\_heap(Process *pro[]*, *int* *n*)

{

    for (*int* i = *n* / 2; i > 0; i--)

        min\_heapify(*pro*, i, *n*);

}

*void* decrease\_key(Process *pro[]*, *int* *i*, *int* *size*)

{

*int* key = *pro*[*i*].BT;

    while (*i* > 1 && *pro*[*i* / 2].BT > key)

    {

        swap(&*pro*[*i*], &*pro*[*i* / 2]);

*i* /= 2;

    }

}

Process extract\_min(Process *pro[]*, *int* \**size*)

{

    Process temp = *pro*[1];

*pro*[1] = *pro*[\**size*];

    (\**size*)--;

    min\_heapify(*pro*, 1, \**size*);

    return temp;

}

*void* insert(Process *pro[]*, *int* \**size*, Process *newproc*)

{

    (\**size*)++;

*pro*[\**size*] = *newproc*;

    decrease\_key(*pro*, \**size*, \**size*);

}

*int* main()

{

*int* n, m = 0;

    printf("Enter no of process: ");

    scanf("%d", &n);

    Process pro[n + 1], arr[n];

    for (*int* i = 0; i < n; i++)

    {

        printf("Enter pid, Arrival time and Burst time : ");

        scanf("%s", arr[i].pid);

        scanf("%d%d", &arr[i].AT, &arr[i].BT);

        arr[i].WT = 0;

        arr[i].REM\_TIME = arr[i].BT;

        arr[i].TAT = 0;

    }

    sort(arr, n);

    printf("\n-------------------------------------------------\n");

    printf("|\tPID\t|\tAT\t|\tBT\t|\n");

    for (*int* i = 0; i < n; i++)

    {

        printf("-------------------------------------------------\n");

        printf("|\t%s\t|\t", arr[i].pid);

        printf("%d\t|\t%d\t|\n", arr[i].AT, arr[i].BT);

    }

    printf("-------------------------------------------------\n\n");

*int* clk = -1, ind = 0;

    printf("Process scheduling as follows\n");

*// printf("PID\t\tAT\t\tBT\t\tWT\t\tTAT\n");*

    for (*int* i = 0; i < n;)

    {

        insert(pro, &m, arr[i++]);

        if (clk == -1)

        {

            clk = pro[1].AT;

        }

        if (i < n && arr[i].AT < clk + pro[1].REM\_TIME)

        {

            if (arr[i].BT < pro[1].BT - (arr[i].AT - pro[1].AT))

            {

                Process t = extract\_min(pro, &m);

                t.REM\_TIME -= (arr[i].AT - t.AT);

                printf("->( CLOCK=%d ) \t: %s ", clk, t.pid);

                printf("executed for %d ms\n", arr[i].AT - t.AT);

*// printf("%d\t\t%d\t\t%d\n",t.AT,t.BT,t.key);*

                clk += arr[i].AT - t.AT;

                insert(pro, &m, t);

                continue;

            }

        }

        Process t = extract\_min(pro, &m);

        printf("->( CLOCK=%d ) \t: %s ", clk, t.pid);

        printf("executed for %d ms\n", t.REM\_TIME);

        clk += t.REM\_TIME;

*// t.WT = clk;*

        t.CT = clk;

        t.TAT = t.CT - t.AT;

        t.WT = t.TAT - t.BT;

        arr[ind++] = t;

    }

    while (m)

    {

        Process t = extract\_min(pro, &m);

        printf("->( CLOCK=%d ) \t: %s ", clk, t.pid);

        printf("executed for %d ms\n", t.REM\_TIME);

        clk += t.REM\_TIME;

*// t.WT = clk;*

        t.CT = clk;

        t.TAT = t.CT - t.AT;

        t.WT = t.TAT - t.BT;

        arr[ind++] = t;

    }

    printf("------------------------------------------");

    printf("-------------------------------------------------------\n");

*int* total\_TAT = 0, total\_WT = 0;

    printf("|\tPID\t|\tAT\t|\tBT\t|\tCT\t|\tTAT\t|\tWT\t|\n");

    for (*int* i = 0; i < n; i++)

