NOTRE DAME UNIVERSITY
BANGLADESH

Operating System Lab Report

Course Code: CSE-3206
Course Title: Operating System Lab
Lab Report: Lab Task- 04, 05, 06, 07, 08

Submitted by:

Name: Istiak Alam
ID: 0692230005101005
Batch: CSE-20

Submission Date: May 29, 2025

Submitted to:

Khorshed Alam
Lecturer,
Notre Dame University Bangladesh

Table of Contents

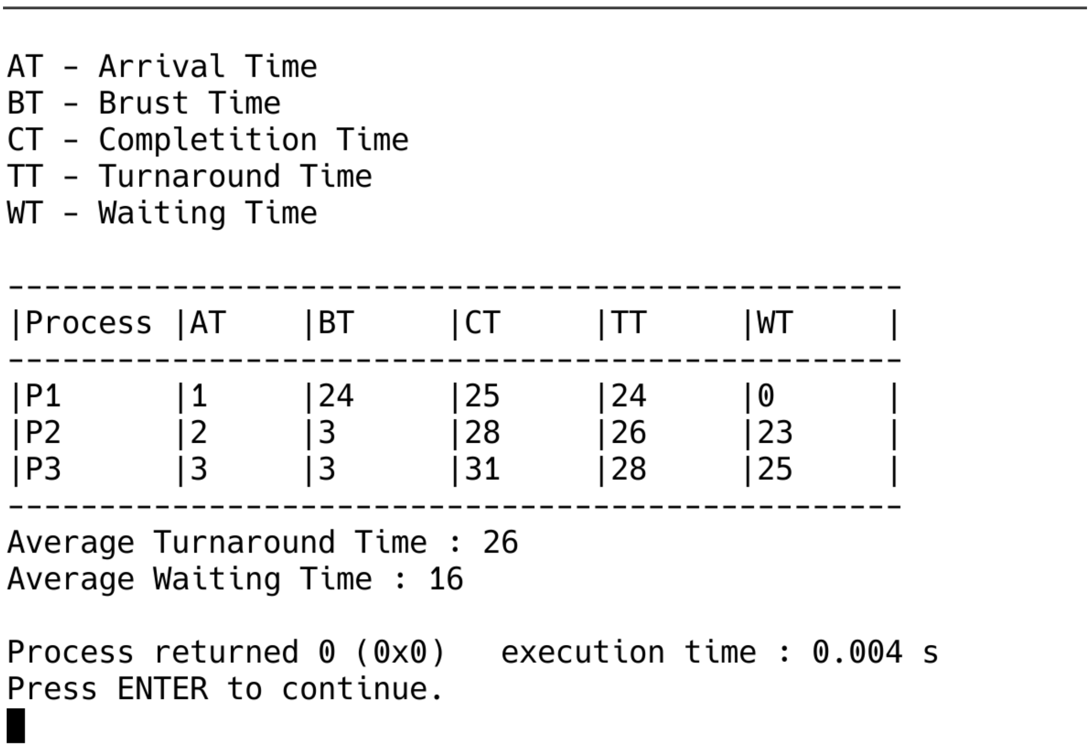
4 Lab Report-04 : CPU Scheduling

4.1 Implementation of FCFS Scheduling

- **Code :**

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

int main()
{
    //cout << "Enter Number of Process : ";
    int n=3;
    //cin >> n;
    int i, j, CT[n], TAT[n], WT[n];
    int PS[]={1,2,3};
    int AT[]={1,2,3};
    int BT[]={24,3,3};
   /* for(i=1; i<=n; i++)
    {
        cout << "Enter Arrival Time for P" << i << ": ";
        cin >> AT[i];
        cout << "Enter Burst Time for P" << i << ": ";
        cin >> BT[i];
    } */
    cout<< endl;
    for(i=0; i<n-1; i++)
    {
        for(j=0; j<n-i-1; j++)
        {
            if(AT[j] > AT[j+1])
            {
                swap(AT[j],AT[j+1]);
            }
        }
    }
    CT[0]=AT[0]+BT[0];
    for(i=1; i<n; i++)
    {
        CT[i]=CT[i-1]+BT[i];
    }
    int Total_T, Total_W;
    for(i=0; i<n; i++)
    {
        TAT[i]=CT[i]-AT[i];
        WT[i]=TAT[i]-BT[i];

        Total_T=Total_T+TAT[i];
        Total_W=Total_W+WT[i];
    }
    cout << "AT - Arrival Time" << endl;
    cout << "BT - Brust Time" << endl;
    cout << "CT - Completition Time" << endl;
    cout << "TT - Turnaround Time" << endl;
    cout << "WT - Waiting Time" << endl<<endl;
    cout << "-------------------------------------------------"<<endl;
    cout << "|Process |AT\t|BT\t|CT\t|TT\t|WT\t|"<<endl;
    cout << "-------------------------------------------------"<<endl;

    for (i=0; i<n; i++)
    {
        cout << "|P"<<PS[i]<<"\t |"<<AT[i]<<"\t|"<<BT[i]<<"\t|"
        <<CT[i]<<"\t|"<<TAT[i]<<"\t|"<<WT[i]<<"\t|"<<endl;
    }
    cout << "-------------------------------------------------"<<endl;
    cout << "Average Turnaround Time : " <<Total_T/n<<endl;
    cout << "Average Waiting Time : " <<Total_W/n<<endl;
}
```

- **Input / Output :**



```
AT - Arrival Time
BT - Brust Time
CT - Completition Time
TT - Turnaround Time
WT - Waiting Time

-------------------------------------------------
|Process |AT    |BT     |CT     |TT     |WT     |
-------------------------------------------------
|P1      |1     |24     |25     |24     |0      |
|P2      |2     |3      |28     |26     |23     |
|P3      |3     |3      |31     |28     |25     |
-------------------------------------------------
Average Turnaround Time : 26
Average Waiting Time : 16

