Index Page

Name: Suman Bisunkhe **Sub Code:** CSC264

Faculty: B.Sc.CSIT

LN	TITLE	DOS	GRADE	REMARKS
1	Write a program to calculate the no. of page fault for user input page reference and frame size using FIFO page replacement algorithm.	2024/08/06		
2	Write a program to calculate the no. of page fault for user input page reference and frame size using LRU page replacement algorithm.	2024/08/07		
3	Write a program to calculate the no. of page fault for user input page reference and frame size using OPR page replacement algorithm.	2024/08/07		
4	Write a program to simulate memory allocation strategy for user input free space list and incoming process request using Best Fit Allocation.	2024/08/07		
5	Write a program to simulate memory allocation strategy for user input free space list and incoming process request using Worst Fit Allocation.	2024/08/07		
6	Write a program to simulate memory allocation strategy for user input free space list and incoming process request using First Fit Allocation.	2024/08/07		
7	Write a program to demonstrate Belady's Anomaly for user input page reference string.	2024/08/07		
8	Write a program to calculate average Turn Around Time, and Waiting Time for user input processes using FCFS and their parameters. Also draw it's Gantt chart.	2024/08/07		
9	Write a program to calculate average Turn Around Time, and Waiting Time for user input processes using SJF and their parameters. Also draw it's Gantt chart.	2024/08/07		
10	Write a program to calculate average Turn Around Time, and Waiting Time for user input processes using RR and their parameters. Also draw it's Gantt chart.	2024/08/07		
11	Write a program to calculate average Turn Around Time, and Waiting Time for user input processes using Priority Scheduling and their parameters. Also draw it's Gantt chart.	2024/08/07		

**Faculty must grade as A, B, C or D*

...for a complete human being
Texas
International college

LAB NO: 01 **DATE: 2024/08/06**

Write a program to find the number of page fault for user input page reference string and frame size for FIFO page replacement algorithm.

FIFO (First-In-First-Out)

FIFO is a page replacement algorithm that replaces the oldest page in the memory (the one that arrived first) when a new page needs to be loaded and there is no free frame available.

Page Fault

When a page requested by the program is not found in the main memory, requiring the system to bring the page from secondary storage into the main memory, it is called Page Fault.

Reference String

A reference string is a sequence of memory page numbers that a program will access, used to simulate and analyze page replacement algorithms

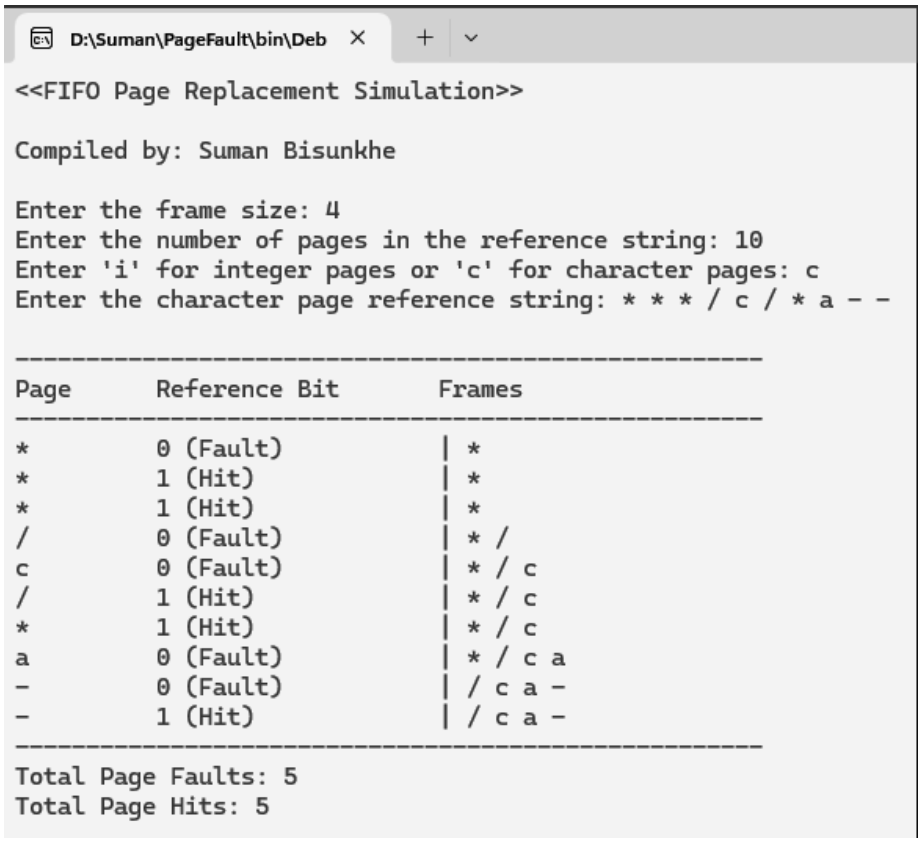
Algorithm

```
// Algorithm for FIFO Page replacement algorithm

1. Initialize an empty queue pageQueue
2. Initialize an empty set pagesInFrame
3. Initialize pageFaults to 0

4. Input frameSize (number of frames)
5. Input numPages (number of pages in the reference string)

6. For each page in the reference string:
7. If the page is not in pagesInFrame:
8. Increment pageFaults

9. If the number of pages in pagesInFrame is equal to frameSize:
       a. Remove the oldest page from pageQueue
       b. Remove this page from pagesInFrame

10. Add the new page to pageQueue
11. Add the new page to pagesInFrame

12. Output pageFaults
```

```
// FIFO page replacement algorithm


#include <iostream>
#include <vector>
#include <set>
#include <iomanip>

using namespace std;

template <typename T>
int countPageFaults(const vector<T>& pageReference, int frameSize) {
    set<T> pagesInFrame;
    vector<T> pageQueue;
    int pageFaults = 0;
    int pageHits = 0;

    cout << "\n-----------------------------------------------------" << endl;
    cout << left << setw(10) << "Page" << setw(20) << "Reference Bit" << "Frames" << endl;
    cout << "-----------------------------------------------------" << endl;

    for (const auto& page : pageReference) {
        bool hit = false;

        // If the page is not in the frame
        if (pagesInFrame.find(page) == pagesInFrame.end()) {
            // Increment page faults
            ++pageFaults;

            // If the frame is full, remove the oldest page
            if (pagesInFrame.size() == frameSize) {
                T oldestPage = pageQueue[0];
                pageQueue.erase(pageQueue.begin());
                pagesInFrame.erase(oldestPage);
            }

            // Add the new page to the frame
            pagesInFrame.insert(page);
            pageQueue.push_back(page);
        } else {
            ++pageHits;
            hit = true;
        }

        // Display the current frame content
        cout << left << setw(10) << page << setw(20) << (hit ? "1 (Hit)" : "0 (Fault)") <<
"| ";
        for (const auto& p : pageQueue) {
            cout << p << " ";
        }
        cout << endl;
    }
```

```
    cout << "-----------------------------------------------------" << endl;
    cout << "Total Page Faults: " << pageFaults << endl;
    cout << "Total Page Hits: " << pageHits << endl;
    return pageFaults;

}

int main() {
    int frameSize, numPages;
    char type;

    cout << "<<FIFO Page Replacement Simulation>>\n" << endl;

    cout << "\nCompiled by Suman Bisunkhe" << endl;


    cout << "Enter the frame size: ";
    cin >> frameSize;

    cout << "Enter the number of pages in the reference string: ";
    cin >> numPages;

    cout << "Enter 'i' for integer pages or 'c' for character pages: ";
    cin >> type;

    if (type == 'i') {
        vector<int> pageReference(numPages);
        cout << "Enter the integer page reference string: ";
        for (int i = 0; i < numPages; ++i) {
            cin >> pageReference[i];
        }

        countPageFaults(pageReference, frameSize);
    } else if (type == 'c') {
        vector<char> pageReference(numPages);
        cout << "Enter the character page reference string: ";
        for (int i = 0; i < numPages; ++i) {
            cin >> pageReference[i];
        }

        countPageFaults(pageReference, frameSize);
    } else {
        cout << "Invalid input type." << endl;
    }


    return 0;
}
```

OUTPUT


D:\Suman\PageFault\bin\Deb
<<FIFO Page Replacement Simulation>>
Compiled by: Suman Bisunkhe
Enter the frame size: 4
Enter the number of pages in the reference string: 10
Enter 'i' for integer pages or 'c' for character pages: c
Enter the character page reference string: * * * / c / * a - -
Page Reference Bit Frames
* 0 (Fault) | *
* 1 (Hit) | *
* 1 (Hit) | *
/ 0 (Fault) | * /
c 0 (Fault) | * / c
/ 1 (Hit) | * / c
* 1 (Hit) | * / c
a 0 (Fault) | * / c a
- 0 (Fault) | / c a -
- 1 (Hit) | / c a -
Total Page Faults: 5
Total Page Hits: 5

...for a complete human being
Texas International college

LAB NO: 02 **DATE: 2024/08/06**

Write a program to find the number of page fault for user input page reference string and frame size for LRU page replacement algorithm.

LRU (Least Recently Used) Algorithm

LRU is a page replacement algorithm that replaces the least recently used page in memory when a new page needs to be loaded and there is no free frame available.

Page Fault

When a page requested by the program is not found in the main memory, requiring the system to bring the page from secondary storage into the main memory, it is called Page Fault.

