1.Producer_Consumer Problem

```
#include <iostream>
#include <thread>
#include <mutex>
#include <semaphore.h>
#include <vector>
#include <unistd.h>
using namespace std;

int item = 0;
const int Buff = 5;
vector<int> buff;
mutex mtx;

sem_t emt;
sem_t full;

void producer(int id) {
    while (true) {
        sleep(1);
        sem_wait(&emt);

        int producedItem;
        {
            lock_guard<mutex> lock(mtx);
            item++;
            producedItem = item;
            buff.push_back(producedItem);
```

```
            cout << "Producer " << id << " produced: "
<< producedItem << endl;
        }

        sem_post(&full);
    }
}

void consumer(int id) {
    while (true) {
        sem_wait(&full);

        int consumedItem;
        {
            lock_guard<mutex> lock(mtx);
            consumedItem = buff.back();
            buff.pop_back();
            cout << "Consumer " << id << " consumed: "
<< consumedItem << endl;
        }

        sem_post(&emt);
        sleep(1);
    }
}

int main() {
    sem_init(&emt, 0, Buff); // Buff empty slots
initially
    sem_init(&full, 0, 0);   // 0 full slots initially
```

```
    thread p1(producer, 1);
    thread p2(producer, 2);
    thread c1(consumer, 1);
    thread c2(consumer, 2);

    p1.join();
    p2.join();
    c1.join();
    c2.join();

    sem_destroy(&emt);
    sem_destroy(&full);

    return 0;
}
```

2.Reader and Writer Problem

```
#include <iostream>
#include <thread>
#include <mutex>
#include <chrono>
#include <unistd.h>
using namespace std;

int sd;
```

```
int rc;
mutex mtx ,rmtx;

void reader(int id){
    for(int i=0;i<2;i++){
        mtx.lock();

        rc++;
        if(rc==1){
            rmtx.lock();
        }

        mtx.unlock();

         cout << "Reader " << id << " reads value: "
<< sd << endl;
         sleep(1);


        mtx.lock();

        rc--;
        if(rc==0){
            rmtx.unlock();
        }

        mtx.unlock();

        sleep(1);
```

```
    }

}

void writer(int id){
    for(int i=0;i<2;i++){
        rmtx.lock();

        sd++;
        cout << "Writer " << id << " updates value to:
" << sd << endl;

        rmtx.unlock();

        sleep(1);
    }
}


int main(){

    thread w1(writer,1);
    thread r1(reader,1);
    thread r2(reader,2);
    thread w2(writer,2);
    thread r3(reader,3);

    r1.join();
    r2.join();
```

```
    w1.join();
    w2.join();
    r3.join();
}
```

3.Race condition

```
#include <iostream>
#include <thread>

using namespace std;

int counter = 0; // shared variable

void increment(int n) {
    for (int i = 0; i < n; i++) {
        counter++; // race condition here
    }
}

void decrement(int n) {
    for (int i = 0; i < n; i++) {
        counter--; // race condition here
    }
}

int main() {
    int n = 1000000;
```

```
    thread t1(increment, n);
    thread t2(decrement, n);

    t1.join();
    t2.join();

    cout << "Final counter without synchronization: "
<< counter << endl;
    return 0;
}
```

4.Mutex Lock

```
#include <iostream>
#include <thread>
#include <mutex>
#include<unistd.h>

using namespace std;

int counter = 0;
mutex mtx;

void increment(int n) {
    for (int i = 0; i < n; i++) {
        mtx.lock();
        counter++;
```

```
        cout<<"Incre"<<i<<"counter: "<<counter<<endl;
        mtx.unlock();
    }
}

void decrement(int n) {
    for (int i = 0; i < n; i++) {
        mtx.lock();
        counter--;
        cout<<"decre"<<i<<"counter: "<<counter<<endl;
        mtx.unlock();
    }
}

int main() {
    int n = 10000;

    thread t1(increment, n);
    thread t2(decrement, n);

    t1.join();
    t2.join();

    cout << "Final counter with mutex: " << counter <<
endl;
    return 0;
}
```

5.Semaphore

```
#include <iostream>
#include <thread>
#include <semaphore.h>

using namespace std;

int counter = 0;
sem_t sem;

void increment(int n) {
    for (int i = 0; i < n; i++) {
        sem_wait(&sem); // lock
        counter++;
        sem_post(&sem); // unlock
    }
}

void decrement(int n) {
    for (int i = 0; i < n; i++) {
        sem_wait(&sem); // lock
        counter--;
        sem_post(&sem); // unlock
    }
}

int main() {
    int n = 1000000;
```

```
    sem_init(&sem, 0, 1); // binary semaphore

    thread t1(increment, n);
    thread t2(decrement, n);

    t1.join();
    t2.join();

    cout << "Final counter with semaphore: " <<
counter << endl;

    sem_destroy(&sem);
    return 0;
}
```