import threading

import random

import time

class Frame:

    def \_\_init\_\_(self, seq\_num):

        self.seq\_num = seq\_num

class Sender:

    WINDOW\_SIZE = 4

    MAX\_FRAMES = 10

    TIMEOUT = 2  # seconds

    def \_\_init\_\_(self):

        self.frames = [Frame(i) for i in range(self.MAX\_FRAMES)]

        self.base = 0

        self.next\_seq\_num = 0

        self.ack\_received = [False] \* self.MAX\_FRAMES

        self.lock = threading.Lock()

        self.timer = None

        self.stop\_sending = False

    def send\_frame(self, frame):

        print(f"Sending frame: {frame.seq\_num}")

        if random.random() < 0.9:  # 90% chance to "send" successfully

            print(f"Frame {frame.seq\_num} sent.")

        else:

            print(f"Frame {frame.seq\_num} lost.")

    def receive\_ack(self, ack\_num):

        with self.lock:

            if ack\_num < self.MAX\_FRAMES and not self.ack\_received[ack\_num]:

                self.ack\_received[ack\_num] = True

                print(f"ACK received for frame {ack\_num}")

                while self.base < self.MAX\_FRAMES and self.ack\_received[self.base]:

                    self.base += 1

                if self.base < self.next\_seq\_num and not self.stop\_sending:

                    self.start\_timer()

    def start\_timer(self):

        if self.timer is not None:

            self.timer.cancel()

        self.timer = threading.Timer(self.TIMEOUT, self.timeout)

        self.timer.start()

        print(f"Timer started at {self.base}")

    def timeout(self):

        with self.lock:

            print(f"Timeout: Resending frames {self.base} to {self.next\_seq\_num - 1}")

            for i in range(self.base, self.next\_seq\_num):

                self.send\_frame(self.frames[i])

            self.start\_timer()

    def run(self):

        def sender\_thread():

            while self.base < self.MAX\_FRAMES and not self.stop\_sending:

                with self.lock:

                    while self.next\_seq\_num < self.base + self.WINDOW\_SIZE and self.next\_seq\_num < self.MAX\_FRAMES:

                        self.send\_frame(self.frames[self.next\_seq\_num])

                        self.next\_seq\_num += 1

                        if self.base == self.next\_seq\_num - 1:

                            self.start\_timer()

                time.sleep(0.1)  # Adjusted sleep time for responsiveness

        threading.Thread(target=sender\_thread).start()

    def stop(self):

        self.stop\_sending = True

        if self.timer is not None:

            self.timer.cancel()

class Receiver:

    def \_\_init\_\_(self, sender):

        self.sender = sender

        self.expected\_seq\_num = 0

        self.stop\_receiving = False

    def receive\_frame(self, frame):

        print(f"Received frame: {frame.seq\_num}")

        if frame.seq\_num == self.expected\_seq\_num:

            if random.random() < 0.9:  # 90% chance to "acknowledge" successfully

                print(f"Acknowledging frame {frame.seq\_num}")

                self.sender.receive\_ack(frame.seq\_num)

                self.expected\_seq\_num += 1

            else:

                print(f"ACK for frame {frame.seq\_num} lost.")

        else:

            print(f"Unexpected frame {frame.seq\_num}. Expected {self.expected\_seq\_num}. Ignoring.")

    def run(self):

        def receiver\_thread():

            while not self.stop\_receiving and self.sender.base < Sender.MAX\_FRAMES:

                for i in range(self.sender.base, self.sender.next\_seq\_num):

                    if not self.sender.ack\_received[i]:

                        self.receive\_frame(self.sender.frames[i])

                time.sleep(0.5)

        threading.Thread(target=receiver\_thread).start()

    def stop(self):

        self.stop\_receiving = True

if \_\_name\_\_ == "\_\_main\_\_":

    sender = Sender()

    receiver = Receiver(sender)

    # Start the sender and receiver

    sender.run()

    receiver.run()

    # Let the simulation run for a certain period

    try:

        while sender.base < Sender.MAX\_FRAMES:

            time.sleep(1)

    except KeyboardInterrupt:

        pass

    finally:

        sender.stop()

        receiver.stop()

        print("Simulation ended.")