Reference String

A reference string is a sequence of memory page numbers that a program will access, used to simulate and analyze page replacement algorithms

Algorithm

// Algorithm for LRU Page replacement algorithm

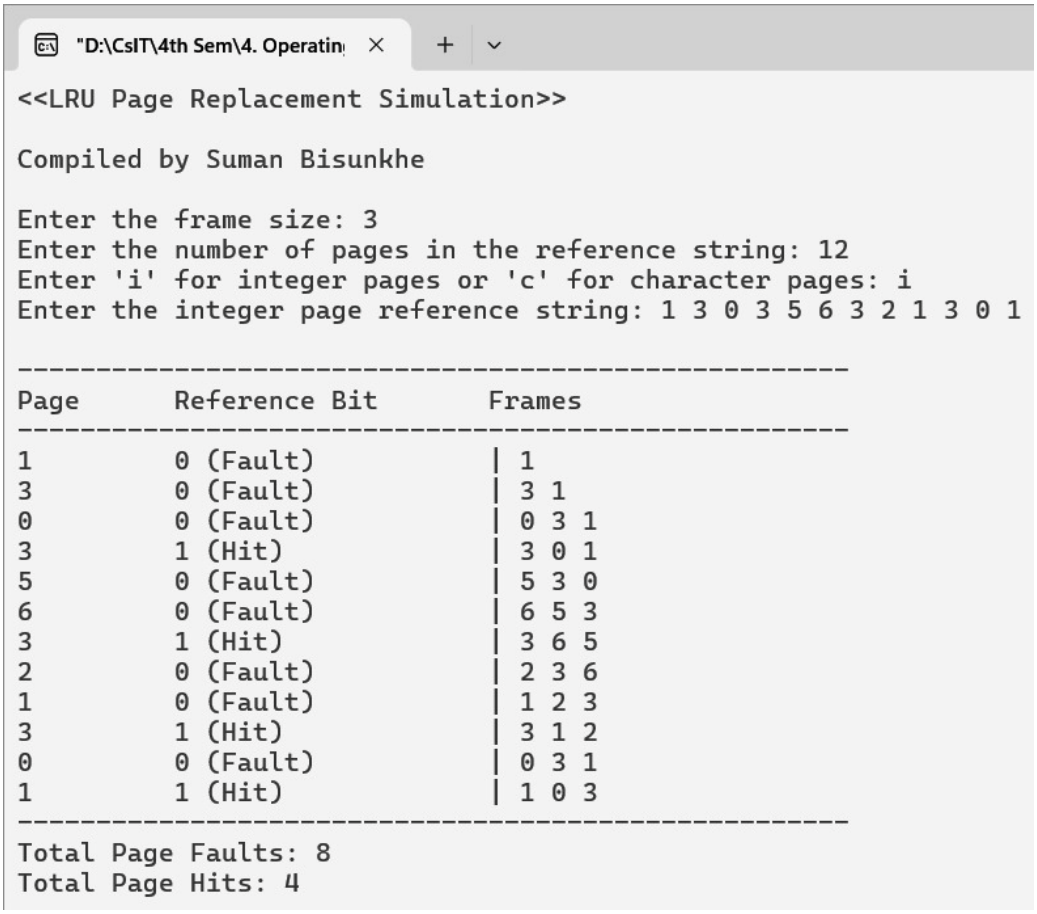
```
1) Initialize lruList, pageMap, and pageFaults.

2) Input frameSize and numPages.

3) For each page:

4) If not in pageMap:

   a) Increment pageFaults.

   b) If lruList is full, remove the oldest page.

5) Update lruList and pageMap.

6) Output pageFaults.
```

SOURCE CODE

```
// LRU page replacement algorithm

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

using namespace std;

template <typename T>
int countPageFaultsLRU(const vector<T>& pageReference, int frameSize) {
    unordered_map<T, typename list<T>::iterator> pageMap;
    list<T> lruList;
    int pageFaults = 0;
    int pageHits = 0;

    cout << "\n-----------------------------------------------------" << endl;
    cout << left << setw(10) << "Page" << setw(20) << "Reference Bit" << "Frames" << endl;
    cout << "-----------------------------------------------------" << endl;

    for (const auto& page : pageReference) {
        bool hit = false;

        // If the page is not in the frame
        if (pageMap.find(page) == pageMap.end()) {
            // Increment page faults
            ++pageFaults;

            // If the frame is full, remove the least recently used page
            if (lruList.size() == frameSize) {
                T lruPage = lruList.back();
                lruList.pop_back();
                pageMap.erase(lruPage);
            }
        } else {
            // Page hit
            ++pageHits;
            hit = true;
            lruList.erase(pageMap[page]);
        }

        // Add the new page or update its position in the frame
        lruList.push_front(page);
        pageMap[page] = lruList.begin();

        // Display the current frame content
        cout << left << setw(10) << page << setw(20) << (hit ? "1 (Hit)" : "0 (Fault)") <<
"| ";
```

```
  for (const auto& p : lruList) {
            cout << p << " ";
        }
        cout << endl;
    }

    cout << "-----------------------------------------------------" << endl;
    cout << "Total Page Faults: " << pageFaults << endl;
    cout << "Total Page Hits: " << pageHits << endl;
    return pageFaults;
}

int main() {
    int frameSize, numPages;
    char type;

    cout << "<<LRU Page Replacement Simulation>>\n" << endl;
    cout << "Compiled by Suman Bisunkhe\n" << endl;

    cout << "Enter the frame size: ";
    cin >> frameSize;

    cout << "Enter the number of pages in the reference string: ";
    cin >> numPages;

    cout << "Enter 'i' for integer pages or 'c' for character pages: ";
    cin >> type;

    if (type == 'i') {
        vector<int> pageReference(numPages);
        cout << "Enter the integer page reference string: ";
        for (int i = 0; i < numPages; ++i) {
            cin >> pageReference[i];
        }

        countPageFaultsLRU(pageReference, frameSize);
    } else if (type == 'c') {
        vector<char> pageReference(numPages);
        cout << "Enter the character page reference string: ";
        for (int i = 0; i < numPages; ++i) {
            cin >> pageReference[i];
        }

        cout << "\n-- LRU Page Replacement Algorithm --\n";
        countPageFaultsLRU(pageReference, frameSize);
    } else {
        cout << "Invalid input type." << endl;
    }

    return 0;
}
```

OUTPUT



```
"D:\CsIT\4th Sem\4. Operatin

<<LRU Page Replacement Simulation>>

Compiled by Suman Bisunkhe

Enter the frame size: 3
Enter the number of pages in the reference string: 12
Enter 'i' for integer pages or 'c' for character pages: i
Enter the integer page reference string: 1 3 0 3 5 6 3 2 1 3 0 1

-------------------------------------------------
Page      Reference Bit       Frames
-------------------------------------------------
1         0 (Fault)           | 1
3         0 (Fault)           | 3 1
0         0 (Fault)           | 0 3 1
3         1 (Hit)             | 3 0 1
5         0 (Fault)           | 5 3 0
6         0 (Fault)           | 6 5 3
3         1 (Hit)             | 3 6 5
2         0 (Fault)           | 2 3 6
1         0 (Fault)           | 1 2 3
3         1 (Hit)             | 3 1 2
0         0 (Fault)           | 0 3 1
1         1 (Hit)             | 1 0 3
-------------------------------------------------
Total Page Faults: 8
Total Page Hits: 4
```

...for a complete human being
Texas
International college

LAB NO: 03 **DATE: 2024/08/06**

Write a program to find the number of page fault for user input page reference string and frame size for Optimal page replacement algorithm.

Optimal Page Replacement Algorithm

The Optimal Page Replacement Algorithm replaces the page that will not be used for the longest time in the future. It aims to minimize page faults by choosing the optimal page to evict.

Page Fault

When a page requested by the program is not found in the main memory, requiring the system to bring the page from secondary storage into the main memory, it is called Page Fault.

Reference String

A reference string is a sequence of memory page numbers that a program will access, used to simulate and analyze page replacement algorithms

Algorithm

```
// Algorithm for Optimal Page replacement algorithm

1) Initialize an empty list pageQueue to track pages in memory.
2) Initialize pageFaults to 0.
3) Input frameSize and numPages.
4) For each page in the reference string:
5) If the page is not in pageQueue:
   a) Increment pageFaults.
   b) If pageQueue is full:
      i) Find the page in pageQueue that will not be used for the longest time in
         the future.
      ii) Remove this page from pageQueue.
   c) Add the new page to pageQueue.
6) Output pageFaults.
```

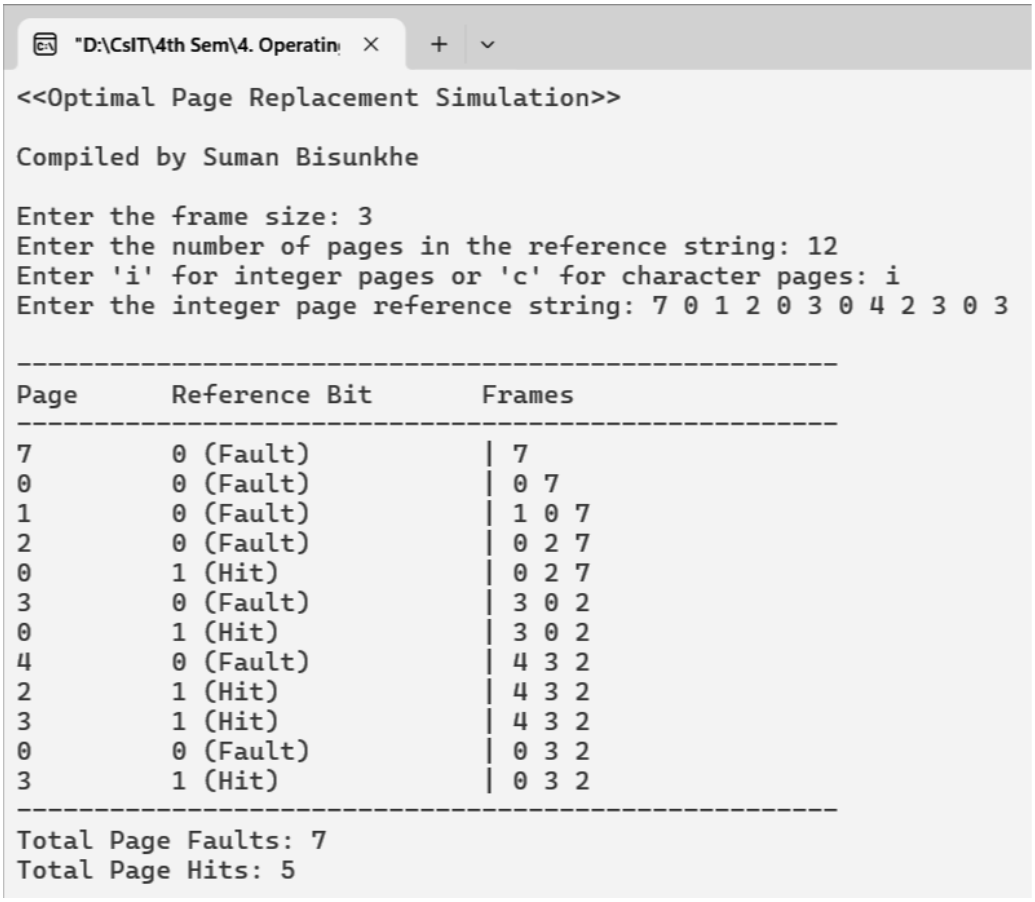
SOURCE CODE

```
// Optimal page replacement algorithm

#include <iostream>
#include <vector>
#include <unordered_set>
#include <unordered_map>
#include <iomanip>
#include <algorithm>

using namespace std;

template <typename T>
int countPageFaultsOptimal(const vector<T>& pageReference, int frameSize) {
    unordered_set<T> pagesInFrame;
    unordered_map<T, int> futureUse;
    int pageFaults = 0;
    int pageHits = 0;
    int numPages = pageReference.size();

    cout << "\n-----------------------------------------------------" << endl;
    cout << left << setw(10) << "Page" << setw(20) << "Reference Bit" << "Frames" << endl;
    cout << "-----------------------------------------------------" << endl;

    for (int i = 0; i < numPages; ++i) {
        const auto& page = pageReference[i];
        bool hit = false;

        // If the page is already in the frame
        if (pagesInFrame.find(page) != pagesInFrame.end()) {
            ++pageHits;
            hit = true;
        } else {
            ++pageFaults;

            // If there is space in the frame, add the new page
            if (pagesInFrame.size() < frameSize) {
                pagesInFrame.insert(page);
            } else {
```

```
                // Determine which page to replace
                T pageToReplace;
                int farthestUse = -1;

                // Calculate the future use of each page
                for (const auto& p : pagesInFrame) {
                    auto it = find(pageReference.begin() + i, pageReference.end(), p);
                    int nextUse = (it == pageReference.end()) ? numPages :
distance(pageReference.begin(), it);

                    if (nextUse > farthestUse) {
                        farthestUse = nextUse;
                        pageToReplace = p;
                    }
                }

                // Replace the page that is used farthest in the future
                pagesInFrame.erase(pageToReplace);
                pagesInFrame.insert(page);
            }
        }

        // Display the current frame content
        cout << left << setw(10) << page << setw(20) << (hit ? "1 (Hit)" : "0 (Fault)") <<
"| ";
        for (const auto& p : pagesInFrame) {
            cout << p << " ";
        }
        cout << endl;
    }

    cout << "------------------------------------------------------" << endl;
    cout << "Total Page Faults: " << pageFaults << endl;
    cout << "Total Page Hits: " << pageHits << endl;
    return pageFaults;
}

int main() {
    int frameSize, numPages;
    char type;

    cout << "<<Optimal Page Replacement Simulation>>\n" << endl;
```

```
    cout << "Compiled by Suman Bisunkhe\n" << endl;

    cout << "Enter the frame size: ";
    cin >> frameSize;

    cout << "Enter the number of pages in the reference string: ";
    cin >> numPages;

    cout << "Enter 'i' for integer pages or 'c' for character pages: ";
    cin >> type;

    if (type == 'i') {
        vector<int> pageReference(numPages);
        cout << "Enter the integer page reference string: ";
        for (int i = 0; i < numPages; ++i) {
            cin >> pageReference[i];
        }

        countPageFaultsOptimal(pageReference, frameSize);
    } else if (type == 'c') {
        vector<char> pageReference(numPages);
        cout << "Enter the character page reference string: ";
        for (int i = 0; i < numPages; ++i) {
            cin >> pageReference[i];
        }

        cout << "\n-- Optimal Page Replacement Algorithm --\n";
        countPageFaultsOptimal(pageReference, frameSize);
    } else {
        cout << "Invalid input type." << endl;
    }

    return 0;
}
```

