EAST WEST UNIVERSITY

Project Report

Iced tea and coffee problem

Group 01

Course Name:	Operating System
Course Code:	CSE325
Section:	01
Semester:	Summer2024
Course instructor:	**Dr. Md. Nawab Yousuf Ali** Professor Department of Computer Science & Engineering

Name	ID
Rafia Noor Nithin	2022-2-60-120
Fahimah Karim Oishi	2022-2-60-134
Sanjana Kazi Supti	2022-2-60-036

Submission Date: 22 September,2024

Index

Problem Statement

In a charming coffee shop, two friends, Person 1 and Person 2, gather regularly to catch up over their favorite iced coffee. However, Person 2 has a habit of mistakenly calling it "iced tea," leading to some confusion. To streamline their conversations and enhance their experience, we need to create a communication system that addresses these challenges.

The system should achieve the following:

1. Establish a turn-taking system that allows only one person to speak at a time, ensuring that the other listens attentively, thus promoting clear dialogue.
2. Employ semaphores to control access to speaking and listening roles, preventing any interruptions during their discussions.
3. Implement an error detection and correction mechanism that identifies when Person 2 refers to "iced coffee" as "iced tea," automatically correcting it to maintain clarity in the conversation.
4. Simulate the conversation using threads or processes, providing an output that reflects each participant’s contributions along with corrections made to Person 2’s statements.
