NAME – **SUKHAD SHARMA**

SECTION – **K21FG**

REGISTRATION NUMBER – **12110106**

ROLL NUMBER – **RK21FGA12**

GITHUB LINK TO PROJECT –

https://github.com/SCORLEOs773/CSE-316-CA2/blob/main/code.cpp

CSE 316

CA-2

CODE

```c
#include <stdio.h>

#define MAX_PROCESSES 100

struct process
{
    int pid;
    int priority;
    int remainingTime;
};

int main()
{

    struct process queue1[MAX_PROCESSES];
    struct process queue2[MAX_PROCESSES];
    int queue1Size = 0;
    int queue2Size = 0;

    int numProcesses;
    printf("Enter the number of processes you want to 
execute - ");
    scanf("%d", &numProcesses);

    for (int i = 0; i < numProcesses; i++)
    {
        struct process p;
        printf("Enter the Process ID, Priority Value, and
Remaining Time for the process %d -  ", i + 1);
        scanf("%d %d %d", &p.pid, &p.priority,
&p.remainingTime);
        if (p.priority == 0)
        {
            queue1[queue1Size++] = p;
        }
        else
        {
            queue2[queue2Size++] = p;
        }
    }

    int time = 0;
    while (queue1Size > 0 || queue2Size > 0)
    {
        if (queue1Size > 0)
        {
            int highest_priority = 0;
            int highest_priority_index = -1;
            for (int i = 0; i < queue1Size; i++)
            {
                if (queue1[i].priority > highest_priority)
                {
                    highest_priority = queue1[i].priority;
                    highest_priority_index = i;
                }
            }

            struct process p =
queue1[highest_priority_index];

            printf("Time %d: Running process %d\n", time,
p.pid);
            p.remainingTime -= 2;
            time += 2;
            if (p.remainingTime == 0)
            {
                for (int i = highest_priority_index; i <
queue1Size - 1; i++)
                {
                    queue1[i] = queue1[i + 1];
                }
                queue1Size--;

                printf("Time %d: Process %d finished\n",
time, p.pid);
            }
            else
            {
                queue2[queue2Size++] = p;
                for (int i = highest_priority_index; i <
queue1Size - 1; i++)
                {
                    queue1[i] = queue1[i + 1];
                }
                queue1Size--;
            }
        }

        if (queue2Size > 0)
        {
            int shortest_time = 0;
            int shortest_time_index = -1;
            for (int i = 0; i < queue2Size; i++)
            {
                if (shortest_time_index == -1 ||
queue2[i].remainingTime < shortest_time)
                {
                    shortest_time = queue2[i].remainingTime;
                    shortest_time_index = i;
                }
            }
```

```c
            struct process p = queue2[shortest_time_index];

            printf("Time %d: Running process %d\n", time,
p.pid);
            time += p.remainingTime;
            p.remainingTime = 0;

            for (int i = shortest_time_index; i < queue2Size
- 1; i++)
            {
                queue2[i] = queue2[i + 1];
            }
            queue2Size--;

            printf("Time %d: Process %d finished\n", time,
p.pid);
        }
    }

    return 0;
}
```

METHODOLOGY USED TO SOLVE THE PROBLEM

The program simulates a simple priority-based scheduling algorithm for processes in an operating system. The program uses two queues - queue1 and queue2 - to hold the processes, where queue1 holds processes with priority 0 and queue2 holds processes with priority greater than 0.

The program then starts executing processes based on priority and remaining time. First execute the processes in Queue1, including the process with the highest priority value. It chooses the process with the highest priority value and runs it for a set amount of time (in this case multiple of 2 units). If the process completes within that time, it is removed from the queue and the program advances to the next process in the queue. If the process has not completed, it is moved to queue 2. Queue 2 contains processes with low priority values.

When all processes in queue1 have run, the program continues running processes in queue2. Pick the process with the least remaining time and run it to completion. If the operation has not completed, it is moved to the end of the queue. The program keeps running processes until all processes are completed. The program prints the progress of each process and the time it ran and completed.

