ASSIGNMENT 3

Name: Lakshita

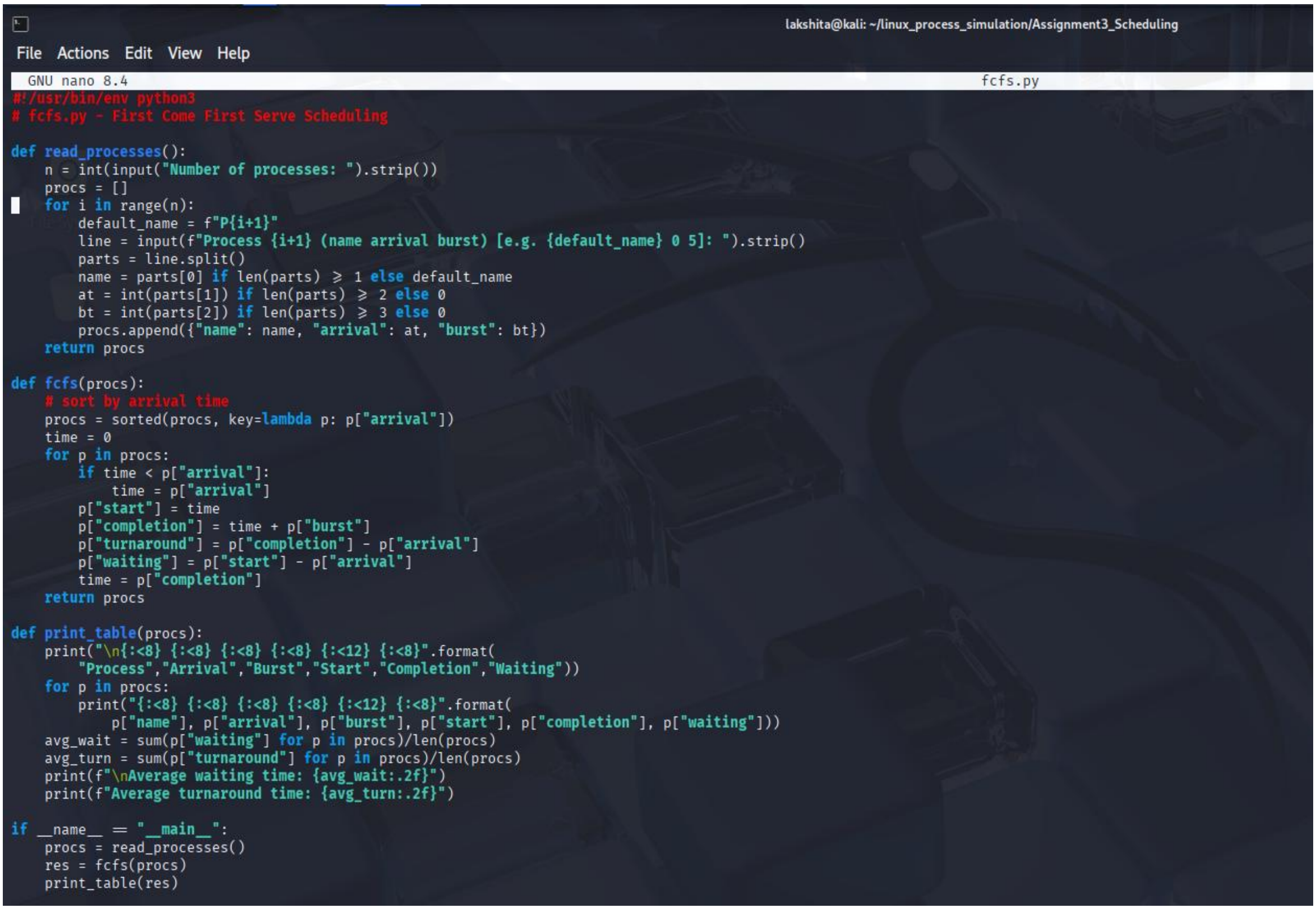
Roll No: 2301420008

Course: BTech CSE (DS)

(SCHEDULING)