    {

        printf("------------------------------------------");

        printf("-------------------------------------------------------\n");

        printf("|\t%s\t|\t", arr[i].pid);

        printf("%d\t|\t%d\t|", arr[i].AT, arr[i].BT);

        printf("\t%d\t|\t%d\t|\t%d\t|\n", arr[i].CT, arr[i].TAT, arr[i].WT);

        total\_WT += arr[i].WT;

        total\_TAT += arr[i].TAT;

    }

    printf("-----------------------------------------");

    printf("--------------------------------------------------------\n\n");

    printf("Average waiting time is : %f\n", (*float*)(total\_WT) / n);

    printf("Average turnaround time is : %f\n", (*float*)(total\_TAT) / n);

}

*/\**

*4*

*p1 2 10*

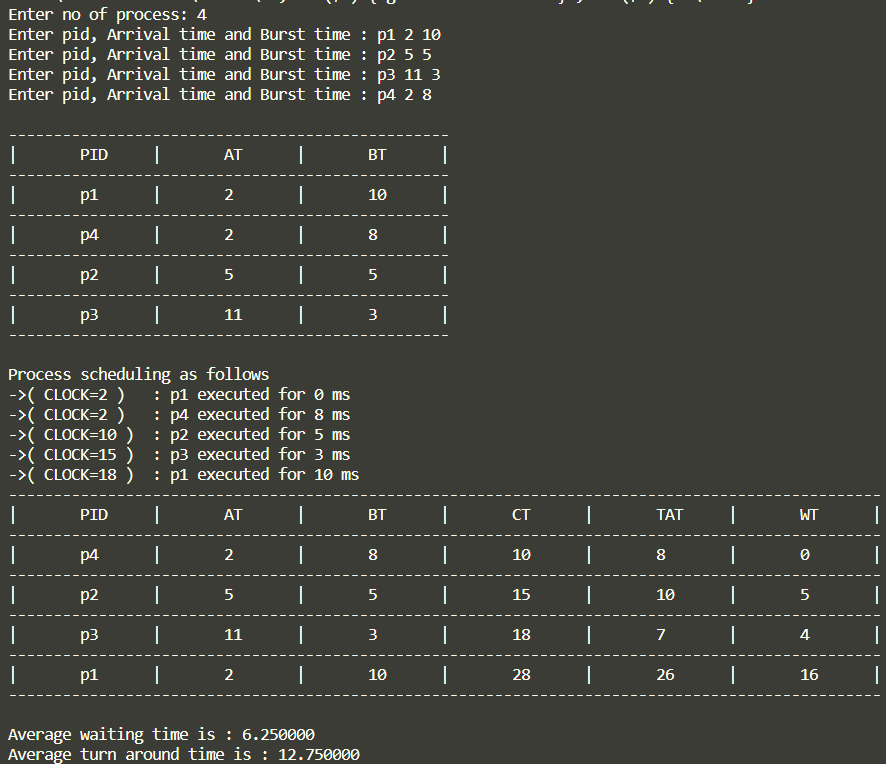
*p2 5 5*

*p3 11 3*

*p4 2 8*

*\*/*

OUTPUT



*NON-Pre-emptive SJF*

**-inserting into ready queue based on arrival time and processing based on burst time**

#include <stdio.h>

typedef *struct* Process

{

*char* pid[10];

*int* AT, BT, WT, TAT, REM\_TIME, CT;

} Process;

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

{

    Process t = \**a*;

    \**a* = \**b*;

    \**b* = t;

}

*void* sort(Process *pro[]*, *int* *n*)

{

    for (*int* i = *n* - 1; i >= 1; i--)

    {

        for (*int* j = 0; j < i; j++)

        {

            if (*pro*[j].AT > *pro*[j + 1].AT)

                swap(&*pro*[j], &*pro*[j + 1]);

        }

    }

}

*void* min\_heapify(Process *pro[]*, *int* *i*, *int* *n*)

{

*int* min = *i*;

*int* left = 2 \* *i*;

*int* right = 2 \* *i* + 1;

    if (left <= *n* && *pro*[left].BT < *pro*[min].BT)

        min = left;

    if (right <= *n* && *pro*[right].BT < *pro*[min].BT)

        min = right;

    if (min != *i*)

    {

        swap(&*pro*[min], &*pro*[*i*]);

        min\_heapify(*pro*, min, *n*);