5. Keep a record of the number of errors detected and corrected throughout the conversation, demonstrating effective communication and reducing misunderstandings.

This system will create a seamless and enjoyable conversational experience for the two friends as they bond over their shared love for iced coffee.

Project Description

Design a C program that simulates a conversation between two friends, Person 1 and Person 2, who regularly meet in a coffee shop. The goal is to ensure that both friends speak without interrupting each other, while correcting Person 2's tendency to misspell "iced coffee" as "iced tea." The program will use semaphores, threads, and processes to coordinate the conversation, ensuring proper turn-taking and error detection.

Requirements:

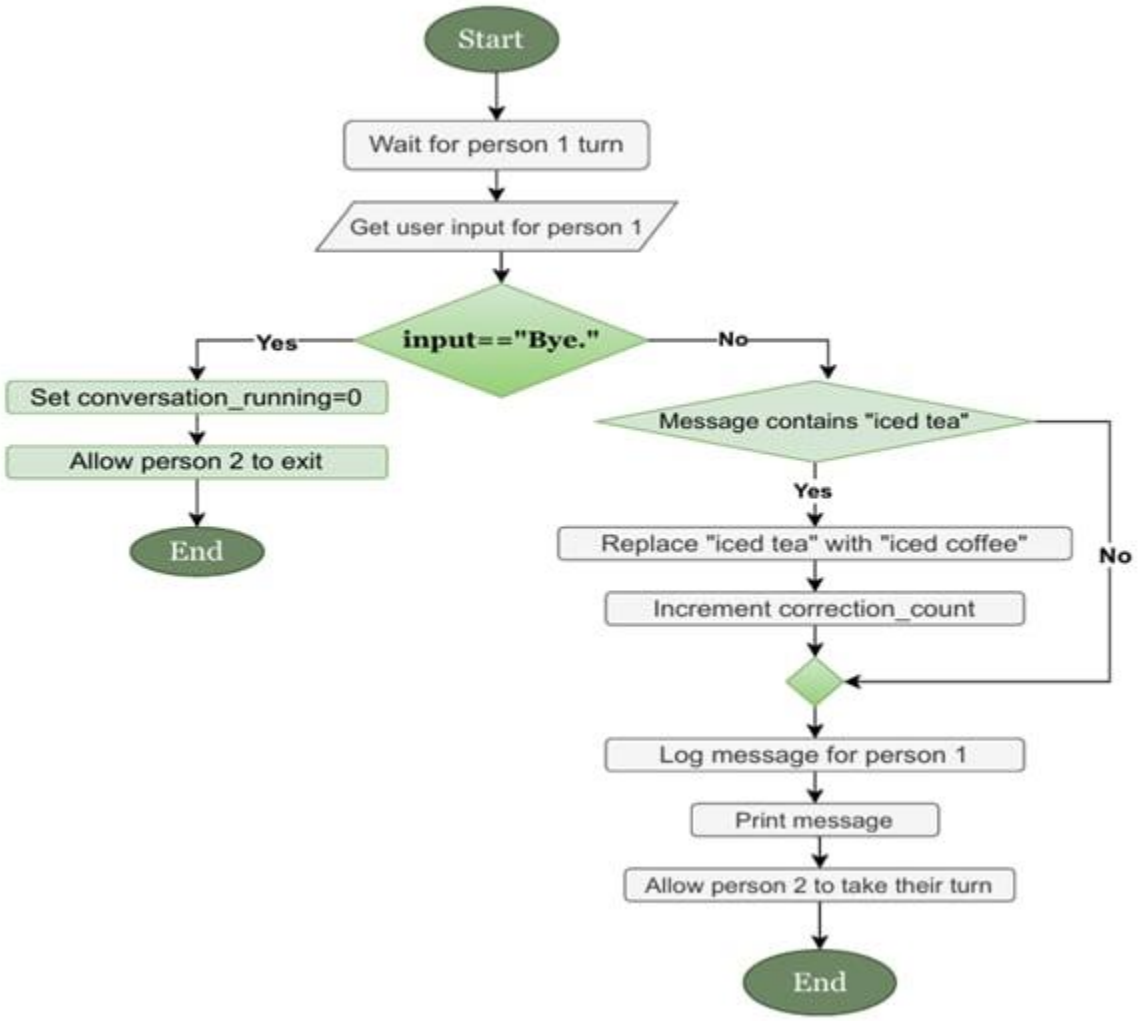
1. **Conversation Simulation**: The program simulates a dialogue between two friends, with each speaking in turn.
2. **Turn-Taking Mechanism**: Implement semaphores to ensure that only one person speaks at a time, allowing the other to listen.
3. **Error Detection and Correction**: Automatically detect and correct Person 2's misspelling of "iced coffee" as "iced tea."
4. **Threads/Processes**: Use threads or processes to manage the conversation and implement error correction.
5. **Output**: Display the corrected conversation and track the number of errors detected and corrected.

