VIT
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

Fall Semester 2024-25

Digital Assessment 4

SLOT: L25+L26 & L47+L48

Programme Name & Branch: B. Tech CSE

Course Name & Code: BCSE303P Operating Systems Lab

1) **Given a set of dynamic disk requests with different track numbers, implement the following algorithms to calculate the total seek time:**

a) FCFS disk scheduling algorithm
b) SSTF disk scheduling algorithm
c) SCAN algorithm
d) C-SCAN algorithm
e) LOOK disk scheduling algorithm
f) C-LOOK disk scheduling algorithm

The algorithm should detect and print the efficient disk scheduling algorithm for any given input sequence.

CODE:

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>

using namespace std;

int FCFS(const vector<int>& fcfs_requests, int fcfs_head_position) {
    int fcfs_seek_time = 0;
    for (int fcfs_request : fcfs_requests) {
        fcfs_seek_time += abs(fcfs_request - fcfs_head_position);
        fcfs_head_position = fcfs_request;
    }
    return fcfs_seek_time;
}
```

```
int SSTF(vector<int> sstf_requests, int sstf_head_position) {
    int sstf_seek_time = 0;
    while (!sstf_requests.empty()) {
        auto closest_request = min_element(sstf_requests.begin(),
sstf_requests.end(),
                                           [sstf_head_position](int a, int b)
{
                                               return abs(a -
sstf_head_position) < abs(b - sstf_head_position);
                                           });
        sstf_seek_time += abs(*closest_request - sstf_head_position);
        sstf_head_position = *closest_request;
        sstf_requests.erase(closest_request);
    }
    return sstf_seek_time;
}

int SCAN(vector<int> scan_requests, int scan_head_position, int
scan_disk_size) {
    int scan_seek_time = 0;
    scan_requests.push_back(0);
    scan_requests.push_back(scan_disk_size - 1);
    sort(scan_requests.begin(), scan_requests.end());

    auto scan_position = lower_bound(scan_requests.begin(),
scan_requests.end(), scan_head_position);
    for (auto it = scan_position; it != scan_requests.end(); ++it) {
        scan_seek_time += abs(*it - scan_head_position);
        scan_head_position = *it;
    }
    for (auto it = scan_position - 1; it >= scan_requests.begin(); --it) {
        scan_seek_time += abs(*it - scan_head_position);
        scan_head_position = *it;
    }
    return scan_seek_time;
}

int C_SCAN(vector<int> cscan_requests, int cscan_head_position, int
cscan_disk_size) {
    int cscan_seek_time = 0;
    cscan_requests.push_back(0);
    cscan_requests.push_back(cscan_disk_size - 1);
    sort(cscan_requests.begin(), cscan_requests.end());

    auto cscan_position = lower_bound(cscan_requests.begin(),
cscan_requests.end(), cscan_head_position);
    for (auto it = cscan_position; it != cscan_requests.end(); ++it) {
```

```
        cscan_seek_time += abs(*it - cscan_head_position);
        cscan_head_position = *it;
    }
    cscan_seek_time += abs(cscan_disk_size - 1 - cscan_head_position);
    cscan_head_position = 0;
    for (auto it = cscan_requests.begin(); it < cscan_position; ++it) {
        cscan_seek_time += abs(*it - cscan_head_position);
        cscan_head_position = *it;
    }
    return cscan_seek_time;
}

int LOOK(vector<int> look_requests, int look_head_position) {
    int look_seek_time = 0;
    sort(look_requests.begin(), look_requests.end());

    auto look_position = lower_bound(look_requests.begin(),
look_requests.end(), look_head_position);
    for (auto it = look_position; it != look_requests.end(); ++it) {
        look_seek_time += abs(*it - look_head_position);
        look_head_position = *it;
    }
    for (auto it = look_position - 1; it >= look_requests.begin(); --it) {
        look_seek_time += abs(*it - look_head_position);
        look_head_position = *it;
    }
    return look_seek_time;
}

int C_LOOK(vector<int> clook_requests, int clook_head_position) {
    int clook_seek_time = 0;
    sort(clook_requests.begin(), clook_requests.end());

    auto clook_position = lower_bound(clook_requests.begin(),
clook_requests.end(), clook_head_position);
    for (auto it = clook_position; it != clook_requests.end(); ++it) {
        clook_seek_time += abs(*it - clook_head_position);
        clook_head_position = *it;
    }
    for (auto it = clook_requests.begin(); it < clook_position; ++it) {
        clook_seek_time += abs(*it - clook_head_position);
        clook_head_position = *it;
    }
    return clook_seek_time;
}

int main() {
    int total_requests, initial_head_position, disk_capacity;
```

```
    cout << "Enter the number of requests: ";
    cin >> total_requests;
    vector<int> input_requests(total_requests);
    cout << "Enter the requests: ";
    for (int i = 0; i < total_requests; ++i) cin >> input_requests[i];
    cout << "Enter the initial head position: ";
    cin >> initial_head_position;
    cout << "Enter the disk size: ";
    cin >> disk_capacity;

    int fcfs_seek = FCFS(input_requests, initial_head_position);
    int sstf_seek = SSTF(input_requests, initial_head_position);
    int scan_seek = SCAN(input_requests, initial_head_position,
disk_capacity);
    int c_scan_seek = C_SCAN(input_requests, initial_head_position,
disk_capacity);
    int look_seek = LOOK(input_requests, initial_head_position);
    int c_look_seek = C_LOOK(input_requests, initial_head_position);