    }

}

*void* build\_min\_heap(Process *pro[]*, *int* *n*)

{

    for (*int* i = *n* / 2; i > 0; i--)

        min\_heapify(*pro*, i, *n*);

}

*void* decrease\_key(Process *pro[]*, *int* *i*, *int* *size*)

{

*int* key = *pro*[*i*].BT;

    while (*i* > 1 && *pro*[*i* / 2].BT > key)

    {

        swap(&*pro*[*i*], &*pro*[*i* / 2]);

*i* /= 2;

    }

}

Process extract\_min(Process *pro[]*, *int* \**size*)

{

    Process temp = *pro*[1];

*pro*[1] = *pro*[\**size*];

    (\**size*)--;

    min\_heapify(*pro*, 1, \**size*);

    return temp;

}

*void* insert(Process *pro[]*, *int* \**size*, Process *newproc*)

{

    (\**size*)++;

*pro*[\**size*] = *newproc*;

    decrease\_key(*pro*, \**size*, \**size*);

}

*int* main()

{

*int* n, m = 0;

    printf("Enter no of process: ");

    scanf("%d", &n);

    Process pro[n + 1], arr[n];

    for (*int* i = 0; i < n; i++)

    {

        printf("Enter pid, Arrival time and Burst time : ");

        scanf("%s", arr[i].pid);

        scanf("%d%d", &arr[i].AT, &arr[i].BT);

        arr[i].WT = 0;

        arr[i].REM\_TIME = arr[i].BT;

        arr[i].TAT = 0;

    }

    sort(arr, n);

    printf("\n-------------------------------------------------\n");

    printf("|\tPID\t|\tAT\t|\tBT\t|\n");

    for (*int* i = 0; i < n; i++)

    {

        printf("-------------------------------------------------\n");

        printf("|\t%s\t|\t", arr[i].pid);

        printf("%d\t|\t%d\t|\n", arr[i].AT, arr[i].BT);

    }

    printf("-------------------------------------------------\n\n");

*int* clk = 0, ind = 0;

    printf("Process scheduling as follows\n");

    for (*int* i = 0; i < n;)

    {

        if (m == 0)

            insert(pro, &m, arr[i++]);

        if (clk == -1)

        {

            clk = pro[1].AT;

        }

        Process t = extract\_min(pro, &m);

        printf("->( CLOCK=%d ) \t: %s ", clk, t.pid);

        printf("executed for %d ms\n", t.REM\_TIME);

        clk += t.REM\_TIME;

*// t.WT = clk;*

        t.CT = clk;

        t.TAT = t.CT - t.AT;

        t.WT = t.TAT - t.BT;

        arr[ind++] = t;

        while (i < n && arr[i].AT < clk)

        {

            insert(pro, &m, arr[i++]);

        }

    }

    while (m)

    {

        Process t = extract\_min(pro, &m);

        printf("->( CLOCK=%d ) \t: %s ", clk, t.pid);

        printf("executed for %d ms\n", t.REM\_TIME);

        clk += t.REM\_TIME;

*// t.WT = clk;*

        t.CT = clk;

        t.TAT = t.CT - t.AT;

        t.WT = t.TAT - t.BT;

        arr[ind++] = t;

    printf("------------------------------------------");

    printf("-------------------------------------------------------\n");

*int* total\_TAT = 0, total\_WT = 0;

    printf("|\tPID\t|\tAT\t|\tBT\t|\tCT\t|\tTAT\t|\tWT\t|\n");

    for (*int* i = 0; i < n; i++)

    {

        printf("------------------------------------------");

        printf("-------------------------------------------------------\n");

        printf("|\t%s\t|\t", arr[i].pid);

        printf("%d\t|\t%d\t|", arr[i].AT, arr[i].BT);

        printf("\t%d\t|\t%d\t|\t%d\t|\n", arr[i].CT, arr[i].TAT, arr[i].WT);

        total\_WT += arr[i].WT;

        total\_TAT += arr[i].TAT;

    }

    printf("-----------------------------------------");

