1) Lamport's

CODE - 1

```
#include <bits/stdc++.h>
using namespace std;
using namespace chrono;
using namespace this_thread;

// Node class representing a node in the distributed system
class Node
{
public:
    vector<Node> &nodes;     // reference to all nodes in the system
    int id;                  // unique id of the node
    int timestamp;           // timestamp of the node's latest request
    bool inCriticalSection;  // whether the node is currently in the critical section
    queue<int> requestQueue; // queue of nodes requesting access to the critical section

public:
    // constructor for initializing a node
    Node(int id, vector<Node> &nodes) : nodes(nodes)
    {
        this->id = id;
        this->timestamp = 0;
        this->inCriticalSection = false;
    }

    // function for requesting access to the critical section
    void requestCriticalSection()
    {
        this->timestamp++;
        for (auto &node : nodes)
        {
            if (node.id != this->id)
            {
                node.receiveRequest(this->id, this->timestamp);
            }
        }
        this->requestQueue.push(this->id);
    }

    // function for releasing access to the critical section
    void releaseCriticalSection()
    {
        this->inCriticalSection = false;
        for (auto &node : nodes)
        {
            if (node.id != this->id)
            {
                node.receiveRelease(this->id);
            }
        }
        this->requestQueue.pop();
    }

    // function for receiving a request for access to the critical section
    void receiveRequest(int senderID, int senderTimestamp)
    {
        this->timestamp = max(this->timestamp, senderTimestamp);
        if (this->inCriticalSection || (senderTimestamp < this->timestamp) || ((senderTimestamp == this->timestamp) && (senderID <
this->id)))
        {
            this->sendReply(senderID);
        }
        else
        {
            this->requestQueue.push(senderID);
        }
    }

    // function for receiving a release of access to the critical section
    void receiveRelease(int senderID)
    {
        this->requestQueue.pop();
    }
```

```
    // function for sending a reply to a node
    void sendReply(int destinationID)
    {
        for (auto &node : nodes)
        {
            if (node.id == destinationID)
            {
                node.receiveReply(this->id);
            }
        }
    }

    // function for receiving a reply
    void receiveReply(int senderID)
    {
        if (this->requestQueue.front() == this->id)
        {
            this->inCriticalSection = true;
        }
    }
};

int main()
{
    int num_nodes, num_requests;
    cout << "Number of nodes available in the distributed system: ";
    cin >> num_nodes;
    vector<pair<int, int>> requests;
    cout << "Number of requests to access critical section: ";
    cin >> num_requests;
    for (int i = 0; i < num_requests; i++)
    {
        int node, time;
        cout << "Enter node id and time for requesting access: ";
        cin >> node >> time;
        requests.push_back(make_pair(node, time));
    }
    // create the nodes in the system
    vector<Node> nodes;
    for (int i = 0; i < num_nodes; i++)
    {
        nodes.push_back(Node(i, nodes));
    }

    // sort the requests by time
    sort(requests.begin(), requests.end(), [](auto &a, auto &b)
         { return a.second < b.second; });

    cout << "<---Sequence of accessing the critical section--->" << endl;
    for (auto &request : requests)
    {
        // wait for the specified time before making request
        sleep_for(seconds(request.second));
        nodes[request.first].requestCriticalSection();
        cout << "           Node ID - " << request.first << " | Timestamp - " << nodes[request.first].timestamp << endl;
        // wait for 2 seconds before releasing access to the critical section
        sleep_for(seconds(2));
        nodes[request.first].releaseCriticalSection();
    }
    cout << "           END" << endl;
    return 0;
}
```

CODE 2

```
#include <iostream>
#include <vector>
#include <queue>

using namespace std;
```

```
class Node
{
private:
    vector<Node> &nodes;
    int id;
    int timestamp;
    bool in_cs;
    std::queue<int> request_queue;

public:
    Node(int id, vector<Node> &nodes) : nodes(nodes)
    {
        this->id = id;
        this->timestamp = 0;
        this->in_cs = false;
    }

    void request_entry()
    {
        this->timestamp++;
        for (auto &node : nodes)
        {
            if (node.id != this->id)
            {
                node.receive_request(this->id, this->timestamp);
            }
        }
        this->request_queue.push(this->id);
        std::cout << "Process " << this->id << " is requesting entry to the critical section." << std::endl;
    }

    void release_section()
    {
        this->in_cs = false;
        for (auto &node : nodes)
        {
            if (node.id != this->id)
            {
                node.receive_release(this->id);
            }
        }
        this->request_queue.pop();
        std::cout << "Process "<< this->id << " is leaving the critical section." << std::endl;
    }
    void receive_request(int sender_id, int sender_timestamp)
    {
        this->timestamp = std::max(this->timestamp, sender_timestamp);
        if (this->in_cs || (sender_timestamp < this->timestamp) ||
            ((sender_timestamp == this->timestamp) && (sender_id < this->id)))
        {
            this->send_acknowledgment(sender_id);
        }
        else
        {
            this->request_queue.push(sender_id);
        }
    }

    void receive_release(int sender_id)
    {
        this->request_queue.pop();
    }

    void send_acknowledgment(int dest_id)
```

```
    {
        for (auto &node : nodes)
        {
            if (node.id == dest_id)
            {
                node.receive_acknowledgment(this->id);
            }
        }
    }

    void receive_acknowledgment(int sender_id)
    {
        if (this->request_queue.front() == this->id)
        {
            this->in_cs = true;
            std::cout << "Process " << this->id << " is entering the critical section." << std::endl;
        }
    }
};

int main(int argc, char **argv)
{
    int num_nodes;
    std::cout << "Enter the number of nodes in the distributed system: ";
    std::cin >> num_nodes;
    vector<Node> nodes;
    for (int i = 0; i < num_nodes; i++)
    {
        nodes.push_back(Node(i, nodes));
    }

    std::cout << "Enter the nodes requesting for access to the critical section, one at a time: " << std::endl;
    int requesting_node;
    while (std::cin >> requesting_node)
    {
        nodes[requesting_node].request_entry();
        nodes[requesting_node].release_section();
    }
    return 0;
}
```

2) Ricart Agarwala

CODE-1

```
#include <iostream>

#include <vector>

#include <map>

#include <queue>

#include <cstdio>

#include <cstdlib>

#include <ctime>

#include <unistd.h>



using namespace std;



int num_of_sites;

int time_stamp = 0;

bool in_crit_sec[100];

map<int, bool> deferred_reqs[100];



// structure to store REQUEST message

struct ReqMessage

{

    int site_identifier;

    int req_timestamp;

};



// function to send REQUEST message

void send_request_message(int site_identifier)

{
```

```
    time_stamp++;

    ReqMessage request;

    request.site_identifier = site_identifier;

    request.req_timestamp = time_stamp;

    for (int i = 0; i < num_of_sites; i++)

    {

        if (i == site_identifier)

            continue;

        // receive request at site i

        if (!in_crit_sec[i])

        {

            // if site i is not in critical section, it sends REPLY

            cout << "Site " << i << " sent REPLY to Site " << site_identifier << endl;

        }

        else

        {

            // if site i is in critical section, it defers the request

            deferred_reqs[i][site_identifier] = true;

            cout << "Site " << i << " deferred request from Site " << site_identifier << endl;

        }

    }

}



// function to enter critical section

void enter_critical_section(int site_identifier)

{

    in_crit_sec[site_identifier] = true;

    cout << "Site " << site_identifier << " entered critical section" << endl;
```

```
    // simulate critical section
    int wait_duration;
    cout << "Enter wait duration (in sec) for Site " << site_identifier << ": ";
    cin >> wait_duration;
    cout << "Site " << site_identifier << " is executing critical section for " << wait_duration << " sec" << endl;
    sleep(wait_duration);
}

// function to release critical section
void release_critical_section(int site_identifier)
{
    in_crit_sec[site_identifier] = false;
    cout << "Site " << site_identifier << " exited critical section" << endl;
    for (auto deferred_request : deferred_reqs[site_identifier])
    {
        int deferred_site_id = deferred_request.first;
        cout << "Site " << site_identifier << " sent REPLY to Site " << deferred_site_id << endl;
    }
    deferred_reqs[site_identifier].clear();
}

int main()
{
    cout << "Enter number of sites: ";
    cin >> num_of_sites;

    for (int i = 0; i < num_of_sites; i++)
    {
```

```
    send_request_message(i);

    enter_critical_section(i);

    release_critical_section(i);

  }

  return 0;

}
```