The program first prompts the user to input the number of processes to execute. Then, the program reads in the details of each process - process ID, priority value, and remaining time - using a loop. The program adds each process to the appropriate queue based on its priority value.

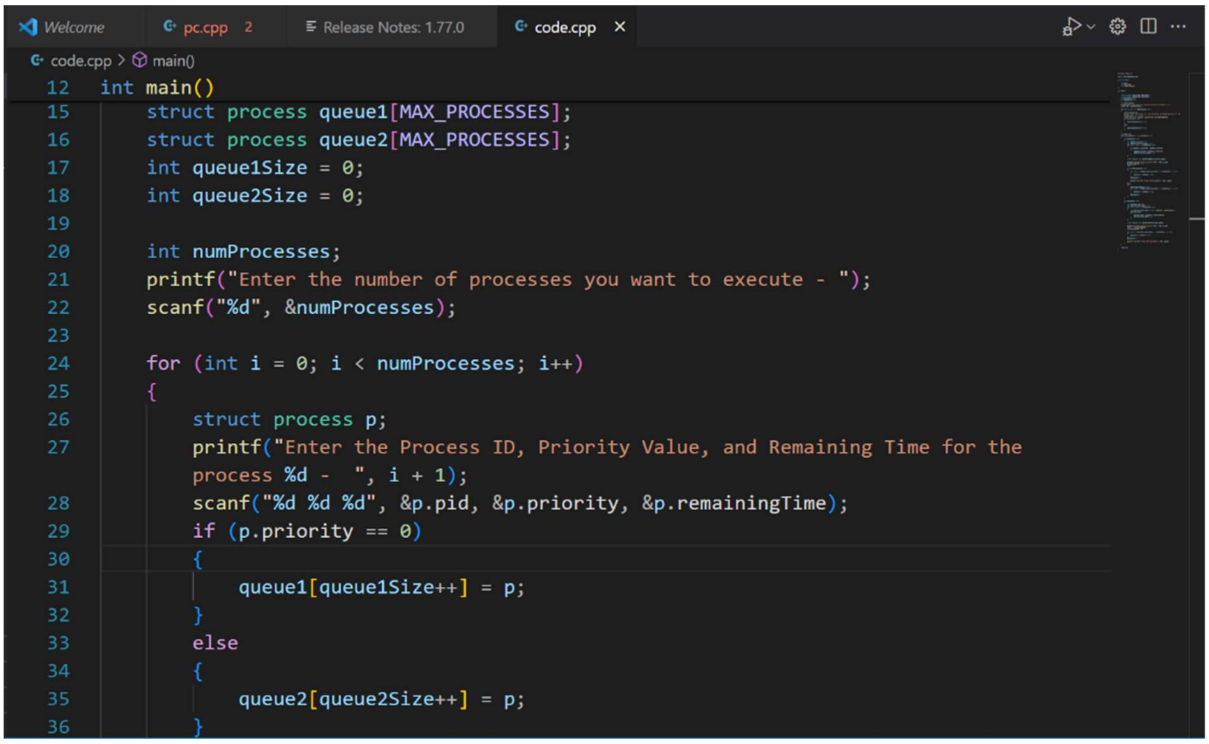
The program then starts executing the processes in the queues using a loop that runs until both queues are empty. The program follows the following logic to execute the processes:

1. If queue1 is not empty, the program selects the process with the highest priority from queue1 based on its priority value. The program then runs this process for 2 time units and decrements its remaining time by 2.
2. If the remaining time of the process becomes 0, the program removes it from queue1 and prints a message indicating that the process has finished.
3. If the remaining time of the process is still greater than 0, the program moves it to queue2 and removes it from queue1.
4. If queue1 is empty and queue2 is not empty, the program selects the process with the shortest remaining time from queue2. The program then runs this process for the remaining time and sets its remaining time to 0.
5. The program then removes the process from queue2 and prints a message indicating that the process has finished.

The program prints messages to indicate when a process starts and finishes running and displays the time at which each event occurs.