OUTPUT



```
"D:\CsIT\4th Sem\4. Operatin
<<Optimal Page Replacement Simulation>>

Compiled by Suman Bisunkhe

Enter the frame size: 3
Enter the number of pages in the reference string: 12
Enter 'i' for integer pages or 'c' for character pages: i
Enter the integer page reference string: 7 0 1 2 0 3 0 4 2 3 0 3

--------------------------------------------------
Page      Reference Bit       Frames
--------------------------------------------------
7         0 (Fault)           | 7
0         0 (Fault)           | 0 7
1         0 (Fault)           | 1 0 7
2         0 (Fault)           | 0 2 7
0         1 (Hit)             | 0 2 7
3         0 (Fault)           | 3 0 2
0         1 (Hit)             | 3 0 2
4         0 (Fault)           | 4 3 2
2         1 (Hit)             | 4 3 2
3         1 (Hit)             | 4 3 2
0         0 (Fault)           | 0 3 2
3         1 (Hit)             | 0 3 2
--------------------------------------------------
Total Page Faults: 7
Total Page Hits: 5
```

...for a complete human being

Texas International college

LAB NO: 04 **DATE: 2024/04/06**

Write a program to simulate memory allocation strategy for user input, free space list and incoming process request using Best Fit Allocation.

Memory Allocation: Memory allocation is the process of assigning blocks of memory to various data and programs to optimize the use of the available memory space.

Free Space List: A free space list is a data structure that keeps track of all the free memory blocks in the system. This list helps in quickly finding available memory blocks for allocation.

Incoming Process Request: An incoming process request is a request from a process (program) that needs a certain amount of memory to execute. The memory management system must find a suitable block of memory for this request.

Best Fit Allocation: Best fit allocation is a memory allocation strategy that searches the entire free space list to find the smallest free block that is large enough to accommodate the incoming process request. This strategy minimizes wasted memory but may lead to external fragmentation.

Algorithm

```
// Algorithm for Best Fit Memory Allocation

1. Initialize freeList with memory blocks.
2. Sort freeList by size.

3. For each process request:
   a. If suitable block found:
      i. Allocate memory.
      ii. Update or remove block.
   b. If not, increment processFailures.

4. Output processFailures.

5. For each memory free request:
   a. Add block to freeList.
   b. Sort freeList by size.
```

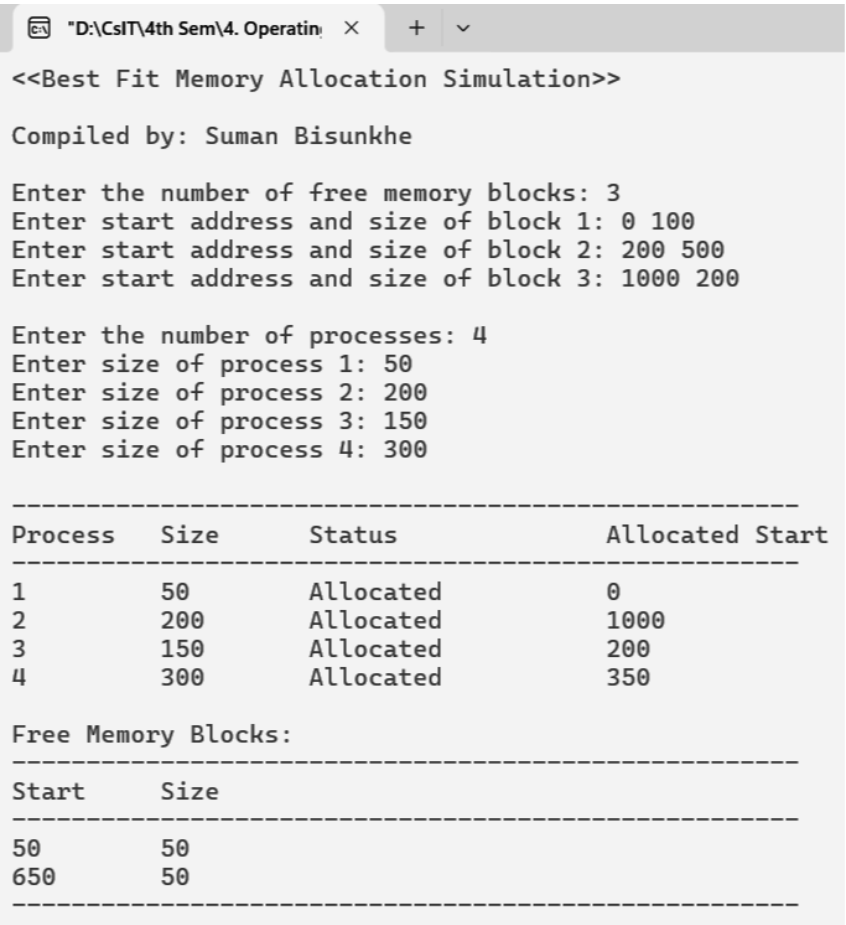
SOURCE CODE

// Best Fit Memory Allocation

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iomanip>

using namespace std;

struct MemoryBlock {
    int start;
    int size;
};

struct Process {
    int id;
    int size;
    bool allocated;
    int allocatedStart;
};

bool compareBlocks(const MemoryBlock &a, const MemoryBlock &b) {
    return a.size < b.size;
}

void allocateMemory(vector<MemoryBlock> &freeList, Process &p) {
    // Sort the free memory blocks by size (Best Fit)
    sort(freeList.begin(), freeList.end(), compareBlocks);
    for (auto it = freeList.begin(); it != freeList.end(); ++it) {
        if (it->size >= p.size) {
            p.allocated = true;
            p.allocatedStart = it->start;
            it->start += p.size;
            it->size -= p.size;
            if (it->size == 0) freeList.erase(it);
            return;
        }
    }
    p.allocated = false;
}

void displayTable(const vector<MemoryBlock> &freeList, const vector<Process> &processes)
{
    cout << "\n-----------------------------------------------------" << endl;
    cout << left << setw(10) << "Process" << setw(10) << "Size" << setw(20) << "Status"
<< "Allocated Start" << endl;
    cout << "-----------------------------------------------------" << endl;
    for (const auto &p : processes) {
        cout << left << setw(10) << p.id << setw(10) << p.size << setw(20) <<
(p.allocated ? "Allocated" : "Not Allocated")
             << (p.allocated ? to_string(p.allocatedStart) : "-") << endl;
    }
```

```
    cout << "\nFree Memory Blocks:\n";
    cout << "-----------------------------------------------------" << endl;
    cout << left << setw(10) << "Start" << "Size" << endl;

    cout << "-----------------------------------------------------" << endl;
    for (const auto &block : freeList) {
        cout << left << setw(10) << block.start << block.size << endl;
    }
    cout << "-----------------------------------------------------" << endl;
}

int main() {
    int numBlocks, numProcesses;

    cout << "<<Best Fit Memory Allocation Simulation>>\n" << endl;
    cout << "Compiled by: Suman Bisunkhe\n" << endl;

    cout << "Enter the number of free memory blocks: ";
    cin >> numBlocks;

    vector<MemoryBlock> freeList(numBlocks);
    for (int i = 0; i < numBlocks; ++i) {
        cout << "Enter start address and size of block " << i + 1 << ": ";
        cin >> freeList[i].start >> freeList[i].size;
    }

    cout << "\nEnter the number of processes: ";
    cin >> numProcesses;

    vector<Process> processes(numProcesses);
    for (int i = 0; i < numProcesses; ++i) {
        processes[i].id = i + 1;
        cout << "Enter size of process " << processes[i].id << ": ";
        cin >> processes[i].size;
    }

    for (auto &process : processes) {
        allocateMemory(freeList, process);
    }

    displayTable(freeList, processes);