CODE - 2

```
#include <iostream>
#include <vector>
#include <map>
#include <queue>
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <unistd.h>
using namespace std;

int num_sites;

struct Request
{
    int site_id;
    int request_timestamp;
};

struct Reply
{
    int site_id;
};

class Site
{
private:
    int site_id;
    int timestamp;
    bool in_critical_section;
    map<int, bool> deferred_requests;

    void send_request()
    {
        Request req;
        req.site_id = site_id;
        req.request_timestamp = timestamp;
        for (int i = 0; i < num_sites; i++)
        {
            if (i == site_id)
                continue;
            if (!in_critical_section)
            {
                cout << "Site " << site_id << " -> REQUEST -> Site " << i;
                cout << "   ==  ";
                cout << "Site " << i << " -> REPLY -> Site " << site_id << endl;
            }
            else
```

```
            {
                deferred_requests[i] = true;
                cout << "Site " << i << " deferred REQUEST from Site " << site_id << endl;
            }
        }
    }

    void enter_critical_section()
    {
        in_critical_section = true;
        cout << "Site " << site_id << " entered critical section" << endl;
        srand(time(0));
        int wait_time = rand() % 5 + 1;
        cout << "Site " << site_id << " is executing critical section for " << wait_time << " sec" << endl;
        sleep(wait_time);
    }

    void release_critical_section()
    {
        in_critical_section = false;
        cout << "Site " << site_id << " exited critical section" << endl;
        for (auto deferred_request : deferred_requests)
        {
            int deferred_site_id = deferred_request.first;
            cout << "Site " << site_id << " sent REPLY to Site " << deferred_site_id << endl;
        }
        deferred_requests.clear();
        cout << endl;
    }

public:
    Site(int id, int ts)
    {
        site_id = id;
        timestamp = ts;
        in_critical_section = false;
    }

    void run()
    {
        send_request();
        enter_critical_section();
        release_critical_section();
    }
};

int main()
{
    cout << "Enter number of Sites: ";
    cin >> num_sites;
    vector<Site> sites;
    for (int i = 0; i < num_sites; i++)
    {
        cout << "Enter id for site " << i + 1 << " - ";
        int id;
        cin >> id;
        cout << "Enter timestamp for site " << i + 1 << " - ";
        int timestamp;
        cin >> timestamp;
        sites.push_back(Site(id, timestamp));
    }
    cout << endl;
    for (auto site : sites)
    {
        site.run();
```

```
    }
    return 0;
}
```

3) Mackawa

CODE-1

```
#include <iostream>
#include <map>
#include <vector>
#include <queue>
#include <thread>
#include <chrono>

std::map<int, std::vector<int>> requestSet;
std::map<int, std::queue<int>> requestQueue;
std::map<int, bool> replyStatus;
int n = 0;

void send_request(int site, int process)
{
    if (replyStatus[process] == false)
    {
        replyStatus[process] = true;
        std::cout << "Site " << site << " sent REQUEST to Process " << process << " and received REPLY" <<
std::endl;
    }
    else
    {
        requestQueue[process].push(site);
        std::cout << "Site " << site << " sent REQUEST to Process " << process << " and is waiting in queue" <<
std::endl;
    }
}

void send_release(int site, int process)
{
    if (!requestQueue[process].empty())
    {
        int nextSite = requestQueue[process].front();
        requestQueue[process].pop();
        replyStatus[process] = true;
        std::cout << "Site " << site << " sent RELEASE to Process " << process << " and sent REPLY to Site " <<
nextSite << std::endl;
    }
    else
    {
        replyStatus[process] = false;
        std::cout << "Site " << site << " sent RELEASE to Process " << process << " and updated reply status" <<
std::endl;
    }
}

void enter_critical_section(int site)
{
    for (int process : requestSet[site])
    {
        send_request(site, process);
    }
    for (int process : requestSet[site])
    {
```

```
        while (replyStatus[process] == false)
        {
        }
    }
    std::cout << "Site " << site << " entered Critical Section" << std::endl;
    std::this_thread::sleep_for(std::chrono::seconds(2));
    std::cout << "Site " << site << " finished executing Critical Section" << std::endl;
    for (int process : requestSet[site])
    {
        send_release(site, process);
    }
}

int main()
{
    // Input the number of sites
    std::cout << "Enter the number of sites: ";
    std::cin >> n;

    // Input the request set for each site
    for (int i = 0; i < n; i++)
    {
        int process;
        std::vector<int> set;
        std::cout << "Enter the request set for Site " << i << ": ";
        while (std::cin >> process)
        {
            if (process == -1)
                break;
            set.push_back(process);
        }
        requestSet[i] = set;
    }

    // Initialize the reply status for each process
    for (int i = 0; i < n; i++)
    {
        replyStatus[i] = false;
    }

    // Enter the critical section for each site
    for (int i = 0; i < n; i++)
    {
        std::thread t(enter_critical_section, i);
        t.detach();
    }
    // Wait for all sites to finish executing the critical section
    while (true)
    {
    }

    return 0;
}
```

CODE - 2

```
#include <iostream>
#include <vector>
#include <queue>
#include <ctime>
#include <unistd.h>

using namespace std;

const int N = 5; // number of sites
```

```
const int K = 2; // size of request set

vector<int> request_set[N];
bool reply[N];
queue<int> request_queue[N];
bool in_critical_section[N];

void print_request_set()
{
    cout << "Request sets: " << endl;
    for (int i = 0; i < N; i++)
    {
        cout << "Site " << i << ": [ ";
        for (int j = 0; j < request_set[i].size(); j++)
        {
            cout << request_set[i][j] << " ";
        }
        cout << "]" << endl;
    }
    cout << endl;
}

void send_request(int site)
{
    cout << "Site " << site << " is sending REQUEST to sites: [ ";
    for (int i = 0; i < request_set[site].size(); i++)
    {
        int recipient = request_set[site][i];
        cout << recipient << " ";
        if (!reply[recipient])
        {
            reply[recipient] = true;
            request_queue[recipient].push(site);
        }
    }
    cout << "]" << endl;
}

void send_reply(int site, int recipient)
{
    cout << "Site " << site << " is sending REPLY to Site " << recipient << endl;
    reply[site] = false;
}

void send_release(int site)
{

    for (int i = 0; i < request_set[site].size(); i++)
    {
        int recipient = request_set[site][i];
        cout << "Site " << site << " is sending RELEASE to Site " << recipient << endl;
        if (!request_queue[recipient].empty())
        {
            int next = request_queue[recipient].front();
            request_queue[recipient].pop();
            send_reply(recipient, next);
        }
        else
        {
            reply[recipient] = false;
        }
    }
    cout << endl;
}

void enter_critical_section(int site)
{
    cout << "Site " << site << " is entering the critical section" << endl;
    in_critical_section[site] = true;
    // simulation of critical section
    srand(time(0));
    int wait_time = rand() % 5 + 1;
    cout << "Site " << site << " is executing critical section for " << wait_time << " sec" << endl;
    sleep(wait_time);
}
```

```
void release_critical_section(int site)
{
    cout << "Site " << site << " is releasing the critical section" << endl;
    in_critical_section[site] = false;
    send_release(site);
}

int main()
{
    // initialize request set
    cout << "Enter number of sites: ";
    int n;
    cin >> n;
    cout << "Enter number of request set: ";
    int k;
    cin >> k;
    for (int i = 0; i < N; i++)
    {
        int j = i;
        for (int k = 1; k <= K; k++)
        {
            request_set[i].push_back((j + k) % N);
        }
    }
    print_request_set();

    for (int i = 0; i < N; i++)
    {
        send_request(i);
        int site = i;
        bool granted = false;
        while (!granted)
        {
            granted = true;
            for (int j = 0; j < N; j++)
            {
                if (reply[j] && request_queue[j].front() < site)
                {
                    granted = false;
                    break;
                }
            }
        }
        enter_critical_section(site);
        release_critical_section(site);
    }

    return 0;
}
```