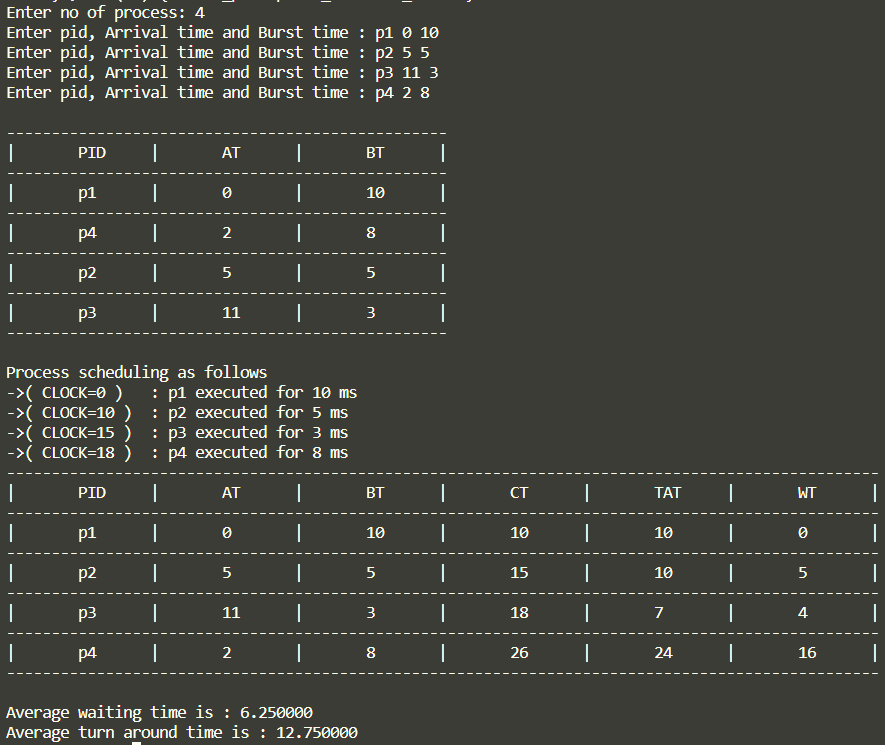
    printf("--------------------------------------------------------\n\n");

    printf("Average waiting time is : %f\n", (*float*)(total\_WT) / n);

    printf("Average turnaround time is : %f\n", (*float*)(total\_TAT) / n);

}

OUTPUT



*NON-Pre-emptive Priority Scheduling*

**-inserting into ready queue based on arrival time and processing based on priority**

#include <stdio.h>

*// key is priority*

*// lower number denotes higher priority*

typedef *struct* Process

{

*char* pid[10];

*int* key, BT, REM\_TIME, AT, WT, TAT, CT;

} Process;

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

{

    Process temp = \**a*;

    \**a* = \**b*;

    \**b* = temp;

}

*void* sort(Process *pro[]*, *int* *n*)

{

    for (*int* i = *n* - 1; i >= 1; i--)

    {

        for (*int* j = 0; j < i; j++)

        {

            if (*pro*[j].AT > *pro*[j + 1].AT)

                swap(&*pro*[j], &*pro*[j + 1]);

        }

    }

}

*void* min\_heapify(Process *pro[]*, *int* *i*, *int* *n*)

{

*int* min = *i*;

*int* left = 2 \* *i*;

*int* right = 2 \* *i* + 1;

    if (left <= *n* && *pro*[left].key < *pro*[min].key)

        min = left;

    if (right <= *n* && *pro*[right].key < *pro*[min].key)

        min = right;

    if (min != *i*)

    {

        swap(&*pro*[min], &*pro*[*i*]);

        min\_heapify(*pro*, min, *n*);

    }

}

*void* build\_min\_heap(Process *pro[]*, *int* *n*)

{

    for (*int* i = *n* / 2; i > 0; i--)

        min\_heapify(*pro*, i, *n*);

}

*void* decrease\_key(Process *pro[]*, *int* *i*, *int* *size*)

{

*int* key = *pro*[*i*].key;

    while (*i* > 1 && *pro*[*i* / 2].key > key)

    {

        swap(&*pro*[*i*], &*pro*[*i* / 2]);

*i* /= 2;

    }

}

Process extract\_min(Process *pro[]*, *int* \**size*)

{

    Process temp = *pro*[1];

*pro*[1] = *pro*[\**size*];

