OPERATING SYSTEMS PROJECT REPORT

on

IMPLEMENTATION OF SCHEDULING ALGORITHMS

by

Sathvika

Name – 2

Name – 2

Table of Contents:

1. Introduction

This project is centered on implementing key CPU scheduling algorithms in Python, including First-Come-First Serve (FCFS), Round Robin (RR), Shortest Process Next (SPN), Shortest Remaining Time (SRT), Highest Response Ratio Next (HRRN), and Multilevel Feedback Queue (MFQ). Designed to be interactive and user-friendly, it utilizes Python's capabilities to handle complex scheduling logic and Streamlit for creating an accessible graphical user interface (GUI). This combination allows for a dynamic exploration of algorithm behaviors, where users can input various process loads and immediately visualize the outcomes. In addition, a new algorithm has also been developed which is named as Adaptive Priority Scheduling Algorithm (APSA). A detailed analysis is also presented at the later section of report which highlights about the average waiting time and average turnaround time of various input types by various algorithms implemented as a part of basic and advance requirements.

2. Basic Requirements:

All CPU scheduling algorithms has been implemented in different files and placed in algorithms folder where they can be imported with a single import statement `from algorithms import fcfs as fcfs`. Each algorithm is designed to handle process scheduling in distinct ways, catering to different requirements and scenarios in an operating system.

Static Inputs

For testing, a set of static inputs is used: Arrival Times - [0, 1, 3, 4, 7] and Service Times - [10, 2, 5, 9, 7]. These inputs are consistently applied across all algorithms to evaluate and compare their performance.

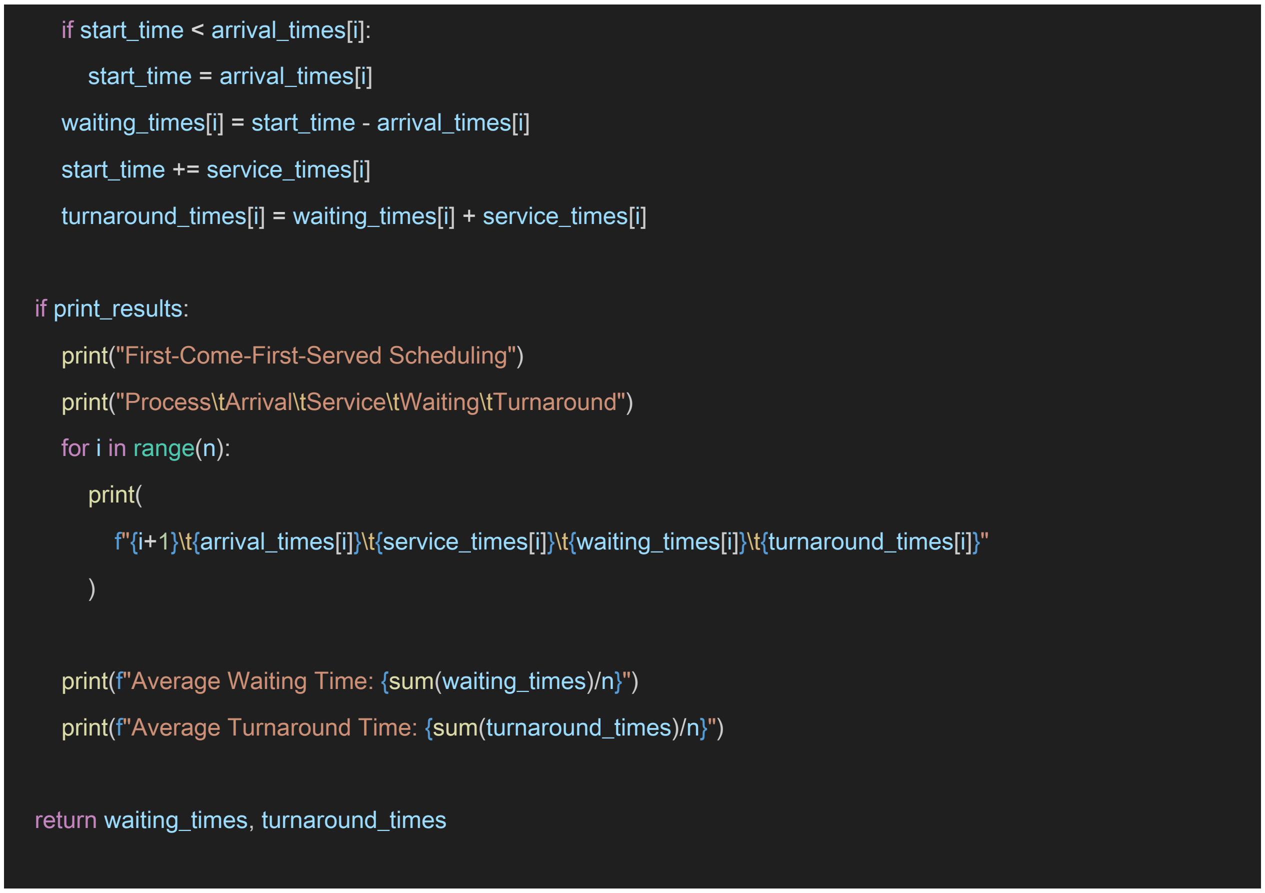
Print Results Feature

Each algorithm includes a **print_results** boolean parameter. When set to **True**, the algorithms output detailed information, including the waiting and turnaround times for each process.

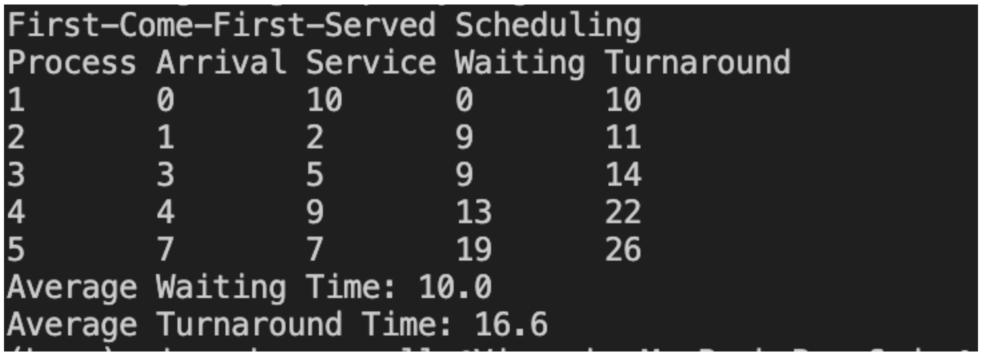
Below are the implementation details of various algorithms implemented as part of the static requirements.

a) First Come First Serve(FCFS) : FCFS is the simplest scheduling method, operating on a first-come, first-serve basis. Processes are executed in the order they arrive, without preemption. It's straightforward but can lead to long waiting times for processes arriving later. The function signature is **fcfs(arrival_times, service_times, print_results)**. It outputs the process execution sequence based on arrival times. Below is the code implementation of the algorithm.

```
"""
First-Come-First-Served Scheduling Algorithm
"""


__all__ = ["fcfs"]


def fcfs(arrival_times, service_times, print_results=False):
    """
    First-Come-First-Served Scheduling Algorithm

    This function implements the First-Come-First-Served (FCFS) scheduling algorithm.
    It takes a list of arrival times and service times of processes as input and returns
    the waiting times and turnaround times of each process.

    Parameters:
    arrival_times (list): List of arrival times of processes.
    service_times (list): List of service (burst) times of processes.
    print_results (bool): If True, prints the scheduling details.

    Returns:
    waiting_times (list): List of waiting times of each process.
    turnaround_times (list): List of turnaround times of each process.
    """
    n = len(arrival_times)
    waiting_times = [0] * n
    turnaround_times = [0] * n
    start_time = 0

    for i in range(n):
```



```
        if start_time < arrival_times[i]:
            start_time = arrival_times[i]
        waiting_times[i] = start_time - arrival_times[i]
        start_time += service_times[i]
        turnaround_times[i] = waiting_times[i] + service_times[i]

    if print_results:
        print("First-Come-First-Served Scheduling")
        print("Process\tArrival\tService\tWaiting\tTurnaround")
        for i in range(n):
            print(
                f"{i+1}\t{arrival_times[i]}\t{service_times[i]}\t{waiting_times[i]}\t{turnaround_times[i]}"
            )

        print(f"Average Waiting Time: {sum(waiting_times)/n}")
        print(f"Average Turnaround Time: {sum(turnaround_times)/n}")

    return waiting_times, turnaround_times
```



```
First-Come-First-Served Scheduling
Process Arrival Service Waiting Turnaround
1       0       10      0       10
2       1       2       9       11
3       3       5       9       14
4       4       9       13      22
5       7       7       19      26
Average Waiting Time: 10.0
Average Turnaround Time: 16.6
```

Figure 2.1 FCFS

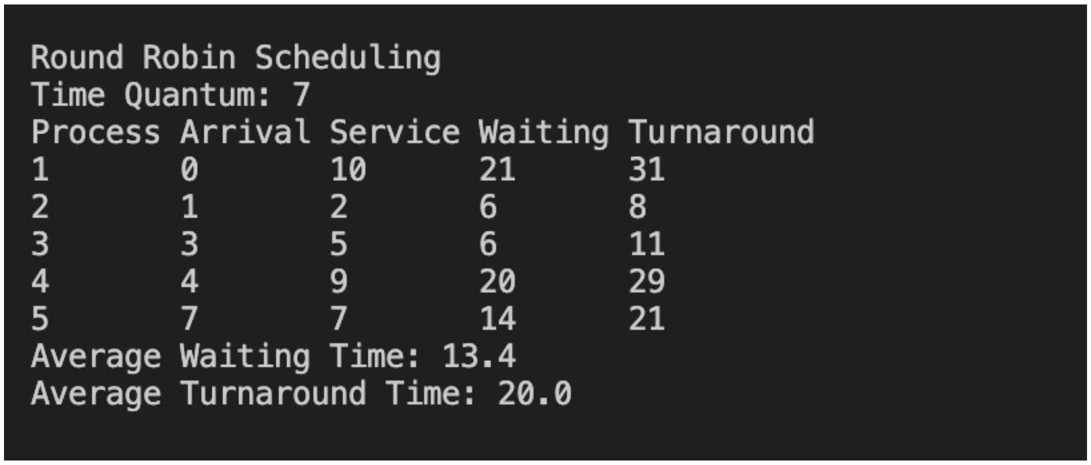
b) Round Robin (RR): RR assigns a fixed time slice to each process in a cyclic order. If a process doesn't complete within its time slice, it's moved to the end of the queue. This ensures a more equitable CPU allocation. The signature is **round_robin(arrival_times, service_times, time_quantum, print_results)**. It's designed to ensure all processes get an equal share of the CPU.

Time quantum = 7

Below is the implementation code for Round Robin Algorithm

```
"""
Round Robin Scheduling Algorithm
"""

__all__ = ["round_robin"]


def round_robin(arrival_times, service_times, quantum, print_results=False):
    """
    Round Robin Scheduling Algorithm

    Parameters:
    arrival_times (list): List of arrival times of processes.
    service_times (list): List of service (burst) times of processes.
    quantum (int): Time quantum for the round-robin scheduling.
    print_results (bool): If True, prints the scheduling details.

    Returns:
    waiting_times (list): List of waiting times for each process.
    turnaround_times (list): List of turnaround times for each process.
    """
    n = len(arrival_times)
    remaining_times = list(service_times)
    waiting_times = [0] * n
    turnaround_times = [0] * n
    t = 0  # Current time

    while True:
        done = True
        for i in range(n):
            if remaining_times[i] > 0:
                done = False
                if remaining_times[i] > quantum:
                    t += quantum
                    remaining_times[i] -= quantum
                else:
                    t += remaining_times[i]
                    waiting_times[i] = t - service_times[i] - arrival_times[i]
                    remaining_times[i] = 0

        if done:
            break

    for i in range(n):
        turnaround_times[i] = service_times[i] + waiting_times[i]

    if print_results:
        print("Round Robin Scheduling")
        print(f"Time Quantum: {quantum}")
        print("Process\tArrival\tService\tWaiting\tTurnaround")
        for i in range(n):
            print(
                f"{i+1}\t{arrival_times[i]}\t{service_times[i]}\t{waiting_times[i]}\t{turnaround_times[i]}"
            )

        print(f"Average Waiting Time: {sum(waiting_times)/n}")
        print(f"Average Turnaround Time: {sum(turnaround_times)/n}")

    return waiting_times, turnaround_times
```

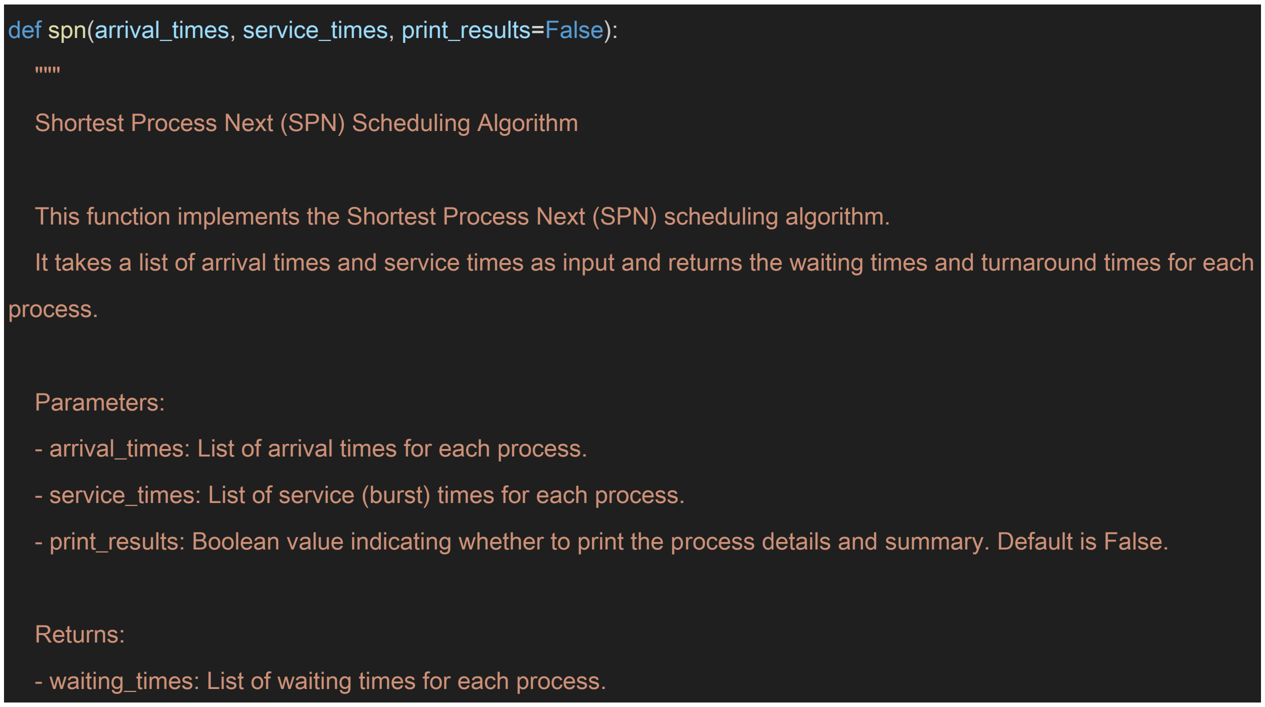
Execution Output:



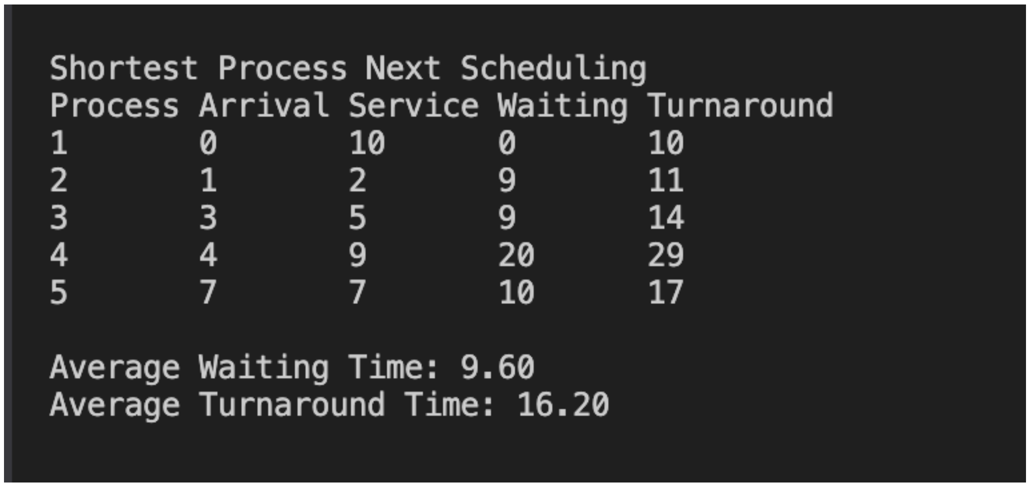
```
Round Robin Scheduling
Time Quantum: 7
Process Arrival Service Waiting Turnaround
1       0       10      21      31
2       1       2       6       8
3       3       5       6       11
4       4       9       20      29
5       7       7       14      21
Average Waiting Time: 13.4
Average Turnaround Time: 20.0
```

