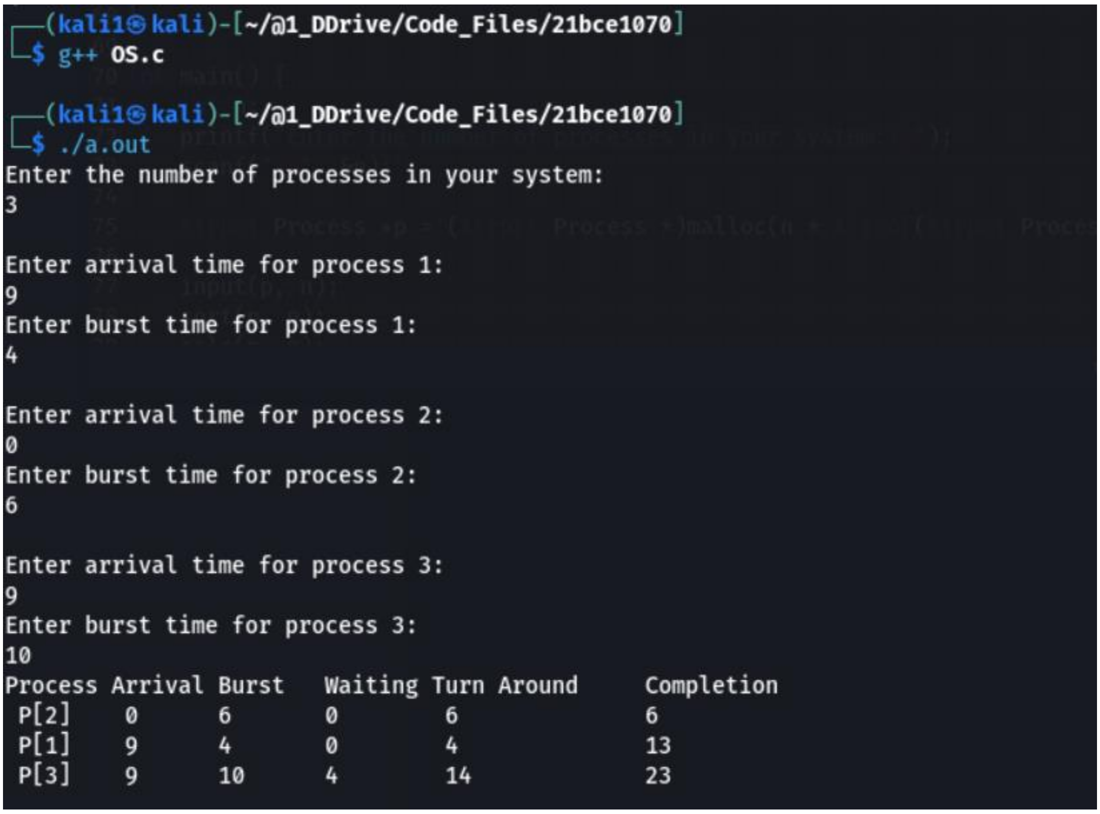
Process Scheduling (with arrival time)

FCFS

Code:

```
#include <stdio.h>
#include <stdlib.h>

struct Process {
   int id;
   int bt;
   int at;
   int ct;
   int tat;
   int wt;
};

void input(struct Process *p, int n) {
   for (int i = 0; i < n; i++) {
      printf("\nEnter arrival time for process %d:\n", i + 1);
      scanf("%d", &p[i].at);
      printf("Enter burst time for process %d:\n", i + 1);
      scanf("%d", &p[i].bt);
      p[i].id = i + 1;
   }
}

void calc(struct Process *p, int n) {
   int sum = 0;
   sum = sum + p[0].at;
   for (int i = 0; i < n; i++) {
      sum = sum + p[i].bt;

      p[i].ct = sum;

      p[i].tat = p[i].ct - p[i].at;

      p[i].wt = p[i].tat - p[i].bt;

      if (sum < p[i + 1].at) {

          int t = p[i + 1].at - sum;

          sum = sum + t;

      }

   }

}


void sort(struct Process *p, int n) {

   for (int i = 0; i < n - 1; i++) {

      for (int j = 0; j < n - i - 1; j++) {

          if (p[j].at > p[j + 1].at) {

              int temp;


              // Sorting burst times

              temp = p[j].bt;

              p[j].bt = p[j + 1].bt;

              p[j + 1].bt = temp;


              // Sorting arrival times

              temp = p[j].at;

              p[j].at = p[j + 1].at;

              p[j + 1].at = temp;


              // Sorting their respective IDs

              temp = p[j].id;

              p[j].id = p[j + 1].id;

              p[j + 1].id = temp;
            }
        }
    }
}

void show(struct Process *p, int n) {
    printf("Process\tArrival\tBurst\tWaiting\tTurn Around\tCompletion\n");
    for (int i = 0; i < n; i++) {
        printf(" P[%d]\t %d\t%d\t%d\t %d\t\t%d\n", p[i].id, p[i].at, p[i].bt, p[i].wt, p[i].tat, p[i].ct);
    }
}

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

    struct Process *p = (struct Process *)malloc(n * sizeof(struct Process));

    input(p, n);
    sort(p, n);
    calc(p, n);
    show(p, n);

    free(p);

    return 0;
}
```

