Lab no:1 **Date: 2082/04/27**

Title: Write a program to display the no. of page faults for user input reference string and frame size using FIFO.

Introduction:

FIFO (First In, First Out) Page Replacement

FIFO is one of the simplest page replacement algorithms used in memory management. The page that entered the memory earliest is removed first when a new page needs to be loaded.

Steps:

- Keep track of the order in which pages enter memory.
- When memory is full and a new page is needed, remove the oldest page.

Page Hit

A page hit happens when the page requested by the CPU is already in memory (RAM).

Page Fault

A page fault occurs when the requested page is not in memory, so it must be loaded from the slower secondary storage (like a hard disk).

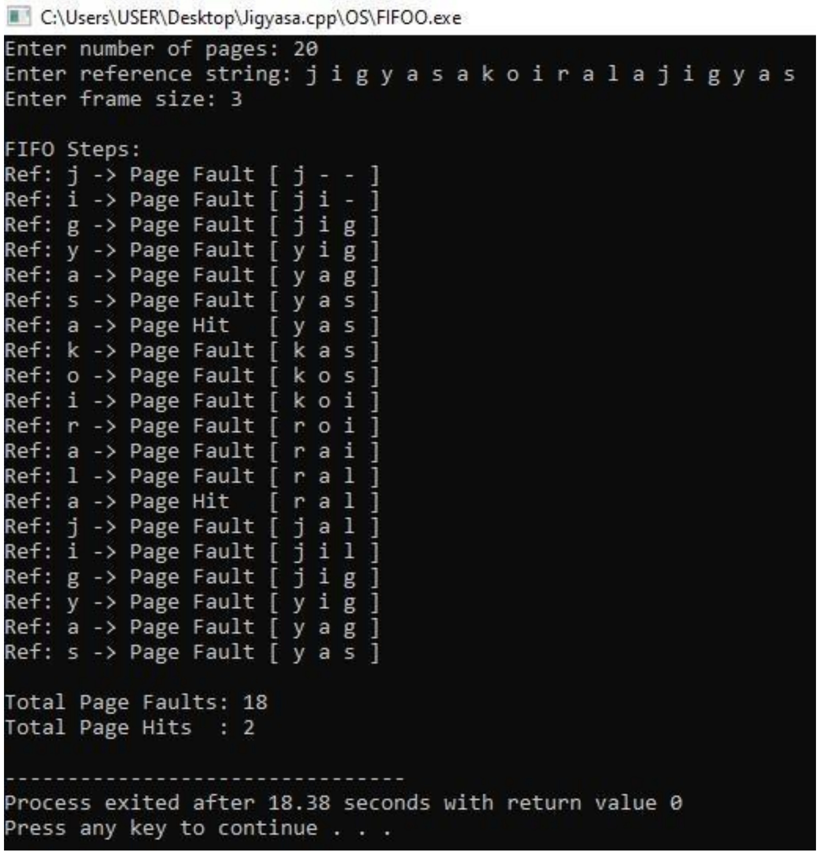
Compiler: Dev C++
Language: C

Source code:

```
#include <stdio.h>

#define MAX 100


// Check if a page is already in frame int isInFrame(char
frame[], int frameSize, char page) {

   for (int i = 0; i < frameSize; i++)
if (frame[i] == page) return 1;     return
0;

}


// Print current frame void printFrame(char
frame[], int frameSize) {

   printf("[ ");

   for (int i = 0; i < frameSize; i++)

       (frame[i] == '\0') ? printf("- ") : printf("%c ", frame[i]);
printf("]");

}


// FIFO Page Replacement

void fifoPageReplacement(char ref[], int n, int frameSize) {
char frame[frameSize];     int front = 0, faults = 0, hits = 0;


   for (int i = 0; i < frameSize; i++) frame[i] = '\0';


   printf("\nFIFO Steps:\n");     for (int i = 0; i <
n; i++) {        printf("Ref: %c -> ", ref[i]);        if
(!isInFrame(frame, frameSize, ref[i])) {
frame[front] = ref[i];            front = (front + 1)
% frameSize;
```

```
            faults++;
printf("Page Fault ");

        } else {
hits++;

            printf("Page Hit   ");

        }

        printFrame(frame, frameSize);

        printf("\n");

    }


    printf("\nTotal Page Faults: %d", faults);

    printf("\nTotal Page Hits  : %d\n", hits);

}


int main() {     char
ref[MAX];

    int n, frameSize;


    printf("Enter number of pages: ");
scanf("%d", &n);


    printf("Enter reference string: ");

    for (int i = 0; i < n; i++) scanf(" %c", &ref[i]);


    printf("Enter frame size: ");     scanf("%d",
&frameSize);


    fifoPageReplacement(ref, n, frameSize);
return 0;

}
```

Output:



```
C:\Users\USER\Desktop\Jigyasa.cpp\OS\FIFOO.exe
Enter number of pages: 20
Enter reference string: j i g y a s a k o i r a l a j i g y a s
Enter frame size: 3

FIFO Steps:
Ref: j -> Page Fault [ j - - ]
Ref: i -> Page Fault [ j i - ]
Ref: g -> Page Fault [ j i g ]
Ref: y -> Page Fault [ y i g ]
Ref: a -> Page Fault [ y a g ]
Ref: s -> Page Fault [ y a s ]
Ref: a -> Page Hit   [ y a s ]
Ref: k -> Page Fault [ k a s ]
Ref: o -> Page Fault [ k o s ]
Ref: i -> Page Fault [ k o i ]
Ref: r -> Page Fault [ r o i ]
Ref: a -> Page Fault [ r a i ]
Ref: l -> Page Fault [ r a l ]
Ref: a -> Page Hit   [ r a l ]
Ref: j -> Page Fault [ j a l ]
Ref: i -> Page Fault [ j i l ]
Ref: g -> Page Fault [ j i g ]
Ref: y -> Page Fault [ y i g ]
Ref: a -> Page Fault [ y a g ]
Ref: s -> Page Fault [ y a s ]

Total Page Faults: 18
Total Page Hits  : 2

--------------------------------
Process exited after 18.38 seconds with return value 0
Press any key to continue . . .
```

Lab no: 2 **Date: 2082/04/27**

Title: Write a program to display the no. of page faults for user input reference string and frame size using OPR.

Introduction :

OPR (Optimal Page Replacement)

When a page fault occurs, replace the page that will not be used for the longest time in the future.

Limitation:

Needs knowledge of future page requests so it’s mostly used for theoretical comparison, not in real systems.

Page Hit

CPU requests a page that is already in memory.

Page Fault

CPU requests a page not in memory.

Compiler: Dev C++
Language: C

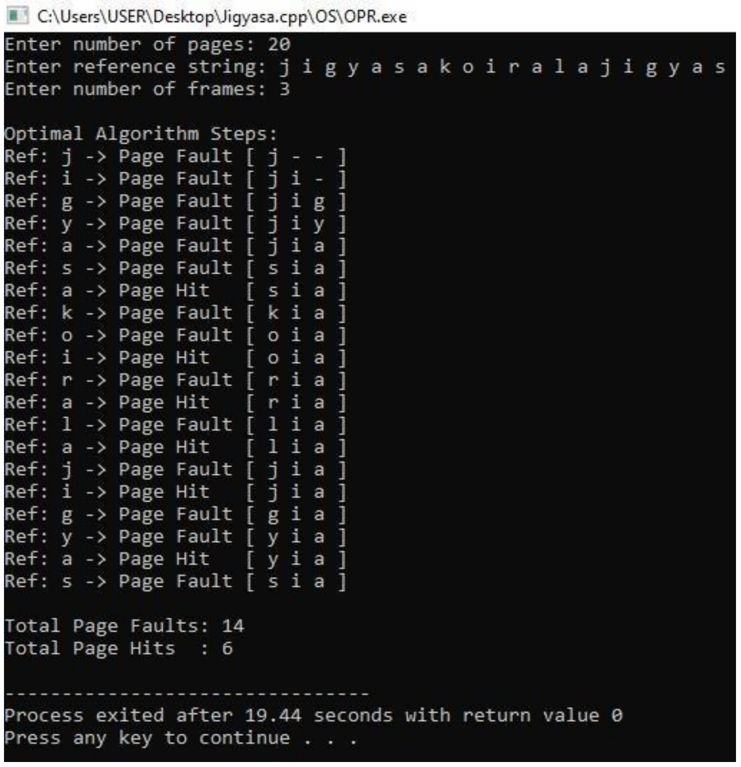
Source code:

```
#include <stdio.h>

#define MAX 100


// Check if page is in frame int inFrame(char
frame[], int fSize, char page) {     for (int i = 0; i <
fSize; i++)         if (frame[i] == page) return 1;
return 0;

}


// Find index to replace (Optimal)

int findOptimal(char frame[], int fSize, char ref[], int n, int curr) {

   int farthest = curr, idx = -1;

   for (int i = 0; i < fSize; i++) {
int j;

       for (j = curr + 1; j < n; j++)
if (frame[i] == ref[j]) {

               if (j > farthest) { farthest = j; idx = i; }

               break;

           }

       if (j == n) return i; // never used again

       if (idx == -1) idx = 0;

   }

   return idx;

}


// Print frame content void
printFrame(char frame[], int fSize) {
printf("[ ");

   for (int i = 0; i < fSize; i++)
```

```
        (frame[i] == '\0') ? printf("- ") : printf("%c ", frame[i]);
printf("]");

}


// Optimal Page Replacement

void optimal(char ref[], int n, int fSize) {     char
frame[fSize];

    int faults = 0, hits = 0;


    for (int i = 0; i < fSize; i++) frame[i] = '\0';


    printf("\nOptimal Algorithm Steps:\n");     for (int i = 0; i < n; i++)
{        printf("Ref: %c -> ", ref[i]);        if (!inFrame(frame, fSize,
ref[i])) {            int placed = 0;            for (int j = 0; j < fSize; j++)
if (frame[j] == '\0') { frame[j] = ref[i]; placed = 1; break; }            if
(!placed) frame[findOptimal(frame, fSize, ref, n, i)] = ref[i];
faults++;            printf("Page Fault ");

        } else {
hits++;

            printf("Page Hit   ");

        }

        printFrame(frame, fSize);

        printf("\n");

    }


    printf("\nTotal Page Faults: %d", faults);

    printf("\nTotal Page Hits  : %d\n", hits);

}


int main() {     char
ref[MAX];
```

```
    int n, fSize;

    printf("Enter number of pages: ");
scanf("%d", &n);

    printf("Enter reference string: ");     for (int i =
0; i < n; i++) scanf(" %c", &ref[i]);

    printf("Enter number of frames: ");
scanf("%d", &fSize);

    optimal(ref, n, fSize);     return
0;
}
```

Output:



C:\Users\USER\Desktop\Jigyasa.cpp\OS\OPR.exe

```
Enter number of pages: 20
Enter reference string: j i g y a s a k o i r a l a j i g y a s
Enter number of frames: 3

Optimal Algorithm Steps:
Ref: j -> Page Fault [ j - - ]
Ref: i -> Page Fault [ j i - ]
Ref: g -> Page Fault [ j i g ]
Ref: y -> Page Fault [ j i y ]
Ref: a -> Page Fault [ j i a ]
Ref: s -> Page Fault [ s i a ]
Ref: a -> Page Hit   [ s i a ]
Ref: k -> Page Fault [ k i a ]
Ref: o -> Page Fault [ o i a ]
Ref: i -> Page Hit   [ o i a ]
Ref: r -> Page Fault [ r i a ]
Ref: a -> Page Hit   [ r i a ]
Ref: l -> Page Fault [ l i a ]
Ref: a -> Page Hit   [ l i a ]
Ref: j -> Page Fault [ j i a ]
Ref: i -> Page Hit   [ j i a ]
Ref: g -> Page Fault [ g i a ]
Ref: y -> Page Fault [ y i a ]
Ref: a -> Page Hit   [ y i a ]
Ref: s -> Page Fault [ s i a ]

Total Page Faults: 14
Total Page Hits  : 6

--------------------------------
Process exited after 19.44 seconds with return value 0
Press any key to continue . . .
```

Lab no: 3 **Date: 2082/04/27**

Title: **Write a program to display the no. of page faults for user input reference string and frame size using LRU.**

Introduction :

LRU (Least Recently Used) Page Replacement

When a page needs to be replaced, LRU removes the page that has not been used for the longest time in the past.

Past behavior predicts near future use if a page hasn't been used for a while, it's less likely to be used soon.

Page Hit

CPU requests a page that is already in memory.

Page Fault

CPU requests a page that is not in memory.