    (\**size*)--;

    min\_heapify(*pro*, 1, \**size*);

    return temp;

}

*void* insert(Process *pro[]*, *int* \**size*, Process *newproc*)

{

    (\**size*)++;

*pro*[\**size*] = *newproc*;

    decrease\_key(*pro*, \**size*, \**size*);

}

*int* main()

{

*int* n, m = 0;

    printf("Enter no of process: ");

    scanf("%d", &n);

    Process pro[n + 1];

    Process arr[n];

    for (*int* i = 0; i < n; i++)

    {

        printf("Enter PID, AT, BT and Priority : ");

        scanf("%s", arr[i].pid);

        scanf("%d %d %d", &arr[i].AT, &arr[i].BT, &arr[i].key);

        arr[i].WT = 0;

        arr[i].REM\_TIME = arr[i].BT;

        arr[i].TAT = 0;

    }

    sort(arr, n);

    printf("\n-----------------------------------------------------------------\n");

    printf("|\tPID\t|\tAT\t|\tBT\t|\tPRI\t|\n");

    for (*int* i = 0; i < n; i++)

    {

        printf("-----------------------------------------------------------------\n");

        printf("|\t%s\t|\t", arr[i].pid);

        printf("%d\t|\t%d\t|\t%d\t|\n", arr[i].AT, arr[i].BT, arr[i].key);

    }

    printf("-----------------------------------------------------------------\n\n");

    printf("Process scheduling as follows\n");

*int* clk = 0, ind = 0;

    for (*int* i = 0; i < n;)

    {

        if (m == 0)

            insert(pro, &m, arr[i++]);

        if (clk == -1)

        {

            clk = pro[1].AT;

        }

        Process t = extract\_min(pro, &m);

        printf("->( CLOCK=%d ) \t: %s ", clk, t.pid);

        printf("executed for %d ms\n", t.REM\_TIME);

        clk += t.REM\_TIME;

*// t.WT = clk;*

        t.CT = clk;

        t.TAT = t.CT - t.AT;

        t.WT = t.TAT - t.BT;

        arr[ind++] = t;

        while (i < n && arr[i].AT < clk)

        {

            insert(pro, &m, arr[i++]);

        }

    }

    while (m)

    {

        Process t = extract\_min(pro, &m);

        printf("->( CLOCK=%d ) \t: %s ", clk, t.pid);

        printf("executed for %d ms\n", t.REM\_TIME);

        clk += t.REM\_TIME;

*// t.WT = clk;*

        t.CT = clk;

        t.TAT = t.CT - t.AT;

        t.WT = t.TAT - t.BT;

        arr[ind++] = t;

    }

    printf("---------------------------------------------------------");

    printf("--------------------------------------------------------\n");

*int* total\_TAT = 0, total\_WT = 0;

    printf("|\tPID\t|\tAT\t|\tBT\t|\tPRI\t|\tCT\t|\tTAT\t|\tWT\t|\n");

    for (*int* i = 0; i < n; i++)

    {

        printf("---------------------------------------------------------");

        printf("--------------------------------------------------------\n");

        printf("|\t%s\t|\t", arr[i].pid);

        printf("%d\t|\t%d\t|\t%d\t|", arr[i].AT, arr[i].BT, arr[i].key);

        printf("\t%d\t|\t%d\t|\t%d\t|\n", arr[i].CT, arr[i].TAT, arr[i].WT);

        total\_WT += arr[i].WT;

        total\_TAT += arr[i].TAT;

    }

    printf("---------------------------------------------------------");

