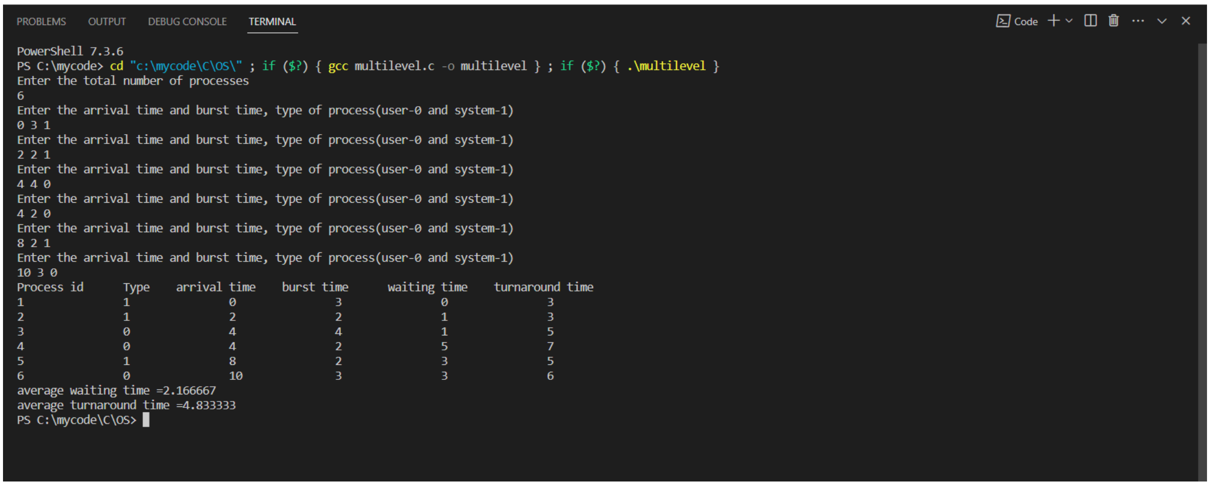
1) MultiLevel Queue Scheduling:

```c
#include<stdio.h>
void swap(int *a,int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
void main()
{
    int n,pid[10],burst[10],type[10],arr[10],wt[10],ta[10],ct[10],i,j;
    float avgwt=0,avgta=0;
    int sum = 0;
    printf("Enter the total number of processes\n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter the process id, type of process(user-0 and system-1), arrival time and burst time\n");
        scanf("%d",&pid[i]);
        scanf("%d",&type[i]);
        scanf("%d",&arr[i]);
        scanf("%d",&burst[i]);
    }
    //sorting the processes according to arrival time
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(arr[j]>arr[j+1])
            {
                swap(&arr[j],&arr[j+1]);
                swap(&pid[j],&pid[j+1]);
                swap(&burst[j],&burst[j+1]);
                swap(&type[j],&type[j+1]);

            }
        }
    }
    //assuming only two process can have same arrival time and different priority
    for(i=0;i<n-1;i++)
    {
        for(j=0;j<n-i-1;j++)
        {
            if(arr[j]==arr[j+1] && type[j]<type[j+1])
            {
                swap(&arr[j],&arr[j+1]);
                swap(&pid[j],&pid[j+1]);
                swap(&burst[j],&burst[j+1]);
                swap(&type[j],&type[j+1]);
            }
        }
    }
    //calculating completion time, arrival time and waiting time
    sum = sum + arr[0];
    for(i = 0;i<n;i++){
        sum = sum + burst[i];
        ct[i] = sum;
        ta[i] = ct[i] - arr[i];
        wt[i] = ta[i] - burst[i];
        if(sum<arr[i+1]){
            int t = arr[i+1]-sum;
            sum = sum+t;
        }
    }

    printf("Process id\tType\tarrival time\tburst time\twaiting time\tturnaround time\n");
    for(i=0;i<n;i++)
    {
        avgta+=ta[i];
        avgwt+=wt[i];
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",pid[i],type[i],arr[i],burst[i],wt[i],ta[i]);
    }
    printf("average waiting time =%f\n",avgwt/n);
    printf("average turnaround time =%f",avgta/n);

}
```

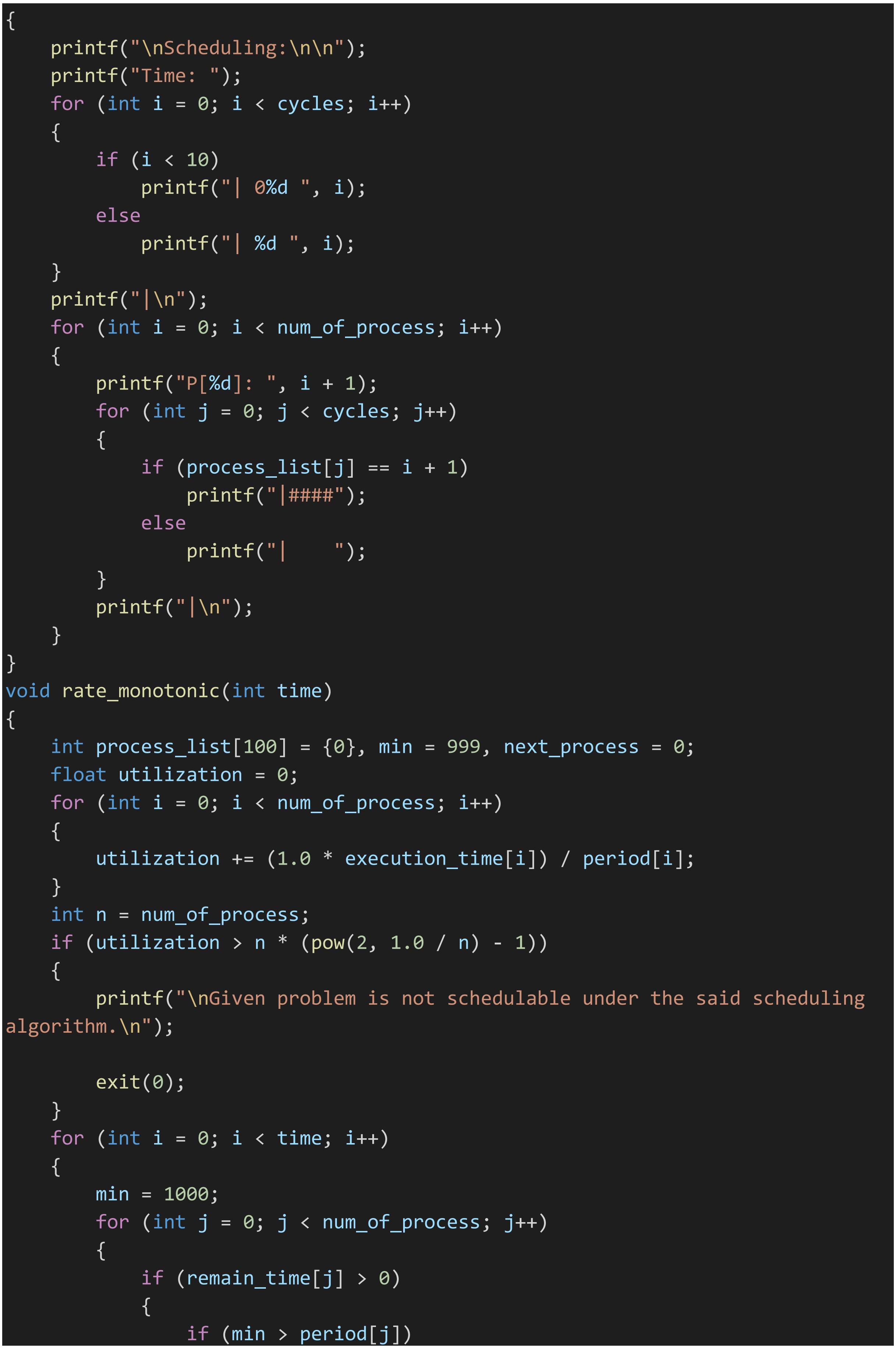