1. First Come First Serve Scheduling

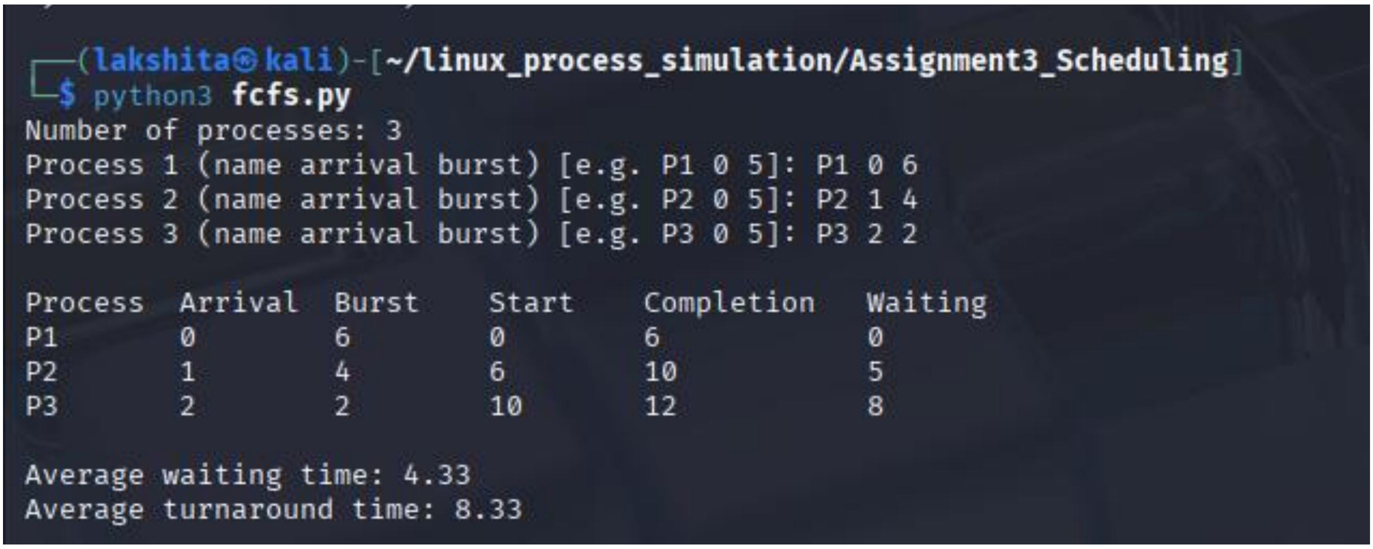
CODE



lakshita@kali: ~/linux_process_simulation/Assignment3_Scheduling
File Actions Edit View Help
GNU nano 8.4
fcfs.py

```
#!/usr/bin/env python3
# fcfs.py - First Come First Serve Scheduling

def read_processes():
    n = int(input("Number of processes: ").strip())
    procs = []
    for i in range(n):
        default_name = f"P{i+1}"
        line = input(f"Process {i+1} (name arrival burst) [e.g. {default_name} 0 5]: ").strip()
        parts = line.split()
        name = parts[0] if len(parts) >= 1 else default_name
        at = int(parts[1]) if len(parts) >= 2 else 0
        bt = int(parts[2]) if len(parts) >= 3 else 0
        procs.append({"name": name, "arrival": at, "burst": bt})
    return procs

def fcfs(procs):
    # sort by arrival time
    procs = sorted(procs, key=lambda p: p["arrival"])
    time = 0
    for p in procs:
        if time < p["arrival"]:
            time = p["arrival"]
        p["start"] = time
        p["completion"] = time + p["burst"]
        p["turnaround"] = p["completion"] - p["arrival"]
        p["waiting"] = p["start"] - p["arrival"]
        time = p["completion"]
    return procs

def print_table(procs):
    print("\n{:<8} {:<8} {:<8} {:<8} {:<12} {:<8}".format(
        "Process","Arrival","Burst","Start","Completion","Waiting"))
    for p in procs:
        print("{:<8} {:<8} {:<8} {:<8} {:<12} {:<8}".format(
            p["name"], p["arrival"], p["burst"], p["start"], p["completion"], p["waiting"]))
    avg_wait = sum(p["waiting"] for p in procs)/len(procs)
    avg_turn = sum(p["turnaround"] for p in procs)/len(procs)
    print(f"\nAverage waiting time: {avg_wait:.2f}")
    print(f"Average turnaround time: {avg_turn:.2f}")

if __name__ == "__main__":
    procs = read_processes()
    res = fcfs(procs)
    print_table(res)
```