Process returned 0 (0x0)   execution time : 0.004 s
Press ENTER to continue.
```

Figure 1: Output

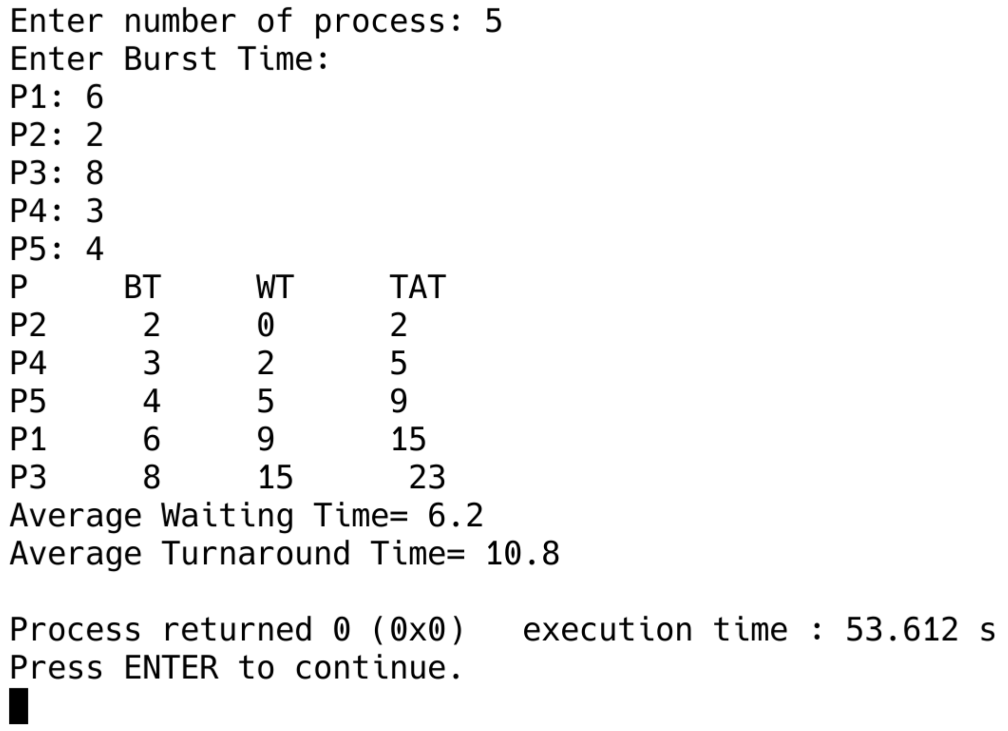
- **Description :**

4.2 Implementation of Shortest Job First (SJF)

- **Code :**

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

int main() {
    int A[100][4];
    int i, j, n, total = 0, index, temp;
    float avg_wt, avg_tat;
    cout << "Enter number of process: ";
    cin >> n;
    cout << "Enter Burst Time:" << endl;

    // User Input Burst Time and alloting Process Id.
    for (i = 0; i < n; i++) {
        cout << "P" << i + 1 << ": ";
        cin >> A[i][1];
        A[i][0] = i + 1;
    }

    // Sorting process according to their Burst Time.
    for (i = 0; i < n; i++) {
        index = i;
        for (j = i + 1; j < n; j++)
            if (A[j][1] < A[index][1])
                index = j;
        temp = A[i][1];
        A[i][1] = A[index][1];
        A[index][1] = temp;

        temp = A[i][0];
        A[i][0] = A[index][0];
        A[index][0] = temp;
    }

    A[0][2] = 0;
    // Calculation of Waiting Times
    for (i = 1; i < n; i++) {
        A[i][2] = 0;
        for (j = 0; j < i; j++)
            A[i][2] += A[j][1];
        total += A[i][2];
    }

    avg_wt = (float)total / n;
    total = 0;
    cout << "P     BT     WT     TAT" << endl;

    // Calculation of Turn Around Time and printing the
    // data.
    for (i = 0; i < n; i++) {
        A[i][3] = A[i][1] + A[i][2];
        total += A[i][3];
        cout << "P" << A[i][0] << "     " << A[i][1] << "     " << A[i][2] << "
    }

    avg_tat = (float)total / n;
    cout << "Average Waiting Time= " << avg_wt << endl;
    cout << "Average Turnaround Time= " << avg_tat << endl;
}
```

- **Input / Output :**



```
Enter number of process: 5
Enter Burst Time:
P1: 6
P2: 2
P3: 8
P4: 3
P5: 4
P     BT     WT     TAT
P2     2     0      2
P4     3     2      5
P5     4     5      9
P1     6     9      15
P3     8     15      23
Average Waiting Time= 6.2
Average Turnaround Time= 10.8

Process returned 0 (0x0)   execution time : 53.612 s
Press ENTER to continue.
```

Figure 2: Output

- **Description :**

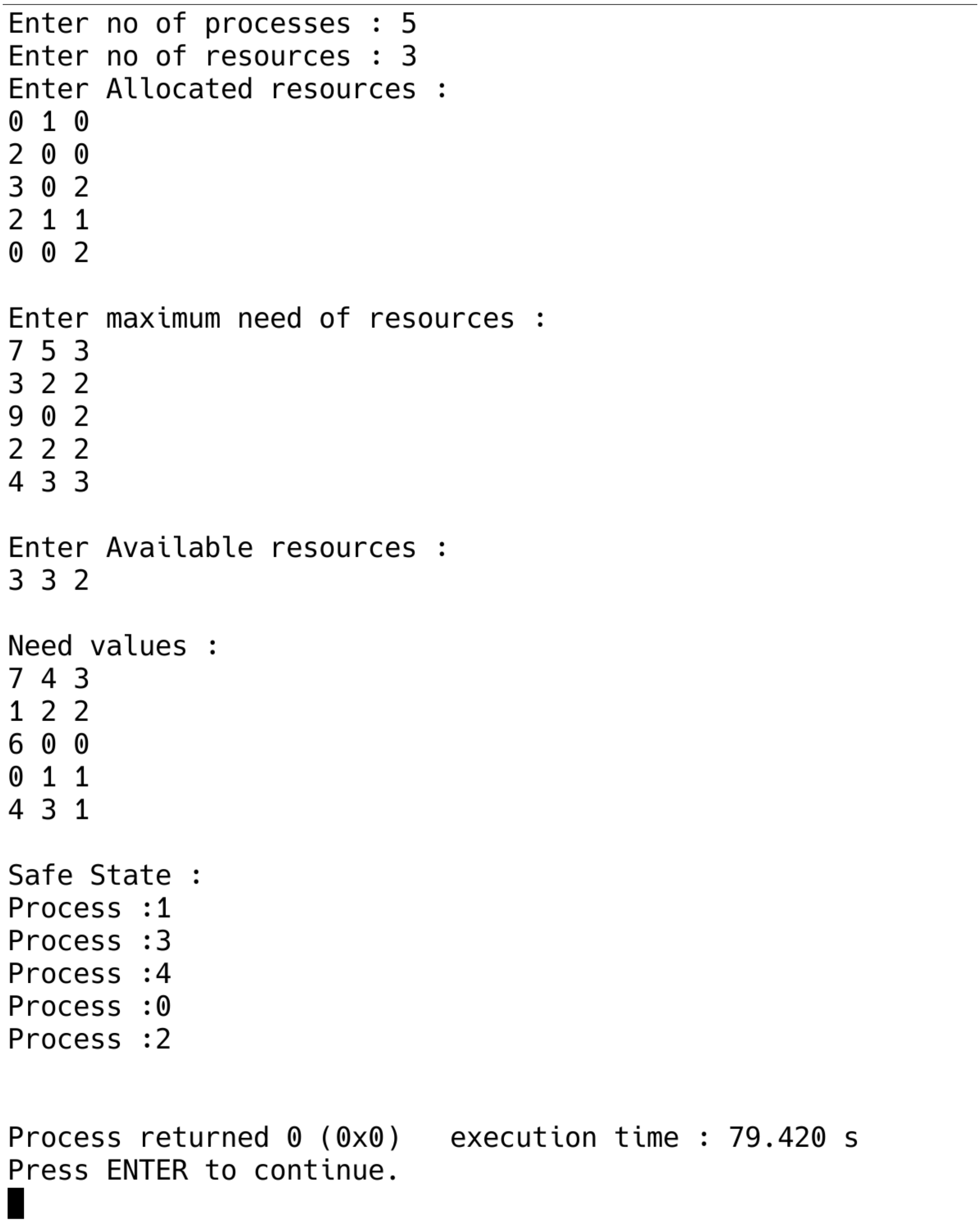
5 Lab Report-05 : Deadlock - Bankers Algorithm

5.1 Implementation of Bankers Algorithm

- **Code :**

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

int main()
{
    int p, r ;
    cout << "Enter no of processes : ";
    cin >> p;
    cout << "Enter no of resources : ";
    cin >> r;

    int al[p][r], mx[p][r], av[r], need[p][r];

    //input allocation
    cout << "Enter Allocated resources : " << endl;
    for (int i=0; i<p; i++)
    {
        for (int j=0; j<r; j++)
        {
            cin >> al[i][j];
        }
    }
    cout << endl;

    //input max instances
    cout << "Enter maximum need of resources : " << endl;
    for (int i=0; i<p; i++)
    {
        for (int j=0; j<r; j++)
        {
            cin >> mx[i][j];
        }
    }
    cout << endl;