Output:



```
PowerShell 7.3.6
PS C:\mycode> cd "c:\mycode\C\OS\" ; if ($?) { gcc multilevel.c -o multilevel } ; if ($?) { .\multilevel }
Enter the total number of processes
6
Enter the arrival time and burst time, type of process(user-0 and system-1)
0 3 1
Enter the arrival time and burst time, type of process(user-0 and system-1)
2 2 1
Enter the arrival time and burst time, type of process(user-0 and system-1)
4 4 0
Enter the arrival time and burst time, type of process(user-0 and system-1)
4 2 0
Enter the arrival time and burst time, type of process(user-0 and system-1)
8 2 1
Enter the arrival time and burst time, type of process(user-0 and system-1)
10 3 0
Process id      Type    arrival time    burst time      waiting time    turnaround time
1               1               0               3               0               3
2               1               2               2               1               3
3               0               4               4               1               5
4               0               4               2               5               7
5               1               8               2               3               5
6               0               10              3               3               6
average waiting time =2.166667
average turnaround time =4.833333
PS C:\mycode\C\OS> 
```

2) Rate Monotonic Scheduling:

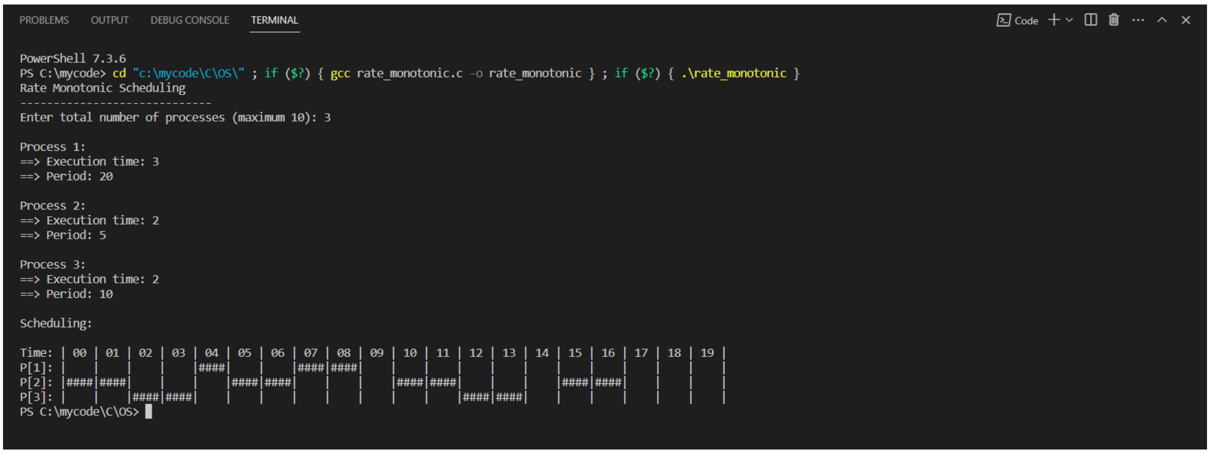
```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#define MAX_PROCESS 10
int num_of_process = 3, count, remain, time_quantum;
int execution_time[MAX_PROCESS], period[MAX_PROCESS],
remain_time[MAX_PROCESS], deadline[MAX_PROCESS], remain_deadline[MAX_PROCESS];
int burst_time[MAX_PROCESS], wait_time[MAX_PROCESS],
completion_time[MAX_PROCESS], arrival_time[MAX_PROCESS];
// collecting details of processes
void get_process_info()
{
    printf("Enter total number of processes (maximum %d): ", MAX_PROCESS);
    scanf("%d", &num_of_process);
    if (num_of_process <= 1)
    {
        printf("Insufficient processes.", num_of_process);
        exit(0);
    }
    for (int i = 0; i < num_of_process; i++)
    {
