```python
# Practical-4: A supermarket wants to optimize its checkout process to minimize customer wait times and ensure efficient allocation of cashiers. Management has observed that customer arrivals follow a Poisson distribution with an average rate of 10 customers per hour. They want to determine the optimal number of checkout counters to open to minimize customer wait times and maximize resource utilization.

# Name: Angat Shah
# Enrollment No: 202203103510097
# Branch: B.Tech Computer Science and Engineering

import random
import math

def generateCustomerArrivals(avgArrivalRate, simulationTime):
    arrivals = []
    totalArrivals = 0
    time = 0
    while time < simulationTime:
        interArrivalTime = -1 / avgArrivalRate * math.log(random.random())
        time += interArrivalTime
        if time < simulationTime:
            arrivals.append(time)
            totalArrivals += 1
        else:
            break
    return arrivals


def generateExponential(avgServiceRate):
    rand = random.random()
    return -math.log(1 - rand) / avgServiceRate


def simulateCheckoutProcess(numCounters, avgServiceRate, arrivalTimes):
    checkoutCounters = [[] for _ in range(numCounters)]
    waitTimes = []

    for arrivalTime in arrivalTimes:
        minWaitCounter = min(range(numCounters), key=lambda i: len(checkoutCounters[i]))

        if checkoutCounters[minWaitCounter]:
            lastCheckoutTime = max(checkoutCounters[minWaitCounter])
            waitTime = max(0, lastCheckoutTime - arrivalTime)
        else:
            waitTime = 0

        waitTimes.append(waitTime)

        serviceTime = generateExponential(avgServiceRate)
        checkoutTime = arrivalTime + waitTime + serviceTime

        checkoutCounters[minWaitCounter].append(checkoutTime)

    return waitTimes, checkoutCounters

def evaluatePerformance(waitTimes, checkoutCounters, simulationTime):
    totalWaitTime = sum(waitTimes)
    avgWaitTime = totalWaitTime / len(waitTimes)

```

```
    totalServiceTime = sum(
        [max(counter, default=simulationTime) - min(counter, default=simulationTime) for counter in checkoutCounters]
    )
    utilization = totalServiceTime / (len(checkoutCounters) * simulationTime)

    return avgWaitTime, utilization

def optimizeCheckoutProcess(avgArrivalRate, avgServiceTime, simulationTime):
    avgServiceRate = 1 / avgServiceTime
    bestNumCounters = None
    minAvgWaitTime = float('inf')
    maxUtilization = float('-inf')

    for numCounters in range(1, 11):
        arrivalTimes = generateCustomerArrivals(avgArrivalRate, simulationTime)
        waitTimes, checkoutCounters = simulateCheckoutProcess(numCounters, avgServiceRate, arrivalTimes)
        avgWaitTime, utilization = evaluatePerformance(waitTimes, checkoutCounters, simulationTime)

        if avgWaitTime < minAvgWaitTime:
            minAvgWaitTime = avgWaitTime
            maxUtilization = utilization
            bestNumCounters = numCounters

    return bestNumCounters, minAvgWaitTime, maxUtilization

def main():
    avgArrivalRate = 10
    avgServiceTime = 4
    avgServiceRate = avgServiceTime / 60
    simulationTime = 8

    bestNumCounters, minAvgWaitTime, maxUtilization = optimizeCheckoutProcess(avgArrivalRate,avgServiceRate,simulationTime)

    print("--> Optimal Number of Checkout Counters: {}".format(bestNumCounters))
    print("--> Minimum Average Customer Wait Time: {}".format(minAvgWaitTime))
    print("--> Optimal Utilization of Checkout Counters at Peak Capacity.: {}\n".format(maxUtilization))

main()

print("\n-*-*-*-*-*-END OF PRACTICAL 4-*-*-*-*-*-\n")
```