Compiler: Dev C++
Language: C

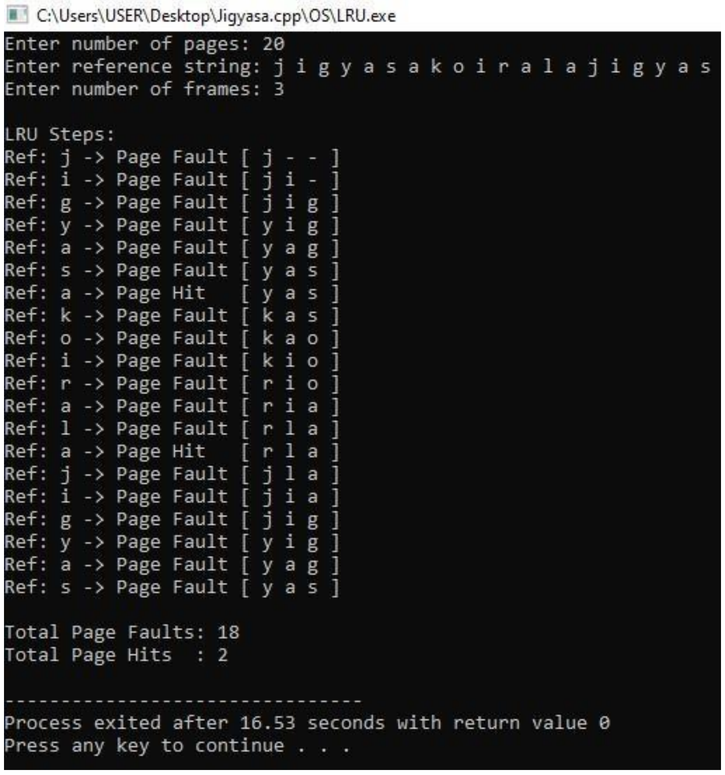
Source code:

```
#include <stdio.h>

#define MAX 100


// Check if page is in frame int inFrame(char
frame[], int fSize, char page) {     for (int i = 0; i <
fSize; i++)         if (frame[i] == page) return 1;
return 0;

}


// Find Least Recently Used index

int findLRU(char frame[], int fSize, char ref[], int curr) {

   int min = curr, idx = -1;

   for (int i = 0; i < fSize; i++) {
int j;

       for (j = curr - 1; j >= 0; j--)            if
(frame[i] == ref[j]) {                 if (j < min)
{ min = j; idx = i; }

             break;

          }

       if (j < 0) return i; // never used before

   }

   return idx;

}


// Print frame

void printFrame(char frame[], int fSize) {
printf("[ ");

   for (int i = 0; i < fSize; i++)
```

```
        (frame[i] == '\0') ? printf("- ") : printf("%c ", frame[i]);
printf("]");

}


// LRU Algorithm void LRU(char ref[],
int n, int fSize) {     char frame[fSize];

    int faults = 0, hits = 0;


    for (int i = 0; i < fSize; i++) frame[i] = '\0';


    printf("\nLRU Steps:\n");     for (int i =
0; i < n; i++) {         printf("Ref: %c -> ",
ref[i]);         if (!inFrame(frame, fSize,
ref[i])) {             int placed = 0;             for
(int j = 0; j < fSize; j++)

                if (frame[j] == '\0') { frame[j] = ref[i]; placed = 1; break; }
if (!placed) frame[findLRU(frame, fSize, ref, i)] = ref[i];
faults++;             printf("Page Fault ");

        } else {
hits++;
printf("Page Hit   ");

        }

        printFrame(frame, fSize);

        printf("\n");

    }

    printf("\nTotal Page Faults: %d", faults);

    printf("\nTotal Page Hits  : %d\n", hits);

}


int main() {     char
ref[MAX];
```

```
    int n, fSize;

    printf("Enter number of pages: ");
scanf("%d", &n);

    printf("Enter reference string: ");     for (int i =
0; i < n; i++) scanf(" %c", &ref[i]);

    printf("Enter number of frames: ");
scanf("%d", &fSize);

    LRU(ref, n, fSize);     return
0;
}
```

Output:



C:\Users\USER\Desktop\Jigyasa.cpp\OS\LRU.exe

```
Enter number of pages: 20
Enter reference string: j i g y a s a k o i r a l a j i g y a s
Enter number of frames: 3

LRU Steps:
Ref: j -> Page Fault [ j - - ]
Ref: i -> Page Fault [ j i - ]
Ref: g -> Page Fault [ j i g ]
Ref: y -> Page Fault [ y i g ]
Ref: a -> Page Fault [ y a g ]
Ref: s -> Page Fault [ y a s ]
Ref: a -> Page Hit   [ y a s ]
Ref: k -> Page Fault [ k a s ]
Ref: o -> Page Fault [ k a o ]
Ref: i -> Page Fault [ k i o ]
Ref: r -> Page Fault [ r i o ]
Ref: a -> Page Fault [ r i a ]
Ref: l -> Page Fault [ r l a ]
Ref: a -> Page Hit   [ r l a ]
Ref: j -> Page Fault [ j l a ]
Ref: i -> Page Fault [ j i a ]
Ref: g -> Page Fault [ j i g ]
Ref: y -> Page Fault [ y i g ]
Ref: a -> Page Fault [ y a g ]
Ref: s -> Page Fault [ y a s ]

Total Page Faults: 18
Total Page Hits  : 2

--------------------------------
Process exited after 16.53 seconds with return value 0
Press any key to continue . . .
```

Lab no: 4 **Date: 2082/05/10**

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

FCFS (First Come First Serve) Scheduling Algorithm

- FCFS is the simplest CPU scheduling algorithm.
- Processes are executed in the order in which they arrive.
- The first process that comes is the first to be given the CPU.
- It is non-preemptive (once a process starts executing, it runs till it finishes).
- Works like a queue at a ticket counter (first person to arrive gets served first).

Average Turnaround Time (TAT)

- Turnaround Time = Completion Time – Arrival Time
- It shows how long a process takes from the moment it arrives in the system until it finishes.
- Average Turnaround Time =Sum of Turnaround Times of all processes / Total number of processes

Waiting Time

Waiting Time of a process = Turnaround Time – Burst Time
It is the total time a process spends waiting in the ready queue (not executing, just waiting for CPU).

Algorithm

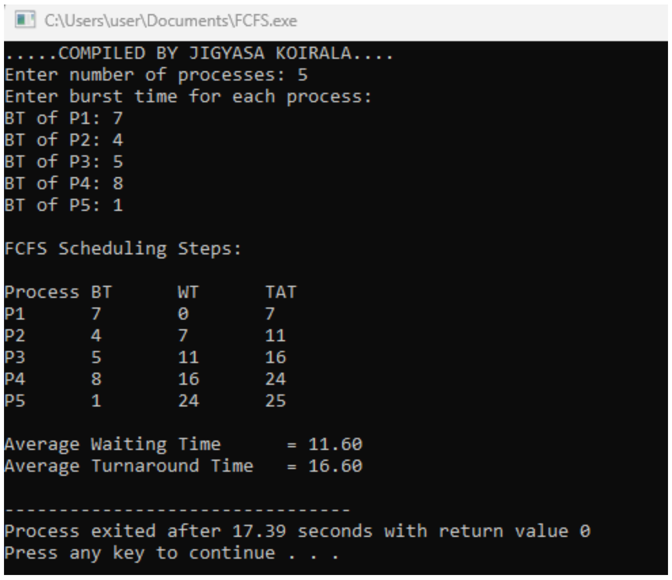
- Arrange processes in the order of their arrival.
- Execute the processes one by one.
- Calculate Completion Time (CT).
- Calculate Turnaround Time:
  TAT = CT – AT
- Calculate Waiting Time:
  WT = TAT – BT
- Find average TAT and WT.

Compiler: DEV C++
Language: C

Source Code:

```
    // Summation for average calculation
    for (int i = 0; i < n; i++) {
        totalWT += wt[i];
        totalTAT += tat[i];
    }
    // Print steps (just like your FIFO code)
    printf("\nFCFS Scheduling Steps:\n");
    printTable(pid, bt, wt, tat, n);
    // Print averages
    printf("\nAverage Waiting Time      = %.2f", (float) totalWT / n);
    printf("\nAverage Turnaround Time   = %.2f\n", (float) totalTAT / n);
}
int main() {
    int n, bt[MAX];
    printf(".....COMPILED BY JIGYASA KOIRALA....\n");
    printf("Enter number of processes: ");
    scanf("%d", &n);
    printf("Enter burst time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("BT of P%d: ", i + 1);
        scanf("%d", &bt[i]);
    }
    fcfsScheduling(bt, n);
    return 0;
}
```

Output:



```
C:\Users\user\Documents\FCFS.exe
.....COMPILED BY JIGYASA KOIRALA....
Enter number of processes: 5
Enter burst time for each process:
BT of P1: 7
BT of P2: 4
BT of P3: 5
BT of P4: 8
BT of P5: 1

FCFS Scheduling Steps:

Process BT      WT      TAT
P1      7       0       7
P2      4       7       11
P3      5       11      16
P4      8       16      24
P5      1       24      25

Average Waiting Time      = 11.60
Average Turnaround Time   = 16.60

--------------------------------
Process exited after 17.39 seconds with return value 0
Press any key to continue . . .
```

Lab no: 5

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

SJF (Shortest Job First) Scheduling Algorithm

SJF is a CPU scheduling algorithm in which the process with the shortest burst time (execution time) is selected next for execution.

Types of SJF

- **Non-preemptive SJF:** Once a process starts executing, it runs until completion.
- **Preemptive SJF** (Shortest Remaining Time First – SRTF)**:** A new process with a shorter burst time can interrupt the currently running process.

Advantages

- Gives minimum average waiting time.
- Efficient for short processes.

Disadvantages

- May cause starvation for long processes.
- Hard to know burst time in advance.

Average Turnaround Time (TAT)

- Turnaround time = Completion Time – Arrival Time
- It is the total time taken for a process from the moment it arrives in the system until it finishes executing.
- **Average Turnaround Time** = $\frac{\text{Sum of Turnaround Times of all processes}}{\text{Total number of processes}}$

Waiting Time (WT)

- Waiting time = Turnaround Time – Burst Time
- It is the total amount of time a process spends waiting in the ready queue before it gets the CPU.
- **Average Waiting Time** = $\frac{\text{Sum of waiting times of all processes}}{\text{Total number of processes}}$