        printf("\nProcess %d:\n", i + 1);
        printf("Enter the Execution time and Period:\n");
        scanf("%d %d", &execution_time[i], &period[i]);
        remain_time[i] = execution_time[i];
    }
}
// get maximum of three numbers
int max(int a, int b, int c)
{
    int max;
    if (a >= b && a >= c)
        max = a;
    else if (b >= a && b >= c)
        max = b;
    else if (c >= a && c >= b)
        max = c;
    return max;
}
// calculating the observation time for scheduling timeline
int get_observation_time()
{
    return max(period[0], period[1], period[2]);
}
// print scheduling sequence
void print_schedule(int process_list[], int cycles)
```



```
{
    printf("\nScheduling:\n\n");
    printf("Time: ");
    for (int i = 0; i < cycles; i++)
    {
        if (i < 10)
            printf("| 0%d ", i);
        else
            printf("| %d ", i);
    }
    printf("|\n");
    for (int i = 0; i < num_of_process; i++)
    {
        printf("P[%d]: ", i + 1);
        for (int j = 0; j < cycles; j++)
        {
            if (process_list[j] == i + 1)
                printf("|####");
            else
                printf("|    ");
        }
        printf("|\n");
    }
}
void rate_monotonic(int time)
{
    int process_list[100] = {0}, min = 999, next_process = 0;
    float utilization = 0;
    for (int i = 0; i < num_of_process; i++)
    {
        utilization += (1.0 * execution_time[i]) / period[i];
    }
    int n = num_of_process;
    if (utilization > n * (pow(2, 1.0 / n) - 1))
    {
        printf("\nGiven problem is not schedulable under the said scheduling
algorithm.\n");

        exit(0);
    }
    for (int i = 0; i < time; i++)
    {
        min = 1000;
        for (int j = 0; j < num_of_process; j++)
        {
            if (remain_time[j] > 0)
            {
                if (min > period[j])
```

```c
                {
                    min = period[j];
                    next_process = j;
                }
            }
        }
        if (remain_time[next_process] > 0)
        {
            process_list[i] = next_process + 1; // +1 for catering 0 array
index.