    //input available resources
    cout << "Enter Available resources : " << endl;
    for (int i=0; i<r; i++)
    {
        cin >> av[i];
    }
    cout << endl;

    // Print need values
    cout << "Need values : " << endl;
    for (int i=0; i<p; i++)
    {
        for (int j=0; j<r; j++)
        {
            //need = max - allocation
            need[i][j] = mx[i][j] - al[i][j];
            cout << need[i][j] << " ";
        }
        cout << endl;
    }
    cout << endl;

    int x = 1;
    int y = 0;
    cout << "Safe State : " << endl;
    while (x != 0)
    {
        for (int i=0; i<p; i++)
        {
        int z = 0;
        for (int j=0; j<r; j++)
        {
            if (need[i][j] <= av[j] && (need[i][0] != -1))
            {
                z++;  // counting process
            }
        }
            if (z == r) // all resourse satisfied
            {
                for (int k=0; k<r; k++)
                {
                    av[k] += al[i][k];
                }
                cout << "Process :" << i << endl;
                need[i][0] = -1;
                y++;    // counting if process done
            }
        }

        if (y == p)  x = 0;
    }
    cout << endl;
    return 0;
}
```

- **Input / Output :**



```
Enter no of processes : 5
Enter no of resources : 3
Enter Allocated resources :
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Enter maximum need of resources :
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Enter Available resources :
3 3 2

Need values :
7 4 3
1 2 2
6 0 0
0 1 1
4 3 1

Safe State :
Process :1
Process :3
Process :4
Process :0
Process :2


Process returned 0 (0x0)   execution time : 79.420 s
Press ENTER to continue.
█
```

Figure 3: Output

- **Description :**

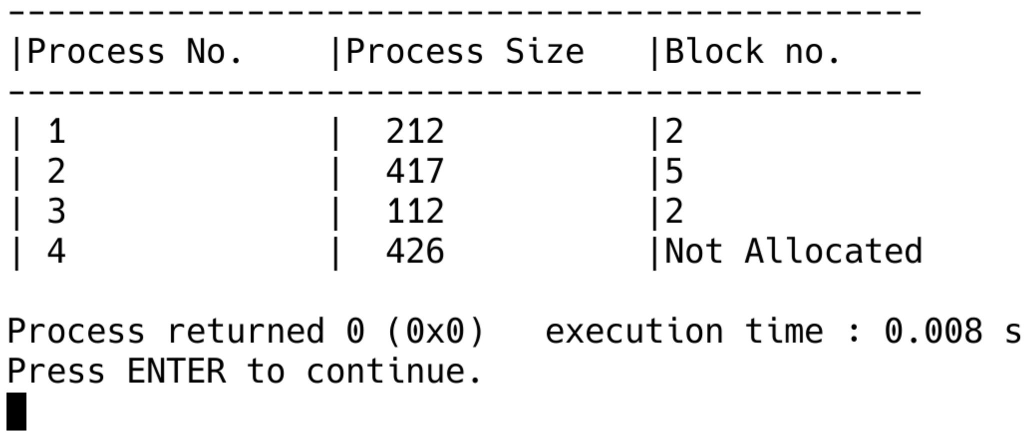
6 Lab Report-06 : Dynamic Storage-Allocation Problem

6.1 Dynamic Storage-Allocation using FirstFit

- **Code :**

```
//First - Fit algorithm
#include<bits/stdc++.h>
using namespace std;

void firstFit(int blockSize[], int m,
              int processSize[], int n){
    int allocation[n];
    memset(allocation, -1, sizeof(allocation));
    for (int i = 0; i < n; i++){
        for (int j = 0; j < m; j++){
            if (blockSize[j] >= processSize[i]){
                // allocate block j to p[i] process
                allocation[i] = j;

                // Reduce available memory in this block.
                blockSize[j] -= processSize[i];
                break;
            }   }    }
    cout << "\n----------------------------------------------"<<endl;
    cout << "|Process No.\t|Process Size\t|Block no.\n";
    cout << "----------------------------------------------"<<endl;
    for (int i = 0; i < n; i++){
        cout << "| " << i+1 << "\t\t|   "
             << processSize[i] << "\t\t|";
        if (allocation[i] != -1)
            cout << allocation[i] + 1<<"  ";
        else
            cout << "Not Allocated";
        cout << endl;
    }
}
int main()
{
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);

    firstFit(blockSize, m, processSize, n);
    return 0 ;
}
```

- **Input / Output :**



```
----------------------------------------------
|Process No.    |Process Size   |Block no.
----------------------------------------------
| 1             |  212          |2
| 2             |  417          |5
| 3             |  112          |2
| 4             |  426          |Not Allocated

Process returned 0 (0x0)   execution time : 0.008 s
Press ENTER to continue.
```

Figure 4: Output

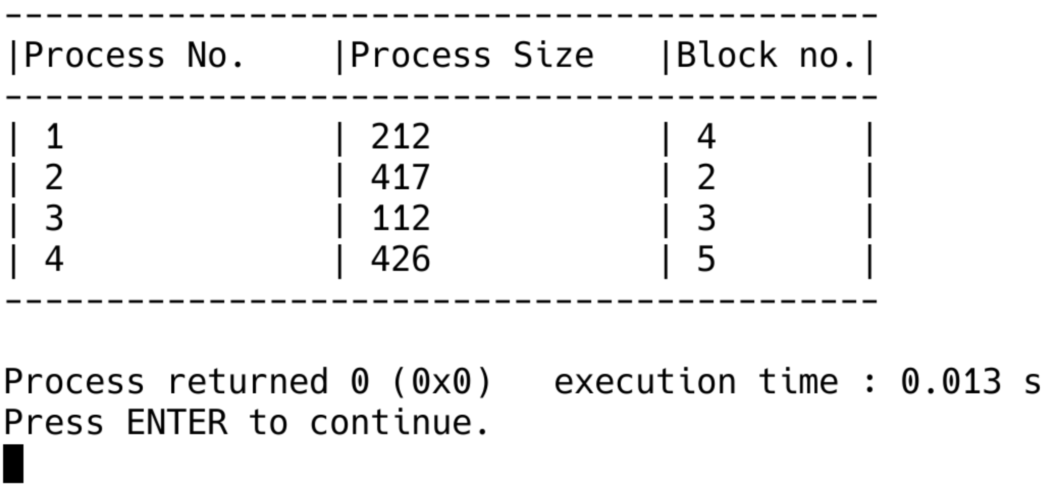
- **Description :**

6.2 Dynamic Storage-Allocation using BestFit

- **Code :**

```
// C++ implementation of Best - Fit algorithm
#include<iostream>
using namespace std;

void bestFit(int blockSize[], int m, int processSize[], int n){
    int allocation[n];
    for (int i = 0; i < n; i++)
        allocation[i] = -1;

    for (int i = 0; i < n; i++){
        int bestIdx = -1;
        for (int j = 0; j < m; j++){
            if (blockSize[j] >= processSize[i]){
                if (bestIdx == -1)
                    bestIdx = j;
                else if (blockSize[bestIdx] > blockSize[j])
                    bestIdx = j;
            }
        }
        if (bestIdx != -1){
            allocation[i] = bestIdx;

            blockSize[bestIdx] -= processSize[i];
        }
    }
    cout << "\n------------------------------------------"<<endl;
    cout << "|Process No.\t|Process Size\t|Block no.|\n";
    cout << "------------------------------------------"<<endl;
    for (int i = 0; i < n; i++){
        cout << "| " << i+1 << "\t\t| " << processSize[i] << "\t\t| ";
        if (allocation[i] != -1)
            cout << allocation[i] + 1<<"\t  |";
        else
            cout << "Not Allocated";
        cout << endl;
    }
}
int main(){
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);

    bestFit(blockSize, m, processSize, n);
    return 0 ;
}
```

- **Input / Output :**



