Dillon Hughes

Programming Assignment 1

CSF T/Th @ 11am

dah249

**Problem 1**

**Overview:**

“P1.c”: This C file contains a program that generates a workload simulation for a system. The system receives processes, wherein the arrival times of these processes follow a Poisson distribution, and the requested CPU service times follow an Exponential distribution.

**Program Features:**

Uniform Random Generation: The code employs the rand () function to produce uniformly distributed random numbers.

Exponential Distribution Mapping: The generated numbers are then transformed into exponentially distributed random values using the exponential\_rand function.

**Program Functionality:**

Simulating Arrival Times: The inter-arrival times for the processes are modeled by employing a Poisson distribution. This is achieved by converting uniform random numbers into exponentially distributed ones. The expected average arrival rate is defined to be 2 processes per second.

Service Time Simulation: For every individual process, the required CPU service duration is modeled to be distributed exponentially. The expected average of this service time is set to 1 second.

**Output Structure:**

The code is constructed to display, for every process, a tuple that comprises its process ID, arrival time, and the requested service time. After this, the actual computed average arrival rate, and the service time, based on the simulation, are presented.

Example output:

Process ID Arrival Time Requested Service Time

---------------------------------------------------

1 0.09 0.93

2 0.21 0.23

…

999 491.44 1.65

1000 491.79 0.39

Actual Average Arrival Rate: 2.03 processes per second

Actual Average Service Time: 1.01 seconds

**Conclusion:**

The program offers a precise simulation of system workloads, capturing both the arrival and service times of processes in line with the Poisson and Exponential distributions, respectively. The output provides a glimpse of the simulated processes along with important average metrics. This serves as a foundational pillar for more advanced system simulations, workload analytics, or enhancements in future projects.

**Problem 2**

**Overview:**

The P2.py contains a program designed to simulate server failures and restorations, adhering to the exponential distribution, over a span of 20 years.

**Program Features:**

Exponential Distribution Simulation: Utilizing random.random(), uniform random numbers are mapped to an exponential distribution through the generate\_exponential function.

Server Failure and Restoration: Failure and restoration times of two mirrored servers are simulated, adhering to specified MTBF and restoration time parameters.

**Program Functionality:**

Simulating Uptimes and Downtimes: The program uses the exponential distribution to simulate continuous uptimes and deterministic downtimes (10 hours for restoration) of two servers for 20 years, ensuring the logic adheres to an MTBF of 500 hours.

System Failure Check: Through the check\_system\_failure function, the program identifies instances where both servers fail within the same 10-hour window.

Output Insights: The program outputs detailed failure and restoration times for both servers and calculates and presents an average time until a system failure occurs.

**Output Structure:**

Detailed failure and restoration times for both servers over the 20-year period.

The average time until a total system failure (both servers being down concurrently) occurs, based on multiple simulations with varied seeds.

Example Output:

Server 1 Server 2

--------------------------------------------------

Failure Time Restoration Time Failure Time Restoration Time

296.54 306.54 380.65 390.65

…

167250.66 167260.66 172952.74 172962.74

167583.59 167593.59 173055.47 173065.47

Average time until system failure: 14020.84 hours

**Conclusion:**

In this assignment, I used Python to create a straightforward simulation of two servers over 20 years. With a clear focus on failures and repair times, the code efficiently produces a useful dataset and helps predict when both servers might fail at the same time. This exercise offers insights into managing server reliability and gives us a peek into the world of IT infrastructure and its unexpected challenges. Moving forward, it provides a basis for exploring more about how to keep systems running smoothly and avoid simultaneous server failures in real-world scenarios.