Functional Approach:

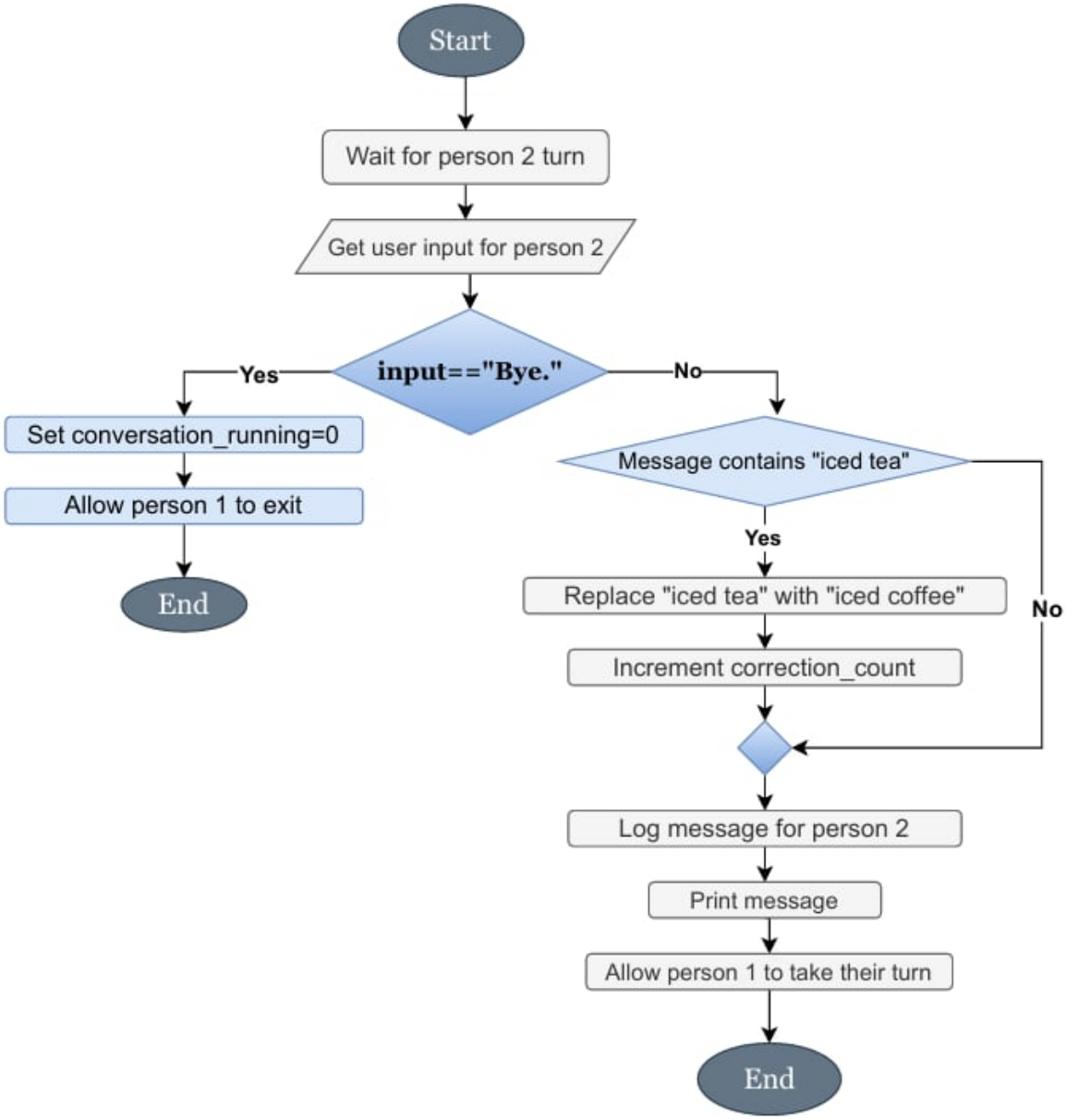
- **Semaphores**: Manage mutual exclusion, ensuring only one friend speaks while the other listens.
- **Person 1 and Person 2**: Implement as threads or processes, where Person 1 always says "I enjoy iced coffee" and Person 2 may say "iced tea."
- **Error Correction Logic**: Monitor Person 2's messages, replacing "iced tea" with "iced coffee" and counting corrections.
- **Infinite Conversation**: Allow the dialogue to continue indefinitely, simulating ongoing discussions.

