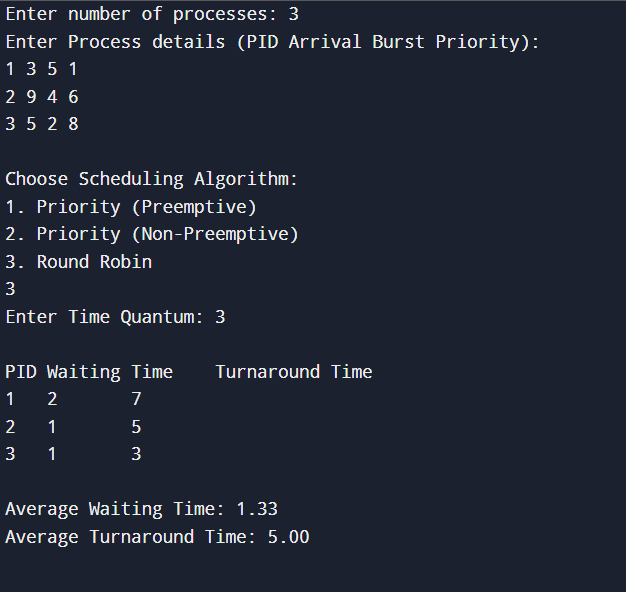
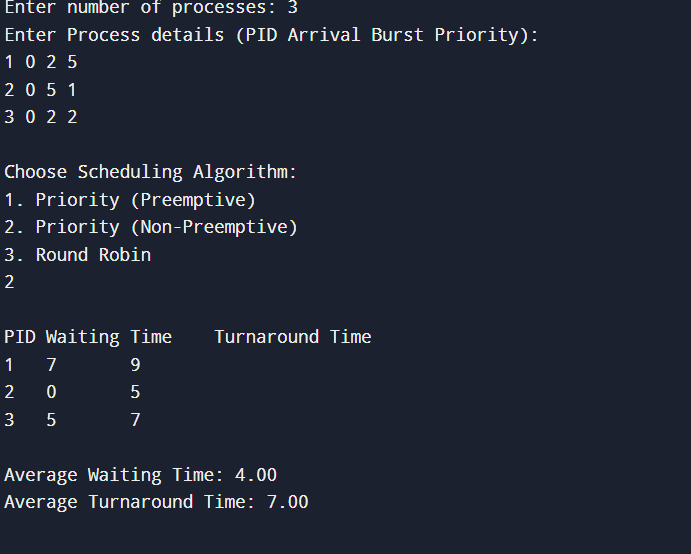
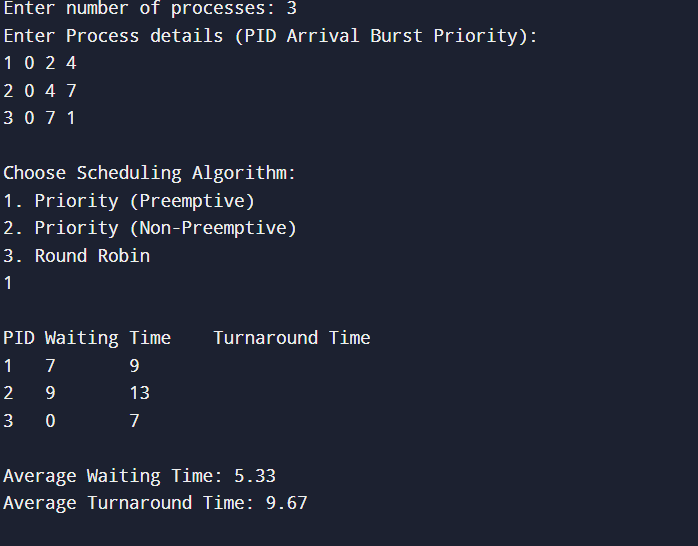
Lab 3- Priority(Preemptive and non- preemptive) And Round Robin

struct Process {  
    int pid;  
    int arrival;  
    int burst;  
    int priority;  
    int remaining;  
    int waiting;  
    int turnaround;  
};  
  