Overall, the program prioritizes processes with a higher priority value and executes them before processes with lower priority values. If there are no processes in queue1, the program executes processes in queue2 based on their remaining time.

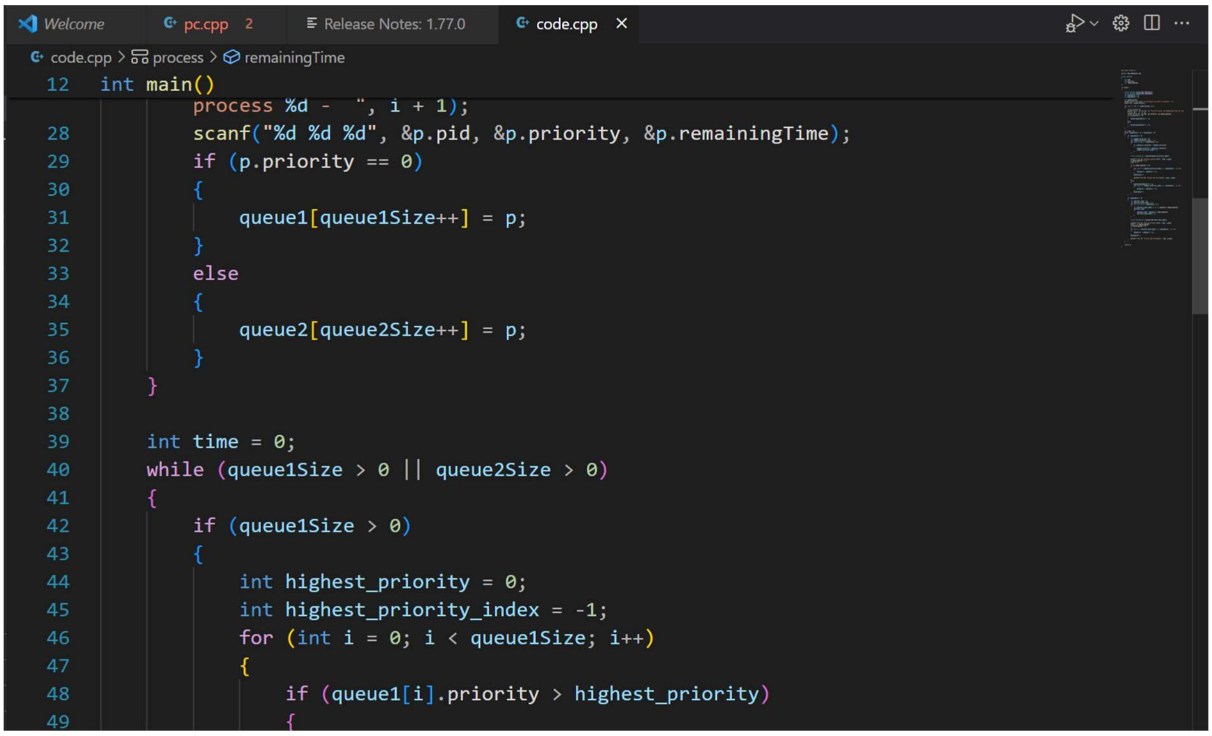
PICTURE OF CODE IN VISUAL STUDIO CODE



```c
int main()
    struct process queue1[MAX_PROCESSES];
    struct process queue2[MAX_PROCESSES];
    int queue1Size = 0;
    int queue2Size = 0;

    int numProcesses;
    printf("Enter the number of processes you want to execute - ");
    scanf("%d", &numProcesses);

    for (int i = 0; i < numProcesses; i++)
    {
        struct process p;
        printf("Enter the Process ID, Priority Value, and Remaining Time for the process %d -  ", i + 1);
        scanf("%d %d %d", &p.pid, &p.priority, &p.remainingTime);
        if (p.priority == 0)
        {
            queue1[queue1Size++] = p;
        }
        else
        {
            queue2[queue2Size++] = p;
        }
```