```
-------------------------------------------
|Process No.    |Process Size   |Block no.|
-------------------------------------------
| 1             | 212           | 4       |
| 2             | 417           | 2       |
| 3             | 112           | 3       |
| 4             | 426           | 5       |
-------------------------------------------

Process returned 0 (0x0)   execution time : 0.013 s
Press ENTER to continue.
█
```

Figure 5: Output

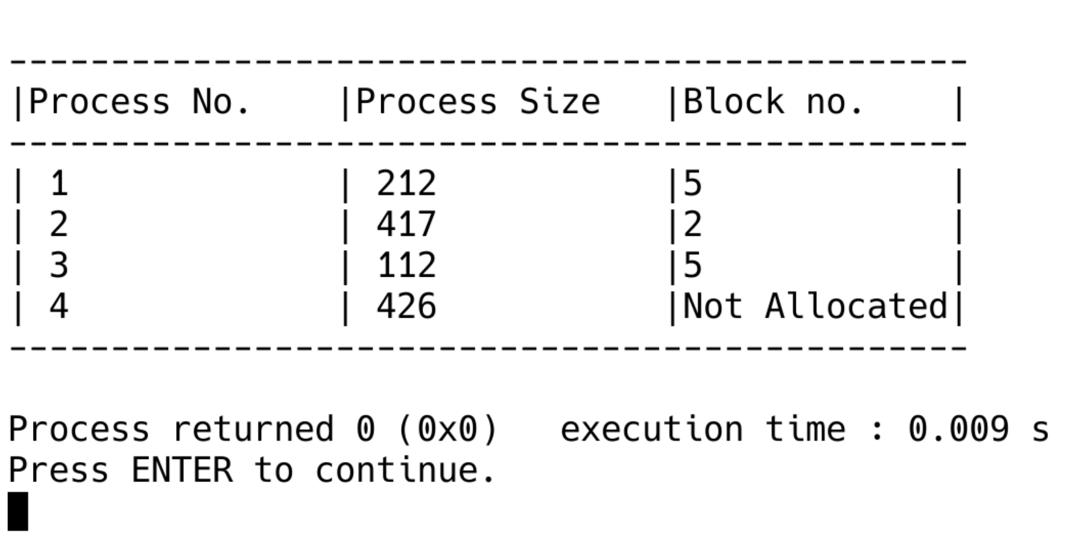
- **Description :**

6.3 Dynamic Storage-Allocation using WorstFit

- **Code :**

```
// C++ implementation of worst - Fit algorithm
#include<bits/stdc++.h>
using namespace std;

void worstFit(int blockSize[], int m, int processSize[],int n){
    int allocation[n];
    memset(allocation, -1, sizeof(allocation));

    for (int i=0; i<n; i++){
        int wstIdx = -1;
        for (int j=0; j<m; j++){
            if (blockSize[j] >= processSize[i]){
                if (wstIdx == -1)
                    wstIdx = j;
                else if (blockSize[wstIdx] < blockSize[j])
                    wstIdx = j;
            }
        }

        if (wstIdx != -1){
            allocation[i] = wstIdx;
            blockSize[wstIdx] -= processSize[i];
        }
    }
    cout << "\n-----------------------------------------------"<<endl;
    cout << "|Process No.\t|Process Size\t|Block no.    |\n";
    cout << "-----------------------------------------------"<<endl;
    for (int i = 0; i < n; i++){
        cout << "| " << i+1 << "\t\t| " << processSize[i] << "\t\t|";
        if (allocation[i] != -1)
            cout << allocation[i] + 1<<"\t      |";
        else
            cout << "Not Allocated|";
        cout << endl;
    }
    cout << "-----------------------------------------------"<<endl;
}
int main(){
    int blockSize[] = {100, 500, 200, 300, 600};
    int processSize[] = {212, 417, 112, 426};
    int m = sizeof(blockSize)/sizeof(blockSize[0]);
    int n = sizeof(processSize)/sizeof(processSize[0]);

    worstFit(blockSize, m, processSize, n);

    return 0 ;
}
```

- **Input / Output :**



```
-----------------------------------------------
|Process No.    |Process Size   |Block no.    |
-----------------------------------------------
| 1             | 212           |5            |
| 2             | 417           |2            |
| 3             | 112           |5            |
| 4             | 426           |Not Allocated|
-----------------------------------------------

Process returned 0 (0x0)   execution time : 0.009 s
Press ENTER to continue.
```

Figure 6: Output

- **Description :**

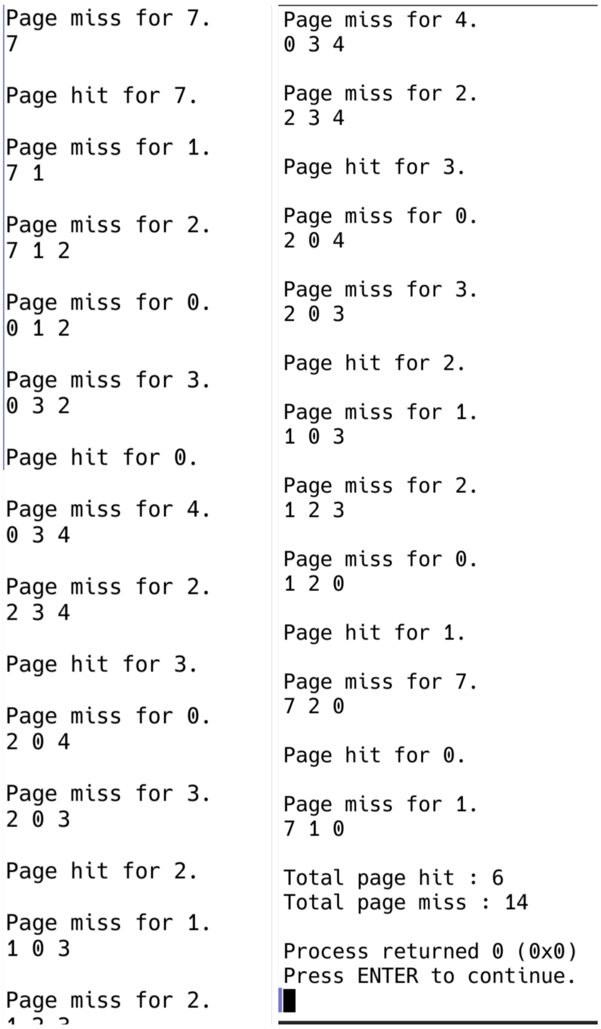
7 Lab Report-07 : Page Replacement Algorithms

7.1 Implementation of Page Replacement using FIFO Algorithm

- **Code :**

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

int main(){
    int process[] = {7,7,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1};
    int length = sizeof(process) / sizeof(process[0]);
    vector<int> main_mem;
    int max_main_mem = 3, start = 0;
    int hit = 0, miss = 0;
    for(int i=0; i<length; i++){
        auto fnd = find(main_mem.begin(), main_mem.end(), process[i]);
        if(main_mem.size() < 3){
            if(fnd != main_mem.end()){
                cout << "Page hit for " << process[i] << ".\n\n";
                hit++;
            }
            else{
                main_mem.push_back(process[i]);
                cout << "Page miss for " << process[i] << ".\n";
                for(auto it: main_mem) cout << it << " ";
                cout << "\n\n";
                miss++;
            }
            continue;
        }
        if(fnd != main_mem.end()) {
            cout << "Page hit for " << process[i] << ".\n\n";
            hit++;
        }
        else{
            cout << "Page miss for " << process[i] << ".\n";
            main_mem[start] = process[i];
            miss++;
            start++;

            for(auto it: main_mem) cout << it << " ";
            if(start == max_main_mem) start = 0;
            cout << "\n\n";
        }
    }
    cout << "Total page hit : " << hit << endl;
    cout << "Total page miss : " << miss << endl;
    return 0;
}
```

- **Input / Output :**



```
Page miss for 7.
7

Page hit for 7.

Page miss for 1.
7 1

Page miss for 2.
7 1 2

Page miss for 0.
0 1 2

Page miss for 3.
0 3 2

Page hit for 0.

Page miss for 4.
0 3 4

Page miss for 2.
2 3 4

Page hit for 3.

Page miss for 0.
2 0 4

Page miss for 3.
2 0 3

Page hit for 2.

Page miss for 1.
1 0 3

Page miss for 2.
```

```
Page miss for 4.
0 3 4

Page miss for 2.
2 3 4

Page hit for 3.

Page miss for 0.
2 0 4

Page miss for 3.
2 0 3

Page hit for 2.

Page miss for 1.
1 0 3

Page miss for 2.
1 2 3

Page miss for 0.
1 2 0

Page hit for 1.

Page miss for 7.
7 2 0

Page hit for 0.

Page miss for 1.
7 1 0

Total page hit : 6
Total page miss : 14

Process returned 0 (0x0)
Press ENTER to continue.
```

Figure 7: Output

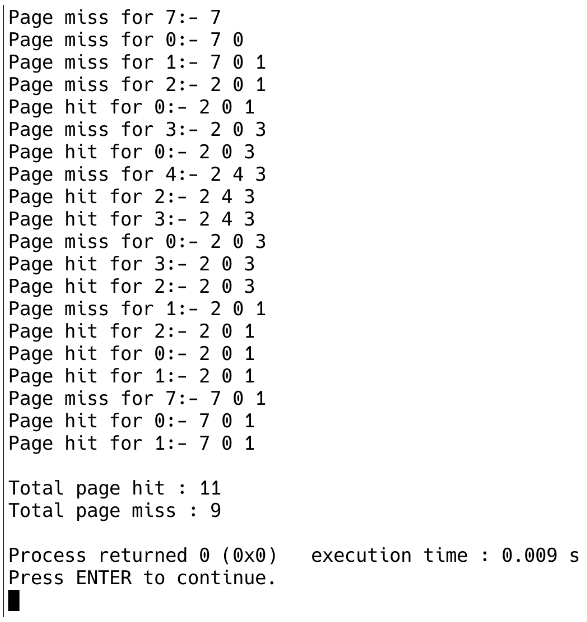
- **Description :**

7.2 Implementation of Optimal Algorithm

- **Code :**

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

void print(vector<int> main_mem){
    for(auto x: main_mem)
        cout << x << " ";
    cout << "\n";
}
int victim(int process[], int length, int start,
vector<int> main_mem, int max_mem){
    int page_index = 0;
    int distance = 0;
    for(int i = 0; i<max_mem; i++){
        bool found = false;
        for(int j = start+1; j<length; j++){
            if(main_mem[i] == process[j]){
                found = true;

                int current_distance = j - start;
                if(distance < current_distance)
                {
                    distance = current_distance;
                    page_index = i;
                }
                break;
            }
        }
        if(found == false) return i;
    }
    return page_index;
}
int main(){
    int process[] = {7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1};
    int length = sizeof(process) / sizeof(process[0]);

    vector<int> main_mem;
    int max_main_mem = 3, start = 0;
    int hit = 0, miss = 0;

    for(int i=0; i<length; i++){
        auto fnd = find(main_mem.begin(), main_mem.end(), process[i]);
        if(main_mem.size() < 3){
            if(fnd != main_mem.end()){
                cout << "Page hit for " << process[i] << ":- ";
                print(main_mem);
                hit++;
            }
            else{
                main_mem.push_back(process[i]);
                cout << "Page miss for " << process[i] << ":- " ;
                print(main_mem);
                miss++;
            }
            continue;
        }
        else{
            if(fnd != main_mem.end( )){
                cout << "Page hit for " << process[i] << ":- ";
                print(main_mem);
                hit++;
            }
            else{
                cout << "Page miss for " << process[i] << ":- ";
                miss++;

    int replace  = victim(process, length, i, main_mem, max_main_mem);
                main_mem[replace] = process[i];
                print(main_mem);
            }
        }
    }
    cout << "\nTotal page hit : " << hit << endl;
    cout << "Total page miss : " << miss << endl;
    return 0;
}
```

- **Input / Output :**



```
Page miss for 7:- 7
Page miss for 0:- 7 0
Page miss for 1:- 7 0 1
Page miss for 2:- 2 0 1
Page hit for 0:- 2 0 1
Page miss for 3:- 2 0 3
Page hit for 0:- 2 0 3
Page miss for 4:- 2 4 3
Page hit for 2:- 2 4 3
Page hit for 3:- 2 4 3
Page miss for 0:- 2 0 3
Page hit for 3:- 2 0 3
Page hit for 2:- 2 0 3
Page miss for 1:- 2 0 1
Page hit for 2:- 2 0 1
Page hit for 0:- 2 0 1
Page hit for 1:- 2 0 1
Page miss for 7:- 7 0 1
Page hit for 0:- 7 0 1
Page hit for 1:- 7 0 1

Total page hit : 11
Total page miss : 9

Process returned 0 (0x0)   execution time : 0.009 s
Press ENTER to continue.
```

Figure 8: Output

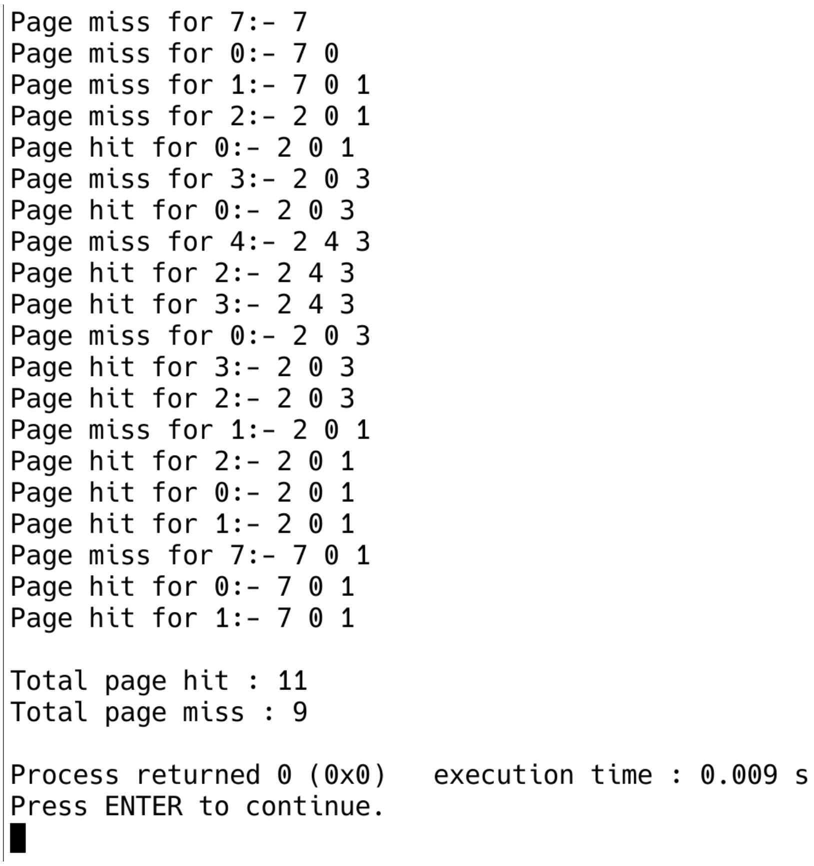
- **Description :**

7.3 Page Replacement using LRU

- **Code :**

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

void print(const list<int>& main_mem)
{
    for(auto x: main_mem)
        cout << x << " ";
    cout << "\n";
}

int main()
{
    int process[] = {7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1};
    int length = sizeof(process) / sizeof(process[0]);

    int max_main_mem = 3;
    int hit = 0, miss = 0;

    list<int> main_mem;
    unordered_map<int, list<int>::iterator> page_table;

    for(int i=0; i<length; i++)
    {
        int page = process[i];

        // Page hit: present in memory (LRU cache)
        if(page_table.find(page) != page_table.end())
        {
            // Move this page to front (most recently used)
            main_mem.erase(page_table[page]);
            main_mem.push_front(page);
            page_table[page] = main_mem.begin();

            cout << "Page hit for " << page << ":- ";
            print(main_mem);
            hit++;
        }
        else
        {
            // Page miss
            miss++;
            cout << "Page miss for " << page << ":- ";

            // If memory is full, remove least recently used (from back)
            if((int)main_mem.size() == max_main_mem)
            {
                int lru_page = main_mem.back();
                main_mem.pop_back();
                page_table.erase(lru_page);
            }
            // Insert the new page at front (most recent)
            main_mem.push_front(page);
            page_table[page] = main_mem.begin();

            print(main_mem);
        }
    }
    cout << "\nTotal page hit : " << hit << endl;
    cout << "Total page miss : " << miss << endl;
    return 0;
}
```