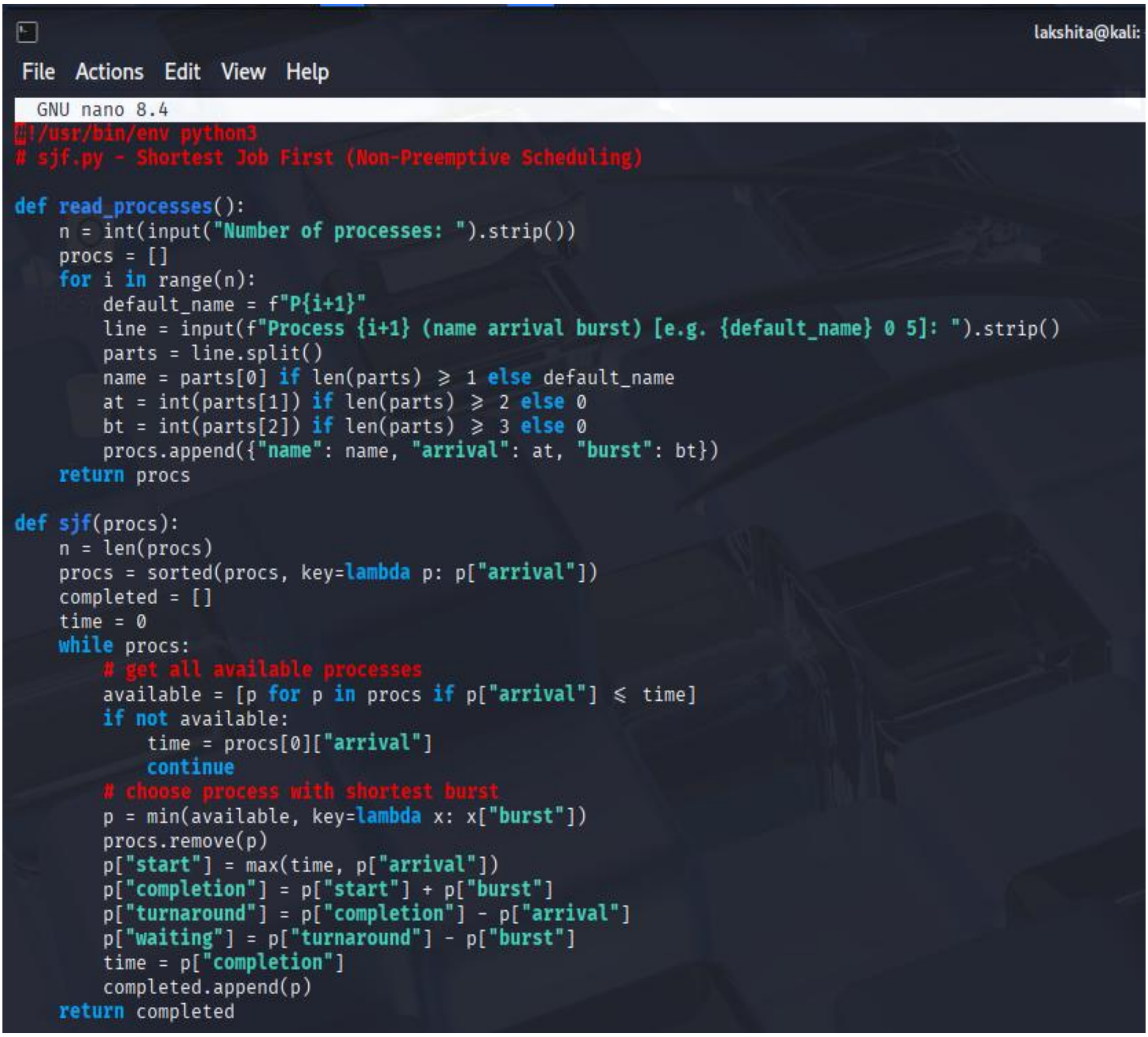
OUTPUT



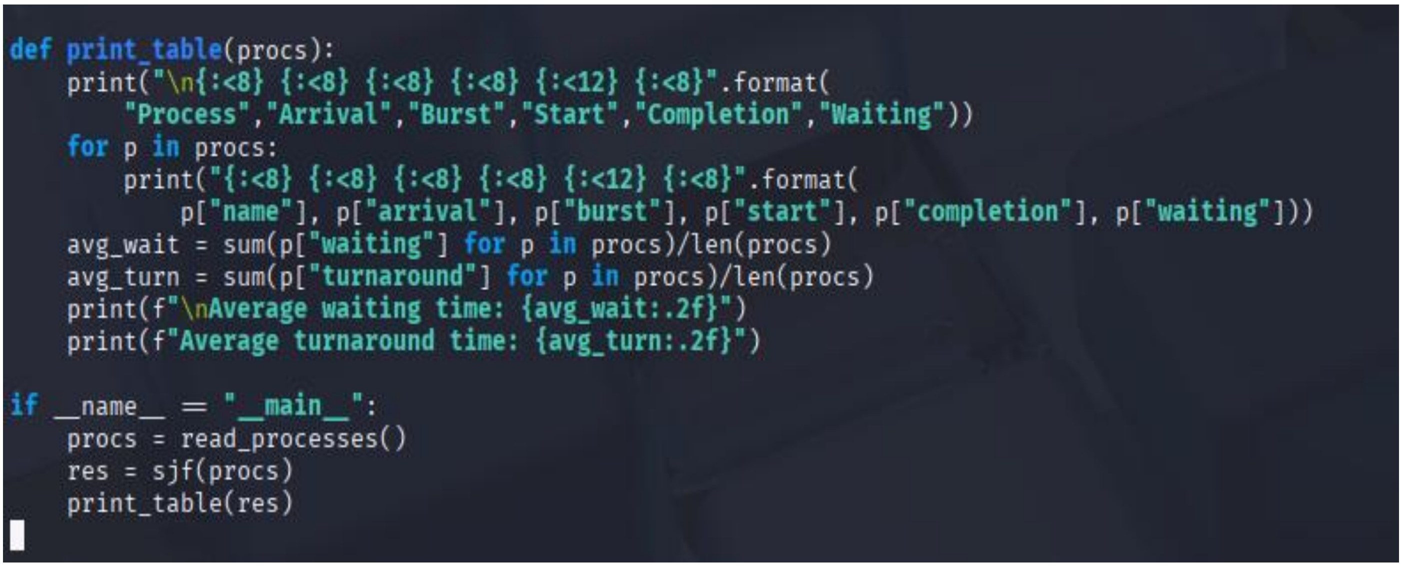
```
┌──(lakshita㉿kali)-[~/linux_process_simulation/Assignment3_Scheduling]
└─$ python3 fcfs.py
Number of processes: 3
Process 1 (name arrival burst) [e.g. P1 0 5]: P1 0 6
Process 2 (name arrival burst) [e.g. P2 0 5]: P2 1 4
Process 3 (name arrival burst) [e.g. P3 0 5]: P3 2 2

Process  Arrival  Burst    Start    Completion   Waiting
P1       0        6        0        6            0
P2       1        4        6        10           5
P3       2        2        10       12           8

Average waiting time: 4.33
Average turnaround time: 8.33
```

