

PROJECT REPORT

Topic: Process Synchronization

1) PROJECT DESCRIPTION

- Abstract:

Anytime we share data between two or more processes or threads, we run the risk of having a race condition where our data could become corrupted. In order to avoid these situations, we have various mechanisms to ensure that one program's critical regions are guarded from another's.

One place that we might use parallelism is to simulate real-world situations that involve multiple independently acting entities, such as people. In this project, we will use the semaphore implementation to model the safe apartment inspection problem,

whereby (potential) tenants and real-estate agents synchronize so that no deadlock occurs and only 9 tenants can see apartment at a time.

- Key learning points:

Mutual exclusion

Deadlock

Starvation

Race conditions

Semaphores

Critical Section

Producer Consumer Problem

2) SRS

- Purpose: The job of this project is to write a program that
 - (a) always satisfies the above constraints and
 - (b) where under no conditions will a deadlock occur.

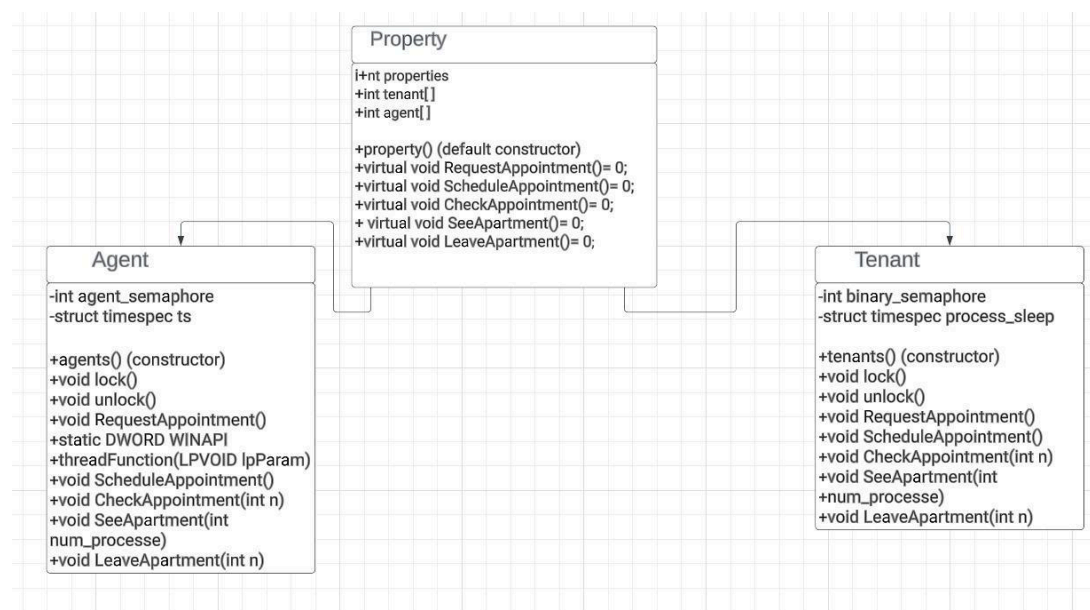
(A deadlock happens, for example, when the apartment is empty, an agent and a tenant arrive, but cannot enter.
- Intended Audience: Property dealers, tenants and agents who want to deal with various property issues.
- Intended Use: A person can schedule and see apartment in a synchronized way.

- User Needs: To interact with the single agent for various properties without dealing with their owners.
- Expected Outcome: All the process will be synchronized and no starvation or deadlock will be achieved
- Functional Requirements: To make our shared data and our semaphores, what we need is for multiple processes to be able to share the same memory region.
- Non-functional Requirements: WINDOW 11, VS CODE, LEARN MICROSOFT.

3) HLD/LLD

Low-level description of Process Synchronization:

In the case of Process Synchronization project in C++, the LLD would include defining the classes, functions, and algorithms that would be required to implement the system. Here is an LLD for this project file in C++:



The above LLD provides a basic outline of the design for our project in C++.

- Classes:

1. Property

2. Tenant

3. Agents

- Functions:

1. Requestappointment()

2. ScheduleAppointment()

3. CheckAppointment()

4. SeeApartment()

5. LeaveApartment()

- Libraries:

1. iostream
2. window.h
- 3.csthread
- 4.vector
- 5.algorithm
- 6.chrono
- 7.time.h
- 8.stdexcept

4) Initial code:

```
#include<iostream>
using namespace std;

class property
{
    public:
        int properties;
        int tenant[10];
        int agent[3];
        //lets take we will be having four properties where different dealers would be
        //dealing to find new tenants/
        //we will be making several functions for the class property they are as follows/
        property()
        {
            properties=4;
        }
};
//this is a tenant class where the tenants will ask for appointment and have a look at the
//apartment/
class tenants:public property
{
    public:
        //the functions for the tenant class are as follows/
        void RequestAppointment()
        {
            int choice;
            cout<<" Which property you to visit ?\nAvailable properties: 1,2,3,4\n";
            cout<<"Enter choice:";
            cin>>choice;
            if(choice== 1|| choice==2 || choice==3 || choice==4)
            {
                cout<<"Kindly schedule appointment"<<endl;
            }
        }
        void ScheduleAppointment()
        {
            int time;
            cout<<"choose the time slot you wish to come and visit the apart!"
            <<"\n1.(11:00am-13:00pm)"
```

```

        <<"\n2.(13:00pm-14:00pm)"
        <<"\n3.(15:00pm-16:00pm)";
        cin>>time;
    }
    void CheckAppointment(int n)
    {
        if(n<=9)
        {
            cout<<"Your appointment is confirmed "<<endl;
        }
        else
        {
            cout<<"Your appointment is on waiting"<<endl;
        }
    }
    void SeeApartment(int n)
    {
        for (int id = 1; id <= n; id++)
        {
            cout << "\n";
            cout << endl;
            cout << "Tenant " << id << " Arrives to inspect the apart\n ";
        }
    }
    void LeaveApartment(int n)
    {
        for (int id = 1; id <= n; id++)
        {
            cout << "\n";
            cout << endl;
            cout << "Tenant " << id << " leaves the apart\n ";
        }
    }
};

class agents:public property
{
public:
    void RequestAppointment()
    {
        int choice;
        cout<<" Which property do you want to visit ?\nAvailable properties:
1,2,3,4\n";
        cout<<"Enter choice:";
        cin>>choice;
        if(choice==1 || choice==2 || choice==3 || choice==4)
        {
            cout<<"Kindly schedule appointment"<<endl;
        }
    }
    void ScheduleAppointment()
    {
        int time;
        cout<<"choose the time slot you wish to come and visit the apart!"

```

```

        <<"\n1.(11:00am-13:00pm)"
        <<"\n2.(13:00pm-14:00pm)"
        <<"\n3.(15:00pm-16:00pm)";
        cin>>time;
    }
    void CheckAppointment(int n)
    {
        if(n>0 || n<0)
        {
            cout<<"Your appointment is on waiting"<<endl;
        }
        else
        {
            cout<<"You can visit the apartment"<<endl;
        }
    }

    void SeeApartment()
    {
        if(*tenant>0 && *tenant<=9)
        {
            cout<<"Agent arrived at apartment "<<endl;
        }
    }
    void LeaveApartment()
    {
        if(tenant==0)
        {
            cout<<"You can leave apartment"<<endl;
        }
        else
        {
            cout<<"Wait for tenant to leave "<<endl;
        }
    }
};

property *factory(int y)
{
    switch(y % 2) {

        case 0: return new tenants;
        case 1: return new agents;
        }
    return 0;
}

int main()
{
    property *property1;
    tenants *tenant1;
    agents *agent1;
    property1=factory(0);
    int tenant=0;
    int agent=0;
    int choice;
    while (2)