Output:



```
┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
└─$ g++ OS.c

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
└─$ ./a.out
Enter the number of processes in your system:
3

Enter arrival time for process 1:
9
Enter burst time for process 1:
4

Enter arrival time for process 2:
0
Enter burst time for process 2:
6

Enter arrival time for process 3:
9
Enter burst time for process 3:
10
Process Arrival Burst   Waiting Turn Around     Completion
 P[2]    0      6       0        6              6
 P[1]    9      4       0        4              13
 P[3]    9      10      4        14             23
```

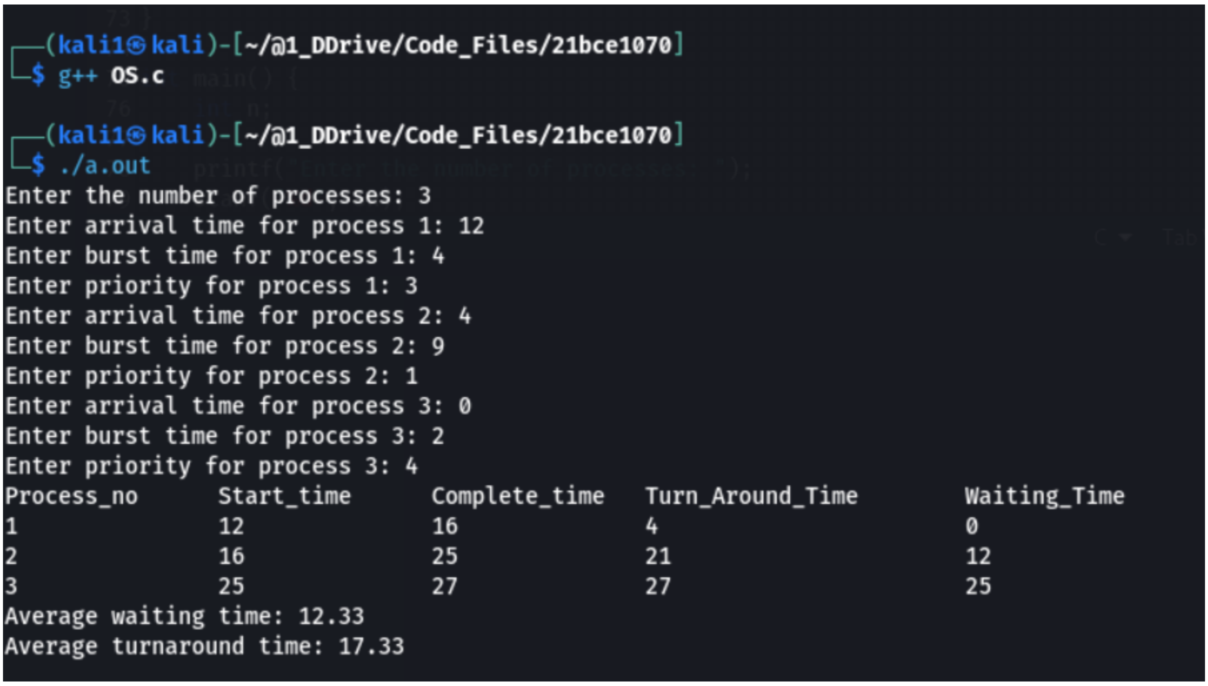
Priority (non pre emptive)