4) Suzuki Kasami

```
#include <iostream>
#include <queue>
#include <vector>
#include <unistd.h>
#include <ctime>
using namespace std;

// Number of sites
#define N 5

// Data structures
int RN[N] = {0};     // Request number array
int LN[N] = {0};     // Last executed request array
bool token = false; // Token availability
queue<int> Q;        // Queue for waiting sites

// Function to send request message to all sites
void sendRequest(int siteId)
{
    RN[siteId]++; // Increment request number
```

```
    int sn = RN[siteId];
    cout << "Site " << siteId << " sends request message with sequence number " << sn << endl;
    for (int i = 0; i < N; i++)
    {
        if (i != siteId)
        {
            // Send request message to site i
            cout << "Site " << siteId << " sends request message to site " << i << endl;
            // Update RN[i]
            RN[i] = max(RN[i], sn);
        }
    }
}

// Function to send token to site with siteId
void sendToken(int siteId)
{
    token = false; // Token not available now
    cout << "Site " << siteId << " sends token" << endl;
    LN[siteId]++; // Update LN[i]
}

// Function to enter critical section
void enterCritical(int siteId)
{
    if (token) // If token available
    {
        cout << "Site " << siteId << " enters critical section" << endl;
    }
    else
    {
        // Send request message to all sites
        sendRequest(siteId);
    }
}

// Function to execute critical section
void executeCritical(int siteId)
{
    cout << "Site " << siteId << " executes critical section" << endl;
    srand(time(0));
    int wait_time = rand() % 5 + 1;
    cout << "Site " << siteId << " is executing critical section for " << wait_time << " sec" << endl;
    sleep(wait_time);
}

// Function to release critical section
void releaseCritical(int siteId)
{
    cout << "Site " << siteId << " releases critical section" << endl
         << endl;
    LN[siteId] = RN[siteId]; // Update LN[i]
    // Update queue Q
    for (int j = 0; j < N; j++)
    {
        if (j != siteId && RN[j] == LN[j] + 1)
        {
            Q.push(j);
        }
    }
    // If queue Q is non-empty, send token to first site in Q
    if (!Q.empty())
    {
        int nextSite = Q.front();
        Q.pop();
```

```
        sendToken(nextSite);
    }
    else // Else keep token
    {
        token = true;
        cout << "Site " << siteId << " keeps token" << endl;
    }
}

// Main function
int main()
{
    // Sample execution sequence
    int n;
    cout << "Enter number of sites: ";
    cin >> n;
    cout << endl;
    for (int i = 0; i < n; i++)
    {
        int site = (int)1 + (rand() % n);
        enterCritical(site);
        executeCritical(site);
        releaseCritical(site);
    }
    return 0;
}
```

5) Singhal's Heuristic

```
#include <iostream>

#include <vector>

#include <queue>

#include <thread>

#include <chrono>

#include <algorithm>

#include <cstring>



using namespace std;



// Function to handle the reply event

void reply_handler(int site_id, int my_id, vector<int>& request_set, vector<int>& reply_set) {

   cout << "Site " << my_id << " received reply from site " << site_id << endl;

   // Remove the site from the request set
```

```
    remove(request_set.begin(), request_set.end(), site_id);
    // Add the site to the reply set
    reply_set.push_back(site_id);
}

// Function to handle the request event
void request_handler(int site_id, int my_id, vector<int>& request_set, vector<int>& reply_set) {
    cout << "Site " << my_id << " received request from site " << site_id << endl;
    // Add the site to the request set
    request_set.push_back(site_id);
    // Sort the request set
    sort(request_set.begin(), request_set.end());
    // If the site is the first in the request set, reply to it
    if (request_set[0] == my_id) {
        // Remove the site from the request set
        remove(request_set.begin(), request_set.end(), my_id);
        // Add all the sites in the reply set to the request set
        for (auto site : reply_set) {
            request_set.push_back(site);
        }
        // Clear the reply set
        reply_set.clear();
        // Reply to the site
        for (auto site : request_set) {
            if (site != my_id) {
                // Simulate network delay
                this_thread::sleep_for(chrono::milliseconds(100));
                reply_handler(site, my_id, request_set, reply_set);
```

```
            }
        }
    }
}

int main() {
    // Number of sites in the system
    int num_sites;
    cout << "Enter the number of sites in the system: ";
    cin >> num_sites;

    // Create request and reply sets for each site
    vector<vector<int>> request_sets(num_sites);
    vector<vector<int>> reply_sets(num_sites);

    // Run the algorithm for 10 iterations
    int num_iterations = 10;
    while (num_iterations--) {
        // Get the request from the user
        int site_id;
        cout << "Enter the site id requesting access to critical section: ";
        cin >> site_id;

        // Simulate a request from the site
        request_handler(site_id, site_id, request_sets[site_id], reply_sets[site_id]);

        // Simulate some processing
        this_thread::sleep_for(chrono::milliseconds(100));
```

```
  }



  return 0;

}
```

6) Raymond Tree Based (V. Hard)

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <unistd.h>
#include <ctime>
using namespace std;
struct Node
{
    int id;
    int value;
    struct Node *left;
    struct Node *right;
    int required[20];
};

typedef struct Node Node;

void inorderTraversal(Node *rootNode)
{
    if (rootNode == NULL)
    {
        return;
    }
    inorderTraversal(rootNode->left);
    printf("%d %d\n", rootNode->id, rootNode->value);
    inorderTraversal(rootNode->right);
}

void token(Node *rootNode, int targetId)
{
    if (targetId == rootNode->id)
    {
        printf("%d\n", rootNode->id);
        rootNode->value = rootNode->id;
        return;
    }
    else if (targetId < rootNode->id)
    {
        rootNode->value = (rootNode->left)->id;
        printf("%d->", rootNode->id);
        rootNode = rootNode->left;
        token(rootNode, targetId);
    }
    else if (targetId > rootNode->id)
    {
        rootNode->value = (rootNode->right)->id;
        printf("%d->", rootNode->id);
        rootNode = rootNode->right;
        token(rootNode, targetId);
    }
}
void enterCritical(int siteId)
```

```
{
    cout << "Site " << siteId << " enters critical section" << endl;
}
void executeCritical(int siteId)
{
    srand(time(0));
    int wait_time = rand() % 5 + 1;
    cout << "Site " << siteId << " is executing critical section for " << wait_time << " sec" << endl;
    sleep(wait_time);
}
void releaseCritical(int siteId)
{
    cout << "Site " << siteId << " releases critical section" << endl;
}
void insertNode(Node *newNode, Node *rootNode)
{
    if (newNode->id > rootNode->id)
    {
        if (rootNode->right == NULL)
        {
            rootNode->right = newNode;
            newNode->value = rootNode->id;
        }
        else
        {
            insertNode(newNode, rootNode->right);
        }
    }
    if (newNode->id < rootNode->id)
    {
        if (rootNode->left == NULL)
        {
            rootNode->left = newNode;
            newNode->value = rootNode->id;
        }
        else
        {
            insertNode(newNode, rootNode->left);
        }
    }
}
int main()
{
    Node *rootNode = NULL, *newNode = NULL, *node1;
    int i;
    int numNodes;
    cout << "Enter number of nodes: ";
    cin >> numNodes;
    int rootNodeId;
    cout << "Enter root node ID: ";
    cin >> rootNodeId;
    int idValue;
    int arr[numNodes];
    for (i = 0; i < numNodes; i++)
    {
        cout << "Enter Node " << i + 1 << " : ";
        cin >> arr[i];
    }

    int targetId, option;
    rootNode = (struct Node *)malloc(sizeof(Node));
    node1 = (struct Node *)malloc(sizeof(Node));
    rootNode->id = rootNodeId;
    rootNode->right = rootNode->left = NULL;
    rootNode->value = rootNode->id;
```