```

```

{

    cout<<"1. Tenant \n2. Agent"<<endl;
    cout<<"Enter you choice"<<endl;
    cin>>choice;
    if(choice==1)
    {
        tenant++;
        while(2)
        {

            property1->tenant;
            cout<<"1.Request appointment\n2.Schedule Appointment\n3.Check
Appointment\n4.See Apartment\n5.Leave Apartment"<<endl;
            cout<<"Enter your choice"<<endl;
            cin>>choice;
            switch (choice)
            {
                case 1:
                    tenant1->RequestAppointment();
                    break;
                case 2:
                    tenant1->ScheduleAppointment();
                    break;
                case 3:
                    tenant1->CheckAppointment(tenant);
                    break;
                case 4:
                    tenant1->SeeApartment(tenant);
                    break;
                case 5:
                    tenant1->LeaveApartment(tenant);
                    break;
                default:
                    cout<<"invalid choice"<<endl;
                    break;
            }
            cout<<"You want to continue as tenant?If yes press 0"<<endl;
            cin>>choice;
            if(choice!=0)
                break;
        }
    }
    if(choice==2)
    {
        agent++;
        while(2)
        {
            property1->agent;
            cout<<"1.Request appointment\n2.Schedule Appointment\n3.Check
Appointment\n4.See Apartment\n5.Leave Apartment"<<endl;
            cout<<"Enter your choice"<<endl;
            cin>>choice;
            switch (choice)
            {
                case 1:
                    agent1->RequestAppointment();
                    break;

```

```

        case 2:
            agent1->ScheduleAppointment();
            break;
        case 3:
            agent1->CheckAppointment(agent);
            break;
        case 4:
            agent1->SeeApartment();
            break;
        case 5:
            agent1->LeaveApartment();
            break;
        default:
            cout<<"invalid choice"<<endl;
            break;
    }
    cout<<"You want to continue as agent ?If yes press 0"<<endl;
    cin>>choice;
    if(choice!=0)
        break;
}
else
{
    cout<<"invalid choice"<<endl;
}
cout<<"You want to continue ?If yes press 0"<<endl;
cin>>choice;
if(choice!=0)
    break;
}

return 0;
}

```

Final Code:

Here is the final code implementation after learning all the necessary topics and functions:

```

#include <iostream>

#include <vector>

#include <algorithm>

#include <chrono>

#include <ctime>

#include <thread>

#include <windows.h>

using namespace std;

class property

```

```

{
public:

    int properties;

    int tenant[10];

    int agent[3];
    // lets take we will be having four properties where different dealers would be
    // dealing to find new tenants/
    // we will be making several functions for the class property they are as follows/
    property()
    {
        properties = 4;
    }

};

class agents : public property
{
private:
    int agent_semaphore;
    struct timespec ts;

public:
    agents()
    {
        agent_semaphore = 0;
    }
    // here including the mutex lock functions//
    void lock()
    {
        agent_semaphore++;
    }

    void unlock()
    {
        agent_semaphore--;
    }

    void RequestAppointment()
    {
        int choice;

        cout << " Which property you to visit ?\nAvailable properties: 1,2,3,4\n";

        cout << "Enter choice:";

        cin >> choice;

        if (choice == 1 || choice == 2 || choice == 3 || choice == 4)
        {
            cout << "Kindly schedule appointment" << endl;
        }
    }

    static DWORD WINAPI threadFunction(LPVOID lpParam)
    {

```



```

        cout << "\n\nAgent is providing information to Tenant in other room." << endl;
        return 0;
    }

void ScheduleAppointment()
{
    int time;
    cout << "choose the time slot you wish to come and visit the apart!"

        << "\n1.(11:00am-13:00pm)"

        << "\n2.(13:00pm-14:00pm)"

        << "\n3.(15:00pm-16:00pm)";
    cin >> time;
}

void CheckAppointment(int n)
{
    if (n <= 9)
    {
        cout << "Your appointment is confirmed " << endl;
    }
    else
    {
        cout << "Your appointment is on waiting" << endl;
    }
}

void SeeApartment(int num_processe)
{
    /*here we will be using the binary semaphore where when one of the tenants would
be viewing the apart other tenant cannot intrude or come to
view the apart */

    for (int i = 1; i <= num_processe; i++)
    {
        lock();

        if (agent_semaphore > 1)
        {
            MessageBox(NULL, "Mutual exclusion wont be followed!\nYou wont be able to
access the flat", "error 502", MB_OK);

            MessageBox(NULL, "Sorry for the Inconvenience :", "error 503", MB_OK);

            break;
        }
        else if (agent_semaphore == 1)
        {
            HANDLE hThread = CreateThread(NULL, 0, threadFunction, NULL, 0, NULL);

            if (hThread == NULL)
            {
                cout << "Thread creation failed" << endl;
            }

            CONTEXT context;
            context.ContextFlags = CONTEXT_FULL;

