**Bugs and its solutions**

1. In the main function, after creating the producer and consumer threads, there is a missing call to destroy the semaphore and mutex resources using sem\_destroy and pthread\_mutex\_destroy. It is good practice to release these resources after they are no longer needed. To fix this, add the following lines at the end of the main function:

cCopy code

sem\_destroy(&full); sem\_destroy(&empty); pthread\_mutex\_destroy(&lock);

1. he remove\_data function doesn't handle the case when the buffer is empty. In such cases, it should return a default value or indicate that the buffer is empty. To handle this situation, you can modify the remove\_data function to return a special value (e.g., -1) when the buffer is empty. Update the remove\_data function as follows

int remove\_data() {

if (head == NULL) {

printf("Buffer is empty.\n");

return -1; // Return a special value to indicate buffer is empty

}

Node \*temp = head;

int data = temp->data;

head = head->next;

count--;

free(temp);

return data;

}

1. The add\_data function is missing the handling for the case when the buffer is full. You need to check if the buffer is already full before adding a new node. Modify the add\_data function as follows:

void add\_data(int data) {

Node \*new\_node = (Node\*)malloc(sizeof(Node));

new\_node->data = data;

new\_node->next = NULL;

if (count < BUFFER\_SIZE) {

if (tail == NULL) {

head = tail = new\_node;

} else {

tail->next = new\_node;

tail = new\_node;

}

count++;

} else {

printf("Buffer is full. Cannot add data.\n");

}

}

1. Missing pthread\_exit statements in the producer and consumer functions: To properly exit the threads, you need to include pthread\_exit(NULL) statements at the end of both the producer and consumer functions.

Now FULL code :-

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define BUFFER\_SIZE 10

typedef struct node {

int data;

struct node \*next;

} Node;

Node \*head = NULL, \*tail = NULL;

int count = 0;

sem\_t full, empty;

pthread\_mutex\_t lock;

void add\_data(int data) {

Node \*new\_node = (Node\*)malloc(sizeof(Node));

new\_node->data = data;

new\_node->next = NULL;

if(tail == NULL) {

head = tail = new\_node;

} else {

tail->next = new\_node;

tail = new\_node;

}

count++;

}

int remove\_data() {

// bug fix

if (head == NULL) {

printf("Buffer is empty.\n");

return -1; // Return a special value to indicate buffer is empty

}

Node \*temp = head;

int data = temp->data;

head = head->next;

count--;

return data;

}

void \*producer(void \*arg) {

int i, data;

for (i = 0; i < 100; i++) {

data = rand() % 100;

sem\_wait(&empty);

pthread\_mutex\_lock(&lock);

add\_data(data);

printf("Produced: %d\n", data);

sem\_post(&full); // Move the sem\_post call before releasing the lock

pthread\_mutex\_unlock(&lock);

}

// bug fix

pthread\_exit(NULL);

}

void \*consumer(void \*arg) {

int i, data;

for(i=0; i<100; i++) {

sem\_wait(&full);

pthread\_mutex\_lock(&lock);

data = remove\_data();

printf("Consumed: %d\n", data);

pthread\_mutex\_unlock(&lock);

sem\_post(&empty);

}

// bug fix

pthread\_exit(NULL);

}

int main() {

pthread\_t producer\_thread, consumer\_thread;

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

sem\_init(&empty, 0, BUFFER\_SIZE);

pthread\_mutex\_init(&lock, NULL);

pthread\_create(&producer\_thread, NULL, producer, NULL);

pthread\_create(&consumer\_thread, NULL, consumer, NULL);

pthread\_join(producer\_thread, NULL);

pthread\_join(consumer\_thread, NULL);

// bug fix

cCopy code

sem\_destroy(&full);

sem\_destroy(&empty);

pthread\_mutex\_destroy(&lock);

return 0;

}