2. SJF (Shortest Job First)

CODE

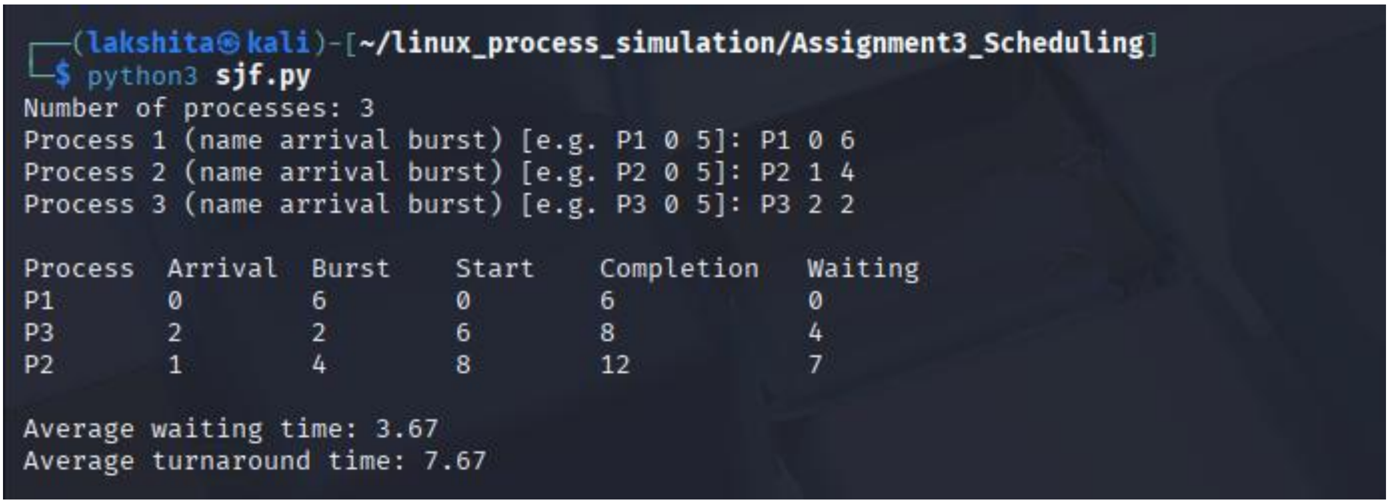


```python
#!/usr/bin/env python3
# sjf.py - Shortest Job First (Non-Preemptive Scheduling)

def read_processes():
    n = int(input("Number of processes: ").strip())
    procs = []
    for i in range(n):
        default_name = f"P{i+1}"
        line = input(f"Process {i+1} (name arrival burst) [e.g. {default_name} 0 5]: ").strip()
        parts = line.split()
        name = parts[0] if len(parts) >= 1 else default_name
        at = int(parts[1]) if len(parts) >= 2 else 0
        bt = int(parts[2]) if len(parts) >= 3 else 0
        procs.append({"name": name, "arrival": at, "burst": bt})
    return procs

def sjf(procs):
    n = len(procs)
    procs = sorted(procs, key=lambda p: p["arrival"])
    completed = []
    time = 0
    while procs:
        # get all available processes
        available = [p for p in procs if p["arrival"] <= time]
        if not available:
            time = procs[0]["arrival"]
            continue
        # choose process with shortest burst
        p = min(available, key=lambda x: x["burst"])
        procs.remove(p)
        p["start"] = max(time, p["arrival"])
        p["completion"] = p["start"] + p["burst"]
        p["turnaround"] = p["completion"] - p["arrival"]
        p["waiting"] = p["turnaround"] - p["burst"]
        time = p["completion"]
        completed.append(p)
    return completed
```



```python
def print_table(procs):
    print("\n{:<8} {:<8} {:<8} {:<8} {:<12} {:<8}".format(
        "Process","Arrival","Burst","Start","Completion","Waiting"))
    for p in procs:
        print("{:<8} {:<8} {:<8} {:<8} {:<12} {:<8}".format(
            p["name"], p["arrival"], p["burst"], p["start"], p["completion"], p["waiting"]))
    avg_wait = sum(p["waiting"] for p in procs)/len(procs)
    avg_turn = sum(p["turnaround"] for p in procs)/len(procs)
    print(f"\nAverage waiting time: {avg_wait:.2f}")
    print(f"Average turnaround time: {avg_turn:.2f}")

if __name__ == "__main__":
    procs = read_processes()
    res = sjf(procs)
    print_table(res)
```