void priorityScheduling(struct Process p[], int n, int isPreemptive) {  
    int completed = 0, time = 0;  
    int minPriorityIdx;  
    while (completed < n) {  
        minPriorityIdx = -1;  
        int minPriority = 99999;  
         
        for (int i = 0; i < n; i++) {  
            if (p[i].arrival <= time && p[i].remaining > 0) {  
                if (p[i].priority < minPriority) {  
                    minPriority = p[i].priority;  
                    minPriorityIdx = i;  
                }  
            }  
        }  
         
        if (minPriorityIdx == -1) {  
            time++;  
            continue;  
        }  
         
        if (isPreemptive) {  
            p[minPriorityIdx].remaining--;  
            time++;  
            if (p[minPriorityIdx].remaining == 0) {  
                completed++;  
                p[minPriorityIdx].turnaround = time - p[minPriorityIdx].arrival;  
                p[minPriorityIdx].waiting = p[minPriorityIdx].turnaround - p[minPriorityIdx].burst;  
            }  
        } else {  
            time += p[minPriorityIdx].remaining;  
            p[minPriorityIdx].turnaround = time - p[minPriorityIdx].arrival;  
            p[minPriorityIdx].waiting = p[minPriorityIdx].turnaround - p[minPriorityIdx].burst;  
            p[minPriorityIdx].remaining = 0;  
            completed++;  
        }  
    }  
}  
  
void roundRobin(struct Process p[], int n, int quantum) {  
    int time = 0, completed = 0;  
    while (completed < n) {  
        int done = 1;  
        for (int i = 0; i < n; i++) {  
            if (p[i].remaining > 0 && p[i].arrival <= time) {  
                done = 0;  
                if (p[i].remaining > quantum) {  
                    time += quantum;  
                    p[i].remaining -= quantum;  
                } else {  
                    time += p[i].remaining;  
                    p[i].turnaround = time - p[i].arrival;  
                    p[i].waiting = p[i].turnaround - p[i].burst;  
                    p[i].remaining = 0;  
                    completed++;  
                }  
            }  
        }  
        if (done) time++;  
    }  
}  
  
void displayResults(struct Process p[], int n) {  
    printf("\nPID\tWaiting Time\tTurnaround Time\n");  
    float totalWT = 0, totalTAT = 0;  
    for (int i = 0; i < n; i++) {  
        printf("%d\t%d\t\t%d\n", p[i].pid, p[i].waiting, p[i].turnaround);  
        totalWT += p[i].waiting;  
        totalTAT += p[i].turnaround;  
    }  
    printf("\nAverage Waiting Time: %.2f", totalWT / n);  
    printf("\nAverage Turnaround Time: %.2f\n", totalTAT / n);  
}  
  
int main() {  
    int n, choice, quantum;  
    printf("Enter number of processes: ");  
    scanf("%d", &n);  
    struct Process p[n];  
     
    printf("Enter Process details (PID Arrival Burst Priority):\n");  
    for (int i = 0; i < n; i++) {  
        scanf("%d %d %d %d", &p[i].pid, &p[i].arrival, &p[i].burst, &p[i].priority);  
        p[i].remaining = p[i].burst;  
    }  
     
    printf("\nChoose Scheduling Algorithm:\n");  
    printf("1. Priority (Preemptive)\n2. Priority (Non-Preemptive)\n3. Round Robin\n");  
    scanf("%d", &choice);  
     
    switch (choice) {  
        case 1:  
            priorityScheduling(p, n, 1);  
            break;  
        case 2:  
            priorityScheduling(p, n, 0);  
            break;  
        case 3:  
            printf("Enter Time Quantum: ");  
            scanf("%d", &quantum);  
            roundRobin(p, n, quantum);  
            break;  
        default:  
            printf("Invalid choice!\n");  
            return 1;  
    }  
     
    displayResults(p, n);  
    return 0;

**Output:**

}

|  |  |  |
| --- | --- | --- |
|  |  |  |