Flow Chart

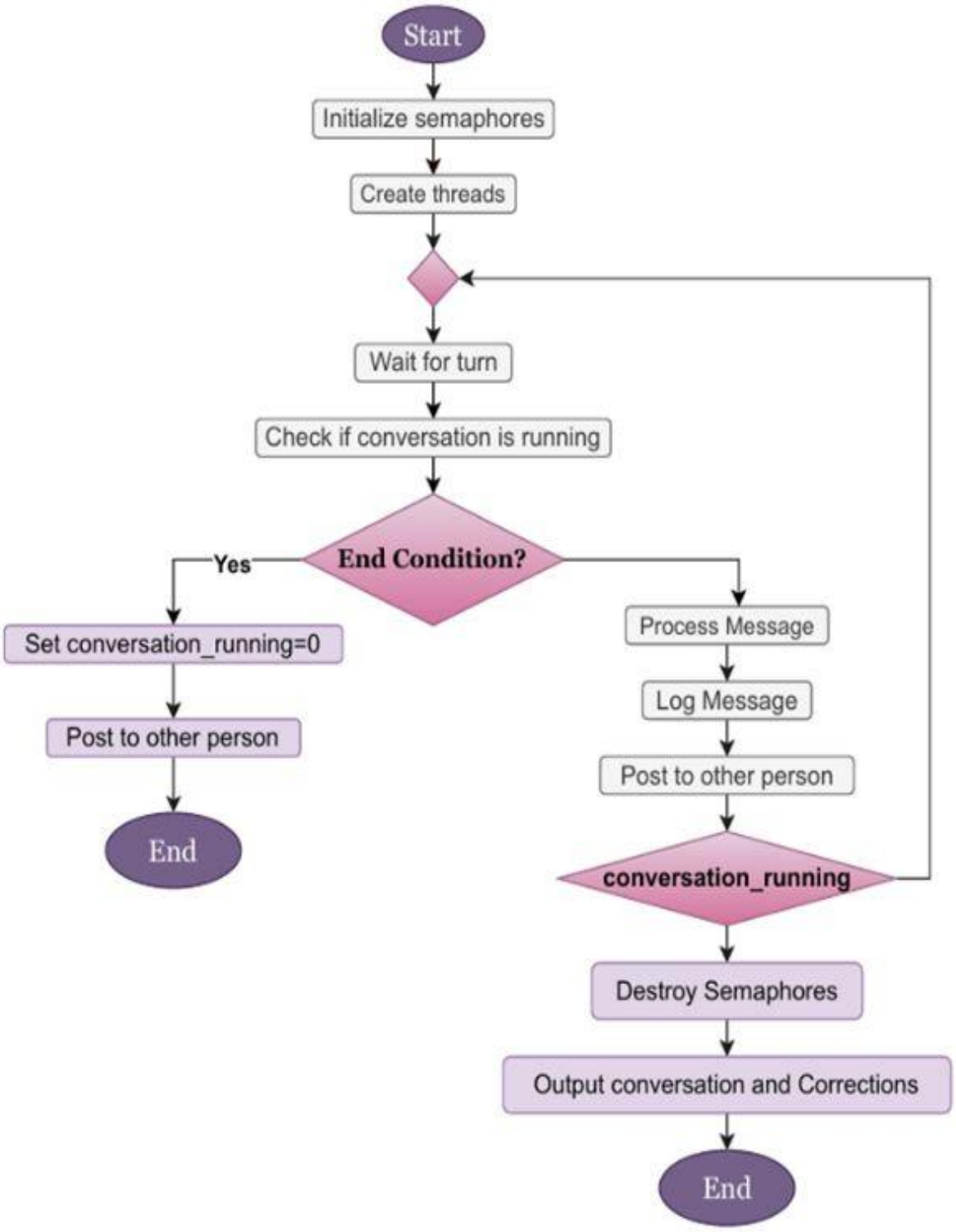
Person 1:



Person 2:


Start
Wait for person 2 turn
Get user input for person 2
input=="Bye."
Yes
No
Set conversation_running=0
Allow person 1 to exit
End
Message contains "iced tea"
Yes
No
Replace "iced tea" with "iced coffee"
Increment correction_count
Log message for person 2
Print message
Allow person 1 to take their turn
End

Main:


Start
Initialize semaphores
Create threads
Wait for turn
Check if conversation is running
End Condition?
Yes
Set conversation_running=0
Post to other person
End
Process Message
Log Message
Post to other person
conversation_running
Destroy Semaphores
Output conversation and Corrections
End