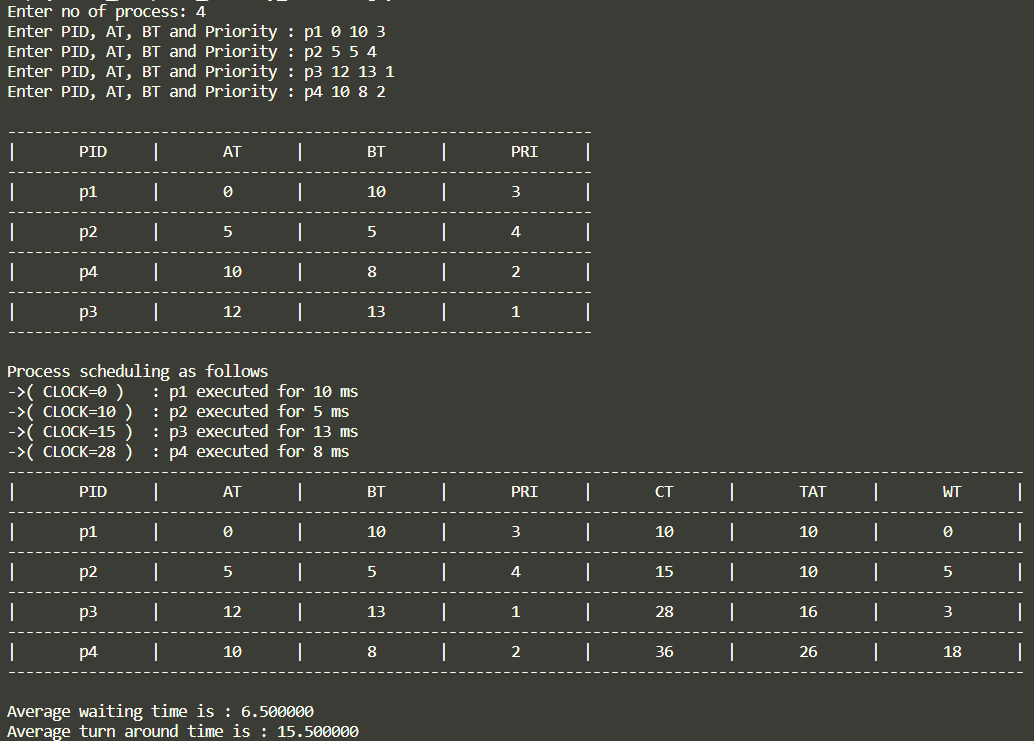
    printf("--------------------------------------------------------\n\n");

    printf("Average waiting time is : %f\n", (*float*)(total\_WT) / n);

    printf("Average turnaround time is : %f\n", (*float*)(total\_TAT) / n);

}

OUTPUT



*Pre-emptive Priority Scheduling*

**-Pre-empting the processes based on priority**

#include <stdio.h>

*// key is priority*

*// lower number denotes higher priority*

typedef *struct* Process

{

*char* pid[10];

*int* key, BT, REM\_TIME, AT, WT, TAT, CT;

} Process;

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

{

    Process temp = \**a*;

    \**a* = \**b*;

    \**b* = temp;

}

*void* sort(Process *pro[]*, *int* *n*)

{

    for (*int* i = *n* - 1; i >= 1; i--)

    {

        for (*int* j = 0; j < i; j++)

        {

            if (*pro*[j].AT > *pro*[j + 1].AT)

                swap(&*pro*[j], &*pro*[j + 1]);

        }

    }

}

*void* min\_heapify(Process *pro[]*, *int* *i*, *int* *n*)

{

*int* min = *i*;

*int* left = 2 \* *i*;

*int* right = 2 \* *i* + 1;

    if (left <= *n* && *pro*[left].key < *pro*[min].key)

        min = left;

    if (right <= *n* && *pro*[right].key < *pro*[min].key)

        min = right;

    if (min != *i*)

    {

        swap(&*pro*[min], &*pro*[*i*]);

        min\_heapify(*pro*, min, *n*);

    }

}

*void* build\_min\_heap(Process *pro[]*, *int* *n*)

{

    for (*int* i = *n* / 2; i > 0; i--)

        min\_heapify(*pro*, i, *n*);

}

*void* decrease\_key(Process *pro[]*, *int* *i*, *int* *size*)

{

*int* key = *pro*[*i*].key;

    while (*i* > 1 && *pro*[*i* / 2].key > key)

    {

        swap(&*pro*[*i*], &*pro*[*i* / 2]);

*i* /= 2;

    }

}

Process extract\_min(Process *pro[]*, *int* \**size*)

{

    Process temp = *pro*[1];

*pro*[1] = *pro*[\**size*];

    (\**size*)--;

    min\_heapify(*pro*, 1, \**size*);

    return temp;

}

*void* insert(Process *pro[]*, *int* \**size*, Process *newproc*)

{

    (\**size*)++;