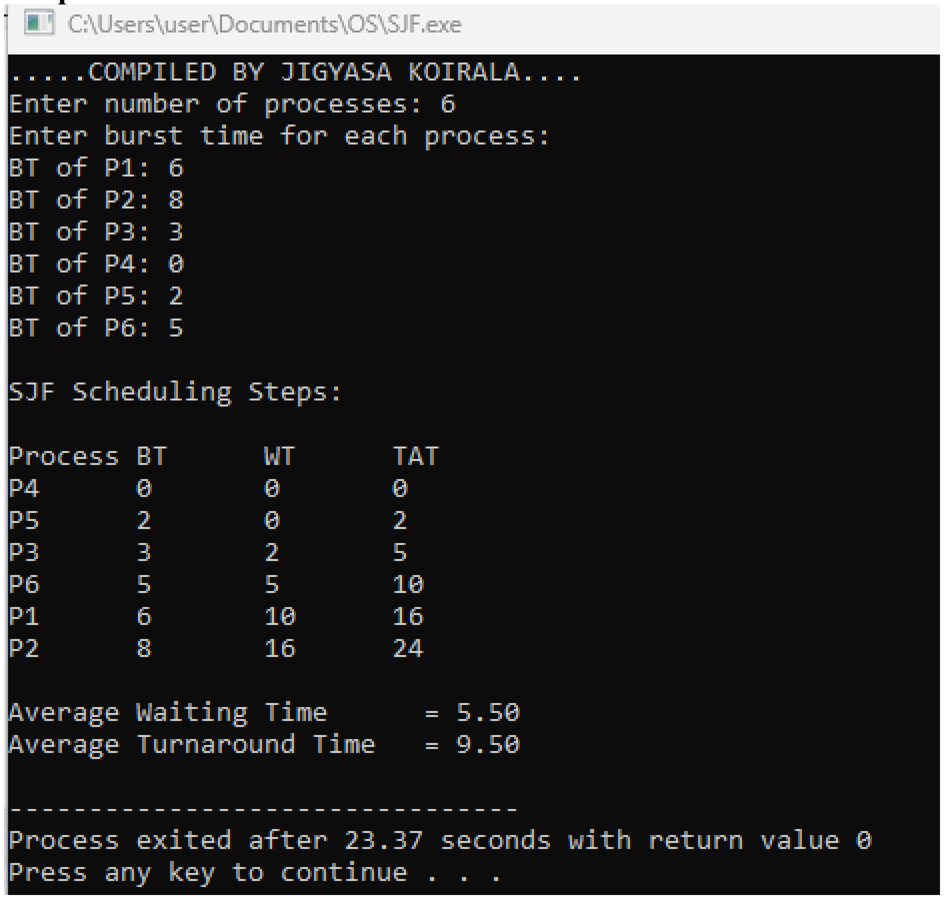
Compiler: DEV C++
Language: C

Source Code:

```
#include <stdio.h>
#define MAX 100

// Print result table
void printTable(int pid[], int bt[], int wt[], int tat[], int n) {
   printf("\nProcess\tBT\tWT\tTAT\n");
   for (int i = 0; i < n; i++) {
      printf("P%d\t%d\t%d\t%d\n", pid[i], bt[i], wt[i], tat[i]);
   }
}

// Non-Preemptive SJF Scheduling
void sjfScheduling(int bt[], int n) {
   int pid[MAX], wt[MAX], tat[MAX];
   int temp, totalWT = 0, totalTAT = 0;

   // Assign process IDs
   for (int i = 0; i < n; i++)
      pid[i] = i + 1;

   // Sort processes by burst time (SJF)
   for (int i = 0; i < n - 1; i++) {
      for (int j = 0; j < n - i - 1; j++) {
         if (bt[j] > bt[j + 1]) {

            // Swap burst times
            temp = bt[j];
            bt[j] = bt[j + 1];
            bt[j + 1] = temp;

            // Swap process IDs
            temp = pid[j];
            pid[j] = pid[j + 1];
            pid[j + 1] = temp;
         }
      }
   }

   // First process waiting time
   wt[0] = 0;

   // Compute waiting times
```

```
    for (int i = 1; i < n; i++) {
        wt[i] = wt[i - 1] + bt[i - 1];
    }

    // Compute turnaround times
    for (int i = 0; i < n; i++) {
        tat[i] = wt[i] + bt[i];
    }

    // Sum for averages
    for (int i = 0; i < n; i++) {
        totalWT += wt[i];
        totalTAT += tat[i];
    }

    // Display steps
    printf("\nSJF Scheduling Steps:\n");
    printTable(pid, bt, wt, tat, n);

    // Averages
    printf("\nAverage Waiting Time      = %.2f", (float) totalWT / n);
    printf("\nAverage Turnaround Time   = %.2f\n", (float) totalTAT / n);
}

int main() {
    int n, bt[MAX];
    printf(".....COMPILED BY JIGYASA KOIRALA....\n");
    printf("Enter number of processes: ");
    scanf("%d", &n);

    printf("Enter burst time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("BT of P%d: ", i + 1);
        scanf("%d", &bt[i]);
    }

    sjfScheduling(bt, n);

    return 0;
}
```

Output:



C:\Users\user\Documents\OS\SJF.exe

```
.....COMPILED BY JIGYASA KOIRALA....
Enter number of processes: 6
Enter burst time for each process:
BT of P1: 6
BT of P2: 8
BT of P3: 3
BT of P4: 0
BT of P5: 2
BT of P6: 5

SJF Scheduling Steps:

Process BT      WT      TAT
P4      0       0       0
P5      2       0       2
P3      3       2       5
P6      5       5       10
P1      6       10      16
P2      8       16      24

Average Waiting Time      = 5.50
Average Turnaround Time   = 9.50

--------------------------------
Process exited after 23.37 seconds with return value 0
Press any key to continue . . .
```

Lab no: 6 **Date: 2082/05/10**

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

Introduction:

RR (Round Robin) Process Scheduling Algorithm

Round Robin (RR) is a preemptive CPU scheduling algorithm.

- Each process gets a fixed time slice called Time Quantum (q).
- The CPU gives each process equal time in a circular order.
- If a process doesn't finish within the time quantum, it is preempted and put back at the end of the ready queue.
- Used in time-sharing systems to provide fairness to all processes.

Advantages:

- Fair (every process gets equal chance).
- No starvation.

Disadvantage:

- Too small time quantum → too many context switches.
- Too large time quantum → works like FCFS (not good)

Average Turnaround Time (TAT)

- Turnaround Time = Completion Time – Arrival Time
- It shows how long a process takes from the moment it arrives in the system until it finishes.
- Average Turnaround Time = Sum of turnaround time of all processes ÷ Number of processes

Average Waiting Time (WT)

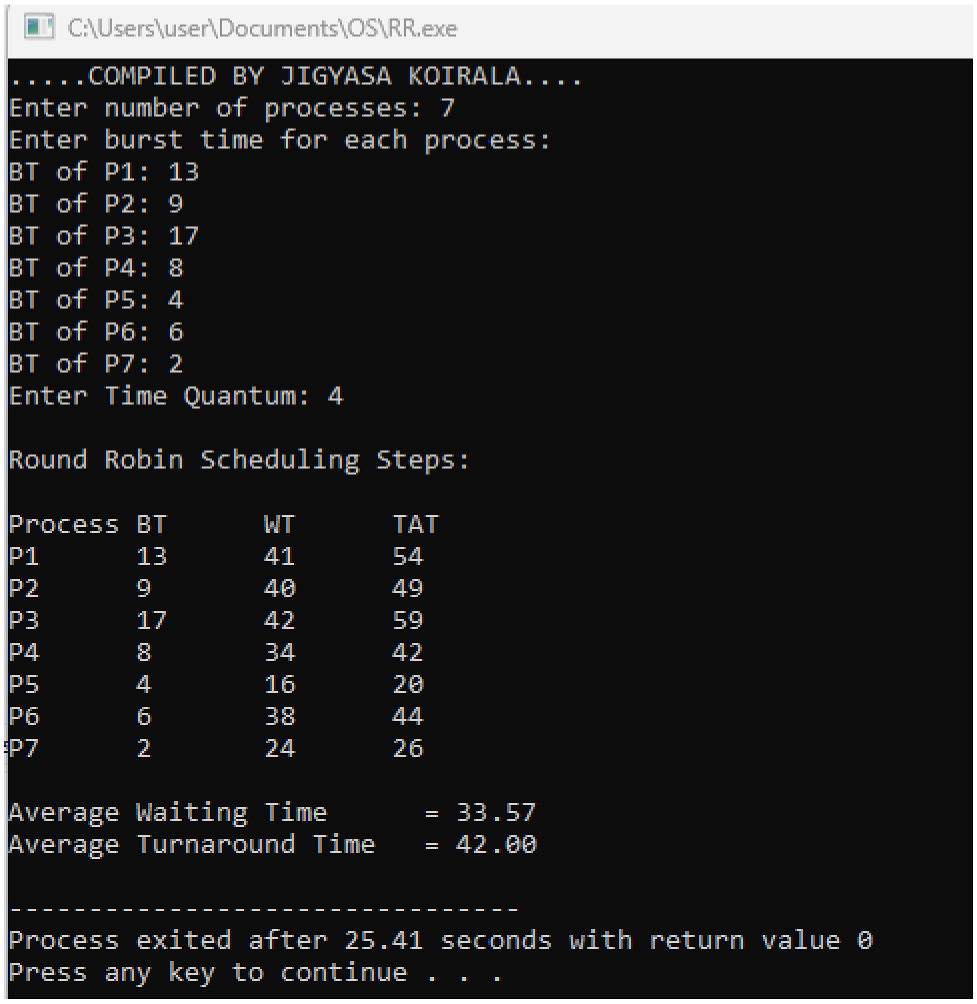
- Waiting Time = Turnaround Time – Burst Time
- It means the total time a process spends waiting in the ready queue (not running, not finished).
- Average Waiting Time = Sum of waiting time of all processes ÷ Number of processes

Compiler: DEV C++
Language: C

Source Code:

```
#include <stdio.h>
#define MAX 100

// Print table
void printTable(int pid[], int bt[], int wt[], int tat[], int n) {
    printf("\nProcess\tBT\tWT\tTAT\n");
    for (int i = 0; i < n; i++) {
        printf("P%d\t%d\t%d\t%d\n", pid[i], bt[i], wt[i], tat[i]);
    }
}

// Round Robin Scheduling
void rrScheduling(int bt[], int n, int tq) {
    int pid[MAX], wt[MAX], tat[MAX], rem_bt[MAX];
    int time = 0;  // Current time

    // Initialize
    for (int i = 0; i < n; i++) {
        pid[i] = i + 1;
        rem_bt[i] = bt[i];   // Remaining burst time
        wt[i] = 0;
    }

    int done;

    // Processing using Round Robin
    do {
        done = 1;

        for (int i = 0; i < n; i++) {
            if (rem_bt[i] > 0) {
                done = 0;

                if (rem_bt[i] > tq) {
                    time += tq;
                    rem_bt[i] -= tq;
                } else {
                    time += rem_bt[i];
                    wt[i] = time - bt[i];
                    rem_bt[i] = 0;
                }
            }
        }
    } while (!done);

    // Turnaround Time
    for (int i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
    }

    // Averages
    int totalWT = 0, totalTAT = 0;
    for (int i = 0; i < n; i++) {
        totalWT += wt[i];
        totalTAT += tat[i];
    }

    // Print steps
    printf("\nRound Robin Scheduling Steps:\n");
    printTable(pid, bt, wt, tat, n);

    printf("\nAverage Waiting Time      = %.2f", (float) totalWT / n);
    printf("\nAverage Turnaround Time   = %.2f\n", (float) totalTAT / n);
}

int main() {
    int n, bt[MAX], tq;
    printf(".....COMPILED BY JIGYASA KOIRALA....\n");
    printf("Enter number of processes: ");
    scanf("%d", &n);

    printf("Enter burst time for each process:\n");
    for (int i = 0; i < n; i++) {
        printf("BT of P%d: ", i + 1);
        scanf("%d", &bt[i]);
    }

    printf("Enter Time Quantum: ");
    scanf("%d", &tq);

    rrScheduling(bt, n, tq);

    return 0;
}
```

Output:



C:\Users\user\Documents\OS\RR.exe

```
.....COMPILED BY JIGYASA KOIRALA....
Enter number of processes: 7
Enter burst time for each process:
BT of P1: 13
BT of P2: 9
BT of P3: 17
BT of P4: 8
BT of P5: 4
BT of P6: 6
BT of P7: 2
Enter Time Quantum: 4

Round Robin Scheduling Steps:

Process BT      WT      TAT
P1      13      41      54
P2      9       40      49
P3      17      42      59
P4      8       34      42
P5      4       16      20
P6      6       38      44
P7      2       24      26

Average Waiting Time      = 33.57
Average Turnaround Time   = 42.00

--------------------------------
Process exited after 25.41 seconds with return value 0
Press any key to continue . . .
```

Lab no:7 **Date: 2082/04/29**

Title: **Wap to print the memory allocation techniques for user input number and size of free blocks and process using Best fit allocation.**

Introduction:

Best fit allocation is a memory management strategy used in operating systems (and sometimes in dynamic memory allocation in programming) where the system searches the list of free memory blocks (holes) and allocates the smallest block that is large enough to satisfy the request.

Allocation

The process of assigning a part of the computer's memory (RAM) to a program or process so it can run.

Free Blocks

Parts of memory that are currently unused and available for allocation. Also called holes memory.

Incoming Process

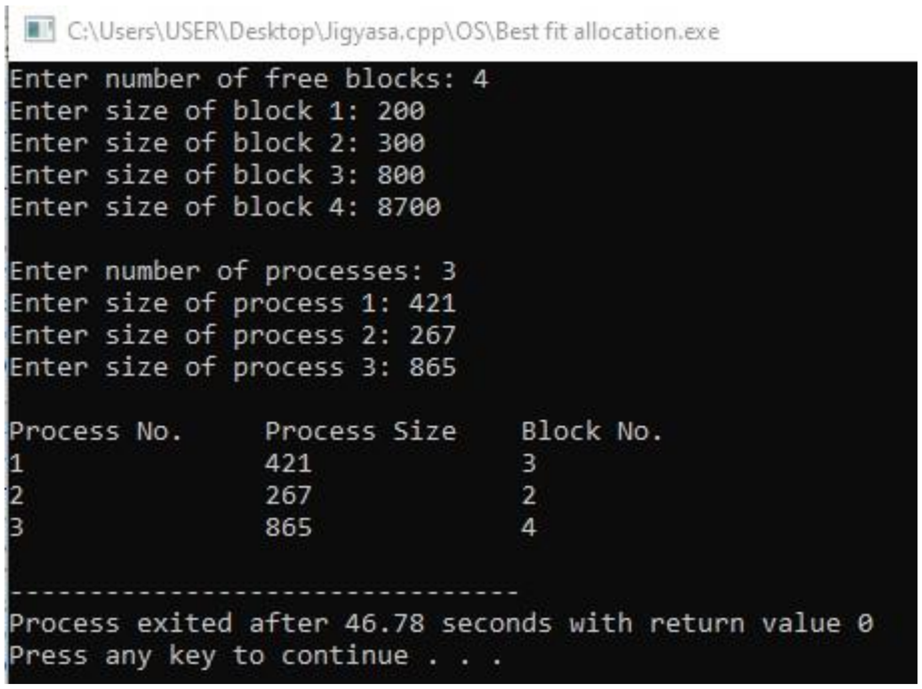
A new process (program) that has just arrived and needs memory to execute. The OS will look for a suitable free block to allocate to this process.

Compiler: Dev C++
Language: C

Source code:

```
#include <stdio.h>

int main() {

  int nb, np, i, j;

  printf("Enter number of free blocks: ");

  scanf("%d", &nb);

  int blockSize[nb], blockAllocated[nb];

  for (i = 0; i < nb; i++) {

    printf("Enter size of block %d: ", i + 1);

    scanf("%d", &blockSize[i]);

    blockAllocated[i] = 0; // initially not allocated

  }

  printf("\nEnter number of processes: ");

  scanf("%d", &np);

  int processSize[np], allocation[np];

  for (i = 0; i < np; i++) {

    printf("Enter size of process %d: ", i + 1);

    scanf("%d", &processSize[i]);

    allocation[i] = -1; // initially not allocated

  }

  // Best Fit Allocation

  for (i = 0; i < np; i++) {
        int bestIndex = -1;

        for (j = 0; j < nb; j++) {

            if (!blockAllocated[j] && blockSize[j] >= processSize[i]) {

                if (bestIndex == -1 || blockSize[j] < blockSize[bestIndex]) {

                    bestIndex = j;

                }

            }

        }

        if (bestIndex != -1) {

            allocation[i] = bestIndex;

            blockAllocated[bestIndex] = 1; // mark block as allocated

        }

    }

    printf("\nProcess No.\tProcess Size\tBlock No.\n");

    for (i = 0; i < np; i++) {

        printf("%d\t\t%d\t\t", i + 1, processSize[i]);

        if (allocation[i] != -1)

            printf("%d\n", allocation[i] + 1);
         else

            printf("Not Allocated\n");

    }    return 0;

}
```

Output:



C:\Users\USER\Desktop\Jigyasa.cpp\OS\Best fit allocation.exe

```
Enter number of free blocks: 4
Enter size of block 1: 200
Enter size of block 2: 300
Enter size of block 3: 800
Enter size of block 4: 8700

Enter number of processes: 3
Enter size of process 1: 421
Enter size of process 2: 267
Enter size of process 3: 865

Process No.     Process Size    Block No.
1               421             3
2               267             2
3               865             4

--------------------------------
Process exited after 46.78 seconds with return value 0
Press any key to continue . . .
```

Lab no:8 **Date: 2082/04/29**

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

Introduction:

Worst Fit Allocation is a memory management strategy used in operating systems to assign free memory blocks to incoming processes.

In this method, the OS searches the list of free memory blocks (holes) and allocates the largest available block that can fit the process.

Allocation

Assigning a memory block to a process according to the Worst Fit rule. In Worst Fit, allocation means finding the largest free block that can hold the incoming process and giving that memory space to it.

Free Blocks

Parts of memory (holes) that are currently empty and can be given to processes. In Worst Fit, before allocation, we check all free blocks and pick the largest one. After allocation, the free block's size is reduced by the size of the process.

Incoming Process

A new process arriving in the system that requests a specific amount of memory. In Worst Fit, we take this process size and search for the largest possible free block to place it.

If no free block is big enough → process is not allocated.

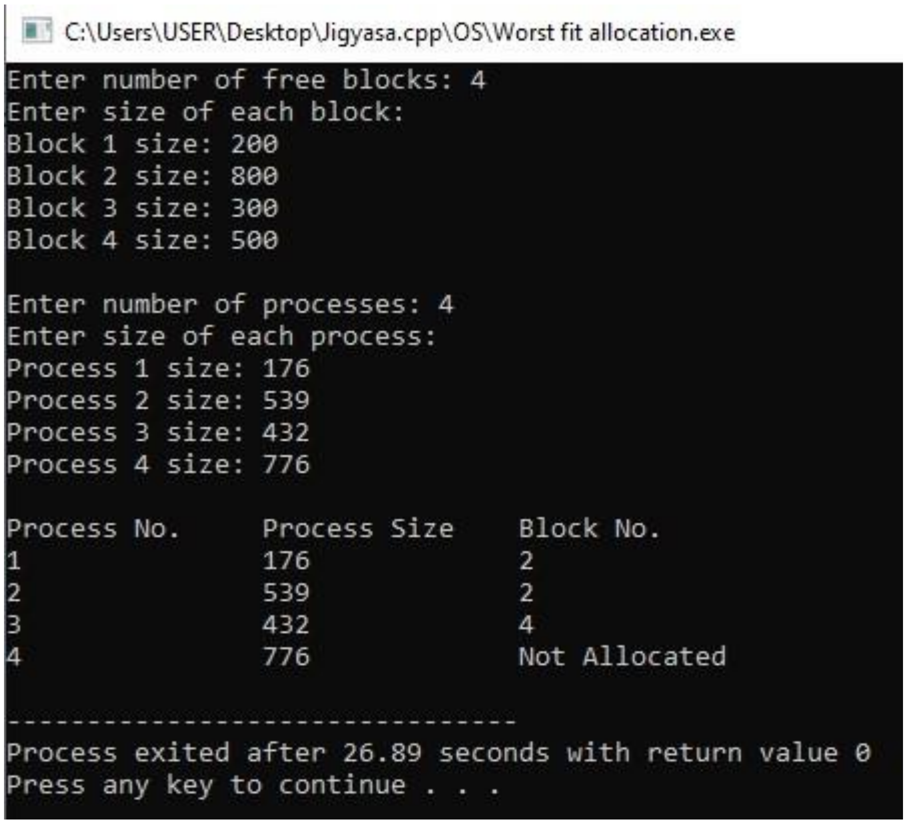
Compiler: Dev C++
Language: C

Source code:

```
#include <stdio.h>

int main() {     int nb, np; // nb = number of blocks, np = number
of processes     int blockSize[20], processSize[20], allocation[20];

   printf("Enter number of free blocks: ");
scanf("%d", &nb);

   printf("Enter size of each block:\n");
for (int i = 0; i < nb; i++) {
printf("Block %d size: ", i + 1);

      scanf("%d", &blockSize[i]);

   }

   printf("\nEnter number of processes: ");
scanf("%d", &np);

   printf("Enter size of each process:\n");
for (int i = 0; i < np; i++) {
printf("Process %d size: ", i + 1);
scanf("%d", &processSize[i]);

      allocation[i] = -1; // -1 means not allocated yet

   }

   // Worst Fit Allocation     for (int i = 0; i < np; i++) {        int
worstIdx = -1;        for (int j = 0; j < nb; j++) {           if (blockSize[j]
>= processSize[i]) {               if (worstIdx == -1 || blockSize[j] >
blockSize[worstIdx]) {                  worstIdx = j;

            }

         }

      }

      if (worstIdx != -1) {
//Allocate block to process
allocation[i] = worstIdx;

         blockSize[worstIdx] -= processSize[i];
```

```
        }

    }


    // Display Allocation Result

    printf("\nProcess No.\tProcess Size\tBlock No.\n");

    for (int i = 0; i < np; i++) {
printf("%d\t\t%d\t\t", i + 1, processSize[i]);

        if (allocation[i] != -1)
printf("%d\n", allocation[i] + 1);
else            printf("Not Allocated\n");

    }

    return 0;

}
```

Output:



C:\Users\USER\Desktop\Jigyasa.cpp\OS\Worst fit allocation.exe

```
Enter number of free blocks: 4
Enter size of each block:
Block 1 size: 200
Block 2 size: 800
Block 3 size: 300
Block 4 size: 500

Enter number of processes: 4
Enter size of each process:
Process 1 size: 176
Process 2 size: 539
Process 3 size: 432
Process 4 size: 776

Process No.     Process Size    Block No.
1               176             2
2               539             2
3               432             4
4               776             Not Allocated

--------------------------------
Process exited after 26.89 seconds with return value 0
Press any key to continue . . .
```

Lab no:9 **Date: 2082/04/29**

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

Introduction:

First Fit Allocation

First Fit is a memory allocation strategy where the first free block (hole) that is big enough to hold the process is allocated to it.

- A process arrives requesting some memory.
- The OS checks the free blocks from the start of the list.
- The first block that is big enough is chosen.
- Remaining space in that block stays as a smaller free block.

Free Blocks

Memory segments (holes) that are empty and available to store a process. In First Fit, the search always starts from the beginning of the free block list.

Incoming Process

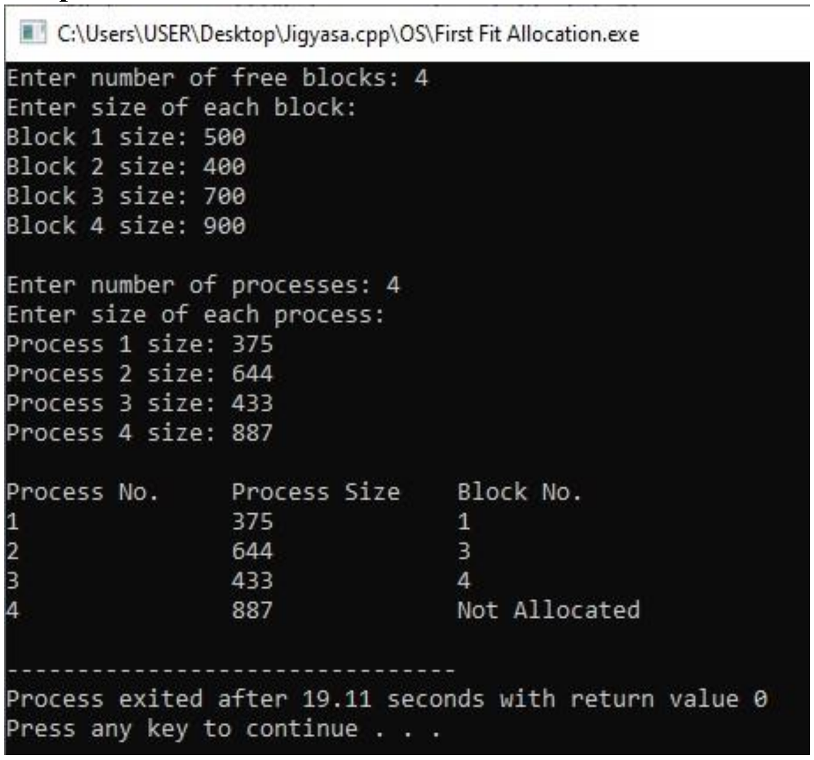
A process (program) that has just arrived and needs memory to execute. It comes with a request for a specific amount of memory. In First Fit, we give it the first free block large enough for it.

Compiler: Dev C++
Language: C

Source code:

```
#include <stdio.h>

int main() {     int nb, np; // nb = number of blocks, np = number
of processes

    int blockSize[20], processSize[20], allocation[20];

    printf("Enter number of free blocks: ");
scanf("%d", &nb);

    printf("Enter size of each block:\n");
for (int i = 0; i < nb; i++) {
printf("Block %d size: ", i + 1);

        scanf("%d", &blockSize[i]);

    }

    printf("\nEnter number of processes: ");
scanf("%d", &np);

    printf("Enter size of each process:\n");     for
(int i = 0; i < np; i++) {         printf("Process %d
size: ", i + 1);         scanf("%d", &processSize[i]);
allocation[i] = -1; // Not allocated initially

    }

    // First Fit Allocation     for (int i = 0; i <
np; i++) {         for (int j = 0; j < nb; j++) {
if (blockSize[j] >= processSize[i]) {
allocation[i] = j;                 blockSize[j] -=
processSize[i];

                break; // move to the next process

            }
        }
    }

    // Display result
    printf("\nProcess No.\tProcess Size\tBlock No.\n");
for (int i = 0; i < np; i++) {         printf("%d\t\t%d\t\t", i
+ 1, processSize[i]);
        if (allocation[i] != -1)
printf("%d\n", allocation[i] + 1);
        else
            printf("Not Allocated\n");
    }

    return 0;
}
```

Output:



C:\Users\USER\Desktop\Jigyasa.cpp\OS\First Fit Allocation.exe

```
Enter number of free blocks: 4
Enter size of each block:
Block 1 size: 500
Block 2 size: 400
Block 3 size: 700
Block 4 size: 900

Enter number of processes: 4
Enter size of each process:
Process 1 size: 375
Process 2 size: 644
Process 3 size: 433
Process 4 size: 887

Process No.     Process Size    Block No.
1               375             1
2               644             3
3               433             4
4               887             Not Allocated

--------------------------------
Process exited after 19.11 seconds with return value 0
Press any key to continue . . .
```

Lab no: 10 **Date: 2082/05/10**

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.

The FCFS (First Come First Serve) disk scheduling algorithm services the disk I/O requests in the exact order they arrive, without reordering them based on distance or priority. It is the simplest disk scheduling method and works similar to the FCFS CPU scheduling technique.

- Start
- Input the number of requests, request queue, current head position, and total cylinders.
- Set total_seek = 0.
- For each request in the order they arrive:
- Calculate seek = |current_head − request|
- Add seek to total_seek
- Update current_head = request
- Display total seek time.
- Stop

Advantages

- Very simple to implement.
- Completely fair; processes are served in arrival order.
- Suitable for small workloads.

Disadvantages

- Causes **high total seek time** if requests are far apart.
- Long waiting time for some requests (convoy effect).
- Not efficient for systems with heavy disk traffic.

Example

Current head = 50
Requests = 82, 170, 43, 140, 24
Order of service:
50 → 82 → 170 → 43 → 140 → 24

Seek distances

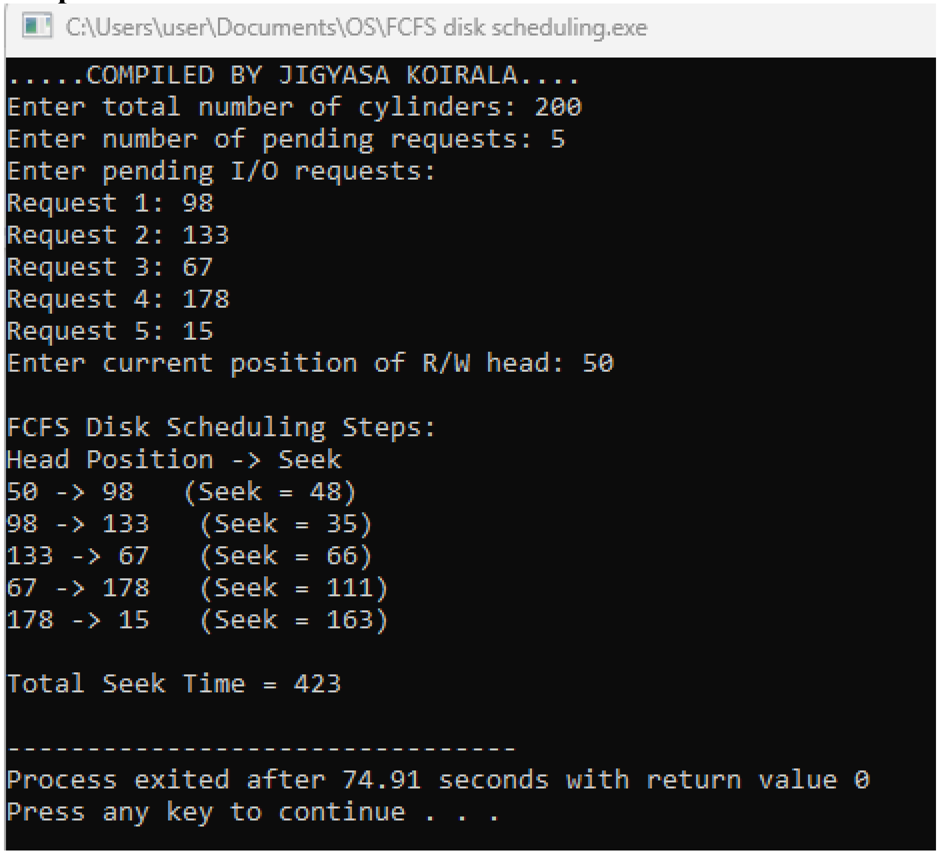
|50−82| + |82−170| + |170−43| + |43−140| + |140−24|
= 32 + 88 + 127 + 97 + 116
= **460 cylinders moved**

Compiler: DEV C++
Language: C

Source Code:

```
#include <stdio.h>
#define MAX 100

// FCFS Disk Scheduling Algorithm
void fcfsDiskScheduling(int req[], int n, int head) {
    int seekTime = 0;

    printf("\nFCFS Disk Scheduling Steps:\n");
    printf("Head Position -> Seek\n");

    for (int i = 0; i < n; i++) {
        int diff = req[i] - head;
        if (diff < 0) diff = -diff;

        printf("%d -> %d   (Seek = %d)\n", head, req[i], diff);

        seekTime += diff;
        head = req[i];
    }


    printf("\nTotal Seek Time = %d\n", seekTime);

}

int main() {

    int n, totalCylinders, head;

    int req[MAX];

    printf(".....COMPILED BY JIGYASA KOIRALA....\n");

    printf("Enter total number of cylinders: ");

    scanf("%d", &totalCylinders);

    printf("Enter number of pending requests: ");

    scanf("%d", &n);

    printf("Enter pending I/O requests:\n");

    for (int i = 0; i < n; i++) {

        printf("Request %d: ", i + 1);
        scanf("%d", &req[i]);
    }
    printf("Enter current position of R/W head: ");
    scanf("%d", &head);
    fcfsDiskScheduling(req, n, head);
    return 0;
}
```

Output:



```
C:\Users\user\Documents\OS\FCFS disk scheduling.exe
.....COMPILED BY JIGYASA KOIRALA....
Enter total number of cylinders: 200
Enter number of pending requests: 5
Enter pending I/O requests:
Request 1: 98
Request 2: 133
Request 3: 67
Request 4: 178
Request 5: 15
Enter current position of R/W head: 50

FCFS Disk Scheduling Steps:
Head Position -> Seek
50 -> 98   (Seek = 48)
98 -> 133   (Seek = 35)
133 -> 67   (Seek = 66)
67 -> 178   (Seek = 111)
178 -> 15   (Seek = 163)

Total Seek Time = 423

--------------------------------
Process exited after 74.91 seconds with return value 0
Press any key to continue . . .
```

Lab no: 11 **Date: 2082/05/10**

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.

SCAN also known as the Elevator Algorithm, moves the disk head in one fixed direction (left or right) servicing all requests, and when it reaches the end, it reverses direction and continues servicing remaining request, just like an elevator moving up and down.

How SCAN Works

1. Choose a direction (left or right).
2. Move the head in that direction.
3. Service every request on the way.
4. When the boundary (0 or max cylinder) is reached, reverse the direction.
5. Continue servicing remaining requests.

Advantages

- More efficient than FCFS because the head moves in an ordered path.
- Reduces variance in response time.
- Prevents starvation.

Disadvantages

- Longer wait time for requests just behind the head's direction.
- Not optimal, but better than FCFS.

Example

Head = 50
Requests = 55, 58, 39, 18, 90
Direction = Right
Disk size = 0–199

Order of service:

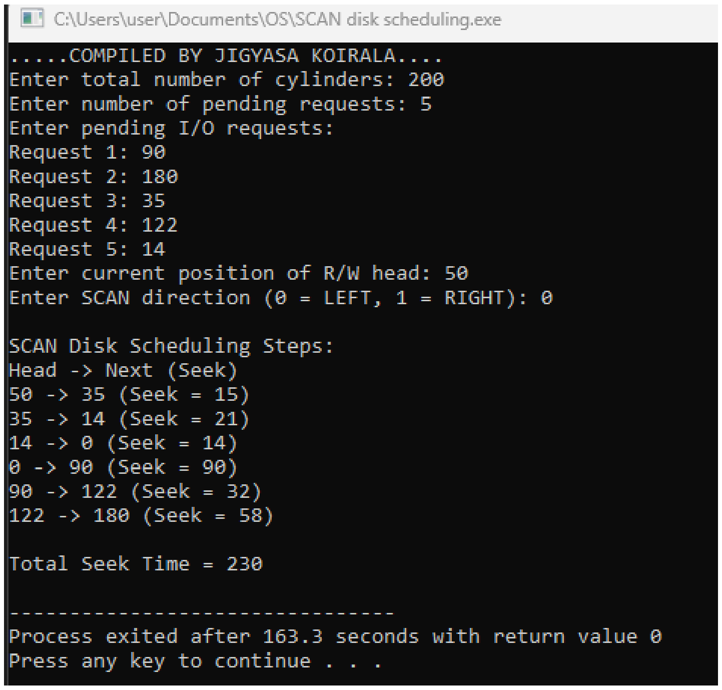
55 → 58 → 90 → 199 → 39 → 18 → 0
Total seek = 348 tracks

Compiler: DEV C++
Language: C

Source Code:

```
void scanDiskScheduling(int req[], int n, int head, int cylinders, int direction) {
    int seekTime = 0;
    int sorted[MAX];
    int i, j, temp;

    // Copy array
    for (i = 0; i < n; i++)
        sorted[i] = req[i];

    // Sort requests
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (sorted[j] > sorted[j + 1]) {
                temp = sorted[j];
                sorted[j] = sorted[j + 1];
                sorted[j + 1] = temp;
            }
        }
    }
    printf("\nSCAN Disk Scheduling Steps:\n");
    printf("Head -> Next (Seek)\n");

    // Find index where head fits in sorted list
    int idx = 0;
    for (i = 0; i < n; i++)
        if (sorted[i] > head) {
            idx = i;
            break;
        }
    int prevHead = head;

    // SCAN LEFT direction
    if (direction == 0) {
        // Move left first
        for (i = idx - 1; i >= 0; i--) {
            int diff = prevHead - sorted[i];
            printf("%d -> %d (Seek = %d)\n", prevHead, sorted[i], diff);
            seekTime += diff;
            prevHead = sorted[i];
        }

        // Move to cylinder 0
        printf("%d -> 0 (Seek = %d)\n", prevHead, prevHead);
        seekTime += prevHead;
        prevHead = 0;

        // Then move right
        for (i = idx; i < n; i++) {
            int diff = sorted[i] - prevHead;
            printf("%d -> %d (Seek = %d)\n", prevHead, sorted[i], diff);
            seekTime += diff;
            prevHead = sorted[i];
        }
    }

    // SCAN RIGHT direction
    else {
        // Move right first
        for (i = idx; i < n; i++) {
            int diff = sorted[i] - prevHead;
            printf("%d -> %d (Seek = %d)\n", prevHead, sorted[i], diff);
            seekTime += diff;
            prevHead = sorted[i];
        }

        // Move to last cylinder
        printf("%d -> %d (Seek = %d)\n", prevHead, cylinders - 1,
               (cylinders - 1) - prevHead);
        seekTime += (cylinders - 1) - prevHead;
        prevHead = cylinders - 1;

        // Then move left
        for (i = idx - 1; i >= 0; i--) {
            int diff = prevHead - sorted[i];
            printf("%d -> %d (Seek = %d)\n", prevHead, sorted[i], diff);
            seekTime += diff;
            prevHead = sorted[i];
        }
    }
    printf("\nTotal Seek Time = %d\n", seekTime);
}

int main() {
    int n, cylinders, head, direction;
    int req[MAX];
    printf(".....COMPILED BY JIGYASA KOIRALA....\n");
    printf("Enter total number of cylinders: ");
    scanf("%d", &cylinders);

    printf("Enter number of pending requests: ");
    scanf("%d", &n);

    printf("Enter pending I/O requests:\n");
    for (int i = 0; i < n; i++) {
        printf("Request %d: ", i + 1);
        scanf("%d", &req[i]);
    }

    printf("Enter current position of R/W head: ");
    scanf("%d", &head);

    printf("Enter SCAN direction (0 = LEFT, 1 = RIGHT): ");
    scanf("%d", &direction);

    scanDiskScheduling(req, n, head, cylinders, direction);

    return 0;
}
```

Output:



```
C:\Users\user\Documents\OS\SCAN disk scheduling.exe
.....COMPILED BY JIGYASA KOIRALA....
Enter total number of cylinders: 200
Enter number of pending requests: 5
Enter pending I/O requests:
Request 1: 90
Request 2: 180
Request 3: 35
Request 4: 122
Request 5: 14
Enter current position of R/W head: 50
Enter SCAN direction (0 = LEFT, 1 = RIGHT): 0

SCAN Disk Scheduling Steps:
Head -> Next (Seek)
50 -> 35 (Seek = 15)
35 -> 14 (Seek = 21)
14 -> 0 (Seek = 14)
0 -> 90 (Seek = 90)
90 -> 122 (Seek = 32)
122 -> 180 (Seek = 58)

Total Seek Time = 230

--------------------------------
Process exited after 163.3 seconds with return value 0
Press any key to continue . . .
```

Lab no: 12 **Date: 2082/05/10**

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.

LOOK is an improved version of **SCAN.**
Instead of going all the way to the last cylinder (0 or max), the head only goes as far as the last requested track in each direction, then reverses.
So it “looks” ahead for the last request before turning back.

- Choose a direction (left or right).
- Move the head in that direction.
- Service every request until the last request in that direction.
- Do not go to the physical end of the disk.
- Reverse direction when no more requests exist ahead.
- Continue servicing remaining requests.

Advantages
Faster than SCAN because it avoids unnecessary movement to 0 or max cylinder.
Reduces total seek time.
Prevents starvation & maintains fairness.
Disadvantages
More complex than FCFS and SSTF.
Still not fully optimal like C-SCAN / C-LOOK.

Example
Head = 50
Requests = 55, 58, 39, 18, 90
Direction = Right
Disk size = 0–199 (LOOK doesn't need full boundary)

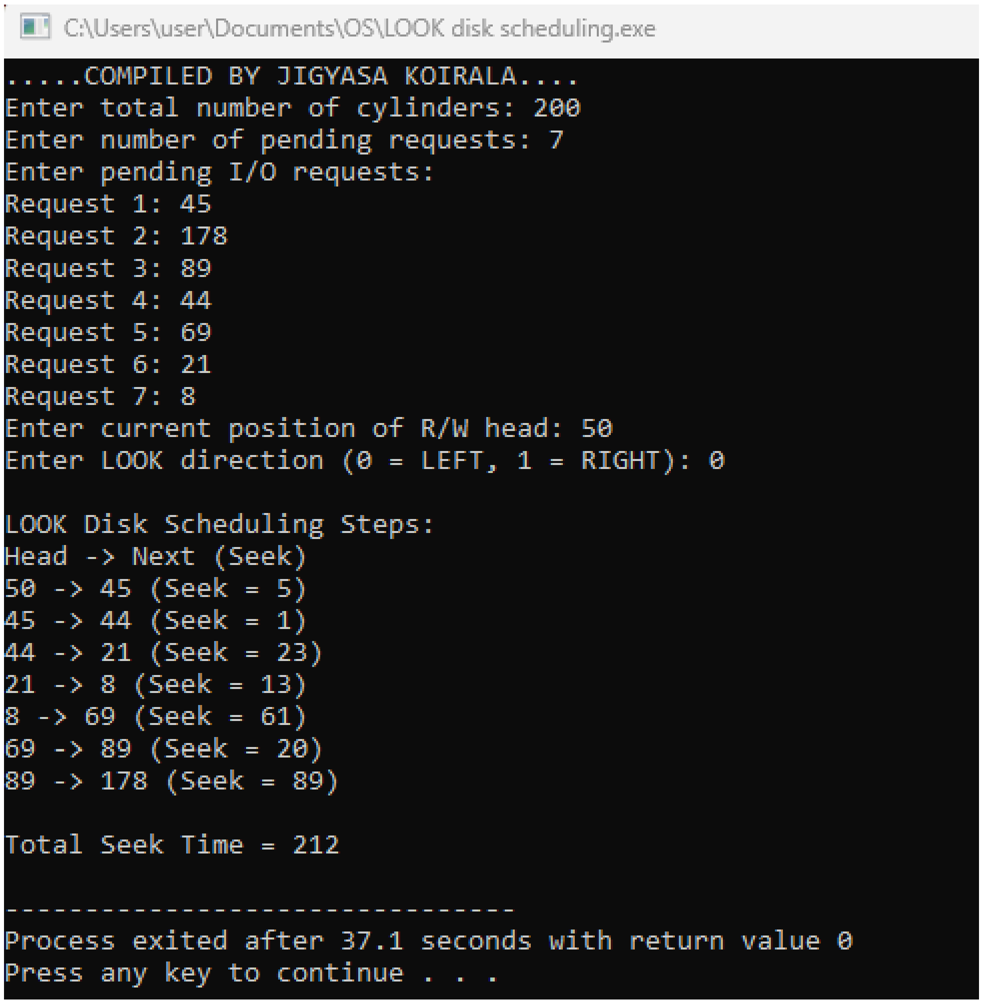
Step-by-Step Order
We move right, servicing requests in ascending order:
Right side requests from 50 ⇒ 55, 58, 90
(90 is the last request on the right, so stop there and reverse.)
Then move left to remaining requests ⇒ 39, 18

Compiler: DEV C++
Language: C

Source Code:

```
void lookDiskScheduling(int req[], int n, int head, int direction) {
    int sorted[MAX];
    int i, j, temp;
    int seekTime = 0, prevHead = head;

    // Copy request array
    for (i = 0; i < n; i++)
        sorted[i] = req[i];

    // Sort requests
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (sorted[j] > sorted[j + 1]) {
                temp = sorted[j];
                sorted[j] = sorted[j + 1];
                sorted[j + 1] = temp;
            }
        }
    }

    printf("\nLOOK Disk Scheduling Steps:\n");
    printf("Head -> Next (Seek)\n");

    // Find index where head would fit in sorted list
    int idx = 0;
    for (i = 0; i < n; i++) {
        if (sorted[i] > head) {
            idx = i;
            break;
        }
    }

    // LOOK LEFT direction first
    if (direction == 0) {
        // Move LEFT to the smallest request
        for (i = idx - 1; i >= 0; i--) {
            int diff = prevHead - sorted[i];
            printf("%d -> %d (Seek = %d)\n", prevHead, sorted[i], diff);
            seekTime += diff;
            prevHead = sorted[i];
        }
        // Then move RIGHT to the remaining requests
        for (i = idx; i < n; i++) {
            int diff = sorted[i] - prevHead;
            printf("%d -> %d (Seek = %d)\n", prevHead, sorted[i], diff);
            seekTime += diff;
            prevHead = sorted[i];
        }
    }

    // LOOK RIGHT direction first
    else {
        // Move RIGHT to the largest request
        for (i = idx; i < n; i++) {
            int diff = sorted[i] - prevHead;
            printf("%d -> %d (Seek = %d)\n", prevHead, sorted[i], diff);
            seekTime += diff;
            prevHead = sorted[i];
        }

        // Then move LEFT to the smaller requests
        for (i = idx - 1; i >= 0; i--) {
            int diff = prevHead - sorted[i];
            printf("%d -> %d (Seek = %d)\n", prevHead, sorted[i], diff);
            seekTime += diff;
            prevHead = sorted[i];
        }
    }

    printf("\nTotal Seek Time = %d\n", seekTime);
}

int main() {
    int n, cylinders, head, direction;
    int req[MAX];

    printf(".....COMPILED BY JIGYASA KOIRALA....\n");
    printf("Enter total number of cylinders: ");
    scanf("%d", &cylinders);

    printf("Enter number of pending requests: ");
    scanf("%d", &n);

    printf("Enter pending I/O requests:\n");
    for (int i = 0; i < n; i++) {
        printf("Request %d: ", i + 1);
        scanf("%d", &req[i]);
    }

    printf("Enter current position of R/W head: ");
    scanf("%d", &head);

    printf("Enter LOOK direction (0 = LEFT, 1 = RIGHT): ");
    scanf("%d", &direction);

    lookDiskScheduling(req, n, head, direction);

    return 0;
}
```

Output:



```
C:\Users\user\Documents\OS\LOOK disk scheduling.exe
.....COMPILED BY JIGYASA KOIRALA....
Enter total number of cylinders: 200
Enter number of pending requests: 7
Enter pending I/O requests:
Request 1: 45
Request 2: 178
Request 3: 89
Request 4: 44
Request 5: 69
Request 6: 21
Request 7: 8
Enter current position of R/W head: 50
Enter LOOK direction (0 = LEFT, 1 = RIGHT): 0

LOOK Disk Scheduling Steps:
Head -> Next (Seek)
50 -> 45 (Seek = 5)
45 -> 44 (Seek = 1)
44 -> 21 (Seek = 23)
21 -> 8 (Seek = 13)
8 -> 69 (Seek = 61)
69 -> 89 (Seek = 20)
89 -> 178 (Seek = 89)

Total Seek Time = 212

--------------------------------
Process exited after 37.1 seconds with return value 0
Press any key to continue . . .
```

Lab no: 13 **Date: 2082/05/10**

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.

Banker's Algorithm
It is a resource allocation and deadlock avoidance algorithm used in operating systems. It ensures that a system remains in a safe state by carefully allocating resources to processes while avoiding unsafe states that could lead to deadlocks.

- The Banker's Algorithm is a smart way for computer systems to manage how programs use resources, like memory or CPU time.
- It helps prevent situations where programs get stuck and can not finish their tasks. This condition is known as deadlock.
- By keeping track of what resources each program needs and what's available, the banker algorithm makes sure that programs only get what they need in a safe order.

Safe State: There exists at least one sequence of processes such that each process can obtain the needed resources, complete its execution, release its resources, and thus allow other processes to eventually complete without entering a deadlock.

Unsafe State: Even though the system can still allocate resources to some processes, there is no guarantee that all processes can finish without potentially causing a deadlock.

Compiler: DEV C++
Language: C

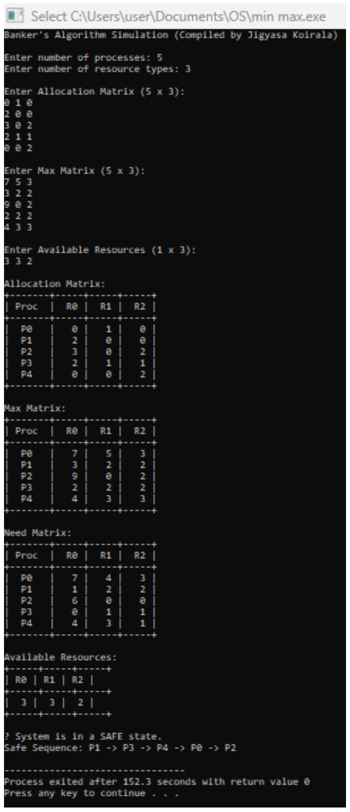
Source Code:

```
#include <stdio.h>
#include <stdbool.h>
#define MAX_P 10
#define MAX_R 10                                           // Function to print a matrix
void printMatrix(int matrix[MAX_P][MAX_R], int rows, int cols, const char *name) {
   printf("\n%s Matrix:\n", name);
   printf("+-------+");
   for (int j = 0; j < cols; j++) printf("-----+");
   printf("\n| Proc  |");
   for (int j = 0; j < cols; j++) printf("  R%d |", j);
   printf("\n+-------+");
   for (int j = 0; j < cols; j++) printf("-----+");
   printf("\n");
   for (int i = 0; i < rows; i++) {
      printf("|  P%-3d |", i);
      for (int j = 0; j < cols; j++) {
         printf("  %2d |", matrix[i][j]);
      }
      printf("\n");
   }
   printf("+-------+");
   for (int j = 0; j < cols; j++) printf("-----+");
   printf("\n");
}                                          // Function to print available resources
void printAvailable(int available[MAX_R], int r) {
   printf("\nAvailable Resources:\n");
   printf("+");
   for (int j = 0; j < r; j++) printf("-----+");
   printf("\n|");
   for (int j = 0; j < r; j++) printf(" R%d |", j);
   printf("\n+");
   for (int j = 0; j < r; j++) printf("-----+");
   printf("\n|");
   for (int j = 0; j < r; j++) printf(" %2d |", available[j]);
   printf("\n+");
   for (int j = 0; j < r; j++) printf("-----+");
   printf("\n");
}
int main() {
   int p, r;
   int allocation[MAX_P][MAX_R], max[MAX_P][MAX_R], need[MAX_P][MAX_R];
   int available[MAX_R];
   bool finish[MAX_P] = {false};
```

```
int safeSequence[MAX_P];
int count = 0;
printf("Banker's Algorithm Simulation (Compiled by Jigyasa Koirala)\n\n");
                                                    // Input
printf("Enter number of processes: ");
scanf("%d", &p);
printf("Enter number of resource types: ");
scanf("%d", &r);
printf("\nEnter Allocation Matrix (%d x %d):\n", p, r);
for (int i = 0; i < p; i++)
   for (int j = 0; j < r; j++)
      scanf("%d", &allocation[i][j]);
printf("\nEnter Max Matrix (%d x %d):\n", p, r);
for (int i = 0; i < p; i++)
   for (int j = 0; j < r; j++)
      scanf("%d", &max[i][j]);
printf("\nEnter Available Resources (1 x %d):\n", r);
for (int i = 0; i < r; i++)
   scanf("%d", &available[i]);
                                                 // Calculate Need Matrix
for (int i = 0; i < p; i++)
   for (int j = 0; j < r; j++)
      need[i][j] = max[i][j] - allocation[i][j];
printMatrix(allocation, p, r, "Allocation");
printMatrix(max, p, r, "Max");
printMatrix(need, p, r, "Need");
printAvailable(available, r);
                                                       // Banker's Algorithm
while (count < p) {
   bool found = false;
   for (int i = 0; i < p; i++) {
      if (!finish[i]) {
         bool possible = true;
         for (int j = 0; j < r; j++) {
            if (need[i][j] > available[j]) {
               possible = false;
               break;
            }
         }
         if (possible) {
                                                         // Process can finish
            for (int j = 0; j < r; j++) {
               available[j] += allocation[i][j];
            }
```

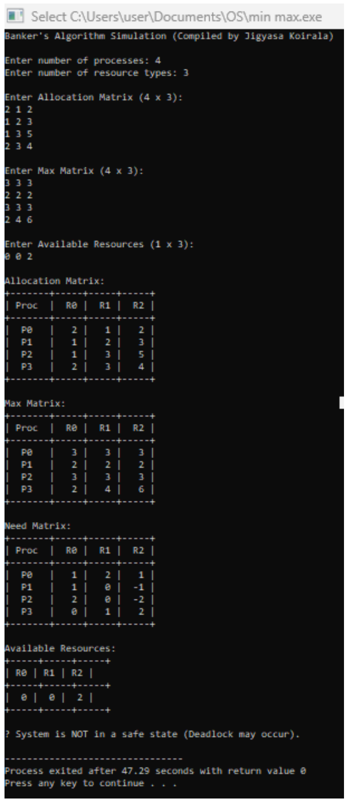
```
                    safeSequence[count++] = i;
                    finish[i] = true;
                    found = true;
                }
            }
        }
        if (!found) break;                        // No process can proceed
    }
    if (count == p) {
printf("\n? System is in a SAFE state.\n");
   printf("Safe Sequence: ");
        for (int i = 0; i < p; i++) {
            printf("P%d", safeSequence[i]);
            if (i != p - 1) printf(" -> ");
        }
        printf("\n");
    } else {
        printf("\n? System is NOT in a safe state (Deadlock may occur).\n");
    }
    return 0;
}
```

Output(With Safe Sequence):



```
Select C:\Users\user\Documents\OS\min max.exe
Banker's Algorithm Simulation (Compiled by Jigyasa Koirala)

Enter number of processes: 5
Enter number of resource types: 3

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

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

Enter Available Resources (1 x 3):
3 3 2

Allocation Matrix:
+-------+-----+-----+-----+
| Proc  |  R0 |  R1 |  R2 |
+-------+-----+-----+-----+
|  P0   |   0 |   1 |   0 |
|  P1   |   2 |   0 |   0 |
|  P2   |   3 |   0 |   2 |
|  P3   |   2 |   1 |   1 |
|  P4   |   0 |   0 |   2 |
+-------+-----+-----+-----+

Max Matrix:
+-------+-----+-----+-----+
| Proc  |  R0 |  R1 |  R2 |
+-------+-----+-----+-----+
|  P0   |   7 |   5 |   3 |
|  P1   |   3 |   2 |   2 |
|  P2   |   9 |   0 |   2 |
|  P3   |   2 |   2 |   2 |
|  P4   |   4 |   3 |   3 |
+-------+-----+-----+-----+

Need Matrix:
+-------+-----+-----+-----+
| Proc  |  R0 |  R1 |  R2 |
+-------+-----+-----+-----+
|  P0   |   7 |   4 |   3 |
|  P1   |   1 |   2 |   2 |
|  P2   |   6 |   0 |   0 |
|  P3   |   0 |   1 |   1 |
|  P4   |   4 |   3 |   1 |
+-------+-----+-----+-----+

Available Resources:
+-----+-----+-----+
| R0 | R1 | R2 |
+-----+-----+-----+
|  3 |  3 |  2 |
+-----+-----+-----+

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

--------------------------------
Process exited after 152.3 seconds with return value 0
Press any key to continue . . .
```

Output(Without Safe Sequence):



```
Select C:\Users\user\Documents\OS\min max.exe
Banker's Algorithm Simulation (Compiled by Jigyasa Koirala)

Enter number of processes: 4
Enter number of resource types: 3

Enter Allocation Matrix (4 x 3):
2 1 2
1 2 3
1 3 5
2 3 4

Enter Max Matrix (4 x 3):
3 3 3
2 2 2
3 3 3
2 4 6

Enter Available Resources (1 x 3):
0 0 2

Allocation Matrix:
+-------+-----+-----+-----+
| Proc  |  R0 |  R1 |  R2 |
+-------+-----+-----+-----+
|  P0   |   2 |   1 |   2 |
|  P1   |   1 |   2 |   3 |
|  P2   |   1 |   3 |   5 |
|  P3   |   2 |   3 |   4 |
+-------+-----+-----+-----+

Max Matrix:
+-------+-----+-----+-----+
| Proc  |  R0 |  R1 |  R2 |
+-------+-----+-----+-----+
|  P0   |   3 |   3 |   3 |
|  P1   |   2 |   2 |   2 |
|  P2   |   3 |   3 |   3 |
|  P3   |   2 |   4 |   6 |
+-------+-----+-----+-----+

Need Matrix:
+-------+-----+-----+-----+
| Proc  |  R0 |  R1 |  R2 |
+-------+-----+-----+-----+
|  P0   |   1 |   2 |   1 |
|  P1   |   1 |   0 |  -1 |
|  P2   |   2 |   0 |  -2 |
|  P3   |   0 |   1 |   2 |
+-------+-----+-----+-----+

Available Resources:
+-----+-----+-----+
| R0 | R1 | R2 |
+-----+-----+-----+
|  0 |  0 |  2 |
+-----+-----+-----+

? System is NOT in a safe state (Deadlock may occur).

--------------------------------
Process exited after 47.29 seconds with return value 0
Press any key to continue . . .
```

Lab no: 14 **Date: 2082/05/22**

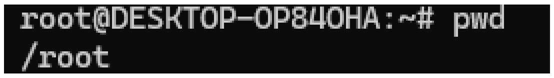
Title: Prepare a lab report for basic Linux command.

Introduction

Linux commands are instructions given to the Linux operating system to perform various tasks. They are executed in a command-line interface (CLI), also known as a terminal or shell. These commands allow users to interact with the system, manage files and directories, control processes, configure settings, and more.

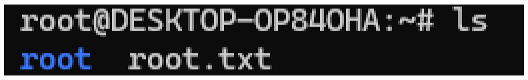
Basics Linux Commands are:

1) pwd → Show current working directory



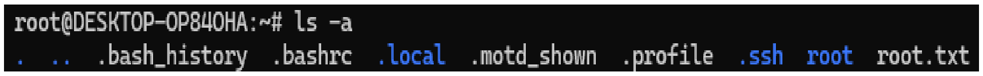
```
root@DESKTOP-OP840HA:~# pwd
/root
```

2) ls → List files and directories



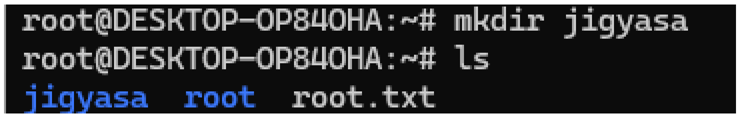
```
root@DESKTOP-OP840HA:~# ls
root  root.txt
```

- ls -a → Show hidden files



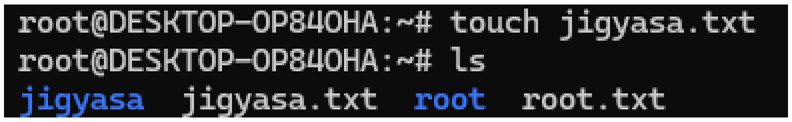
```
root@DESKTOP-OP840HA:~# ls -a
.  ..  .bash_history  .bashrc  .local  .motd_shown  .profile  .ssh  root  root.txt
```

3) mkdir <dir> → Create new directory



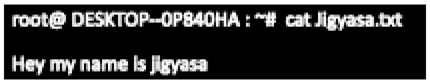
```
root@DESKTOP-OP840HA:~# mkdir jigyasa
root@DESKTOP-OP840HA:~# ls
jigyasa  root  root.txt
```

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



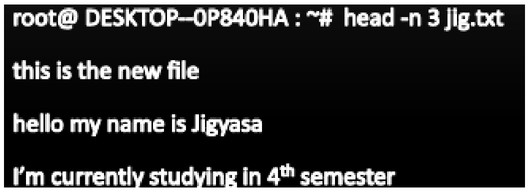
```
root@DESKTOP-OP840HA:~# touch jigyasa.txt
root@DESKTOP-OP840HA:~# ls
jigyasa  jigyasa.txt  root  root.txt
```

5) cat <file> → Display file contents



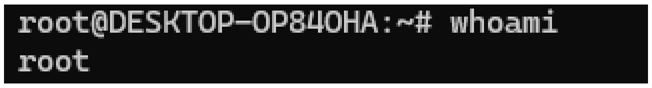
```
root@ DESKTOP--OP840HA : ~# cat Jigyasa.txt

Hey my name is Jigyasa
```

6) head <file> → Show first 10 lines



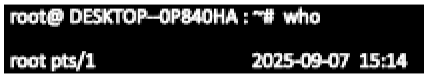
```
root@ DESKTOP--OP840HA : ~#  head -n 3 jig.txt

this is the new file

hello my name is Jigyasa

I'm currently studying in 4th semester
```

7) whoami → Show current user



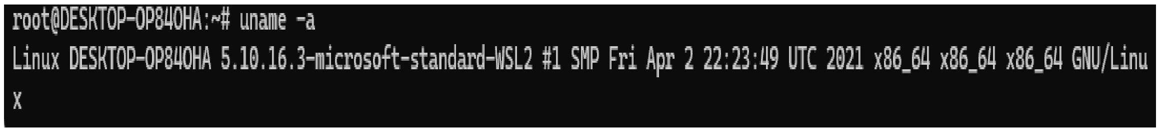
```
root@DESKTOP-OP840HA:~# whoami
root
```

8) who → Show logged-in users



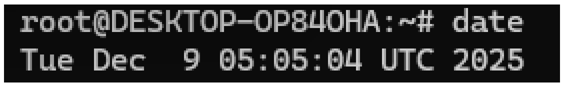
```
root@ DESKTOP--OP840HA : ~#  who

root pts/1                2025-09-07  15:14
```

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



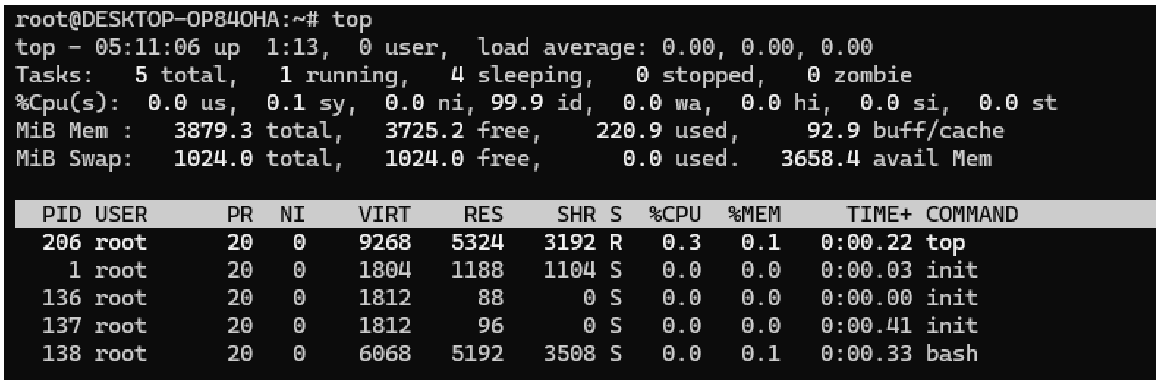
```
root@DESKTOP-OP840HA:~# uname -a
Linux DESKTOP-OP840HA 5.10.16.3-microsoft-standard-WSL2 #1 SMP Fri Apr 2 22:23:49 UTC 2021 x86_64 x86_64 x86_64 GNU/Linu
x
```

