ASSIGNMENT 2

Name – Sarvesh Sanjay Shingare
Roll No – 144
PRN – 202101050031
Batch – B4

✷ Problem Statement :- : Implement clock synchronization alogorithms

1)Berkeley Algorithm
2)Lamport's Logical Clock

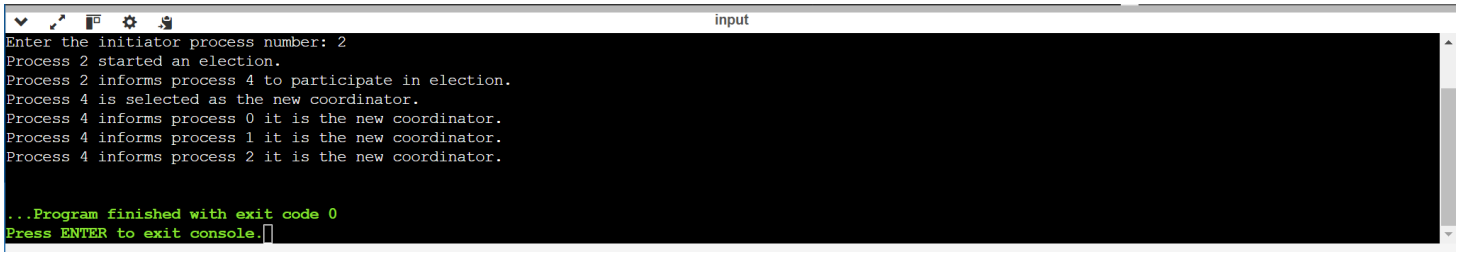
> Berkeley Algorithm

Code :-

```cpp
// Sarvesh Shingare
// 202101050031

#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// Function to handle election among processes using the Bully Algorithm
void startElection(vector<int>& processes, int initiator) {
    cout << "Process " << initiator << " started an election.\n";

    // Inform higher numbered processes
    for (int i = initiator + 1; i < processes.size(); i++) {
        if (processes[i] != -1) {  // if process is active
            cout << "Process " << initiator << " informs process " << i << " to participate in e
        }
    }

    // Find the highest numbered active process
    int coordinator = initiator;
    for (int i = initiator + 1; i < processes.size(); i++) {
        if (processes[i] != -1) {
            coordinator = i;
        }
    }

    // Declare the highest numbered process as coordinator
    cout << "Process " << coordinator << " is selected as the new coordinator.\n";
    for (int i = 0; i < processes.size(); i++) {
        if (processes[i] != -1 && i != coordinator) {
            cout << "Process " << coordinator << " informs process " << i << " it is the new coo
        }
    }
}

int main() {
    int n;  // number of processes
```

```
    cout << "Enter the number of processes: ";
    cin >> n;

    vector<int> processes(n);
    for (int i = 0; i < n; i++) {
        processes[i] = i;  // initializing processes with their IDs
    }

    int failedProcess;
    cout << "Enter the process number that failed (-1 for no failures): ";
    cin >> failedProcess;

    if (failedProcess != -1 && failedProcess < n) {
        processes[failedProcess] = -1;  // mark the process as failed
        cout << "Process " << failedProcess << " has failed.\n";
    }

    int initiator;
    cout << "Enter the initiator process number: ";
    cin >> initiator;

    // Check if initiator is valid and not failed
    if (initiator >= 0 && initiator < n && processes[initiator] != -1) {
        startElection(processes, initiator);
    } else {
        cout << "Invalid initiator or the process has failed.\n";
    }

    return 0;
}
```