    return 0;
}
```

OUTPUT


"D:\CsIT\4th Sem\4. Operatin
<<Best Fit Memory Allocation Simulation>>
Compiled by: Suman Bisunkhe
Enter the number of free memory blocks: 3
Enter start address and size of block 1: 0 100
Enter start address and size of block 2: 200 500
Enter start address and size of block 3: 1000 200
Enter the number of processes: 4
Enter size of process 1: 50
Enter size of process 2: 200
Enter size of process 3: 150
Enter size of process 4: 300
Process Size Status Allocated Start
1 50 Allocated 0
2 200 Allocated 1000
3 150 Allocated 200
4 300 Allocated 350
Free Memory Blocks:
Start Size
50 50
650 50

...for a complete human being
Texas International college

LAB NO: 05 **DATE: 2024/08/06**

Write a program to simulate memory allocation strategy for user input free space list and incoming process request using Worst Fit Allocation.

Worst Fit Allocation

The Worst Fit Page Replacement Algorithm allocates incoming process requests to the largest available block of free memory space, minimizing fragmentation by using the largest available slot.

Page Fault

When a page requested by the program is not found in the main memory, requiring the system to bring the page from secondary storage into the main memory, it is called Page Fault.

Reference String

A reference string is a sequence of memory page numbers that a program will access, used to simulate and analyze page replacement algorithms

Algorithm

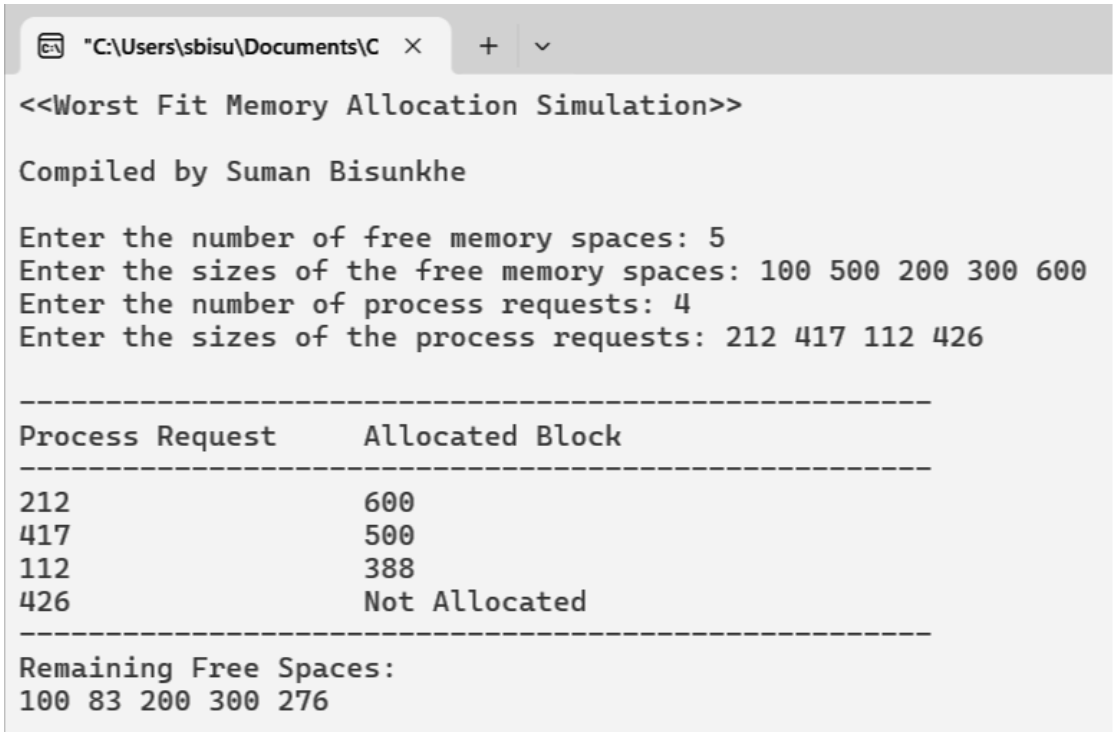
```
// Algorithm for Worst Fit Page Replacement

1. Initialize pageQueue as an empty list to track pages in memory.
2. Initialize pageFaults to 0.
3. Input frameSize and numPages.
4. For each page in the reference string:
   a. If the page is not in pageQueue:
          i. Increment pageFaults.
          ii. If pageQueue is full:
                  1. Find the page in pageQueue with the largest block size.
                  2. Remove this page from pageQueue.
          iii.      Add the new page to pageQueue.
5. Output pageFaults.
```

SOURCE CODE

```
// Worst Fit page replacement algorithm

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

using namespace std;

void worstFit(vector<int>& freeSpaces, vector<int>& processRequests) {
    cout << "\n-------------------------------------------------" << endl;
    cout << left << setw(20) << "Process Request" << setw(20) << "Allocated Block" << endl;
    cout << "-------------------------------------------------" << endl;

    for (const auto& process : processRequests) {
        // Find the worst fit block
        auto worstBlock = max_element(freeSpaces.begin(), freeSpaces.end(), [process](int
a, int b) {
            if (a >= process && b >= process)
                return a < b;  // Both blocks are suitable, choose the larger one
            return a < process;  // Only choose if block is large enough
        });

        // Check if we found a block that can fit the process
        if (*worstBlock >= process) {
            cout << left << setw(20) << process << setw(20) << *worstBlock << endl;
            *worstBlock -= process;  // Allocate memory
        } else {
            cout << left << setw(20) << process << setw(20) << "Not Allocated" << endl;
        }
    }

    cout << "-------------------------------------------------" << endl;
    cout << "Remaining Free Spaces:" << endl;
    for (const auto& space : freeSpaces) {
        cout << space << " ";
    }
    cout << endl;
}
```

```
int main() {
    int numFreeSpaces, numProcesses;

    cout << "<<Worst Fit Memory Allocation Simulation>>\n" << endl;
    cout << "Compiled by Suman Bisunkhe\n" << endl;

    cout << "Enter the number of free memory spaces: ";
    cin >> numFreeSpaces;
    vector<int> freeSpaces(numFreeSpaces);
    cout << "Enter the sizes of the free memory spaces: ";
    for (int i = 0; i < numFreeSpaces; ++i) {
        cin >> freeSpaces[i];
    }

    cout << "Enter the number of process requests: ";
    cin >> numProcesses;
    vector<int> processRequests(numProcesses);
    cout << "Enter the sizes of the process requests: ";
    for (int i = 0; i < numProcesses; ++i) {
        cin >> processRequests[i];
    }

    worstFit(freeSpaces, processRequests);

    return 0;
}
```

OUTPUT


"C:\Users\sbisu\Documents\C
<<Worst Fit Memory Allocation Simulation>>
Compiled by Suman Bisunkhe
Enter the number of free memory spaces: 5
Enter the sizes of the free memory spaces: 100 500 200 300 600
Enter the number of process requests: 4
Enter the sizes of the process requests: 212 417 112 426
Process Request Allocated Block
212 600
417 500
112 388
426 Not Allocated
Remaining Free Spaces:
100 83 200 300 276

...for a complete human being
Texas
International college

LAB NO: 06 **DATE: 2024/08/06**

Write a program to simulate memory allocation strategy for user input free space list and incoming process request using First Fit Allocation.

First Fit Allocation

First Fit Allocation is a memory allocation strategy that places a process into the first available block of memory that is large enough to accommodate it.

Page Fault

When a page requested by the program is not found in the main memory, requiring the system to bring the page from secondary storage into the main memory, it is called Page Fault.

Reference String

A reference string is a sequence of memory page numbers that a program will access, used to simulate and analyze page replacement algorithms

Algorithm

```
// Algorithm for First Fit Page replacement

1. Initialize pageQueue as an empty list to track pages in memory.
2. Initialize pageFaults to 0.
3. Input frameSize and numPages.
4. For each page in the reference string:
   a. If the page is not in pageQueue:
          i. Increment pageFaults.
          ii. If pageQueue is full:
                  1. Find the page in pageQueue with the largest block size.
                  2. Remove this page from pageQueue.
          iii.      Add the new page to pageQueue.
5. Output pageFaults.
```

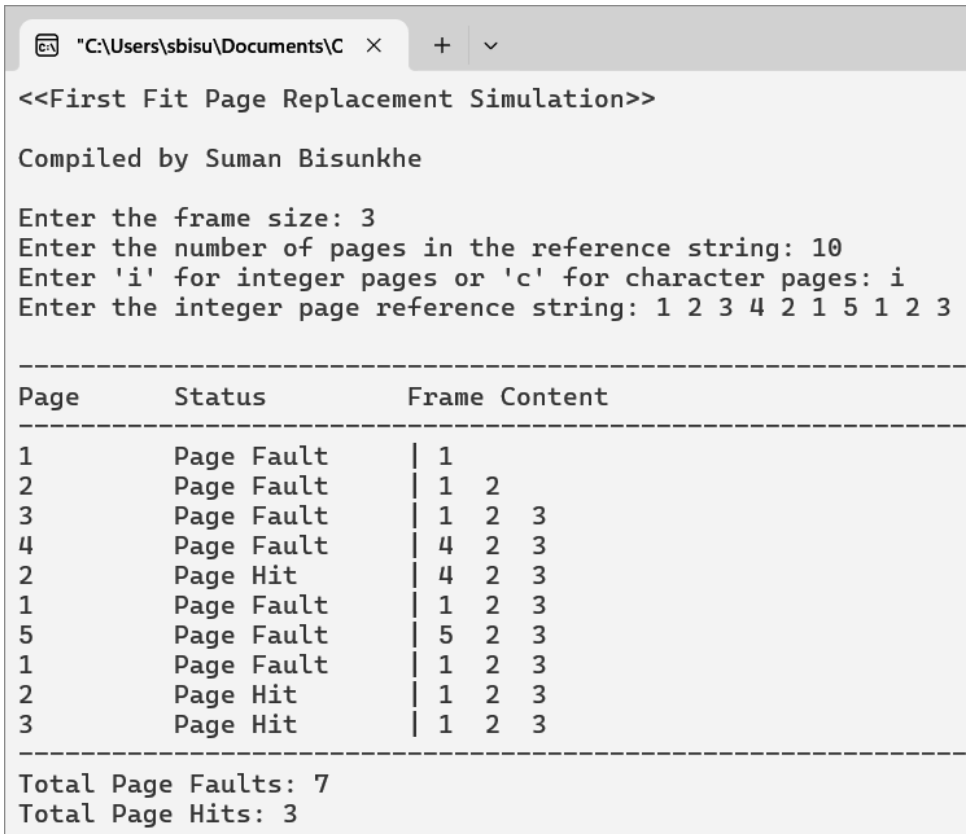
SOURCE CODE

```
// First Fit page replacement algorithm

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

using namespace std;

template <typename T>
void firstFitPageReplacement(vector<T>& pageReference, int frameSize) {
    vector<T> frames(frameSize, -1); // Initialize frames with -1 indicating empty slots
    int pageFaults = 0;
    int pageHits = 0;

    cout << "\n-------------------------------------------------------------" << endl;
    cout << left << setw(10) << "Page" << setw(15) << "Status" << "Frame Content" << endl;
    cout << "-------------------------------------------------------------" << endl;

    for (const auto& page : pageReference) {
        auto it = find(frames.begin(), frames.end(), page);

        bool isPageFault = (it == frames.end());

        if (isPageFault) {
            ++pageFaults;
            auto emptySlotIt = find(frames.begin(), frames.end(), -1);

            if (emptySlotIt != frames.end()) {
                *emptySlotIt = page;
            } else {
                frames[0] = page; // Replace the first page (First Fit approach)
            }
        } else {
            ++pageHits;
        }