```

```

        if (GetThreadContext(hThread, &context))
        {
            cout << "\nAgent " << i << " is viewing the flat along with the
tenant";
            cout << "\n No other agent or tenant is allowed to enter the flat!";

            CloseHandle(hThread);

            unlock();
            cout << "\n The flat has been viewed and now,Agent " << i << " is
leaving the flat !";
        }
        else if (agent_semaphore == 0)
        {
            cout << "\nNo process is in the critical section!";
        }
        ts.tv_sec = 3;
        ts.tv_nsec = 0;
        int result = nanosleep(&ts, nullptr);

        if (result == -1)
        {
            MessageBox(NULL, "Error calling Nanosleep....!", "Error 505", MB_OK);
        }
    }
}
}
void LeaveApartment(int n)
{
    for (int id = 1; id <= n; id++)
    {
        cout << "\n";
        cout << endl;
        cout << "Agent " << id << " leaves the apart\n ";
    }
}
};
class tenants : public property
{
private:
    int binary_semaphore;
    struct timespec process_sleep;

public:
    tenants()
    {
        binary_semaphore = 0;
    }
    // here including the mutex lock functions/
    void lock()
    {
        binary_semaphore++;
    }

    void unlock()
    {

```

```

    binary_semaphore--;
}

void ScheduleAppointment()
{
    int time;
    cout << "choose the time slot you wish to come and visit the apart!"
        << "\n1.(11:00am-13:00pm)"
        << "\n2.(13:00pm-14:00pm)"
        << "\n3.(15:00pm-16:00pm)";
    cin >> time;
}

void RequestAppointment()
{
    int choice;
    cout << " Which property you to visit ?\nAvailable properties: 1,2,3,4\n";
    cout << "Enter choice:";
    cin >> choice;
    if (choice == 1 || choice == 2 || choice == 3 || choice == 4)
    {
        cout << "Kindly schedule appointment" << endl;
    }
}

void CheckAppointment(int n)
{
    if (n > 0 || n < 0)
    {
        cout << "Your appointment is on waiting" << endl;
    }
    else
    {
        cout << "You can visit the apartment" << endl;
    }
}

static DWORD WINAPI threadFunction(LPVOID lpParam)
{
    cout << "\n\nAgent is providing information to Tenant in other room." << endl;
    return 0;
}

void SeeApartment(int num_processes)
{
    if (*tenant > 0 && *tenant <= 9)
    {
        cout << "Agent arrived at apartment " << endl;
    }
    /*here we will be using the binary semaphore where when one of the tenants would
be viewing the apart other tenant cannot intrude or come to
view the apart */

    for (int i = 0; i <= num_processes; i++)
    {
        unlock();

        if (binary_semaphore > 1)
        {

