Project Program

```python
import cv2
import numpy as np
import tensorflow as tf
from concurrent.futures import ThreadPoolExecutor
import threading

# Load the Haar Cascade classifier for face detection (you can use a more advanced face detection model)
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')

# Load your actual emotion recognition model here
# Replace this placeholder code with your model loading code
emotion_model = tf.keras.models.load_model("C:\\Users\\jeeva\\Downloads\\FacialExpressionModel.h5")  # Replace with the path to your model

# Define a dictionary to map emotion class numbers to emotion labels
emotion_labels = {
    0: "Angry",
    1: "Disgust",
    2: "Fear",
    3: "Happy",
    4: "Sad",
    5: "Surprise",
    6: "Neutral"
}

# Load YOLO for weapon detection
weapon_net = cv2.dnn.readNet("C:/Users/jeeva/Downloads/yolov3_training_2000.weights", "C:/Users/jeeva/Downloads/yolov3_testing.cfg.txt")
weapon_net.setPreferableBackend(cv2.dnn.DNN_BACKEND_DEFAULT)
weapon_net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)
weapon_classes = ["Weapon"]

# Increase frame size for larger webcam display
frame_width = 640
frame_height = 480

# Open a connection to the camera with increased frame size
cap = cv2.VideoCapture(0)
cap.set(3, frame_width)
cap.set(4, frame_height)

# Lock for synchronization
lock = threading.Lock()
```

```
# Variables to store latest results
emotion_result = ""
weapon_result = ""
suspicious_activity_detected = False  # Flag for suspicious activity
detection

# Function for face emotion recognition
def recognize_emotion(face_frame):
    global emotion_result, suspicious_activity_detected
  # Resize the face for emotion recognition
    preprocessed_face = cv2.resize(face_frame, (96, 96))

  # Perform any necessary normalization or scaling here if required by your model

  # Replace this with your actual emotion recognition code using the loaded model
    emotion_probs =
emotion_model.predict(np.expand_dims(preprocessed_face, axis=0))[0]
    detected_emotion = np.argmax(emotion_probs)  # Assuming your
model returns class probabilities

    # Get the emotion label from the dictionary
    emotion_label = emotion_labels[detected_emotion]

    with lock:
        emotion_result = f"Emotion: {emotion_label}"

        # Check for suspicious emotions (anger, fear, stress)
        if emotion_label in ["Angry", "Fear", "Stress"]:
            suspicious_activity_detected = True
        else:
            suspicious_activity_detected = False  # Clear the flag
if no suspicious emotion detected

# Function for weapon detection
def detect_weapon(frame):
    global weapon_result, suspicious_activity_detected
    height, width, channels = frame.shape

    # Detecting objects
    blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0,
0), True, crop=False)
    weapon_net.setInput(blob)

    layer_names = weapon_net.getLayerNames()
    output_layers = [layer_names[i - 1] for i in
weapon_net.getUnconnectedOutLayers()]
    colors = (0, 255, 0)  # Set color to green (0, 255, 0)
    outs = weapon_net.forward(output_layers)

    # Showing information on the screen
    weapon_class_ids = []
```

```
    weapon_confidences = []
    weapon_boxes = []

    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.6 and weapon_classes[class_id] ==
"Weapon":   # Adjust confidence threshold
                # Object detected
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)

             # Rectangle coordinates
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)

                weapon_boxes.append([x, y, w, h])
                weapon_confidences.append(float(confidence))
                weapon_class_ids.append(class_id)

    indexes = cv2.dnn.NMSBoxes(weapon_boxes, weapon_confidences,
0.5, 0.4)

    for i in range(len(weapon_boxes)):
        if i in indexes:
            x, y, w, h = weapon_boxes[i]
            label = str(weapon_classes[weapon_class_ids[i]])
            cv2.rectangle(frame, (x, y), (x + w, y + h), colors, 2)
            cv2.putText(frame, label, (x, y + 30),
cv2.FONT_HERSHEY_PLAIN, 3, colors, 3)

    with lock:
        if len(weapon_boxes) > 0:
            weapon_result = "Weapon Detected"
            suspicious_activity_detected = True
        else:
            weapon_result = ""  # Clear the weapon detection message

# Create a thread pool with a limited number of threads
executor = ThreadPoolExecutor(max_workers=4)

# Main loop for processing frames
frame_count = 0
frame_skip = 2  # Process every second frame

while True:
    ret, frame = cap.read()  # Read a frame from the camera
```

```
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Limit processing frequency for better performance
    if cv2.waitKey(10) & 0xFF == ord('q'):
        break

    # Use downscaled frame for face detection
    small_gray = cv2.resize(gray, (0, 0), fx=0.25, fy=0.25)
    faces = face_cascade.detectMultiScale(small_gray,
scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    for (x, y, w, h) in faces:
        # Scale face coordinates back to the original frame size
        x *= 4
        y *= 4
        w *= 4
        h *= 4

        face = frame[y:y+h, x:x+w]

        # Process face emotion recognition using the thread pool
    executor.submit(recognize_emotion, face)

    # Draw the rectangle around the face and display emotion result
        cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)
# Draw a blue rectangle around the face
        cv2.putText(frame, emotion_result, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)  # Display emotion
result

    frame_count += 1
    if frame_count % frame_skip == 0:
        # Detect weapons in the frame in the main thread
        detect_weapon(frame)

        # Display the results on the screen
        with lock:
            cv2.putText(frame, weapon_result, (10, 40),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)  # Display weapon
detection result
            if suspicious_activity_detected:
                cv2.putText(frame, "Suspicious Activity Detected",
(10, 80), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)  # Display
suspicious activity message
        cv2.imshow('Security Monitoring', frame)

# Release resources
cap.release()
cv2.destroyAllWindows()
```

```
executor.shutdown()  # Properly shut down the thread pool
```