```
    for (i = 0; i < numNodes; i++)
    {
        idValue = arr[i];
        newNode = (struct Node *)malloc(sizeof(Node));
        newNode->left = newNode->right = NULL;
        if (i == rootNodeId)
            i++;
        newNode->id = idValue;
        insertNode(newNode, rootNode);
    }
    inorderTraversal(rootNode);
    cout << endl;
    for (i = 0; i < numNodes; i++)
    {
        int siteID = (int)1 + (rand() % numNodes - 1);
        cout << "Site " << siteID << " sends REQUEST to = ";
        token(rootNode, siteID);
        enterCritical(siteID);
        executeCritical(siteID);
        releaseCritical(siteID);
        cout << endl;
    }
    return 0;
}
```

7) Ramamoorthy 1 Phase Algo

```
#include <bits/stdc++.h>

using namespace std;

class WaitForGraph
{
public:
    vector<unordered_set<int>> adjList;
    vector<unordered_set<int>> modifiedList;
    map<int, bool> visited;

    WaitForGraph(int n)
    {
        adjList.resize(n);
        modifiedList.resize(n);
    }
    void createEdge(int p, int r)
    {
        adjList[p].insert(r);
    }

    void createModifiedList()
    {
        for (int i = 0; i < adjList.size(); i++)
        {
            int count = 0;
            for (auto it = adjList[i].begin(); it != adjList[i].end(); it++)
            {
                if (count != 0)
                {
                    // cout << i << endl;
                    // cout << i << " " << count << " " << *it << endl;
                    for (int j = 0; j < adjList.size(); j++)
                    {
                        if (j != i)
```

```
                        {
                            auto ele = adjList[j].begin();
                            // cout<<i<<" "<<*ele<<endl;
                            if (*ele == *it)
                            {
                                modifiedList[i].insert(j);
                            }
                        }
                    }
                }
                count++;
            }
        }
    }

    void printGraph(vector<unordered_set<int>> l)
    {
        for (int i = 0; i < l.size(); i++)
        {
            cout << i << "->";
            for (auto it = l[i].begin(); it != l[i].end(); it++)
            {
                cout << *it << " ";
            }
            cout << endl;
        }
    }
};

int main()
{
    WaitForGraph graph(2);
    graph.createEdge(0, 2); // ASSIGNED
    graph.createEdge(0, 1); // REQUESTING

    graph.createEdge(1, 1); // ASSIGNED
    graph.createEdge(1, 2); // REQUESTING

    // graph.printGraph(graph.adjList);

    graph.createModifiedList();
    graph.printGraph(graph.modifiedList);

    return 1;
}
```

8) Ramamoorthy 2 Phase Algo

9) 2 Phase commit protocol

10) Cristian Algo

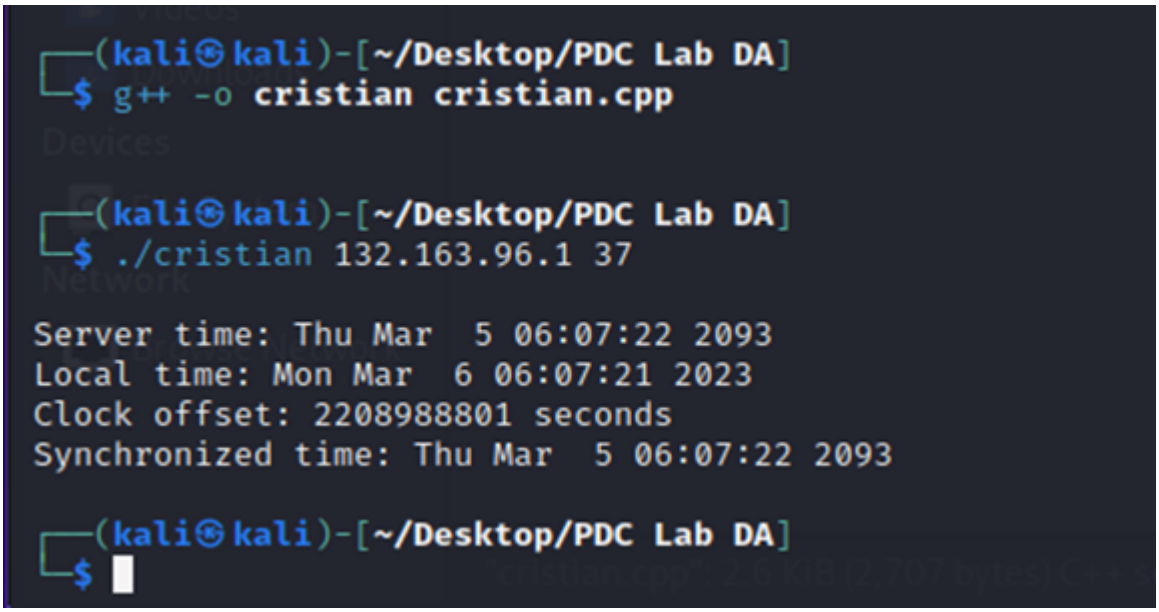
```
#include <iostream>

#include <ctime>

#include <unistd.h>
```

```cpp
#include <sys/types.h>

#include <sys/socket.h>

#include <netdb.h>

#include <unistd.h>

#include <ctime>

#include <strings.h>



using namespace std;



// Function to connect to server and return client socket file descriptor

int connectToServer(char* serverIP, int serverPort) {

    int clientSocket;

    struct sockaddr_in serverAddr;

    struct hostent *server;



    // Get server IP address

    server = gethostbyname(serverIP);

    if (server == NULL) {

        cerr << "ERROR: Could not resolve server address" << endl;

        exit(1);

    }



    // Create socket

    clientSocket = socket(AF_INET, SOCK_STREAM, 0);

    if (clientSocket < 0) {

        cerr << "ERROR: Could not create socket" << endl;

        exit(1);

    }
```

```
    // Connect to server
    bzero((char *) &serverAddr, sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    bcopy((char *) server->h_addr, (char *) &serverAddr.sin_addr.s_addr, server->h_length);
    serverAddr.sin_port = htons(serverPort);
    if (connect(clientSocket, (struct sockaddr *) &serverAddr, sizeof(serverAddr)) < 0) {
        cerr << "ERROR: Could not connect to server" << endl;
        exit(1);
    }

    return clientSocket;
}

time_t getTimeFromServer(int clientSocket) {
    time_t serverTime;
    int n;

    // Read server time from socket
    n = read(clientSocket, &serverTime, sizeof(serverTime));
    if (n < 0) {
        cerr << "ERROR: Could not read from socket" << endl;
        exit(1);
    }

    // Convert from network byte order to host byte order
    serverTime = ntohl(serverTime);
```

```
    return serverTime;
}

int main(int argc, char *argv[]) {
    // Check that server IP and port were provided as command line arguments
    if (argc != 3) {
        cerr << "Usage: " << argv[0] << " <server IP> <server port>" << endl;
        exit(1);
    }

    // Get server IP address and port from command line arguments
    char* serverIP = argv[1];
    int serverPort = atoi(argv[2]);

    // Connect to server and retrieve current time
    int clientSocket = connectToServer(serverIP, serverPort);
    time_t serverTime = getTimeFromServer(clientSocket);
    close(clientSocket);