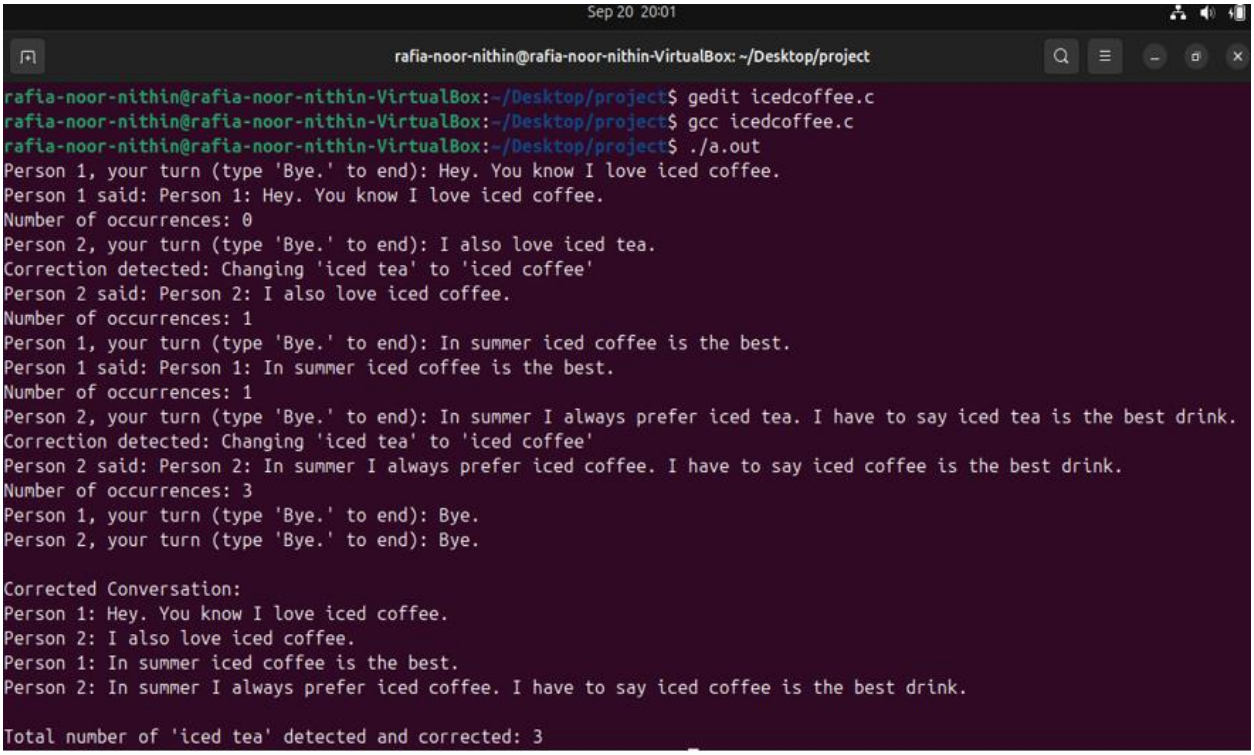
Code

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <string.h>
#define MAX_CONVERSATION_LENGTH 100
#define MAX_MESSAGE_LENGTH 512
#define PREFIX_LENGTH 10
sem_t person1_turn;
sem_t person2_turn;
char conversation[MAX_CONVERSATION_LENGTH][MAX_MESSAGE_LENGTH];
int conversation_index = 0;
int correction_count = 0;
int iced_tea_count = 0;
int conversation_running = 1;

void replace_iced_tea(char *message) {
    char buffer[MAX_MESSAGE_LENGTH];
    char *insert_point = &buffer[0];
    const char *temp = message;
    const char *match;
    while ((match = strstr(temp, "iced tea")) != NULL) {
        size_t len = match - temp;
        strncpy(insert_point, temp, len);
        insert_point += len;
        strcpy(insert_point, "iced coffee");
        insert_point += strlen("iced coffee");
        temp = match + strlen("iced tea");
        iced_tea_count++;
    }
    strcpy(insert_point, temp);
    strcpy(message, buffer);
}

void *person1(void *arg) {
```

```
    while (conversation_running) {
        sem_wait(&person1_turn);
        char message[MAX_MESSAGE_LENGTH];
        printf("Person 1, your turn (type 'Bye.' to end): ");
        fgets(message, MAX_MESSAGE_LENGTH, stdin);
        message[strcspn(message, "\n")] = 0;
        if (strcmp(message, "Bye.") == 0) {
            conversation_running = 0;
            sem_post(&person2_turn);
            break;
        }

 if (strstr(message, "iced tea") != NULL) {
            printf("Correction detected: Changing 'iced tea' to 'iced coffee'\n");
            replace_iced_tea(message);
            correction_count++;
        }
        char corrected_message[MAX_MESSAGE_LENGTH];
        snprintf(corrected_message, sizeof(corrected_message), "Person 1: %.501s",
message);
        strncpy(conversation[conversation_index++],
corrected_message,MAX_MESSAGE_LENGTH - 1);
        printf("Person 1 said: %s\n", corrected_message);
        printf("Number of occurrences: %d\n", iced_tea_count);
        sem_post(&person2_turn);
    }
    return NULL;
}


void *person2(void *arg) {
    while (conversation_running) {
        sem_wait(&person2_turn);
        char message[MAX_MESSAGE_LENGTH];
        printf("Person 2, your turn (type 'Bye.' to end): ");
        fgets(message, MAX_MESSAGE_LENGTH, stdin);
        message[strcspn(message, "\n")] = 0;
        if (strcmp(message, "Bye.") == 0) {
            conversation_running = 0;
            sem_post(&person1_turn);
```