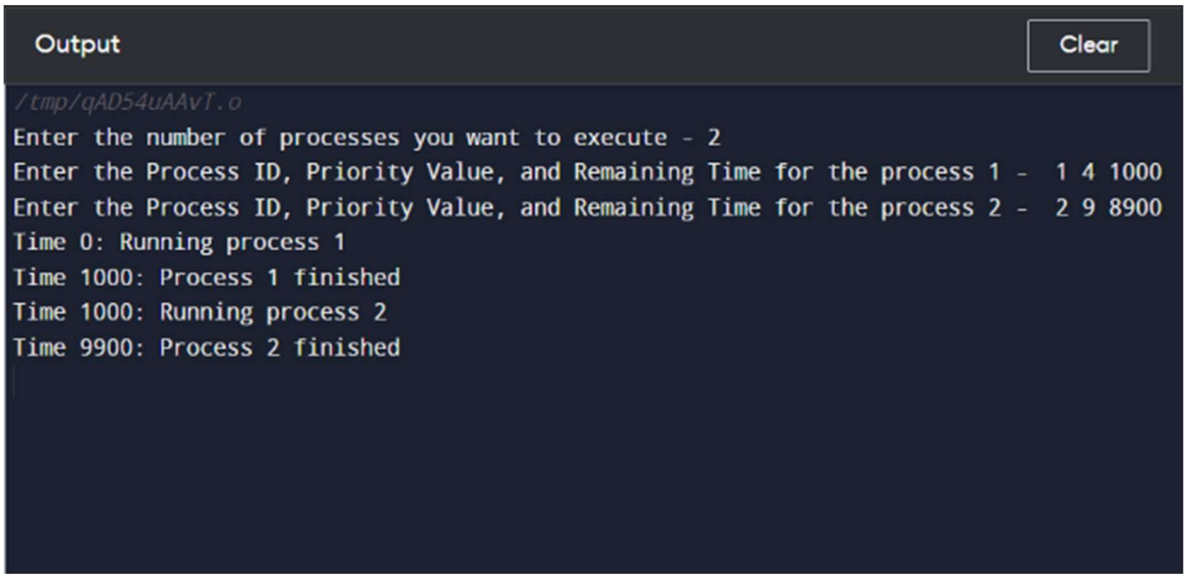


```
int main()
        process %d - ", i + 1);
        scanf("%d %d %d", &p.pid, &p.priority, &p.remainingTime);
        if (p.priority == 0)
        {
            queue1[queue1Size++] = p;
        }
        else
        {
            queue2[queue2Size++] = p;
        }
    }

    int time = 0;
    while (queue1Size > 0 || queue2Size > 0)
    {
        if (queue1Size > 0)
        {
            int highest_priority = 0;
            int highest_priority_index = -1;
            for (int i = 0; i < queue1Size; i++)
            {
                if (queue1[i].priority > highest_priority)
                {
```