    // Calculate client's clock offset from server's clock
    time_t localTime = time(NULL);
    time_t clockOffset = serverTime - localTime;

    // Synchronize client's clock with server's clock
    time_t synchronizedTime = time(NULL) + clockOffset;
    cout << "Server time: " << ctime(&serverTime);
    cout << "Local time: " << ctime(&localTime);
    cout << "Clock offset: " << clockOffset << " seconds" << endl;
```

```
    cout << "Synchronized time: " << ctime(&synchronizedTime);



    return 0;

}
```



11) Berkeley Algo

Client.cpp

```
#include <stdio.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <string.h>

#include <iostream>

#include <stdlib.h>

#include <ctime>
```

```cpp
#include <vector>



#define PORT 8080



using namespace std;



// function for splitting a string

vector<string> split_string(string s, string delimiter) {

    size_t start_pos = 0, end_pos, delim_len = delimiter.length();

    string token;

    vector<string> res;



    while ((end_pos = s.find(delimiter, start_pos)) != string::npos) {

        token = s.substr(start_pos, end_pos - start_pos);

        start_pos = end_pos + delim_len;

        res.push_back(token);

    }



    res.push_back(s.substr(start_pos));

    return res;

}



int main(int argc, char const *argv[]) {

    srand((unsigned int)time(NULL)); // avoid always same output of rand()

    float client_clock = rand() % 10; // range from 0 to 9

    printf("Client starts. Client pid is %d \n", getpid());

    printf("Client local clock is %f \n\n", client_clock);
```

```
int client_sock_fd, valread;
char client_read_buf[1024] = {0};

struct sockaddr_in server_addr;
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(PORT);

// Creating socket file descriptor (IPv4, TCP, IP)
if ((client_sock_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    printf("\n Client: Socket creation error \n");
    return -1;
}

// Converting IPv4 and IPv6 addresses from text to binary form,
// from character string src into a network
// address structure in the af address family, then copies the
// network address structure to dst.
if(inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0) {
    printf("\nClient: Invalid address/ Address not supported \n");
    return -1;
}

// Connecting server, return 0 with success, return -1 with error
if (connect(client_sock_fd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
    printf("\nClient: Connection Failed \n");
    return -1;
}
```

```
char server_ip[INET_ADDRSTRLEN]="";
inet_ntop(AF_INET, &server_addr.sin_addr, server_ip, INET_ADDRSTRLEN);
printf("Client: connected server(%s:%d). \n", server_ip, ntohs(server_addr.sin_port));
printf("\n\n");


//
// First round communication
//


// Receiving from server
valread = read(client_sock_fd, client_read_buf, 1024);
printf("Client: read: '%s'\n", client_read_buf);


// Convert char array to string
string received_msg = string(client_read_buf);


// Reply according to what client received
if (strcmp(client_read_buf, "Hello from server, please tell me your local clock value.") == 0) {
    // Prepare msg
    string msg_str = "Hello from client, my local clock value is " + to_string(client_clock);
    char msg_char_array[msg_str.length() + 1];
    strcpy(msg_char_array, msg_str.c_str());


    // Sending a message to server
      send(client_socket_fd , &msg_char_array , strlen(msg_char_array) , 0 );
    printf("Client: sent message: '%s'\n", msg_char_array);
}
```

```
//

// second round communicattion

//



// receiving form server

valread = read( client_socket_fd , client_read_buffer, 1024);

printf("Client: read: '%s'\n",client_read_buffer );



// convert char array to string

recv_msg = string(client_read_buffer);



if (recv_msg.find("From server, your clock adjustment offset is") != string::npos){ // if latter is a substring of former

    string substr_after_lastbutone_space;

    string substr_after_last_space;

    vector<string> split_str = split(recv_msg, " ");

    substr_after_lastbutone_space = split_str[ split_str.size() - 2 ];

    substr_after_last_space = split_str[ split_str.size() - 1 ];



    cout << "Client: received local clock adjustment offset (string) is " << substr_after_lastbutone_space << " " << substr_after_last_space << endl;

    float substr_after_last_space_f = stof(substr_after_last_space);

    cout << "Client: received local clock adjustment offset (float) is " << substr_after_lastbutone_space << " " << substr_after_last_space_f << endl;



    char oper_char_array[substr_after_lastbutone_space.length() + 1];

    strcpy(oper_char_array, substr_after_lastbutone_space.c_str());

    if (strcmp(oper_char_array, "add") == 0 ){

        client_local_clock += substr_after_last_space_f;

    }else if (strcmp(oper_char_array, "minus") == 0 ){

        client_local_clock -= substr_after_last_space_f;
```

```
        }



        printf("Client local clock is %f \n\n", client_local_clock);

    }

    close(client_socket_fd);

    return 0;

}
```

Server.cpp

```
#include <iostream>

#include <iomanip>

#include <cstdlib>

#include <unistd.h>

#include <stdio.h>

#include <sys/socket.h>

#include <stdlib.h>

#include <netinet/in.h>

#include <string.h>

#include <arpa/inet.h>

#include <vector>
```

```cpp
#include <cstdlib>

#include <ctime>



#define PORT 8080



using namespace std;



vector<string> split_string(string str, string delimiter)

{

    size_t pos_start = 0, pos_end, delim_len = delimiter.length();

    string token;

    vector<string> res;



    while ((pos_end = str.find(delimiter, pos_start)) != string::npos)

    {

        token = str.substr(pos_start, pos_end - pos_start);

        pos_start = pos_end + delim_len;

        res.push_back(token);

    }



    res.push_back(str.substr(pos_start));

    return res;

}



float get_server_local_clock()

{

    srand((unsigned int)time(NULL));

    return rand() % 10;
```

```
}


void print_server_start_message(int server_pid, float server_local_clock)
{
    printf("Sever starts. Server pid is %d \n", server_pid);
    printf("Server local clock is %f \n\n", server_local_clock);
}


int create_socket_file_descriptor()
{
    int socket_fd;
    if ((socket_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
    {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    return socket_fd;
}


void set_reuse_of_address_and_port(int socket_fd)
{
    int opt = 1;
    if (setsockopt(socket_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt)))
    {
        perror("Socket setsockopt() failed");
        exit(EXIT_FAILURE);
    }
```

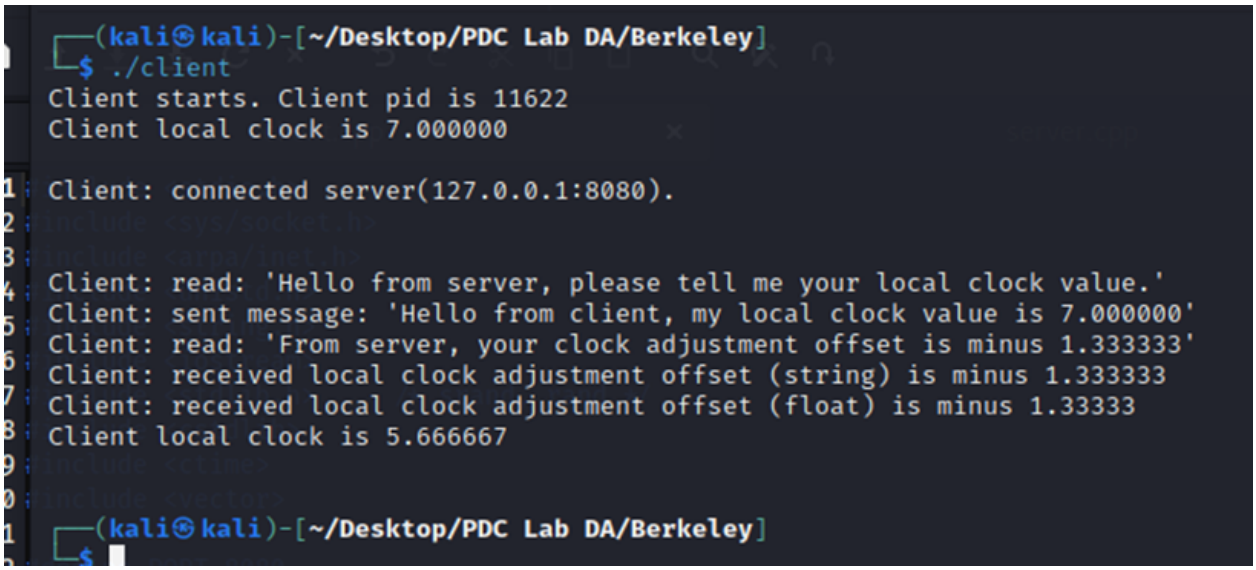
```
}



void bind_socket_to_port(int socket_fd, struct sockaddr_in address)