```
            break;
        }
        if (strstr(message, "iced tea") != NULL) {
            printf("Correction detected: Changing 'iced tea' to 'iced coffee'\n");
            replace_iced_tea(message);
            correction_count++;
        }
        char corrected_message[MAX_MESSAGE_LENGTH];
        snprintf(corrected_message, sizeof(corrected_message), "Person 2: %.501s",
message);
        strncpy(conversation[conversation_index++], corrected_message,
MAX_MESSAGE_LENGTH - 1);
        printf("Person 2 said: %s\n", corrected_message);
        printf("Number of occurrences: %d\n", iced_tea_count);
        sem_post(&person1_turn);
    }
    return NULL;
}



int main() {
    pthread_t t1, t2;
    sem_init(&person1_turn, 0, 1);
    sem_init(&person2_turn, 0, 0);
    pthread_create(&t1, NULL, person1, NULL);
    pthread_create(&t2, NULL, person2, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    sem_destroy(&person1_turn);
    sem_destroy(&person2_turn);
    printf("\nCorrected Conversation:\n");
    for (int i = 0; i < conversation_index; i++) {
        printf("%s\n", conversation[i]);
    }
    printf("\n Total number of iced tea detected and corrected  'iced tea': %d\n",
iced_tea_count);
    return 0;
}
```

Output:



Code Explanation

This C code is a multi-threaded program that simulates a conversation between two people, Person 1 and Person 2. The two people take turns typing messages, and the program detects and corrects any instance of the phrase "iced tea" by replacing it with "iced coffee". The conversation continues until either person types "Bye." to end it.

1. `replace_iced_tea(char *message)`:

```
void replace_iced_tea(char *message) {
    char buffer[MAX_MESSAGE_LENGTH];
    char *insert_point = &buffer[0];
    const char *temp = message;
    const char *match;
    while ((match = strstr(temp, "iced tea")) != NULL) {
        size_t len = match - temp;
        strncpy(insert_point, temp, len);
        insert_point += len;
        strcpy(insert_point, "iced coffee");
```

```
        insert_point += strlen("iced coffee");
        temp = match + strlen("iced tea");
        iced_tea_count++;
    }
    strcpy(insert_point, temp);
    strcpy(message, buffer);
}
```

- **Purpose**: This function searches for the phrase "iced tea" in a given message and replaces it with "iced coffee". It also tracks the number of replacements.
- **Parameters**: It accepts a single parameter, `message`, which is the string (message) that Person 1 or Person 2 has typed.
- **Working**:
    1. A temporary buffer (`char buffer[MAX_MESSAGE_LENGTH]`) is created to store the modified message.
    2. The function searches for occurrences of "iced tea" using `strstr()`, a C standard library function that finds substrings.
    3. For each occurrence of "iced tea", it copies the text before the match into the buffer, then replaces "iced tea" with "iced coffee".
    4. After replacing all occurrences, the modified message is copied back into the original string, overwriting the original message.
    5. The `iced_tea_count` is incremented to track the number of replacements.
- **Example**: If Person 1 types "I want iced tea", this function will change the message to "I want iced coffee".

2. `void *person1(void *arg)`:

```
void *person1(void *arg) {
    while (conversation_running) {
        sem_wait(&person1_turn);
        char message[MAX_MESSAGE_LENGTH];
        printf("Person 1, your turn (type 'Bye.' to end): ");
        fgets(message, MAX_MESSAGE_LENGTH, stdin);
        message[strcspn(message, "\n")] = 0;
        if (strcmp(message, "Bye.") == 0) {
            conversation_running = 0;
            sem_post(&person2_turn);
            Break;
        }
```