```

```

        MessageBox(NULL, "Mutual exclusion wont be followed!\nYou wont be able to
access the flat", "error 502", MB_OK);

        MessageBox(NULL, "Sorry for the Inconvenience :", "error 504", MB_OK);

        break;
    }
    else if (binary_semaphore == 1)
    {
        HANDLE hThread = CreateThread(NULL, 0, threadFunction, NULL, 0, NULL);
        if (hThread == NULL)
        {
            cout << "Thread creation failed" << endl;
        }

        CONTEXT context;
        context.ContextFlags = CONTEXT_FULL;
        if (GetThreadContext(hThread, &context))
        {

            cout << "\n\nTenant " << i << " is viewing the flat";
            cout << "\n No other tenant or agent is allowed to enter the flat!";

            CloseHandle(hThread);

            lock();
            cout << "\n The flat has been viewed and now,Tenant " << i << " is
leaving the flat !";
        }
        else if (binary_semaphore == 0)
        {
            cout << "\nNo process is in the critical section!";
        }
        process_sleep.tv_sec = 3;
        process_sleep.tv_nsec = 0;
        int result = nanosleep(&process_sleep, nullptr);

        if (result == -1)
        {
            MessageBox(NULL, "Error calling Nanosleep....!", "Error 505", MB_OK);
        }
    }
}

void LeaveApartment(int tenant)
{
    if (tenant == 0)
    {
        cout << "You can leave apartment" << endl;
    }
    else
    {
        cout << "Wait for tenant to leave " << endl;
    }
}

};
int main()

```



```

        tenant1->RequestAppointment();

        break;

    case 2:

        tenant1->ScheduleAppointment();

        break;

    case 3:

        tenant1->CheckAppointment(tenant);

        MessageBox(NULL, "Appointment Scheduled.....!!!!", "Success", MB_OK);

        break;

    case 4:

        int num_processes;

        cout << "Enter the number of Tenants that would arrive to view the
flat:";

        cin >> num_processes;

        tenant1->SeeApartment(num_processes);

        break;

    case 5:

        tenant1->LeaveApartment(tenant);

        MessageBox(NULL, "Have a nice day", "notice", MB_OK);

        break;

    default:

        cout << "invalid choice" << endl;

        break;

    }

    cout << "You want to continue as tenant?If yes press 0 and press 1 to exit
the Menu" << endl;

    cin >> choice;

    if (choice != 0)

        break;

```

```

    }

}

if (choice == 2)
{
    agent++;

    while (2)
    {
        property1->agent;

        cout << "1.Request appointment\n2.Schedule Appointment\n3.Check
Appointment\n4.See Apartment\n5.Leave Apartment" << endl;

        cout << "Enter your choice" << endl;

        cin >> choice;

        switch (choice)
        {
            case 1:

                agent1->RequestAppointment();

                break;

            case 2:

                agent1->ScheduleAppointment();

                break;

            case 3:

                agent1->CheckAppointment(agent);

                MessageBox(NULL, "Appointment Scheduled.....!!!!", "Success", MB_OK);

                break;

            case 4:

                agent1->SeeApartment(agent);

                break;

            case 5:

```

```

        agent1->LeaveApartment(agent);

        MessageBox(NULL, "Have a Nice Day....! :)", "Notice!", MB_OK);

        break;

    default:

        cout << "invalid choice" << endl;

        break;

    }

    cout << "You want to continue as agent ?If yes press 0" << endl;

    cin >> choice;

    if (choice != 0)

        break;

    }

}

else

{

    cout << "invalid choice" << endl;

}

cout << "You want to continue ?If yes press 0" << endl;

cin >> choice;

if (choice != 0)

    break;

}

}

catch(...){
    MessageBox(NULL,"Error Encountered","Error 69",MB_OK);
}

return 0;

}

```


Output:




```
if (choice == 1 || choice == 2 || choice == 3 || choice == 4)
{
    cout << "Kindly schedule appointment" << endl;
}

static DWORD WINAPI threadFunction(LPVOID lpParam)
{
    cout << "\n\nAgent is providing information to Tenant in other room." << endl;
    return 0;
}

void ScheduleAppointment()
{
    int time;
    cout << "choose the time slot you wish to come" << endl;
    << "\n1. (11:00am-13:00pm)"
    << "\n2. (13:00pm-14:00pm)"
    << "\n3. (15:00pm-16:00pm)";
    cin >> time;
}

void CheckAppointment(int n)
{
}
```