PICTURE OF OUTPUT

Output 1

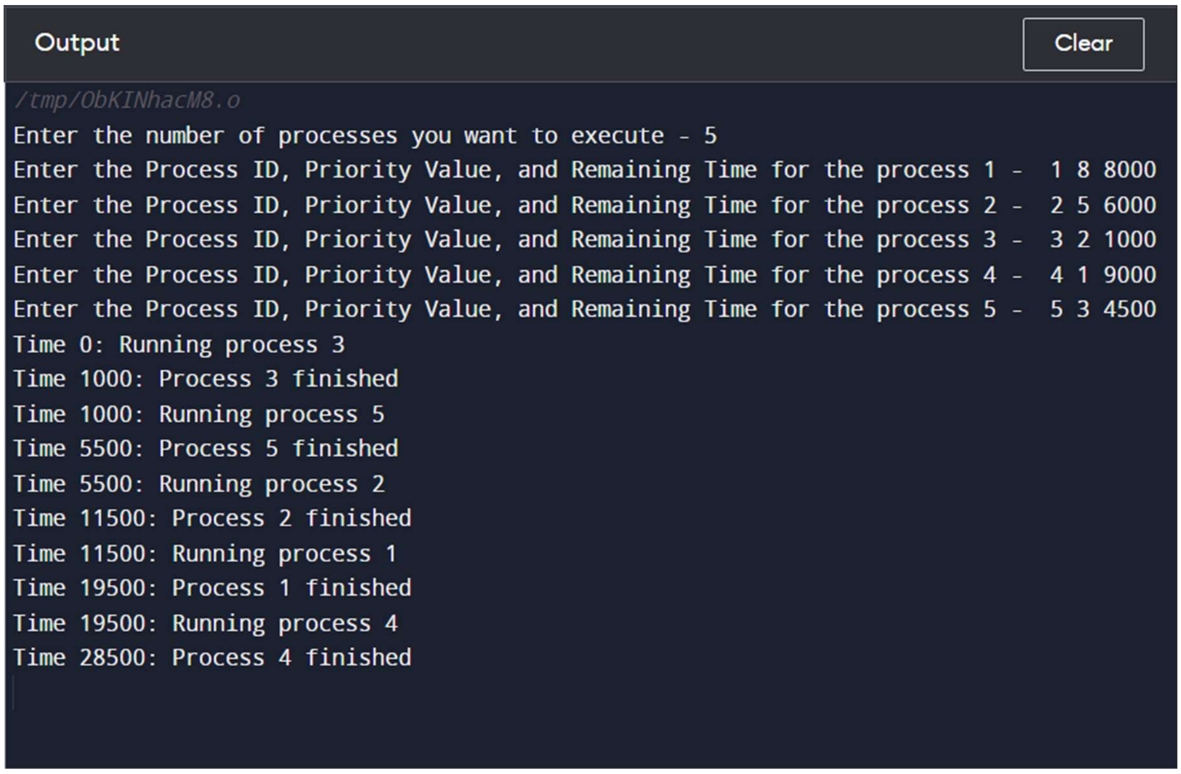


```
/tmp/qAD54uAAvT.o
Enter the number of processes you want to execute - 2
Enter the Process ID, Priority Value, and Remaining Time for the process 1 -  1 4 1000
Enter the Process ID, Priority Value, and Remaining Time for the process 2 -  2 9 8900
Time 0: Running process 1
Time 1000: Process 1 finished
Time 1000: Running process 2
Time 9900: Process 2 finished
```

<u>Explanation –</u>

The output shows the execution of two processes with process ID 1 and 2, priority values of 4 and 9, and remaining times of 1000 and 8900 units, respectively. The program first executes process 1 since it has the highest priority in queue1. After 1000 units, process 1 completes its execution and is removed from the queue. The program then executes process 2, which takes 9900 units to complete its execution. Finally, the program terminates since there are no more processes in either queue.

<u>Output 2</u>



```
/tmp/ObKINhacM8.o
Enter the number of processes you want to execute - 5
Enter the Process ID, Priority Value, and Remaining Time for the process 1 -  1 8 8000
Enter the Process ID, Priority Value, and Remaining Time for the process 2 -  2 5 6000
Enter the Process ID, Priority Value, and Remaining Time for the process 3 -  3 2 1000
Enter the Process ID, Priority Value, and Remaining Time for the process 4 -  4 1 9000
Enter the Process ID, Priority Value, and Remaining Time for the process 5 -  5 3 4500
Time 0: Running process 3
Time 1000: Process 3 finished
Time 1000: Running process 5
Time 5500: Process 5 finished
Time 5500: Running process 2
Time 11500: Process 2 finished
Time 11500: Running process 1
Time 19500: Process 1 finished
Time 19500: Running process 4
Time 28500: Process 4 finished
```

<u>Explanation –</u>

The output shows the order in which the processes are executed and finished, as well as the time at which each process is started and finished. The output shows that process 3 is the first process to be executed, since it has the highest priority value in the first queue. After process 3 finishes, process 5 is executed, since it has the shortest remaining time in the second queue. Next, process 2 is executed, followed by process 1, and finally process 4. The program finishes executing all processes at time 28500.

<u>------------The End ------------</u>