- **Input / Output :**



```
Page miss for 7:- 7
Page miss for 0:- 7 0
Page miss for 1:- 7 0 1
Page miss for 2:- 2 0 1
Page hit for 0:- 2 0 1
Page miss for 3:- 2 0 3
Page hit for 0:- 2 0 3
Page miss for 4:- 2 4 3
Page hit for 2:- 2 4 3
Page hit for 3:- 2 4 3
Page miss for 0:- 2 0 3
Page hit for 3:- 2 0 3
Page hit for 2:- 2 0 3
Page miss for 1:- 2 0 1
Page hit for 2:- 2 0 1
Page hit for 0:- 2 0 1
Page hit for 1:- 2 0 1
Page miss for 7:- 7 0 1
Page hit for 0:- 7 0 1
Page hit for 1:- 7 0 1

Total page hit : 11
Total page miss : 9

Process returned 0 (0x0)   execution time : 0.009 s
Press ENTER to continue.
```

Figure 9: Output

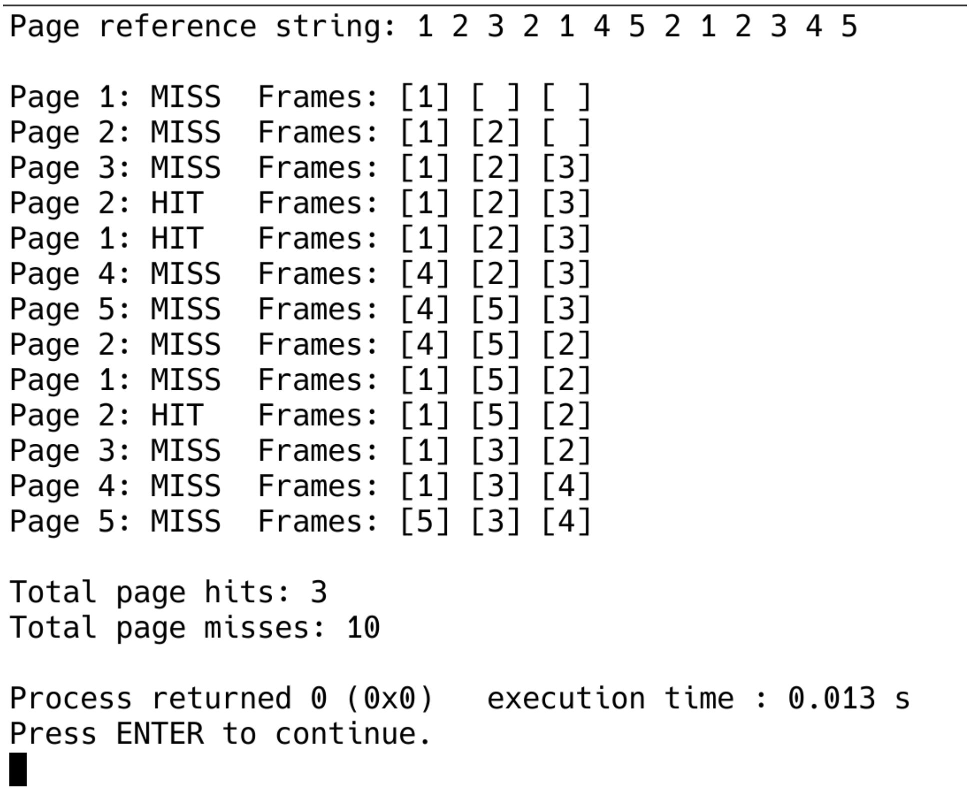
- **Description :**

7.4 Implementation of Page fitting (hit/miss)

- **Code :**

```
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;

void printFrames(const vector<int>& frames) {
    cout << "Frames: ";
    for (int page : frames) {
        if (page == -1)
            cout << "[ ] ";
        else
            cout << "[" << page << "] ";
    }
    cout << "\n";
}
int main() {
    int frames_count = 3;
    int pages[] = {1,2,3,2,1,4,5,2,1,2,3,4,5};
    int n = sizeof(pages) / sizeof(pages[0]);
    vector<int> frames(frames_count, -1); // -1 means empty
    queue<int> fifo_order;
    int hits = 0, misses = 0;

    cout << "Page reference string: ";
    for (int i = 0; i < n; ++i)
        cout << pages[i] << " ";
    cout << "\n\n";

    for (int i = 0; i < n; ++i) {
        int page = pages[i];
        // Check if the page is already in a frame
        auto it = find(frames.begin(), frames.end(), page);
        if (it != frames.end()) {
            cout << "Page " << page << ": HIT   ";
            hits++;
        } else {
            cout << "Page " << page << ": MISS  ";
            misses++;
            // Find an empty frame if available
            auto empty = find(frames.begin(), frames.end(), -1);
            if (empty != frames.end()) {
                *empty = page;
                fifo_order.push(distance(frames.begin(), empty));
            } else {
                // Replace the oldest page (FIFO order)
                int idx = fifo_order.front(); fifo_order.pop();
                frames[idx] = page;
                fifo_order.push(idx);
            }
        }
        printFrames(frames);
    }
    cout << "\nTotal page hits: " << hits << endl;
    cout << "Total page misses: " << misses << endl;
    return 0;
}
```

- **Input / Output :**



```
Page reference string: 1 2 3 2 1 4 5 2 1 2 3 4 5

Page 1: MISS  Frames: [1] [ ] [ ]
Page 2: MISS  Frames: [1] [2] [ ]
Page 3: MISS  Frames: [1] [2] [3]
Page 2: HIT   Frames: [1] [2] [3]
Page 1: HIT   Frames: [1] [2] [3]
Page 4: MISS  Frames: [4] [2] [3]
Page 5: MISS  Frames: [4] [5] [3]
Page 2: MISS  Frames: [4] [5] [2]
Page 1: MISS  Frames: [1] [5] [2]
Page 2: HIT   Frames: [1] [5] [2]
Page 3: MISS  Frames: [1] [3] [2]
Page 4: MISS  Frames: [1] [3] [4]
Page 5: MISS  Frames: [5] [3] [4]

Total page hits: 3
Total page misses: 10

Process returned 0 (0x0)   execution time : 0.013 s
Press ENTER to continue.
```