            remain_time[next_process] -= 1;
        }
        for (int k = 0; k < num_of_process; k++)
        {
            if ((i + 1) % period[k] == 0)
            {
                remain_time[k] = execution_time[k];
                next_process = k;
            }
        }
    }
    print_schedule(process_list, time);
}
int main(int argc, char *argv[])
{
    int option = 0 , observation_time;
    printf("Rate Monotonic Scheduling\n");
    printf("-----------------------------\n");
    get_process_info(); // collecting processes detail
    observation_time = get_observation_time();
    rate_monotonic(observation_time);
    return 0;
}
```

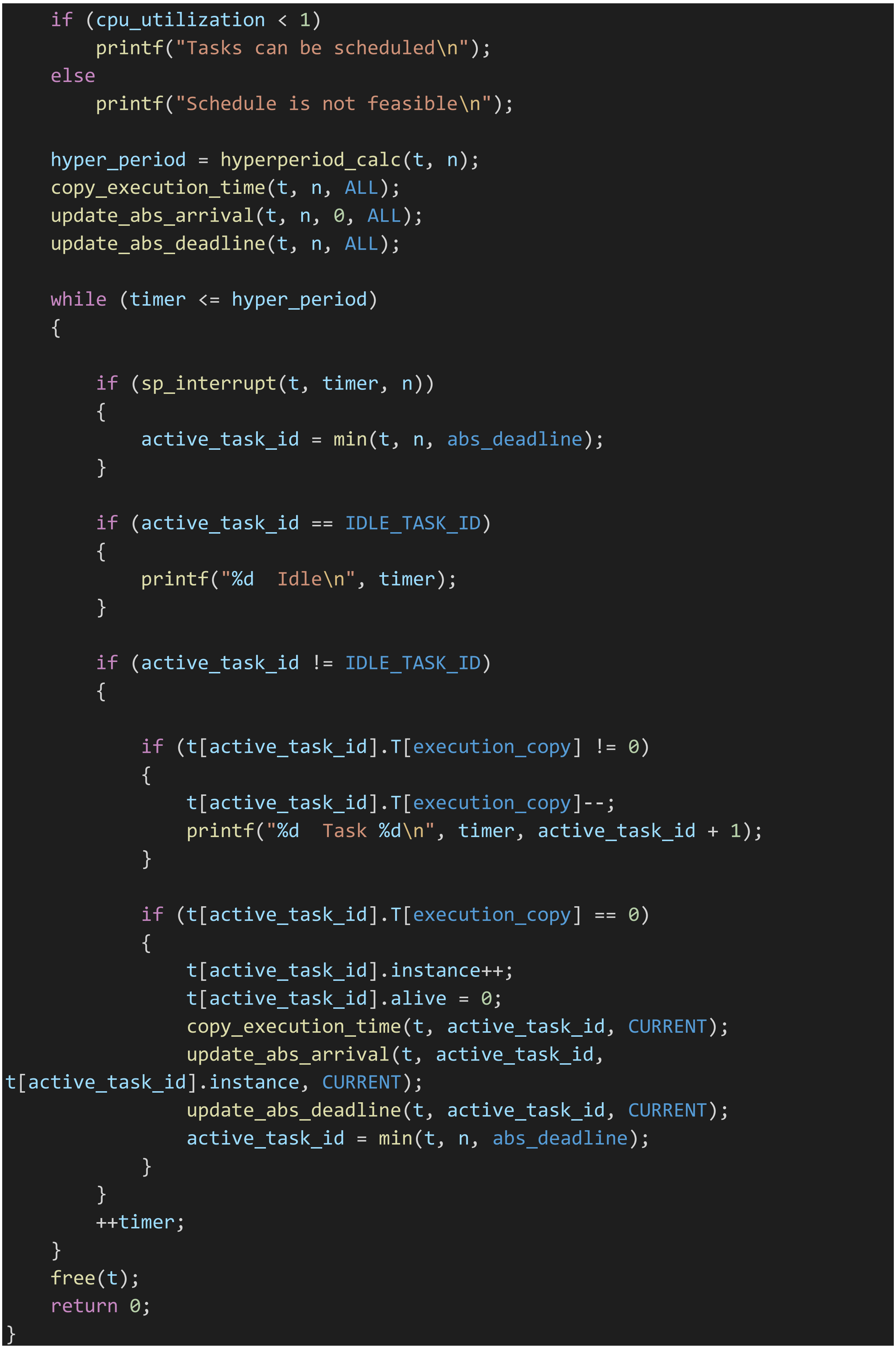
Output:



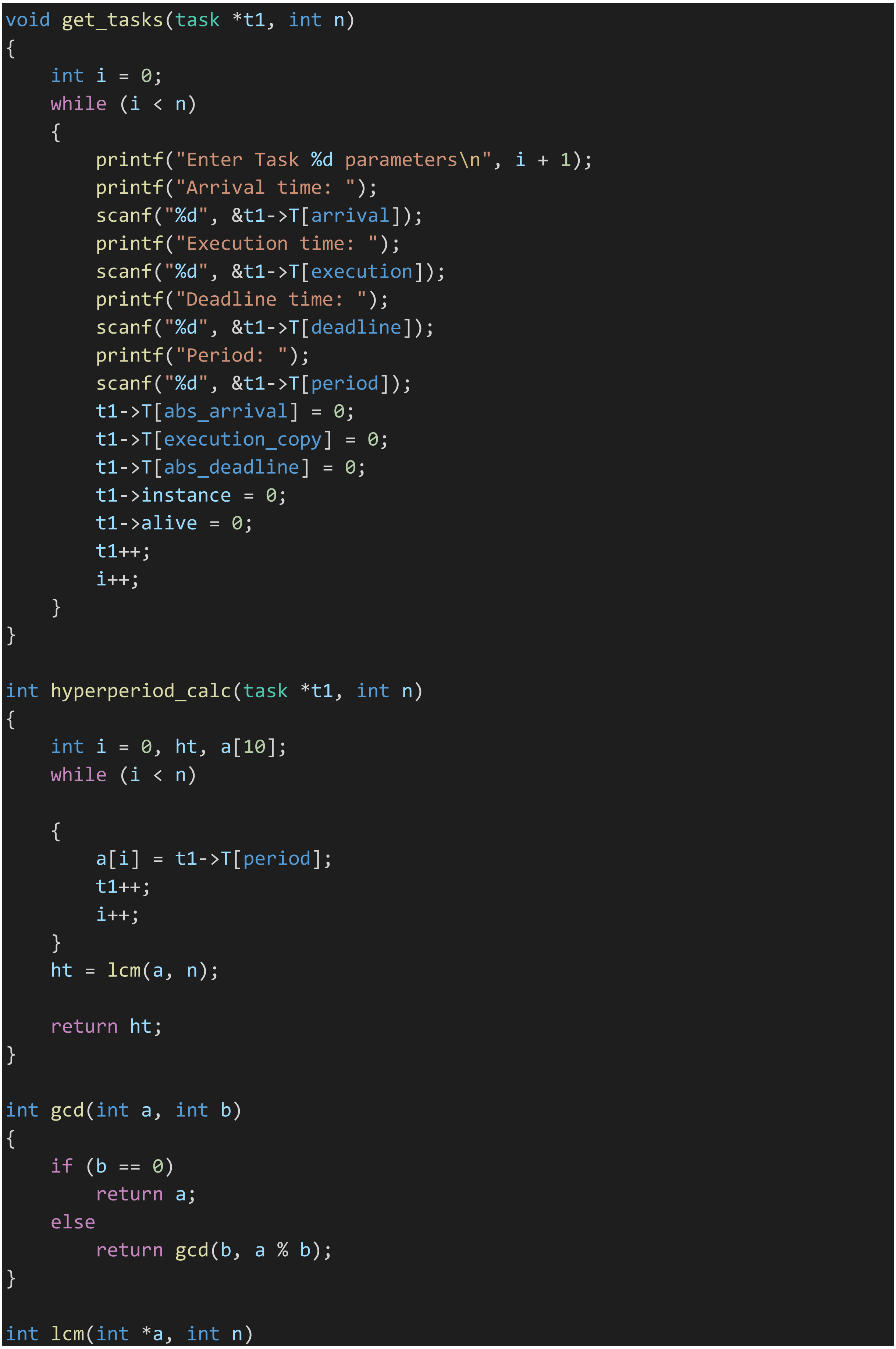
```
PowerShell 7.3.6
PS C:\mycode> cd "c:\mycode\C\OS\" ; if ($?) { gcc rate_monotonic.c -o rate_monotonic } ; if ($?) { .\rate_monotonic }
Rate Monotonic Scheduling
-----------------------------
Enter total number of processes (maximum 10): 3

Process 1:
==> Execution time: 3
==> Period: 20

Process 2:
==> Execution time: 2
==> Period: 5

Process 3:
==> Execution time: 2
==> Period: 10

Scheduling:

Time: | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
P[1]: |    |    |    |    |####|    |    |####|####|    |    |    |    |    |    |    |    |    |    |    |
P[2]: |####|####|    |    |    |####|####|    |    |    |####|####|    |    |    |####|####|    |    |    |
P[3]: |    |    |####|####|    |    |    |    |    |    |    |    |####|####|    |    |    |    |    |    |
PS C:\mycode\C\OS> 
```