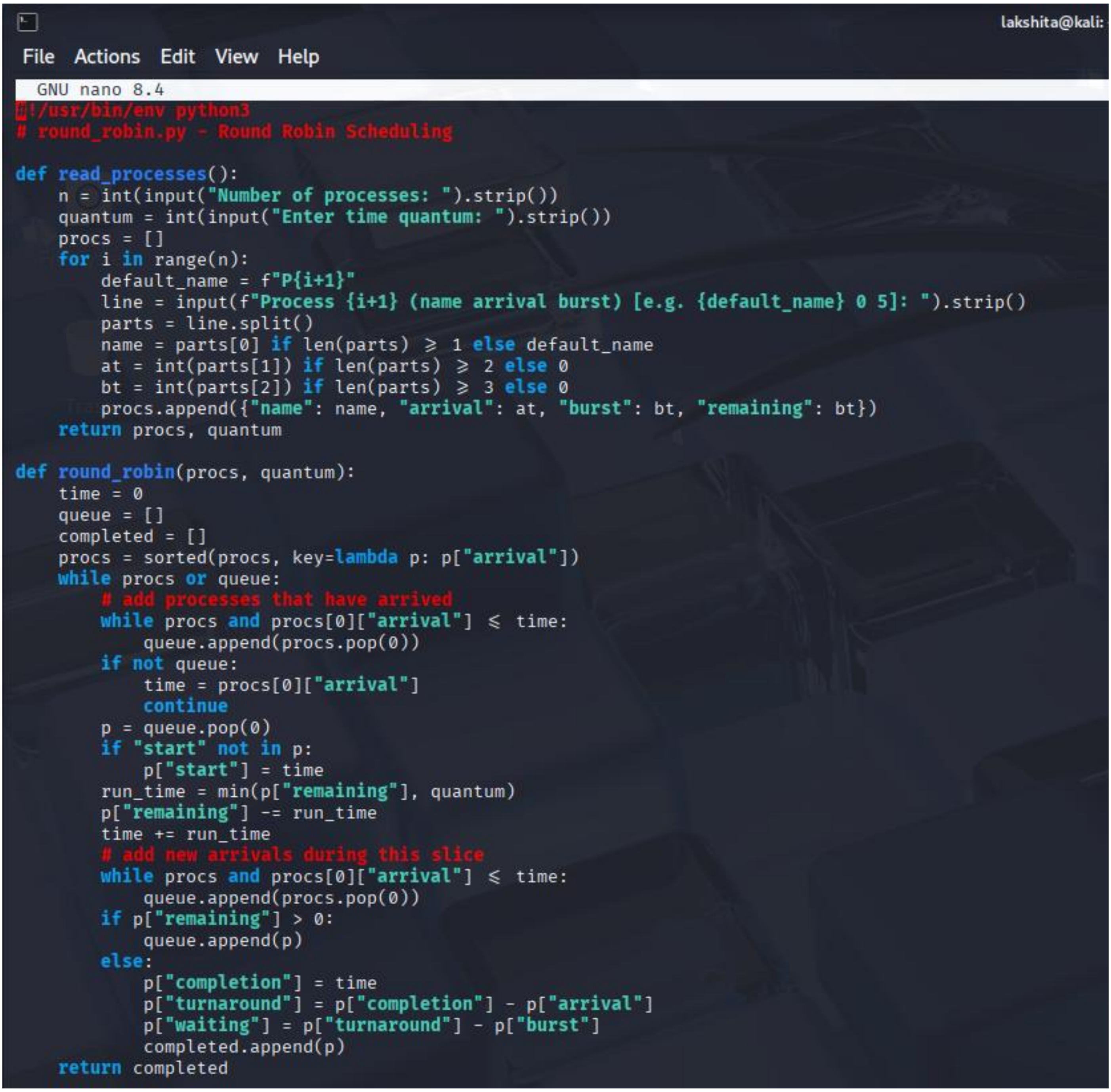
OUTPUT



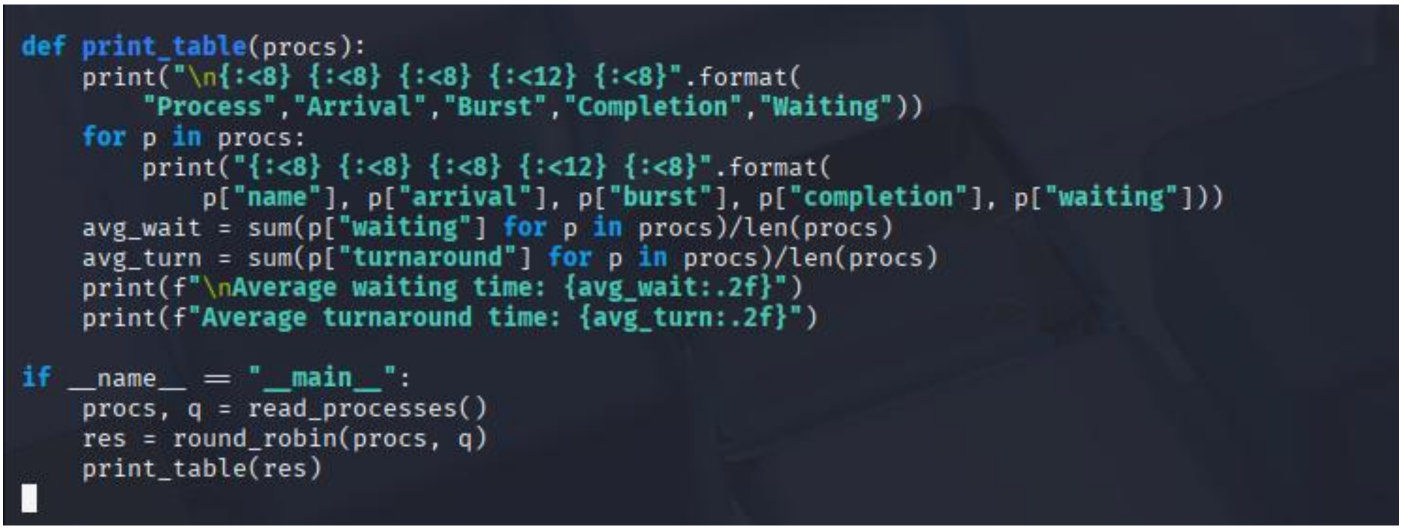
```
┌──(lakshita㉿kali)-[~/linux_process_simulation/Assignment3_Scheduling]
└─$ python3 sjf.py
Number of processes: 3
Process 1 (name arrival burst) [e.g. P1 0 5]: P1 0 6
Process 2 (name arrival burst) [e.g. P2 0 5]: P2 1 4
Process 3 (name arrival burst) [e.g. P3 0 5]: P3 2 2

Process  Arrival  Burst    Start    Completion   Waiting
P1       0        6        0        6            0
P3       2        2        6        8            4
P2       1        4        8        12           7

Average waiting time: 3.67
Average turnaround time: 7.67
```