Code:

```
#include <stdio.h>

#include <stdlib.h>


#define MAX_PROCESS 50


struct Process {

   int at;

   int bt;

   int pr;

   int pno;

};


int comp(const void *a, const void *b) {
   struct Process *p1 = (struct Process *)a;
   struct Process *p2 = (struct Process *)b;

   if (p1->at == p2->at) {
       return p1->pr < p2->pr;
   } else {
       return p1->at < p2->at;
   }
}

void get_wt_time(struct Process *proc, int n, int wt[]) {
   int service[MAX_PROCESS];
   service[0] = proc[0].at;
   wt[0] = 0;

   for (int i = 1; i < n; i++) {
       service[i] = proc[i - 1].bt + service[i - 1];
       wt[i] = service[i] - proc[i].at;

       if (wt[i] < 0) {
           wt[i] = 0;
       }
   }
}

void get_tat_time(struct Process *proc, int n, int tat[], int wt[]) {
   for (int i = 0; i < n; i++) {
       tat[i] = proc[i].bt + wt[i];
   }
}
```

```
void findgc(struct Process *proc, int n) {
    int wt[MAX_PROCESS], tat[MAX_PROCESS];
    double wavg = 0, tavg = 0;

    get_wt_time(proc, n, wt);
    get_tat_time(proc, n, tat, wt);

    int stime[MAX_PROCESS], ctime[MAX_PROCESS];

    stime[0] = proc[0].at;
    ctime[0] = stime[0] + tat[0];

    for (int i = 1; i < n; i++) {
        stime[i] = ctime[i - 1];
        ctime[i] = stime[i] + tat[i] - wt[i];
    }

    printf("Process_no\tStart_time\tComplete_time\tTurn_Around_Time\tWaiting_Time\n");

    for (int i = 0; i < n; i++) {
        wavg += wt[i];
        tavg += tat[i];

        printf("%d\t\t%d\t\t%d\t\t%d\t\t\t%d\n", proc[i].pno, stime[i], ctime[i], tat[i], wt[i]);
    }

    printf("Average waiting time: %.2f\n", wavg / (double)n);
    printf("Average turnaround time: %.2f\n", tavg / (double)n);
}

int main() {
    int n;

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

    struct Process *proc = (struct Process *)malloc(n * sizeof(struct Process));

    for (int i = 0; i < n; i++) {
        printf("Enter arrival time for process %d: ", i + 1);
        scanf("%d", &proc[i].at);

        printf("Enter burst time for process %d: ", i + 1);
        scanf("%d", &proc[i].bt);

        printf("Enter priority for process %d: ", i + 1);
        scanf("%d", &proc[i].pr);

        proc[i].pno = i + 1;
    }

    qsort(proc, n, sizeof(struct Process), comp);

    findgc(proc, n);

    free(proc);

    return 0;
}
```

Output:



```
┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
└─$ g++ OS.c

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
└─$ ./a.out
Enter the number of processes: 3
Enter arrival time for process 1: 12
Enter burst time for process 1: 4
Enter priority for process 1: 3
Enter arrival time for process 2: 4
Enter burst time for process 2: 9
Enter priority for process 2: 1
Enter arrival time for process 3: 0
Enter burst time for process 3: 2
Enter priority for process 3: 4
Process_no      Start_time      Complete_time   Turn_Around_Time        Waiting_Time
1               12              16              4                       0
2               16              25              21                      12
3               25              27              27                      25
Average waiting time: 12.33
Average turnaround time: 17.33
```

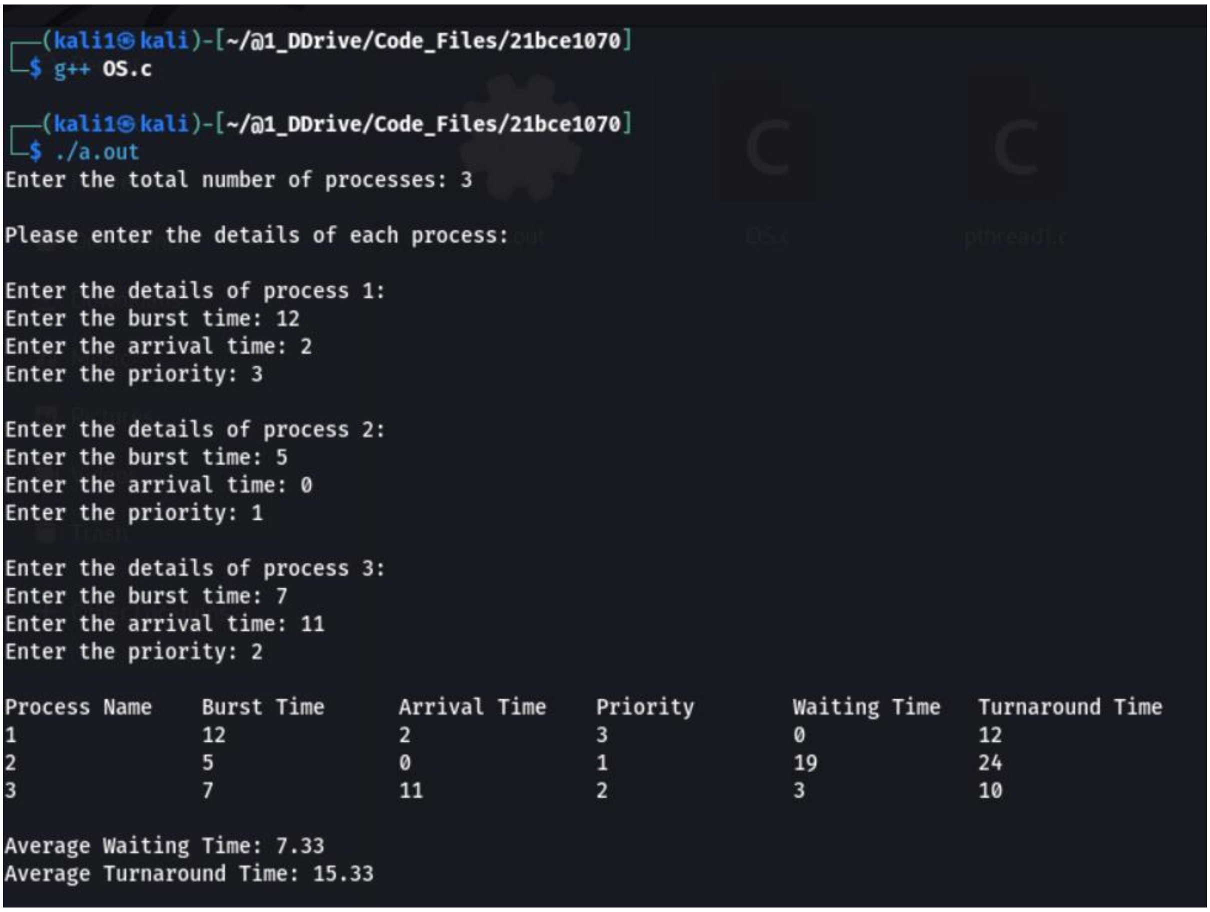
Priority (Pre emptive)

Code:

```
#include<stdio.h>


struct Process {

  int name;

  int bt;

  int at;

  int priority;

  int rt;

  int wt;

  int tat;

};


void calcWT(struct Process p[], int n) {

  int rem = n;

  int currentTime = 0;
```

```
 while (rem > 0) {
  int nextProcess = -1;
  int highestPriority = -1;

  for (int i = 0; i < n; i++) {
   if (p[i].at <= currentTime && p[i].rt > 0) {
    if (p[i].priority > highestPriority || highestPriority == -1) {
     highestPriority = p[i].priority;
     nextProcess = i;
    }
   }
  }

  if (nextProcess == -1) {
   currentTime++;
   continue;
  }

  p[nextProcess].rt--;
  currentTime++;

  if (p[nextProcess].rt == 0) {
   rem--;
   p[nextProcess].tat = currentTime - p[nextProcess].at;
   p[nextProcess].wt = p[nextProcess].tat - p[nextProcess].bt;
  }
 }
}

void calcTAT(struct Process p[], int n) {
 for (int i = 0; i < n; i++) {
    p[i].tat = p[i].bt + p[i].wt;
  }
}

void calcAvgTimes(struct Process p[], int n, float *avgWT, float *avgTAT) {
  int totalWT = 0;
  int totalTAT = 0;

  for (int i = 0; i < n; i++) {
    totalWT += p[i].wt;
    totalTAT += p[i].tat;
  }

  *avgWT = (float)totalWT / n;
  *avgTAT = (float)totalTAT / n;
}

void displayDetails(struct Process p[], int n) {
  printf("\nProcess Name\tBurst Time\tArrival Time\tPriority\tWaiting Time\tTurnaround Time\n");
  for (int i = 0; i < n; i++) {
    printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", p[i].name, p[i].bt, p[i].at, p[i].priority, p[i].wt, p[i].tat);
  }
}

int main() {
  int n;
  float avgWT, avgTAT;

  printf("Enter the total number of processes: ");
  scanf("%d", &n);
  struct Process p[n];

  printf("\nPlease enter the details of each process:\n");

  for (int i = 0; i < n; i++) {
    p[i].name = i + 1;

    printf("\nEnter the details of process %d:\n", p[i].name);
    printf("Enter the burst time: ");
    scanf("%d", &p[i].bt);

    printf("Enter the arrival time: ");
    scanf("%d", &p[i].at);

    printf("Enter the priority: ");
    scanf("%d", &p[i].priority);

    p[i].rt = p[i].bt;
  }

  calcWT(p, n);
  calcTAT(p, n);
  calcAvgTimes(p, n, &avgWT, &avgTAT);
  displayDetails(p, n);

  printf("\nAverage Waiting Time: %.2f", avgWT);
  printf("\nAverage Turnaround Time: %.2f\n", avgTAT);

  return 0;
}
```

Output:



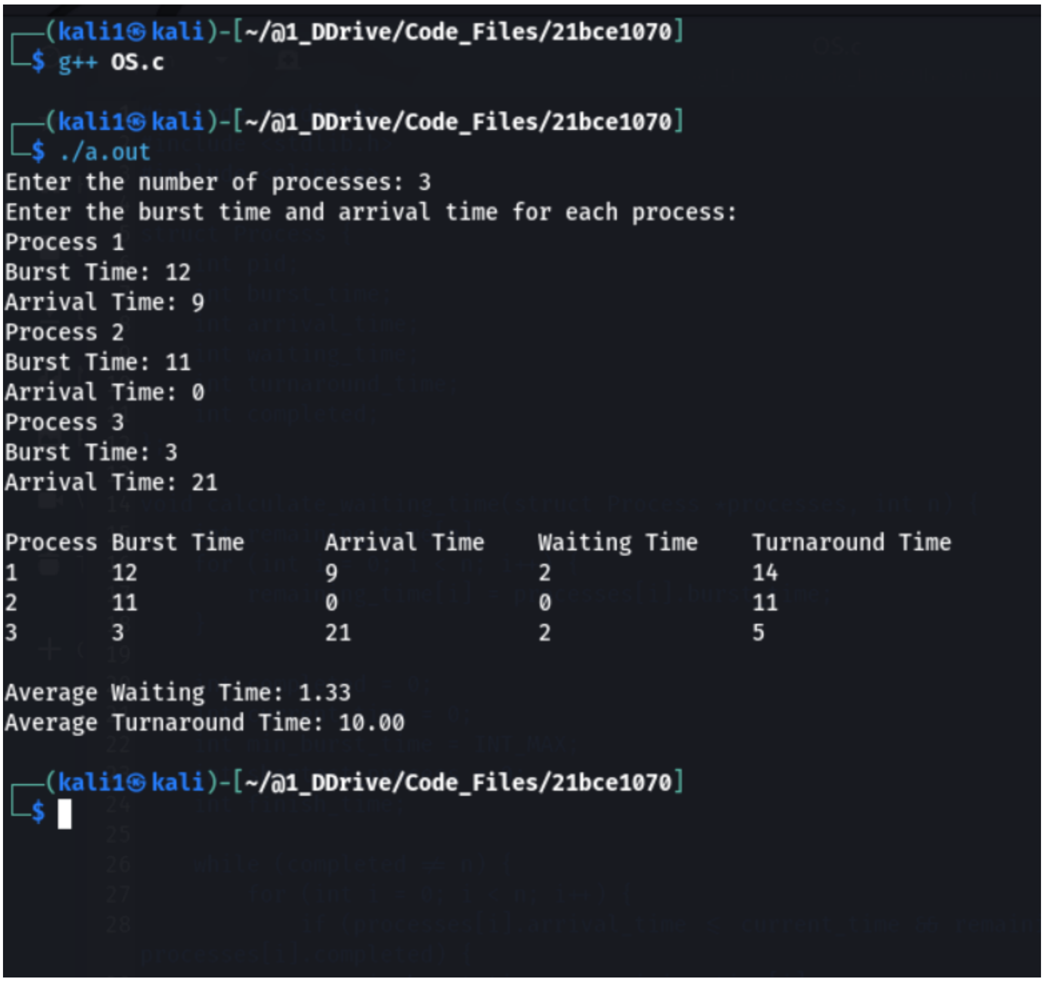
```
┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
└─$ g++ OS.c

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
└─$ ./a.out
Enter the total number of processes: 3

Please enter the details of each process:

Enter the details of process 1:
Enter the burst time: 12
Enter the arrival time: 2
Enter the priority: 3

Enter the details of process 2:
Enter the burst time: 5
Enter the arrival time: 0
Enter the priority: 1

Enter the details of process 3:
Enter the burst time: 7
Enter the arrival time: 11
Enter the priority: 2

Process Name    Burst Time      Arrival Time    Priority        Waiting Time   Turnaround Time
1               12              2               3               0              12
2               5               0               1               19             24
3               7               11              2               3              10

Average Waiting Time: 7.33
Average Turnaround Time: 15.33
```

Shortest Job First (non pre emptive)

```
#include <stdio.h>

#include <stdlib.h>

#include <climits>


struct Process {

  int pid;

  int bt;

  int at;

  int wt;
```

```
    int tat;
    int completed;
};

void calculate_waiting_time(struct Process *proc, int n) {
    int remaining_time[n];
    for (int i = 0; i < n; i++) {
        remaining_time[i] = proc[i].bt;
    }

    int completed = 0;
    int current_time = 0;
    int min_burst_time = INT_MAX;
    int shortest_process = 0;
    int finish_time;

    while (completed != n) {
        for (int i = 0; i < n; i++) {
            if (proc[i].at <= current_time && remaining_time[i] < min_burst_time && !proc[i].completed)
{
                min_burst_time = remaining_time[i];
                shortest_process = i;
            }
        }

        if (min_burst_time == INT_MAX) {
            current_time++;
            continue;
        }

        remaining_time[shortest_process]--;
        min_burst_time = remaining_time[shortest_process];
        if (min_burst_time == 0) {
            min_burst_time = INT_MAX;
        }

        if (remaining_time[shortest_process] == 0) {
            completed++;
            finish_time = current_time + 1;
            proc[shortest_process].wt = finish_time - proc[shortest_process].bt -
proc[shortest_process].at;
            proc[shortest_process].tat = finish_time - proc[shortest_process].at;
            proc[shortest_process].completed = 1;
        }

        current_time++;
    }
}

void calculate_turnaround_time(struct Process *proc, int n) {
    for (int i = 0; i < n; i++) {
        proc[i].tat = proc[i].bt + proc[i].wt;
    }
}

void calculate_average_times(struct Process *proc, int n, float *avg_wt, float *avg_tat) {
    int total_wt = 0;
    int total_tat = 0;

    for (int i = 0; i < n; i++) {
        total_wt += proc[i].wt;
        total_tat += proc[i].tat;
    }

    *avg_wt = (float)total_wt / n;
    *avg_tat = (float)total_tat / n;
}

void display_results(struct Process *proc, int n, float avg_wt, float avg_tat) {
    printf("\nProcess\tBurst Time\tArrival Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < n; i++) {
        printf("%d\t%d\t\t%d\t\t%d\t\t%d\n", proc[i].pid, proc[i].bt, proc[i].at, proc[i].wt, proc[i].tat);
    }
    printf("\nAverage Waiting Time: %.2f\n", avg_wt);
    printf("Average Turnaround Time: %.2f\n", avg_tat);
}

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

    struct Process *proc = (struct Process *)malloc(n * sizeof(struct Process));

    printf("Enter the burst time and arrival time for each process:\n");
    for (int i = 0; i < n; i++) {
        proc[i].pid = i + 1;
        printf("Process %d\n", proc[i].pid);
        printf("Burst Time: ");
        scanf("%d", &proc[i].bt);
        printf("Arrival Time: ");
        scanf("%d", &proc[i].at);
        proc[i].completed = 0;
    }

    calculate_waiting_time(proc, n);
    calculate_turnaround_time(proc, n);

    float avg_wt, avg_tat;
    calculate_average_times(proc, n, &avg_wt, &avg_tat);

    display_results(proc, n, avg_wt, avg_tat);

    free(proc);

    return 0;
}
```

Output:



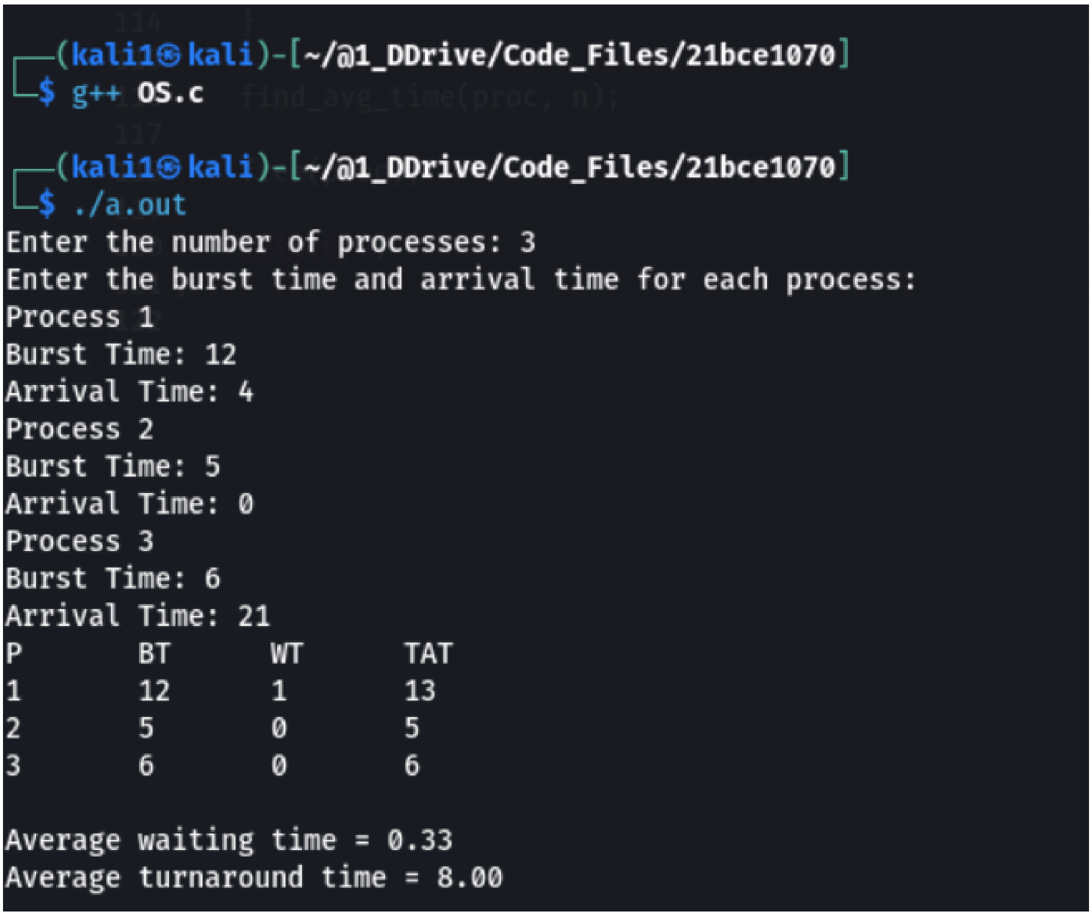
```
┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
└─$ g++ OS.c

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
└─$ ./a.out
Enter the number of processes: 3
Enter the burst time and arrival time for each process:
Process 1
Burst Time: 12
Arrival Time: 9
Process 2
Burst Time: 11
Arrival Time: 0
Process 3
Burst Time: 3
Arrival Time: 21

Process Burst Time      Arrival Time    Waiting Time    Turnaround Time
1       12              9               2               14
2       11              0               0               11
3       3               21              2               5

Average Waiting Time: 1.33
Average Turnaround Time: 10.00

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
└─$ █
```