        // Display the current frame content
```

```
        cout << left << setw(10) << page
             << setw(15) << (isPageFault ? "Page Fault" : "Page Hit")
             << "| ";
        for (const auto& f : frames) {
            if (f == -1) {
                cout << setw(3) << " "; // Empty slot
            } else {
                cout << setw(3) << f; // Display page number
            }
        }
        cout << endl;
    }

    cout << "-------------------------------------------------------------" << endl;
    cout << "Total Page Faults: " << pageFaults << endl;
    cout << "Total Page Hits: " << pageHits << endl;
}

int main() {
    int frameSize, numPages;
    char type;

    cout << "<<First Fit Page Replacement Simulation>>\n" << endl;
    cout << "Compiled by Suman Bisunkhe\n" << endl;

    cout << "Enter the frame size: ";
    cin >> frameSize;

    cout << "Enter the number of pages in the reference string: ";
    cin >> numPages;

    cout << "Enter 'i' for integer pages or 'c' for character pages: ";
    cin >> type;

    if (type == 'i') {
        vector<int> pageReference(numPages);
        cout << "Enter the integer page reference string: ";
        for (int i = 0; i < numPages; ++i) {
            cin >> pageReference[i];
        }
```

```
        firstFitPageReplacement(pageReference, frameSize);
    } else if (type == 'c') {
        vector<char> pageReference(numPages);
        cout << "Enter the character page reference string: ";
        for (int i = 0; i < numPages; ++i) {
            cin >> pageReference[i];
        }

        firstFitPageReplacement(pageReference, frameSize);
    } else {
        cout << "Invalid input type." << endl;
    }

    return 0;
}
```

OUTPUT



...for a complete human being
Texas International college

LAB NO: 07 **DATE: 2024/08/06**

Write a program to demonstrate Belady's Anomaly for user input page reference string.

Belady's Anomaly

Belady's Anomaly occurs when increasing the number of page frames results in an increase in the number of page faults for a page replacement algorithm, contrary to the expected decrease in page faults.

Page Fault

When a page requested by the program is not found in the main memory, requiring the system to bring the page from secondary storage into the main memory, it is called Page Fault.

Reference String

A reference string is a sequence of memory page numbers that a program will access, used to simulate and analyze page replacement algorithms

Algorithm

```
// Algorithm for Belady's Anomaly

1. Input: Page reference string, frame sizes.
2. For Each Frame Size:
      a. Initialize frame and queue.
      b. Set pageFaults to 0.
3. For Each Page:
      a. If Page Not in Frame:
            i. Increment pageFaults.
           ii. If Full: Remove oldest page.
          iii. Add new page.
4. Output: Page faults for each frame size.
```

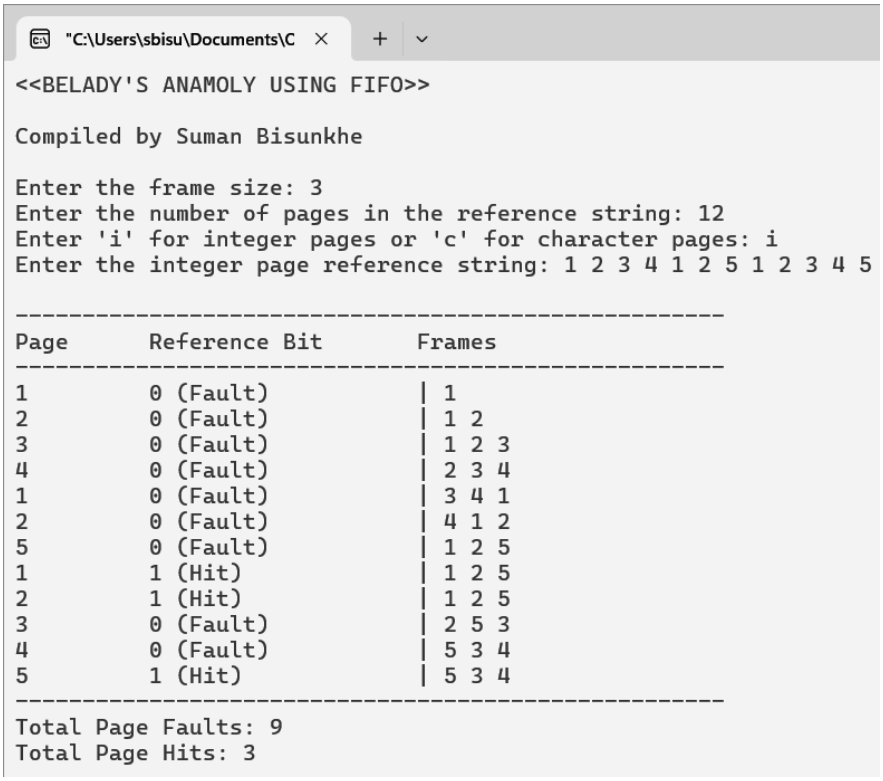
SOURCE CODE

```
// Belady's Anomaly Using FIFO

#include <iostream>
#include <vector>
#include <set>
#include <iomanip>

using namespace std;

template <typename T>
int countPageFaults(const vector<T>& pageReference, int frameSize) {
    set<T> pagesInFrame;
    vector<T> pageQueue;
    int pageFaults = 0;
    int pageHits = 0;

    cout << "\n-----------------------------------------------------" << endl;
    cout << left << setw(10) << "Page" << setw(20) << "Reference Bit" << "Frames"
<< endl;
    cout << "-----------------------------------------------------" << endl;

    for (const auto& page : pageReference) {
        bool hit = false;

        // If the page is not in the frame
        if (pagesInFrame.find(page) == pagesInFrame.end()) {
            // Increment page faults
            ++pageFaults;

            // If the frame is full, remove the oldest page
            if (pagesInFrame.size() == frameSize) {
                T oldestPage = pageQueue[0];
                pageQueue.erase(pageQueue.begin());
                pagesInFrame.erase(oldestPage);
            }
```

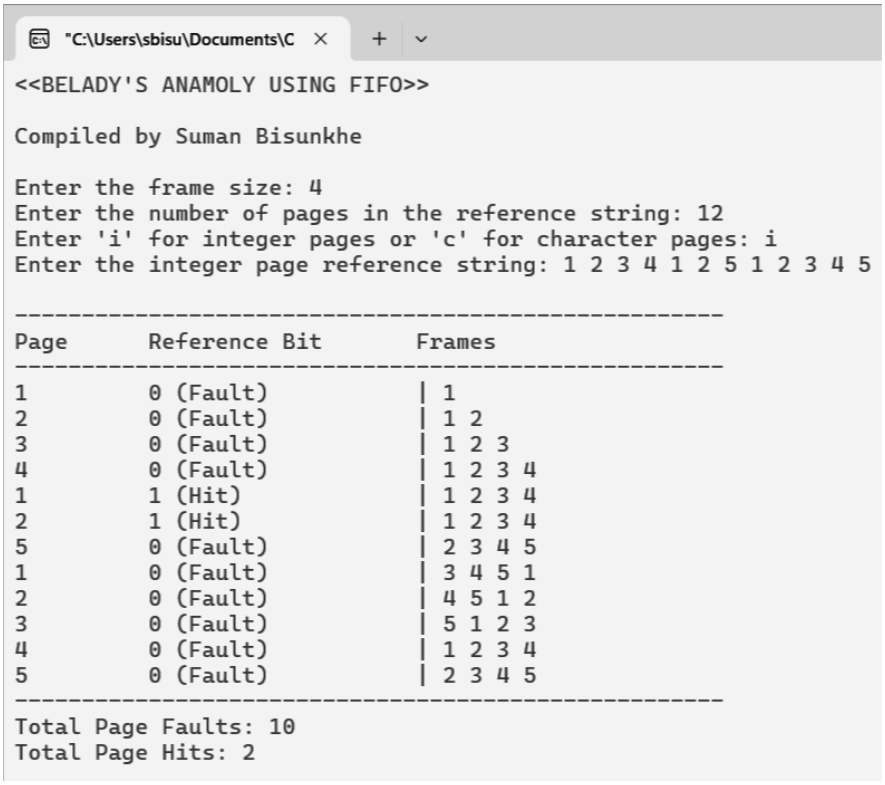
```
            // Add the new page to the frame
            pagesInFrame.insert(page);
            pageQueue.push_back(page);
        } else {
            ++pageHits;
            hit = true;
        }

        // Display the current frame content
        cout << left << setw(10) << page << setw(20) << (hit ? "1 (Hit)" : "0
(Fault)") << "| ";
        for (const auto& p : pageQueue) {
            cout << p << " ";
        }
        cout << endl;
    }

    cout << "-----------------------------------------------------" << endl;
    cout << "Total Page Faults: " << pageFaults << endl;
    cout << "Total Page Hits: " << pageHits << endl;
    return pageFaults;

}

int main() {
    int frameSize, numPages;
    char type;

    cout << "<<BELADY'S ANAMOLY USING FIFO>>" << endl;

    cout << "\nCompiled by Suman Bisunkhe\n" << endl;


    cout << "Enter the frame size: ";
    cin >> frameSize;

    cout << "Enter the number of pages in the reference string: ";
    cin >> numPages;
```

```
    cout << "Enter 'i' for integer pages or 'c' for character pages: ";
    cin >> type;

    if (type == 'i') {
        vector<int> pageReference(numPages);
        cout << "Enter the integer page reference string: ";
        for (int i = 0; i < numPages; ++i) {
            cin >> pageReference[i];
        }

        countPageFaults(pageReference, frameSize);
    } else if (type == 'c') {
        vector<char> pageReference(numPages);
        cout << "Enter the character page reference string: ";

        for (int i = 0; i < numPages; ++i) {
            cin >> pageReference[i];
        }

        countPageFaults(pageReference, frameSize);
    } else {
        cout << "Invalid input type." << endl;
    }


    return 0;
}
```

OUTPUT



```
"C:\Users\sbisu\Documents\C
<<BELADY'S ANAMOLY USING FIFO>>

Compiled by Suman Bisunkhe

Enter the frame size: 3
Enter the number of pages in the reference string: 12
Enter 'i' for integer pages or 'c' for character pages: i
Enter the integer page reference string: 1 2 3 4 1 2 5 1 2 3 4 5

-----------------------------------------------------
Page      Reference Bit       Frames
-----------------------------------------------------
1         0 (Fault)           | 1
2         0 (Fault)           | 1 2
3         0 (Fault)           | 1 2 3
4         0 (Fault)           | 2 3 4
1         0 (Fault)           | 3 4 1
2         0 (Fault)           | 4 1 2
5         0 (Fault)           | 1 2 5
1         1 (Hit)             | 1 2 5
2         1 (Hit)             | 1 2 5
3         0 (Fault)           | 2 5 3
4         0 (Fault)           | 5 3 4
5         1 (Hit)             | 5 3 4
-----------------------------------------------------
Total Page Faults: 9
Total Page Hits: 3
```