Figure 10: Output

- **Description :**

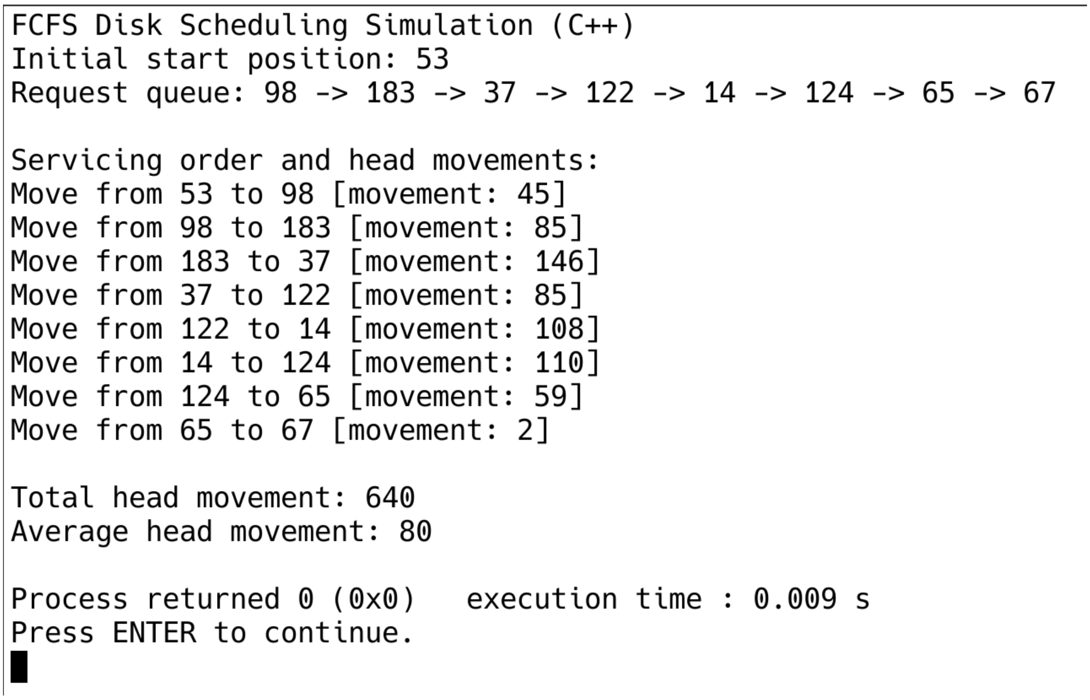
8 Lab Report-08 : Disk Scheduling Algorithms

8.1 Disk scheduling using FCFS Algorithm

- **Code :**

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

int main() {
    vector<int> requests = {98, 183, 37, 122, 14, 124, 65, 67};
    int start = 53;

    cout << "FCFS Disk Scheduling Simulation (C++)" << endl;
    cout << "Initial start position: " << start << endl;
    cout << "Request queue: ";
    for (size_t i = 0; i < requests.size(); ++i)
        cout << requests[i] << (i < requests.size()-1 ? " -> " : "");
    cout << endl << endl;

    int total_movement = 0;
    int current = start;

    cout << "Servicing order and head movements:" << endl;
    for (size_t i = 0; i < requests.size(); ++i) {
        int move = abs(requests[i] - current);
        cout << "Move from " << current << " to " << requests[i]
             << " [movement: " << move << "]" << endl;
        total_movement += move;
        current = requests[i];
    }

    double average_movement = (double)total_movement / requests.size();

    cout << "\nTotal head movement: " << total_movement << endl;
    cout << "Average head movement: " << average_movement << endl;

    return 0;
}
```

- **Input / Output :**



```
FCFS Disk Scheduling Simulation (C++)
Initial start position: 53
Request queue: 98 -> 183 -> 37 -> 122 -> 14 -> 124 -> 65 -> 67

Servicing order and head movements:
Move from 53 to 98 [movement: 45]
Move from 98 to 183 [movement: 85]
Move from 183 to 37 [movement: 146]
Move from 37 to 122 [movement: 85]
Move from 122 to 14 [movement: 108]
Move from 14 to 124 [movement: 110]
Move from 124 to 65 [movement: 59]
Move from 65 to 67 [movement: 2]

Total head movement: 640
Average head movement: 80

Process returned 0 (0x0)   execution time : 0.009 s
Press ENTER to continue.
```

Figure 11: Output

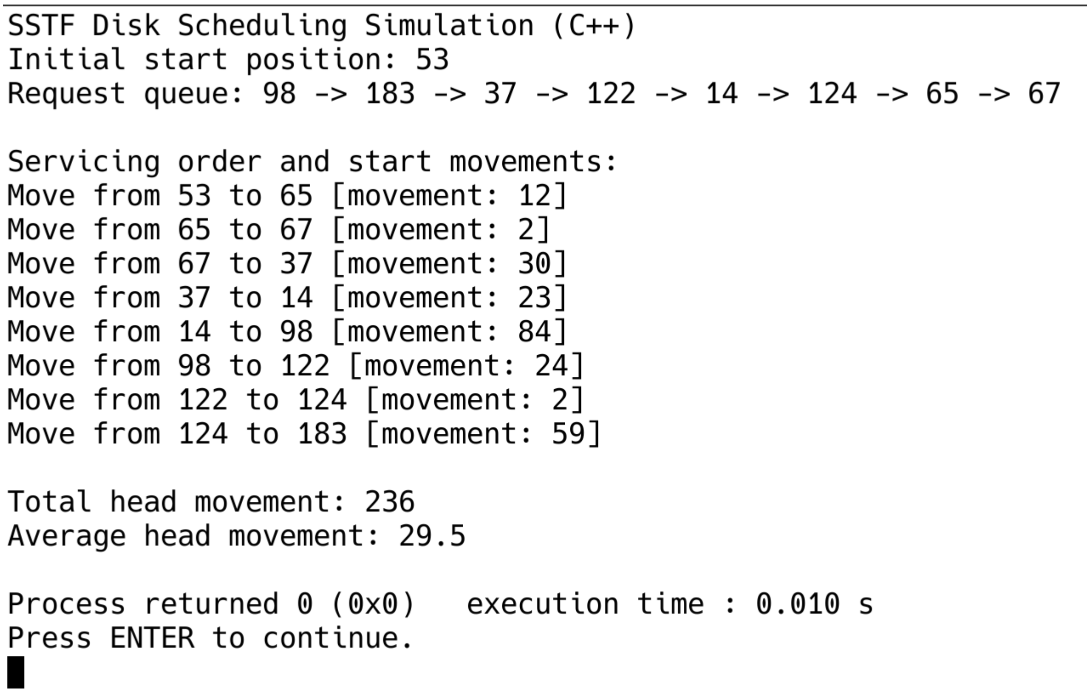
- **Description :**

8.2 Disk scheduling using SSTF Algorithm

- **Code :**

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <climits>
using namespace std;

int main() {
    vector<int> requests = {98, 183, 37, 122, 14, 124, 65, 67};
    vector<bool> visited(requests.size(), false);
    int start = 53;

    cout << "SSTF Disk Scheduling Simulation (C++)" << endl;
    cout << "Initial start position: " << start << endl;
    cout << "Request queue: ";
    for (size_t i = 0; i < requests.size(); ++i)
        cout << requests[i] << (i < requests.size()-1 ? " -> " : "");
    cout << endl << endl;

    int total_movement = 0;
    int current = start;

    cout << "Servicing order and start movements:" << endl;
    for (size_t done = 0; done < requests.size(); ++done) {
        int min_dist = INT_MAX, idx = -1;
        // Find unserviced request with minimum distance to current head
        for (size_t i = 0; i < requests.size(); ++i) {
            if (!visited[i]) {
                int dist = abs(current - requests[i]);
                if (dist < min_dist) {
                    min_dist = dist;
                    idx = i;
                }
            }
        }
        // Service this request
        cout << "Move from " << current << " to " << requests[idx]
             << " [movement: " << min_dist << "]" << endl;
        total_movement += min_dist;
        current = requests[idx];
        visited[idx] = true;
    }
    double average_movement = (double)total_movement / requests.size();

    cout << "\nTotal head movement: " << total_movement << endl;
    cout << "Average head movement: " << average_movement << endl;
    return 0;
}
```