{

    if (bind(socket_fd, (struct sockaddr *)&address, sizeof(address)) < 0)

    {

        perror("Socket bind() failed");

        exit(EXIT_FAILURE);

    }

}



void listen_for_connections(int socket_fd)

{

    if (listen(socket_fd, 7) < 0)

    {

        perror("Socket listen() failed");

        exit(EXIT_FAILURE);

    }

}



void print_server_listening_message()

{

    printf("Server: server is listening ...\n\nYou can open one or multiple new terminal windows now to run ./client\n");

}



int accept_new_connection(int socket_fd, struct sockaddr_in client_address)

{

    int new_socket;
```

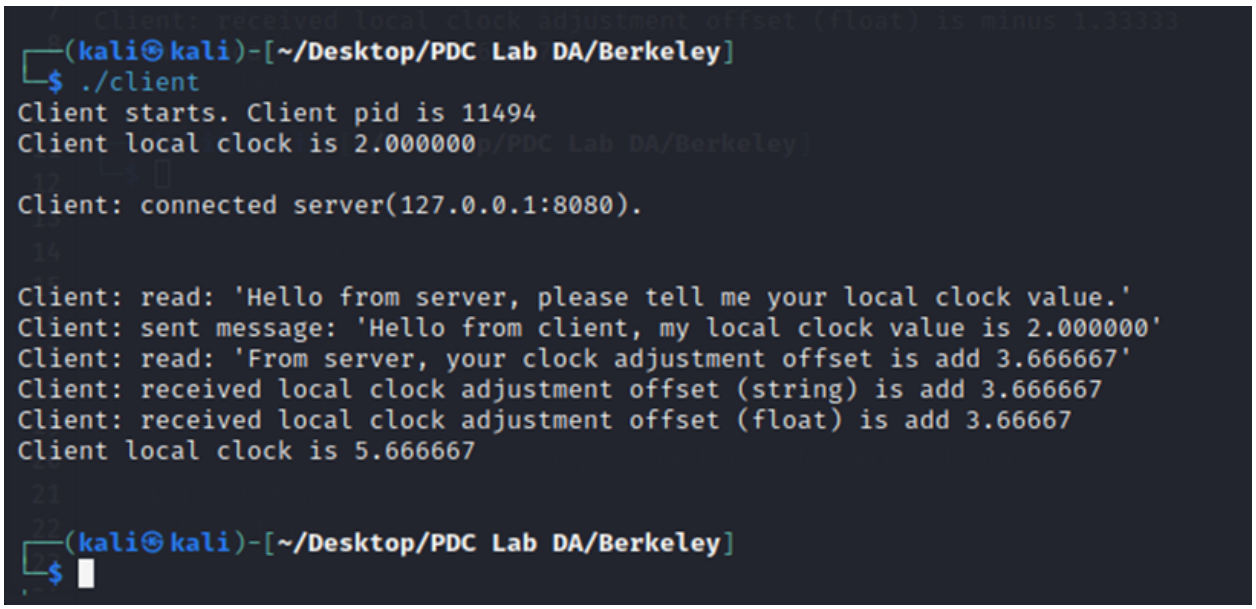
```
    socklen_t length = sizeof(client_address);



    if ((new_socket = accept(socket_fd, (struct sockaddr *)&client_address, (socklen_t *)&length)) < 0)

    {

        perror("Socket accept() failed");

        exit(EXIT_FAILURE);

    }



    return new_socket;

}



void print_new_client_accepted_message(int clients_ctr, char client_ip[], int client_port)

{

    printf("\nYou have connected %d client(s) now.", clients_ctr);

    printf("Server: new client accepted. client ip and port: %s:%d\n", client_ip, ntohs(client_port));

}



bool is_client_enough()

{

    int enough_clients;

    cout << "Do you have enough clients? (please input '1' for yes, '0' for no): ";

    cin >> enough_clients;

    return enough_clients == 1;

}



int main(int argc, char *argv[])

{

    srand((unsigned int)time(NULL));
```

```
float server_local_clock = get_server_local_clock();

print_server_start_message(getpid(), server_local_clock);



int server_socket_fd = create_socket_file_descriptor();

set_reuse_of_address

    // Using different variable and function names

    int server_socket_fd_2,

    new_socket_2, valread_2;

vector<int> client_sockets_2;

vector<string> client_ips_2;

vector<int> client_ports_2;

struct sockaddr_in server_address_2;

server_address_2.sin_family = AF_INET;         // IPv4

server_address_2.sin_addr.s_addr = INADDR_ANY; // localhost

server_address_2.sin_port = htons(PORT_2);     // 8000

int opt_2 = 1;                                 // for setsockopt



// Creating socket file descriptor (IPv4, TCP, IP)

if ((server_socket_fd_2 = socket(AF_INET, SOCK_STREAM, 0)) == 0)

{

    perror("Server: socket failed");

    exit(EXIT_FAILURE);

}



// Optional: it helps in reuse of address and port. Prevents error such as: “address already in use”.

if (setsockopt(server_socket_fd_2, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,

               &opt_2, sizeof(opt_2)))

{
```

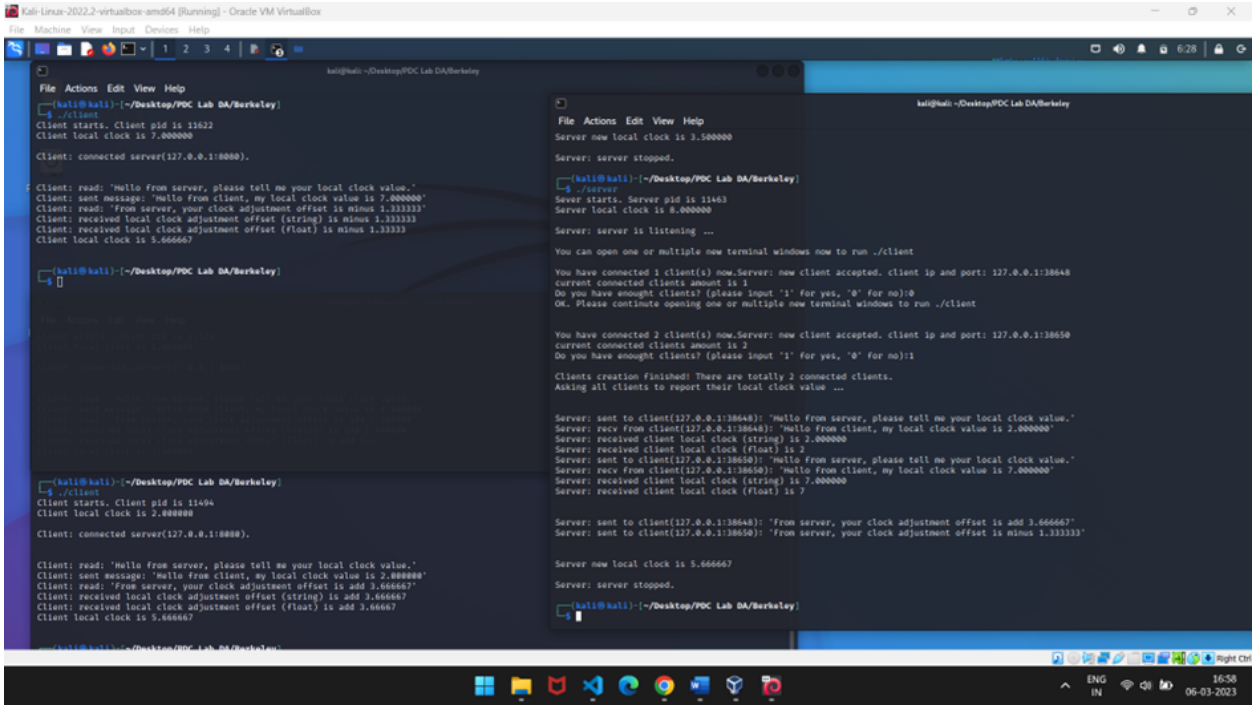
```
    perror("Server: setsockopt");
    exit(EXIT_FAILURE);
}