```
"C:\Users\sbisu\Documents\C
<<BELADY'S ANAMOLY USING FIFO>>

Compiled by Suman Bisunkhe

Enter the frame size: 4
Enter the number of pages in the reference string: 12
Enter 'i' for integer pages or 'c' for character pages: i
Enter the integer page reference string: 1 2 3 4 1 2 5 1 2 3 4 5

-----------------------------------------------------
Page      Reference Bit       Frames
-----------------------------------------------------
1         0 (Fault)           | 1
2         0 (Fault)           | 1 2
3         0 (Fault)           | 1 2 3
4         0 (Fault)           | 1 2 3 4
1         1 (Hit)             | 1 2 3 4
2         1 (Hit)             | 1 2 3 4
5         0 (Fault)           | 2 3 4 5
1         0 (Fault)           | 3 4 5 1
2         0 (Fault)           | 4 5 1 2
3         0 (Fault)           | 5 1 2 3
4         0 (Fault)           | 1 2 3 4
5         0 (Fault)           | 2 3 4 5
-----------------------------------------------------
Total Page Faults: 10
Total Page Hits: 2
```

...for a complete human being
Texas
International college

LAB NO: 08 **DATE: 2024/08/06**

Write a program to calculate average Turn Around Time, and Waiting Time for user input processes using FCFS and their parameters. Also draw it's Gantt chart.

First-Come, First-Served (FCFS)
A scheduling algorithm where processes are executed in the order they arrive, with no preemption or priority, ensuring that the first process to arrive gets executed first.

Turn Around Time
The total time taken from the submission of a process to its completion, including both the waiting time and the execution time of the process.

Waiting Time
The total time a process spends waiting in the ready queue before it starts execution, excluding the time it spends actually being executed.

Algorithm

```
// Algorithm for First-Come, First-Served (FCFS)

1) Start:

   a) Initialize current_time to 0.

2) For each process in the queue:

   a) Set arrival_time and burst_time for the process.

   b) Calculate waiting_time as current_time - arrival_time.

   c) Calculate turnaround_time as waiting_time + burst_time.

   d) Update current_time by adding the burst_time of the process.

3) End:

   a) Print the waiting_time and turnaround_time for each process.
```

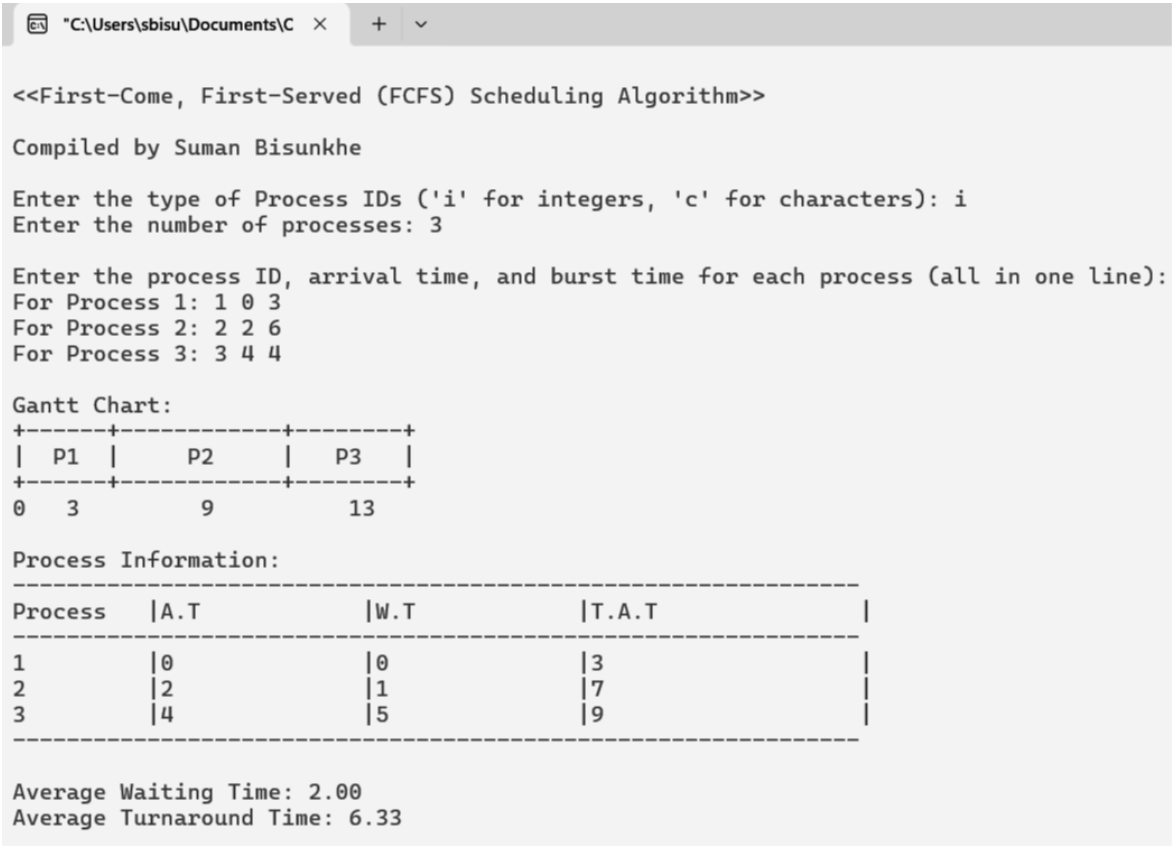
SOURCE CODE

```
// First-Come, First-Served (FCFS)


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

using namespace std;

template <typename T>
struct Process {
    T id;           // Process ID
    int arrival;    // Arrival Time
    int burst;      // Burst Time
    int start;      // Start Time
    int finish;     // Finish Time
    int waiting;    // Waiting Time
    int turnaround; // Turnaround Time
};

template <typename T>
void calculateFCFS(vector<Process<T>>& processes) {
    int n = processes.size();
    int totalWaitingTime = 0, totalTurnaroundTime = 0;

    // Sort processes by arrival time
    sort(processes.begin(), processes.end(), [](const Process<T>& a, const
Process<T>& b) {
        return a.arrival < b.arrival;
    });

    // Calculate start, finish, waiting, and turnaround times
    for (int i = 0; i < n; ++i) {
        if (i == 0) {
            processes[i].start = processes[i].arrival;
        } else {
            processes[i].start = max(processes[i - 1].finish,
processes[i].arrival);
        }
        processes[i].finish = processes[i].start + processes[i].burst;
        processes[i].waiting = processes[i].start - processes[i].arrival;
        processes[i].turnaround = processes[i].finish - processes[i].arrival;

        totalWaitingTime += processes[i].waiting;
        totalTurnaroundTime += processes[i].turnaround;
    }
```

```
    // Display Gantt chart
    cout << "\nGantt Chart:\n";
    cout << "+";
    for (const auto& p : processes) {
        cout << string(p.burst * 2, '-') << "+";
    }
    cout << "\n|";
    for (const auto& p : processes) {
        cout << string(p.burst - 1, ' ') << "P" << p.id << string(p.burst - 1, '
') << "|";
    }
    cout << "\n+";
    for (const auto& p : processes) {
        cout << string(p.burst * 2, '-') << "+";
    }
    cout << "\n";

    int currentTime = 0;
    cout << currentTime;
    for (const auto& p : processes) {
        int gap = (p.burst * 2) - to_string(p.finish).length();
        cout << string(gap / 2 + 1, ' ') << p.finish << string(gap - gap / 2, '
');
    }
    cout << "\n";

    // Display process information and average calculations
    cout << "\nProcess Information:\n";
    cout << "------------------------------------------------------------------" <<
endl;

    cout << left << setw(10) << "Process" << "|" << setw(15) << "A.T"
         << "|" << setw(15) << "W.T" << "|" << setw(20) << "T.A.T" << "|" << endl;
    cout << "------------------------------------------------------------------" <<
endl;
    for (const auto& p : processes) {
        cout << left << setw(10) << p.id
             << "|" << setw(15) << p.arrival
             << "|" << setw(15) << p.waiting
             << "|" << setw(20) << p.turnaround << "|" << endl;
    }

    cout << "------------------------------------------------------------------" <<
endl;

    cout << "\nAverage Waiting Time: " << fixed << setprecision(2) <<
(float)totalWaitingTime / n << endl;
    cout << "Average Turnaround Time: " << fixed << setprecision(2) <<
(float)totalTurnaroundTime / n << endl;
}
```

```
int main() {
    int numProcesses;
    char type;

    cout << "\n<<First-Come, First-Served (FCFS) Scheduling Algorithm>>\n" <<
endl;
    cout << "Compiled by Suman Bisunkhe\n" << endl;
    cout << "Enter the type of Process IDs ('i' for integers, 'c' for characters):
";
    cin >> type;

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

    if (type != 'i' && type != 'c') {
        cout << "Invalid type. Exiting program." << endl;
        return 1;
    }

    if (type == 'i') {
        vector<Process<int>> processes(numProcesses);
        cout << "\nEnter the process ID, arrival time, and burst time for each
process (all in one line):\n";
        for (int i = 0; i < numProcesses; ++i) {
            cout << "For Process " << i + 1 << ": ";
            cin >> processes[i].id >> processes[i].arrival >> processes[i].burst;
        }

        calculateFCFS(processes);
    } else if (type == 'c') {
        vector<Process<char>> processes(numProcesses);
        cout << "\nEnter the process ID (character), arrival time, and burst time
for each process (all in one line):\n";
        for (int i = 0; i < numProcesses; ++i) {
            cout << "For Process " << i + 1 << ": ";
            cin >> processes[i].id >> processes[i].arrival >> processes[i].burst;
        }

        calculateFCFS(processes);
    }

    return 0;
}
```

OUTPUT



```
"C:\Users\sbisu\Documents\C

<<First-Come, First-Served (FCFS) Scheduling Algorithm>>

Compiled by Suman Bisunkhe

Enter the type of Process IDs ('i' for integers, 'c' for characters): i
Enter the number of processes: 3

Enter the process ID, arrival time, and burst time for each process (all in one line):
For Process 1: 1 0 3
For Process 2: 2 2 6
For Process 3: 3 4 4

Gantt Chart:
+------+------------+--------+
|  P1  |     P2     |   P3   |
+------+------------+--------+
0   3         9          13

Process Information:
-----------------------------------------------------------------
Process   |A.T            |W.T            |T.A.T                |
-----------------------------------------------------------------
1         |0              |0              |3                    |
2         |2              |1              |7                    |
3         |4              |5              |9                    |
-----------------------------------------------------------------

Average Waiting Time: 2.00
Average Turnaround Time: 6.33
```

...for a complete human being

Texas International college

LAB NO: 09 **DATE: 2024/08/06**

Write a program to calculate average Turn Around Time, and Waiting Time for user input processes using SJF and their parameters. Also draw it's Gantt chart.

Shortest Job First (SJF)

Shortest Job First (SJF) is a scheduling algorithm that prioritizes processes with the shortest burst time to optimize overall system performance and reduce average waiting time.

Turn Around Time

The total time taken from the submission of a process to its completion, including both the waiting time and the execution time of the process.

Waiting Time

The total time a process spends waiting in the ready queue before it starts execution, excluding the time it spends actually being executed.

Algorithm

```
// Shortest Job First (SJF)

   1. Start:

      a. Initialize current_time to 0.