3) Earliest Deadline First:

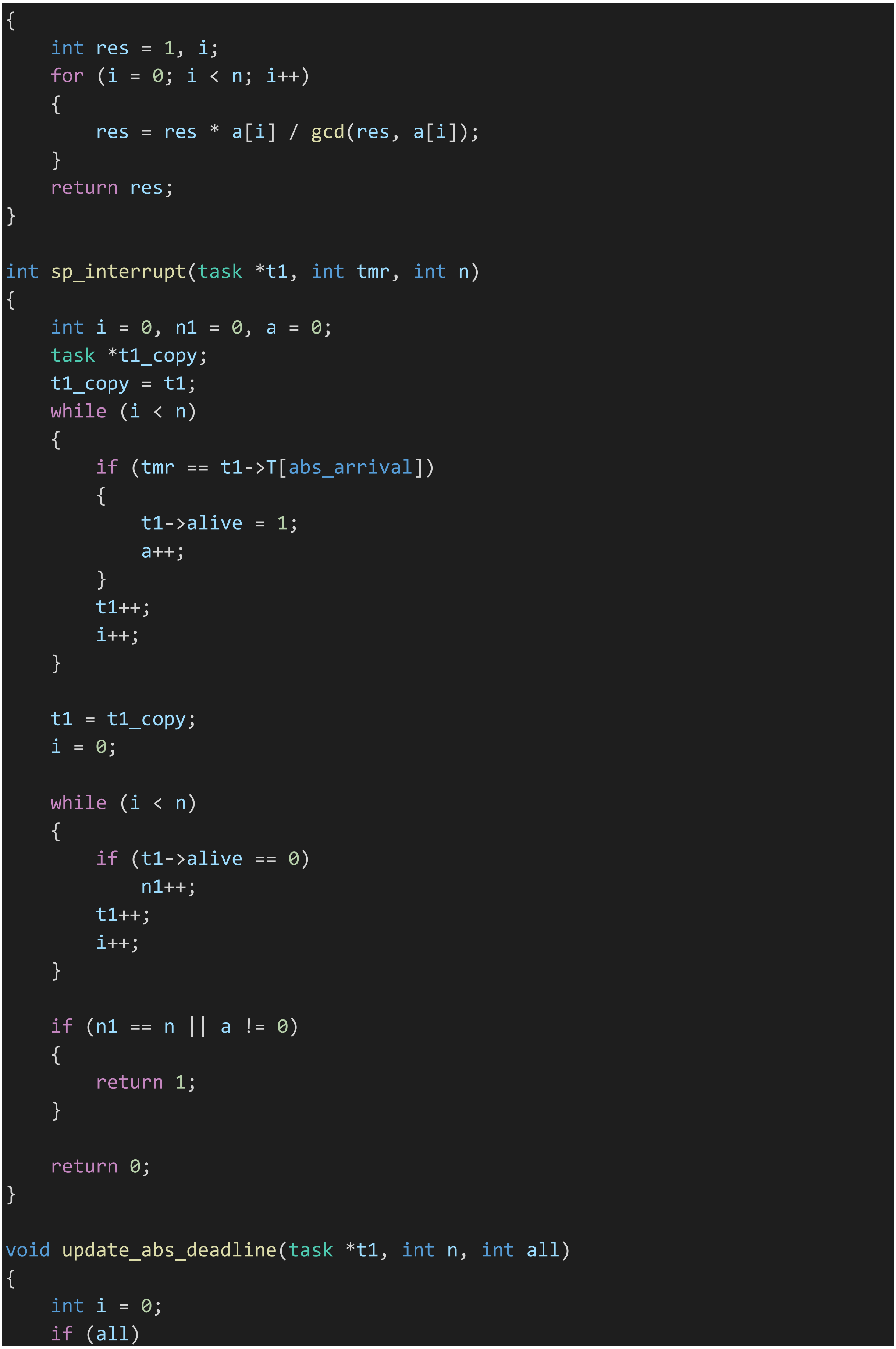
```
#include <stdio.h>
#include<stdlib.h>
#define arrival         0
#define execution       1
#define deadline        2
#define period          3
#define abs_arrival     4
#define execution_copy  5
#define abs_deadline    6


typedef struct
{
    int T[7],instance,alive;

}task;

#define IDLE_TASK_ID 1023
#define ALL 1
#define CURRENT 0

void get_tasks(task *t1,int n);
int hyperperiod_calc(task *t1,int n);
float cpu_util(task *t1,int n);
int gcd(int a, int b);
int lcm(int *a, int n);
int sp_interrupt(task *t1,int tmr,int n);
int min(task *t1,int n,int p);
void update_abs_arrival(task *t1,int n,int k,int all);
void update_abs_deadline(task *t1,int n,int all);
void copy_execution_time(task *t1,int n,int all);


int timer = 0;

int main()
{
    task *t;
    int n, hyper_period, active_task_id;
    float cpu_utilization;
    printf("Enter number of tasks\n");
    scanf("%d", &n);
    t = malloc(n * sizeof(task));
    get_tasks(t, n);
    cpu_utilization = cpu_util(t, n);
    printf("CPU Utilization %f\n", cpu_utilization);
```



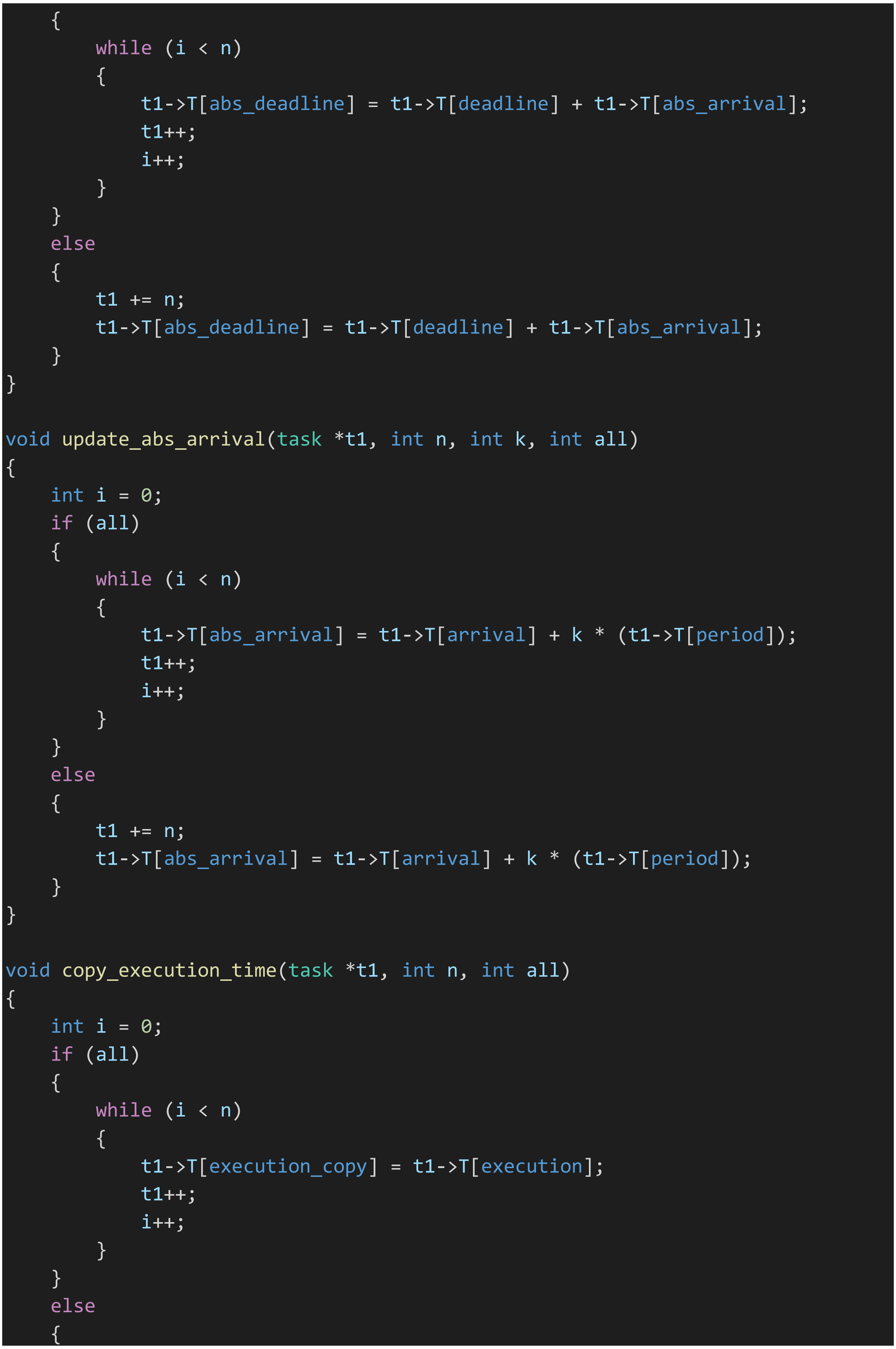
```c
    if (cpu_utilization < 1)
        printf("Tasks can be scheduled\n");
    else
        printf("Schedule is not feasible\n");

    hyper_period = hyperperiod_calc(t, n);
    copy_execution_time(t, n, ALL);
    update_abs_arrival(t, n, 0, ALL);
    update_abs_deadline(t, n, ALL);

    while (timer <= hyper_period)
    {

        if (sp_interrupt(t, timer, n))
        {
            active_task_id = min(t, n, abs_deadline);