// Forcefully attaching socket to the port 8000
if (bind(server_socket_fd_2, (struct sockaddr *)&server_address_2,
         sizeof(server_address_2)) < 0)
{
    perror("Server: bind failed");
    exit(EXIT_FAILURE);
}


// Putting the server socket in a passive mode, waiting for the client to approach the server to make a connection
// The backlog=7, defines the maximum length to which the queue of pending connections for sockfd may grow.
// If a connection request arrives when the queue is full, the client may receive an error with an indication of ECONNREFUSED.
if (listen(server_socket_fd_2, 7) < 0)
{
    perror("Server: listen");
    exit(EXIT_FAILURE);
}
printf("Server: server is listening ...\n\nYou can open one or multiple new terminal windows now to run ./client\n");
int clients_ctr_2 = 0;


// Setting up buffer for receiving msg
char recv_buf_2[65536];
memset(recv_buf_2, '\0', sizeof(recv_buf_2));


int in_client_enough_2 = 0;
```

```
  while (in_client_enough_2 == 0)

  { // block on accept() until positive fd or error

    struct sockaddr_in client_addr_2;

    socklen_t length_2 = sizeof(client_addr_2);

    // Extracting the first connection request on the queue of pending connections for the listening socket (server_socket_fd)

    // Creates a new connected socket, and returns a new file descriptor referring to that socket

    if ((new_socket_2 = accept(server_socket_fd_2, (struct sockaddr *)&client_addr_2,

                               (socklen_t *)&length_2)) < 0)

    {

      perror("Server: accept");

      exit(EXIT_FAILURE);

    }



    clients_ctr_2++;

    printf("\nYou have connected %d client(s) now.", clients_ctr_2);



    // converting the network address structure src in the af address family into a character string.

    char client_ip_2[INET_ADDRSTRLEN] = "";

    inet_ntop(AF_INET, &client_addr_2.sin_addr, client_ip_2, INET_ADDRSTRLEN);

    printf("Server: new client accepted. client ip and port: %s:%d\n", client_ip_2, ntohs(client_addr_2.sin_port));



    // store new client connection into array

    client_sockets_2.push_back(new_socket_2);

    client_ips_2.push_back(client_addr_2);

  }

  return 1;

}
```


(kali㉿kali)-[~/Desktop/PDC Lab DA/Berkeley]
$ ./client
Client starts. Client pid is 11622
Client local clock is 7.000000
Client: connected server(127.0.0.1:8080).
Client: read: 'Hello from server, please tell me your local clock value.'
Client: sent message: 'Hello from client, my local clock value is 7.000000'
Client: read: 'From server, your clock adjustment offset is minus 1.333333'
Client: received local clock adjustment offset (string) is minus 1.333333
Client: received local clock adjustment offset (float) is minus 1.33333
Client local clock is 5.666667
(kali㉿kali)-[~/Desktop/PDC Lab DA/Berkeley]


(kali㉿kali)-[~/Desktop/PDC Lab DA/Berkeley]
$ ./client
Client starts. Client pid is 11494
Client local clock is 2.000000
Client: connected server(127.0.0.1:8080).
Client: read: 'Hello from server, please tell me your local clock value.'
Client: sent message: 'Hello from client, my local clock value is 2.000000'
Client: read: 'From server, your clock adjustment offset is add 3.666667'
Client: received local clock adjustment offset (string) is add 3.666667
Client: received local clock adjustment offset (float) is add 3.66667
Client local clock is 5.666667
(kali㉿kali)-[~/Desktop/PDC Lab DA/Berkeley]

```
┌──(kali㉿kali)-[~/Desktop/PDC Lab DA/Berkeley]
└─$ ./server
Sever starts. Server pid is 11463
Server local clock is 8.000000

Server: server is listening ...

You can open one or multiple new terminal windows now to run ./client

You have connected 1 client(s) now.Server: new client accepted. client ip and port: 127.0.0.1:38648
current connected clients amount is 1
Do you have enought clients? (please input '1' for yes, '0' for no):0
OK. Please continute opening one or multiple new terminal windows to run ./client


You have connected 2 client(s) now.Server: new client accepted. client ip and port: 127.0.0.1:38650
current connected clients amount is 2
Do you have enought clients? (please input '1' for yes, '0' for no):1

Clients creation finished! There are totally 2 connected clients.
Asking all clients to report their local clock value ...


Server: sent to client(127.0.0.1:38648): 'Hello from server, please tell me your local clock value.'
Server: recv from client(127.0.0.1:38648): 'Hello from client, my local clock value is 2.000000'
Server: received client local clock (string) is 2.000000
Server: received client local clock (float) is 2
Server: sent to client(127.0.0.1:38650): 'Hello from server, please tell me your local clock value.'
Server: recv from client(127.0.0.1:38650): 'Hello from client, my local clock value is 7.000000'
Server: received client local clock (string) is 7.000000
Server: received client local clock (float) is 7


Server: sent to client(127.0.0.1:38648): 'From server, your clock adjustment offset is add 3.666667'
Server: sent to client(127.0.0.1:38650): 'From server, your clock adjustment offset is minus 1.333333'


Server new local clock is 5.666667
```



12) Ring Algo

```
#include <bits/stdc++.h>
using namespace std;

// Printing details of current processes status, current coordinator
void printDetails(int nodes[], int coordinator, int n)
{
    // Printing Processes
    cout << "Processes--> ";
    for (int i = 1; i <= n; i++)
    {
        cout << "\t " << i;
    }
    cout << endl;
    // Printing Processes Status
    cout << "Alive--> ";
    for (int i = 1; i <= n; i++)
    {
        cout << "\t " << nodes[i];
    }
    cout << endl;
    // Printing Coordinator
    cout << "Coordinator--> " << coordinator << endl
         << endl;
}
int main()
{
    int n, coordinator;
    // Taking input of number of processes
    cout << "Enter number of processes: ";
    cin >> n;
    // Taking input of status of processes
    int processes[n + 1];
    for (int i = 1; i <= n; i++)
    {
        cout << "Enter 1 if process " << i << " is alive, 0 if dead: ";
        cin >> processes[i];
        if (processes[i])
        {
            coordinator = i;
        }
    }
    cout << endl;
    // Printing details
    printDetails(processes, coordinator, n);

    // Starting Ring Algorithm
    int message[n], ring_n, k, i;
    int ch, crash, activate, gid, flag, sub_coordinator;
    do
    {
        cout << "Choose an option from below." << endl;
        cout << "1.Crash the Process\n2.Activate the Process\n3.Exit the Program\n"
             << endl;
        cout << "Enter Choice: ";
        cin >> ch;
        switch (ch)
        {
        case 1: // Crashing the process
            cout << "Enter process to crash: ";
            cin >> crash;
            // Changing Status
            if (processes[crash])
            {
```