SJF (Pre emptive)

```
#include <stdio.h>

#include <stdlib.h>

#include <limits.h>


struct Process {

    int pid; // Process ID

    int bt; // Burst Time

    int art; // Arrival Time

};


// Function to find the waiting time for all processes

void find_wt(struct Process proc[], int n, int wt[]) {
    int rt[n];

    // Copy the burst time into rt[]
    for (int i = 0; i < n; i++)
        rt[i] = proc[i].bt;

    int complete = 0, t = 0, minm = INT_MAX;
    int shortest = 0, finish_time;
    int check = 0;

    // Process until all processes get completed
    while (complete != n) {
        // Find process with minimum remaining time among the processes that arrive till the current time
        for (int j = 0; j < n; j++) {
            if ((proc[j].art <= t) && (rt[j] < minm) && rt[j] > 0) {
                minm = rt[j];
                shortest = j;
                check = 1;
            }
        }

        if (check == 0) {
            t++;
            continue;
        }

        // Reduce remaining time by one
        rt[shortest]--;

        // Update minimum
        minm = rt[shortest];
        if (minm == 0)
            minm = INT_MAX;

        // If a process gets completely executed
        if (rt[shortest] == 0) {
            // Increment complete
            complete++;
            check = 0;

            // Find finish time of current process
            finish_time = t + 1;

            // Calculate waiting time
            wt[shortest] = finish_time - proc[shortest].bt - proc[shortest].art;

            if (wt[shortest] < 0)
                wt[shortest] = 0;
        }
        // Increment time
        t++;
    }
}

// Function to calculate turn around time
void find_tat(struct Process proc[], int n, int wt[], int tat[]) {
    // calculating turnaround time by adding bt[i] + wt[i]
    for (int i = 0; i < n; i++)
        tat[i] = proc[i].bt + wt[i];
}
```

```
// Function to calculate average time
void find_avg_time(struct Process proc[], int n) {
    int wt[n], tat[n], total_wt = 0, total_tat = 0;

    // Function to find waiting time of all processes
    find_wt(proc, n, wt);

    // Function to find turn around time for all processes
    find_tat(proc, n, wt, tat);

    // Display processes along with all details
    printf("P\tBT\tWT\tTAT\n");

    // Calculate total waiting time and total turnaround time
    for (int i = 0; i < n; i++) {
        total_wt = total_wt + wt[i];
        total_tat = total_tat + tat[i];
        printf("%d\t%d\t%d\t%d\n", proc[i].pid, proc[i].bt, wt[i], tat[i]);
    }

    printf("\nAverage waiting time = %.2f", (float)total_wt / n);
    printf("\nAverage turnaround time = %.2f", (float)total_tat / n);
}

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

    struct Process *proc = (struct Process *)malloc(n * sizeof(struct Process));
    printf("Enter the burst time and arrival time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("Process %d\n", i + 1);
        printf("Burst Time: ");
        scanf("%d", &proc[i].bt);
        printf("Arrival Time: ");
        scanf("%d", &proc[i].art);
        proc[i].pid = i + 1;
    }

    find_avg_time(proc, n);

    free(proc);

    return 0;
}
```

Output:



```
┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
└─$ g++ OS.c

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files/21bce1070]
└─$ ./a.out
Enter the number of processes: 3
Enter the burst time and arrival time for each process:
Process 1
Burst Time: 12
Arrival Time: 4
Process 2
Burst Time: 5
Arrival Time: 0
Process 3
Burst Time: 6
Arrival Time: 21
P       BT      WT      TAT
1       12      1       13
2       5       0       5
3       6       0       6

Average waiting time = 0.33
Average turnaround time = 8.00
```