Output :-



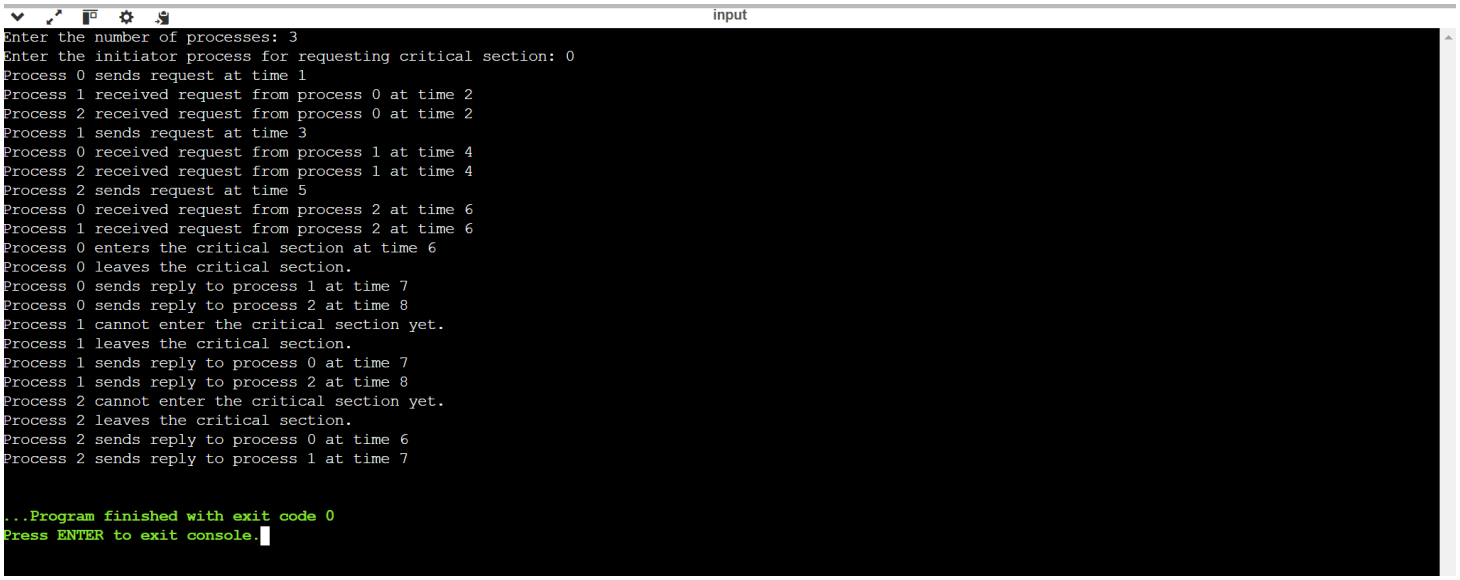
> Lamport's Logical Clock

```cpp
// sarvesh shingare

#include <iostream>
#include <queue>
#include <vector>
#include <algorithm>

using namespace std;

// Structure for request messages
struct Request {
    int processID;
    int timestamp;

    // Overload the less-than operator to prioritize by timestamp first, then by process ID
    bool operator<(const Request& r) const {
        return timestamp > r.timestamp || (timestamp == r.timestamp && processID > r.processID);
    }
};

// Simulating a process in the distributed system
class Process {
public:
    int processID;
    int clock;   // Logical clock
    priority_queue<Request> requestQueue;

    Process(int id) {
        processID = id;
        clock = 0;
    }

    // Simulates sending a request for critical section
    void sendRequest(vector<Process>& processes) {
        clock++;  // Increment clock before sending
        cout << "Process " << processID << " sends request at time " << clock << "\n";

        Request req{processID, clock};
        requestQueue.push(req);  // Add own request to its queue

        // Simulate sending the request to all other processes
```

```
    for (auto& process : processes) {
        if (process.processID != processID) {
            process.receiveRequest(req);
        }
    }
}

// Simulates receiving a request from another process
void receiveRequest(Request req) {
    clock = max(clock, req.timestamp) + 1;  // Update logical clock
    cout << "Process " << processID << " received request from process " << req.processID <<
    requestQueue.push(req);  // Add request to its queue
}

// Simulates sending a reply to a process
void sendReply(int destinationID) {
    clock++;  // Increment clock before sending
    cout << "Process " << processID << " sends reply to process " << destinationID << " at t
}

// Checks if the process can enter the critical section
bool canEnterCriticalSection() {
    if (requestQueue.empty()) return false;

    Request topRequest = requestQueue.top();
    return topRequest.processID == processID;  // Can enter if its own request is at the top
}

// Enters the critical section
void enterCriticalSection() {
    if (canEnterCriticalSection()) {
        cout << "Process " << processID << " enters the critical section at time " << clock
        requestQueue.pop();  // Remove the request after entering
    } else {
        cout << "Process " << processID << " cannot enter the critical section yet.\n";
    }
}

// Simulates replying to all other processes after leaving the critical section
void leaveCriticalSection(vector<Process>& processes) {
    cout << "Process " << processID << " leaves the critical section.\n";
    for (auto& process : processes) {
        if (process.processID != processID) {
```

```
                sendReply(process.processID);
            }
        }
    }
};

int main() {
    int n;
    cout << "Enter the number of processes: ";
    cin >> n;

    vector<Process> processes;
    for (int i = 0; i < n; ++i) {
        processes.push_back(Process(i));
    }

    int initiator;
    cout << "Enter the initiator process for requesting critical section: ";
    cin >> initiator;

    if (initiator < 0 || initiator >= n) {
        cout << "Invalid initiator process.\n";
        return 0;
    }

    // The initiator sends a request for the critical section
    processes[initiator].sendRequest(processes);

    // Simulate the other processes sending requests and replying
    for (int i = 0; i < n; ++i) {
        if (i != initiator) {
            processes[i].sendRequest(processes);
        }
    }

    // Simulate entering and leaving the critical section
    for (int i = 0; i < n; ++i) {
        processes[i].enterCriticalSection();
        processes[i].leaveCriticalSection(processes);
    }

    return 0;
}
```

Output :-


input
Enter the number of processes: 3
Enter the initiator process for requesting critical section: 0
Process 0 sends request at time 1
Process 1 received request from process 0 at time 2
Process 2 received request from process 0 at time 2
Process 1 sends request at time 3
Process 0 received request from process 1 at time 4
Process 2 received request from process 1 at time 4
Process 2 sends request at time 5
Process 0 received request from process 2 at time 6
Process 1 received request from process 2 at time 6
Process 0 enters the critical section at time 6
Process 0 leaves the critical section.
Process 0 sends reply to process 1 at time 7
Process 0 sends reply to process 2 at time 8
Process 1 cannot enter the critical section yet.
Process 1 leaves the critical section.
Process 1 sends reply to process 0 at time 7
Process 1 sends reply to process 2 at time 8
Process 2 cannot enter the critical section yet.
Process 2 leaves the critical section.
Process 2 sends reply to process 0 at time 6
Process 2 sends reply to process 1 at time 7
...Program finished with exit code 0
Press ENTER to exit console.