- **Input / Output :**



```
SSTF Disk Scheduling Simulation (C++)
Initial start position: 53
Request queue: 98 -> 183 -> 37 -> 122 -> 14 -> 124 -> 65 -> 67

Servicing order and start movements:
Move from 53 to 65 [movement: 12]
Move from 65 to 67 [movement: 2]
Move from 67 to 37 [movement: 30]
Move from 37 to 14 [movement: 23]
Move from 14 to 98 [movement: 84]
Move from 98 to 122 [movement: 24]
Move from 122 to 124 [movement: 2]
Move from 124 to 183 [movement: 59]

Total head movement: 236
Average head movement: 29.5

Process returned 0 (0x0)   execution time : 0.010 s
Press ENTER to continue.
```

Figure 12: Output

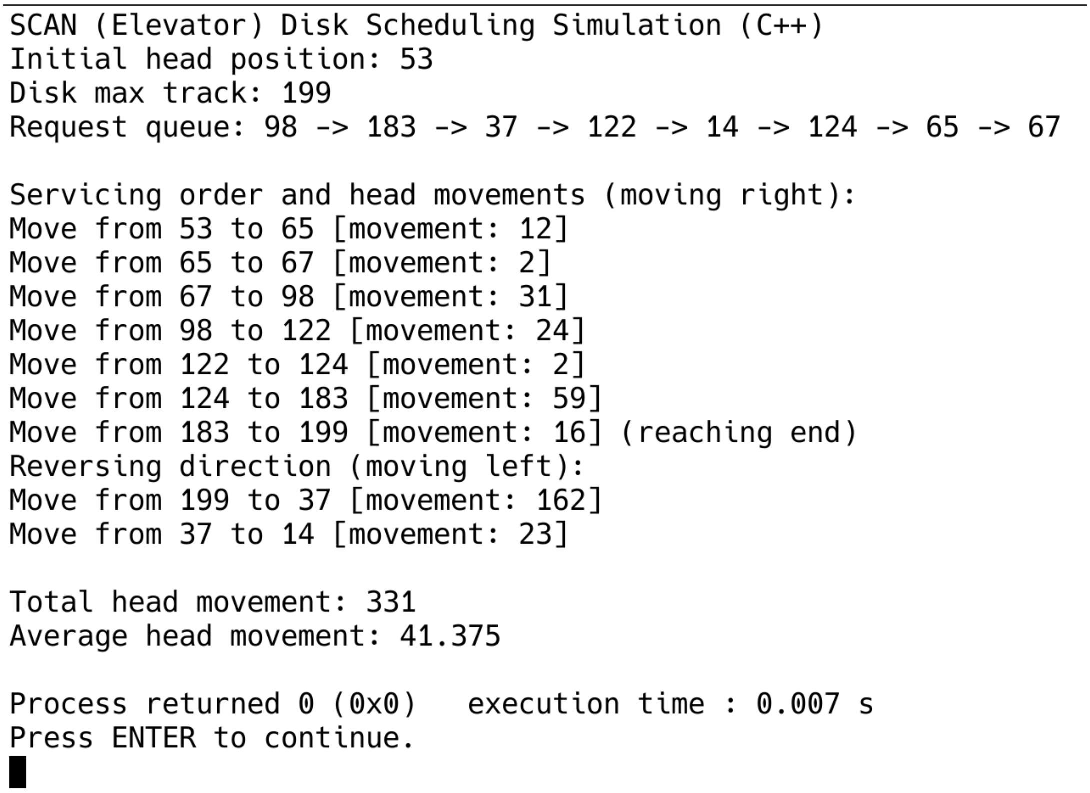
- ## Description :

8.3 Disk scheduling using SCAN Algorithm

- **Code :**

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cstdlib>
using namespace std;

int main() {
    vector<int> requests = {98, 183, 37, 122, 14, 124, 65, 67};
    int head = 53;
    int disk_max = 199;

    cout << "SCAN (Elevator) Disk Scheduling Simulation (C++)" << endl;
    cout << "Initial head position: " << head << endl;
    cout << "Disk max track: " << disk_max << endl;
    cout << "Request queue: ";
    for (size_t i = 0; i < requests.size(); ++i)
        cout << requests[i] << (i < requests.size()-1 ? " -> " : "");
    cout << endl << endl;

    vector<int> to_service = requests;
    to_service.push_back(head);
    sort(to_service.begin(), to_service.end());

    int pos = find(to_service.begin(), to_service.end(), head) -
    to_service.begin();

    int total_movement = 0;
    int current = head;
    cout << "Servicing order and head movements (moving right):" << endl;

    for (size_t i = pos+1; i < to_service.size(); ++i) {
        cout << "Move from " << current << " to " << to_service[i]
        << " [movement: " << abs(to_service[i] - current) << "]" << endl;
        total_movement += abs(to_service[i] - current);
        current = to_service[i];
    }
    if (current != disk_max) {
        cout << "Move from " << current << " to " << disk_max
    << " [movement: " << abs(disk_max - current) << "] (reaching end)" << endl;
        total_movement += abs(disk_max - current);
        current = disk_max;
    }
    cout << "Reversing direction (moving left):" << endl;
    for (int i = pos-1; i >= 0; --i) {
        cout << "Move from " << current << " to " << to_service[i]
        << " [movement: " << abs(to_service[i] - current) << "]" << endl;
        total_movement += abs(to_service[i] - current);
        current = to_service[i];
    }
    int serviced_requests = requests.size();
    double avg_movement = (double)total_movement / serviced_requests;

    cout << "\nTotal head movement: " << total_movement << endl;
    cout << "Average head movement: " << avg_movement << endl;

    return 0;
}
```

- **Input / Output :**



```
SCAN (Elevator) Disk Scheduling Simulation (C++)
Initial head position: 53
Disk max track: 199
Request queue: 98 -> 183 -> 37 -> 122 -> 14 -> 124 -> 65 -> 67

Servicing order and head movements (moving right):
Move from 53 to 65 [movement: 12]
Move from 65 to 67 [movement: 2]
Move from 67 to 98 [movement: 31]
Move from 98 to 122 [movement: 24]
Move from 122 to 124 [movement: 2]
Move from 124 to 183 [movement: 59]
Move from 183 to 199 [movement: 16] (reaching end)
Reversing direction (moving left):
Move from 199 to 37 [movement: 162]
Move from 37 to 14 [movement: 23]

Total head movement: 331
Average head movement: 41.375

Process returned 0 (0x0)   execution time : 0.007 s
Press ENTER to continue.
```

Figure 13: Output

- **Description :**