SOMAIYA VIDYAVIHAR UNIVERSITY
K. J. Somaiya College of Engineering, Mumbai -77
(A Constituent College of Somaiya Vidyavihar University)

Batch: A1	**Roll No.: 16010121045**
Experiment / Assignment / Tutorial No: 4	

Experiment: 4

Title: Implementation of Event Scheduling Algorithm for Dump Truck Problem

---

Problem Statement: Dump truck is used to haul load from the entrance of a small mine to the railroad. Each truck is loaded by one of two loaders. After a loading, truck immediately moves to the scale, to be weighed as soon as possible. Both the loaders and the scale have a first-come-first-served waiting line (queue) for trucks. Travel time from a loader to the scale is considered negligible. After being weighed, a truck begins a travel time (during which the truck unloads) and then afterwards returns to the loader queue.

The distribution of loading time, weighing time and travel time are given along with random digits assigned. Simulate the dump truck problem using C/C++/Java.

Expected Outcome of Experiment:

Index	Outcome
CO2	Analyse and apply general principles of event scheduling algorithm & various statistical methods on different applications.

Books/ Journals/ Websites referred:

1. Jerry Banks, John Carson, Barry Nelson, and David M. Nicol, "Discrete Event System Simulation"; Fifth Edition, Prentice-Hall.
2. Averill M Law, "System Modeling & Analysis"; 4th Edition TMH.
3. Banks C M, Sokolowski J A, "Principles of Modeling and Simulation", Wiley

---

Pre Lab/ Prior Concepts:

Theory:

To estimate the loader and scale utilization (% of time busy)

Concept in Discrete event simulation:

1. System: A collection of entities (e.g. peoples & machines) that interact together over

time to accomplish one or more goals.
2. Model: An abstract representation of a system, usually containing structures, logical or mathematical relationships that describe a system in terms of state, entities and their attributes, sets, process, events.
3. System state: A collection of variables that contain all the information necessary to describe the system at any time.
4. Entity: Any objector component in the system that requires explicit representation in the model.
5. Attributes: The properties of a given entity.
6. List: A collection of associated entities (permanent / temporary) associated entities, ordered in some logical fashion.
7. Event: An instantaneous occurrence that changes the state of the system.
8. Event notice: A record of an event to occur at the current / future time, along with any associated data necessary to execute the event.
9. Activity: Duration of time of specified length, which is known when it begins.
10. Delay: A duration of time specified length which is not known until it ends.
11. Clock: A variable representing simulated time.

World View:

Even scheduling approach: When using the event scheduling approach, a simulation analyst concentrates on events & their effect on system state. Loading & weighing are two events which affect the system at loader & weighing queue. Based on the queue is busy or idle, imminent event is loaded.

Process Interaction approach: It describes the lifespan of activities. The analyst defines the simulation model in terms of entities or objects and their life cycle as they flow through the system, demanding resource and queuing to wait for resources. This life cycle consists of various events & activities. It is based on fixed time advance. Disadvantage of it is that we need to scan activity again and again.

Activity scanning approach: It is also known as three phase approach. It considers the activities of duration zero time units. Based on this, activities are categorized as B activities: It includes activities which are bound to occur, all primary events and unconditional activities. C activities: It includes activities or events that are conditional upon certain conditions being true.

In three phase approach, the simulation has
Phase A: Remove imminent event from the FEL and advance the clock to its event time. Remove form FEL any other events that have the same event time.
Phase B: Execute all B-type events that were removed from FEL.
Phase C: Scan the condition that triggers each C type activity and rescan until no additional C- type activities can begin & no events occur.

Conceptual Model:

SOMAIYA VIDYAVIHAR UNIVERSITY
K. J. Somaiya College of Engineering, Mumbai -77
(A Constituent College of Somaiya Vidyavihar University)

1. **System states**: [LQ(t), L(t), WQ(t), W(t)]

Where, LQ(t) = No. of trucks in loader queue
L(t) = No. of trucks (0,1 or 2) being loaded
WQ(t) = No. of trucks in weigh queue
W(t) = No. of trucks (0 or 1) being weighed
All at simulation time t.

2. **Entities**: Six dump trucks [DT1,DT2, … ,DT6]
3. **Lists**: Loading queue - All trucks waiting to begin loading
   Weighing queue – All trucks waiting to be weighed
4. **Events**: An instantaneous occurrence that changes the state of the system.
5. **Event notices**: (ALQ, t, DTi) Dump truck i arrives at loader queue ALQ at time t
(EL, t, DTi) Dump truck i end loading EL at time t
(EW, t, DTi) Dump truck i end weighing EW at time t

6. **Activities**: Loading time, weighing time & travel time
7. **Delay**: Time required at loader queue & scale.

Random Numbers:
Random numbers used in this simulation are for loading time & weighing time of dump truck.