Figure 2.2 RR

c) Shortest Process Next (SPN): SPN selects the process with the shortest estimated run time for execution next. It's non-preemptive, favoring shorter jobs and reducing overall average waiting time but can cause longer jobs to wait excessively. The function is **spn(arrival_times, service_times, print_results)**, which favors shorter jobs to reduce the average waiting time.



```
def spn(arrival_times, service_times, print_results=False):
    """
    Shortest Process Next (SPN) Scheduling Algorithm

    This function implements the Shortest Process Next (SPN) scheduling algorithm.
    It takes a list of arrival times and service times as input and returns the waiting times and turnaround times for each
process.

    Parameters:
    - arrival_times: List of arrival times for each process.
    - service_times: List of service (burst) times for each process.
    - print_results: Boolean value indicating whether to print the process details and summary. Default is False.

    Returns:
    - waiting_times: List of waiting times for each process.
    - turnaround_times: List of turnaround times for each process.
    """
    n = len(arrival_times)
    waiting_times = [0] * n
    finish_times = [0] * n
    service_times_remaining = list(service_times)
    time = 0
    processes = set(range(n))

    while processes:
        available_processes = [i for i in processes if arrival_times[i] <= time]
        if not available_processes:
            time += 1
            continue

        shortest_process = min(
            available_processes, key=lambda i: service_times_remaining[i]
        )
        processes.remove(shortest_process)
        waiting_times[shortest_process] = time - arrival_times[shortest_process]
        time += service_times_remaining[shortest_process]
        finish_times[shortest_process] = time

    turnaround_times = [finish_times[i] - arrival_times[i] for i in range(n)]

    if print_results:
        print("Shortest Process Next Scheduling")
        print("Process\tArrival\tService\tWaiting\tTurnaround")
        for i in range(n):
            print(
                f"{i + 1}\t{arrival_times[i]}\t{service_times[i]}\t{waiting_times[i]}\t{turnaround_times[i]}"
            )
        print(f"\nAverage Waiting Time: {sum(waiting_times) / n:.2f}")
        print(f"Average Turnaround Time: {sum(turnaround_times) / n:.2f}")

    return waiting_times, turnaround_times
```



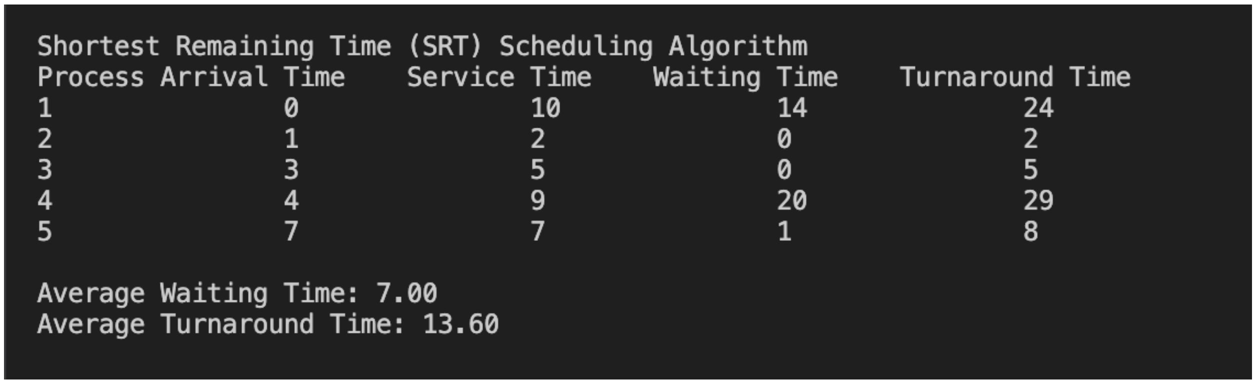
```
Shortest Process Next Scheduling
Process Arrival Service Waiting Turnaround
1       0       10      0       10
2       1       2       9       11
3       3       5       9       14
4       4       9       20      29
5       7       7       10      17

Average Waiting Time: 9.60
Average Turnaround Time: 16.20
```

Figure 2.3 SPN

d) Shortest Remaining Time (SRT): SRT is like SPN but preemptive. It selects the process with the shortest remaining time to completion. This approach often results in lower average waiting times, especially for short jobs. The signature is **srt(arrival_times, service_times, print_results)**, ideally suited for quickly executing short jobs.

```
def srt(arrival_times, service_times, print_results=False):
    """
    Shortest Remaining Time (SRT) Scheduling Algorithm
    Parameters:
    - arrival_times: List of arrival times
    - service_times: List of service (burst) times
    - print_results: Boolean, if True, print the process details in table format

    Returns:
    - waiting_times: List of waiting times for each process
    - turnaround_times: List of turnaround times for each process
    """
    n = len(arrival_times)
    waiting_times = [0] * n
    finish_times = [0] * n
    remaining_times = list(service_times)
```

```
time = 0
completed = 0

while completed != n:
    # Find process with minimum remaining time
    shortest = min(
        [
            i
            for i in range(n)
            if arrival_times[i] <= time and remaining_times[i] > 0
        ],
        key=lambda x: remaining_times[x],
        default=-1,
    )

    if shortest == -1:
        time += 1
        continue

    remaining_times[shortest] -= 1
    time += 1

    if remaining_times[shortest] == 0:
        completed += 1
        finish_times[shortest] = time
        waiting_times[shortest] = (
            finish_times[shortest]
            - arrival_times[shortest]
            - service_times[shortest]
        )

turnaround_times = [finish_times[i] - arrival_times[i] for i in range(n)]

if print_results:
    print("\nShortest Remaining Time (SRT) Scheduling Algorithm")
    print("Process\tArrival Time\tService Time\tWaiting Time\tTurnaround Time")
    for i in range(n):
        print(
            f"{i + 1}\t\t{arrival_times[i]}\t\t{service_times[i]}\t\t{waiting_times[i]}\t\t{turnaround_times[i]}"
        )
    print(f"\nAverage Waiting Time: {sum(waiting_times) / n:.2f}")
    print(f"Average Turnaround Time: {sum(turnaround_times) / n:.2f}")

  return waiting_times, turnaround_times
```



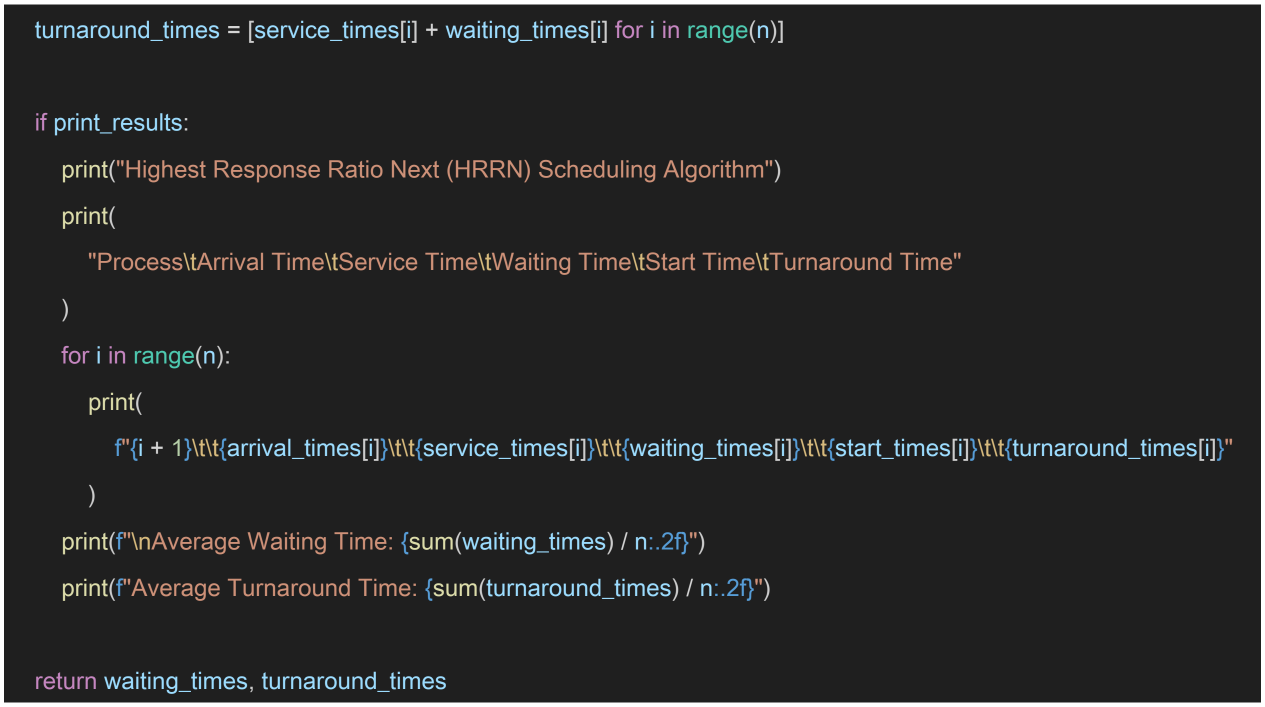
```
Shortest Remaining Time (SRT) Scheduling Algorithm
Process Arrival Time    Service Time    Waiting Time    Turnaround Time
1               0               10              14              24
2               1               2               0               2
3               3               5               0               5
4               4               9               20              29
5               7               7               1               8

Average Waiting Time: 7.00
Average Turnaround Time: 13.60
```

Figure 2.4 SRT

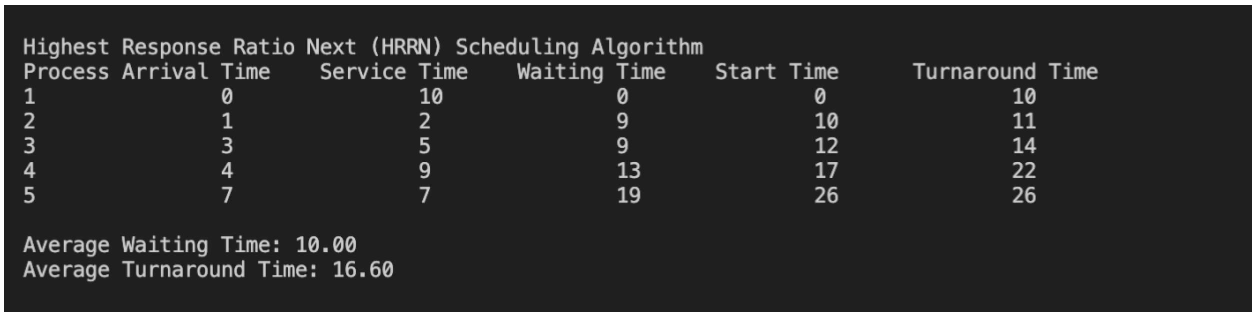
e) Highest Response Ratio Next (HRRN): HRRN calculates a response ratio for each process and schedules the one with the highest ratio next. This ratio is based on waiting time and service time, balancing the needs of both short and long processes. The function **hrrn(arrival_times, service_times, print_results)** balances between shorter and longer processes, aiming for optimal turnaround time.

```
def hrrn(arrival_times, service_times, print_results=False):
  """
  Highest Response Ratio Next (HRRN) Scheduling Algorithm

  This function implements the Highest Response Ratio Next (HRRN) scheduling algorithm.
  It schedules the processes based on their response ratio, which is calculated as the ratio
  of the sum of the waiting time and the service time to the service time.
```

```
Parameters:
- arrival_times: List of arrival times for each process.
- service_times: List of service (burst) times for each process.
- print_results: Boolean value indicating whether to print the process details in table format.

Returns:
- waiting_times: List of waiting times for each process.
- turnaround_times: List of turnaround times for each process.
"""
n = len(arrival_times)
waiting_times = [0] * n
finish_times = [0] * n
start_times = [0] * n
remaining_times = list(service_times)
time = 0
processes = set(range(n))

while processes:
    available_processes = [
        (i, ((time - arrival_times[i]) + service_times[i]) / service_times[i])
        for i in processes
        if arrival_times[i] <= time
    ]
    if not available_processes:
        time += 1
        continue

    # Process with the highest response ratio
    next_process = max(available_processes, key=lambda x: x[1])[0]
    processes.remove(next_process)
    waiting_times[next_process] = time - arrival_times[next_process]
    time += service_times[next_process]
    finish_times[next_process] = time
    start_times[next_process] = (
        waiting_times[next_process] + arrival_times[next_process]
    )
```



```
turnaround_times = [service_times[i] + waiting_times[i] for i in range(n)]

if print_results:
    print("Highest Response Ratio Next (HRRN) Scheduling Algorithm")
    print(
        "Process\tArrival Time\tService Time\tWaiting Time\tStart Time\tTurnaround Time"
    )
    for i in range(n):
        print(
            f"{i + 1}\t\t{arrival_times[i]}\t\t{service_times[i]}\t\t{waiting_times[i]}\t\t{start_times[i]}\t\t{turnaround_times[i]}"
        )
    print(f"\nAverage Waiting Time: {sum(waiting_times) / n:.2f}")
    print(f"Average Turnaround Time: {sum(turnaround_times) / n:.2f}")

return waiting_times, turnaround_times
```



```
Highest Response Ratio Next (HRRN) Scheduling Algorithm
Process Arrival Time    Service Time    Waiting Time    Start Time      Turnaround Time
1               0               10              0               0               10
2               1               2               9               10              11
3               3               5               9               12              14
4               4               9               13              17              22
5               7               7               19              26              26

Average Waiting Time: 10.00
Average Turnaround Time: 16.60
```

Figure 2.5 HRRN

f) Multilevel Feedback Queue (MFQ): MFQ uses multiple queues with different scheduling criteria. Processes move between queues based on their age or execution characteristics. It's designed to dynamically prioritize tasks, offering flexibility and responsiveness. The signature is **mfq(arrival_times, service_times, t1, t2, print_results)**. It dynamically adjusts a process's priority based on its behavior and requirements.

Below are the important implementation details of Multilevel Feedback Queue:

It has three queues:

1. Queue 1 (highest priority): Round Robin with t1 time quanta.
2. Queue 2: Round Robin with t2 time quanta.
3. Queue 3 (lowest priority): First-Come-First-Serve (FCFS).

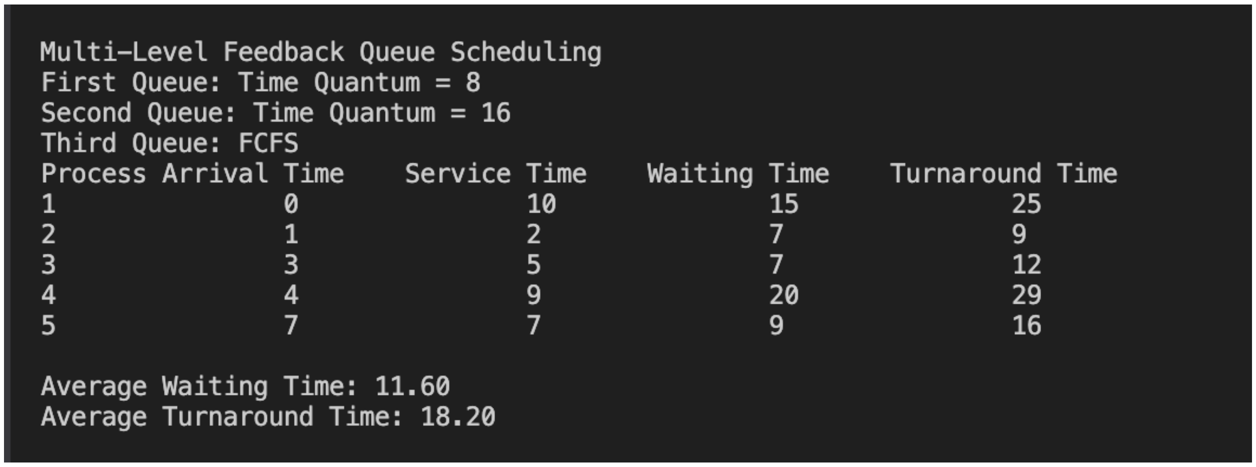
The algorithm works as follows:

- Processes are initially placed in the highest priority queue (Queue 1).
- If a process uses up all its time quanta in Queue 1, it's moved to Queue 2.
- Similarly, if it uses up all its time in Queue 2, it's moved to Queue 3.
- In Queue 3, processes are executed on a FCFS basis without time quantum.
- If a new process arrives in a higher priority queue, it preempts the currently running process in a lower priority queue.

```python
class Process:
    def __init__(self, pid, arrival_time, burst_time):
        self.pid = pid
        self.arrival_time = arrival_time
        self.burst_time = burst_time
        self.remaining_time = burst_time
        self.waiting_time = 0
        self.completed = False
        self.priority = None


def get_process_by_pid(processes, pid):
    for process in processes:
        if process.pid == pid:
            return process
    return None


def mlfq(arrival_times, service_times, t1, t2, print_results=False):
    """
```

```
Implements the Multi-Level Feedback Queue (MLFQ) scheduling algorithm.

Args:
    arrival_times (list): List of arrival times for each process.
    service_times (list): List of service times for each process.
    t1 (int): Time quantum for the first queue.
    t2 (int): Time quantum for the second queue.
    print_results (bool, optional): Whether to print the scheduling results. Defaults to False.

Returns:
    tuple: A tuple containing the waiting times and turnaround times for each process.
"""
n = len(arrival_times)
processes = [Process(i, arrival_times[i], service_times[i]) for i in range(n)]
queues = [Queue() for _ in range(3)]
time_quantum = [t1, t2, float("inf")]
current_time = 0
current_process = None
gantt_chart = []

while any(p.remaining_time > 0 for p in processes):
    for process in processes:
        if (
            process.arrival_time <= current_time
            and not process.completed
            and process.priority is None
        ):
            queues[0].put(process.pid)
            process.priority = 0

    for i in range(len(queues)):
        if not queues[i].empty() and (
            current_process is None or i < current_process.priority
        ):
            if current_process:
                queues[current_process.priority].put(current_process.pid)
            pid = queues[i].get()
            current_process = get_process_by_pid(processes, pid)
            break

        if current_process:
            current_process.remaining_time -= 1
            time_quantum[current_process.priority] -= 1

            if current_process.remaining_time == 0:
                current_process.waiting_time = (
                    current_time
                    - current_process.arrival_time
                    - current_process.burst_time
                    + 1
                )
                current_process.completed = True
                current_process = None
            elif time_quantum[current_process.priority] == 0:
                if current_process.priority < 2:
                    queues[current_process.priority + 1].put(current_process.pid)
                    current_process.priority += 1
                current_process = None
                time_quantum = [8, 16, float("inf")]

        current_time += 1

    if print_results:
        print("Multi-Level Feedback Queue Scheduling")
        print("First Queue: Time Quantum = 8")
        print("Second Queue: Time Quantum = 16")
        print("Third Queue: FCFS")
        print("Process\tArrival Time\tService Time\tWaiting Time\tTurnaround Time")
        for i in range(n):
            print(
                f"{i + 1}\t\t{arrival_times[i]}\t\t{service_times[i]}\t\t{processes[i].waiting_time}\t\t{processes[i].waiting_time + 
service_times[i]}"
            )
    waiting_times = [p.waiting_time for p in processes]
```

```
turnaround_times = [p.waiting_time + p.burst_time for p in processes]

print(f"\nAverage Waiting Time: {sum(waiting_times) / n:.2f}")

print(f"Average Turnaround Time: {sum(turnaround_times) / n:.2f}")

return waiting_times, turnaround_times
```



```
Multi-Level Feedback Queue Scheduling
First Queue: Time Quantum = 8
Second Queue: Time Quantum = 16
Third Queue: FCFS
Process Arrival Time    Service Time    Waiting Time    Turnaround Time
1       0               10              15              25
2       1               2               7               9
3       3               5               7               12
4       4               9               20              29
5       7               7               9               16

Average Waiting Time: 11.60
Average Turnaround Time: 18.20
```

Figure 2.6 Multi Level Feedback Queue

Below is the code to run all the algorithms in a single click using the static inputs.

```
"""
This file contains static inputs for testing the algorithms which covers the
basic requirements of the project.
"""


from algorithms.fcfs import fcfs
from algorithms.hrrn import hrrn
from algorithms.srt import srt
from algorithms.spn import spn
from algorithms.rr import round_robin
from algorithms.mfq import mlfq as mfq
from algorithms.custom import apsa


# define static inputs
arrival_times = [0, 1, 3, 4, 7]
service_times = [10, 2, 5, 9, 7]


# time quanta for some algorithms
```

```
time_quanta = 7

# now call all of the  algorithms with print_results=True
fcfs(arrival_times, service_times, print_results=True)
print("\n")
round_robin(arrival_times, service_times, time_quanta, print_results=True)
print("\n")
spn(arrival_times, service_times, print_results=True)
print("\n")
srt(arrival_times, service_times, print_results=True)
print("\n")
hrrn(arrival_times, service_times, print_results=True)
print("\n")
mfq(arrival_times, service_times, 8, 16, print_results=True)
print("\n")
apsa(arrival_times, service_times, 0.5, 10, print_results=True)
```

3. Advanced Requirements:

A) New Algorithm:

Designing a new scheduling algorithm can be quite a challenge, but it's also an opportunity to be creative and innovate. For this task, I have come up with a new algorithm that combines aspects of existing algorithms to cater different types of processes. Below are the algorithm details along with the code implementation.

The Adaptive Priority Scheduling Algorithm is a preemptive scheduler that dynamically adjusts the priority of processes based on their burst time, arrival time, and the amount of time they have already been waiting. It aims to minimize the average waiting time and turnaround time, especially for I/O-bound and short tasks, while also ensuring that CPU-bound tasks are processed efficiently.

Concept:

- Each process is assigned an initial priority based on its estimated burst time and arrival time.
- The priority is recalculated at regular intervals and after every process execution.

- Priorities are adjusted to ensure that shorter and I/O-bound processes are given preference, while also preventing starvation of longer, CPU-bound processes.

Initial Priority Calculation:

- Shorter processes and processes that have just arrived are given higher priority.
- The formula for priority is : **Priority = 1 / (Burst Time + Arrival Time Factor)**
- The Arrival Time Factor ensures that newly arrived processes get a boost in priority.

Dynamic Priority Adjustment:

- After each execution cycle, the priority of the waiting processes is recalculated.
- If a process has been waiting, its priority is increased, which can be determined by the formula: **Priority += Waiting Time Factor**
- If a process exhausts its time slice without completing, its priority is slightly reduced to give way to other processes.

Execution:

- At each decision point, the scheduler selects the process with the highest priority.
- If a new process arrives with higher priority than the currently running process, a context switch occurs.

Starvation Prevention:

- If a process's waiting time exceeds a certain threshold, its priority is boosted significantly to ensure it gets CPU time.
- Every process is guaranteed to get a minimum amount of CPU time after a certain waiting period.

Below is the python implementation of the above algorithm

```
def apsa(
    arrival_times,
    burst_times,
    WAITING_TIME_FACTOR,
    ARRIVAL_TIME_FACTOR,
    print_results=False,
):
    """
```

```
Implements the Adaptive Priority Scheduling Algorithm (APSA).

Args:
    arrival_times (list): List of arrival times for each process.
    burst_times (list): List of burst times for each process.
    WAITING_TIME_FACTOR (float): Factor to determine the waiting time boost for processes.
    ARRIVAL_TIME_FACTOR (float): Factor to determine the arrival time boost for processes.
    print_results (bool, optional): Flag to print the results. Defaults to False.

Returns:
    tuple: A tuple containing the waiting times and turnaround times for each process.
"""
num_processes = len(arrival_times)
waiting_times = [0] * num_processes
turnaround_times = [0] * num_processes
priorities = [1 / (burst + arrival_times[i]) for i, burst in enumerate(burst_times)]
completed = [False] * num_processes
time = 0
queue = []

while not all(completed):
    # Add processes to the queue based on arrival time
    queue.extend(
        [
            i
            for i, arrival in enumerate(arrival_times)
            if arrival <= time and not completed[i] and i not in queue
        ]
    )

    # Recalculate priorities for waiting processes
    for i in queue:
        if not completed[i]:
            waiting_times[i] = time - arrival_times[i]
            # Boost priority for processes that have been waiting too long
            waiting_boost = max(waiting_times[i] * WAITING_TIME_FACTOR, 1)
            priorities[i] = (
                1 / (burst_times[i] + arrival_times[i] / ARRIVAL_TIME_FACTOR)
                + waiting_boost
            )

    # Select process with highest priority
    if queue:
        current_process = max(queue, key=lambda i: priorities[i])
        queue.remove(current_process)

        # Process execution simulation for one time unit
        burst_times[current_process] -= 1
        time += 1

        # Check if the process is completed
        if burst_times[current_process] <= 0:
            completed[current_process] = True
            turnaround_times[current_process] = (
                time - arrival_times[current_process]
            )

    else:
        time += 1
if print_results:
    print("Adaptive Priority Scheduling Algorithm (APSA):")
    print("WAITING_TIME_FACTOR:", WAITING_TIME_FACTOR)
    print("ARRIVAL_TIME_FACTOR:", ARRIVAL_TIME_FACTOR)
    print("Process\tArrival\tBurst\tWaiting\tTurnaround")
    for i in range(num_processes):
        print(
            f"{i+1}\t{arrival_times[i]}\t{burst_times[i]}\t{waiting_times[i]}\t{turnaround_times[i]}"
        )
    print("Average Waiting Time:", sum(waiting_times) / num_processes)
    print("Average Turnaround Time:", sum(turnaround_times) / num_processes)

return waiting_times, turnaround_times
```

B) **Dynamic Inputs:**

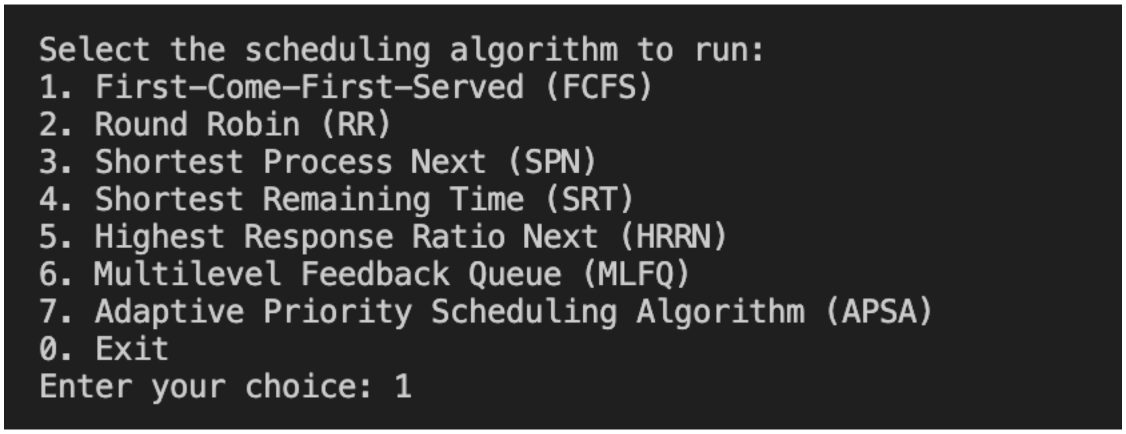
The implementation was enhanced with the capability to accept dynamic inputs, allowing real-time simulation and testing of the algorithms with varying process loads. For every input taken, assertions are made to make sure they are given in right format.

Below is the code that presents the user with all the available algorithms and then takes the inputs from the user interactively.

```
from algorithms.fcfs import fcfs
from algorithms.hrrn import hrrn
from algorithms.srt import srt
from algorithms.spn import spn
from algorithms.rr import round_robin
from algorithms.mfq import mlfq as mfq
from algorithms.custom import apsa


def get_positive_int(prompt):
    while True:
        try:
            value = int(input(prompt).strip())
            if value <= 0:
                raise ValueError("The number must be positive.")
            return value
        except ValueError as ve:
            print(f"Invalid input: {ve}")


def get_positive_float(prompt):
    while True:
        try:
            value = float(input(prompt).strip())
            if value <= 0:
                raise ValueError("The number must be positive.")
            return value
        except ValueError as ve:
            print(f"Invalid input: {ve}")


def get_input(prompt, count):
    while True:
        try:
            values = list(map(int, input(prompt).strip().split()))
            if len(values) != count:
                raise ValueError(f"Exactly {count} numbers are required.")
            if any(v < 0 for v in values):
                raise ValueError("Negative numbers are not allowed.")
            return values
        except ValueError as ve:
            print(f"Invalid input: {ve}")


def run_algorithm(algo, name, num_processes, is_rr=False, is_mfq=False):
    arrival_times = get_input(
        "Enter the processes' arrival times separated by space: ", num_processes
    )
    service_times = get_input(
        "Enter the processes' service times separated by space: ", num_processes
    )

    if is_rr:
        time_quantum = get_positive_int("Enter time quantum for Round Robin: ")
        algo(arrival_times, service_times, time_quantum, print_results=True)
    elif is_mfq:
        t1 = get_positive_int("Enter time quantum t1 for MLFQ: ")
        t2 = get_positive_int("Enter time quantum t2 for MLFQ: ")
        algo(arrival_times, service_times, t1, t2, print_results=True)
    elif "APSA" in name:
        waiting_time_factor = get_positive_float(
            "Enter the waiting time factor for APSA: "
        )
        arrival_time_factor = get_positive_int(
            "Enter the arrival time factor for APSA: "
        )
        algo(
            arrival_times,
            service_times,
            waiting_time_factor,
            arrival_time_factor,
            print_results=True,
        )
    else:
        algo(arrival_times, service_times, print_results=True)


def select_algorithm():
    algorithms = {
        "1": ("First-Come-First-Served (FCFS)", fcfs, False, False),
        "2": ("Round Robin (RR)", round_robin, True, False),
        "3": ("Shortest Process Next (SPN)", spn, False, False),
        "4": ("Shortest Remaining Time (SRT)", srt, False, False),
        "5": ("Highest Response Ratio Next (HRRN)", hrrn, False, False),
        "6": ("Multilevel Feedback Queue (MLFQ)", mfq, False, True),
        "7": ("Adaptive Priority Scheduling Algorithm (APSA)", apsa, False, False),
    }

    while True:
        print("\nSelect the scheduling algorithm to run:")
        for key, (name, _, _, _) in algorithms.items():
            print(f"{key}. {name}")
        print("0. Exit")

        choice = input("Enter your choice: ").strip()
        if choice == "0":
            break

        if choice in algorithms:
            name, algo, is_rr, is_mfq = algorithms[choice]
            print(f"\n{name}:")
            num_processes = get_positive_int("Enter the number of processes: ")
            run_algorithm(algo, name, num_processes, is_rr, is_mfq)
        else:
            print("Invalid selection. Please try again.")


if __name__ == "__main__":
    select_algorithm()
```

Above dynamic input feature of this implementation enhances the project significantly. It allows real-time simulation and testing with various process loads. Each input is thoroughly checked to ensure it's in the correct format.

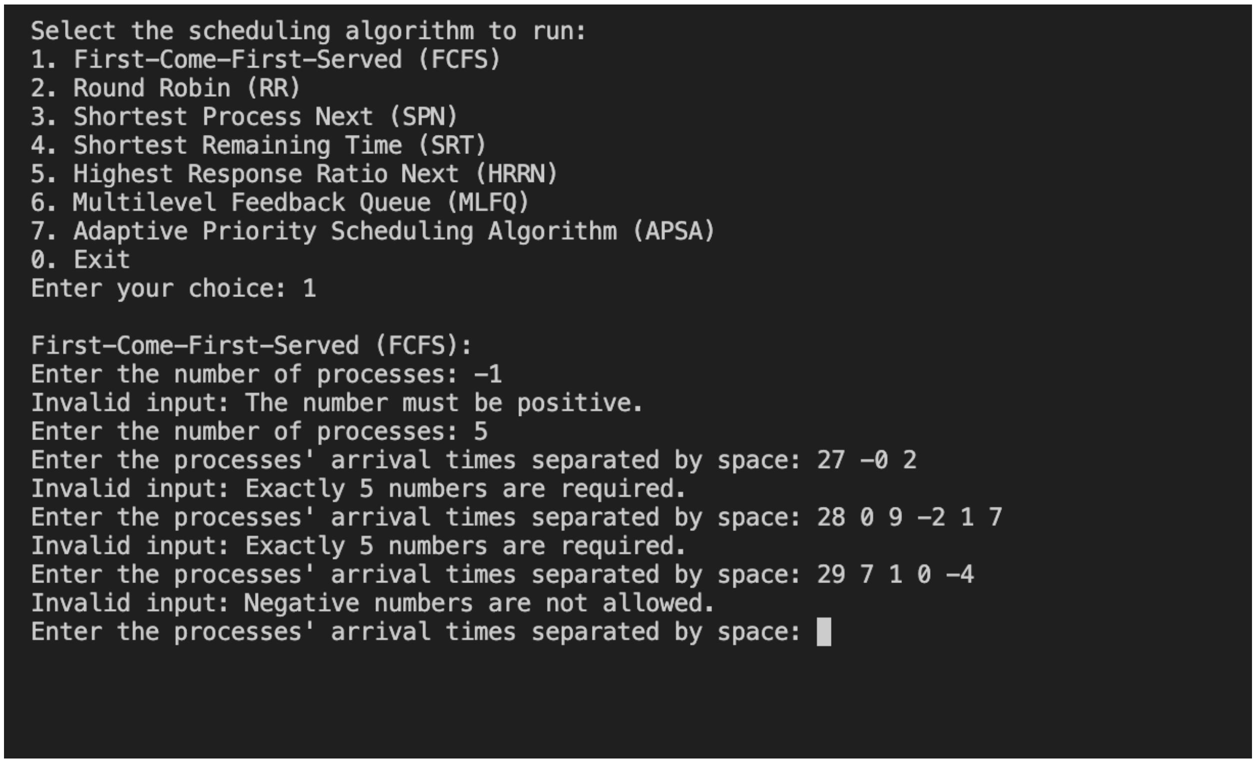
When the program runs, the user sees a prompt:



```
Select the scheduling algorithm to run:
1. First-Come-First-Served (FCFS)
2. Round Robin (RR)
3. Shortest Process Next (SPN)
4. Shortest Remaining Time (SRT)
5. Highest Response Ratio Next (HRRN)
6. Multilevel Feedback Queue (MLFQ)
7. Adaptive Priority Scheduling Algorithm (APSA)
0. Exit
Enter your choice: 1
```

The user is then guided to select an algorithm. Following this, the number of processes is requested, along with the arrival and service times for each, inputted as space-separated values. For algorithms like Round Robin and MFQ, time quanta are also asked for. The program meticulously validates every input to ensure accuracy.

A robust input validation mechanism is in place:



```
Select the scheduling algorithm to run:
1. First-Come-First-Served (FCFS)
2. Round Robin (RR)
3. Shortest Process Next (SPN)
4. Shortest Remaining Time (SRT)
5. Highest Response Ratio Next (HRRN)
6. Multilevel Feedback Queue (MLFQ)
7. Adaptive Priority Scheduling Algorithm (APSA)
0. Exit
Enter your choice: 1

First-Come-First-Served (FCFS):
Enter the number of processes: -1
Invalid input: The number must be positive.
Enter the number of processes: 5
Enter the processes' arrival times separated by space: 27 -0 2
Invalid input: Exactly 5 numbers are required.
Enter the processes' arrival times separated by space: 28 0 9 -2 1 7
Invalid input: Exactly 5 numbers are required.
Enter the processes' arrival times separated by space: 29 7 1 0 -4
Invalid input: Negative numbers are not allowed.
Enter the processes' arrival times separated by space: 
```

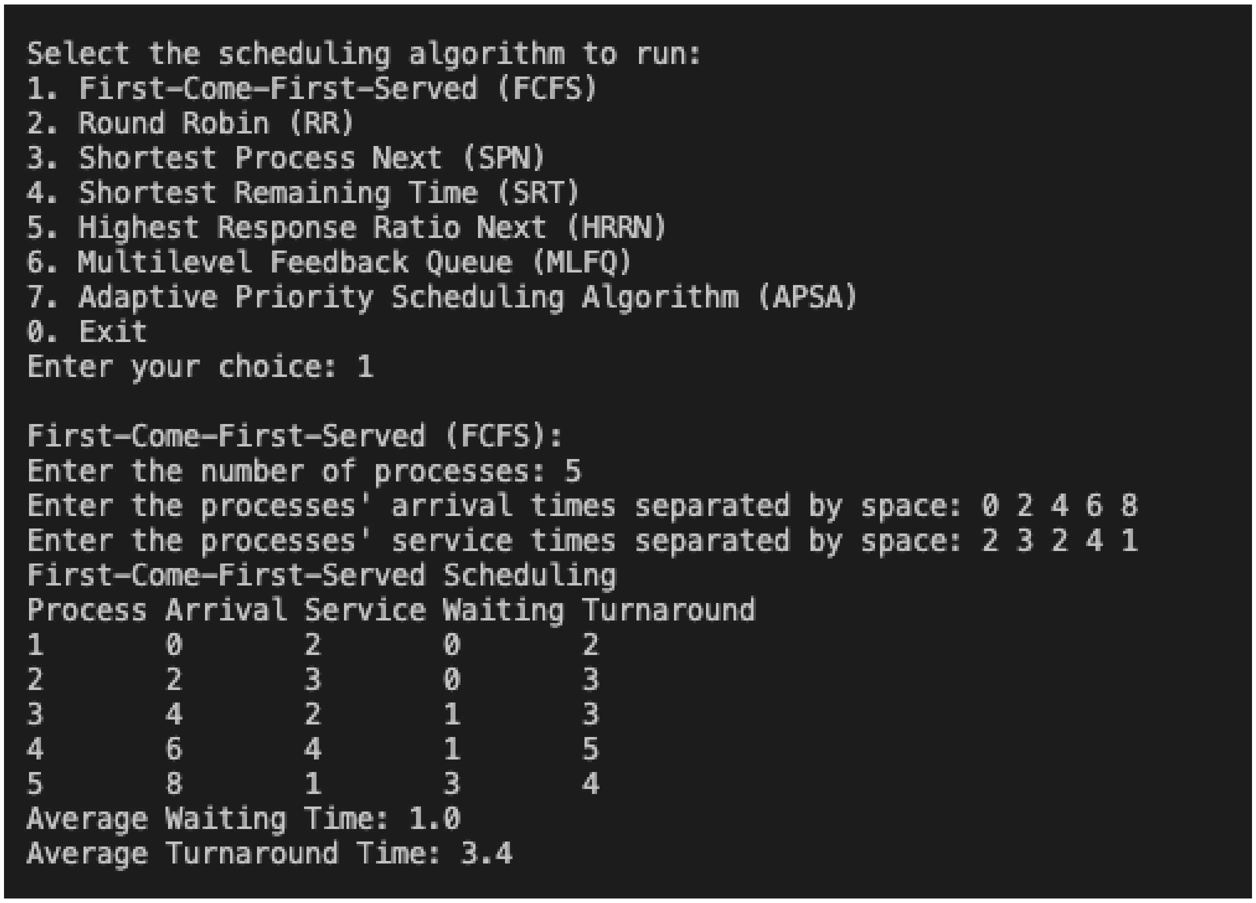
Figure 3.1 Invalid Input Detection

As shown above, if an invalid input is entered, it asks again until valid input is entered.

This ensures that the simulations run smoothly and accurately, reflecting the real-world scenarios they are designed to emulate.

Here's what the process looks like for each algorithm:

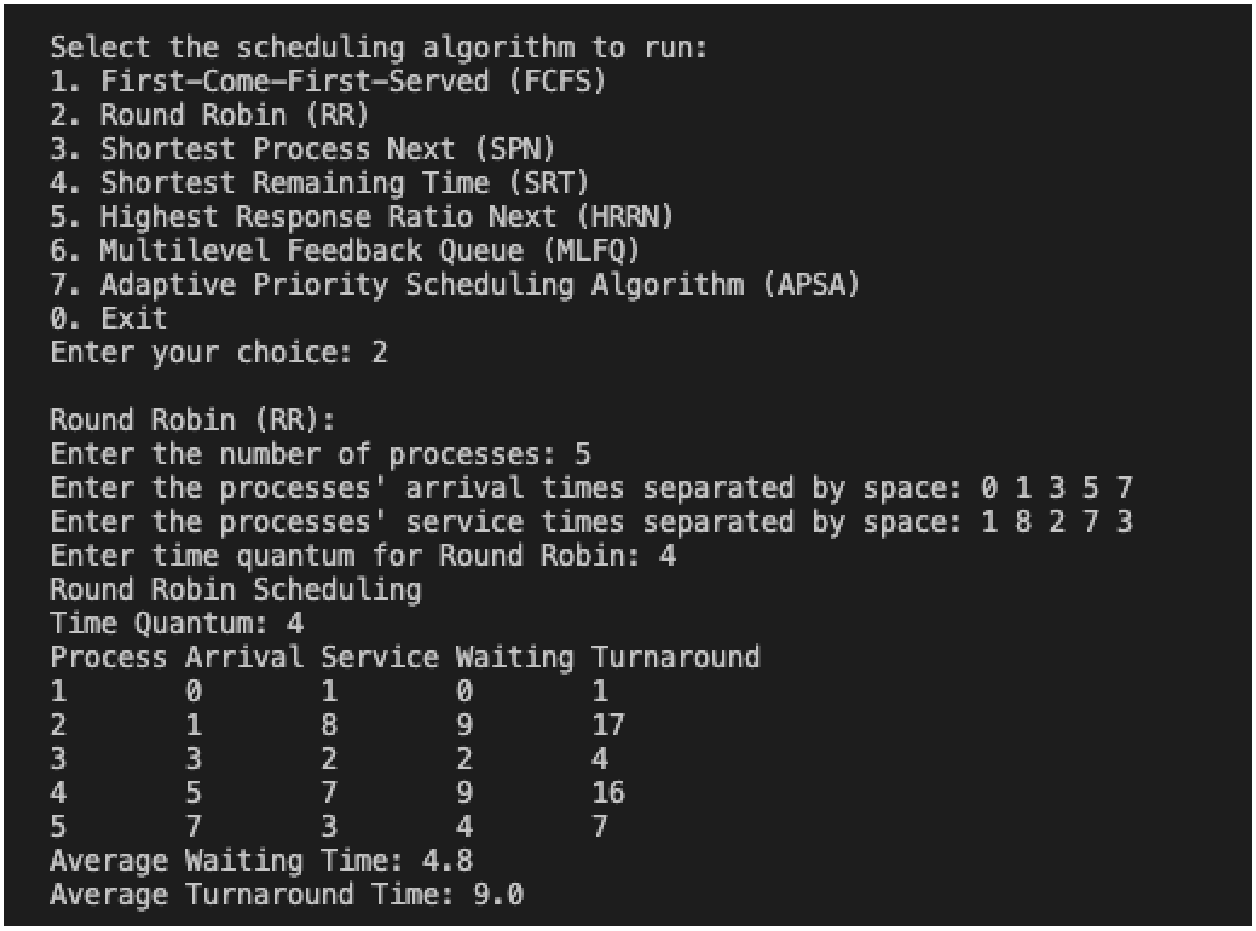
FCFS:



```
Select the scheduling algorithm to run:
1. First-Come-First-Served (FCFS)
2. Round Robin (RR)
3. Shortest Process Next (SPN)
4. Shortest Remaining Time (SRT)
5. Highest Response Ratio Next (HRRN)
6. Multilevel Feedback Queue (MLFQ)
7. Adaptive Priority Scheduling Algorithm (APSA)
0. Exit
Enter your choice: 1

First-Come-First-Served (FCFS):
Enter the number of processes: 5
Enter the processes' arrival times separated by space: 0 2 4 6 8
Enter the processes' service times separated by space: 2 3 2 4 1
First-Come-First-Served Scheduling
Process Arrival Service Waiting Turnaround
1       0       2       0       2
2       2       3       0       3
3       4       2       1       3
4       6       4       1       5
5       8       1       3       4
Average Waiting Time: 1.0
Average Turnaround Time: 3.4
```

Figure 3.2 FCFS Dynamic Output

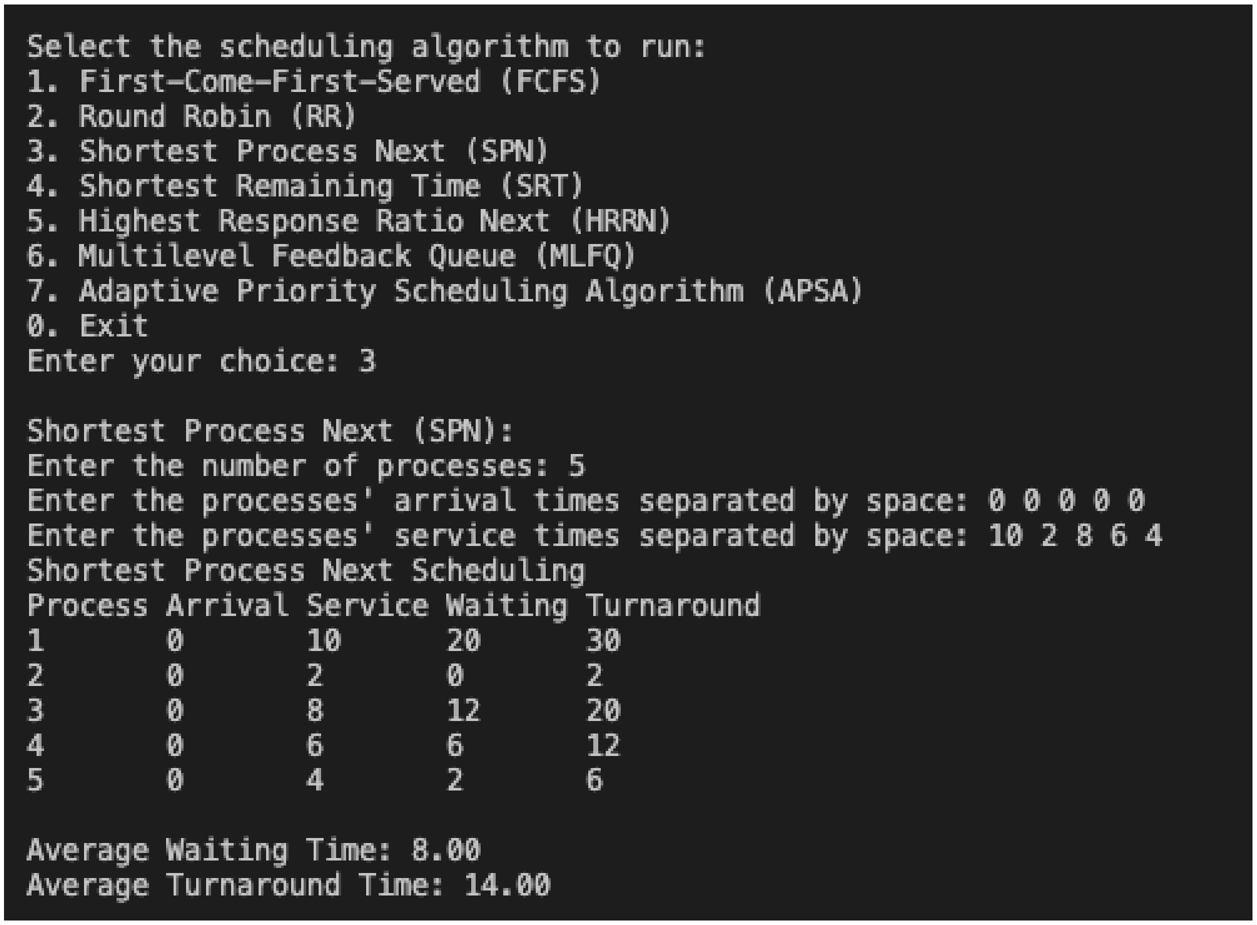
RR:



```
Select the scheduling algorithm to run:
1. First-Come-First-Served (FCFS)
2. Round Robin (RR)
3. Shortest Process Next (SPN)
4. Shortest Remaining Time (SRT)
5. Highest Response Ratio Next (HRRN)
6. Multilevel Feedback Queue (MLFQ)
7. Adaptive Priority Scheduling Algorithm (APSA)
0. Exit
Enter your choice: 2

Round Robin (RR):
Enter the number of processes: 5
Enter the processes' arrival times separated by space: 0 1 3 5 7
Enter the processes' service times separated by space: 1 8 2 7 3
Enter time quantum for Round Robin: 4
Round Robin Scheduling
Time Quantum: 4
Process Arrival Service Waiting Turnaround
1       0       1       0       1
2       1       8       9       17
3       3       2       2       4
4       5       7       9       16
5       7       3       4       7
Average Waiting Time: 4.8
Average Turnaround Time: 9.0
```

Figure 3.3 RR Dynamic Output

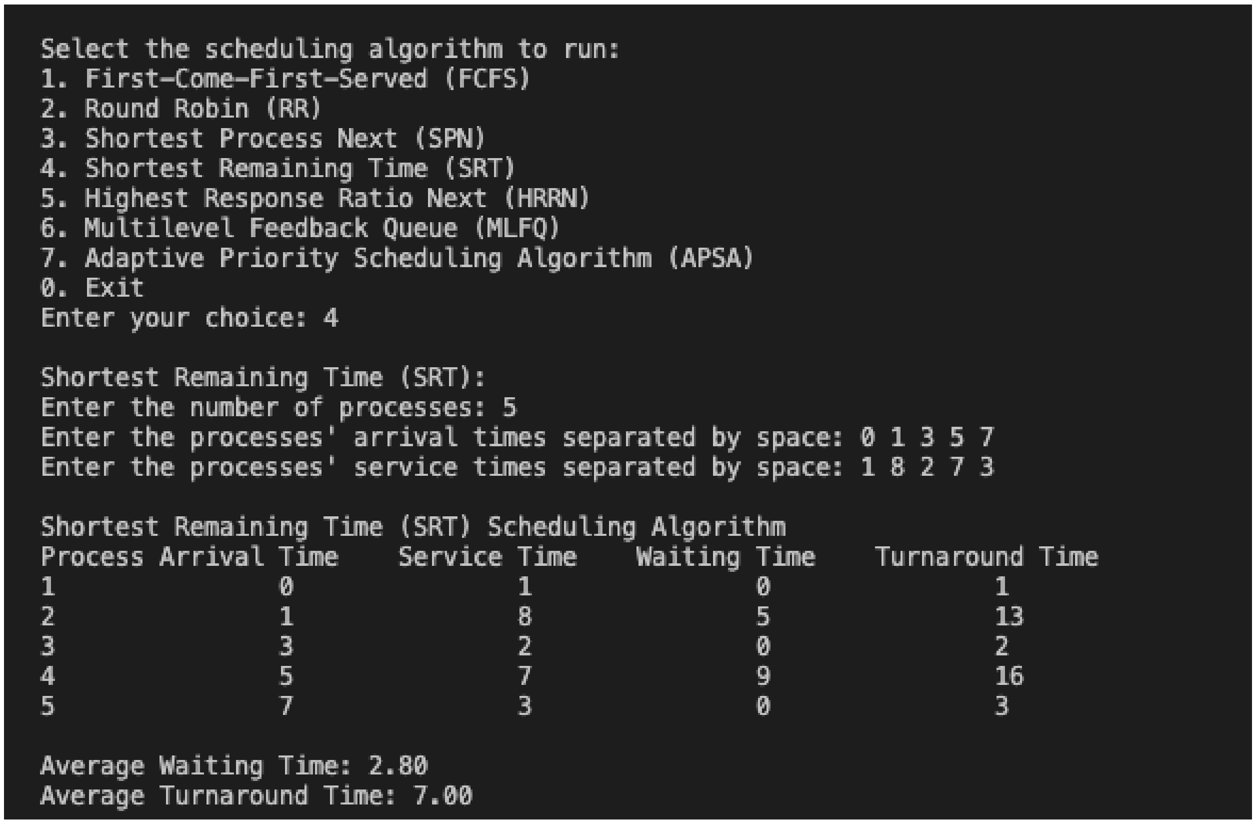
SPN:



```
Select the scheduling algorithm to run:
1. First-Come-First-Served (FCFS)
2. Round Robin (RR)
3. Shortest Process Next (SPN)
4. Shortest Remaining Time (SRT)
5. Highest Response Ratio Next (HRRN)
6. Multilevel Feedback Queue (MLFQ)
7. Adaptive Priority Scheduling Algorithm (APSA)
0. Exit
Enter your choice: 3

Shortest Process Next (SPN):
Enter the number of processes: 5
Enter the processes' arrival times separated by space: 0 0 0 0 0
Enter the processes' service times separated by space: 10 2 8 6 4
Shortest Process Next Scheduling
Process Arrival Service Waiting Turnaround
1       0       10      20      30
2       0       2       0       2
3       0       8       12      20
4       0       6       6       12
5       0       4       2       6

Average Waiting Time: 8.00
Average Turnaround Time: 14.00
```