        }

        if (active_task_id == IDLE_TASK_ID)
        {
            printf("%d  Idle\n", timer);
        }

        if (active_task_id != IDLE_TASK_ID)
        {

            if (t[active_task_id].T[execution_copy] != 0)
            {
                t[active_task_id].T[execution_copy]--;
                printf("%d  Task %d\n", timer, active_task_id + 1);
            }

            if (t[active_task_id].T[execution_copy] == 0)
            {
                t[active_task_id].instance++;
                t[active_task_id].alive = 0;
                copy_execution_time(t, active_task_id, CURRENT);
                update_abs_arrival(t, active_task_id,
t[active_task_id].instance, CURRENT);
                update_abs_deadline(t, active_task_id, CURRENT);
                active_task_id = min(t, n, abs_deadline);
            }
        }
        ++timer;
    }
    free(t);
    return 0;
}
```



```
void get_tasks(task *t1, int n)
{
    int i = 0;
    while (i < n)
    {
        printf("Enter Task %d parameters\n", i + 1);
        printf("Arrival time: ");
        scanf("%d", &t1->T[arrival]);
        printf("Execution time: ");
        scanf("%d", &t1->T[execution]);
        printf("Deadline time: ");
        scanf("%d", &t1->T[deadline]);
        printf("Period: ");
        scanf("%d", &t1->T[period]);
        t1->T[abs_arrival] = 0;
        t1->T[execution_copy] = 0;
        t1->T[abs_deadline] = 0;
        t1->instance = 0;
        t1->alive = 0;
        t1++;
        i++;
    }
}

int hyperperiod_calc(task *t1, int n)
{