*pro*[\**size*] = *newproc*;

    decrease\_key(*pro*, \**size*, \**size*);

}

*int* main()

{

*int* n, m = 0;

    printf("Enter no of process: ");

    scanf("%d", &n);

    Process pro[n + 1];

    Process arr[n];

    for (*int* i = 0; i < n; i++)

    {

        printf("Enter PID, AT, BT and Priority : ");

        scanf("%s", arr[i].pid);

        scanf("%d %d %d", &arr[i].AT, &arr[i].BT, &arr[i].key);

        arr[i].WT = 0;

        arr[i].REM\_TIME = arr[i].BT;

        arr[i].TAT = 0;

    }

    sort(arr, n);

    printf("\n-----------------------------------------------------------------\n");

    printf("|\tPID\t|\tAT\t|\tBT\t|\tPRI\t|\n");

    for (*int* i = 0; i < n; i++)

    {

        printf("-----------------------------------------------------------------\n");

        printf("|\t%s\t|\t", arr[i].pid);

        printf("%d\t|\t%d\t|\t%d\t|\n", arr[i].AT, arr[i].BT, arr[i].key);

    }

    printf("-----------------------------------------------------------------\n\n");

    printf("Process scheduling as follows\n");

*int* clk = 0, ind = 0;

    for (*int* i = 0; i < n;)

    {

        insert(pro, &m, arr[i++]);

        if (clk == -1)

        {

            clk = pro[1].AT;

        }

        if (i < n && arr[i].AT < clk + pro[1].REM\_TIME)

        {

            if (arr[i].key < pro[1].key)

            {

                Process t = extract\_min(pro, &m);

                t.REM\_TIME -= (arr[i].AT - t.AT);

                printf("->( CLOCK=%d ) \t: %s ", clk, t.pid);

                printf("executed for %d ms\n", arr[i].AT - t.AT);

*// printf("%d\t\t%d\t\t%d\n",t.AT,t.BT,t.key);*

                clk += arr[i].AT - t.AT;

                insert(pro, &m, t);

            }

        }

        else

        {

            Process t = extract\_min(pro, &m);

            printf("->( CLOCK=%d ) \t: %s ", clk, t.pid);

            printf("executed for %d ms\n", t.REM\_TIME);

            clk += t.REM\_TIME;

*// t.WT = clk;*

            t.CT = clk;

            t.TAT = t.CT - t.AT;

            t.WT = t.TAT - t.BT;

            arr[ind++] = t;

        }

*// insert(pro, &m, arr[i]);*

    }

    while (m)

    {

        Process t = extract\_min(pro, &m);

        printf("->( CLOCK=%d ) \t: %s ", clk, t.pid);

        printf("executed for %d ms\n", t.REM\_TIME);

        clk += t.REM\_TIME;

*// t.WT = clk;*

        t.CT = clk;

        t.TAT = t.CT - t.AT;

        t.WT = t.TAT - t.BT;

        arr[ind++] = t;

    }

    printf("---------------------------------------------------------");

    printf("--------------------------------------------------------\n");

*int* total\_TAT = 0, total\_WT = 0;

    printf("|\tPID\t|\tAT\t|\tBT\t|\tPRI\t|\tCT\t|\tTAT\t|\tWT\t|\n");

    for (*int* i = 0; i < n; i++)

    {

        printf("---------------------------------------------------------");

        printf("--------------------------------------------------------\n");

        printf("|\t%s\t|\t", arr[i].pid);

        printf("%d\t|\t%d\t|\t%d\t|", arr[i].AT, arr[i].BT, arr[i].key);

        printf("\t%d\t|\t%d\t|\t%d\t|\n", arr[i].CT, arr[i].TAT, arr[i].WT);

        total\_WT += arr[i].WT;

        total\_TAT += arr[i].TAT;

    }

    printf("---------------------------------------------------------");

    printf("--------------------------------------------------------\n\n");

    printf("Average waiting time is : %f\n", (*float*)(total\_WT) / n);

    printf("Average turnaround time is : %f\n", (*float*)(total\_TAT) / n);

}

*/\**

*4*

*p1 0 10 3*

*p2 5 5 4*

*p3 4 13 1*

*p4 2 8 2*

*\*/*

OUTPUT