Figure 3.4 SPN Dynamic Output

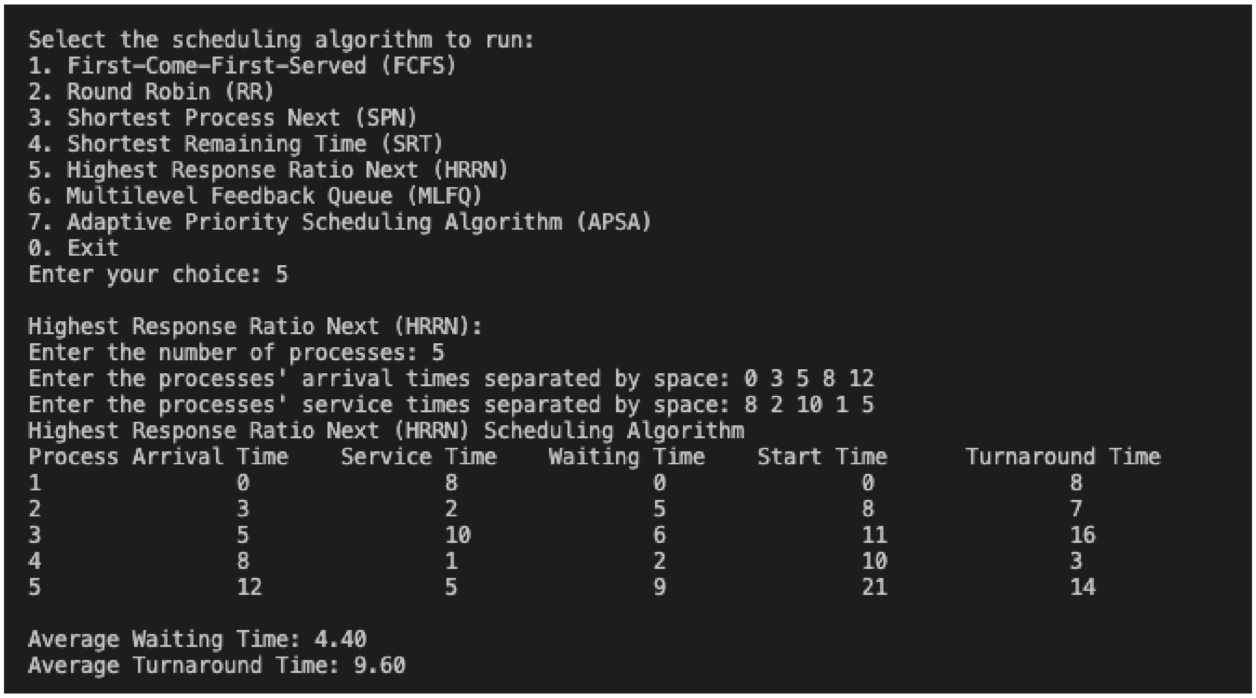
SRT:



```
Select the scheduling algorithm to run:
1. First-Come-First-Served (FCFS)
2. Round Robin (RR)
3. Shortest Process Next (SPN)
4. Shortest Remaining Time (SRT)
5. Highest Response Ratio Next (HRRN)
6. Multilevel Feedback Queue (MLFQ)
7. Adaptive Priority Scheduling Algorithm (APSA)
0. Exit
Enter your choice: 4

Shortest Remaining Time (SRT):
Enter the number of processes: 5
Enter the processes' arrival times separated by space: 0 1 3 5 7
Enter the processes' service times separated by space: 1 8 2 7 3

Shortest Remaining Time (SRT) Scheduling Algorithm
Process Arrival Time    Service Time    Waiting Time    Turnaround Time
1       0               1               0               1
2       1               8               5               13
3       3               2               0               2
4       5               7               9               16
5       7               3               0               3

Average Waiting Time: 2.80
Average Turnaround Time: 7.00
```

Figure 3.5 SRT Dynamic Output

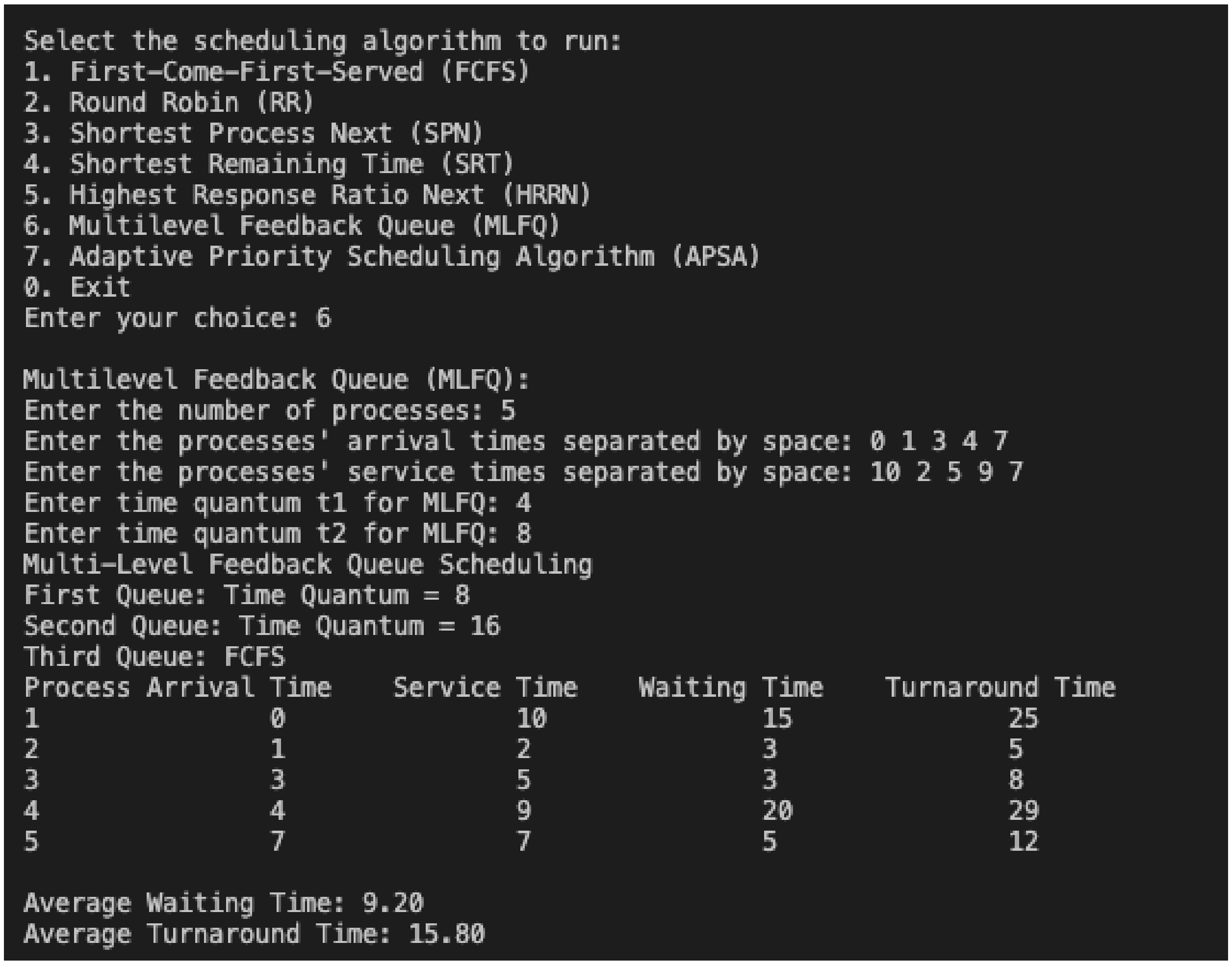
HRRN:



```
Select the scheduling algorithm to run:
1. First-Come-First-Served (FCFS)
2. Round Robin (RR)
3. Shortest Process Next (SPN)
4. Shortest Remaining Time (SRT)
5. Highest Response Ratio Next (HRRN)
6. Multilevel Feedback Queue (MLFQ)
7. Adaptive Priority Scheduling Algorithm (APSA)
0. Exit
Enter your choice: 5

Highest Response Ratio Next (HRRN):
Enter the number of processes: 5
Enter the processes' arrival times separated by space: 0 3 5 8 12
Enter the processes' service times separated by space: 8 2 10 1 5
Highest Response Ratio Next (HRRN) Scheduling Algorithm
Process Arrival Time    Service Time    Waiting Time    Start Time      Turnaround Time
1       0               8               0               0               8
2       3               2               5               8               7
3       5               10              6               11              16
4       8               1               2               10              3
5       12              5               9               21              14

Average Waiting Time: 4.40
Average Turnaround Time: 9.60
```

Figure 3.6 HRRN Dynamic Output

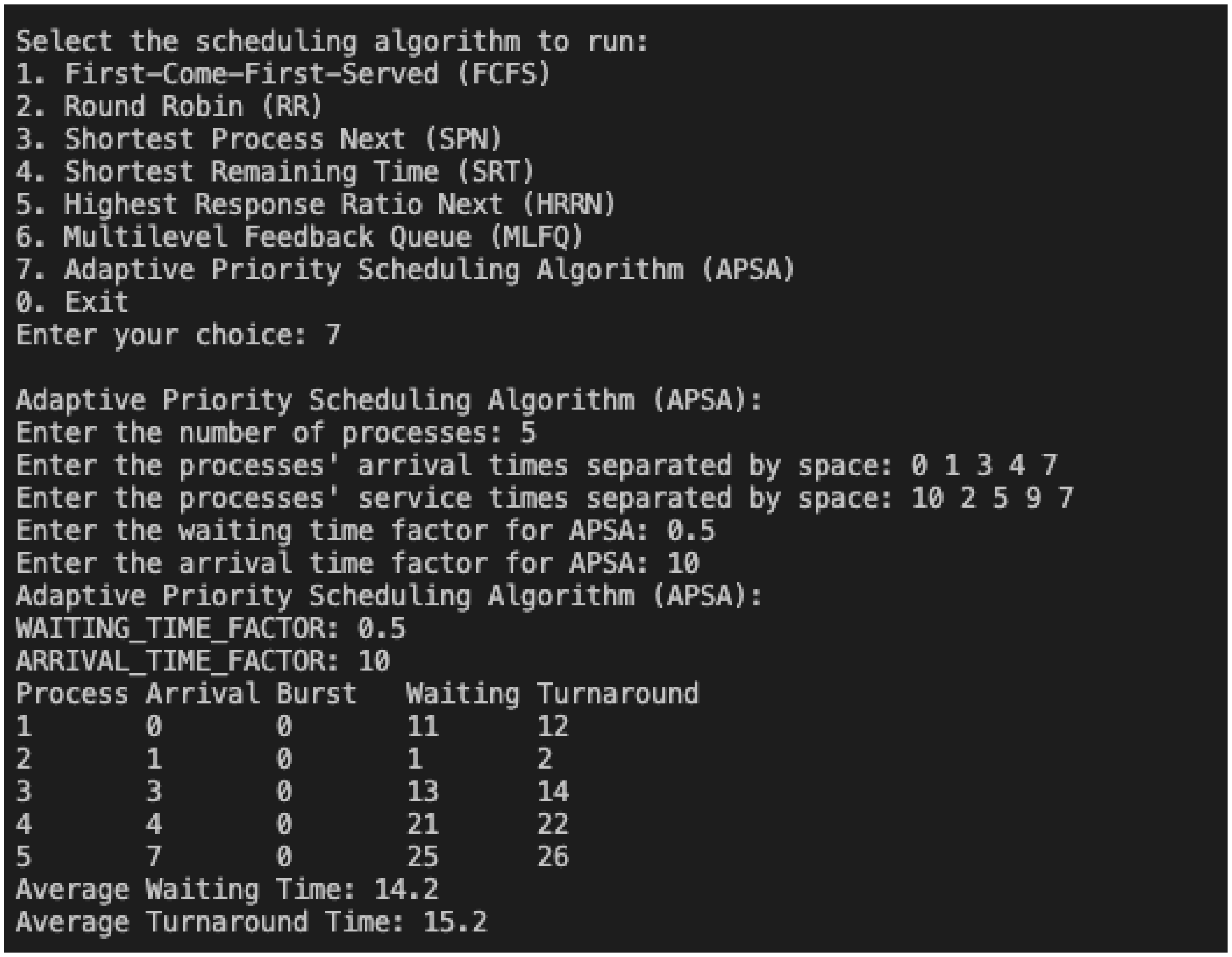
MFS:



```
Select the scheduling algorithm to run:
1. First-Come-First-Served (FCFS)
2. Round Robin (RR)
3. Shortest Process Next (SPN)
4. Shortest Remaining Time (SRT)
5. Highest Response Ratio Next (HRRN)
6. Multilevel Feedback Queue (MLFQ)
7. Adaptive Priority Scheduling Algorithm (APSA)
0. Exit
Enter your choice: 6

Multilevel Feedback Queue (MLFQ):
Enter the number of processes: 5
Enter the processes' arrival times separated by space: 0 1 3 4 7
Enter the processes' service times separated by space: 10 2 5 9 7
Enter time quantum t1 for MLFQ: 4
Enter time quantum t2 for MLFQ: 8
Multi-Level Feedback Queue Scheduling
First Queue: Time Quantum = 8
Second Queue: Time Quantum = 16
Third Queue: FCFS
Process Arrival Time    Service Time    Waiting Time    Turnaround Time
1               0               10              15              25
2               1               2               3               5
3               3               5               3               8
4               4               9               20              29
5               7               7               5               12

Average Waiting Time: 9.20
Average Turnaround Time: 15.80
```

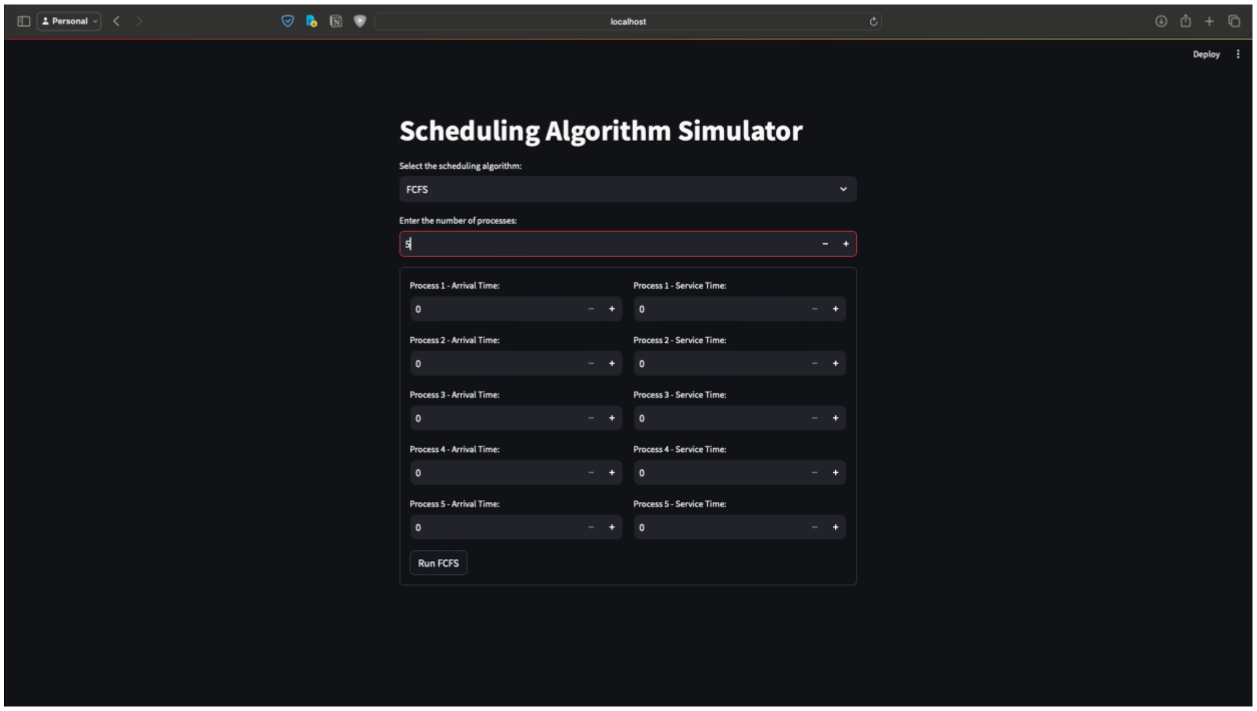
Figure 3.7 MFS Dynamic Output

APSA:



```
Select the scheduling algorithm to run:
1. First-Come-First-Served (FCFS)
2. Round Robin (RR)
3. Shortest Process Next (SPN)
4. Shortest Remaining Time (SRT)
5. Highest Response Ratio Next (HRRN)
6. Multilevel Feedback Queue (MLFQ)
7. Adaptive Priority Scheduling Algorithm (APSA)
0. Exit
Enter your choice: 7

Adaptive Priority Scheduling Algorithm (APSA):
Enter the number of processes: 5
Enter the processes' arrival times separated by space: 0 1 3 4 7
Enter the processes' service times separated by space: 10 2 5 9 7
Enter the waiting time factor for APSA: 0.5
Enter the arrival time factor for APSA: 10
Adaptive Priority Scheduling Algorithm (APSA):
WAITING_TIME_FACTOR: 0.5
ARRIVAL_TIME_FACTOR: 10
Process Arrival Burst   Waiting Turnaround
1       0       0       11      12
2       1       0       1       2
3       3       0       13      14
4       4       0       21      22
5       7       0       25      26
Average Waiting Time: 14.2
Average Turnaround Time: 15.2
```

C) Graphical UI: Utilizing Streamlit, an intuitive and interactive GUI was developed. This interface enables users to easily input process data, select algorithms, and view results, making the exploration of scheduling algorithms more accessible.