    int i = 0, ht, a[10];
    while (i < n)

    {
        a[i] = t1->T[period];
        t1++;
        i++;
    }
    ht = lcm(a, n);

    return ht;
}

int gcd(int a, int b)
{
    if (b == 0)
        return a;
    else
        return gcd(b, a % b);
}

int lcm(int *a, int n)
```

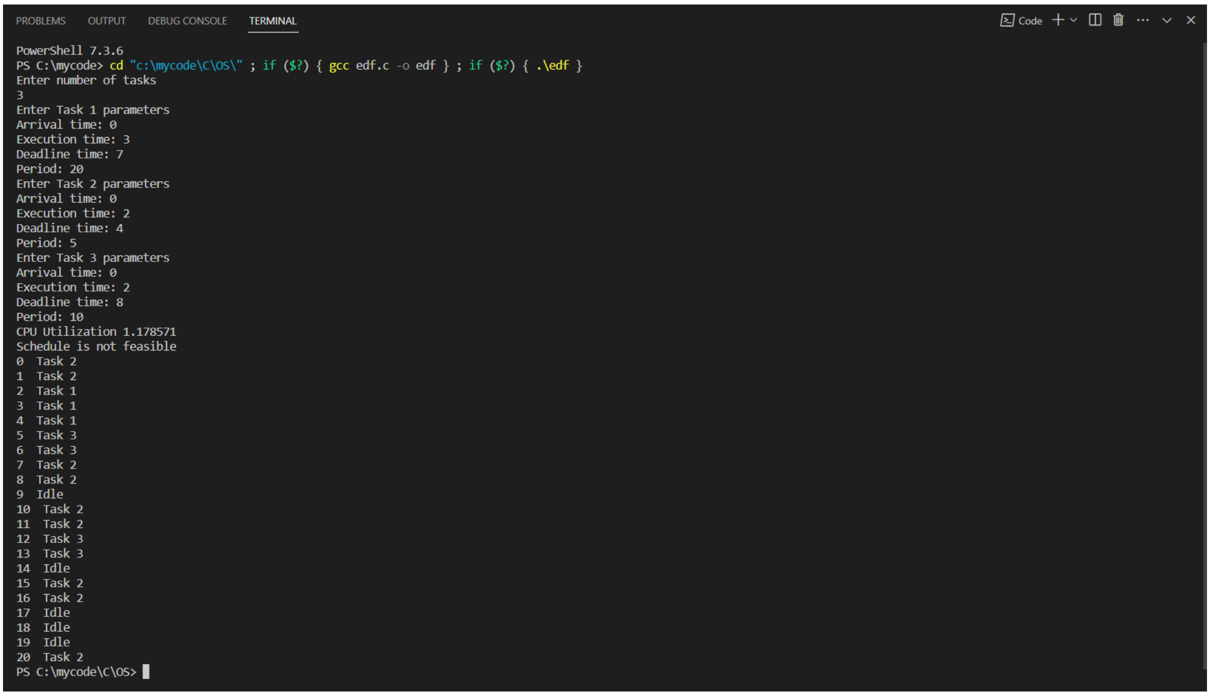


```
{
    int res = 1, i;
    for (i = 0; i < n; i++)
    {
        res = res * a[i] / gcd(res, a[i]);
    }
    return res;
}

int sp_interrupt(task *t1, int tmr, int n)
{
    int i = 0, n1 = 0, a = 0;
    task *t1_copy;
    t1_copy = t1;
    while (i < n)
    {
        if (tmr == t1->T[abs_arrival])
        {
            t1->alive = 1;
            a++;
        }
        t1++;
        i++;
    }

    t1 = t1_copy;
    i = 0;

    while (i < n)
    {
        if (t1->alive == 0)
            n1++;
        t1++;
        i++;
    }

    if (n1 == n || a != 0)
    {
        return 1;
    }

    return 0;
}

void update_abs_deadline(task *t1, int n, int all)
{
    int i = 0;
    if (all)
```



```
    {
        while (i < n)
        {
            t1->T[abs_deadline] = t1->T[deadline] + t1->T[abs_arrival];
            t1++;
            i++;
        }
    }
    else
    {
        t1 += n;
        t1->T[abs_deadline] = t1->T[deadline] + t1->T[abs_arrival];
    }
}

void update_abs_arrival(task *t1, int n, int k, int all)
{
    int i = 0;
    if (all)
    {
        while (i < n)
        {
            t1->T[abs_arrival] = t1->T[arrival] + k * (t1->T[period]);
            t1++;
            i++;
        }
    }
    else
    {
        t1 += n;