```
 if (strstr(message, "iced tea") != NULL) {
```

```
            printf("Correction detected: Changing 'iced tea' to 'iced coffee'\n");
            replace_iced_tea(message);
            correction_count++;
        }
        char corrected_message[MAX_MESSAGE_LENGTH];
        snprintf(corrected_message, sizeof(corrected_message), "Person 1: %.501s",
message);
        strncpy(conversation[conversation_index++],
corrected_message,MAX_MESSAGE_LENGTH - 1);
        printf("Person 1 said: %s\n", corrected_message);
        printf("Number of occurrences: %d\n", iced_tea_count);
        sem_post(&person2_turn);
    }
    return NULL;
}
```

- **Purpose**: This function represents the behavior of Person 1 during the conversation. It runs in a loop, allowing Person 1 to repeatedly input messages until they type "Bye.".
- **Working**:
    1. **Turn Control**: The function starts by waiting for the semaphore `person1_turn` using `sem_wait()`, ensuring that Person 1 speaks when it's their turn.
    2. **User Input**: Person 1 enters a message through the terminal using `fgets()`. The newline character is removed using `strcspn()`.
    3. **Exit Condition**: If Person 1 types "Bye.", the conversation ends by setting the `conversation_running` flag to `0` and signaling Person 2 to exit.
    4. **Error Detection**: The function checks if the message contains "iced tea" using `strstr()`. If it does, it calls the `replace_iced_tea()` function to correct the message and increments the `correction_count`.
    5. **Message Logging**: The (corrected) message is logged in the conversation array with the prefix "Person 1: ".
    6. **Turn Transfer**: After Person 1 has spoken, `sem_post()` is used to signal Person 2's turn, allowing Person 2 to take over.
- **Example**: Person 1 types "I like iced tea", and the function corrects it to "I like iced coffee", logs the message, and then allows Person 2 to respond.

3. `void *person2(void *arg)`:


```
void *person2(void *arg) {
    while (conversation_running) {
```

```
        sem_wait(&person2_turn);
        char message[MAX_MESSAGE_LENGTH];
        printf("Person 2, your turn (type 'Bye.' to end): ");
        fgets(message, MAX_MESSAGE_LENGTH, stdin);
        message[strcspn(message, "\n")] = 0;
        if (strcmp(message, "Bye.") == 0) {
            conversation_running = 0;
            sem_post(&person1_turn);
            break;
        }
        if (strstr(message, "iced tea") != NULL) {
            printf("Correction detected: Changing 'iced tea' to 'iced coffee'\n");
            replace_iced_tea(message);
            correction_count++;
        }
        char corrected_message[MAX_MESSAGE_LENGTH];
        snprintf(corrected_message, sizeof(corrected_message), "Person 2: %.501s",
message);
        strncpy(conversation[conversation_index++], corrected_message,
MAX_MESSAGE_LENGTH - 1);
        printf("Person 2 said: %s\n", corrected_message);
        printf("Number of occurrences: %d\n", iced_tea_count);
        sem_post(&person1_turn);
    }
    return NULL;
}
```

- **Purpose**: This function is identical in structure to `person1()`, but it handles Person 2's input during the conversation.
- **Working**:
    1. **Turn Control**: Person 2 waits for their turn using `sem_wait(&person2_turn)`.
    2. **User Input**: Person 2 types a message, which is processed similarly to Person 1's message.
    3. **Exit Condition**: If Person 2 types "Bye.", the conversation ends.
    4. **Error Detection**: The function checks for "iced tea", calls `replace_iced_tea()` to make the correction if necessary, and logs the corrected message.
    5. **Turn Transfer**: After logging the message, it signals back to Person 1 using `sem_post()`.

- **Example**: If Person 2 types "Iced tea is my favorite", the function replaces it with "Iced coffee is my favorite", logs the corrected message, and passes control back to Person 1.

4. `int main()`:

```
int main() {
    pthread_t t1, t2;
    sem_init(&person1_turn, 0, 1);
    sem_init(&person2_turn, 0, 0);
    pthread_create(&t1, NULL, person1, NULL);
    pthread_create(&t2, NULL, person2, NULL);
    pthread_join(t1, NULL);
    pthread_join(t2, NULL);
    sem_destroy(&person1_turn);
    sem_destroy(&person2_turn);
    printf("\nCorrected Conversation:\n");
    for (int i = 0; i < conversation_index; i++) {
        printf("%s\n", conversation[i]);
    }
    printf("\n Total number of iced tea detected and corrected  'iced tea': %d\n",
iced_tea_count);
    return 0;
}
```

- **Purpose**: The `main()` function is responsible for setting up the conversation, initializing resources, creating threads, and cleaning up once the conversation ends.
- **Working**:
    1. **Semaphore Initialization**: `sem_init()` is used to initialize the semaphores. `person1_turn` starts with a value of 1 (Person 1 can speak first), and `person2_turn` starts with 0 (Person 2 waits).
    2. **Thread Creation**: Two threads are created using `pthread_create()`, one for each person. Each thread runs either `person1()` or `person2()`.
    3. **Thread Synchronization**: The `pthread_join()` calls ensure that the main program waits for both threads to finish before proceeding.
    4. **Semaphore Destruction**: Once the threads have finished, the semaphores are destroyed using `sem_destroy()`.

5. **Conversation Output**: After the conversation ends, the entire log is printed, along with the total number of corrections and the number of times "iced tea" was detected and replaced.

Output Explanation

1. **Person 1's Turn**:
    - **Input**: "Hey. You know I love iced coffee."
    - **Output**: "Person 1: Hey. You know I love iced coffee."
    - **Occurrences**: 0 (No "iced tea" in the message, so no corrections made.)
2. **Person 2's Turn**:
    - **Input**: "I also love iced tea."
    - **Correction**: Detected "iced tea" and replaced it with "iced coffee."
    - **Output**: "Person 2: I also love iced coffee."
    - **Occurrences**: 1 (One occurrence of "iced tea" was detected and corrected.)
3. **Person 1's Turn**:
    - **Input**: "In summer iced coffee is best."
    - **Output**: "Person 1: In summer iced coffee is best."
    - **Occurrences**: 1 (Still only one occurrence counted since no "iced tea" was mentioned.)
4. **Person 2's Turn**:
    - **Input**: "I always prefer iced tea in summer. I have to say iced tea is the best drink."
    - **Correction**: Detected two occurrences of "iced tea" and replaced both with "iced coffee."
    - **Output**: "Person 2: I always prefer iced coffee in summer. I have to say iced coffee is the best drink."
    - **Occurrences**: 3 (Total corrected instances increase to 3, counting the two in this turn along with the previous one.)
5. **Ending the Conversation**:
    - Both participants type "Bye." to indicate they want to end the conversation.

Summary of Outputs

- **Corrected Conversation**: Displays the final conversation with all corrections made, showing how each participant's message was adjusted to replace "iced tea" with "iced coffee."
- **Total Corrections**: The final tally indicates that the phrase "iced tea" was detected and corrected a total of 3 times throughout the conversation.

Explanation of Overall Functionality

- The program maintains a dynamic dialogue, allowing for real-time corrections.
- Each participant can express their thoughts while the system ensures that the preferred phrase ("iced coffee") is used consistently.
- The conversation reflects both the flow of dialogue and the correction mechanism working effectively.

Conclusion

This C program successfully simulates a conversation between two friends, Person 1 and Person2, using multi-threading and semaphores to control turn-taking and ensure smooth communication. The program corrects a recurring error where Person 2 mistakenly says "iced tea" instead of "iced coffee," making the conversation clearer and more consistent.Using threads for each participant, the program allows them to speak without interruptions, controlled by semaphores that manage the flow of conversation. Person 1 speaks first, followed by Person 2, with this alternating process continuing until one of them types "Bye." This simulates a structured, real-life dialogue.A key feature is the error correction mechanism, which automatically detects and replaces instances of "iced tea" with "iced coffee." The function `replace_iced_tea` uses substring detection and replacement to ensure that Person 2's errors are fixed in real-time. The program also tracks the number of corrections made, displaying this count at the end.The use of semaphores ensures thread synchronization, preventing message overlaps and maintaining data integrity in this multi-threaded environment. After the conversation ends, the program outputs the entire dialogue with corrected messages, demonstrating effective error handling.Overall, the program achieves its goals of simulating a natural conversation, ensuring clear communication through error correction, and demonstrating key concepts like semaphores, multi-threading, and string manipulation. It provides a practical example of how software can manage real-time communication and ensure clarity. With further extensions, such as more complex error detection or additional participants, the system could be expanded to handle more diverse scenarios.