**Course**

Operating Systems

Project Report

**Iced Tea Coffee Problem**

Using POSIX threads, mutex locks, and semaphores, I implemented a solution that synchronizes the correct conversation between Muntasir and Fokrul.

**Project Description**

This project is about a communication between two friends, Muntasir and Fokrul. They meet at a coffee shop. While discussing their iced coffee, Fokrul misspells it as “ice tea”, which made Muntasir confused . then he corrected the spelling and at a time their conversation came to an end. Our goal is to implement a synchronized conversation system using threads and semaphores to ensure turn talking, which also corrects misspelling automatically.

**Overview:**

The program fixes a problem between two friends who keep interrupted each other while talking about their favorite drink. One friend keeps saying “iced tea” but was meaning iced coffee.

1. Taking turns to speak: The program make sure that a friend can talk at a time. It works like synchronize, first muntasir then fokrul then again muntasir.
2. Fixing mistakes automatically: If any person says “ice tea” is corrects the problem and replace it with “iced coffee” before showing massage.
3. Each friends has their own thread. Semaphores controls who gets to talk when. the main program manages everything, methods are

Pthread\_create()

Pthread\_join()

Sem\_int()

Sem\_wait()

Sem\_post()

Sem\_destroy()

**Build-in function**

Thread management function:

1. Pthread\_create() :it creates new thread for muntasir and fokrul. And takes 4 argument like thread variable, attributes, functions to run, and arguments.
2. Pthread\_join(): This thread makes main program and wait for threads to finish. It also ensures proper program termination.

Semaphore functions:

1. Sem\_init(): this function initialize the semaphore, sem\_muntasir and sem\_fokrul. It sets initialize values , 1for muntasir and 0 for fokrul.
2. Sen\_wait():this function makes thread wait until semaphore is available. It also used for turn taking control.
3. Sem\_post(): It signals other thread that its turn has come. Like if muntasir turn is over , it gives signals to forkul thread. It releases the semaphore.
4. Sem\_destroy(): it mainly cleans up the semaphore when its done. Like when the task of muntasir semaphore is done, it destroy mustasir semaphore. That’s how it prevents resource leaks.

String function:

1. Strstr(): it finds ‘iced tea’ or ‘Iced tea’ in fokrul massages. The it returns pointer to found that text.
2. Memmove(): this function safely shifts character to make space for coffee. It mainly handles overlapping memory properly.
3. Memcpy(): it is a copy function, it copies ‘iced tea’ to ‘iced coffe’ into right allighment and replace the wrong tea text.

Other functions:

1. Printf(): it displays the functions on the screen and shows us the expected output.
2. Sleep(): depending on the user given time it delay the massages or output. In this code, adds 1second delay between massages to create a natural conversation.
3. Main(): it’s a controlling function that runs everything.

**Operating System:**

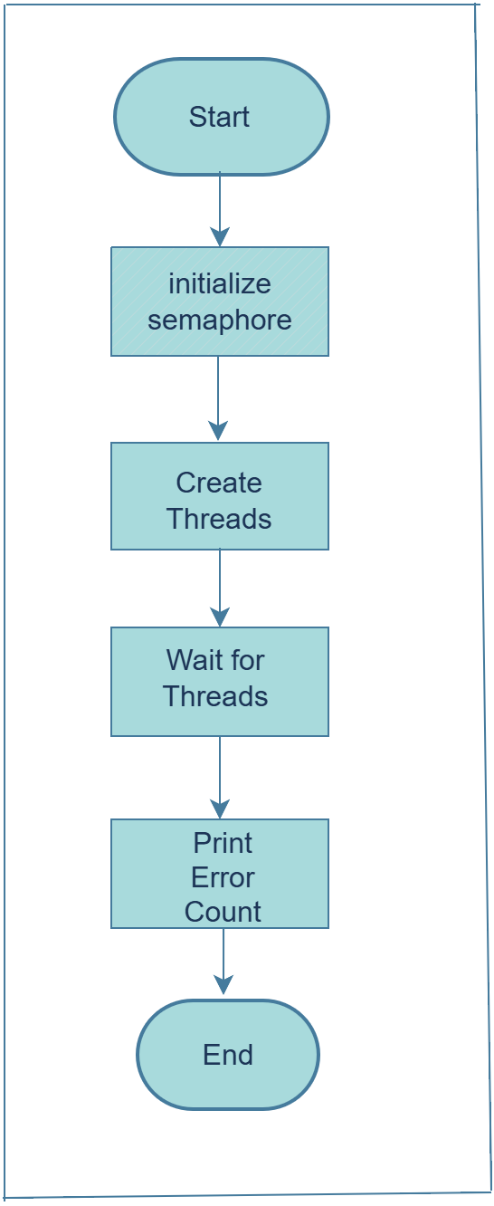
Linux

To run this iced\_coffee.c file, we used these command line arguments in terminal:

gcc iced\_coffee.c -o thread lpthread

./iced\_coffee

**Flowchart:**



Main program has 5 parts,

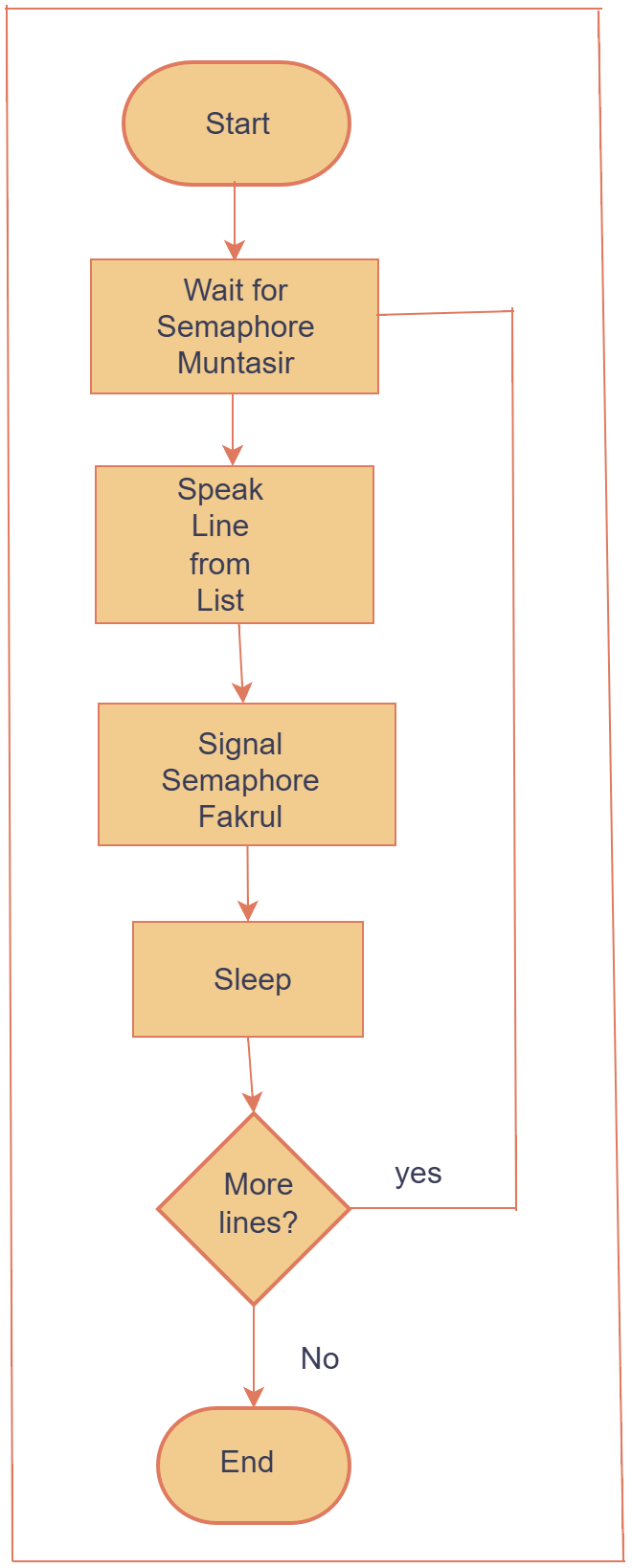
1 The program starts to execution and variable and memory are initialized.

2. Initialized semaphore to set up synchronization primitives

3. Create threads separately, one for Muntasir another for Fokrul, where Mustasir prints original massages and Fokrul corrects them.

4.Wait for threads.

5. Print error count and end, here shows total corrections



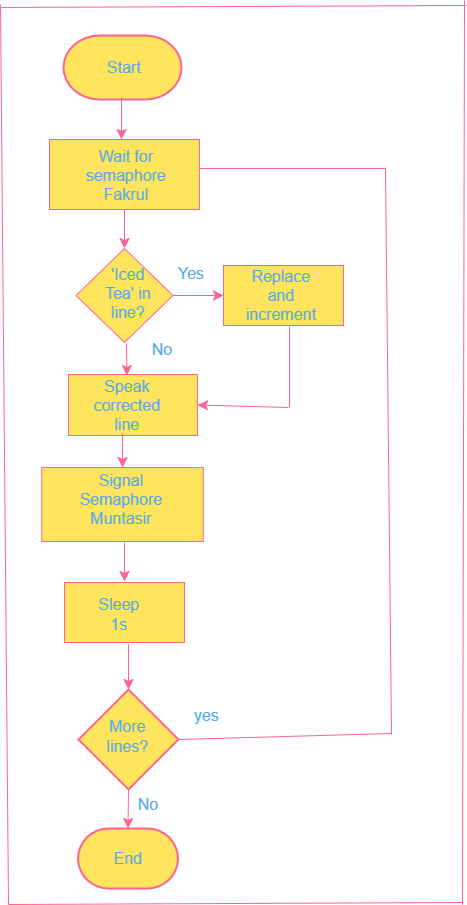
Muntasir thread has 4 parts,

1.Start and wait for turn, here thread begins execution and Muntasir thread waits and he is blocked until it is Muntasir turns.

2. When it is Muntasir turns,his massage shows on the screen. Example: I loved iced tea.

3. After Muntasir line, it takes 1s break (sleep) then Fokrul says his lines.

4. After each turn the thread checks if conversation left any, if it is yes then again start the conversation



Fokrul thread has five parts,

1. It starts by waiting for Muntasir turn to speak.
2. He checks if there is any mistake, if yes, then replace it with the correct one, and if no, then turns to Muntasir massage
3. After solving the mistake, Fokrul pauses and Muntasir speak his lines.
4. And wait for 1sec between their conversation, sleep mode
5. After speaking check if there is more line then go back to waiting. And if no, then end the conversation.

**Conclusion:**

We faced difficulties in coordinating threads and turn them smoothly with error. And after that, replacing iced tea with iced coffee, because their alignments are not same. The semaphore, when one thread is working another should be blocked, it was difficult task to avoid overlapping.

From this, we learn how to use semaphore, how can we maintain time balance in thread and turn taking. And also, if we want a flawless conversation to use a natural gap. How to simulate a real-life problem using programming logic.

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

#define MAX\_LINES 6

#define MAX\_LEN 100

sem\_t sem\_muntasir, sem\_fokrul;

int error\_count = 0;

const char \*muntasir\_lines[MAX\_LINES] = {

"Assalamualaikum fokrul.",

"I'm doing great, you?",

"Have you tried that new iced coffee?",

"It was really refreshing after a long day.",

"Nothing beats iced coffee in summer!",

"See you later, Allah hafez."

};

char fokrul\_lines[MAX\_LINES][MAX\_LEN] = {

"Walaikumassalam muntasir, How are you?",

"I'm fine, thanks!",

"Yes! I love iced tea.",

"Wait, I meant iced tea. Oops.",

"Iced tea is my favorite!",

"Allah hafez."

};

void replace\_word(char \*line) {

char \*pos;

while ((pos = strstr(line, "iced tea")) != NULL || (pos = strstr(line, "Iced tea")) != NULL)

{

memmove(pos + 11, pos + 8, strlen(pos + 8) + 1);

if (strncmp(pos, "Iced tea", 8) == 0)

memcpy(pos, "Iced coffee", 11);

else

memcpy(pos, "iced coffee", 11);

error\_count++;

}

}

void \*muntasir\_thread(void \*arg) {

for (int i = 0; i < MAX\_LINES; i++) {

sem\_wait(&sem\_muntasir);

printf("Muntasir: %s\n", muntasir\_lines[i]);

sleep(1);

sem\_post(&sem\_fokrul);

}

return NULL;

}

void \*fokrul\_thread(void \*arg) {

for (int i = 0; i < MAX\_LINES; i++) {

sem\_wait(&sem\_fokrul);

replace\_word(fokrul\_lines[i]);

printf("Fokrul: %s\n", fokrul\_lines[i]);

sleep(1);

sem\_post(&sem\_muntasir);

}

return NULL;

}

int main() {

pthread\_t t1, t2;

sem\_init(&sem\_muntasir, 0, 1);

sem\_init(&sem\_fokrul, 0, 0);

pthread\_create(&t1, NULL, muntasir\_thread, NULL);

pthread\_create(&t2, NULL, fokrul\_thread, NULL);

pthread\_join(t1, NULL);

pthread\_join(t2, NULL);

sem\_destroy(&sem\_muntasir);

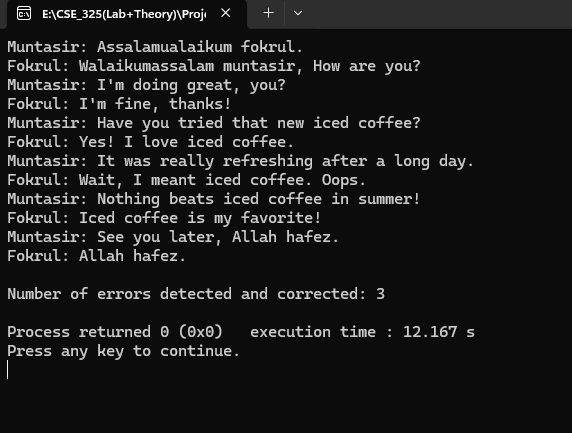
sem\_destroy(&sem\_fokrul);

printf("\nNumber of errors detected and corrected: %d\n", error\_count);

return 0;

}

**Expected Output:**

****