        t1->T[abs_arrival] = t1->T[arrival] + k * (t1->T[period]);
    }
}

void copy_execution_time(task *t1, int n, int all)
{
    int i = 0;
    if (all)
    {
        while (i < n)
        {
            t1->T[execution_copy] = t1->T[execution];
            t1++;
            i++;
        }
    }
    else
    {
        t1 += n;
        t1->T[execution_copy] = t1->T[execution];
    }
}

int min(task *t1, int n, int p)
{
    int i = 0, min = 0x7FFF, task_id = IDLE_TASK_ID;
    while (i < n)
    {
        if (min > t1->T[p] && t1->alive == 1)
        {
            min = t1->T[p];
            task_id = i;
        }
        t1++;
        i++;
    }
    return task_id;
}

float cpu_util(task *t1, int n)
{
    int i = 0;
    float cu = 0;
    while (i < n)
    {
        cu = cu + (float)t1->T[execution] / (float)t1->T[deadline];
        t1++;
        i++;
    }
    return cu;
}
```

Output:



```
PowerShell 7.3.6
PS C:\mycode> cd "c:\mycode\C\OS\" ; if ($?) { gcc edf.c -o edf } ; if ($?) { .\edf }
Enter number of tasks
3
Enter Task 1 parameters
Arrival time: 0
Execution time: 3
Deadline time: 7
Period: 20
Enter Task 2 parameters
Arrival time: 0
Execution time: 2
Deadline time: 4
Period: 5
Enter Task 3 parameters
Arrival time: 0
Execution time: 2
Deadline time: 8
Period: 10
CPU Utilization 1.178571
Schedule is not feasible
0  Task 2
1  Task 2
2  Task 1
3  Task 1
4  Task 1
5  Task 3
6  Task 3
7  Task 2
8  Task 2
9  Idle
10  Task 2
11  Task 2
12  Task 3
13  Task 3
14  Idle
15  Task 2
16  Task 2
17  Idle
18  Idle
19  Idle
20  Task 2
PS C:\mycode\C\OS> 
```