CS-329 - Operating Systems

Complex Engineering Problem

# Problem Chosen

Problem 1: Cigarette smokers’ problem.

# Code

# Global variables

INGREDIENTS\_NAMES *=* {'tobacco', 'paper', 'matches'}

ingredients\_semaphores *=* {

    'tobacco\_paper': threading.Semaphore(0),

    'tobacco\_matches': threading.Semaphore(0),

    'paper\_matches': threading.Semaphore(0)

    }

agent\_semaphore *=* threading.Semaphore(0)

class Agent(threading.Thread):

    def run(self):

*while* *True*:

            # Choose ingredient pair to make available to smokers

*try*:

                random\_seed *=* int(input('\nEnter a number for the random seed: ')) *\** 100

*except* ValueError:

                print("Invalid input, setting random seed to 0")

                random\_seed *=* 0

            random.seed(random\_seed)

            ingredient\_pair *=* random.choice(list(ingredients\_semaphores.keys()))

            # Make the ingredient pair available to smokers

            ingredients\_semaphores[ingredient\_pair].release()  *# call sem signal on ingredients*

            print(f"Agent makes {ingredient\_pair} available to smokers")

            # Wait for the smoker that has the complementary ingredient to

            # pick up the ingredients, make cigarette, and finish smoking

            ingredients\_semaphores[ingredient\_pair].acquire()  *# call sem wait on ingredients*

class Smoker(threading.Thread):

    def \_\_init\_\_(self, name, ingredient):

        super().\_\_init\_\_()

        self.name *=* name

*if* ingredient *==* 'tobacco':

            self.required\_ingredient\_pair *=* 'paper\_matches'

*elif* ingredient *==* 'paper':

            self.required\_ingredient\_pair *=* 'tobacco\_matches'

*elif* ingredient *==* 'matches':

            self.required\_ingredient\_pair *=* 'tobacco\_paper'

    def run(self):

*while* *True*:

            # Wait for complimentary ingredients to be available

            print(f"{self.name} waits for {self.required\_ingredient\_pair}")

            ingredients\_semaphores[self.required\_ingredient\_pair].acquire()  *# sem wait on required ingredient pair semaphore*

            # Make cigarette and smoke it

            print(f"{self.name} makes a cigarette and smokes it")

            time.sleep(random.uniform(0.5, 1.5))

            # Signal agent that the ingredients pair is used up

            print(f"{self.name} signals to agent to make next ingredients pair")

            ingredients\_semaphores[self.required\_ingredient\_pair].release()  *# sem signal on required ingredient pair semaphore*

def main():

    # Create agent and smokers

    agent *=* Agent()

    smoker1 *=* Smoker('Smoker with tobacco', 'tobacco')

    smoker2 *=* Smoker('Smoker with paper', 'paper')

    smoker3 *=* Smoker('Smoker with matches', 'matches')

    smokers *=* {smoker1, smoker2, smoker3}

    # Start threads

*for* smoker *in* smokers:

        smoker.start()

    agent.start()

    # Wait for threads to finish

*for* smoker *in* smokers:

        smoker.join()

    agent.join()

*if* \_\_name\_\_ *==* '\_\_main\_\_':

    main()

    print('Done')

    exit(0)

# Test Cases

For all test cases, the behavior of agents and smokers should be as expected, i.e. as demanded by the problem statement. The individual test cases are based on different types of inputs the user may give. Regardless of input the behavior of agents and smokers should be correct and program shouldn’t crash.

## Test Case 1

Input: User enters the same valid integer value for random seed every time

Expectation: Agent should produce same pair of ingredients each time and the same smoker should pick them up each time.

### Execution:

Text

Description automatically generated

Output: On user entering 4 each time, agent generates tobacco and matches each time and smoker with paper picks the ingredients up each time and smokes a cigarette.

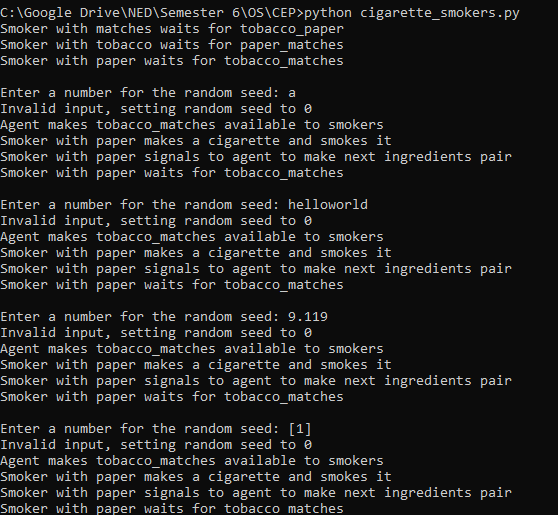
Result: **Test Case Passed.**

## Test Case 2

Input: User enters invalid, i.e., non-integer values (characters, strings, floats, etc.)

Expectation: Program should handle invalid inputs and use the default value of 0 for random seed on each wrong input leading to similar kind of agent and smoker behavior as in Test Case 1 as seed would be same each time.

### Execution:

****

Output: As expected.

Result: **Test Case Passed.**

## Test Case 3

Input: User enters different but valid positive integer values each time for random seed.

Expectation: Agent should generate different pairs of ingredients some times but may produce same pairs for some different inputs (as there are just 3 possible pairs). The correct agent should pick up the ingredient pair each time.

### Execution:

Text

Description automatically generated

Output: Same as expected.

Result: **Test Case Passed.**

## Test Case 4

Input: User enters different but valid positive and negative integer values each time for random seed.

Expectation: Same as Test Case 3, as negative integers are allowed.

### Execution:

Text

Description automatically generated

Output: As expected.

Result: **Test Case Passed.**