    cout << "FCFS Seek Time: " << fcfs_seek << endl;
    cout << "SSTF Seek Time: " << sstf_seek << endl;
    cout << "SCAN Seek Time: " << scan_seek << endl;
    cout << "C-SCAN Seek Time: " << c_scan_seek << endl;
    cout << "LOOK Seek Time: " << look_seek << endl;
    cout << "C-LOOK Seek Time: " << c_look_seek << endl;

    int min_seek_time = min({fcfs_seek, sstf_seek, scan_seek, c_scan_seek,
look_seek, c_look_seek});
    cout << "Most efficient algorithm: ";
    if (min_seek_time == fcfs_seek) cout << "FCFS";
    else if (min_seek_time == sstf_seek) cout << "SSTF";
    else if (min_seek_time == scan_seek) cout << "SCAN";
    else if (min_seek_time == c_scan_seek) cout << "C-SCAN";
    else if (min_seek_time == look_seek) cout << "LOOK";
    else if (min_seek_time == c_look_seek) cout << "C-LOOK";
    cout << " with Seek Time: " << min_seek_time << endl;

    return 0;
}
```

OUTPUT:

input

```
Enter the number of requests: 8
Enter the requests: 98 183 37 122 14 124 65 67
Enter the initial head position: 53
Enter the disk size: 200
FCFS Seek Time: 640
SSTF Seek Time: 236
SCAN Seek Time: 345
C-SCAN Seek Time: 183
LOOK Seek Time: 299
C-LOOK Seek Time: 322
Most efficient algorithm: C-SCAN with Seek Time: 183


...Program finished with exit code 0
Press ENTER to exit console.
```

2) **Memory Management in Virtual Memory Systems**

Problem: Design a virtual memory system that uses paging with dynamic memory allocation. Assume you have a fixed-size page table and a backing store that stores pages of processes. When a page fault occurs, the system should fetch the page from the backing store into physical memory dynamically.

- Implement a function that handles page faults using demand paging and allocates memory dynamically when a page fault occurs.
- Discuss how dynamic memory allocation works for a page table, and how the page table entries are managed.
- How would the system handle page replacement in a least recently used (LRU) or optimal page replacement algorithm?

Challenge:

- Implement page fault handling and memory allocation in a virtual memory context.
- Discuss trade-offs in memory allocation, page replacement algorithms, and their impact on system performance.

- Consider memory fragmentation when pages are allocated and freed dynamically.

CODE:

```
#include <iostream>
#include <unordered_map>
#include <list>
#include <vector>

using namespace std;

class MemoryManager {
    int maxPageTableSize, frameSize, memoryCapacity;
    vector<int> secondaryStorage;
    unordered_map<int, int> pageDirectory;
    list<int> recentlyUsedPages;
    unordered_map<int, list<int>::iterator> pageLookup;
    vector<int> mainMemory;
```

```
public:
    MemoryManager(int maxTableSize, int frameBytes, int totalMemory,
const vector<int>& storage)
        : maxPageTableSize(maxTableSize), frameSize(frameBytes),
memoryCapacity(totalMemory), secondaryStorage(storage) {
        mainMemory.resize(memoryCapacity / frameSize, -1);
    }

    void loadPage(int virtualAddress) {
        int targetPage = virtualAddress / frameSize;
        if (pageDirectory.find(targetPage) == pageDirectory.end()) {
            cout << "Page fault for page: " << targetPage << endl;
            processPageFault(targetPage);
        } else {
            cout << "Page " << targetPage << " found in frame: " <<
pageDirectory[targetPage] << endl;
            refreshLRU(targetPage);
        }
    }

    void processPageFault(int page) {
        int freeFrame = findFreeMemoryFrame();
        if (freeFrame == -1) {
            freeFrame = performPageReplacement();
        }

        pageDirectory[page] = freeFrame;
        mainMemory[freeFrame] = secondaryStorage[page];
        cout << "Page " << page << " loaded into frame " <<
freeFrame << endl;
        refreshLRU(page);
    }

    int findFreeMemoryFrame() {
        for (int i = 0; i < mainMemory.size(); i++) {
            if (mainMemory[i] == -1) return i;
        }
        return -1;
    }

    int performPageReplacement() {
        int leastUsedPage = recentlyUsedPages.back();
        recentlyUsedPages.pop_back();
```

```
        int replacedFrame = pageDirectory[leastUsedPage];
        pageDirectory.erase(leastUsedPage);
        cout << "Page " << leastUsedPage << " removed from frame "
<< replacedFrame << endl;
        return replacedFrame;
    }

    void refreshLRU(int page) {
        if (pageLookup.find(page) != pageLookup.end()) {
            recentlyUsedPages.erase(pageLookup[page]);
        }
        recentlyUsedPages.push_front(page);
        pageLookup[page] = recentlyUsedPages.begin();
    }
};

int main() {
    int tableLimit, frameBytes, memorySize;
    cout << "Enter page table capacity: ";
    cin >> tableLimit;
    cout << "Enter size of each page frame: ";
    cin >> frameBytes;
    cout << "Enter total memory size: ";
    cin >> memorySize;

    int secondaryStorageSize;
    cout << "Enter size of secondary storage (backing store): ";
    cin >> secondaryStorageSize;
    vector<int> secondaryStorage(secondaryStorageSize);
    cout << "Enter data for each page in the secondary storage: ";
    for (int i = 0; i < secondaryStorageSize; i++) {
        cin >> secondaryStorage[i];
    }

    MemoryManager memSystem(tableLimit, frameBytes, memorySize,
secondaryStorage);

    int addressCount;
    cout << "Enter number of virtual addresses to access: ";
    cin >> addressCount;
    cout << "Enter the virtual addresses: ";
    for (int i = 0; i < addressCount; i++) {
        int address;
        cin >> address;
```

```
        memSystem.loadPage(address);
    }

    return 0;
}
```

OUTPUT: