MBM
Madan Bhandari Memorial College

Lab No. 1 **Date: 2025/08/12**

Title: Write a program to calculate the number of page fault for user input references string and frame size using FIFO page replacement algorithm.

In operating systems, memory management plays a crucial role in ensuring efficient execution of processes. When a process is executed, it requires pages to be loaded into main memory. However, the number of available frames in main memory is limited. If a page required by a process is not present in memory, a page fault occurs, and the operating system must bring the page from secondary storage into main memory.

Page Replacement Algorithms decide which page in memory should be replaced when a new page needs to be loaded and the memory is already full.

The FIFO (First-In-First-Out) Page Replacement Algorithm is one of the simplest page replacement strategies. It replaces the oldest page in memory the one that was loaded first regardless of how frequently or recently it has been accessed.

Frame Size: Frame size is the fixed size of a block of physical memory where pages are loaded, typically matching the page size to avoid fragmentation.

Page Fault: A page fault occurs when a program attempts to access a page not currently in physical memory, requiring the operating system to load the page from secondary storage into RAM.

Algorithm:

```
1) Initialize an empty queue with a fixed size.

2) For each page request.

        • If page is in memory, continue

        • If page is not in memory i.e page fault occurs:

        → If there is space in memory, insert the new page into the queue.

        → If memory is full, remove the page at the front of the queue (oldest) and insert the new page
          at the rear and increment the page fault.

3) Output the total number of page faults.
```

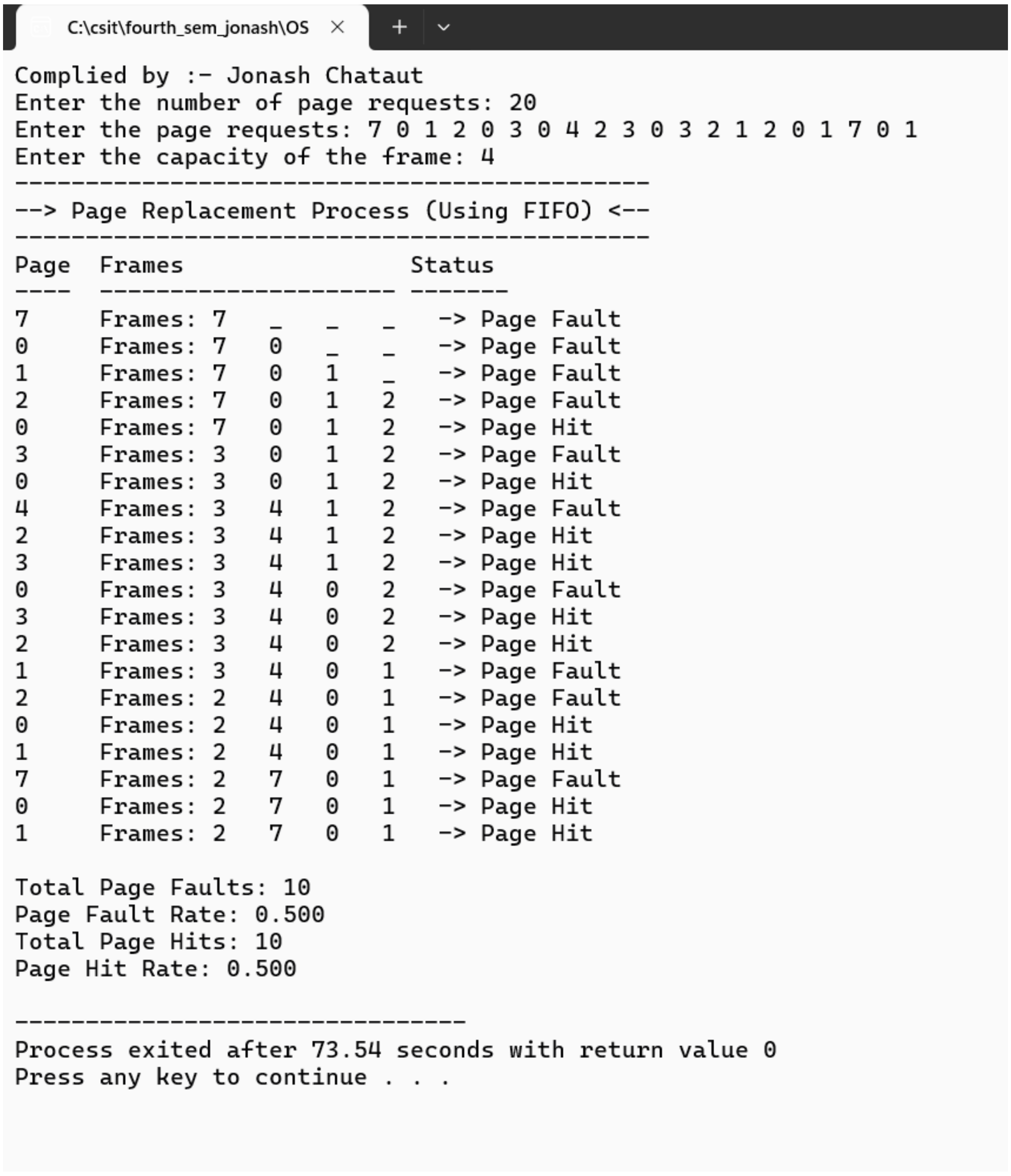
Programming Language: C

IDE: Dev C++

Source Code:

```
#include <iostream>
#include <iomanip>
using namespace std;
// Seach the page
int search(int frames[], int n, int page)
{
   for (int i = 0; i < n; i++)
   {
      if (frames[i] == page)
      {
         return i; // Page found in frame
      }
   }
   return -1; // Page not found in frame
}
void printFrames(int frames[], int frame_size, bool page_hit)
{
   cout << "Frames: ";
   for (int i = 0; i < frame_size; i++)
      if (frames[i] == -1)
         // cout << " _ ";
         cout << setw(4) << "_";
      else
         cout << setw(4) << frames[i];
   cout << setw(22) << (page_hit ? "-> Page Hit" : "-> Page Fault") << "\n";
}
void fifo(int pages[], int n, int capacity)
{
   int frames[capacity];
   int faults = 0;
   int nextFrame = 0; // Index of the next frame to be replaced

   // Initialize frames with -1 to indicate an empty frame
   for (int i = 0; i < capacity; i++)
   {
      frames[i] = -1;
   }
   cout << "--------------------------------------------\n";
   cout << "--> Page Replacement Process (Using FIFO) <--\n";
   cout << "--------------------------------------------\n";
   cout << setw(6) << left << "Page" << setw(22) << "Frames" << "Status\n";
   cout << setw(6) << left << "----" << setw(22) << "---------------------" << "-------\n";
   for (int i = 0; i < n; i++)
   {
      int page = pages[i];
      cout << setw(6) << left << page;
      bool page_hit;
      if (search(frames, capacity, page) == -1)
      {
            // Page fault
            frames[nextFrame] = page;
            nextFrame = (nextFrame + 1) % capacity;
            faults++;
            // cout << "Page Fault!\n";
            page_hit = false;
        }
        else
        {
            page_hit = true;
        }
        printFrames(frames, capacity, page_hit);
    }
    cout << "\nTotal Page Faults: " << faults << "\n";
    cout << "Page Fault Rate: " << fixed << setprecision(3) << (float)faults / n << "\n";
    cout << "Total Page Hits: " << n - faults << "\n";
    cout << "Page Hit Rate: " << fixed << setprecision(3) << (float)(n - faults) / n << "\n";
}

int main()
{
    cout << "Complied by :- Jonash Chataut\n";
    int np;      // Number of page requests
    int capacity; // Capacity of the frame
    cout << "Enter the number of page requests: ";
    cin >> np;
    if (np <= 0)
    {
        cout << "Error: Invalid number of page requests.\n";
        return 0;
    }
    int pages[np];
    cout << "Enter the page requests: ";
    for (int i = 0; i < np; i++)
    {
        cin >> pages[i];
    }
    cout << "Enter the capacity of the frame: ";
    cin >> capacity;
    if (capacity <= 0)
    {
        cout << "Error: Invalid capacity.\n";
        return 0;
    }
    fifo(pages, np, capacity);

    return 0;
}
```

Output:



```
Complied by :- Jonash Chataut
Enter the number of page requests: 20
Enter the page requests: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter the capacity of the frame: 4
---------------------------------------------
--> Page Replacement Process (Using FIFO) <--
---------------------------------------------
Page  Frames                 Status
----  --------------------- -------
7     Frames: 7   _   _   _   -> Page Fault
0     Frames: 7   0   _   _   -> Page Fault
1     Frames: 7   0   1   _   -> Page Fault
2     Frames: 7   0   1   2   -> Page Fault
0     Frames: 7   0   1   2   -> Page Hit
3     Frames: 3   0   1   2   -> Page Fault
0     Frames: 3   0   1   2   -> Page Hit
4     Frames: 3   4   1   2   -> Page Fault
2     Frames: 3   4   1   2   -> Page Hit
3     Frames: 3   4   1   2   -> Page Hit
0     Frames: 3   4   0   2   -> Page Fault
3     Frames: 3   4   0   2   -> Page Hit
2     Frames: 3   4   0   2   -> Page Hit
1     Frames: 3   4   0   1   -> Page Fault
2     Frames: 2   4   0   1   -> Page Fault
0     Frames: 2   4   0   1   -> Page Hit
1     Frames: 2   4   0   1   -> Page Hit
7     Frames: 2   7   0   1   -> Page Fault
0     Frames: 2   7   0   1   -> Page Hit
1     Frames: 2   7   0   1   -> Page Hit

Total Page Faults: 10
Page Fault Rate: 0.500
Total Page Hits: 10
Page Hit Rate: 0.500

--------------------------------
Process exited after 73.54 seconds with return value 0
Press any key to continue . . .
```

Lab No. 2 **Date: 2025/08/12**

Title: Write a program to calculate the number of page fault for user input references string and frame size using OPR page replacement algorithm.

The Optimal Page Replacement (OPR) Algorithm is a theoretical page replacement strategy that achieves the minimum possible number of page faults. It works by replacing the page that will not be used for the longest period of time in the future. Since it requires future knowledge of the page reference string, OPR cannot be implemented in a real operating system but is used for comparison and analysis purposes.

In OPR, when a page fault occurs and the memory is full, the algorithm scans the upcoming page references to determine which page in memory will be used farthest in the future (or not at all). That page is then replaced with the new page.

Algorithm:

1. For each page request:

- If the page is in memory, continue.

- If the page is not in memory:

• If memory is full, determine which page will be used the furthest in the

future (or not at all) and remove it.

• Add the new page to memory.

• Increment the page faults.

2. Output the total number of page faults.

Programming Language: C

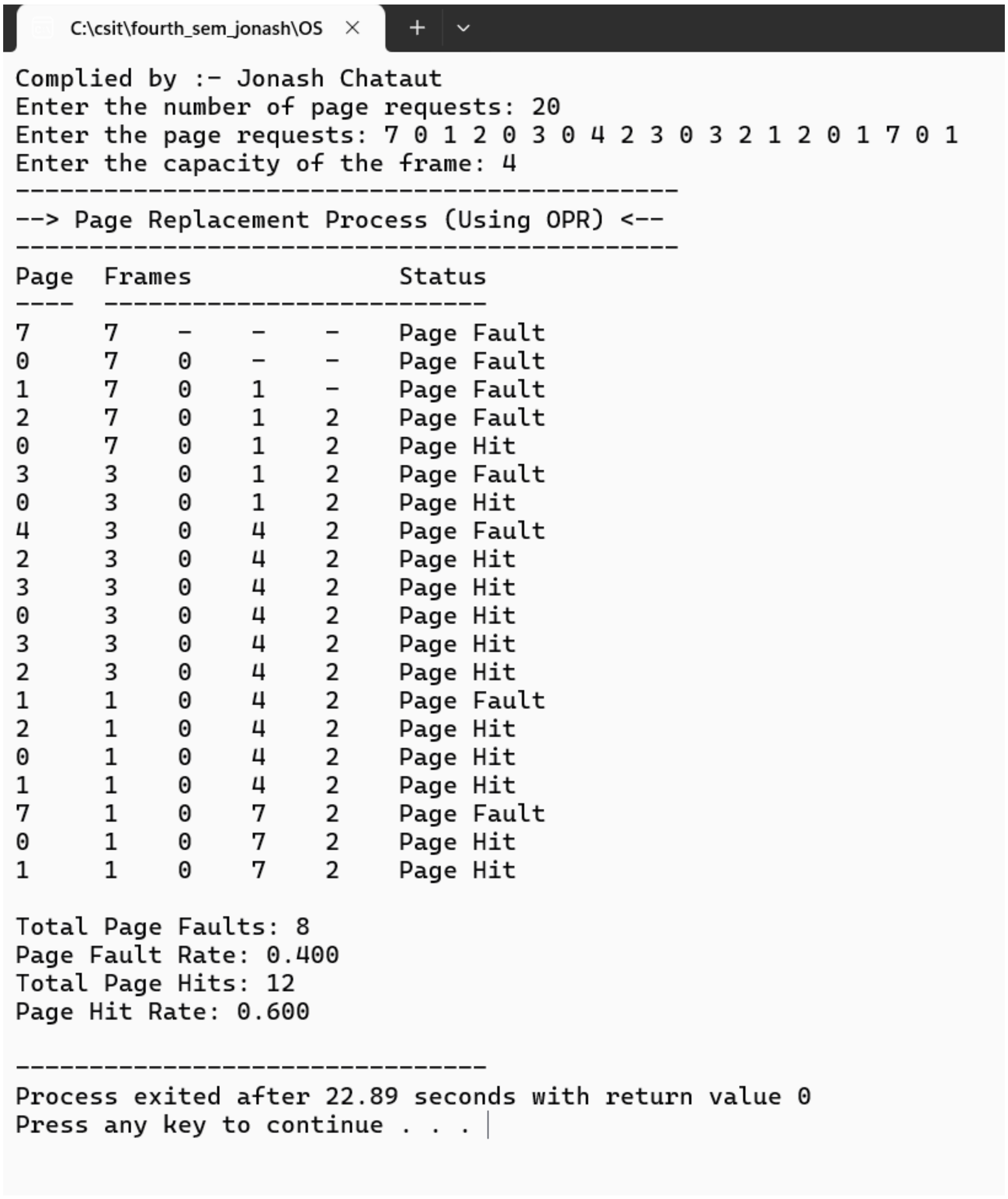
IDE: Dev C++

Source Code:

```
#include <iostream>
#include <iomanip>
using namespace std;
bool search(int frames[], int n, int page)
{
   for (int i = 0; i < n; i++)
   {
      if (frames[i] == page)
         return true;
   }
   return false;
}
int predict(int ref_str[], int frames[], int refStrLen, int index, int frame_occupied)
{
   int result = -1;
   int farthest = index;
   for (int i = 0; i < frame_occupied; i++)
   {
      int j;
      for (j = index; j < refStrLen; j++)
      {
         if (frames[i] == ref_str[j])
         {
            if (j > farthest)
            {
               farthest = j;
               result = i;
            }
            break;
         }
      }
      if (j == refStrLen)
         return i;
   }
   return (result == -1) ? 0 : result;
}
void printFrames(int frames[], int frame_size, int frame_occupied, bool page_hit)
{
   for (int i = 0; i < frame_size; i++)
   {
      if (i < frame_occupied)
         cout << left << setw(5) << frames[i];
      else
         cout << left << setw(5) << "-";
   }
   cout << left << setw(15) << (page_hit ? "Page Hit" : "Page Fault") << "\n";
}
void optimalPage(int pages[], int np, int capacity)
{
   int frames[capacity];
```

```
    int frame_occupied = 0;
    int faults = 0;
    cout << "---------------------------------------------\n";
    cout << "--> Page Replacement Process (Using OPR) <--\n";
    cout << "---------------------------------------------\n";
    cout << setw(6) << left << "Page" << setw(5 * capacity) << left << "Frames" << "Status\n";
    cout << setw(6) << left << "----";
    for (int i = 0; i < capacity; i++)
        cout << setw(5) << "-----";
    cout << "------\n";
    for (int i = 0; i < np; i++)
    {
        int page = pages[i];
        cout << setw(6) << left << page;
        bool page_hit = false;
        if (search(frames, frame_occupied, page))
        {
            page_hit = true;
        }
        else
        {
            // Page fault
            faults++;
            if (frame_occupied < capacity)
            {
                frames[frame_occupied++] = page;
            }
            else
            {
                int pos = predict(pages, frames, np, i + 1, frame_occupied);
                frames[pos] = page;
            }
        }
        printFrames(frames, capacity, frame_occupied, page_hit);
    }
    cout << "\nTotal Page Faults: " << faults << "\n";
    cout << "Page Fault Rate: " << fixed << setprecision(3) << (float)faults / np << "\n";
    cout << "Total Page Hits: " << np - faults << "\n";
    cout << "Page Hit Rate: " << fixed << setprecision(3) << (float)(np - faults) / np << "\n";
}
int main()
{
    cout << "Complied by :- Jonash Chataut\n";
    int np, capacity;
    cout << "Enter the number of page requests: ";
    cin >> np;
    if (np <= 0)
    {
        cout << "Error: Invalid number of page requests.\n";
        return 0;
    }
    int pages[np];
    cout << "Enter the page requests: ";
    for (int i = 0; i < np; i++)
    {
      cin >> pages[i];
    }
    cout << "Enter the capacity of the frame: ";
    cin >> capacity;
    if (capacity <= 0)
    {
      cout << "Error: Invalid capacity.\n";
      return 0;
    }
    optimalPage(pages, np, capacity);
    return 0;
}
```

Output:



C:\csit\fourth_sem_jonash\OS

```
Complied by :- Jonash Chataut
Enter the number of page requests: 20
Enter the page requests: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter the capacity of the frame: 4
---------------------------------------------
--> Page Replacement Process (Using OPR) <--
---------------------------------------------
Page  Frames                  Status
----  --------------------------
7     7    -    -    -    Page Fault
0     7    0    -    -    Page Fault
1     7    0    1    -    Page Fault
2     7    0    1    2    Page Fault
0     7    0    1    2    Page Hit
3     3    0    1    2    Page Fault
0     3    0    1    2    Page Hit
4     3    0    4    2    Page Fault
2     3    0    4    2    Page Hit
3     3    0    4    2    Page Hit
0     3    0    4    2    Page Hit
3     3    0    4    2    Page Hit
2     3    0    4    2    Page Hit
1     1    0    4    2    Page Fault
2     1    0    4    2    Page Hit
0     1    0    4    2    Page Hit
1     1    0    4    2    Page Hit
7     1    0    7    2    Page Fault
0     1    0    7    2    Page Hit
1     1    0    7    2    Page Hit

Total Page Faults: 8
Page Fault Rate: 0.400
Total Page Hits: 12
Page Hit Rate: 0.600

--------------------------------
Process exited after 22.89 seconds with return value 0
Press any key to continue . . .
```

Madan Bhandari Memorial College

Lab No. 3 **Date: 2025/08/12**

Title: Write a program to calculate the number of page fault for user input references string and frame size using LRU page replacement algorithm.

The Least Recently Used (LRU) Page Replacement Algorithm is based on the principle of replacing the page that has not been used for the longest period of time in the past. The assumption is that pages used recently are more likely to be used again soon, while pages not used for a long time are less likely to be needed.

When a page fault occurs and the memory is full, LRU scans the usage history of the pages currently in memory and selects the one with the longest idle time (least recently used) for replacement.

Algorithm:

```
1. Initialize an empty list to keep track of the pages in memory and their last used time.
2. For each page request:
    - If the page is in memory, update its last used time.
    - If the page is not in memory:
        - If memory is full, remove the page that has not been used for the longest time.
        - Add the new page to the memory.
        - Increment the page faults.
3. Output the total number of page faults.
```

Programming Language: C

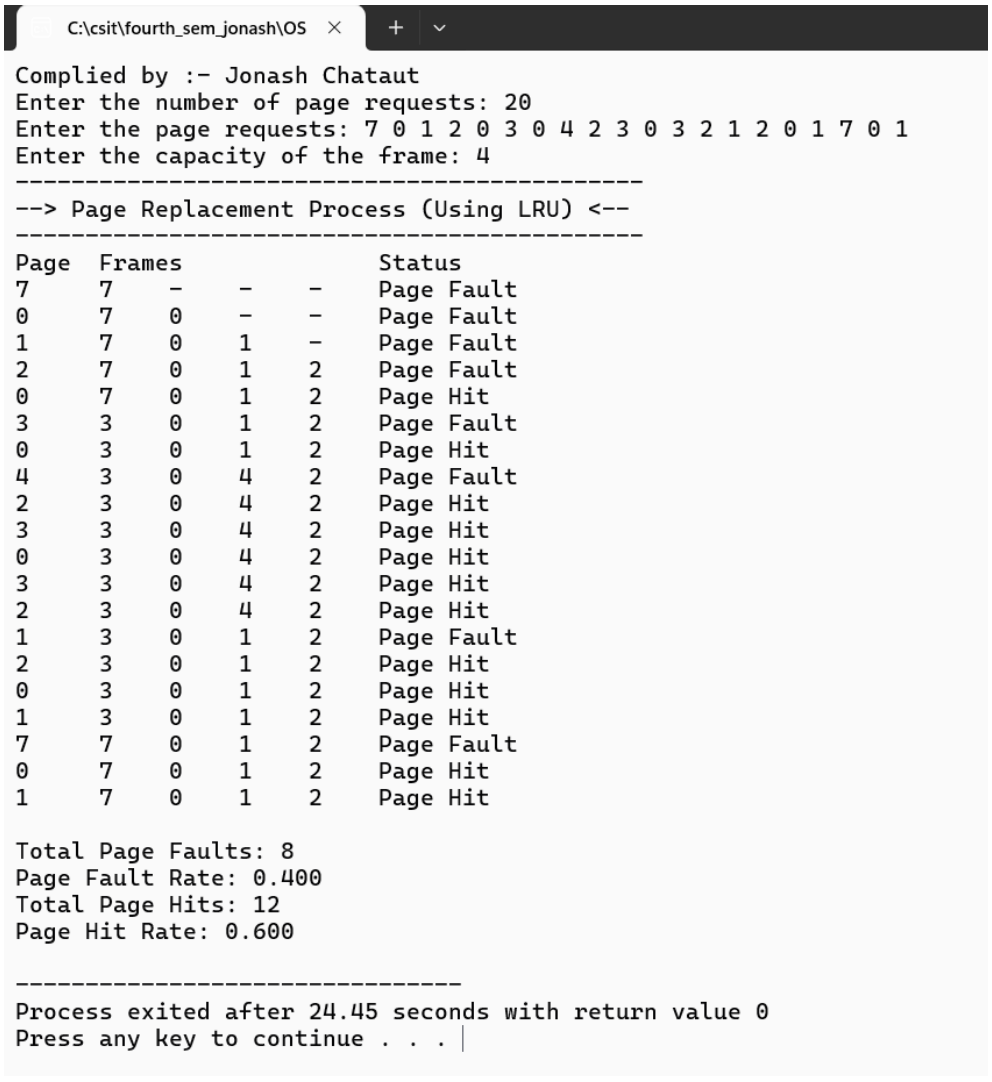
IDE: Dev C++

Source Code:

```
#include <iostream>
#include <iomanip>
using namespace std;
// function to check pagehit
bool isPageHit(int frames[], int n, int page)
{
   for (int i = 0; i < n; i++)
      if (frames[i] == page)
         return true;
   return false;
}
// function to get index of least used page
int getLRUPageIndex(int count[], int frameCount)
{
   int minIndex = 0;
   for (int i = 1; i < frameCount; i++)
   {
      if (count[i] < count[minIndex])
         minIndex = i;
   }
   return minIndex;
}
void printFrames(int frames[], int frame_size, int frame_occupied, bool page_hit)
{
   for (int i = 0; i < frame_size; i++)
   {
      if (i < frame_occupied)
         cout << setw(5) << frames[i];
      else
         cout << setw(5) << "-";
   }
   cout << setw(15) << (page_hit ? "Page Hit" : "Page Fault") << "\n";
}
void lru(int pages[], int np, int capacity)
{
   int frames[capacity];   // stores pages in frames
   int count[capacity];   // stores last used time for each frame
   int frame_occupied = 0; // how many frames currently occupied
   int pageFaults = 0;     // count page faults
   int time = 0;          // global counter to track usage
   // Initialize frames as empty (-1) and count as 0
   for (int i = 0; i < capacity; i++)
   {
      frames[i] = -1;
      count[i] = 0;
   }
   // Print headers for output table
   cout << "--------------------------------------------\n";
   cout << "--> Page Replacement Process (Using LRU) <--\n";
   cout << "--------------------------------------------\n";
```

```
    cout << setw(6) << left << "Page" << setw(5 * capacity) << left << "Frames" << "Status\n";
    for (int i = 0; i < np; i++)
    {
        int page = pages[i];
        cout << setw(6) << left << page;
        // Check if page is already in frames (page hit)
        bool page_hit = isPageHit(frames, frame_occupied, page);
        if (!page_hit)
        {
            // Page fault occurred so incrementing
            pageFaults++;
            if (frame_occupied < capacity)
            {
                // Still space in frames so adding page directly
                frames[frame_occupied] = page;
                count[frame_occupied] = time++; // record last used time
                frame_occupied++;
            }
            else
            {
                // Need to replace the LRU page
                int lruIndex = getLRUPageIndex(count, capacity);
                frames[lruIndex] = page;
                count[lruIndex] = time++;
            }
        }
        else
        {
            // Page hit: update last used time of that frame
            for (int j = 0; j < frame_occupied; j++)
            {
                if (frames[j] == page)
                {
                    count[j] = time++;
                    break;
                }
            }
        }

        // Print current state of frames and status
        printFrames(frames, capacity, frame_occupied, page_hit);
    }
    // Print stats
    cout << "\nTotal Page Faults: " << pageFaults << "\n";
    cout << "Page Fault Rate: " << fixed << setprecision(3) << (float)pageFaults / np << "\n";
    cout << "Total Page Hits: " << np - pageFaults << "\n";
    cout << "Page Hit Rate: " << fixed << setprecision(3) << (float)(np - pageFaults) / np << "\n";
}
int main()
{
    cout << "Complied by :- Jonash Chataut\n";
    int np, capacity;
    cout << "Enter the number of page requests: ";
    cin >> np;
    if (np <= 0)
    {
      cout << "Invalid number of page requests.\n";
      return 0;
    }
    int pages[np];
    cout << "Enter the page requests: ";
    for (int i = 0; i < np; i++)
    {
      cin >> pages[i];
    }
    cout << "Enter the capacity of the frame: ";
    cin >> capacity;
    if (capacity <= 0)
    {
      cout << "Invalid capacity.\n";
      return 0;
    }
    lru(pages, np, capacity);
    return 0;
  }
```

Output:



C:\csit\fourth_sem_jonash\OS

```
Complied by :- Jonash Chataut
Enter the number of page requests: 20
Enter the page requests: 7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
Enter the capacity of the frame: 4
---------------------------------------------
--> Page Replacement Process (Using LRU) <--
---------------------------------------------
Page  Frames              Status
7     7    -    -    -    Page Fault
0     7    0    -    -    Page Fault
1     7    0    1    -    Page Fault
2     7    0    1    2    Page Fault
0     7    0    1    2    Page Hit
3     3    0    1    2    Page Fault
0     3    0    1    2    Page Hit
4     3    0    4    2    Page Fault
2     3    0    4    2    Page Hit
3     3    0    4    2    Page Hit
0     3    0    4    2    Page Hit
3     3    0    4    2    Page Hit
2     3    0    4    2    Page Hit
1     3    0    1    2    Page Fault
2     3    0    1    2    Page Hit
0     3    0    1    2    Page Hit
1     3    0    1    2    Page Hit
7     7    0    1    2    Page Fault
0     7    0    1    2    Page Hit
1     7    0    1    2    Page Hit

Total Page Faults: 8
Page Fault Rate: 0.400
Total Page Hits: 12
Page Hit Rate: 0.600

--------------------------------
Process exited after 24.45 seconds with return value 0
Press any key to continue . . .
```

Madan Bhandari Memorial College

Lab No. 4 **Date: 2025/08/14**

Title: Write a program to display the process allocation for user input free blocks and incoming process using Best Fit Allocation.

Memory allocation is the process of assigning memory blocks to processes from the available free memory space. When multiple free blocks are available, different strategies can be used to decide which block to allocate to a process.

The Best Fit Allocation Algorithm searches for the smallest available block that is large enough to accommodate the process. It helps to minimize unused memory (internal fragmentation) by using the most appropriately sized block. However, it can lead to external fragmentation because small leftover blocks may not be usable later.

Steps in Best Fit Allocation:

a. For each process, scan the list of free memory blocks.

b. Choose the smallest block that is greater than or equal to the process size.

c. Allocate the process to that block and reduce the available block size.

d. If no suitable block is found, the process remains unallocated.

Programming Language: C

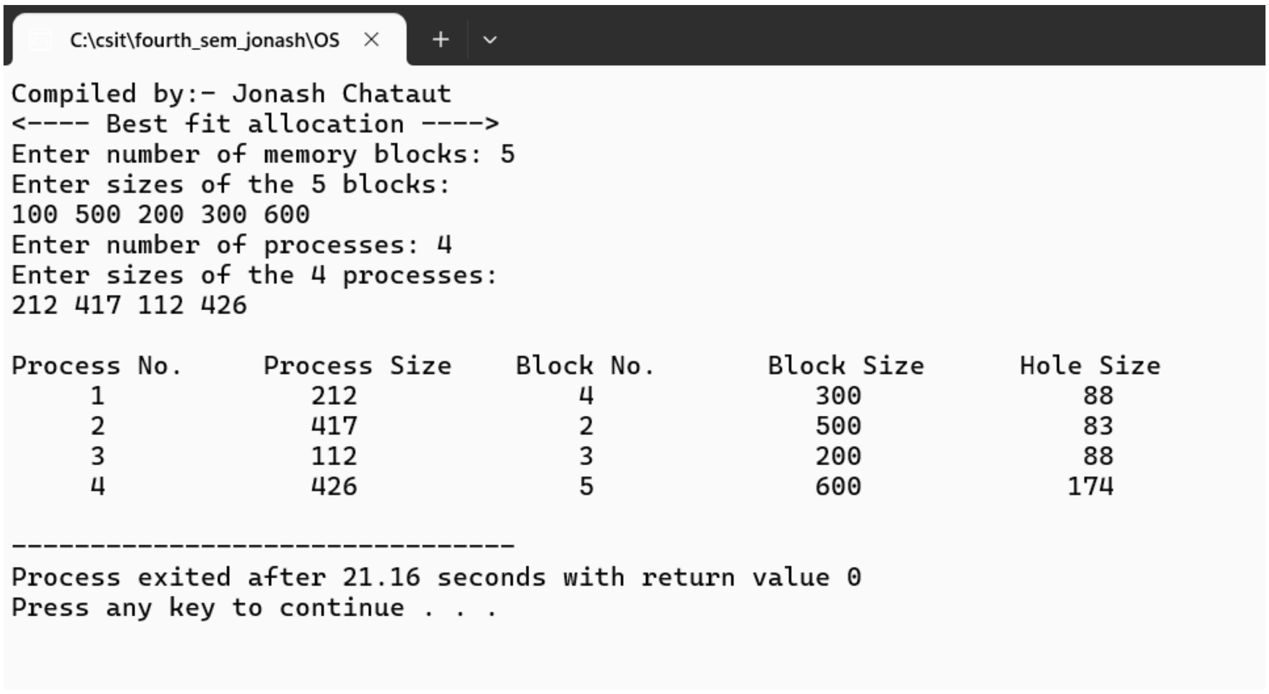
IDE: Dev C++

Source Code:

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

void bestFit(int blockSize[], int originalBlockSize[], int m, int processSize[], int n)
{
   int allocation[n];     // Stores block index for each process (-1 if not allocated)
   int holeAfterAlloc[n]; // Stores hole size immediately after allocation
   // Initialize arrays
   for (int i = 0; i < n; i++)
   {
      allocation[i] = -1;
      holeAfterAlloc[i] = -1;
   }
   // Allocate processes
   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 || blockSize[j] < blockSize[bestIdx])
               bestIdx = j;
         }
      }
      if (bestIdx != -1)
      {
         allocation[i] = bestIdx;
         blockSize[bestIdx] -= processSize[i];
         holeAfterAlloc[i] = blockSize[bestIdx]; // store immediate hole
      }
   }
   // Showing output results
   cout << "\nProcess No.\tProcess Size\tBlock No.\tBlock Size\tHole Size\n";

   for (int i = 0; i < n; i++)
   {
      cout << setw(6) << i + 1 << "\t\t"
          << setw(6) << processSize[i] << "\t\t";
      if (allocation[i] != -1)
      {
         int b = allocation[i];
         cout << setw(5) << b + 1 << "\t\t" << setw(6) << originalBlockSize[b] << "\t\t" << setw(6) <<
     holeAfterAlloc[i] << "\n";
      }
      else
      {
         cout << setw(5) << "NA" << "\t\t" << setw(6) << "-" << "\t\t" << setw(6) << "-" << "\n";
      }
```

```
  }
}

int main()
{
  cout << "Compiled by:- Jonash Chataut\n";
  cout << "<---- Best fit allocation ---->\n";
  int m, n;
  cout << "Enter number of memory blocks: ";
  cin >> m;
  int blockSize[m], originalBlockSize[m];
  cout << "Enter sizes of the " << m << " blocks:\n";
  for (int i = 0; i < m; i++)
  {
    cin >> blockSize[i];
    originalBlockSize[i] = blockSize[i];
  }
  cout << "Enter number of processes: ";
  cin >> n;
  int processSize[n];
  cout << "Enter sizes of the " << n << " processes:\n";
  for (int i = 0; i < n; i++)
    cin >> processSize[i];
  bestFit(blockSize, originalBlockSize, m, processSize, n);
  return 0;
}
```

Output:



```
Compiled by:- Jonash Chataut
<---- Best fit allocation ---->
Enter number of memory blocks: 5
Enter sizes of the 5 blocks:
100 500 200 300 600
Enter number of processes: 4
Enter sizes of the 4 processes:
212 417 112 426

Process No.     Process Size    Block No.       Block Size      Hole Size
     1              212             4              300              88
     2              417             2              500              83
     3              112             3              200              88
     4              426             5              600             174

--------------------------------
Process exited after 21.16 seconds with return value 0
Press any key to continue . . .
```

Lab No. 5 **Date: 2025/08/14**

Title: Write a program to display the process allocation for user input free blocks and incoming process using Worst Fit Allocation.

Worst Fit is a dynamic memory allocation strategy used in operating systems for assigning free memory blocks to processes. In this method, the largest available memory block is allocated to the process requesting memory. The Worst Fit Allocation Algorithm assigns a process to the largest available memory block that is big enough to hold it. The idea is to leave larger leftover holes, which might fit future processes more effectively than small leftover fragments.

Algorithm Steps

For each process:

1. Search all available free memory blocks.
2. Select the block with the largest size that is greater than or equal to the process size.
3. Allocate the process to that block.
4. Reduce the size of that block (update the hole size).
5. If no block can fit the process, it remains unallocated.

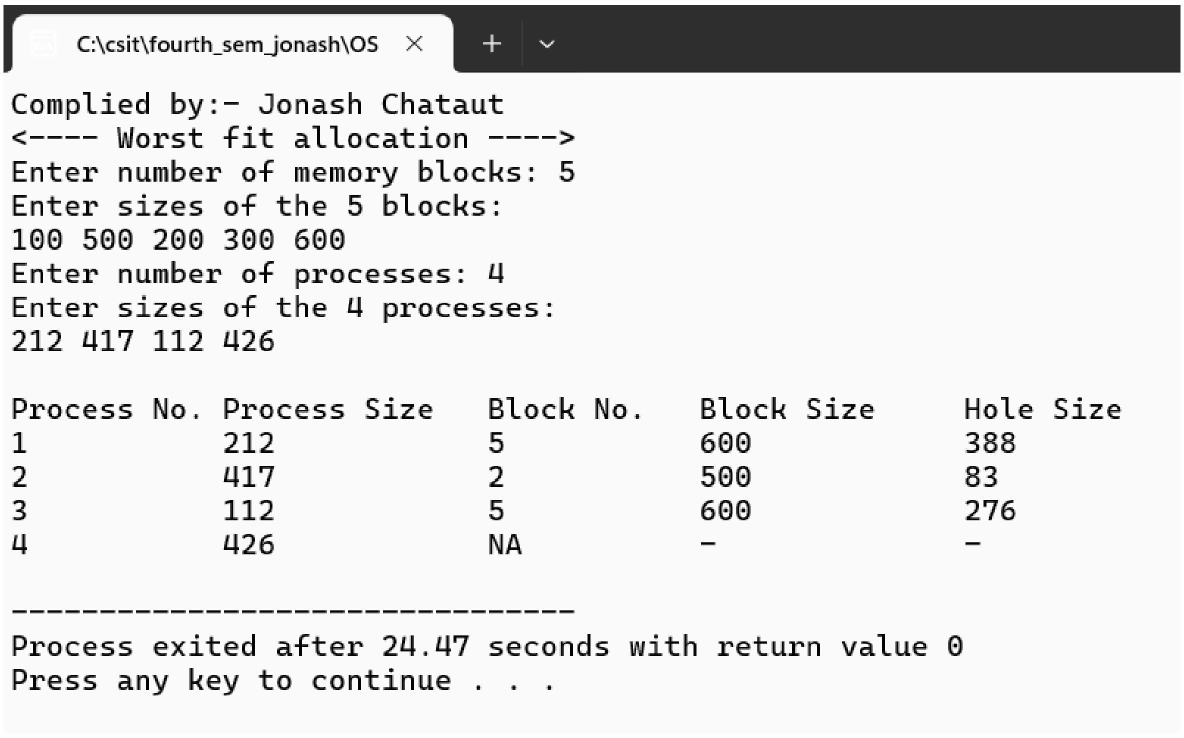
Programming Language: C

IDE: Dev C++

Source Code:

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

void worstFit(int blockSize[], int originalBlockSize[], int m, int processSize[], int n)
{
  int allocation[n];     // Stores block index for each process (-1 if not allocated)
  int holeAfterAlloc[n];  // Stores hole size right after allocation
  // Initialize arrays
  for (int i = 0; i < n; i++)
  {
    allocation[i] = -1;
    holeAfterAlloc[i] = -1;
  }
  // Allocate processes
  for (int i = 0; i < n; i++)
  {
    int worstIdx = -1;
    for (int j = 0; j < m; j++)
    {
      if (blockSize[j] >= processSize[i])
      {
        if (worstIdx == -1 || blockSize[j] > blockSize[worstIdx])
          worstIdx = j;
      }
    }
    if (worstIdx != -1)
    {
      allocation[i] = worstIdx;
      blockSize[worstIdx] -= processSize[i];
      holeAfterAlloc[i] = blockSize[worstIdx]; // store hole size immediately
    }
  }
  // Output results
  cout << "\n"
     << left << setw(12) << "Process No." << setw(15) << "Process Size" << setw(12) << "Block No." <<
   setw(15) << "Block Size" << "Hole Size\n";
  for (int i = 0; i < n; i++)
  {
    cout << left << setw(12) << i + 1
       << setw(15) << processSize[i];
    if (allocation[i] != -1)
    {
      int b = allocation[i];
      cout << setw(12) << b + 1 << setw(15) << originalBlockSize[b] << holeAfterAlloc[i] << "\n";
    }
    else
    {
      cout << setw(12) << "NA" << setw(15) << "-" << "-\n";
    }
    }
}

int main()
{
    cout << "Complied by:- Jonash Chataut\n";
    cout << "<---- Worst fit allocation ---->\n";
    int m, n;
    cout << "Enter number of memory blocks: ";
    cin >> m;
    int blockSize[m], originalBlockSize[m];
    cout << "Enter sizes of the " << m << " blocks:\n";
    for (int i = 0; i < m; i++)
    {
        cin >> blockSize[i];
        originalBlockSize[i] = blockSize[i];
    }
    cout << "Enter number of processes: ";
    cin >> n;
    int processSize[n];
    cout << "Enter sizes of the " << n << " processes:\n";
    for (int i = 0; i < n; i++)
        cin >> processSize[i];
    worstFit(blockSize, originalBlockSize, m, processSize, n);
    return 0;
}
```

Output:



```
Complied by:- Jonash Chataut
<---- Worst fit allocation ---->
Enter number of memory blocks: 5
Enter sizes of the 5 blocks:
100 500 200 300 600
Enter number of processes: 4
Enter sizes of the 4 processes:
212 417 112 426

Process No. Process Size   Block No.   Block Size     Hole Size
1           212            5           600            388
2           417            2           500            83
3           112            5           600            276
4           426            NA          -              -

--------------------------------
Process exited after 24.47 seconds with return value 0
Press any key to continue . . .
```

Madan Bhandari Memorial College

Lab No. 6 **Date: 2025/08/14**

Title: Write a program to display the process allocation for user input free blocks and incoming process using First Fit Allocation.

First fit allocation is a dynamic memory allocation technique in operating systems where a process is allocated to the first available memory block that is large enough to accommodate it. It is simple and quick but may lead to fragmentation.

Algorithm:

a. The operating system maintains a list of free memory blocks (holes).
b. When a process arrives, the OS searches the list from the beginning.
c. The process is allocated to the first block that is large enough.
d. If the block is larger than the process, the remaining portion of the block becomes a hole.
e. If no block is big enough, the process cannot be allocated (it must wait or be rejected).

Programming Language: C++

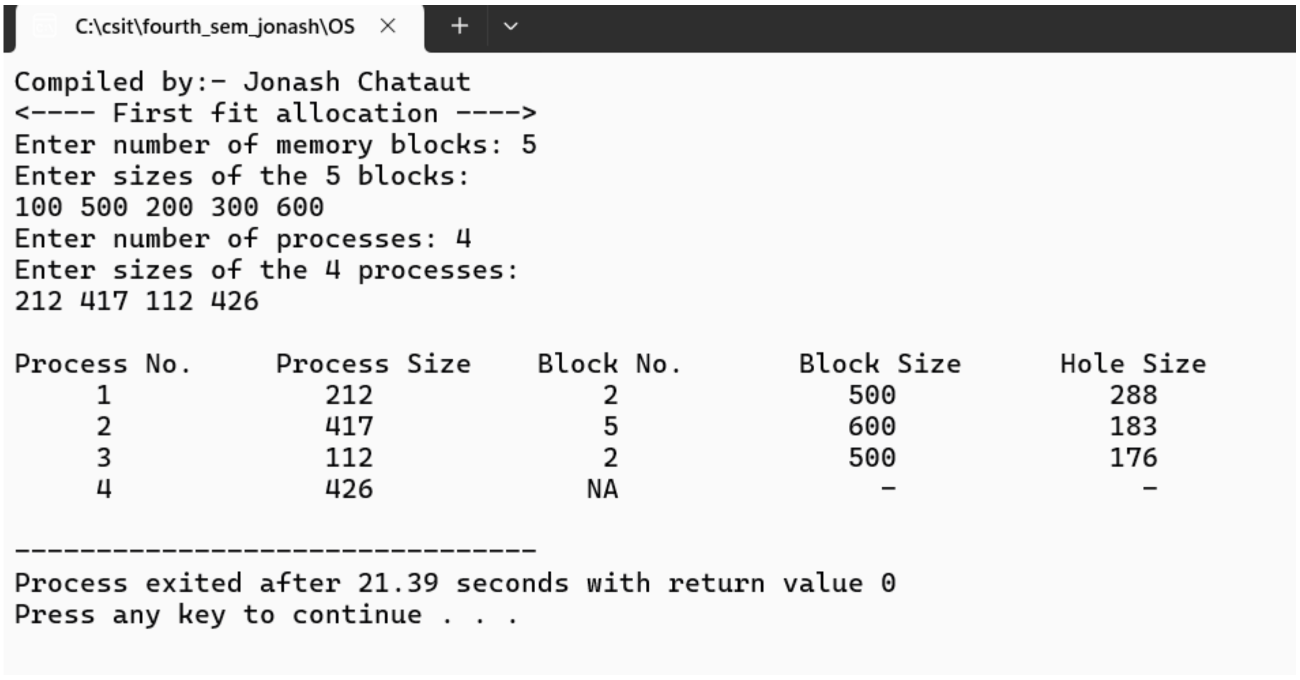
IDE: Dev C++

Source Code:

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

void firstFit(int blockSize[], int originalBlockSize[], int m, int processSize[], int n)
{
  int allocation[n];    // Stores block index for each process (-1 if not allocated)
  int holeAfterAlloc[n]; // Stores hole size immediately after allocation

  // Initialize arrays
  for (int i = 0; i < n; i++)
  {
    allocation[i] = -1;
    holeAfterAlloc[i] = -1;
  }

  // Allocate processes
  for (int i = 0; i < n; i++)
  {
    for (int j = 0; j < m; j++) // First Fit: allocate in first suitable block
    {
      if (blockSize[j] >= processSize[i])
      {
        allocation[i] = j;
        blockSize[j] -= processSize[i];
        holeAfterAlloc[i] = blockSize[j];
        break; // Move to next process
      }
    }
  }

  // Showing output results
  cout << "\nProcess No.\tProcess Size\tBlock No.\tBlock Size\tHole Size\n";
  for (int i = 0; i < n; i++)
  {
    cout << setw(6) << i + 1 << "\t\t"
       << setw(6) << processSize[i] << "\t\t";
    if (allocation[i] != -1)
    {
      int b = allocation[i];
      cout << setw(5) << b + 1 << "\t\t"
         << setw(6) << originalBlockSize[b] << "\t\t"
         << setw(6) << holeAfterAlloc[i] << "\n";
    }
    else
    {
      cout << setw(5) << "NA" << "\t\t" << setw(6) << "-" << "\t\t"<< setw(6) << "-" << "\n";
    }
  }
}
```

```
int main()
{
   cout << "Compiled by:- Jonash Chataut\n";
   cout << "<---- First fit allocation ---->\n";
   int m, n;
   cout << "Enter number of memory blocks: ";
   cin >> m;

   int blockSize[m], originalBlockSize[m];
   cout << "Enter sizes of the " << m << " blocks:\n";
   for (int i = 0; i < m; i++)
   {
      cin >> blockSize[i];
      originalBlockSize[i] = blockSize[i];
   }

   cout << "Enter number of processes: ";
   cin >> n;
   int processSize[n];
   cout << "Enter sizes of the " << n << " processes:\n";
   for (int i = 0; i < n; i++)
      cin >> processSize[i];
   firstFit(blockSize, originalBlockSize, m, processSize, n);
   return 0;
}
```

Output:



```
Compiled by:- Jonash Chataut
<---- First fit allocation ---->
Enter number of memory blocks: 5
Enter sizes of the 5 blocks:
100 500 200 300 600
Enter number of processes: 4
Enter sizes of the 4 processes:
212 417 112 426

Process No.     Process Size    Block No.       Block Size      Hole Size
     1             212             2               500             288
     2             417             5               600             183
     3             112             2               500             176
     4             426            NA                 -               -

--------------------------------
Process exited after 21.39 seconds with return value 0
Press any key to continue . . .
```

Lab No. 4 **Date: 2025/09/01**

Title: Write a program to calculate the average turnaround time and waiting time for user input process parameters using FCFS process scheduling algorithm.

Introduction

First Come First Serve (FCFS) is the simplest CPU scheduling algorithm.

- The process that arrives first is executed first.
- It is a non-preemptive algorithm (once a process starts, it cannot be interrupted).
- It follows the FIFO (First In First Out) principle.

Terminologies

- Burst Time (BT): Time required by a process for execution.
- Arrival Time (AT): Time when a process arrives in the ready queue. (Here we assume AT = 0 for all processes).
- Completion Time (CT): Time when a process finishes execution.
- Turnaround Time (TAT): TAT=CT−AT
- Waiting Time (WT): WT=TAT−BT

Advantages

- Simple and easy to implement.
- Fair for processes (executed in arrival order).

Disadvantages

- Convoy effect: Long processes delay shorter ones.
- Poor performance for varying burst times.

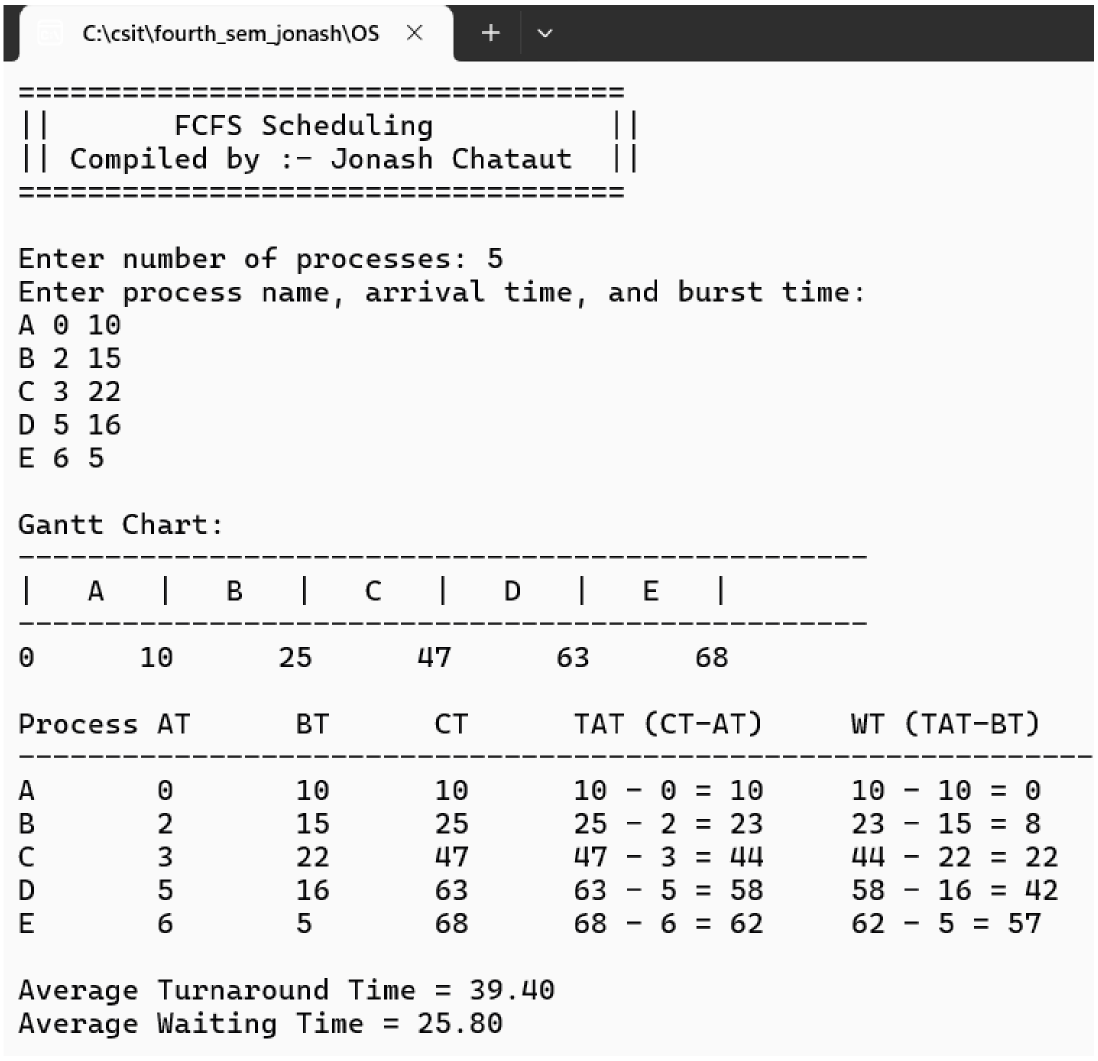
Programming Language: C++

IDE: Dev C++

Source Code:

```
#include <iostream>
#include <iomanip>
using namespace std;
struct Process
{
   string name;
   int arrival;
   int burst;
   int completion;
   int turnaround;
   int waiting;
};

void sortByArrival(Process p[], int n)
{
   for (int i = 0; i < n - 1; i++)
   {
      for (int j = 0; j < n - i - 1; j++)
      {
         if (p[j].arrival > p[j + 1].arrival)
         {
            swap(p[j], p[j + 1]);
         }
      }
   }
}

int main()
{
   cout << "===================================" << endl;
   cout << "||      FCFS Scheduling         ||" << endl;
   cout << "|| Compiled by :- Jonash Chataut  ||" << endl;
   cout << "===================================" << endl << endl;
   int n;
   cout << "Enter number of processes: ";
   cin >> n;
   Process p[n];
   cout << "Enter process name, arrival time, and burst time:\n";
   for (int i = 0; i < n; i++)
   {
      cin >> p[i].name >> p[i].arrival >> p[i].burst;
   }
   sortByArrival(p, n);
   int currentTime = 0;
   double totalTAT = 0, totalWT = 0;
   // Calculate CT, TAT, WT
   for (int i = 0; i < n; i++)
   {
        if (p[i].arrival > currentTime)
        {
            currentTime = p[i].arrival; // CPU idle
        }
        p[i].completion = currentTime + p[i].burst;
        currentTime = p[i].completion;
        p[i].turnaround = p[i].completion - p[i].arrival; // TAT = CT - AT
        p[i].waiting = p[i].turnaround - p[i].burst;     // WT = TAT - BT
        totalTAT += p[i].turnaround;
        totalWT += p[i].waiting;
    }
    // Print Gantt Chart
    cout << "\nGantt Chart:\n";
    cout << "------------------------------------------------\n";
    for (int i = 0; i < n; i++)
    {
        cout << "|   " << p[i].name << "   ";
    }
    cout << "|\n";
    cout << "------------------------------------------------\n";
    cout << p[0].arrival;
    for (int i = 0; i < n; i++)
    {
        cout << setw(8) << p[i].completion;
    }
    cout << "\n";
    // Print Process Table
    cout << "\nProcess\tAT\tBT\tCT\tTAT (CT-AT)\tWT (TAT-BT)\n";
    cout << "------------------------------------------------------------------\n";
    for (int i = 0; i < n; i++)
    {
        cout << p[i].name << "\t"
             << p[i].arrival << "\t"
             << p[i].burst << "\t"
             << p[i].completion << "\t"
             << p[i].completion << " - " << p[i].arrival << " = " << p[i].turnaround << "\t"
             << p[i].turnaround << " - " << p[i].burst << " = " << p[i].waiting << "\n";
    }
    cout << "\nAverage Turnaround Time = " << fixed << setprecision(2) << (totalTAT / n);
    cout << "\nAverage Waiting Time = " << fixed << setprecision(2) << (totalWT / n) << "\n";
    return 0;
}
```

Output:



```
===================================
||        FCFS Scheduling        ||
|| Compiled by :- Jonash Chataut  ||
===================================

Enter number of processes: 5
Enter process name, arrival time, and burst time:
A 0 10
B 2 15
C 3 22
D 5 16
E 6 5

Gantt Chart:
-------------------------------------------------
|   A   |   B   |   C   |   D   |   E   |
-------------------------------------------------
0      10      25      47      63      68

Process AT      BT      CT      TAT (CT-AT)     WT (TAT-BT)
-----------------------------------------------------------------
A       0       10      10      10 - 0 = 10     10 - 10 = 0
B       2       15      25      25 - 2 = 23     23 - 15 = 8
C       3       22      47      47 - 3 = 44     44 - 22 = 22
D       5       16      63      63 - 5 = 58     58 - 16 = 42
E       6       5       68      68 - 6 = 62     62 - 5 = 57

Average Turnaround Time = 39.40
Average Waiting Time = 25.80
```

Lab No. 5 **Date: 2025/09/01**

Title: Write a program to calculate the average turnaround time and waiting time for user input process parameters using SJF process scheduling algorithm.

Introduction

SJF is a non-preemptive scheduling algorithm where the CPU is assigned to the process with the shortest burst time among the available processes in the ready queue. Once a process starts execution, it cannot be interrupted.

Characteristics:

- Non-preemptive: A running process cannot be stopped until it finishes.
- Optimal in terms of minimizing average waiting time if all burst times are known.
- May cause starvation for long processes if short processes keep arriving.

Algorithm:

1. Sort processes by arrival time.
2. At any time, select the available process with the shortest burst time.
3. Execute the selected process until completion.
4. Repeat step 2 until all processes are completed.

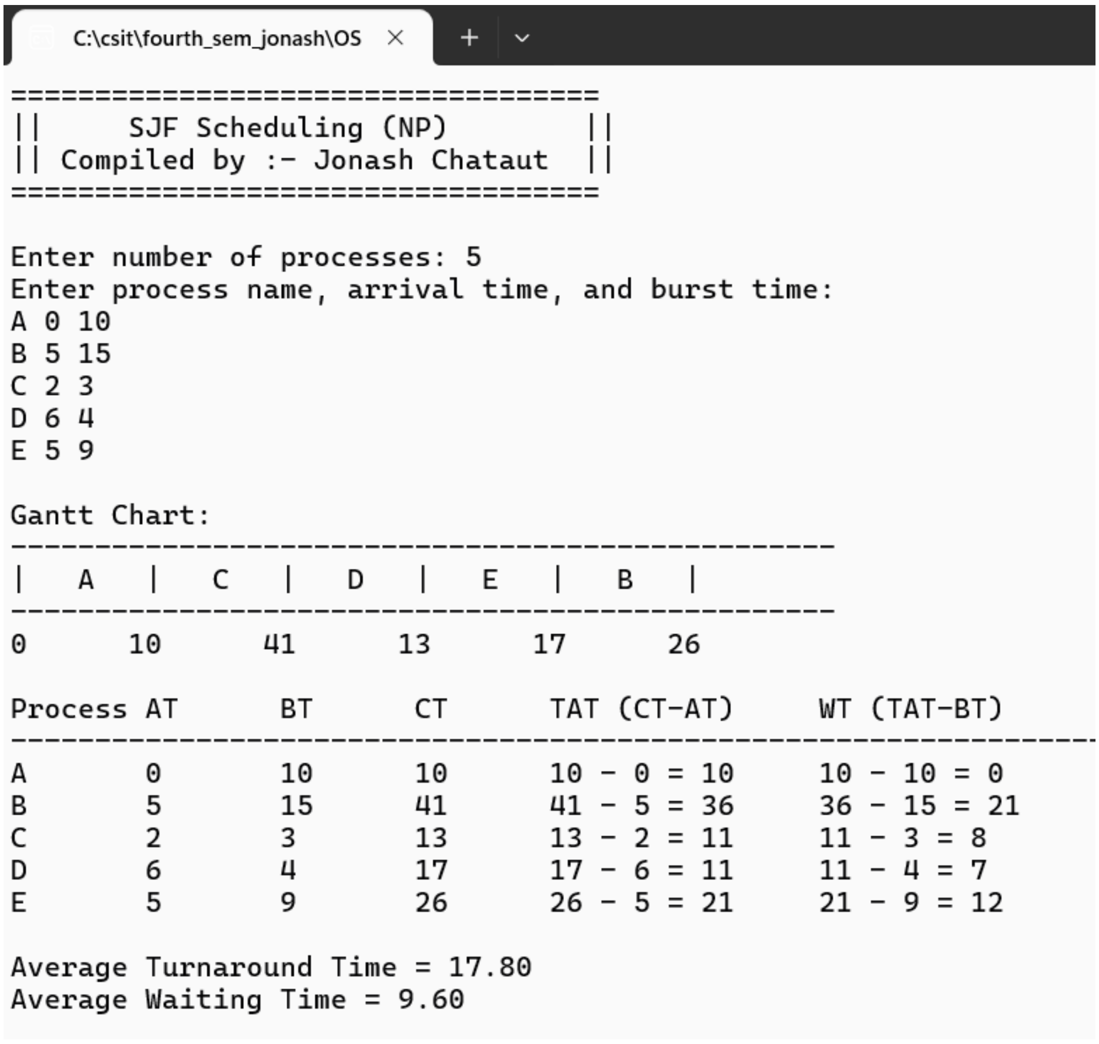
Programming Language: C++

IDE: Dev C++

Source Code:

```
#include <iostream>
#include <iomanip>
using namespace std;
struct Process
{
   string name;
   int arrival;
   int burst;
   int completion;
   int turnaround;
   int waiting;
   bool done;
};
int main()
{
   cout << "===================================" << endl;
   cout << "||    SJF Scheduling (NP)      ||" << endl;
   cout << "|| Compiled by :- Jonash Chataut  ||" << endl;
   cout << "===================================" << endl << endl;
   int n;
   cout << "Enter number of processes: ";
   cin >> n;
   Process p[n];
   cout << "Enter process name, arrival time, and burst time:\n";
   for (int i = 0; i < n; i++)
   {
      cin >> p[i].name >> p[i].arrival >> p[i].burst;
      p[i].done = false;
   }
   int currentTime = 0, completed = 0;
   double totalTAT = 0, totalWT = 0;
   // Print Gantt Chart
   cout << "\nGantt Chart:\n";
   cout << "----------------------------------------------\n";
   while (completed < n)
   {
      int idx = -1;
      int minBT = 1e9;
      // Find the shortest job among arrived processes
      for (int i = 0; i < n; i++)
      {
         if (!p[i].done && p[i].arrival <= currentTime)
         {
            if (p[i].burst < minBT)
            {
               minBT = p[i].burst;
               idx = i;
            }
            }
        }
        if (idx == -1)
        {
            currentTime++; // CPU idle
            continue;
        }
        cout << "|   " << p[idx].name << "   ";
        // Calculate times
        p[idx].completion = currentTime + p[idx].burst;
        currentTime = p[idx].completion;
        p[idx].turnaround = p[idx].completion - p[idx].arrival; // TAT = CT - AT
        p[idx].waiting = p[idx].turnaround - p[idx].burst;     // WT = TAT - BT
        totalTAT += p[idx].turnaround;
        totalWT += p[idx].waiting;
        p[idx].done = true;
        completed++;
    }
    cout << "|\n";
    cout << "-------------------------------------------------\n";
    // Print timeline
    currentTime = 0;
    cout << 0;
    for (int i = 0; i < n; i++)
    {
        if (p[i].done)
        {
            cout << setw(8) << p[i].completion;
        }
    }
    cout << "\n";
    // Print Process Table
    cout << "\nProcess\tAT\tBT\tCT\tTAT (CT-AT)\tWT (TAT-BT)\n";
    cout << "------------------------------------------------------------------\n";
    for (int i = 0; i < n; i++)
    {
        cout << p[i].name << "\t"
             << p[i].arrival << "\t"
             << p[i].burst << "\t"
             << p[i].completion << "\t"
             << p[i].completion << " - " << p[i].arrival << " = " << p[i].turnaround << "\t"
             << p[i].turnaround << " - " << p[i].burst << " = " << p[i].waiting << "\n";
    }
    cout << "\nAverage Turnaround Time = " << fixed << setprecision(2) << (totalTAT / n);
    cout << "\nAverage Waiting Time = " << fixed << setprecision(2) << (totalWT / n) << "\n";
    return 0;
}
```

Output:


C:\csit\fourth_sem_jonash\OS

```
===================================
||      SJF Scheduling (NP)       ||
|| Compiled by :- Jonash Chataut  ||
===================================

Enter number of processes: 5
Enter process name, arrival time, and burst time:
A 0 10
B 5 15
C 2 3
D 6 4
E 5 9

Gantt Chart:
-------------------------------------------------
|   A   |   C   |   D   |   E   |   B   |
-------------------------------------------------
0      10      41      13      17      26

Process AT      BT      CT      TAT (CT-AT)     WT (TAT-BT)
-------------------------------------------------------------------
A       0       10      10      10 - 0 = 10     10 - 10 = 0
B       5       15      41      41 - 5 = 36     36 - 15 = 21
C       2       3       13      13 - 2 = 11     11 - 3 = 8
D       6       4       17      17 - 6 = 11     11 - 4 = 7
E       5       9       26      26 - 5 = 21     21 - 9 = 12

Average Turnaround Time = 17.80
Average Waiting Time = 9.60
```

Lab No. 6 **Date: 2025/09/01**

Title: Write a program to calculate the average turnaround time and waiting time for user input process parameters using RR process scheduling algorithm

Introduction

Round Robin (RR) is a preemptive CPU scheduling algorithm where each process is assigned a fixed time slice or time quantum. The CPU executes each process in the ready queue for the duration of the quantum, then moves to the next process in a cyclic manner.

Characteristics:

- Preemptive: A process may be interrupted if it exceeds the time quantum.
- Fair: Every process gets an equal chance to execute.
- Suitable for time-sharing systems.

Algorithm Steps:

1. Initialize currentTime = 0.
2. Keep a ready queue of processes that have arrived.
3. Pick the first process in the queue and execute it for:
    - Time Quantum if remaining burst > quantum, or
    - Remaining burst time if ≤ quantum.
4. Update remaining burst time and currentTime.
5. If the process is not finished, move it to the end of the ready queue.
6. Repeat steps 3–5 until all processes are completed.

Programming Language: C++

IDE: Dev C++

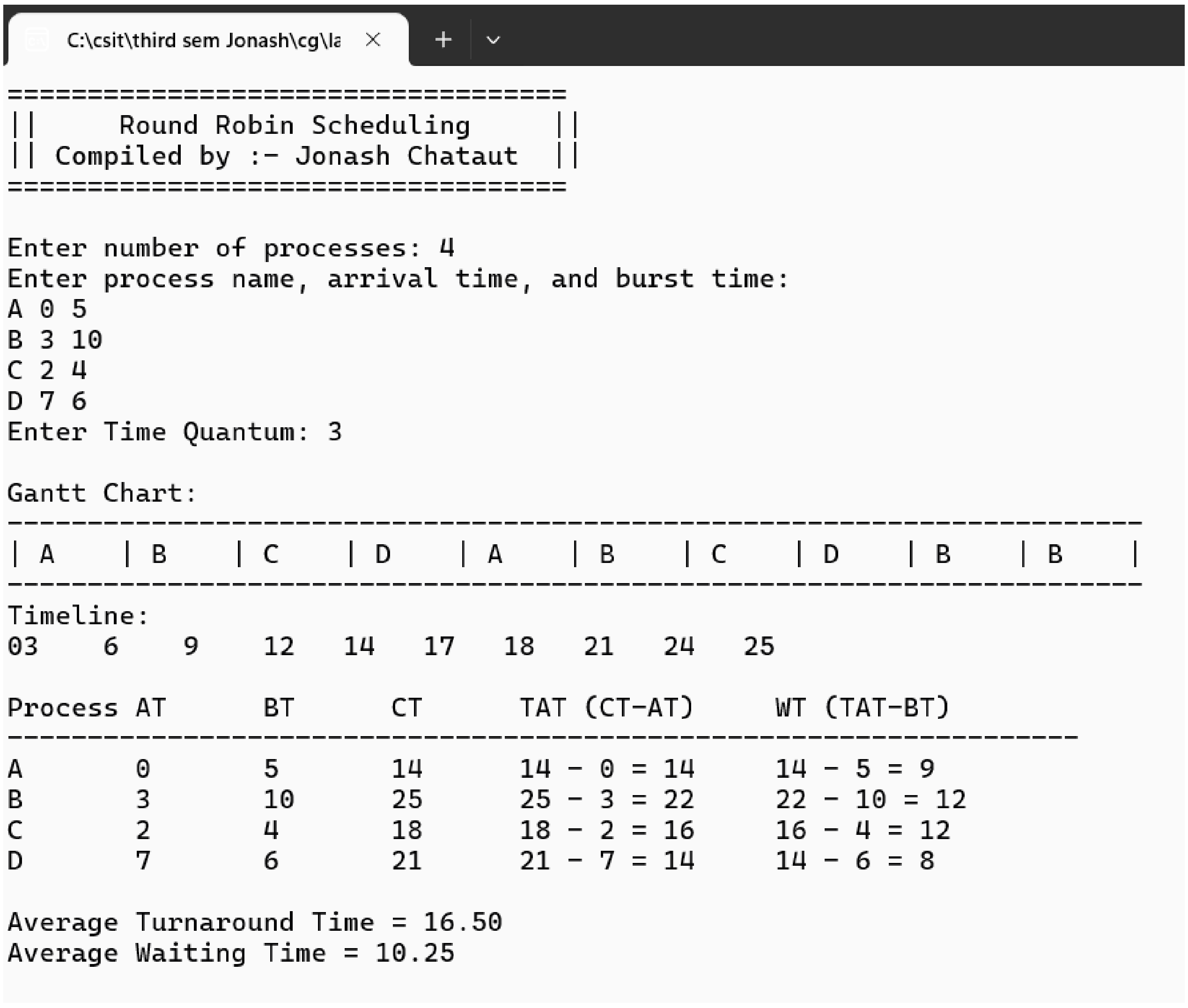
Source Code:

```
#include <iostream>
#include <iomanip>
using namespace std;
struct Process
{
  string name;
  int arrival;
  int burst;
  int remaining; // Remaining burst time
  int completion;
  int turnaround;
  int waiting;
};
int main()
{
  cout << "=====================================" << endl;
  cout << "||    Round Robin Scheduling    ||" << endl;
  cout << "|| Compiled by :- Jonash Chataut  ||" << endl;
  cout << "=====================================" << endl << endl;
  int n, quantum;
  cout << "Enter number of processes: ";
  cin >> n;
  Process p[n];
  cout << "Enter process name, arrival time, and burst time:\n";
  for (int i = 0; i < n; i++)
  {
    cin >> p[i].name >> p[i].arrival >> p[i].burst;
    p[i].remaining = p[i].burst;
    p[i].completion = 0;
  }
  cout << "Enter Time Quantum: ";
  cin >> quantum;
  int currentTime = 0, completed = 0;
  double totalTAT = 0, totalWT = 0;
  cout << "\nGantt Chart:\n";
  cout << "------------------------------------------------\n";
  // RR Scheduling
  while (completed < n)
  {
    bool idle = true;
    for (int i = 0; i < n; i++)
    {
      if (p[i].remaining > 0 && p[i].arrival <= currentTime)
      {
        idle = false;
        cout << "| " << setw(4) << left << p[i].name << " ";

        if (p[i].remaining <= quantum)
```

```
            {
               currentTime += p[i].remaining;
               p[i].completion = currentTime;
               p[i].remaining = 0;
               completed++;
            }
            else
            {
               currentTime += quantum;
               p[i].remaining -= quantum;
            }
         }
      }
      if (idle)
      {
         cout << "| IDLE ";
         currentTime++;
      }
   }
   cout << "|\n";
   cout << "------------------------------------------------\n";
   // Print Timeline
   cout << "Timeline:\n0";
   currentTime = 0;
   int temp[n];
   for (int i = 0; i < n; i++)
      temp[i] = p[i].burst;
   completed = 0;
   while (completed < n)
   {
      bool idle = true;
      for (int i = 0; i < n; i++)
      {
         if (temp[i] > 0 && p[i].arrival <= currentTime)
         {
            idle = false;
            if (temp[i] <= quantum)
            {
               currentTime += temp[i];
               temp[i] = 0;
               completed++;
            }
            else
            {
               currentTime += quantum;
               temp[i] -= quantum;
            }
            cout << setw(5) << currentTime;
         }
      }
      if (idle)
```

```
        currentTime++;
    }
    // Print Process Table with calculations
    cout << "\n\nProcess\tAT\tBT\tCT\tTAT (CT-AT)\tWT (TAT-BT)\n";
    cout << "----------------------------------------------------------------\n";
    for (int i = 0; i < n; i++)
    {
      p[i].turnaround = p[i].completion - p[i].arrival;
      p[i].waiting = p[i].turnaround - p[i].burst;
      totalTAT += p[i].turnaround;
      totalWT += p[i].waiting;
      cout << p[i].name << "\t"
         << p[i].arrival << "\t"
         << p[i].burst << "\t"
         << p[i].completion << "\t"
         << p[i].completion << " - " << p[i].arrival << " = " << p[i].turnaround << "\t"
         << p[i].turnaround << " - " << p[i].burst << " = " << p[i].waiting << "\n";
    }
    cout << "\nAverage Turnaround Time = " << fixed << setprecision(2) << (totalTAT / n);
    cout << "\nAverage Waiting Time = " << fixed << setprecision(2) << (totalWT / n) << "\n";
    return 0;
  }
```

Output:



```
C:\csit\third sem Jonash\cg\la
===================================
||     Round Robin Scheduling     ||
|| Compiled by :- Jonash Chataut  ||
===================================

Enter number of processes: 4
Enter process name, arrival time, and burst time:
A 0 5
B 3 10
C 2 4
D 7 6
Enter Time Quantum: 3

Gantt Chart:
-----------------------------------------------------------------------
| A     | B     | C     | D     | A     | B     | C     | D     | B     | B     |
-----------------------------------------------------------------------
Timeline:
03    6    9    12   14   17   18   21   24   25

Process AT      BT      CT      TAT (CT-AT)     WT (TAT-BT)
-----------------------------------------------------------------
A       0       5       14      14 - 0 = 14     14 - 5 = 9
B       3       10      25      25 - 3 = 22     22 - 10 = 12
C       2       4       18      18 - 2 = 16     16 - 4 = 12
D       7       6       21      21 - 7 = 14     14 - 6 = 8

Average Turnaround Time = 16.50
Average Waiting Time = 10.25
```

Madan Bhandari Memorial College

Lab No. 10 **Date: 2025/09/08**

Title: Write a program to calculate the seek time for user input pending request, total no of cylinders and current position of I/O read/write head using FCFS disk scheduling algorithm.

Introduction:
FCFS is the simplest disk scheduling algorithm where requests are serviced in the order they arrive. It is non-preemptive, easy to implement and ensures that all requests are handled fairly without starvation. However, the performance may vary depending on the order of requests, and the average seek time can be high.

Algorithm Steps:

1. Maintain a queue of disk requests.
2. Serve each request in the order it arrives.
3. Move the disk head to the requested track and continue to the next request in the queue.

Advantages:

- Simple and fair.

Disadvantages:

- High average seek time in some cases.

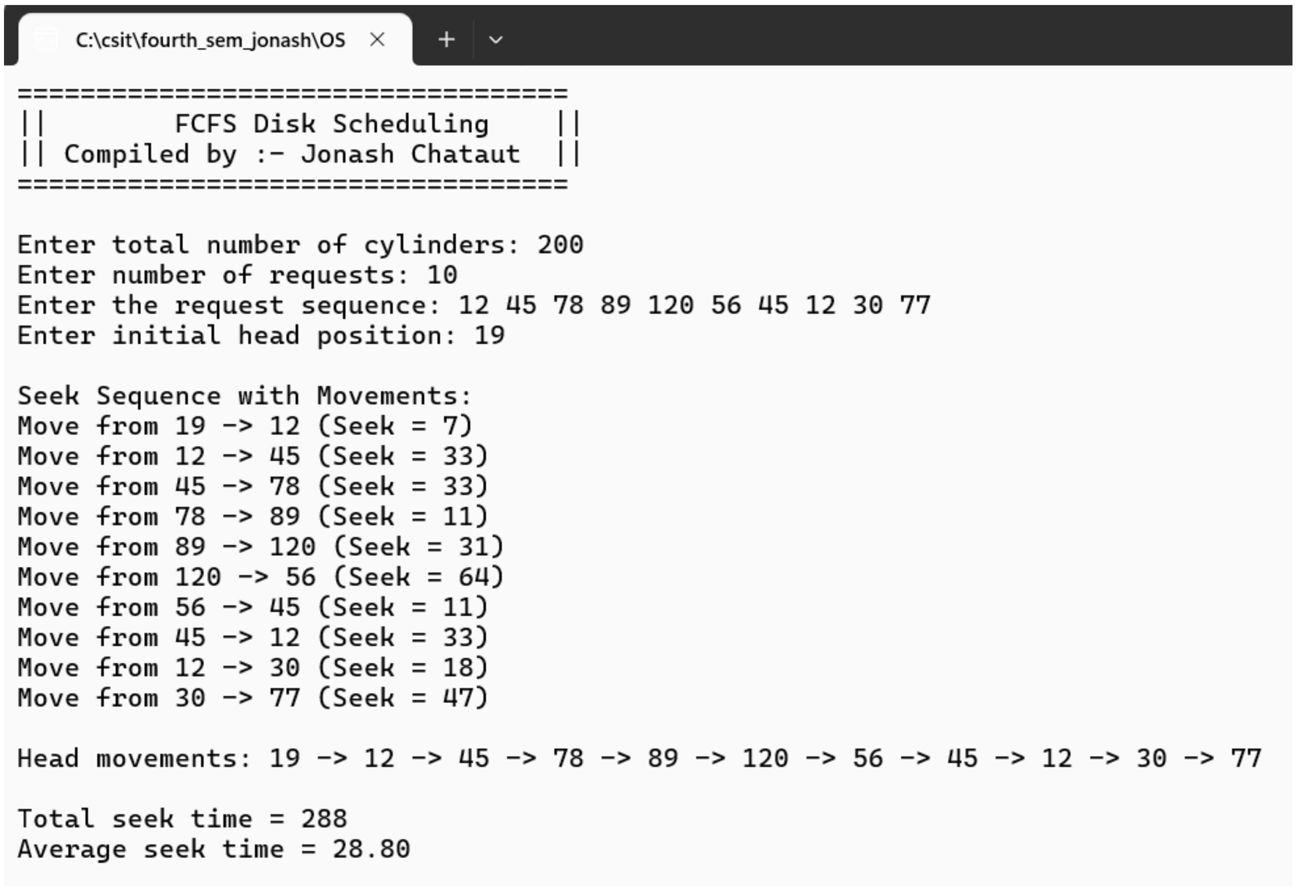
Programming Language: C++

IDE: Dev C++

Source Code:

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
int main()
{
   int n, head, cylinders, seek = 0;
   cout << "==================================" << endl;
   cout << "||       FCFS Disk Scheduling    ||" << endl;
   cout << "|| Compiled by :- Jonash Chataut  ||" << endl;
   cout << "==================================" << endl << endl;
   cout << "Enter total number of cylinders: ";
   cin >> cylinders;
   cout << "Enter number of requests: ";
   cin >> n;
   int arr[n];
   cout << "Enter the request sequence: ";
   for (int i = 0; i < n; i++)
      cin >> arr[i];
   cout << "Enter initial head position: ";
   cin >> head;
   int current = head;
   cout << "\nSeek Sequence with Movements:" << endl;
   for (int i = 0; i < n; i++)
   {
      int move = abs(current - arr[i]);
      cout << "Move from " << current << " -> " << arr[i] << " (Seek = " << move << ")" << endl;
      seek += move;
      current = arr[i];
   }
   // Print head movements
   current = head;
   cout << "\nHead movements: " << current;
   for (int i = 0; i < n; i++)
      cout << " -> " << arr[i];
   cout << "\n\nTotal seek time = " << seek << endl;
   cout << "Average seek time = " << fixed << setprecision(2) << (float)seek / n << endl;
   return 0;
}
```

Output:



```
===================================
||        FCFS Disk Scheduling    ||
|| Compiled by :- Jonash Chataut  ||
===================================

Enter total number of cylinders: 200
Enter number of requests: 10
Enter the request sequence: 12 45 78 89 120 56 45 12 30 77
Enter initial head position: 19

Seek Sequence with Movements:
Move from 19 -> 12 (Seek = 7)
Move from 12 -> 45 (Seek = 33)
Move from 45 -> 78 (Seek = 33)
Move from 78 -> 89 (Seek = 11)
Move from 89 -> 120 (Seek = 31)
Move from 120 -> 56 (Seek = 64)
Move from 56 -> 45 (Seek = 11)
Move from 45 -> 12 (Seek = 33)
Move from 12 -> 30 (Seek = 18)
Move from 30 -> 77 (Seek = 47)

Head movements: 19 -> 12 -> 45 -> 78 -> 89 -> 120 -> 56 -> 45 -> 12 -> 30 -> 77

Total seek time = 288
Average seek time = 28.80
```

Madan Bhandari Memorial College

Lab No. 11 **Date: 2025/09/08**

Title: Write a program to calculate the seek time for user input pending request, total no of cylinders and current position of I/O read/write head using SCAN disk scheduling algorithm.

Introduction:
SCAN also called the elevator algorithm, moves the disk head in one direction servicing all requests until it reaches the end of the disk, then reverses direction. This approach reduces large variations in response time and ensures that all requests are eventually serviced.

Algorithm Steps:

1. Decide the initial direction of head movement (toward higher or lower track numbers).
2. Move the head in that direction, servicing all pending requests.
3. At the end, reverse the direction and continue servicing requests in the opposite direction.

Advantages:

- More efficient than FCFS.
- Reduces the chance of starvation.

Disadvantages:

- Requests at the far end of the disk may have to wait longer.

Programming Language: C++

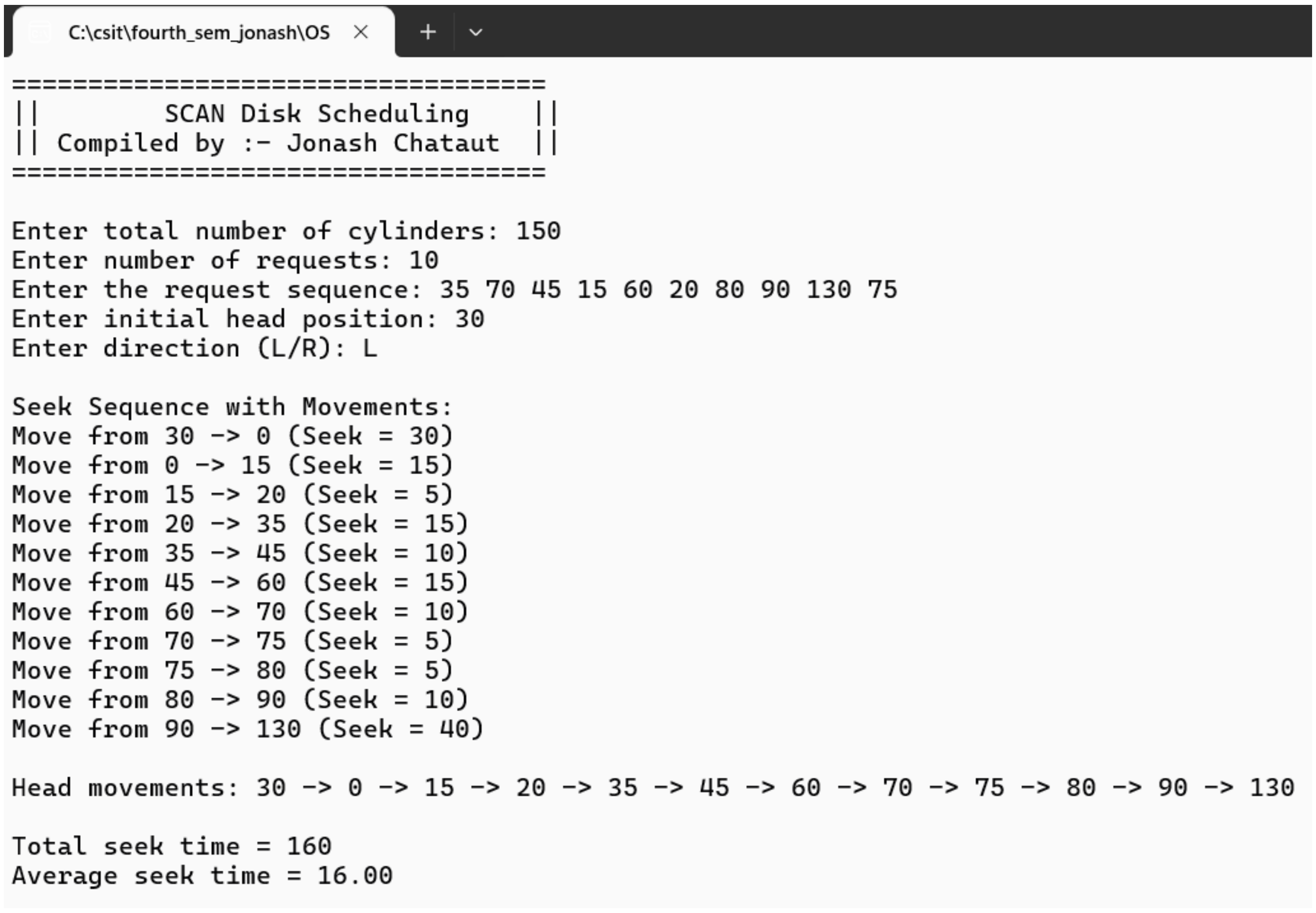
IDE: Dev C++

Source Code:

```cpp
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
int main()
{
  int n, head, cylinders, seek = 0;
  char direction;
  cout << "===================================" << endl;
  cout << "||       SCAN Disk Scheduling    ||" << endl;
  cout << "|| Compiled by :- Jonash Chataut  ||" << endl;
  cout << "===================================" << endl << endl;
  cout << "Enter total number of cylinders: ";
  cin >> cylinders;
  cout << "Enter number of requests: ";
  cin >> n;
  int arr[n], left[20], right[20];
  int lcount = 0, rcount = 0;
  cout << "Enter the request sequence: ";
  for (int i = 0; i < n; i++)
    cin >> arr[i];
  cout << "Enter initial head position: ";
  cin >> head;
  cout << "Enter direction (L/R): ";
  cin >> direction;
  // Split requests
  for (int i = 0; i < n; i++)
  {
    if (arr[i] < head)
      left[lcount++] = arr[i];
    else
      right[rcount++] = arr[i];
  }
  if (direction == 'L')
    left[lcount++] = 0;
  else
    right[rcount++] = cylinders - 1;
  // Sort arrays
  for (int i = 0; i < rcount - 1; i++)
    for (int j = i + 1; j < rcount; j++)
      if (right[i] > right[j])
        swap(right[i], right[j]);
  for (int i = 0; i < lcount - 1; i++)
    for (int j = i + 1; j < lcount; j++)
      if (left[i] < left[j])
        swap(left[i], left[j]);
  // Step-by-step movements
  cout << "\nSeek Sequence with Movements:" << endl;
  int current = head;
```

```
    int movement[n + 2], mcount = 0;
    if (direction == 'R')
    {
      for (int i = 0; i < rcount; i++)
      {
        int move = abs(current - right[i]);
        cout << "Move from " << current << " -> " << right[i] << " (Seek = " << move << ")" << endl;
        seek += move;
        current = right[i];
        movement[mcount++] = current;
      }
      for (int i = 0; i < lcount; i++)
      {
        int move = abs(current - left[i]);
        cout << "Move from " << current << " -> " << left[i] << " (Seek = " << move << ")" << endl;
        seek += move;
        current = left[i];
        movement[mcount++] = current;
      }
    }
    else
    {
      for (int i = lcount - 1; i >= 0; i--)
      {
        int move = abs(current - left[i]);
        cout << "Move from " << current << " -> " << left[i] << " (Seek = " << move << ")" << endl;
        seek += move;
        current = left[i];
        movement[mcount++] = current;
      }
      for (int i = 0; i < rcount; i++)
      {
        int move = abs(current - right[i]);
        cout << "Move from " << current << " -> " << right[i] << " (Seek = " << move << ")" << endl;
        seek += move;
        current = right[i];
        movement[mcount++] = current;
      }
    }
    // Head movements line
    cout << "\nHead movements: " << head;
    for (int i = 0; i < mcount; i++)
      cout << " -> " << movement[i];
    cout << "\n\nTotal seek time = " << seek << endl;
    cout << "Average seek time = " << fixed << setprecision(2) << (float)seek / n << endl;
    return 0;
}
```

Output:



```
===================================
||         SCAN Disk Scheduling    ||
|| Compiled by :- Jonash Chataut  ||
===================================

Enter total number of cylinders: 150
Enter number of requests: 10
Enter the request sequence: 35 70 45 15 60 20 80 90 130 75
Enter initial head position: 30
Enter direction (L/R): L

Seek Sequence with Movements:
Move from 30 -> 0 (Seek = 30)
Move from 0 -> 15 (Seek = 15)
Move from 15 -> 20 (Seek = 5)
Move from 20 -> 35 (Seek = 15)
Move from 35 -> 45 (Seek = 10)
Move from 45 -> 60 (Seek = 15)
Move from 60 -> 70 (Seek = 10)
Move from 70 -> 75 (Seek = 5)
Move from 75 -> 80 (Seek = 5)
Move from 80 -> 90 (Seek = 10)
Move from 90 -> 130 (Seek = 40)

Head movements: 30 -> 0 -> 15 -> 20 -> 35 -> 45 -> 60 -> 70 -> 75 -> 80 -> 90 -> 130

Total seek time = 160
Average seek time = 16.00
```

Madan Bhandari Memorial College

Lab No. 12 **Date: 2025/09/08**

Title: Write a program to calculate the seek time for user input pending request, total no of cylinders and current position of I/O read/write head using LOOK disk scheduling algorithm.

Introduction:
LOOK is a modified version of SCAN. The head only moves as far as the last request in the current direction before reversing, instead of going all the way to the end of the disk. This reduces unnecessary head movement and improves efficiency.

Algorithm Steps:

1. Move the head in the initial direction, servicing requests until the last request in that direction.
2. Reverse direction and continue servicing.

Advantages:

- Optimized SCAN with less head movement.
- More efficient and faster.

Disadvantages:

- Slightly more complex than SCAN.

Programming Language: C++

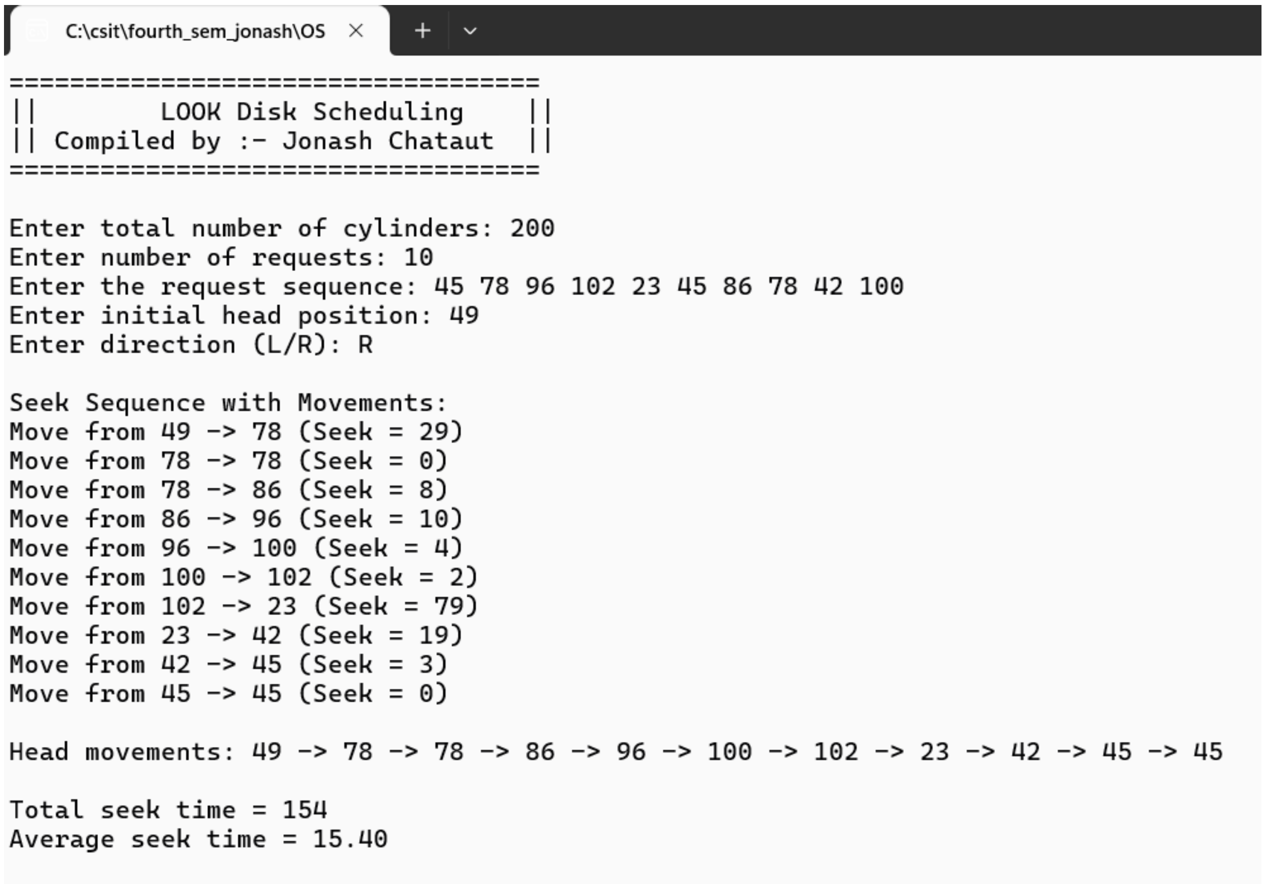
IDE: Dev C++

Source Code:

```
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;
int main()
{
   int n, head, cylinders, seek = 0;
   char direction;
   cout << "===================================" << endl;
   cout << "||       LOOK Disk Scheduling    ||" << endl;
   cout << "|| Compiled by :- Jonash Chataut  ||" << endl;
   cout << "===================================" << endl << endl;
   cout << "Enter total number of cylinders: ";
   cin >> cylinders;
   cout << "Enter number of requests: ";
   cin >> n;
   int arr[n], left[20], right[20];
   int lcount = 0, rcount = 0;
   cout << "Enter the request sequence: ";
   for (int i = 0; i < n; i++)
      cin >> arr[i];
   cout << "Enter initial head position: ";
   cin >> head;
   cout << "Enter direction (L/R): ";
   cin >> direction;
   for (int i = 0; i < n; i++)
   {
      if (arr[i] < head)
         left[lcount++] = arr[i];
      else
         right[rcount++] = arr[i];
   }
   // Sort arrays
   for (int i = 0; i < rcount - 1; i++)
      for (int j = i + 1; j < rcount; j++)
         if (right[i] > right[j])
            swap(right[i], right[j]);
   for (int i = 0; i < lcount - 1; i++)
      for (int j = i + 1; j < lcount; j++)
         if (left[i] < left[j])
            swap(left[i], left[j]);
   cout << "\nSeek Sequence with Movements:" << endl;
   int current = head;
   int movement[n], mcount = 0;
   if (direction == 'R')
   {
      for (int i = 0; i < rcount; i++)
      {
         int move = abs(current - right[i]);
```

```
            cout << "Move from " << current << " -> " << right[i] << " (Seek = " << move << ")" << endl;
            seek += move;
            current = right[i];
            movement[mcount++] = current;
        }
        for (int i = lcount - 1; i >= 0; i--)
        {
            int move = abs(current - left[i]);
            cout << "Move from " << current << " -> " << left[i] << " (Seek = " << move << ")" << endl;
            seek += move;
            current = left[i];
            movement[mcount++] = current;
        }
    }
    else
    {
        for (int i = lcount - 1; i >= 0; i--)
        {
            int move = abs(current - left[i]);
            cout << "Move from " << current << " -> " << left[i] << " (Seek = " << move << ")" << endl;
            seek += move;
            current = left[i];
            movement[mcount++] = current;
        }
        for (int i = 0; i < rcount; i++)
        {
            int move = abs(current - right[i]);
            cout << "Move from " << current << " -> " << right[i] << " (Seek = " << move << ")" << endl;
            seek += move;
            current = right[i];
            movement[mcount++] = current;
        }
    }
    // Head movements line
    cout << "\nHead movements: " << head;
    for (int i = 0; i < mcount; i++)
        cout << " -> " << movement[i];
    cout << "\n\nTotal seek time = " << seek << endl;
    cout << "Average seek time = " << fixed << setprecision(2) << (float)seek / n << endl;

    return 0;
}
```

Output:



```
===================================
||        LOOK Disk Scheduling    ||
|| Compiled by :- Jonash Chataut  ||
===================================

Enter total number of cylinders: 200
Enter number of requests: 10
Enter the request sequence: 45 78 96 102 23 45 86 78 42 100
Enter initial head position: 49
Enter direction (L/R): R

Seek Sequence with Movements:
Move from 49 -> 78 (Seek = 29)
Move from 78 -> 78 (Seek = 0)
Move from 78 -> 86 (Seek = 8)
Move from 86 -> 96 (Seek = 10)
Move from 96 -> 100 (Seek = 4)
Move from 100 -> 102 (Seek = 2)
Move from 102 -> 23 (Seek = 79)
Move from 23 -> 42 (Seek = 19)
Move from 42 -> 45 (Seek = 3)
Move from 45 -> 45 (Seek = 0)

Head movements: 49 -> 78 -> 78 -> 86 -> 96 -> 100 -> 102 -> 23 -> 42 -> 45 -> 45

Total seek time = 154
Average seek time = 15.40
```

Madan Bhandari Memorial College

Lab No. 13 **Date: 2025/09/08**

Title: Write a program to test if the system is free from dead lock or not for the user input allocation, max and available matrix.

Introduction:
In multiprogramming systems, processes compete for limited resources like CPU, memory, and I/O devices. If processes hold some resources and wait indefinitely for others, the system may enter a deadlock. Deadlock prevents processes from making progress and must be detected or prevented.

The Banker's Algorithm is used to avoid deadlock by ensuring the system is always in a safe state. A state is considered safe if the system can allocate resources to each process in some order and still avoid deadlock.

- **Important Terms**
  - Allocation Matrix → Resources currently allocated to each process.
  - Max Matrix → Maximum resources each process may need.
  - Available Vector → Number of resources currently available in the system.
  - Need Matrix → Additional resources a process may still request, calculated as:

$$\text{Need}[i][j] = \text{Max}[i][j] - \text{Allocation}[i][j]$$

- **Algorithm:**

1. Initialization:
   - Work = Available
   - Finish[i] = false for all processes
2. Find a process Pi such that:
   - Finish[i] = false
   - Need[i] ≤ Work
3. If such a process is found:
   - Allocate its resources back: Work = Work + Allocation[i]
   - Mark Finish[i] = true
   - Add process to the safe sequence
4. Repeat until:
   - All processes finish → System is Safe (Deadlock-free)
   - No such process exists → System is Unsafe (Deadlock present)

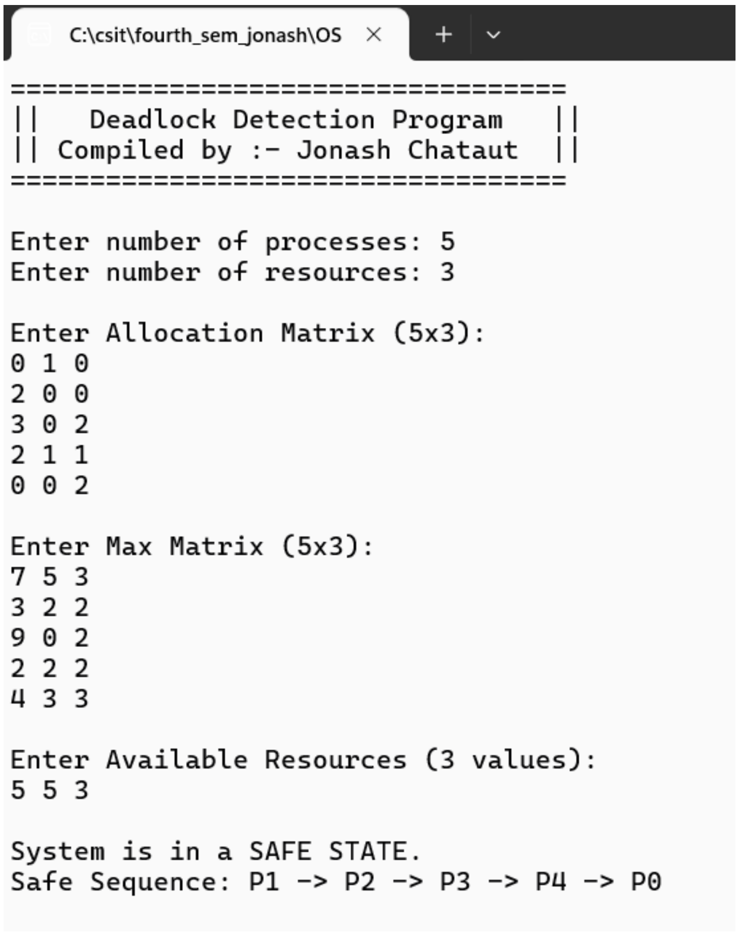
Programming Language: C++

IDE: Dev C++

Source Code:

```
#include <iostream>
using namespace std;
int main()
{
   int n, m; // n = number of processes, m = number of resources
   cout << "==================================" << endl;
   cout << "||   Deadlock Detection Program   ||" << endl;
   cout << "|| Compiled by :- Jonash Chataut  ||" << endl;
   cout << "==================================" << endl << endl;
   cout << "Enter number of processes: ";
   cin >> n;
   cout << "Enter number of resources: ";
   cin >> m;
   int allocation[n][m], maxNeed[n][m], available[m];
   int need[n][m];
   cout << "\nEnter Allocation Matrix (" << n << "x" << m << "):\n";
   for (int i = 0; i < n; i++)
      for (int j = 0; j < m; j++)
         cin >> allocation[i][j];
   cout << "\nEnter Max Matrix (" << n << "x" << m << "):\n";
   for (int i = 0; i < n; i++)
      for (int j = 0; j < m; j++)
         cin >> maxNeed[i][j];
   cout << "\nEnter Available Resources (" << m << " values):\n";
   for (int j = 0; j < m; j++)
      cin >> available[j];
   // Calculate Need matrix
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < m; j++)
      {
         need[i][j] = maxNeed[i][j] - allocation[i][j];
      }
   }
   // Work array = available
   int work[m];
   for (int j = 0; j < m; j++)
      work[j] = available[j];
   bool finish[n];
   for (int i = 0; i < n; i++)
      finish[i] = false;
   int safeSeq[n], count = 0;
   // Safety Algorithm
   bool found;
   while (count < n)
   {
      found = false;
      for (int i = 0; i < n; i++)
      {
            if (!finish[i])
            {
                int j;
                for (j = 0; j < m; j++)
                {
                    if (need[i][j] > work[j])
                        break;
                }
                if (j == m)
                {
                    // process i can be satisfied
                    for (int k = 0; k < m; k++)
                        work[k] += allocation[i][k];
                    safeSeq[count++] = i;
                    finish[i] = true;
                    found = true;
                }
            }
        }
        if (!found)
            break; // no process could be satisfied
    }
    // Check if system is safe
    if (count == n)
    {
        cout << "\nSystem is in a SAFE STATE." << endl;
        cout << "Safe Sequence: ";
        for (int i = 0; i < n; i++)
        {
            cout << "P" << safeSeq[i];
            if (i != n - 1)
                cout << " -> ";
        }
        cout << endl;
    }
    else
    {
        cout << "\nSystem is in DEADLOCK (Not Safe)." << endl;
    }
    return 0;
}
```

Output:



C:\csit\fourth_sem_jonash\OS

```
===================================
||   Deadlock Detection Program   ||
|| Compiled by :- Jonash Chataut  ||
===================================

Enter number of processes: 5
Enter number of resources: 3

Enter Allocation Matrix (5x3):
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Enter Max Matrix (5x3):
7 5 3
3 2 2
9 0 2
2 2 2
4 3 3

Enter Available Resources (3 values):
5 5 3

System is in a SAFE STATE.
Safe Sequence: P1 -> P2 -> P3 -> P4 -> P0
```

MBM
Madan Bhandari Memorial College

Lab No. 14 **Date: 2025/09/09**

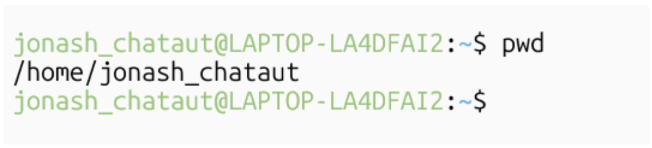
Title: Prepare a lab report for basic linux commands.

Introduction:

Linux is a widely used open-source operating system that provides a powerful command-line interface (CLI) to interact with the system. Unlike GUI-based systems, the Linux shell allows users to execute commands directly, which provides better control and flexibility. Learning basic Linux commands is essential for system navigation, file management, process control, and overall system administration.

Basic linux commands:

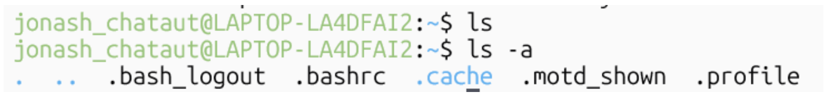
1) pwd → Show current working directory



```
jonash_chataut@LAPTOP-LA4DFAI2:~$ pwd
/home/jonash_chataut
jonash_chataut@LAPTOP-LA4DFAI2:~$
```

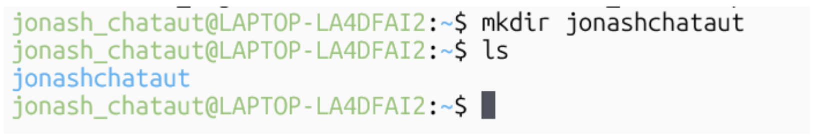
2) ls → List files and directories

ls -a → Show hidden files



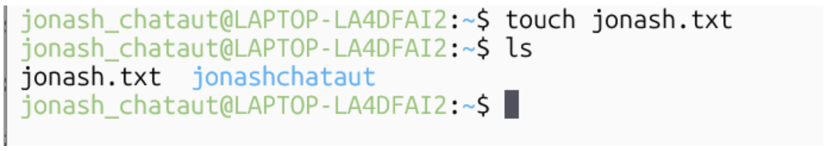
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ ls
jonash_chataut@LAPTOP-LA4DFAI2:~$ ls -a
.  ..  .bash_logout  .bashrc  .cache  .motd_shown  .profile
```

3) mkdir <dir> → Create new directory



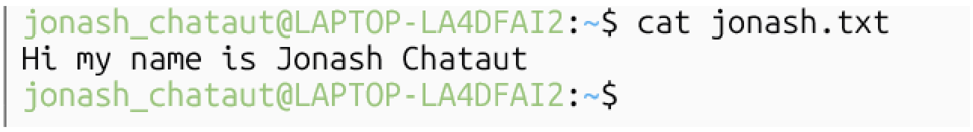
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ mkdir jonashchataut
jonash_chataut@LAPTOP-LA4DFAI2:~$ ls
jonashchataut
jonash_chataut@LAPTOP-LA4DFAI2:~$
```

4) touch <file> → Create empty file / update timestamp



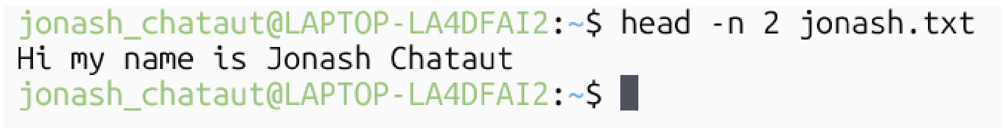
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ touch jonash.txt
jonash_chataut@LAPTOP-LA4DFAI2:~$ ls
jonash.txt  jonashchataut
jonash_chataut@LAPTOP-LA4DFAI2:~$
```

5) cat <file> → Display file contents



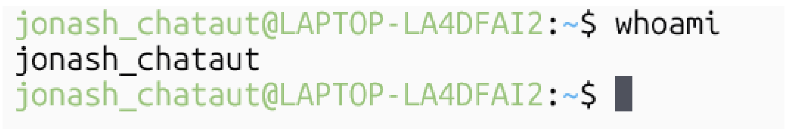
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ cat jonash.txt
Hi my name is Jonash Chataut
jonash_chataut@LAPTOP-LA4DFAI2:~$
```

6) head <file> → Show first 10 lines



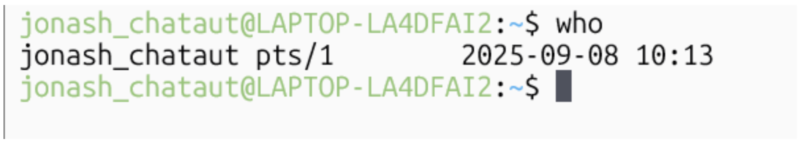
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ head -n 2 jonash.txt
Hi my name is Jonash Chataut
jonash_chataut@LAPTOP-LA4DFAI2:~$
```

7) whoami → Show current user



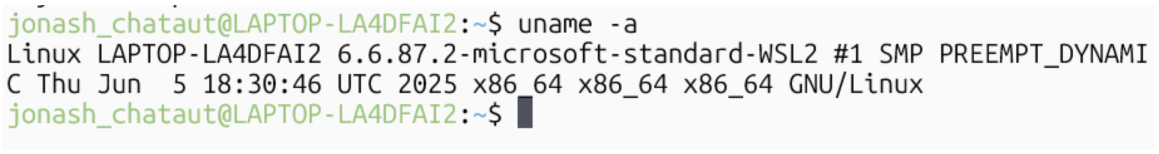
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ whoami
jonash_chataut
jonash_chataut@LAPTOP-LA4DFAI2:~$ █
```

8) who → Show logged-in users



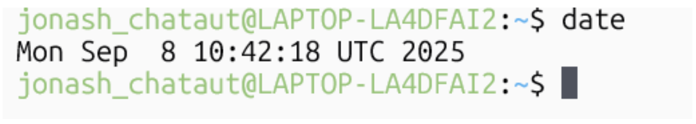
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ who
jonash_chataut pts/1        2025-09-08 10:13
jonash_chataut@LAPTOP-LA4DFAI2:~$ █
```

9) uname -a → Show system info (kernel, OS)



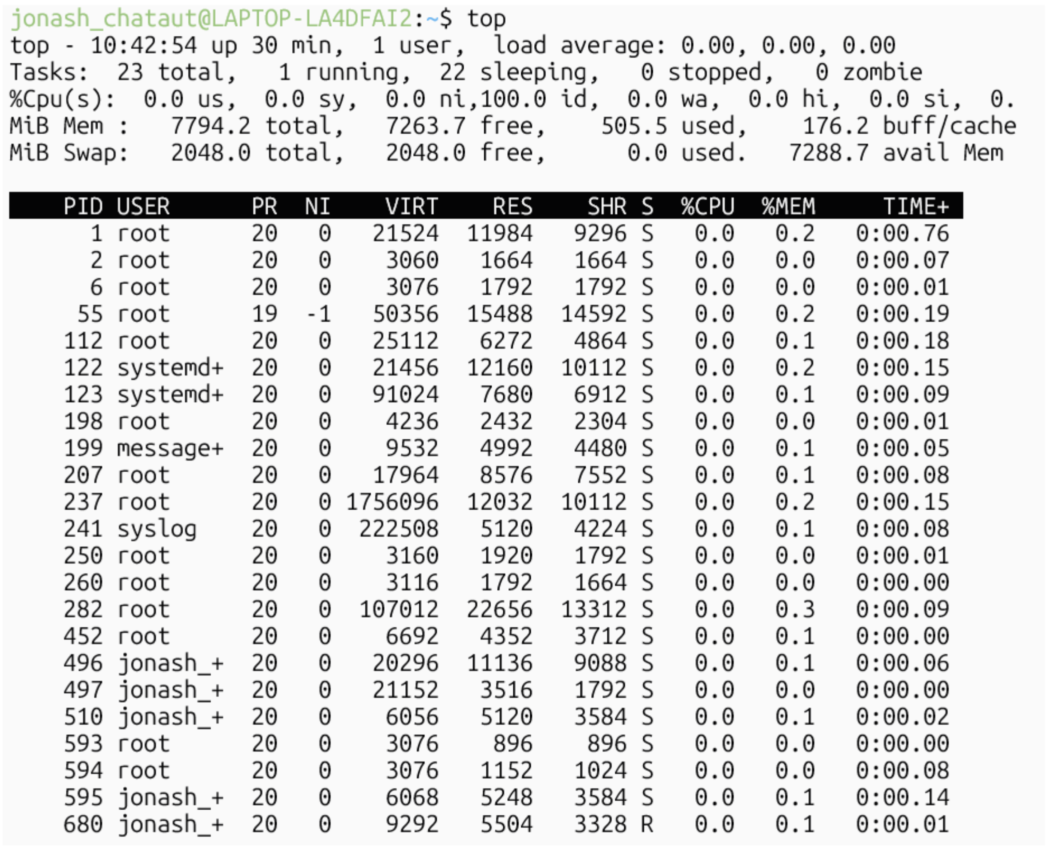
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ uname -a
Linux LAPTOP-LA4DFAI2 6.6.87.2-microsoft-standard-WSL2 #1 SMP PREEMPT_DYNAMI
C Thu Jun  5 18:30:46 UTC 2025 x86_64 x86_64 x86_64 GNU/Linux
jonash_chataut@LAPTOP-LA4DFAI2:~$ █
```

10) date → Show current date and time



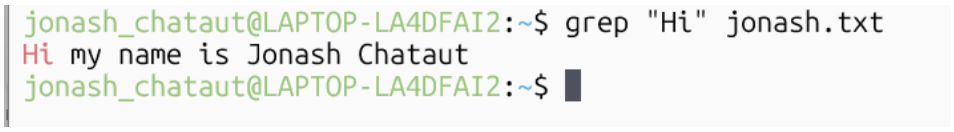
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ date
Mon Sep  8 10:42:18 UTC 2025
jonash_chataut@LAPTOP-LA4DFAI2:~$ █
```

11) top → Live view of processes



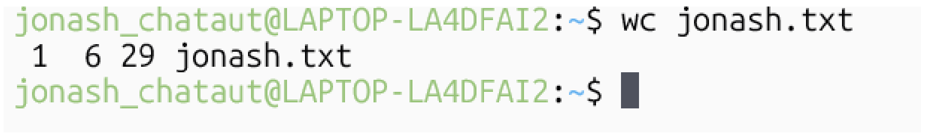
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ top
top - 10:42:54 up 30 min,  1 user,  load average: 0.00, 0.00, 0.00
Tasks:  23 total,   1 running,  22 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.
MiB Mem :   7794.2 total,   7263.7 free,    505.5 used,    176.2 buff/cache
MiB Swap:   2048.0 total,   2048.0 free,      0.0 used.   7288.7 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+
      1 root      20   0   21524  11984   9296 S   0.0   0.2   0:00.76
      2 root      20   0    3060   1664   1664 S   0.0   0.0   0:00.07
      6 root      20   0    3076   1792   1792 S   0.0   0.0   0:00.01
     55 root      19  -1   50356  15488  14592 S   0.0   0.2   0:00.19
    112 root      20   0   25112   6272   4864 S   0.0   0.1   0:00.18
    122 systemd+  20   0   21456  12160  10112 S   0.0   0.2   0:00.15
    123 systemd+  20   0   91024   7680   6912 S   0.0   0.1   0:00.09
    198 root      20   0    4236   2432   2304 S   0.0   0.0   0:00.01
    199 message+  20   0    9532   4992   4480 S   0.0   0.1   0:00.05
    207 root      20   0   17964   8576   7552 S   0.0   0.1   0:00.08
    237 root      20   0 1756096  12032  10112 S   0.0   0.2   0:00.15
    241 syslog    20   0  222508   5120   4224 S   0.0   0.1   0:00.08
    250 root      20   0    3160   1920   1792 S   0.0   0.0   0:00.01
    260 root      20   0    3116   1792   1664 S   0.0   0.0   0:00.00
    282 root      20   0  107012  22656  13312 S   0.0   0.3   0:00.09
    452 root      20   0    6692   4352   3712 S   0.0   0.1   0:00.00
    496 jonash_+  20   0   20296  11136   9088 S   0.0   0.1   0:00.06
    497 jonash_+  20   0   21152   3516   1792 S   0.0   0.0   0:00.00
    510 jonash_+  20   0    6056   5120   3584 S   0.0   0.1   0:00.02
    593 root      20   0    3076    896    896 S   0.0   0.0   0:00.00
    594 root      20   0    3076   1152   1024 S   0.0   0.0   0:00.08
    595 jonash_+  20   0    6068   5248   3584 S   0.0   0.1   0:00.14
    680 jonash_+  20   0    9292   5504   3328 R   0.0   0.1   0:00.01
```

12) grep <pattern> <file> → Search text in file.



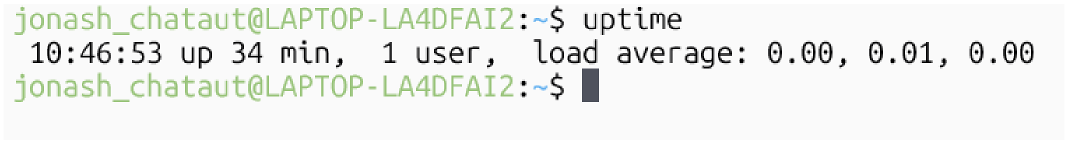
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ grep "Hi" jonash.txt
Hi my name is Jonash Chataut
jonash_chataut@LAPTOP-LA4DFAI2:~$ █
```

13) wc <file> → Count words, lines, characters



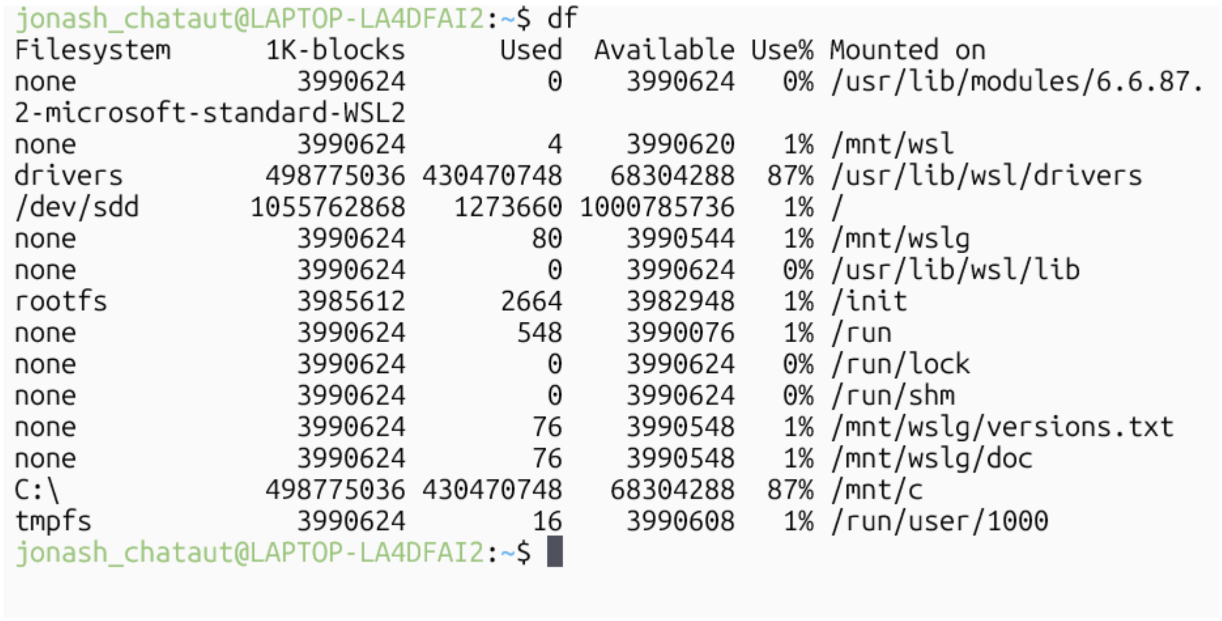
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ wc jonash.txt
 1  6 29 jonash.txt
jonash_chataut@LAPTOP-LA4DFAI2:~$ 
```

14) uptime → Show system uptime



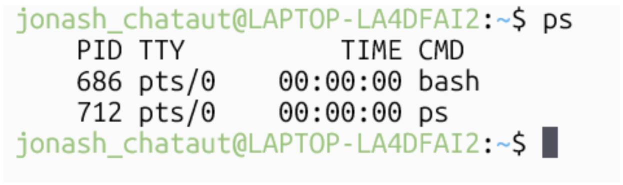
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ uptime
 10:46:53 up 34 min,  1 user,  load average: 0.00, 0.01, 0.00
jonash_chataut@LAPTOP-LA4DFAI2:~$ 
```

15) df → Checks your Linux system's disk usage



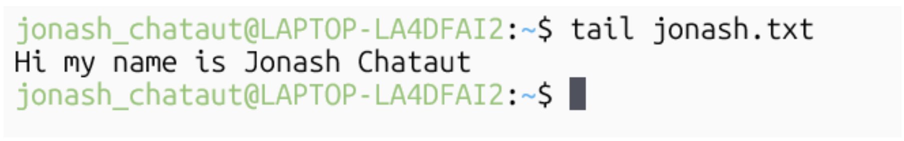
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ df
Filesystem      1K-blocks      Used  Available Use% Mounted on
none              3990624         0    3990624   0% /usr/lib/modules/6.6.87.
2-microsoft-standard-WSL2
none              3990624         4    3990620   1% /mnt/wsl
drivers         498775036 430470748   68304288  87% /usr/lib/wsl/drivers
/dev/sdd       1055762868   1273660 1000785736   1% /
none              3990624        80    3990544   1% /mnt/wslg
none              3990624         0    3990624   0% /usr/lib/wsl/lib
rootfs            3985612      2664    3982948   1% /init
none              3990624       548    3990076   1% /run
none              3990624         0    3990624   0% /run/lock
none              3990624         0    3990624   0% /run/shm
none              3990624        76    3990548   1% /mnt/wslg/versions.txt
none              3990624        76    3990548   1% /mnt/wslg/doc
C:\             498775036 430470748   68304288  87% /mnt/c
tmpfs             3990624        16    3990608   1% /run/user/1000
jonash_chataut@LAPTOP-LA4DFAI2:~$ 
```

16) ps → Summarizes the status of all running processes in your Linux system at a specific time



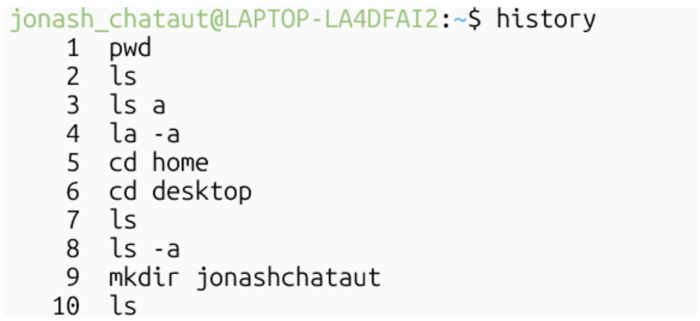
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ ps
    PID TTY          TIME CMD
    686 pts/0    00:00:00 bash
    712 pts/0    00:00:00 ps
jonash_chataut@LAPTOP-LA4DFAI2:~$ 
```

17) tail → It view the last line.



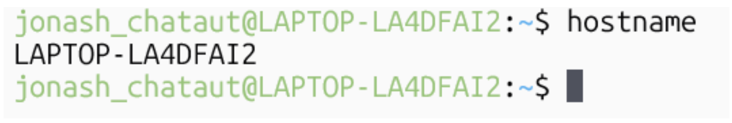
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ tail jonash.txt
Hi my name is Jonash Chataut
jonash_chataut@LAPTOP-LA4DFAI2:~$ 
```

18) history → Check previously run utilities.



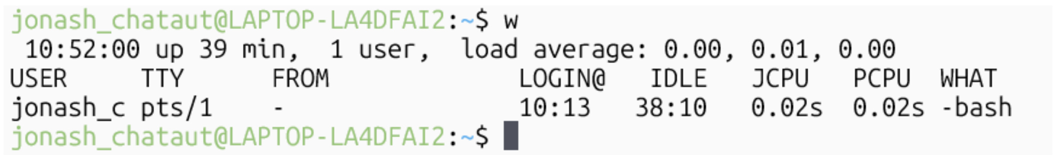
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ history
    1  pwd
    2  ls
    3  ls a
    4  la -a
    5  cd home
    6  cd desktop
    7  ls
    8  ls -a
    9  mkdir jonashchataut
   10  ls
```

19) hostname → Show hostname



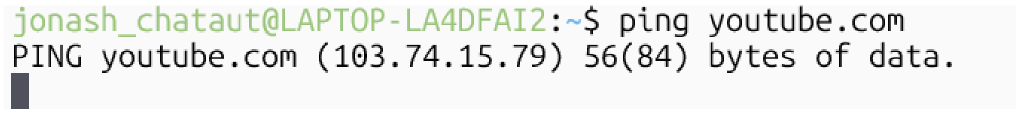
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ hostname
LAPTOP-LA4DFAI2
jonash_chataut@LAPTOP-LA4DFAI2:~$
```

20) w → Detailed user sessions



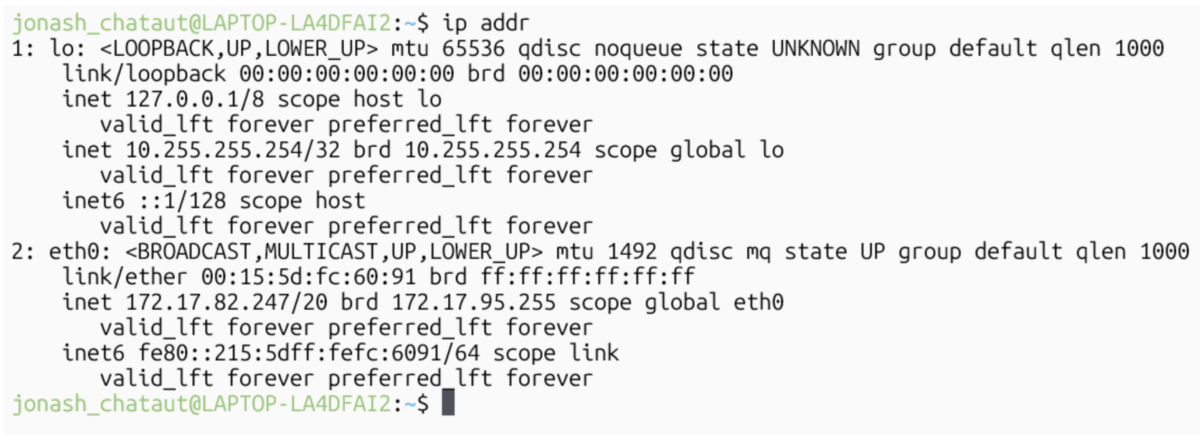
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ w
 10:52:00 up 39 min,  1 user,  load average: 0.00, 0.01, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU  WHAT
jonash_c pts/1    -                10:13   38:10   0.02s  0.02s -bash
jonash_chataut@LAPTOP-LA4DFAI2:~$
```

21) ping → Check connectivity



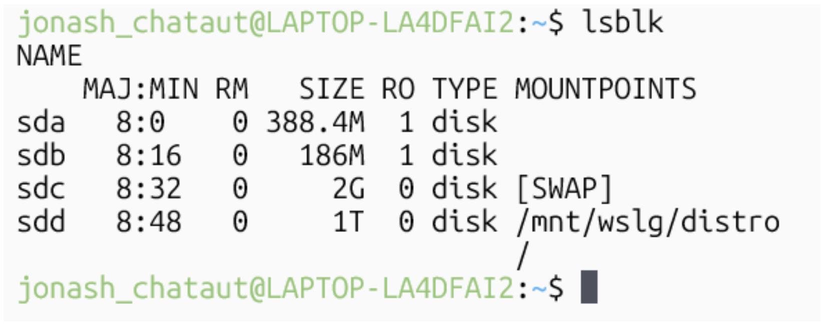
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ ping youtube.com
PING youtube.com (103.74.15.79) 56(84) bytes of data.
```

22) ip addr → Show IP details



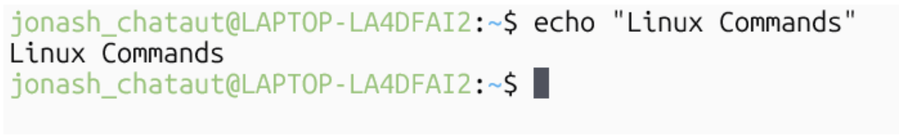
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet 10.255.255.254/32 brd 10.255.255.254 scope global lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1492 qdisc mq state UP group default qlen 1000
    link/ether 00:15:5d:fc:60:91 brd ff:ff:ff:ff:ff:ff
    inet 172.17.82.247/20 brd 172.17.95.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::215:5dff:fefc:6091/64 scope link
       valid_lft forever preferred_lft forever
jonash_chataut@LAPTOP-LA4DFAI2:~$
```

23) lsblk → List block devices



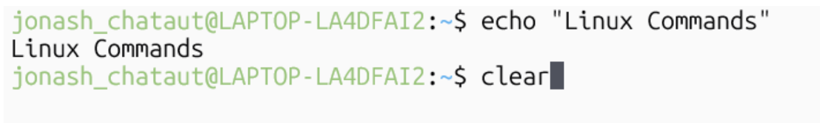
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ lsblk
NAME
    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda    8:0    0 388.4M  1 disk
sdb    8:16   0   186M  1 disk
sdc    8:32   0     2G  0 disk [SWAP]
sdd    8:48   0     1T  0 disk /mnt/wslg/distro
                               /
jonash_chataut@LAPTOP-LA4DFAI2:~$
```

24) echo → Print text

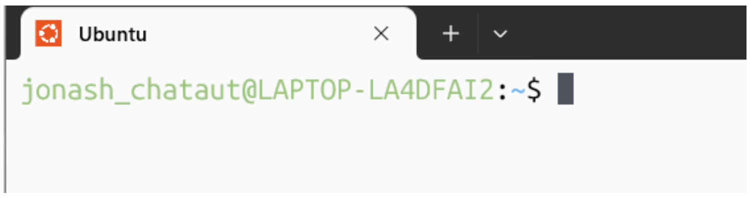


```
jonash_chataut@LAPTOP-LA4DFAI2:~$ echo "Linux Commands"
Linux Commands
jonash_chataut@LAPTOP-LA4DFAI2:~$
```

25) clear → Clear terminal



```
jonash_chataut@LAPTOP-LA4DFAI2:~$ echo "Linux Commands"
Linux Commands
jonash_chataut@LAPTOP-LA4DFAI2:~$ clear
```



```
jonash_chataut@LAPTOP-LA4DFAI2:~$
```

Lab No. 15 **Date: 2025/09/09**

Title: Prepare a lab report to create and terminate process.

Introduction:

A process is an instance of a program running, and its lifecycle includes various stages such as creation, execution, and deletion.

- The operating system handles process creation by allocating necessary resources and assigning each process a unique identifier.
- Process deletion involves releasing resources once a process completes its execution.
- Processes are often organized in a hierarchy, where parent processes create child processes, forming a tree-like structure.

Process Concepts:

- Process**:** An executing program with its own memory space
- Process ID (PID)**:** Unique identifier assigned to each process
- Parent Process**:** Process that creates another process
- Child Process**:** Process created by another process
- Process State**:** Current status of a process (running, sleeping, stopped, zombie)

Process Termination Methods:

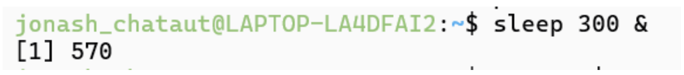
1. Natural Termination**:** Process completes execution
2. Signal Termination**:** Using kill signals
3. Force Termination**:** Forceful process killing
4. Parent Termination**:** When parent process ends

Operating System: Ubuntu

1. Process creation

sleep 300 &   *# Create a sleep process for 300 seconds*

[1] 570   *# Process id*



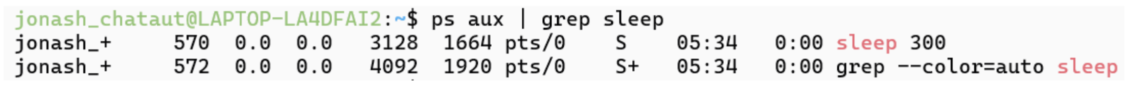
```
jonash_chataut@LAPTOP-LA4DFAI2:~$ sleep 300 &
[1] 570
```

2. Viewing Processes

ps aux | grep sleep   *# Find specific process*

jonash_+   570 0.0 0.0   3128 1664 pts/0   S   05:34   0:00 sleep 300 #process info

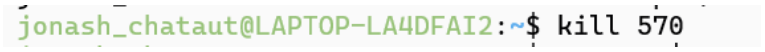
jonash_+   572 0.0 0.0   4092 1920 pts/0   S+   05:34   0:00 grep --color=auto sleep



```
jonash_chataut@LAPTOP-LA4DFAI2:~$ ps aux | grep sleep
jonash_+     570  0.0  0.0   3128  1664 pts/0    S    05:34   0:00 sleep 300
jonash_+     572  0.0  0.0   4092  1920 pts/0    S+   05:34   0:00 grep --color=auto sleep
```

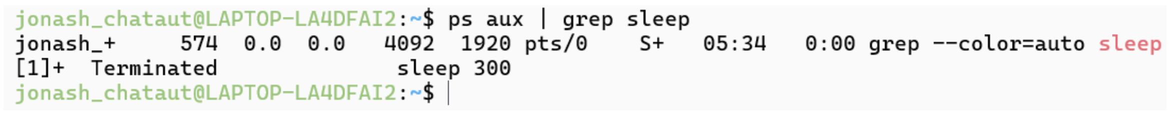
3. Terminating the process

kill 570 *# Stop process by ID number*



```
jonash_chataut@LAPTOP-LA4DFAI2:~$ kill 570
```

4. Checking if the process is terminated or not



```
jonash_chataut@LAPTOP-LA4DFAI2:~$ ps aux | grep sleep
jonash_+     574  0.0  0.0   4092  1920 pts/0    S+   05:34   0:00 grep --color=auto sleep
[1]+  Terminated              sleep 300
jonash_chataut@LAPTOP-LA4DFAI2:~$ 
```