```
            processes[crash] = 0;
        }
        else // If process is already crashed
        {
            cout << "Proccess is already crashed" << endl;
            break;
        }
        do
        {
           // Taking input of election generator process id
            cout << "Enter election generator id: ";
            cin >> gid;
            if (gid == coordinator)
            {
                cout << "Enter valid generator id" << endl;
            }
        } while (gid == crash);
        cout << endl;
        flag = 0;
        k = 1;
       // Sending Election Message to other processes
        if (crash == coordinator)
        {
            message[k++] = gid;
            for (i = (gid + 1) % n; i != gid; i = (i + 1) % n)
            {
                if(i==0) continue;
                else  if (processes[i])
                {
                    printf("Election Message is sent to %d from %d\n", i, k);
                    message[k++] = i;
                }
            }
            sub_coordinator = 0;
            for (i = 1; i < k; i++)
            {
                printf("Message:%d ", message[i]);
                if (sub_coordinator < message[i])
                {
                    sub_coordinator = message[i];
                }
            }
            cout << endl
                 << endl;
            coordinator = sub_coordinator;
        }
        printDetails(processes, coordinator, n);
        break;

    case 2: // Activating the process
        cout << "Enter process to activate: ";
        cin >> activate;
       // Changing Process Status
        if (!processes[activate])
        {
            processes[activate] = 1;
        }
        else // If process is already activated
        {
            cout << "Proccess is already activated" << endl;
            break;
        }
        if (activate == n)
        {
            coordinator = n;
```

```
                printDetails(processes, coordinator, n);
                break;
            }
            for (i = activate + 1; i <= n; i++)
            {
                printf("Message is sent from %d to %d\n", activate, i);
                if (processes[i])
                {
                    sub_coordinator = i;
                    printf("Response is sent from %d to %d\n", i, activate);
                    flag = 1;
                }
            }
            if (flag == 1)
            {
                coordinator = sub_coordinator;
            }
            else
            {
                coordinator = activate;
            }
            printDetails(processes, coordinator, n);
        case 3:
            break;
        default:
            cout << "Enter Valid Choice" << endl
                 << endl;
        }
    } while (ch != 3);
    return 0;
}
```

13) Bully Algo

```
#include <bits/stdc++.h>
using namespace std;

// Printing details of current processes status, current coordinator
void printDetails(int nodes[], int coordinator, int n)
{
    // Printing Processes
    cout << "Processes--> ";
    for (int i = 1; i <= n; i++)
    {
        cout << "\t " << i;
    }
    cout << endl;
    // Printing Processes Status
    cout << "Alive--> ";
    for (int i = 1; i <= n; i++)
    {
        cout << "\t " << nodes[i];
    }
    cout << endl;
    // Printing Coordinator
    cout << "Coordinator--> " << coordinator << endl
         << endl;
}

int main()
{
    int n, coordinator;
```

```
// Taking input of number of processes
cout << "Enter number of processes: ";
cin >> n;
// Taking input of status of processes
int processes[n + 1];
for (int i = 1; i <= n; i++)
{
    cout << "Enter 1 if process " << i << " is alive, 0 if dead: ";
    cin >> processes[i];
    if (processes[i])
    {
        coordinator = i;
    }
}
cout << endl;
// Printing details
printDetails(processes, coordinator, n);

// Starting Bully Algorithm
int ch, crash, activate, gid, flag, sub_coordinator, i;
do
{
    cout << "Choose an option from below." << endl;
    cout << "1.Crash the Process\n2.Activate the Process\n3.Exit the Program\n"
         << endl;
    cout << "Enter Choice: ";
    cin >> ch;
    switch (ch)
    {
    case 1: // Crashing the process
        cout << "Enter process to crash: ";
        cin >> crash;
        // Changing Status
        if (processes[crash])
        {
            processes[crash] = 0;
        }
        else // If process is already crashed
        {
            cout << "Proccess is already crashed" << endl;
            break;
        }
        do
        {
            // Taking input of election generator process id
            cout << "Enter election generator id: ";
            cin >> gid;
            if (gid == coordinator)
            {
                cout << "Enter valid generator id" << endl;
            }
        } while (gid == crash);
        cout << endl;
        flag = 0;
        // Sending Election Message to other processes
        if (crash == coordinator)
        {
            for(int j=gid;j<=n;j++){
                int z = j;//gid
                for (i = z + 1; i <= n; i++)
                {
                    printf("Election Message is sent to %d from %d\n", i, z);
                    if (processes[i])
                    {
                        sub_coordinator = i;
```

```
                            printf("Response is sent from %d to %d\n", z, i);
                            flag = 1;
                        }
                    }
                }
                if (flag == 1)
                {
                    coordinator = sub_coordinator;
                }
                else
                {
                    coordinator = gid;
                }
            }
            printDetails(processes, coordinator, n);
            break;

        case 2: // Activating the process
            cout << "Enter process to activate: ";
            cin >> activate;
            // Changing Process Status
            if (!processes[activate])
            {
                processes[activate] = 1;
            }
            else // If process is already activated
            {
                cout << "Proccess is already activated" << endl;
                break;
            }
            if (activate == n)
            {
                coordinator = n;
                printDetails(processes, coordinator, n);
                break;
            }
            for (i = activate + 1; i <= n; i++)
            {
                printf("Message is sent from %d to %d\n", activate, i);
                if (processes[i])
                {
                    sub_coordinator = i;
                    printf("Response is sent from %d to %d\n", i, activate);
                    flag = 1;
                }
            }
            if (flag == 1)
            {
                coordinator = sub_coordinator;
            }
            else
            {
                coordinator = activate;
            }
            printDetails(processes, coordinator, n);
        case 3:
            break;
        default:
            cout << "Enter Valid Choice" << endl
                 << endl;
        }
    } while (ch != 3);
    return 0;
}
```

14) Sender Initiated

```cpp
#include <iostream>

#include <cstdlib>

#include <ctime>

#include <chrono>

#include <thread>

#include <mutex>

#include <condition_variable>



std::mutex mtx; // Mutex to protect shared variables

std::condition_variable cv; // Condition variable to synchronize sender and receiver

bool is_data_available = false; // Flag to indicate if data is available for sending

int data = 0; // Shared variable to hold the data



void sender(int id) {

    std::srand(std::time(nullptr)); // Seed the random number generator

    int send_count = 0; // Counter for the number of data sent

    while (send_count < 10) { // Send 10 data items

        std::unique_lock<std::mutex> lck(mtx); // Acquire the mutex lock

        while (is_data_available) { // Wait for the receiver to consume the data

            cv.wait(lck);

        }

        data = std::rand() % 100; // Generate random data

        is_data_available = true; // Set the flag to indicate data is available

        std::cout << "Sender " << id << " sent data: " << data << std::endl;
```

```
        lck.unlock(); // Release the mutex lock
        cv.notify_all(); // Notify the receiver that data is available
        std::this_thread::sleep_for(std::chrono::milliseconds(100)); // Wait for a short time
        send_count++;
    }
}

void receiver(int id) {
    int receive_count = 0; // Counter for the number of data received
    while (receive_count < 10) { // Receive 10 data items
        std::unique_lock<std::mutex> lck(mtx); // Acquire the mutex lock
        while (!is_data_available) { // Wait for the sender to produce data
            cv.wait(lck);
        }
        std::cout << "Receiver " << id << " received data: " << data << std::endl;
        is_data_available = false; // Set the flag to indicate data has been consumed
        lck.unlock(); // Release the mutex lock
        cv.notify_all(); // Notify the sender that data has been consumed
        std::this_thread::sleep_for(std::chrono::milliseconds(200)); // Wait for a longer time
        receive_count++;
    }
}

int main() {
    std::thread t1(sender, 1); // Create the sender thread
    std::thread t2(receiver, 1); // Create the receiver thread
    t1.join(); // Wait for the sender thread to finish
    t2.join(); // Wait for the receiver thread to finish
```

```
    return 0;

}
```

15) Receiver Initiated

16) Adaptive/Above average algo