Algorithm / Activity Diagram: (Simulation Approach):

Dump Truck Problem :

```
import random


class Truck:
    def __init__(self,truck_id):
        self.truck_id = truck_id
        self.status = "Init"
        self.time_left = 0



    def get_load_time(self):
        rand = random.random()
        if 0 <= rand <= 0.3:
```

```
        load_time = 5
    elif 0.3 < rand <= 0.8:
        load_time = 10
    else:
        load_time = 15

    return load_time

def get_weigh_time(self):
    rand = random.random()
    if 0 <= rand <= 0.7:
        weigh_time = 12
    else:
        weigh_time = 16

    return weigh_time

def get_travel_time(self):
    rand = random.random()
    if 0 <= rand <= 0.4:
        travel_time = 40
    elif 0.4 < rand <= 0.7:
        travel_time = 60
    elif 0.7 < rand <= 0.9:
        travel_time = 80
    else:
        travel_time = 100
```

```
        return travel_time

def truckParser(Trucks):
    new_list = []
    for truck in Trucks:
        new_list.append(truck.truck_id)

    return new_list

def print_stuff(loader_queue,in_loader,weighing_queue,in_weigh,traveling_trucks):
    print("Loader Queue: " + str(truckParser(loader_queue)))
    print("In Loader: " + str(truckParser(in_loader)))
    print("Weighing_queue: " + str(truckParser(weighing_queue)))
    print("In weighing: " + str(truckParser(in_weigh)))
    print("Travelling: " + str(truckParser(traveling_trucks)))

def simulate_trucks():
    trucks = [Truck(i) for i in range(1, 7)]
    loader_queue = trucks[:]
    in_loader = [] # Only 2 trucks at a time
    weighing_queue = []
    in_weigh = [] # Only 1 truck at a time

    traveling_trucks = []

    Time = 0

    while Time < 100:
```

```
    print(f"Time: {Time}")

    # Start Loading
    for truck in loader_queue:
        if(len(in_loader)<2):
            if truck.status == "Init" or truck.status == "Returning":
                in_loader.append(truck)
                loader_queue.remove(truck)
                truck.status = "Loading"
                truck.time_left = truck.get_load_time()
                print(f"Truck {truck.truck_id} starts loading for {truck.time_left} time units.")

    # Process loading
    for truck in in_loader[:]:
        if truck.status == "Loading":
            truck.time_left -= 1
            if truck.time_left == 0:
                truck.status = "WaitingForWeighing"
                in_loader.remove(truck)
                weighing_queue.append(truck)
                print(f"Truck {truck.truck_id} finished loading and moves to weighing
queue.")

    # Start Weighinh
    for truck in weighing_queue[:]:
        if(len(in_weigh)==0):
            if truck.status == "WaitingForWeighing":
                in_weigh.append(truck)
```

```
                weighing_queue.remove(truck)
                truck.status = "Weighing"
                truck.time_left = truck.get_weigh_time()
                print(f"Truck {truck.truck_id} starts weighing for {truck.time_left} time
units.")

        # Process weighing
        for truck in in_weigh[:]:
            if truck.status == "Weighing":
                truck.time_left -= 1
                if truck.time_left == 0:
                    truck.status = "Traveling"
                    in_weigh.remove(truck)
                    traveling_trucks.append(truck)
                    truck.time_left = truck.get_travel_time()
                    print(f"Truck {truck.truck_id} finished weighing and starts traveling for
{truck.time_left} time units.")

        # Process traveling trucks
        for truck in traveling_trucks[:]:
            if truck.status == "Traveling":
                truck.time_left -= 1
                if truck.time_left == 0:
                    truck.status = "Returning"
                    traveling_trucks.remove(truck)
                    loader_queue.append(truck)
                    print(f"Truck {truck.truck_id} finished traveling and returns to loader
queue.")
```

```
        Time += 1
        print_stuff(loader_queue,in_loader,weighing_queue,in_weigh,traveling_trucks)

simulate_trucks()
```

```
python3 -u "/Users/pargatsinghdhanjal/Sem 7/CSM/exo4.py"
❯ python3 -u "/Users/pargatsinghdhanjal/Sem 7/CSM/exo4.py"
Time: 0
Truck 1 starts loading for 15 time units.
Truck 3 starts loading for 5 time units.
Loader Queue: [2, 4, 5, 6]
In Loader: [1, 3]
Weighing_queue: []
In weighing: []
Travelling: []
Time: 1
Loader Queue: [2, 4, 5, 6]
In Loader: [1, 3]
Weighing_queue: []
In weighing: []
Travelling: []
Time: 2
Loader Queue: [2, 4, 5, 6]
In Loader: [1, 3]
Weighing_queue: []
In weighing: []
Travelling: []
Time: 3
Loader Queue: [2, 4, 5, 6]
In Loader: [1, 3]
Weighing_queue: []
In weighing: []
Travelling: []
Time: 4
Truck 3 finished loading and moves to weighing queue.
Truck 3 starts weighing for 12 time units.
Loader Queue: [2, 4, 5, 6]
In Loader: [1]
Weighing_queue: []
In weighing: [3]
Travelling: []
Time: 5
Truck 2 starts loading for 10 time units.
Loader Queue: [4, 5, 6]
In Loader: [1, 2]
Weighing_queue: []
In weighing: [3]
Travelling: []
Time: 6
Loader Queue: [4, 5, 6]
In Loader: [1, 2]
Weighing_queue: []
In weighing: [3]
Travelling: []
Time: 7
Loader Queue: [4, 5, 6]
In Loader: [1, 2]
Weighing_queue: []
In weighing: [3]
Travelling: []
Time: 8
Loader Queue: [4, 5, 6]
In Loader: [1, 2]
Weighing_queue: []
In weighing: [3]
Travelling: []
Time: 9
Loader Queue: [4, 5, 6]
```