Scheduling Algorithm Simulator

Select the scheduling algorithm: FCFS

Enter the number of processes: 5

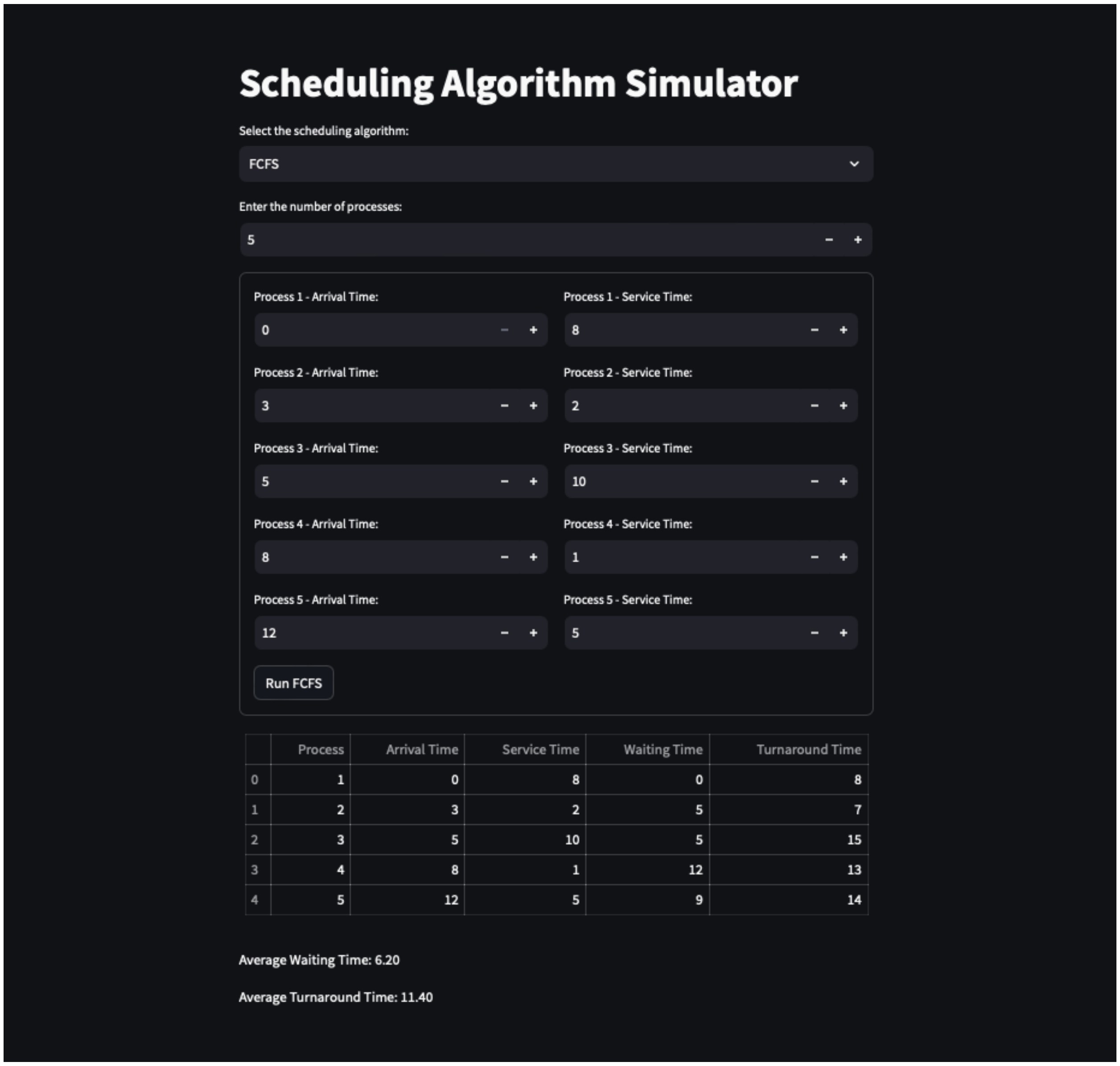
Process 1 - Arrival Time: 0	Process 1 - Service Time: 0
Process 2 - Arrival Time: 0	Process 2 - Service Time: 0
Process 3 - Arrival Time: 0	Process 3 - Service Time: 0
Process 4 - Arrival Time: 0	Process 4 - Service Time: 0
Process 5 - Arrival Time: 0	Process 5 - Service Time: 0

This Python code creates a web app simulating scheduling algorithms for processes.

1. User selects:
   - Algorithm (FCFS, Round Robin, etc.)
   - Number of processes
2. For each process:
   - User inputs arrival and service times
3. Conditional inputs:
   - Time quantum for Round Robin/MFQ
   - Waiting/arrival time factors for APSA
4. Click 'Run':
   - Algorithm runs with user-provided data
5. Output:
   - Table showing process details (arrival, service, waiting, turnaround times)
   - Average waiting and turnaround times

This interactive app leverages libraries like Streamlit and Pandas for a seamless user experience and data visualization. The functions (run_algorithm and main) handle data processing, algorithm execution, and output generation.

GUI app can also be deployed to cloud in a single click.



Scheduling Algorithm Simulator

Select the scheduling algorithm: FCFS

Enter the number of processes: 5

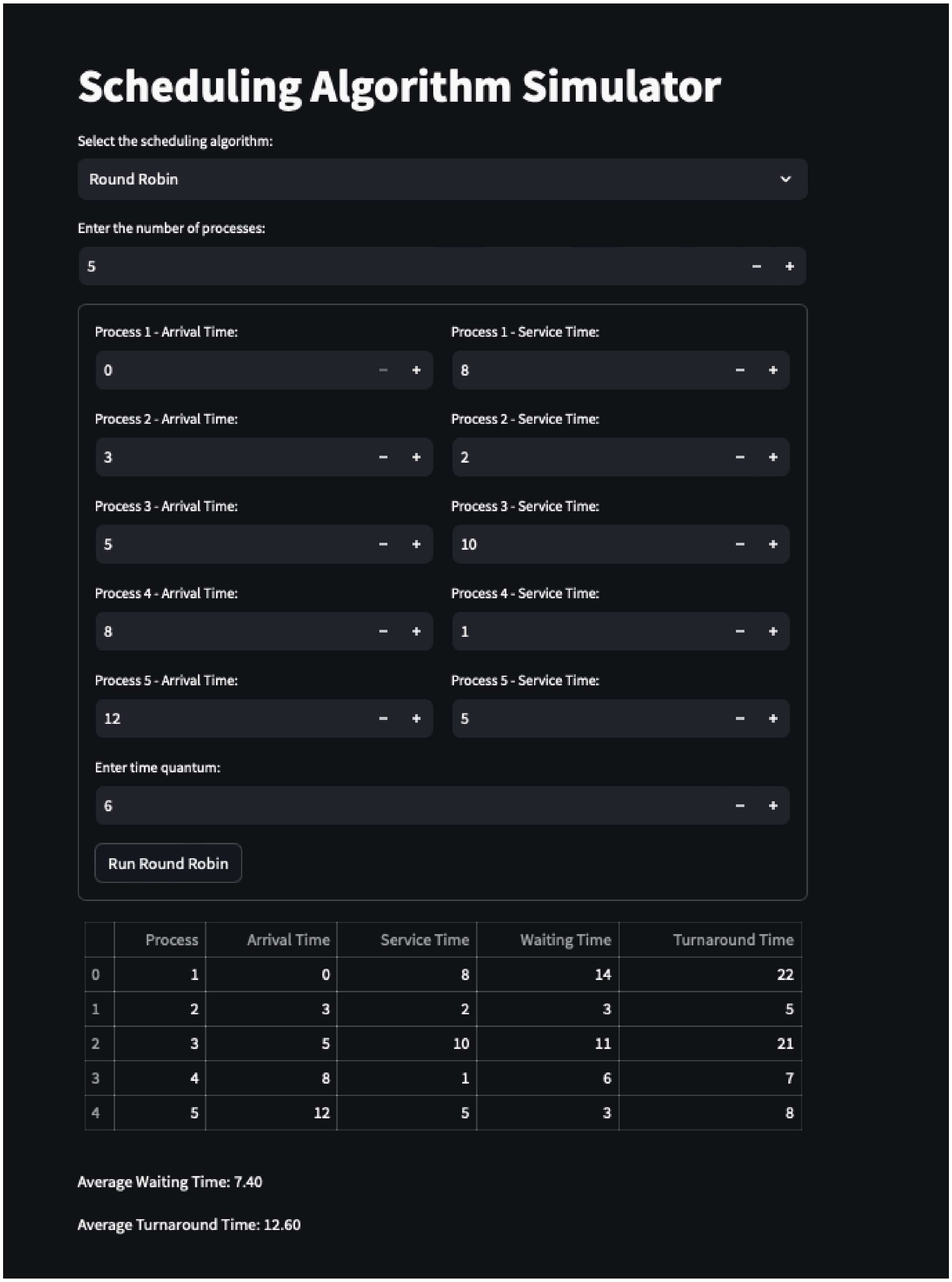
Process 1 - Arrival Time: 0	Process 1 - Service Time: 8
Process 2 - Arrival Time: 3	Process 2 - Service Time: 2
Process 3 - Arrival Time: 5	Process 3 - Service Time: 10
Process 4 - Arrival Time: 8	Process 4 - Service Time: 1
Process 5 - Arrival Time: 12	Process 5 - Service Time: 5

	Process	Arrival Time	Service Time	Waiting Time	Turnaround Time
0	1	0	8	0	8
1	2	3	2	5	7
2	3	5	10	5	15
3	4	8	1	12	13
4	5	12	5	9	14

Average Waiting Time: 6.20

Average Turnaround Time: 11.40

FCFS



Scheduling Algorithm Simulator

Select the scheduling algorithm: Round Robin

Enter the number of processes: 5

Process 1 - Arrival Time: 0	Process 1 - Service Time: 8
Process 2 - Arrival Time: 3	Process 2 - Service Time: 2
Process 3 - Arrival Time: 5	Process 3 - Service Time: 10
Process 4 - Arrival Time: 8	Process 4 - Service Time: 1
Process 5 - Arrival Time: 12	Process 5 - Service Time: 5

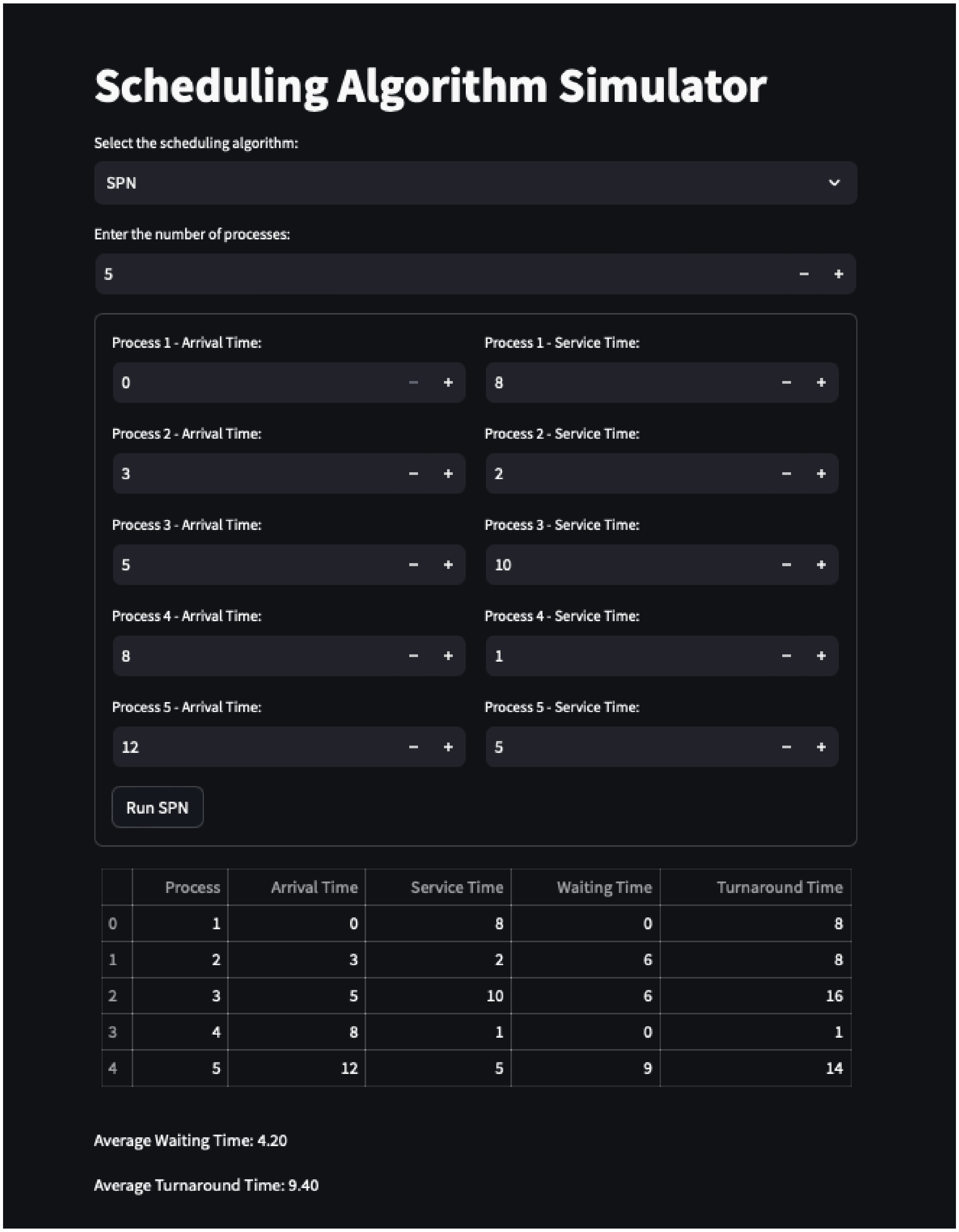
Enter time quantum: 6

	Process	Arrival Time	Service Time	Waiting Time	Turnaround Time
0	1	0	8	14	22
1	2	3	2	3	5
2	3	5	10	11	21
3	4	8	1	6	7
4	5	12	5	3	8

Average Waiting Time: 7.40

Average Turnaround Time: 12.60

Round Robin



Scheduling Algorithm Simulator

Select the scheduling algorithm: SPN

Enter the number of processes: 5

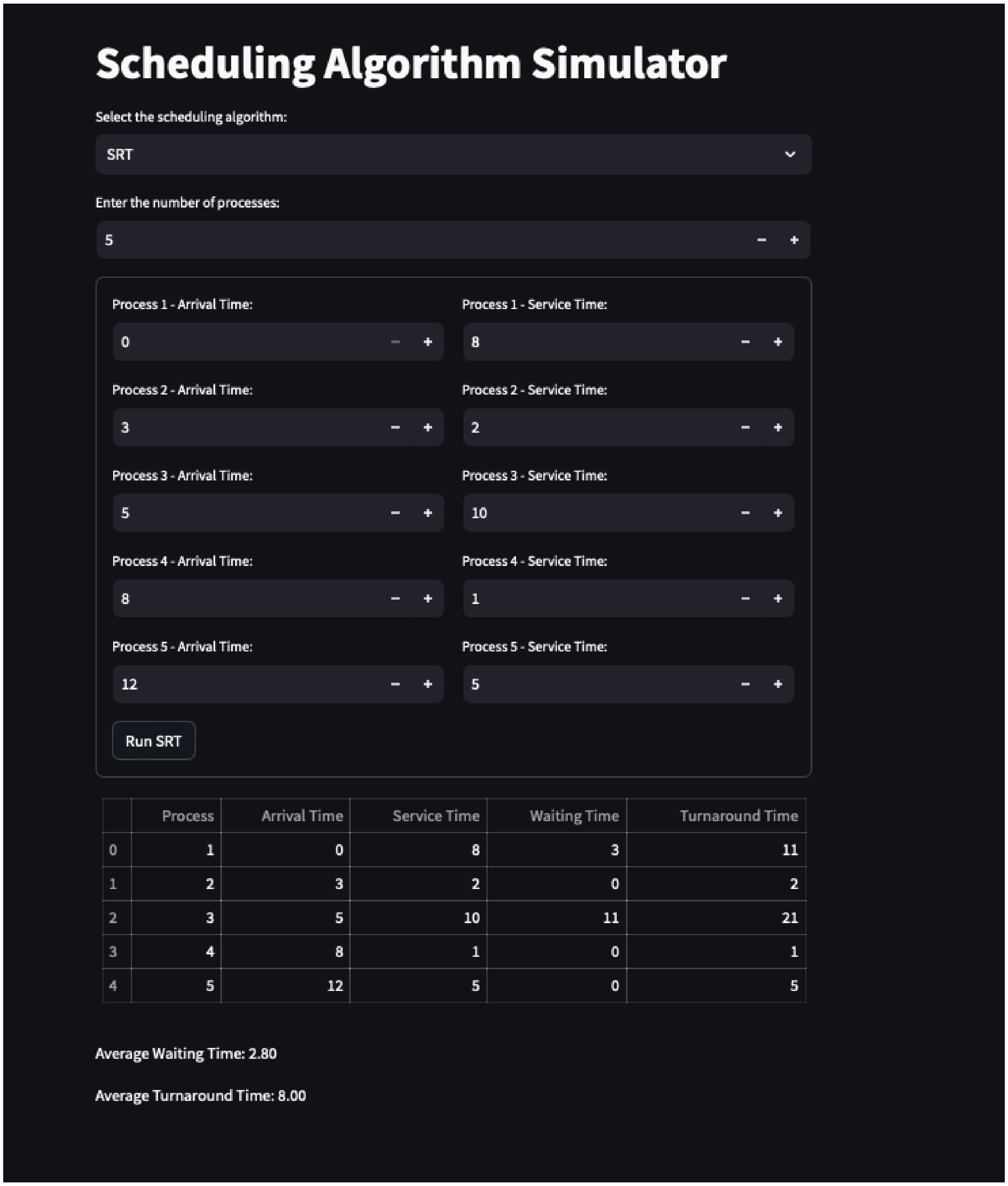
Process 1 - Arrival Time: 0	Process 1 - Service Time: 8
Process 2 - Arrival Time: 3	Process 2 - Service Time: 2
Process 3 - Arrival Time: 5	Process 3 - Service Time: 10
Process 4 - Arrival Time: 8	Process 4 - Service Time: 1
Process 5 - Arrival Time: 12	Process 5 - Service Time: 5

	Process	Arrival Time	Service Time	Waiting Time	Turnaround Time
0	1	0	8	0	8
1	2	3	2	6	8
2	3	5	10	6	16
3	4	8	1	0	1
4	5	12	5	9	14

Average Waiting Time: 4.20

Average Turnaround Time: 9.40

SPN



Scheduling Algorithm Simulator

Select the scheduling algorithm: SRT

Enter the number of processes: 5

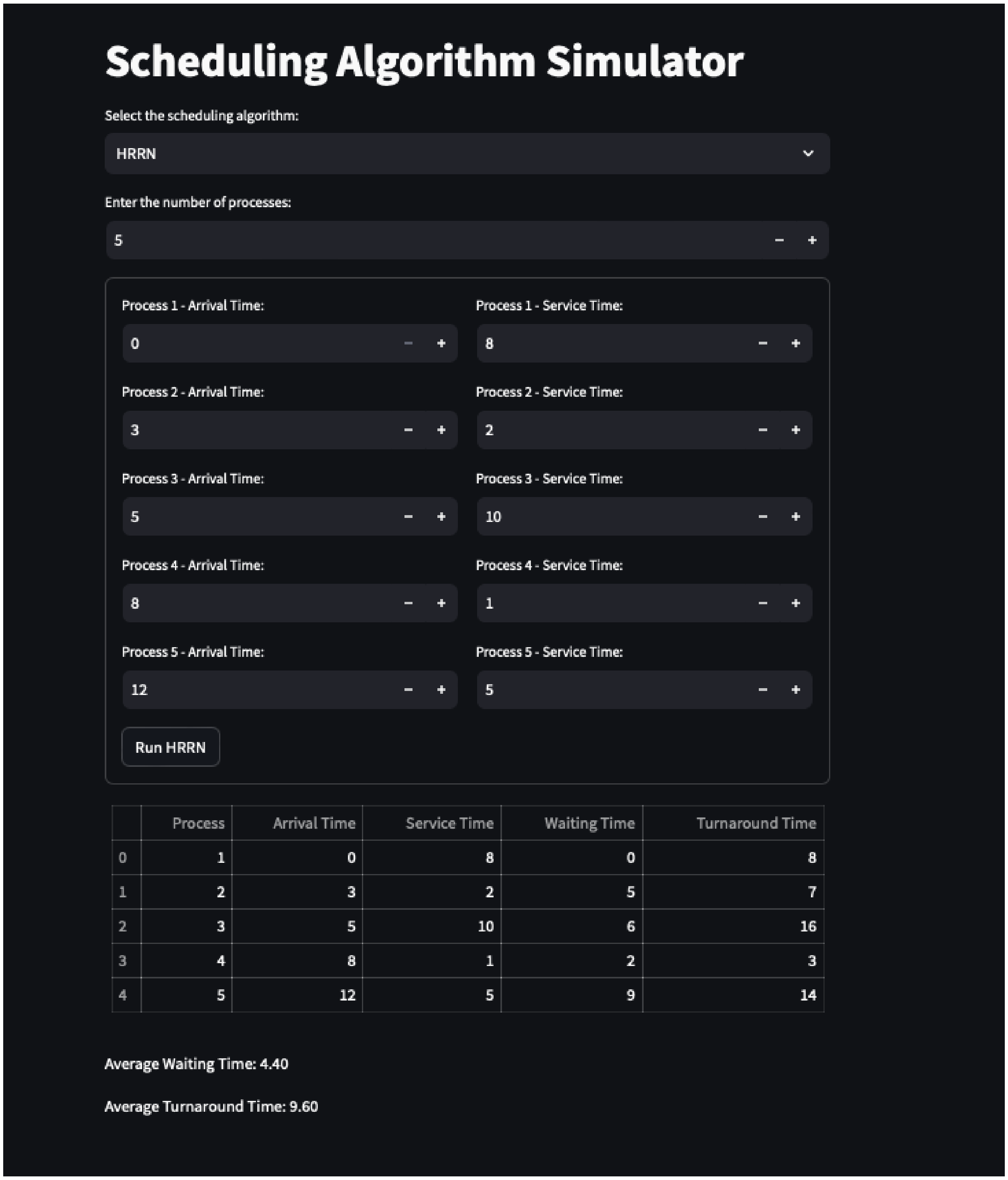
Process 1 - Arrival Time: 0
Process 1 - Service Time: 8
Process 2 - Arrival Time: 3
Process 2 - Service Time: 2
Process 3 - Arrival Time: 5
Process 3 - Service Time: 10
Process 4 - Arrival Time: 8
Process 4 - Service Time: 1
Process 5 - Arrival Time: 12
Process 5 - Service Time: 5

	Process	Arrival Time	Service Time	Waiting Time	Turnaround Time
0	1	0	8	3	11
1	2	3	2	0	2
2	3	5	10	11	21
3	4	8	1	0	1
4	5	12	5	0	5

Average Waiting Time: 2.80

Average Turnaround Time: 8.00

SRT



Scheduling Algorithm Simulator

Select the scheduling algorithm: HRRN

Enter the number of processes: 5

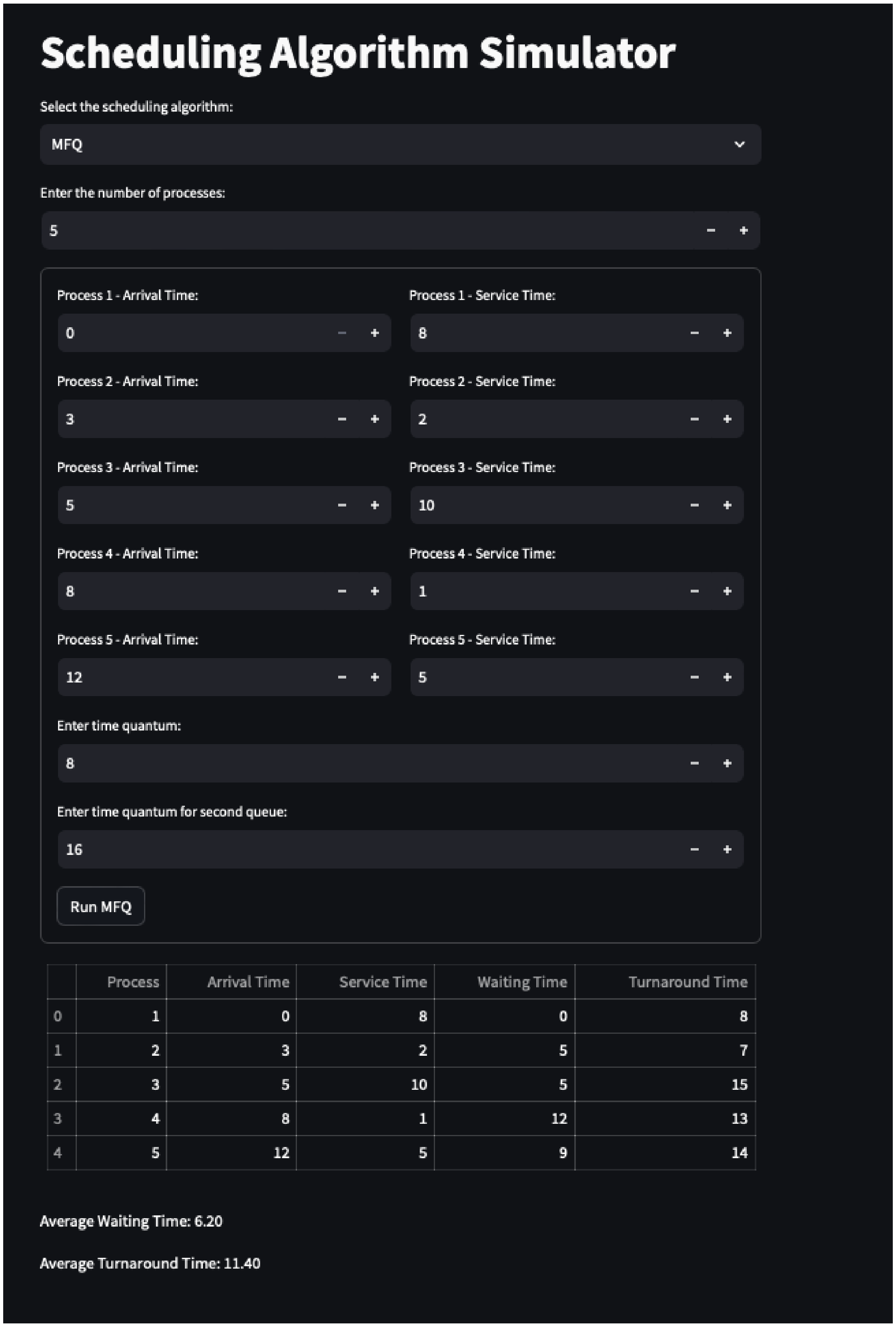
Process 1 - Arrival Time: 0	Process 1 - Service Time: 8
Process 2 - Arrival Time: 3	Process 2 - Service Time: 2
Process 3 - Arrival Time: 5	Process 3 - Service Time: 10
Process 4 - Arrival Time: 8	Process 4 - Service Time: 1
Process 5 - Arrival Time: 12	Process 5 - Service Time: 5

	Process	Arrival Time	Service Time	Waiting Time	Turnaround Time
0	1	0	8	0	8
1	2	3	2	5	7
2	3	5	10	6	16
3	4	8	1	2	3
4	5	12	5	9	14

Average Waiting Time: 4.40

Average Turnaround Time: 9.60

HRNN



Scheduling Algorithm Simulator

Select the scheduling algorithm: MFQ

Enter the number of processes: 5

Process 1 - Arrival Time: 0
Process 1 - Service Time: 8

Process 2 - Arrival Time: 3
Process 2 - Service Time: 2

Process 3 - Arrival Time: 5
Process 3 - Service Time: 10

Process 4 - Arrival Time: 8
Process 4 - Service Time: 1

Process 5 - Arrival Time: 12
Process 5 - Service Time: 5

Enter time quantum: 8

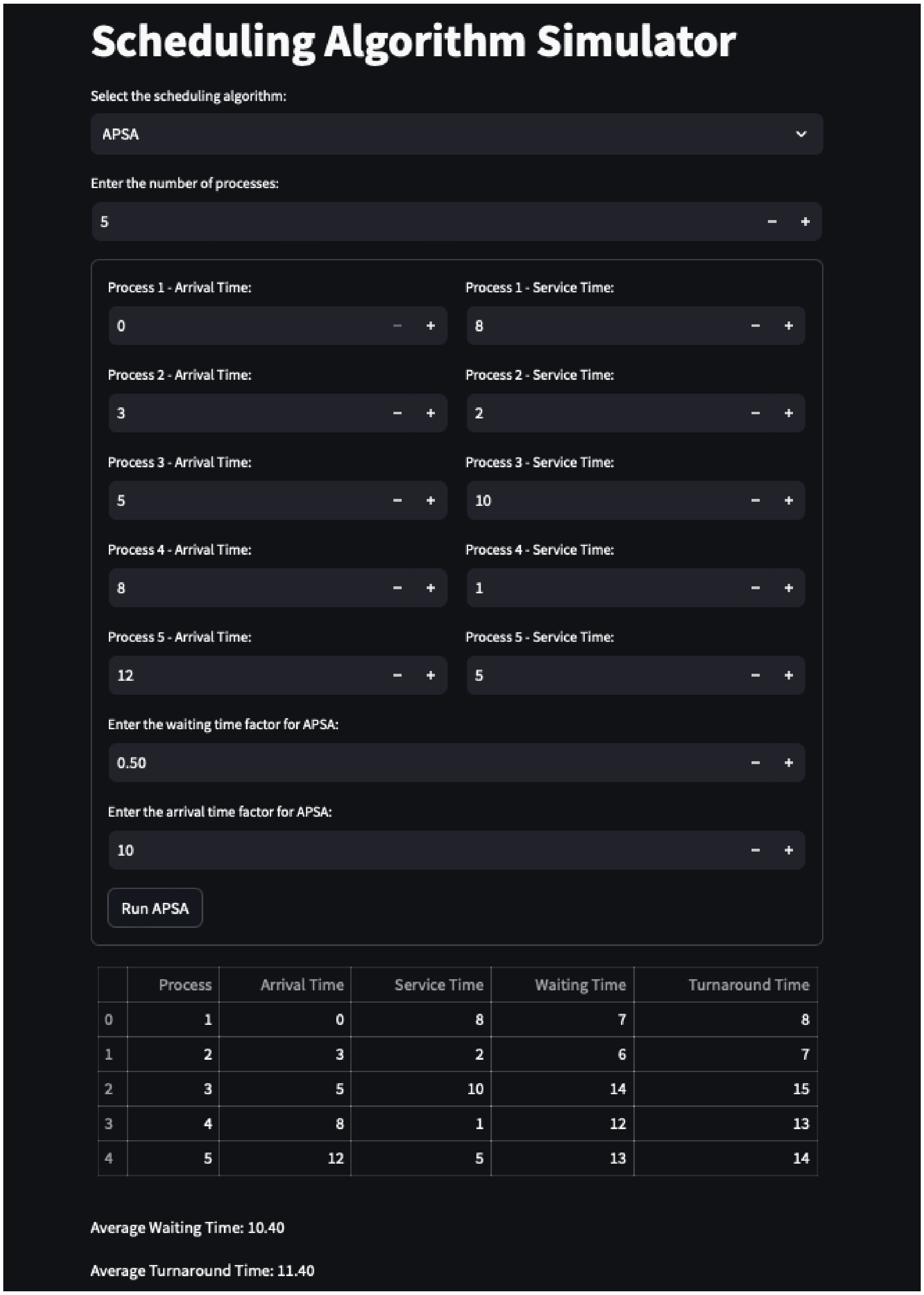
Enter time quantum for second queue: 16

	Process	Arrival Time	Service Time	Waiting Time	Turnaround Time
0	1	0	8	0	8
1	2	3	2	5	7
2	3	5	10	5	15
3	4	8	1	12	13
4	5	12	5	9	14

Average Waiting Time: 6.20

Average Turnaround Time: 11.40

MFQ



Scheduling Algorithm Simulator

Select the scheduling algorithm: APSA

Enter the number of processes: 5

Process 1 - Arrival Time: 0	Process 1 - Service Time: 8
Process 2 - Arrival Time: 3	Process 2 - Service Time: 2
Process 3 - Arrival Time: 5	Process 3 - Service Time: 10
Process 4 - Arrival Time: 8	Process 4 - Service Time: 1
Process 5 - Arrival Time: 12	Process 5 - Service Time: 5

Enter the waiting time factor for APSA: 0.50

Enter the arrival time factor for APSA: 10

Run APSA

	Process	Arrival Time	Service Time	Waiting Time	Turnaround Time
0	1	0	8	7	8
1	2	3	2	6	7
2	3	5	10	14	15
3	4	8	1	12	13
4	5	12	5	13	14

Average Waiting Time: 10.40

Average Turnaround Time: 11.40

ASPA

Code for GUI is as below:

```
import streamlit as st
import pandas as pd
from algorithms.fcfs import fcfs
from algorithms.hrrn import hrrn
from algorithms.srt import srt
from algorithms.spn import spn
from algorithms.rr import round_robin
from algorithms.mfq import mlfq as mfq
from algorithms.custom import apsa


def run_algorithm(
    algo_func,
    arrival_times,
    service_times,
    time_quantum=None,
    waiting_time_factor=None,
    arrival_time_factor=None,
    time_quantum2=None,
):
    service_times1 = service_times.copy()
    arrival_times1 = arrival_times.copy()
    if time_quantum and not time_quantum2:
        waiting_times, turnaround_times = algo_func(
            arrival_times, service_times, time_quantum, print_results=False
        )
    elif time_quantum2:
        waiting_times, turnaround_times = algo_func(
            arrival_times,
            service_times,
            time_quantum,
            time_quantum2,
            print_results=False,
        )
    elif waiting_time_factor and arrival_time_factor:
        waiting_times, turnaround_times = algo_func(
            arrival_times,
            service_times,
            waiting_time_factor,
            arrival_time_factor,
            print_results=False,
        )
    else:
        waiting_times, turnaround_times = algo_func(
            arrival_times, service_times, print_results=False
        )

    # Constructing the table
    process_ids = list(range(1, len(arrival_times) + 1))
    data = {
        "Process": process_ids,
        "Arrival Time": arrival_times1,
        "Service Time": service_times1,
        "Waiting Time": waiting_times,
        "Turnaround Time": turnaround_times,
    }
    df = pd.DataFrame(data)
    st.table(df)

    # Calculate and display averages
    avg_waiting_time = sum(waiting_times) / len(waiting_times)
    avg_turnaround_time = sum(turnaround_times) / len(turnaround_times)
    st.write(f"Average Waiting Time: {avg_waiting_time:.2f}")
    st.write(f"Average Turnaround Time: {avg_turnaround_time:.2f}")


def main():
    st.title("Scheduling Algorithm Simulator")

    # Selection of the algorithm
    algorithm = st.selectbox(
        "Select the scheduling algorithm:",
        ("FCFS", "Round Robin", "SPN", "SRT", "HRRN", "MFQ", "APSA"),
    )

    # Input for processes
    num_processes = st.number_input(
        "Enter the number of processes:",
        min_value=1,
        value=1,
        step=1,
        key="num_processes",
    )

    # Use a form for batch input submission
    form = st.form(key="processes_form")
    arrival_times = []
    service_times = []
    for i in range(num_processes):
        cols = form.columns(2)
        with cols[0]:
            arrival_time = st.number_input(
                f"Process {i+1} - Arrival Time:", min_value=0, key=f"arrival_{i}"
            )
        with cols[1]:
            service_time = st.number_input(
                f"Process {i+1} - Service Time:", min_value=0, key=f"service_{i}"
            )
        arrival_times.append(arrival_time)
        service_times.append(service_time)

    # Conditional input for time quantum if Round Robin or MFQ is selected
    time_quantum = None
    if algorithm in ["Round Robin", "MFQ"]:
        time_quantum = form.number_input(
            "Enter time quantum:", min_value=1, value=1, key="time_quantum"
        )
    time_quantum2 = None
    if algorithm == "MFQ":
        time_quantum2 = form.number_input(
            "Enter time quantum for second queue:",
            min_value=1,
            value=1,
            key="time_quantum2",
        )
    waiting_time_factor = None
    arrival_time_factor = None
    if algorithm == "APSA":
        waiting_time_factor = form.number_input(
            "Enter the waiting time factor for APSA: ",
            min_value=0.0,
            value=0.5,
            key="waiting_time_factor",
        )
        arrival_time_factor = form.number_input(
            "Enter the arrival time factor for APSA: ",
            min_value=0,
            value=10,
            key="arrival_time_factor",
        )
    submit_button = form.form_submit_button(label="Run {}".format(algorithm))

    if submit_button:
        algo_mapping = {
            "FCFS": fcfs,
            "Round Robin": round_robin,
            "SPN": spn,
            "SRT": srt,
            "HRRN": hrrn,
            "MFQ": mfq,
            "APSA": apsa,
        }

        # Call the corresponding algorithm function
        run_algorithm(
            algo_mapping[algorithm],
            arrival_times,
            service_times,
            time_quantum,
            waiting_time_factor,
            arrival_time_factor,
            time_quantum2,
        )


    if __name__ == "__main__":
        main()
```

D) Analysis:

- The analysis involved running the six scheduling algorithms on various input sets. Each set represents a different workload type, providing insight into how each algorithm performs under diverse scenarios.
- The results are captured in a table format, listing the input set, the algorithm used, and the corresponding average turnaround and waiting times.

Table 1: Test Cases

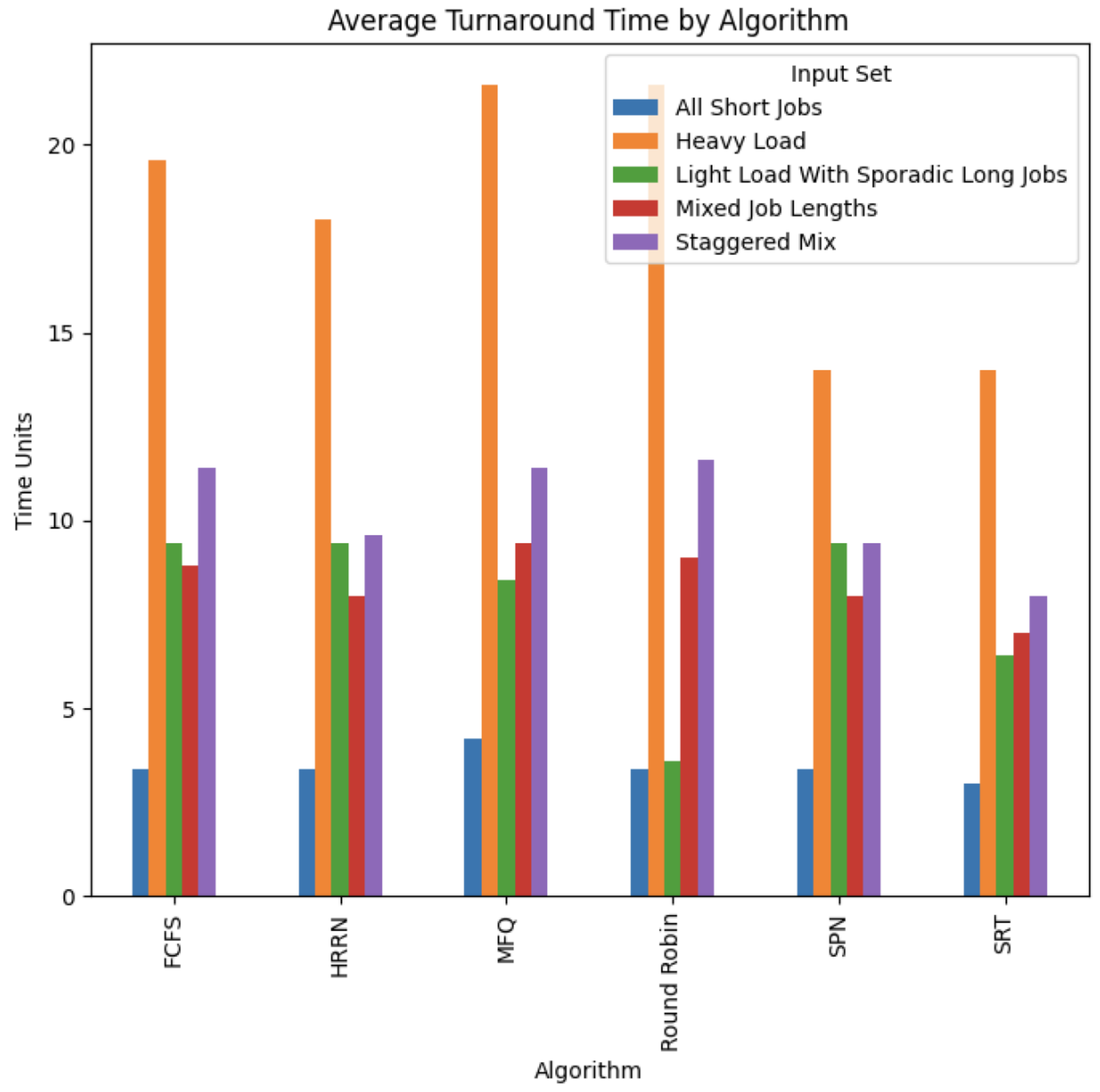
Input Set	Arrival Times	Service Times
All Short Jobs	0, 2, 4, 6, 8	2, 3, 2, 4, 1
Mixed Job Lengths	0, 1, 3, 5, 7	1, 8, 2, 7, 3
Heavy Load	0, 0, 0, 0, 0	10, 2, 8, 6, 4
Light Load With Sporadic Jobs	0, 5, 10, 15, 20	1, 12, 1, 12, 1
Staggered Mix	0, 3, 5, 8, 12	8, 2, 10, 1, 5

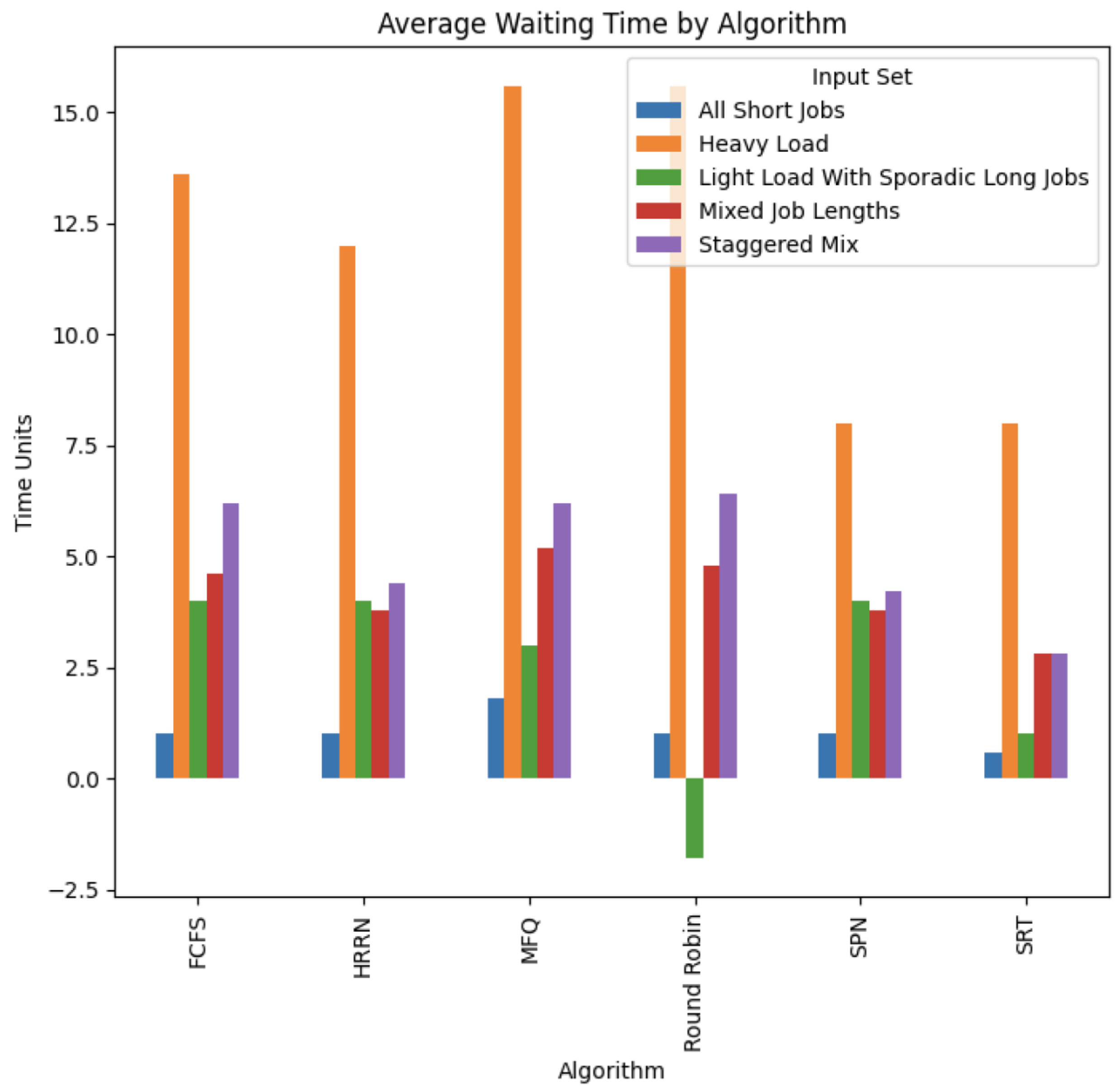
Table 2: Turnaround Time Results

Input Set	Algorithm	Average Turnaround Time	Average Waiting Time
All Short Jobs	FCFS	3.4	1.0
All Short Jobs	Round Robin	3.4	1.0
All Short Jobs	SPN	3.4	1.0
All Short Jobs	SRT	3.0	0.6
All Short Jobs	HRRN	3.4	1.0
All Short Jobs	MFQ	4.2	1.8
Mixed Job Lengths	FCFS	8.8	4.6
Mixed Job Lengths	Round Robin	9.0	4.8
Mixed Job Lengths	SPN	8.0	3.8
Mixed Job Lengths	SRT	7.0	2.8
Mixed Job Lengths	HRRN	8.0	3.8
Mixed Job Lengths	MFQ	9.4	5.2
Heavy Load	FCFS	19.6	13.6
Heavy Load	Round Robin	21.6	15.6
Heavy Load	SPN	14.0	8.0
Heavy Load	SRT	14.0	8.0
Heavy Load	HRRN	18.0	12.0
Heavy Load	MFQ	21.6	15.6
Light Load With Sporadic Jobs	FCFS	9.4	4.0
Light Load With Sporadic Jobs	Round Robin	3.6	-1.8
Light Load With Sporadic Jobs	SPN	9.4	4.0
Light Load With Sporadic Jobs	SRT	6.4	1.0
Light Load With Sporadic Jobs	HRRN	9.4	4.0
Light Load With Sporadic Jobs	MFQ	8.4	3.0
Staggered Mix	FCFS	11.4	6.2
Staggered Mix	Round Robin	11.6	6.4
Staggered Mix	SPN	9.4	4.2

Input Set	**Algorithm**	**Average Turnaround Time**	**Average Waiting Time**
Staggered Mix	SRT	8.0	2.8
Staggered Mix	HRRN	9.6	4.4
Staggered Mix	MFQ	11.4	6.2

Figure 3.18 Average Turnaround Time Comparison


Average Turnaround Time by Algorithm
Input Set
All Short Jobs
Heavy Load
Light Load With Sporadic Long Jobs
Mixed Job Lengths
Staggered Mix
Time Units
0
5
10
15
20
FCFS
HRRN
MFQ
Round Robin
SPN
SRT
Algorithm



- For "All Short Jobs," algorithms like SRT showed improved performance with lower average turnaround times.
- "Mixed Job Lengths" saw better handling by SPN and SRT, demonstrating their efficiency with varied job lengths.
- Under "Heavy Load," SPN and SRT again performed well, indicating their effectiveness in high-load scenarios.

- "Light Load With Sporadic Long Jobs" highlighted discrepancies in the algorithms, with some like Round Robin showing negative waiting times, which indicates potential issues in time quantum settings.
- "Staggered Mix" presented a balanced workload where HRRN and SPN offered better average times.

The results underscore the importance of choosing the right algorithm for a given workload. SPN and SRT generally provided better average times across varied scenarios, indicating their robustness.

Code for analysis:

```
import pandas as pd
import matplotlib.pyplot as plt
from algorithms.fcfs import fcfs
from algorithms.hrrn import hrrn
from algorithms.srt import srt
from algorithms.spn import spn
from algorithms.rr import round_robin
from algorithms.mfq import mlfq as mfq
from algorithms.custom import apsa

# Define the sample inputs
inputs = {
    "All Short Jobs": {
        "arrival_times": [0, 2, 4, 6, 8],
        "service_times": [2, 3, 2, 4, 1],
    },
    "Mixed Job Lengths": {
        "arrival_times": [0, 1, 3, 5, 7],
        "service_times": [1, 8, 2, 7, 3],
    },
    "Heavy Load": {
        "arrival_times": [0, 0, 0, 0, 0],
        "service_times": [10, 2, 8, 6, 4],
    },
    "Light Load With Sporadic Long Jobs": {
        "arrival_times": [0, 5, 10, 15, 20],
        "service_times": [1, 12, 1, 12, 1],
    },
    "Staggered Mix": {
        "arrival_times": [0, 3, 5, 8, 12],
        "service_times": [8, 2, 10, 1, 5],
    },
}

# Define the algorithms
algorithms = {
    "FCFS": fcfs,
    "Round Robin": round_robin,
    "SPN": spn,
    "SRT": srt,
    "HRRN": hrrn,
    "MFQ": mfq,
}

# Define a time quantum for those algorithms that need it
time_quantum1 = 4
time_quantum2 = 8

# Prepare the results table
results_data = {
    "Input Set": [],
    "Algorithm": [],
    "Average Turnaround Time": [],
    "Average Waiting Time": [],
}

# Run the algorithms for each input set and collect results
for input_name, data in inputs.items():
    for algo_name, algo_func in algorithms.items():
        if algo_name == "Round Robin":
            waiting_times, turnaround_times = algo_func(
                data["arrival_times"], data["service_times"], time_quantum1
            )
        elif algo_name == "MFQ":
            waiting_times, turnaround_times = algo_func(
                data["arrival_times"],
                data["service_times"],
                time_quantum1,
                time_quantum2,
            )
        elif algo_name == "APSA":
            waiting_times, turnaround_times = algo_func(
                data["arrival_times"], data["service_times"], 0.5, 10
            )
        else:
            waiting_times, turnaround_times = algo_func(
                data["arrival_times"], data["service_times"]
            )

        avg_turnaround_time = sum(turnaround_times) / len(turnaround_times)
        avg_waiting_time = sum(waiting_times) / len(waiting_times)

        results_data["Input Set"].append(input_name)
        results_data["Algorithm"].append(algo_name)
        results_data["Average Turnaround Time"].append(avg_turnaround_time)
        results_data["Average Waiting Time"].append(avg_waiting_time)

# Convert to DataFrame for easy tabular display
results_df = pd.DataFrame(results_data)
print(results_df)

# Plotting the results
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 7))
results_df.pivot(
    index="Algorithm", columns="Input Set", values="Average Turnaround Time"
).plot(kind="bar", ax=ax1)
ax1.set_title("Average Turnaround Time by Algorithm")
ax1.set_ylabel("Time Units")
```

```
results_df.pivot(
    index="Algorithm", columns="Input Set", values="Average Waiting Time"
).plot(kind="bar", ax=ax2)
ax2.set_title("Average Waiting Time by Algorithm")
ax2.set_ylabel("Time Units")

plt.tight_layout()
plt.show()
```

Conclusion:

This project taps into several types of scheduling, such as FCFS, RR, SJF, SRT, HRRN, MFS and new algorithm. They are effectively coded in the Python language for the specified inputs. In conclusion, this project has been a journey into understanding how different scheduling methods manage computer tasks. By studying and creating these methods, we've learned about their strengths and weaknesses. The project highlights the importance of choosing the right method for the job, as it can make a big difference in efficiency. Our exploration opens opportunities for future improvements in how computers handle tasks, making them faster and more effective.

References

1. https://www.geeksforgeeks.org/multilevel-feedback-queue-scheduling-mlfq-cpu-scheduling/
2. https://www.tutorialspoint.com/operating_system/os_process_scheduling_algorithms.htm