   2. For each process in the queue:

      a. Set arrival_time and burst_time for the process.

      b. Calculate waiting_time as current_time - arrival_time.

      c. Calculate turnaround_time as waiting_time + burst_time.

      d. Update current_time by adding the burst_time of the process.

   3. End:

      a. Print the waiting_time and turnaround_time for each process.
```

SOURCE CODE

```
// Shortest Job First (SJF)

#include <iostream>
#include <vector>
#include <iomanip>
#include <algorithm>
#include <climits>
#include <string>

using namespace std;

template <typename T>
struct Process {
    T id;           // Process ID
    int arrival;    // Arrival Time
    int burst;      // Burst Time
    int start;      // Start Time
    int finish;     // Finish Time
    int waiting;    // Waiting Time
    int turnaround; // Turnaround Time
};

template <typename T>
void calculateSJF(vector<Process<T>>& processes) {
    int n = processes.size();
    int totalWaitingTime = 0, totalTurnaroundTime = 0;
    int currentTime = 0;
    vector<bool> completed(n, false);
    int completedProcesses = 0;

    while (completedProcesses < n) {
        int minBurst = INT_MAX;
        int selectedProcess = -1;

        // Find the process with the shortest burst time among the available
processes
        for (int i = 0; i < n; ++i) {
```

```
            if (!completed[i] && processes[i].arrival <= currentTime &&
processes[i].burst < minBurst) {
                minBurst = processes[i].burst;
                selectedProcess = i;
            }
        }

        // If no process is found, increment the current time
        if (selectedProcess == -1) {
            currentTime++;
            continue;
        }

        // Execute the selected process
        processes[selectedProcess].start = currentTime;
        processes[selectedProcess].finish = currentTime +
processes[selectedProcess].burst;
        processes[selectedProcess].waiting = processes[selectedProcess].start -
processes[selectedProcess].arrival;
        processes[selectedProcess].turnaround = processes[selectedProcess].finish
- processes[selectedProcess].arrival;

        totalWaitingTime += processes[selectedProcess].waiting;
        totalTurnaroundTime += processes[selectedProcess].turnaround;

        currentTime = processes[selectedProcess].finish;
        completed[selectedProcess] = true;
        completedProcesses++;
    }

    // Find the maximum burst time for consistent column width
    int maxBurst = 0;
    for (const auto& p : processes) {
        if (p.burst > maxBurst) {
            maxBurst = p.burst;
        }
    }
```

```
    // Display Gantt chart
    cout << "\nGantt Chart:\n";
    // Top border
    cout << "+";
    for (const auto& p : processes) {
        cout << setw(maxBurst) << setfill('-') << "" << "+";
    }
    cout << setfill(' ') << endl;

    // Process IDs
    cout << "|";
    for (const auto& p : processes) {
        cout << setw(maxBurst) << p.id << "|";
    }
    cout << endl;

    // Middle border
    cout << "+";
    for (const auto& p : processes) {
        cout << setw(maxBurst) << setfill('-') << "" << "+";
    }
    cout << setfill(' ') << endl;

    // Time line
    cout << "0";
    for (const auto& p : processes) {
        cout << setw(maxBurst) << right << p.finish;
    }
    cout << endl;

    // Display process information and average calculations
    cout << "\nProcess Information:\n";
    cout << "------------------------------------------------------------------" <<
endl;
    cout << left << setw(10) << "Process" << "|" << setw(15) << "A.T"
         << "|" << setw(15) << "W.T" << "|" << setw(20) << "T.A.T" << "|" << endl;
    cout << "------------------------------------------------------------------" <<
endl;
```

```
    for (const auto& p : processes) {
        cout << left << setw(10) << p.id
             << "|" << setw(15) << p.arrival
             << "|" << setw(15) << p.waiting
             << "|" << setw(20) << p.turnaround << "|" << endl;
    }

    cout << "------------------------------------------------------------------" <<
endl;

    cout << "\nAverage Waiting Time: " << fixed << setprecision(2) <<
(float)totalWaitingTime / n << endl;
    cout << "Average Turnaround Time: " << fixed << setprecision(2) <<
(float)totalTurnaroundTime / n << endl;
}

int main() {
    int numProcesses;
    char type;

    cout << "\n<<Shortest Job First (SJF) Scheduling Algorithm>>\n" << endl;
    cout << "Compiled by Suman Bisunkhe\n" << endl;
    cout << "Enter the type of Process IDs ('i' for integers, 'c' for characters):
";
    cin >> type;

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

    if (type != 'i' && type != 'c') {
        cout << "Invalid type. Exiting program." << endl;
        return 1;
    }

    if (type == 'i') {
        vector<Process<int>> processes(numProcesses);
        cout << "\nEnter the process ID, arrival time, and burst time for each
process (all in one line):\n";
```

```
        for (int i = 0; i < numProcesses; ++i) {
            cout << "For Process " << i + 1 << ": ";
            cin >> processes[i].id >> processes[i].arrival >> processes[i].burst;
        }

        calculateSJF(processes);
    } else if (type == 'c') {
        vector<Process<char>> processes(numProcesses);
        cout << "\nEnter the process ID (character), arrival time, and burst time
for each process (all in one line):\n";
        for (int i = 0; i < numProcesses; ++i) {
            cout << "For Process " << i + 1 << ": ";
            cin >> processes[i].id >> processes[i].arrival >> processes[i].burst;
        }

        calculateSJF(processes);
    }

    return 0;
}
```

OUTPUT

```
<<Shortest Job First (SJF) Scheduling Algorithm>>

Compiled by Suman Bisunkhe

Enter the type of Process IDs ('i' for integers, 'c' for characters): i
Enter the number of processes: 4

Enter the process ID, arrival time, and burst time for each process (all in one line):
For Process 1: 1 0 3
For Process 2: 2 2 6
For Process 3: 3 4 4
For Process 4: 4 5 5

Gantt Chart:
+------+------+------+------+
|     1|     2|     3|     4|
+------+------+------+------+
0     3     9    13    18

Process Information:
-----------------------------------------------------------------
Process   |A.T            |W.T            |T.A.T               |
-----------------------------------------------------------------
1         |0              |0              |3                   |
2         |2              |1              |7                   |
3         |4              |5              |9                   |
4         |5              |8              |13                  |
-----------------------------------------------------------------

Average Waiting Time: 3.50
Average Turnaround Time: 8.00
```

...for a complete human being
Texas
International college

LAB NO: 10 **DATE: 2024/08/06**

Write a program to calculate average Turn Around Time, and Waiting Time for user input processes using Round Robin algorithm and their parameters. Also draw it's Gantt chart.

Round Robin (RR)

Round Robin (RR) a scheduling algorithm where each process is assigned a fixed time slice or quantum, and processes are executed in a cyclic order to ensure fair allocation of CPU time.

Turn Around Time

The total time taken from the submission of a process to its completion, including both the waiting time and the execution time of the process.

Waiting Time

The total time a process spends waiting in the ready queue before it starts execution, excluding the time it spends actually being executed.

Algorithm

```
// Round Robin algorithm (RR)

1) Start:

      1. Initialize current_time to 0.

2) Process each process in the queue:

      1. For each process:

            (a) Set arrival_time and burst_time.

            (b) Compute waiting_time as current_time - arrival_time.

            (c) Compute turnaround_time as waiting_time + burst_time.

            (d) Update current_time by adding burst_time to it.

3) End:

      1. Print the waiting_time and turnaround_time for each process.
```

SOURCE CODE

```
// Round Robin algorithm (RR)


#include <iostream>
#include <vector>
#include <iomanip>
#include <climits>
#include <string>

using namespace std;

template <typename T>
struct Process {
    T id;           // Process ID
    int arrival;    // Arrival Time
    int burst;      // Burst Time
    int remaining;  // Remaining Burst Time
    int start;      // Start Time
    int finish;     // Finish Time
    int waiting;    // Waiting Time
    int turnaround; // Turnaround Time
};

template <typename T>
void calculateRoundRobin(vector<Process<T>>& processes, int quantum) {
    int n = processes.size();
    int totalWaitingTime = 0, totalTurnaroundTime = 0;
    int currentTime = 0;
    vector<bool> completed(n, false);
    int completedProcesses = 0;
    vector<int> ganttChart;
    vector<int> ganttTimes;

    // Initialize remaining times
    for (auto& p : processes) {
        p.remaining = p.burst;
    }

    while (completedProcesses < n) {
        bool progressMade = false;
```

```
        for (int i = 0; i < n; ++i) {
            if (!completed[i] && processes[i].arrival <= currentTime) {
                if (processes[i].remaining > 0) {
                    progressMade = true;

                    int timeSlice = min(quantum, processes[i].remaining);
                    processes[i].remaining -= timeSlice;
                    currentTime += timeSlice;
                    ganttChart.push_back(processes[i].id);
                    ganttTimes.push_back(currentTime);

                    if (processes[i].remaining == 0) {
                        processes[i].finish = currentTime;
                        processes[i].turnaround = processes[i].finish -
processes[i].arrival;
                        processes[i].waiting = processes[i].turnaround -
processes[i].burst;

                        totalWaitingTime += processes[i].waiting;
                        totalTurnaroundTime += processes[i].turnaround;
                        completed[i] = true;
                        completedProcesses++;
                    }

                    // Mark start time for the first slice
                    if (processes[i].remaining == processes[i].burst - timeSlice) {
                        processes[i].start = currentTime - timeSlice;
                    }
                }
            }
        }

        if (!progressMade) {
            currentTime++;
        }
    }

    // Display Gantt chart
    cout << "\nGantt Chart:\n";
    // Top border
    cout << "+";
    for (const auto& id : ganttChart) {
```

```
        cout << "-----+";
    }
    cout << endl;

    // Process IDs
    cout << "|";
    for (const auto& id : ganttChart) {
        cout << "  " << id << "  |";
    }
    cout << endl;

    // Middle border
    cout << "+";
    for (const auto& id : ganttChart) {
        cout << "-----+";
    }
    cout << endl;

    // Time line
    cout << 0;
    for (const auto& time : ganttTimes) {
        cout << setw(6) << right << time;
    }
    cout << endl;