The Event scheduling algorithm:

The sequence of actions which a simulation language must perform to advance the clock and build a new system snapshot is called Event scheduling algorithm / Time advance algorithm.

STEP 1. Remove the event notice for the imminent event from FEL
STEP 2. Advance clock to imminent event time.
STEP 3. Execute imminent event, update system state, change entity attribute and set membership as needed
STEP 4. Generate future events and place their event notices on FEL, ranked by event time.
STEP 5. Update cumulative statistics and counters.

Conclusion:

The implementation of the event scheduling algorithm for the dump truck problem effectively models the operation of a small mine, where dump trucks are loaded, weighed, and then proceed to unload. By simulating this system using event scheduling, we can observe the impact of various factors such as loading times, weighing times, and queue lengths on the overall efficiency of the process.

The conceptual model, which includes system states, entities, and events, allows for a detailed analysis of the mine's operations. The event scheduling approach, particularly focusing on key events such as loading and weighing, provides an efficient way to simulate and analyze the system's behavior over time. This approach enables the identification of bottlenecks, such as excessive waiting times in queues, and helps in evaluating the performance of the system under different conditions.

Furthermore, the use of random numbers for loading and weighing times introduces variability into the simulation, reflecting the real-world uncertainty and variability in these processes. The results of the simulation can be used to optimize the operation of the mine, potentially leading to reduced waiting times, increased throughput, and improved overall efficiency.

In summary, the event scheduling algorithm serves as a powerful tool for analyzing and optimizing complex systems like the dump truck operation in a mine, providing valuable insights that can guide decision-making and operational improvements.

SOMAIYA VIDYAVIHAR UNIVERSITY
K. J. Somaiya College of Engineering, Mumbai -77
(A Constituent College of Somaiya Vidyavihar University)

Post Lab Questions:

Using the event scheduling approach, do the manual simulation of Able Baker call center problem.

```
import random
import heapq

# Time between calls (in minutes) from Table 10
time_between_calls = [1, 2, 3, 4]

# Service times for Able (in minutes) from Table 11
able_service_times = [2, 3, 4]

# Service times for Baker (in minutes) from Table 12
baker_service_times = [3, 4, 5]

# Initialize the event list (FEL) as a priority queue
event_list = []

# Statistics
total_wait_time = 0
total_idle_time_able = 0
total_idle_time_baker = 0
calls_served_by_able = 0
calls_served_by_baker = 0

# Simulation clock
current_time = 0
```

```
# Able and Baker availability
able_available = True
baker_available = True

# Generate the first call event
next_call_time = random.choice(time_between_calls)
heapq.heappush(event_list, (next_call_time, 'call'))

# Simulation for the first 100 calls
for _ in range(100):
    # Get the imminent event
    event_time, event_type = heapq.heappop(event_list)

    # Advance the clock to the time of the imminent event
    current_time = event_time

    if event_type == 'call':
        if able_available:
            # Assign the call to Able
            service_time = random.choice(able_service_times)
            calls_served_by_able += 1
            able_available = False
            heapq.heappush(event_list, (current_time + service_time, 'able_done'))
        elif baker_available:
            # Assign the call to Baker
            service_time = random.choice(baker_service_times)
            calls_served_by_baker += 1
            baker_available = False
```

```
            heapq.heappush(event_list, (current_time + service_time, 'baker_done'))
        else:
            # If both are busy, increment wait time (not storing it separately here)
            pass

        # Schedule the next call
        next_call_time = current_time + random.choice(time_between_calls)
        heapq.heappush(event_list, (next_call_time, 'call'))

    elif event_type == 'able_done':
        able_available = True
        if not event_list:  # If no more events are scheduled
            total_idle_time_able += 1

    elif event_type == 'baker_done':
        baker_available = True
        if not event_list:  # If no more events are scheduled
            total_idle_time_baker += 1

# Calculate and display statistics
print("Total Calls Served by Able:", calls_served_by_able)
print("Total Calls Served by Baker:", calls_served_by_baker)
print("Total Idle Time of Able:", total_idle_time_able)
print("Total Idle Time of Baker:", total_idle_time_baker)
```

```
❯ python3 -u "/Users/pargatsinghdhanjal/Sem 7/CSM/exo4.py"
Total Calls Served by Able: 33
Total Calls Served by Baker: 16
Total Idle Time of Able: 0
Total Idle Time of Baker: 0
```