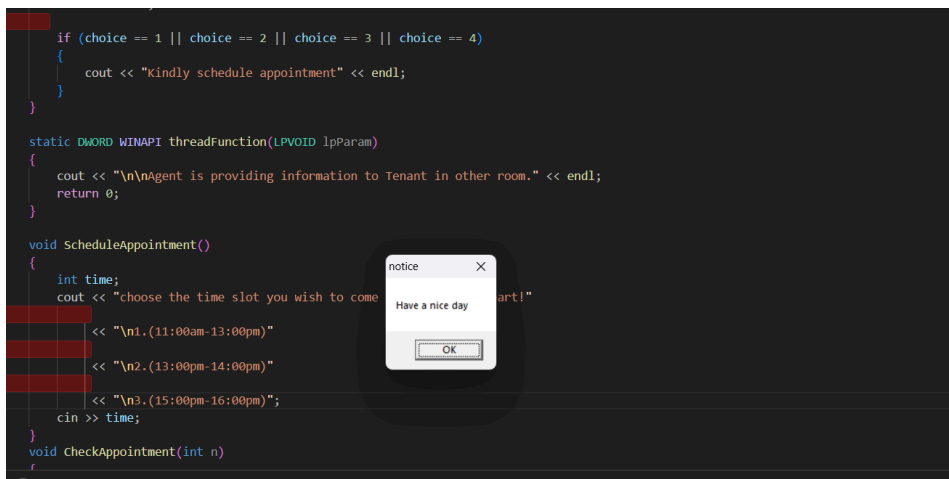


```
if (choice == 1 || choice == 2 || choice == 3 || choice == 4)
{
    cout << "Kindly schedule appointment" << endl;
}

static DWORD WINAPI threadFunction(LPVOID lpParam)
{
    cout << "\n\nAgent is providing information to Tenant in other room." << endl;
    return 0;
}

void ScheduleAppointment()
{
    int time;
    cout << "choose the time slot you wish to come" << endl;
    << "\n1. (11:00am-13:00pm)"
    << "\n2. (13:00pm-14:00pm)"
    << "\n3. (15:00pm-16:00pm)";
    cin >> time;
}

void CheckAppointment(int n)
{
}
```



```
if (choice == 1 || choice == 2 || choice == 3 || choice == 4)
{
    cout << "Kindly schedule appointment" << endl;
}

static DWORD WINAPI threadFunction(LPVOID lpParam)
{
    cout << "\n\nAgent is providing information to Tenant in other room." << endl;
    return 0;
}

void ScheduleAppointment()
{
    int time;
    cout << "choose the time slot you wish to come" << endl;
    << "\n1. (11:00am-13:00pm)"
    << "\n2. (13:00pm-14:00pm)"
    << "\n3. (15:00pm-16:00pm)";
    cin >> time;
}

void CheckAppointment(int n)
{
}
```

● Conclusion

Process synchronization is an important concept in operating system design and implementation, and it plays a crucial role in ensuring the correct and efficient execution of concurrent programs.

There are various mechanisms for achieving process synchronization, including locks, semaphores, monitors, and barriers, each with its own strengths and weaknesses.

Choosing the right synchronization mechanism for a particular application requires careful consideration of factors such as performance, scalability, and ease of use. Poorly designed or implemented synchronization mechanisms can lead to a range of problems, including deadlocks, and race conditions, which can result in incorrect or unpredictable program behaviour.

Testing and debugging concurrent programs can be challenging, and it is important to have robust testing and debugging tools and techniques in place to ensure the correctness of the program.

Future Scope:

1.Workflow management: In property dealing, there are many different steps involved in the process, such as property search, listing, negotiation, and closing the deal. Process synchronization can be used to manage the workflow and ensure that different steps are executed in the correct order.

2.Collaboration: Property dealing often involves multiple parties, such as buyers, sellers, real estate agents, and lawyers. Process synchronization can be used to ensure that everyone is working together effectively and that there are no conflicts or delays.

3.Data management: Property dealing involves a lot of data, such as property listings, contracts, and financial records. Process synchronization can be used to ensure that different parties have access to the correct data at the right time and that there are no conflicts or inconsistencies in the data.