    // Display process information and average calculations
    cout << "\nProcess Information:\n";
    cout << "------------------------------------------------------------------" << endl;
    cout << left << setw(10) << "Process" << "|" << setw(15) << "A.T"
         << "|" << setw(15) << "W.T" << "|" << setw(20) << "T.A.T" << "|" << endl;
    cout << "------------------------------------------------------------------" << endl;
    for (const auto& p : processes) {
        cout << left << setw(10) << p.id
             << "|" << setw(15) << p.arrival
             << "|" << setw(15) << p.waiting
             << "|" << setw(20) << p.turnaround << "|" << endl;
    }

    cout << "------------------------------------------------------------------" << endl;

    cout << "\nAverage Waiting Time: " << fixed << setprecision(2) <<
(float)totalWaitingTime / n << endl;
```

```
    cout << "Average Turnaround Time: " << fixed << setprecision(2) <<
(float)totalTurnaroundTime / n << endl;
}

int main() {
    int numProcesses;
    int timeQuantum;
    char type;

    cout << "\n<<Round Robin Scheduling Algorithm>>\n" << endl;
    cout << "Compiled by Suman Bisunkhe\n" << endl;

    cout << "Enter the type of Process IDs ('i' for integers, 'c' for characters): ";
    cin >> type;

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

    if (type != 'i' && type != 'c') {
        cout << "Invalid type. Exiting program." << endl;
        return 1;
    }
    cout << "Enter the time quantum: ";
    cin >> timeQuantum;

    if (type == 'i') {
        vector<Process<int>> processes(numProcesses);
        cout << "\nEnter the process ID, arrival time, and burst time for each process (all
in one line):\n";
        for (int i = 0; i < numProcesses; ++i) {
            cout << "For Process " << i + 1 << ": ";
            cin >> processes[i].id >> processes[i].arrival >> processes[i].burst;
        }

        calculateRoundRobin(processes, timeQuantum);
    } else if (type == 'c') {
        vector<Process<char>> processes(numProcesses);
        cout << "\nEnter the process ID (character), arrival time, and burst time for each
process (all in one line):\n";
        for (int i = 0; i < numProcesses; ++i) {
            cout << "For Process " << i + 1 << ": ";
            cin >> processes[i].id >> processes[i].arrival >> processes[i].burst;
```

```
        }

        calculateRoundRobin(processes, timeQuantum);
    }

    return 0;
}
```

OUTPUT

```
"C:\Users\sbisu\Documents\C

<<Round Robin Scheduling Algorithm>>

Compiled by Suman Bisunkhe

Enter the type of Process IDs ('i' for integers, 'c' for characters): i
Enter the number of processes: 3
Enter the time quantum: 2

Enter the process ID, arrival time, and burst time for each process (all in one line):
For Process 1: 1 0 10
For Process 2: 2 0 5
For Process 3: 3 0 8

Gantt Chart:
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|  1  |  2  |  3  |  1  |  2  |  3  |  1  |  2  |  3  |  1  |  3  |  1  |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
0     2     4     6     8    10    12    14    15    17    19    21    23

Process Information:
-----------------------------------------------------------------
Process   |A.T            |W.T            |T.A.T               |
-----------------------------------------------------------------
1         |0              |13             |23                  |
2         |0              |10             |15                  |
3         |0              |13             |21                  |
-----------------------------------------------------------------

Average Waiting Time: 12.00
Average Turnaround Time: 19.67
```

...for a complete human being
Texas International college

LAB NO: 11 **DATE: 2024/08/06**

Write a program to calculate average Turn Around Time, and Waiting Time for user input processes using Priority Scheduling and their parameters. Also draw it's Gantt chart.

Priority Scheduling

Priority Scheduling is a scheduling algorithm that assigns a priority to each process and selects the process with the highest priority for execution to ensure that critical tasks are completed first.

Turn Around Time

The total time taken from the submission of a process to its completion, including both the waiting time and the execution time of the process.

Waiting Time

The total time a process spends waiting in the ready queue before it starts execution, excluding the time it spends actually being executed.

Algorithm

```
// Priority Scheduling Algorithm

1) Start:

   a) Initialize current_time to 0.

2) For each process in the queue (sorted by priority):For each process:

   a. Set arrival_time, burst_time, and priority for the process.

   b. Calculate waiting_time as current_time - arrival_time.

   c. Calculate turnaround_time as waiting_time + burst_time.

   d. Update current_time by adding the burst_time of the process.

3) End:

   a) Print the waiting_time and turnaround_time for each process.
```

SOURCE CODE

```
// Priority Scheduling Algorithm

#include <iostream>
#include <vector>
#include <iomanip>
#include <algorithm>
#include <string>

using namespace std;

template <typename T>
struct Process {
    T id;           // Process ID
    int arrival;    // Arrival Time
    int burst;      // Burst Time
    int remaining;  // Remaining Burst Time
    int start;      // Start Time
    int finish;     // Finish Time
    int waiting;    // Waiting Time
    int turnaround; // Turnaround Time
    int priority;   // Priority of the process
};

// Function to sort processes by arrival time and then by priority
template <typename T>
bool compareProcesses(const Process<T>& a, const Process<T>& b) {
    if (a.arrival == b.arrival)
        return a.priority < b.priority;
    return a.arrival < b.arrival;
}

template <typename T>
void calculatePriorityScheduling(vector<Process<T>>& processes) {
    int n = processes.size();
    int totalWaitingTime = 0, totalTurnaroundTime = 0;
    int currentTime = 0;
    vector<bool> completed(n, false);
```

```
    int completedProcesses = 0;

    // Sort processes by arrival time and priority
    sort(processes.begin(), processes.end(), compareProcesses<T>);

    while (completedProcesses < n) {
        Process<T>* currentProcess = nullptr;

        // Select the highest priority process that has arrived and is not
completed
        for (int i = 0; i < n; ++i) {
            if (!completed[i] && processes[i].arrival <= currentTime) {
                if (currentProcess == nullptr || processes[i].priority <
currentProcess->priority) {
                    currentProcess = &processes[i];
                }
            }
        }

        if (currentProcess != nullptr) {
            currentProcess->remaining = currentProcess->burst;
            currentTime += currentProcess->remaining;
            currentProcess->finish = currentTime;
            currentProcess->turnaround = currentProcess->finish - currentProcess-
>arrival;
            currentProcess->waiting = currentProcess->turnaround - currentProcess-
>burst;

            totalWaitingTime += currentProcess->waiting;
            totalTurnaroundTime += currentProcess->turnaround;

            completed[currentProcess - &processes[0]] = true;
            completedProcesses++;
        } else {
            // No process is ready to run, move time forward
            currentTime++;
        }
    }
```

```
// Find the maximum width needed for the table columns
int maxIDWidth = 0;
int maxFinishWidth = 0;
int maxTimeWidth = 0;

for (const auto& p : processes) {
    int idWidth = to_string(p.id).length();
    int finishWidth = to_string(p.finish).length();
    maxIDWidth = max(maxIDWidth, idWidth);
    maxFinishWidth = max(maxFinishWidth, finishWidth);
}

maxTimeWidth = max(maxIDWidth, maxFinishWidth) + 4; // extra space for padding

// Display Gantt chart
cout << "\nGantt Chart:\n";
// Top border
cout << "+";
for (int i = 0; i < processes.size(); ++i) {
    cout << setw(maxTimeWidth) << setfill('-') << "" << "+";
}
cout << setfill(' ') << endl;

// Process IDs
cout << "|";
for (const auto& p : processes) {
    cout << setw(maxTimeWidth) << p.id << "|";
}
cout << endl;

// Middle border
cout << "+";
for (int i = 0; i < processes.size(); ++i) {
    cout << setw(maxTimeWidth) << setfill('-') << "" << "+";
}
cout << setfill(' ') << endl;
```

```
    // Time line
    cout << "0";
    for (const auto& p : processes) {
        cout << setw(maxTimeWidth) << right << p.finish;
    }
    cout << endl;

    // Display process information and average calculations
    cout << "\nProcess Information:\n";
    cout << "-----------------------------------------------------------------" <<
endl;
    cout << left << setw(10) << "Process" << "|" << setw(15) << "A.T"
         << "|" << setw(15) << "W.T" << "|" << setw(20) << "T.A.T" << "|" << endl;
    cout << "-----------------------------------------------------------------" <<
endl;
    for (const auto& p : processes) {
        cout << left << setw(10) << p.id
             << "|" << setw(15) << p.arrival
             << "|" << setw(15) << p.waiting
             << "|" << setw(20) << p.turnaround << "|" << endl;
    }

    cout << "-----------------------------------------------------------------" <<
endl;

    cout << "\nAverage Waiting Time: " << fixed << setprecision(2) <<
(float)totalWaitingTime / n << endl;
    cout << "Average Turnaround Time: " << fixed << setprecision(2) <<
(float)totalTurnaroundTime / n << endl;
}

int main() {
    int numProcesses;
    char type;

    cout << "\n<<Priority Scheduling Algorithm>>\n" << endl;
    cout << "Compiled by Suman Bisunkhe\n" << endl;
```

```
    cout << "Enter the type of Process IDs ('i' for integers, 'c' for characters):
";
    cin >> type;

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

    if (type != 'i' && type != 'c') {
        cout << "Invalid type. Exiting program." << endl;
        return 1;
    }

    if (type == 'i') {
        vector<Process<int>> processes(numProcesses);
        cout << "\nEnter the process ID, arrival time, burst time, and priority
for each process (all in one line):\n";
        for (int i = 0; i < numProcesses; ++i) {
            cout << "For Process " << i + 1 << ": ";
            cin >> processes[i].id >> processes[i].arrival >> processes[i].burst
>> processes[i].priority;
        }

        calculatePriorityScheduling(processes);
    } else if (type == 'c') {
        vector<Process<char>> processes(numProcesses);
        cout << "\nEnter the process ID (character), arrival time, burst time, and
priority for each process (all in one line):\n";
        for (int i = 0; i < numProcesses; ++i) {
            cout << "For Process " << i + 1 << ": ";
            cin >> processes[i].id >> processes[i].arrival >> processes[i].burst
>> processes[i].priority;
        }

        calculatePriorityScheduling(processes);
    }

    return 0;
}
```

OUTPUT

```
<<Priority Scheduling Algorithm>>

Compiled by Suman Bisunkhe

Enter the type of Process IDs ('i' for integers, 'c' for characters): i
Enter the number of processes: 4

Enter the process ID, arrival time, burst time, and priority for each process (all in one line):
For Process 1: 1 0 11 2
For Process 2: 2 5 28 0
For Process 3: 3 12 2 3
For Process 4: 4 2 10 1

Gantt Chart:
+------+------+------+------+
|     1|     4|     2|     3|
+------+------+------+------+
0    11    49    39    51

Process Information:
------------------------------------------------------------
Process   |A.T            |W.T            |T.A.T              |
------------------------------------------------------------
1         |0              |0              |11                 |
4         |2              |37             |47                 |
2         |5              |6              |34                 |
3         |12             |37             |39                 |
------------------------------------------------------------

Average Waiting Time: 20.00
Average Turnaround Time: 32.75
```