|  |  |
| --- | --- |
|  | **DEPARTMENT OF COMPUTER ENGINEERING** |

**PBLE 2**

|  |  |
| --- | --- |
| Semester | S.E. Semester IV – Computer Engineering |
| Subject | Operating System |
| Subject Professor In-charge | Prof. Pankaj Vanvari |
| Assisting Teachers | Prof. Pankaj Vanvari |

|  |  |
| --- | --- |
| Roll Numbers | Name of Students |
| 21102A0014 | Deep Salunkhe |
| 21102A0003 | Omkar Patil |
| 21102A0005 | Pranav Redij |

This code implements a reader-writer problem using threads and semaphores in C++. It uses two semaphores, mutex and wrt, to control access to a shared resource (shared\_data). The writer thread waits for the wrt semaphore before writing to the shared resource, while reader threads wait for the mutex semaphore before reading from the shared resource. The program is made menu-driven, where the user can choose to create a new writer or reader thread, or exit the program. Additionally, the user can choose to exit a writer thread or check if any reader threads are still inside the shared area before exiting**.**

**Code:**

#include <iostream>

#include <semaphore.h>

#include <pthread.h>

using namespace std;

sem\_t mutex, wrt;

int shared\_data = 0;

int reader\_count = 0;

void \*writer(void \*arg)

{

    sem\_wait(&wrt);

    shared\_data++;

    cout << "Data written by the writer is: " << shared\_data << endl;

    sem\_post(&wrt);

}

void \*reader(void \*arg)

{

    sem\_wait(&mutex);

    reader\_count++;

    if(reader\_count == 1)

        sem\_wait(&wrt);

    sem\_post(&mutex);

    cout << "Data read by the reader is: " << shared\_data << endl;

    sem\_wait(&mutex);

    reader\_count--;

    if(reader\_count == 0)

        sem\_post(&wrt);

    sem\_post(&mutex);

}

int main()

{

    sem\_init(&mutex, 0, 1);

    sem\_init(&wrt, 0, 1);

    pthread\_t t1, t2, t3, t4;

    string type;

    while (true) {

        cout << "What would you like to do? (writer, reader, exit\_reader, exit\_writer, exit\_program)" << endl;

        cin >> type;

        if (type == "writer") {

            pthread\_create(&t1, NULL, writer, NULL);

        } else if (type == "reader") {

            pthread\_create(&t2, NULL, reader, NULL);

        } else if (type == "exit\_reader") {

            if (reader\_count > 0) {

                cout << "Reader(s) are still inside. Cannot exit." << endl;

            } else {

                cout << "No reader inside. Cannot exit." << endl;

            }

        } else if (type == "exit\_writer") {

            sem\_post(&wrt);

            cout << "Writer exited the shared area." << endl;

        } else if (type == "exit\_program") {

            break;

        } else {

            cout << "Invalid input. Please try again." << endl;

        }

    }

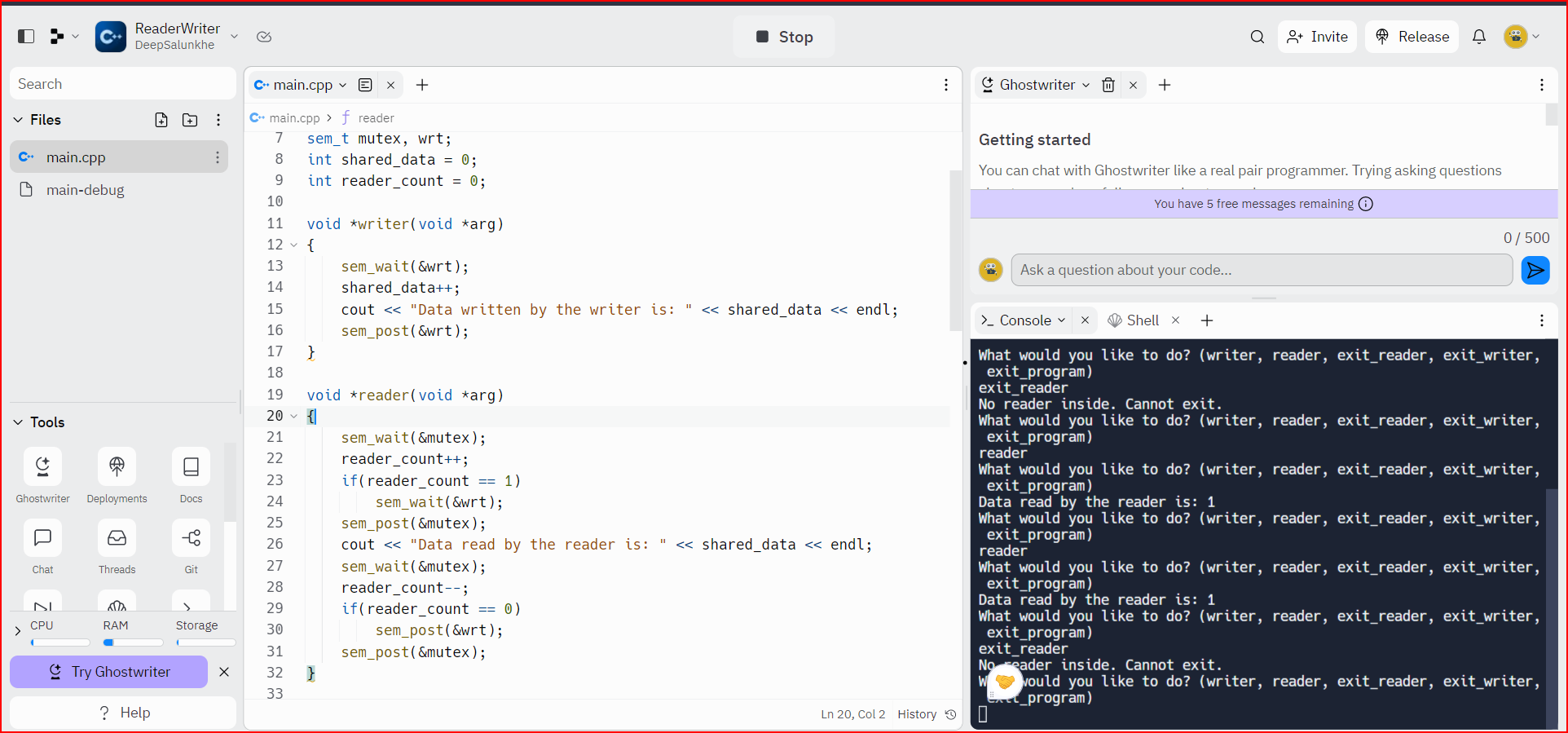
    sem\_destroy(&mutex);

    sem\_destroy(&wrt);

    return 0;

}

**Result/ Output:**

****