Working of the code

Face Detection and Emotion Recognition

- Explanation:

- We begin by bringing in some special tools, like OpenCV, NumPy, and TensorFlow.

- Then, we load a "face detector" and a model that can recognize emotions in people's faces.

Weapon Detection Setup

- Explanation:

- Next, we introduce a tool called YOLO, which can spot objects, including weapons, in real-time.

- We make YOLO work even better by adjusting some of its settings.

- Now, we're preparing our system to spot weapons using YOLO.

Video Input and Locking Mechanism

- Explanation:

- Our code needs to "see" things, so we connect it to a camera and set up the camera to give us the best view.

- We also use a special lock to make sure different parts of our code don't mess each other up.

Emotion Recognition Function

- Explanation:

- Inside our code, there's a special part that can look at a person's face and tell us how they're feeling.

- The code detects faces in video frames using a Haar Cascade classifier.

- It utilizes a pre-trained emotion recognition model to analyze facial expressions.

- The recognized emotion is determined by selecting the class with the highest probability.

- The code checks for suspicious emotions like "Angry," "Fear," or "Stress" among recognized emotions.

- If any suspicious emotion is detected, it activates a flag (`suspicious_activity_detected`).

- This flag triggers security measures or alerts for a proactive response to potential threats based on detected emotional cues.

Weapon Detection Function

- Explanation:

- We're starting the process of finding weapons. To do that, we get the image ready for YOLO.
- YOLO then looks at the image and tries to find objects, including weapons.

- After that, we pick out the objects that are probably weapons.
- We also manage the results and display them so we can see what's going on.
- We make it easy to see where the weapons are by drawing boxes around them on the screen.

Multithreading and Frame Processing

- Explanation:

- Our code is like a team that can do many things at once to be faster. The main loop keeps showing us new frames on the screen.
- To work faster, we have a team of threads that can do different jobs at the same time.

Real-time Display and User Interaction

- **Explanation:**

- We're watching everything happening in real-time on the screen.

- If we want to stop the program, we just press 'q' on the keyboard