fcfs & sjf

```
#include<stdio.h>

// Structure to represent a process
struct Process {
    int pid;      // Process ID
    int burstTime; // Burst time
};

// Function to perform SJF scheduling
void sjfScheduling(struct Process processes[], int n) {
    int waitingTime[n], turnaroundTime[n], completionTime[n];
    int totalWaitingTime = 0, totalTurnaroundTime = 0;

    // Calculate waiting time for each process
    waitingTime[0] = 0; // Waiting time for the first process is 0
    for (int i = 1; i < n; i++) {
        waitingTime[i] = processes[i - 1].burstTime + waitingTime[i - 1];
        totalWaitingTime += waitingTime[i];
    }

    // Calculate turnaround time and completion time for each process
    for (int i = 0; i < n; i++) {
        turnaroundTime[i] = processes[i].burstTime + waitingTime[i];
        totalTurnaroundTime += turnaroundTime[i];
        completionTime[i] = turnaroundTime[i];
    }

    // Print the process details
    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", processes[i].pid, processes[i].burstTime, waitingTime[i], turnaroundTime[i]);
    }

    // Print the average waiting time and average turnaround time
    printf("\nAverage Waiting Time: %.2f\n", (float)totalWaitingTime / n);
    printf("Average Turnaround Time: %.2f\n", (float)totalTurnaroundTime / n);
}

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process processes[n];
    printf("Enter the burst time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d: ", i + 1);
        scanf("%d", &processes[i].burstTime);
        processes[i].pid = i + 1;
    }

    sjfScheduling(processes, n);

    return 0;
}
```

round robin

```
#include <stdio.h>

// Structure to represent a process
struct Process {
    int pid;          // Process ID
    int burstTime;     // Burst time
    int remainingTime; // Remaining burst time
};

// Function to perform Round Robin scheduling
void roundRobinScheduling(struct Process processes[], int n, int quantum) {
    int currentTime = 0;
    int completed = 0;

    // Calculate the total burst time of all processes
    int totalBurstTime = 0;
    for (int i = 0; i < n; i++) {
        totalBurstTime += processes[i].burstTime;
    }

    // Process the tasks in round robin fashion
    while (completed < n) {
        for (int i = 0; i < n; i++) {
            if (processes[i].remainingTime > 0) {
                // Execute the process for the quantum time or until it completes
                if (processes[i].remainingTime <= quantum) {
                    currentTime += processes[i].remainingTime;
                    processes[i].remainingTime = 0;
                } else {
                    currentTime += quantum;
                    processes[i].remainingTime -= quantum;
                }

                // Print the progress
                printf("Process %d executed for %d units.\n", processes[i].pid, currentTime);

                // Check if the process has completed
                if (processes[i].remainingTime == 0) {
                    completed++;
                }
            }
        }
    }

    // Calculate the turnaround time for each process
    int turnaroundTime[n];
    for (int i = 0; i < n; i++) {
        turnaroundTime[i] = currentTime - processes[i].burstTime;
    }

    // Print the process details
    printf("\nProcess\tBurst Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\n", processes[i].pid, processes[i].burstTime, turnaroundTime[i]);
    }
}
```

```
int main() {
   int n, quantum;
   printf("Enter the number of processes: ");
   scanf("%d", &n);

   struct Process processes[n];
   printf("Enter the burst time for each process:\n");
   for (int i = 0; i < n; i++) {
      printf("Process %d: ", i + 1);
      scanf("%d", &processes[i].burstTime);
      processes[i].pid = i + 1;
      processes[i].remainingTime = processes[i].burstTime;
   }

   printf("Enter the time quantum: ");
   scanf("%d", &quantum);

   roundRobinScheduling(processes, n, quantum);

   return 0;
}
```

preority

```
#include <stdio.h>

// Structure to represent a process
struct Process {
   int pid;          // Process ID
   int burstTime;     // Burst time
   int priority;      // Priority value (lower value indicates higher priority)
```

```
};

// Function to perform Priority Scheduling
void priorityScheduling(struct Process processes[], int n) {
    // Sort the processes based on priority using selection sort
    for (int i = 0; i < n - 1; i++) {
        int minIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (processes[j].priority < processes[minIndex].priority) {
                minIndex = j;
            }
        }
        // Swap processes[i] and processes[minIndex]
        struct Process temp = processes[i];
        processes[i] = processes[minIndex];
        processes[minIndex] = temp;
    }

    int currentTime = 0;

    // Calculate the turnaround time for each process
    int turnaroundTime[n];
    for (int i = 0; i < n; i++) {
        currentTime += processes[i].burstTime;
        turnaroundTime[i] = currentTime;
    }

    // Print the process details
    printf("Process\tBurst Time\tPriority\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", processes[i].pid, processes[i].burstTime, processes[i].priority, turnaroundTime[i]);
    }
}

int main() {
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process processes[n];
    printf("Enter the burst time and priority for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d:\n", i + 1);
        printf("Burst Time: ");
        scanf("%d", &processes[i].burstTime);
        printf("Priority: ");
        scanf("%d", &processes[i].priority);
        processes[i].pid = i + 1;
    }

    priorityScheduling(processes, n);

    return 0;
}
```