3. Round Robin

CODE

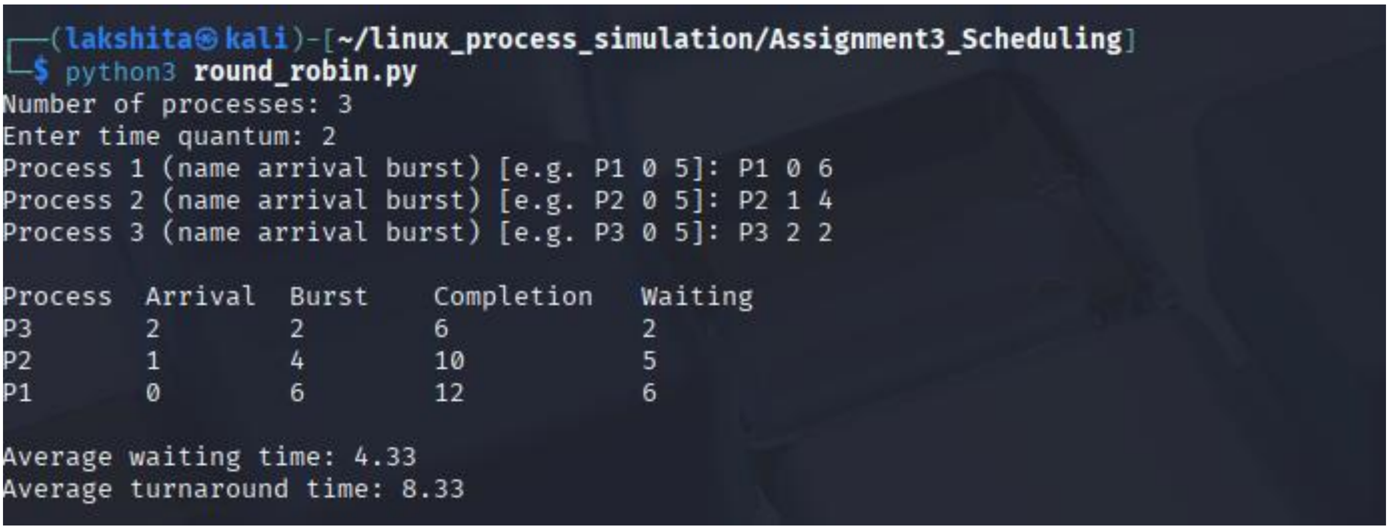


```
#!/usr/bin/env python3
# round_robin.py - Round Robin Scheduling

def read_processes():
    n = int(input("Number of processes: ").strip())
    quantum = int(input("Enter time quantum: ").strip())
    procs = []
    for i in range(n):
        default_name = f"P{i+1}"
        line = input(f"Process {i+1} (name arrival burst) [e.g. {default_name} 0 5]: ").strip()
        parts = line.split()
        name = parts[0] if len(parts) >= 1 else default_name
        at = int(parts[1]) if len(parts) >= 2 else 0
        bt = int(parts[2]) if len(parts) >= 3 else 0
        procs.append({"name": name, "arrival": at, "burst": bt, "remaining": bt})
    return procs, quantum

def round_robin(procs, quantum):
    time = 0
    queue = []
    completed = []
    procs = sorted(procs, key=lambda p: p["arrival"])
    while procs or queue:
        # add processes that have arrived
        while procs and procs[0]["arrival"] <= time:
            queue.append(procs.pop(0))
        if not queue:
            time = procs[0]["arrival"]
            continue
        p = queue.pop(0)
        if "start" not in p:
            p["start"] = time
        run_time = min(p["remaining"], quantum)
        p["remaining"] -= run_time
        time += run_time
        # add new arrivals during this slice
        while procs and procs[0]["arrival"] <= time:
            queue.append(procs.pop(0))
        if p["remaining"] > 0:
            queue.append(p)
        else:
            p["completion"] = time
            p["turnaround"] = p["completion"] - p["arrival"]
            p["waiting"] = p["turnaround"] - p["burst"]
            completed.append(p)
    return completed
```



```
def print_table(procs):
    print("\n{:<8} {:<8} {:<8} {:<12} {:<8}".format(
        "Process","Arrival","Burst","Completion","Waiting"))
    for p in procs:
        print("{:<8} {:<8} {:<8} {:<12} {:<8}".format(
            p["name"], p["arrival"], p["burst"], p["completion"], p["waiting"]))
    avg_wait = sum(p["waiting"] for p in procs)/len(procs)
    avg_turn = sum(p["turnaround"] for p in procs)/len(procs)
    print(f"\nAverage waiting time: {avg_wait:.2f}")
    print(f"Average turnaround time: {avg_turn:.2f}")

if __name__ == "__main__":
    procs, q = read_processes()
    res = round_robin(procs, q)
    print_table(res)
```

OUTPUT



```
┌──(lakshita㉿kali)-[~/linux_process_simulation/Assignment3_Scheduling]
└─$ python3 round_robin.py
Number of processes: 3
Enter time quantum: 2
Process 1 (name arrival burst) [e.g. P1 0 5]: P1 0 6
Process 2 (name arrival burst) [e.g. P2 0 5]: P2 1 4
Process 3 (name arrival burst) [e.g. P3 0 5]: P3 2 2

Process  Arrival  Burst    Completion   Waiting
P3       2        2        6            2
P2       1        4        10           5
P1       0        6        12           6

Average waiting time: 4.33
Average turnaround time: 8.33
```