10) date → Show current date and time



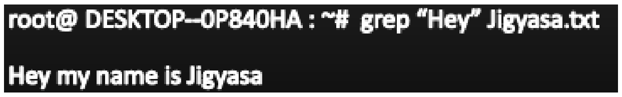
```
root@DESKTOP-OP840HA:~# date
Tue Dec  9 05:05:04 UTC 2025
```

11) top → Live view of processes



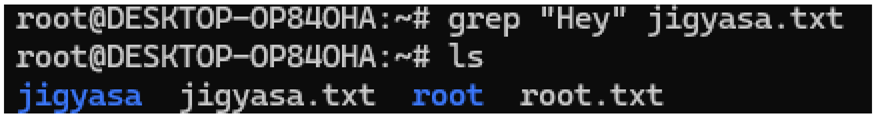
```
root@DESKTOP-OP840HA:~# top
top - 05:11:06 up  1:13,  0 user,  load average: 0.00, 0.00, 0.00
Tasks:   5 total,   1 running,   4 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.1 sy,  0.0 ni, 99.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3879.3 total,   3725.2 free,    220.9 used,     92.9 buff/cache
MiB Swap:   1024.0 total,   1024.0 free,      0.0 used.   3658.4 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  206 root      20   0    9268   5324   3192 R   0.3   0.1   0:00.22 top
    1 root      20   0    1804   1188   1104 S   0.0   0.0   0:00.03 init
  136 root      20   0    1812     88      0 S   0.0   0.0   0:00.00 init
  137 root      20   0    1812     96      0 S   0.0   0.0   0:00.41 init
  138 root      20   0    6068   5192   3508 S   0.0   0.1   0:00.33 bash
```

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

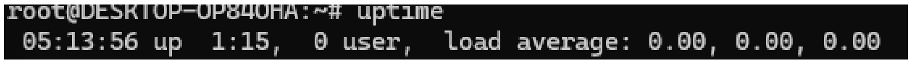


```
root@ DESKTOP--OP840HA : ~#  grep "Hey" Jigyasa.txt

Hey my name is Jigyasa
```

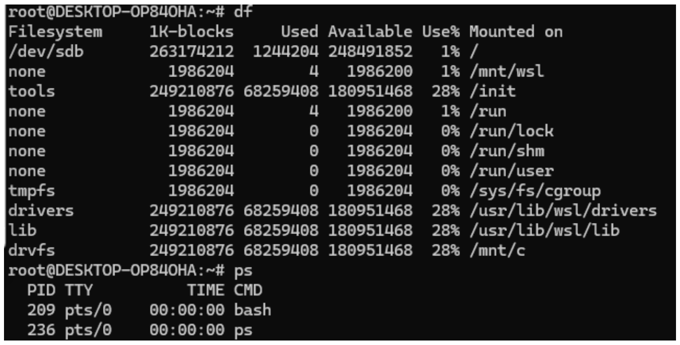
13) wc <file> → Count words, lines, characters


```
root@DESKTOP-OP84OHA:~# grep "Hey" jigyasa.txt
root@DESKTOP-OP84OHA:~# ls
jigyasa  jigyasa.txt  root  root.txt
```

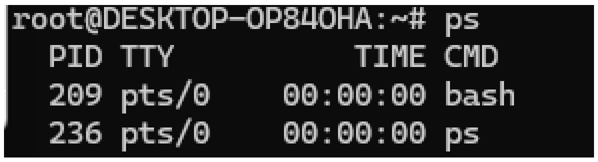
14) uptime → Show system uptime


```
root@DESKTOP-OP84OHA:~# uptime
 05:13:56 up  1:15,  0 user,  load average: 0.00, 0.00, 0.00
```

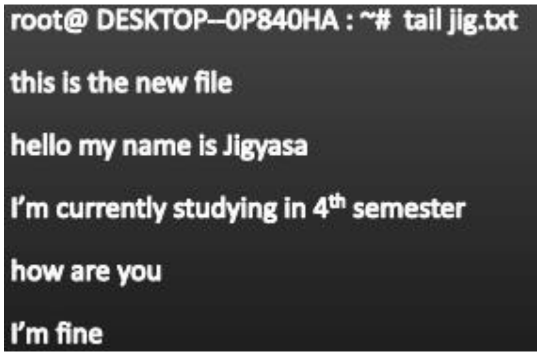
15) df → Checks your Linux system's disk usage


```
root@DESKTOP-OP84OHA:~# df
Filesystem     1K-blocks     Used Available Use% Mounted on
/dev/sdb       263174212  1244204 248491852   1% /
none             1986204        4   1986200   1% /mnt/wsl
tools          249210876 68259408 180951468  28% /init
none             1986204        4   1986200   1% /run
none             1986204        0   1986204   0% /run/lock
none             1986204        0   1986204   0% /run/shm
none             1986204        0   1986204   0% /run/user
tmpfs            1986204        0   1986204   0% /sys/fs/cgroup
drivers        249210876 68259408 180951468  28% /usr/lib/wsl/drivers
lib            249210876 68259408 180951468  28% /usr/lib/wsl/lib
drvfs          249210876 68259408 180951468  28% /mnt/c
root@DESKTOP-OP84OHA:~# ps
  PID TTY          TIME CMD
  209 pts/0    00:00:00 bash
  236 pts/0    00:00:00 ps
```

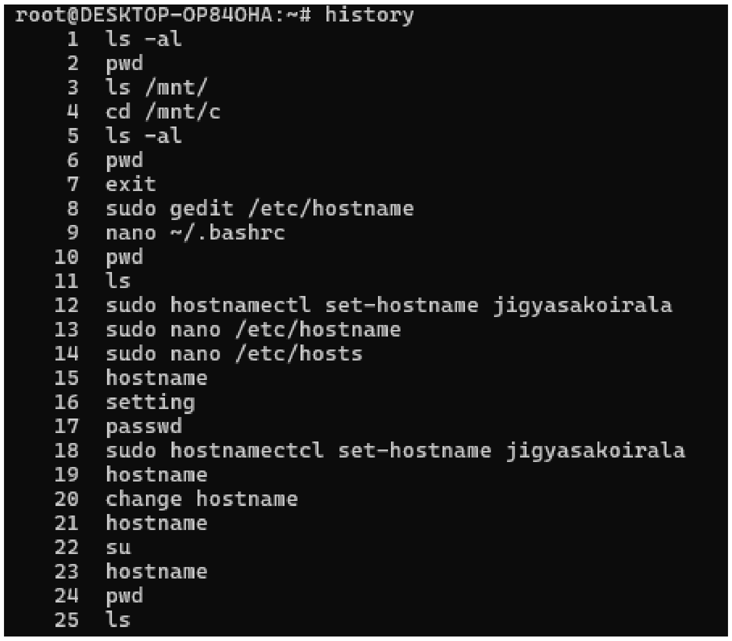
16) ps → Summarizes the status of all running processes in your Linux system at a specific time.


```
root@DESKTOP-OP84OHA:~# ps
  PID TTY          TIME CMD
  209 pts/0    00:00:00 bash
  236 pts/0    00:00:00 ps
```

17) tail → It view the last line.

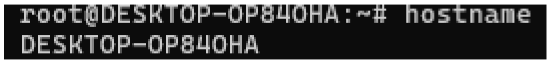

```
root@ DESKTOP–OP840HA : ~#  tail jig.txt
this is the new file
hello my name is Jigyasa
I'm currently studying in 4th semester
how are you
I'm fine
```

18) history → Check previously run utilities.



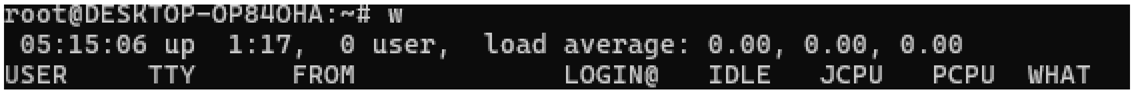
```
root@DESKTOP-OP84OHA:~# history
    1  ls -al
    2  pwd
    3  ls /mnt/
    4  cd /mnt/c
    5  ls -al
    6  pwd
    7  exit
    8  sudo gedit /etc/hostname
    9  nano ~/.bashrc
   10  pwd
   11  ls
   12  sudo hostnamectl set-hostname jigyasakoirala
   13  sudo nano /etc/hostname
   14  sudo nano /etc/hosts
   15  hostname
   16  setting
   17  passwd
   18  sudo hostnamectcl set-hostname jigyasakoirala
   19  hostname
   20  change hostname
   21  hostname
   22  su
   23  hostname
   24  pwd
   25  ls
```

19) hostname → Show hostname



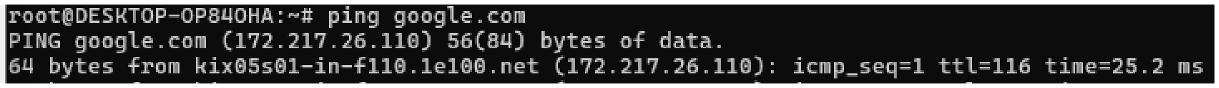
```
root@DESKTOP-OP84OHA:~# hostname
DESKTOP-OP84OHA
```

20) w → Detailed user sessions



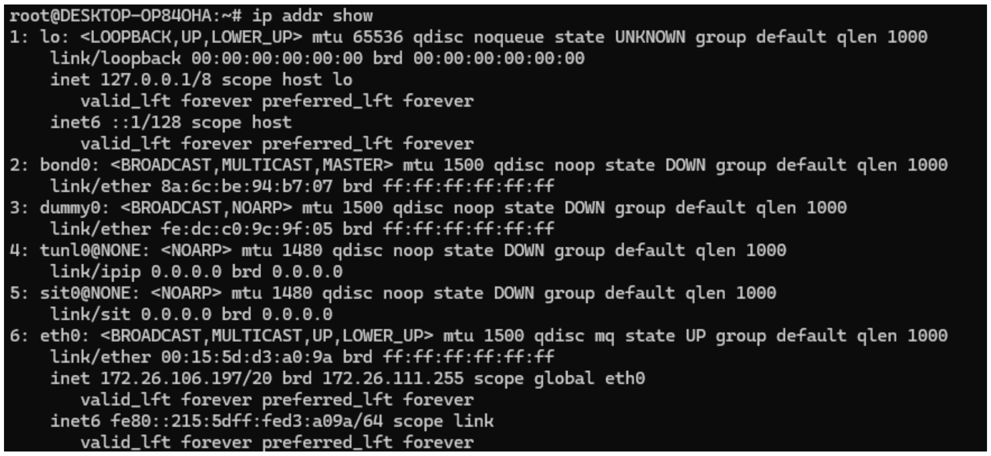
```
root@DESKTOP-OP84OHA:~# w
 05:15:06 up  1:17,  0 user,  load average: 0.00, 0.00, 0.00
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU  WHAT
```

21) ping → Check connectivity



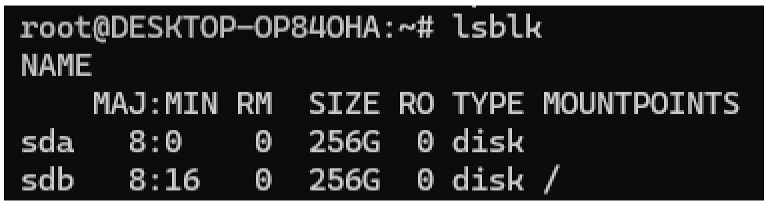
```
root@DESKTOP-OP84OHA:~# ping google.com
PING google.com (172.217.26.110) 56(84) bytes of data.
64 bytes from kix05s01-in-f110.1e100.net (172.217.26.110): icmp_seq=1 ttl=116 time=25.2 ms
```

22) ip addr → Show IP details



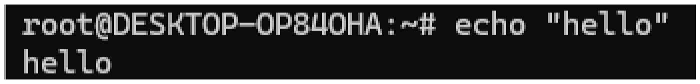
```
root@DESKTOP-OP84OHA:~# ip addr show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: bond0: <BROADCAST,MULTICAST,MASTER> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether 8a:6c:be:94:b7:07 brd ff:ff:ff:ff:ff:ff
3: dummy0: <BROADCAST,NOARP> mtu 1500 qdisc noop state DOWN group default qlen 1000
    link/ether fe:dc:c0:9c:9f:05 brd ff:ff:ff:ff:ff:ff
4: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
5: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN group default qlen 1000
    link/sit 0.0.0.0 brd 0.0.0.0
6: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:15:5d:d3:a0:9a brd ff:ff:ff:ff:ff:ff
    inet 172.26.106.197/20 brd 172.26.111.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::215:5dff:fed3:a09a/64 scope link
       valid_lft forever preferred_lft forever
```

23) lsblk → List block devices



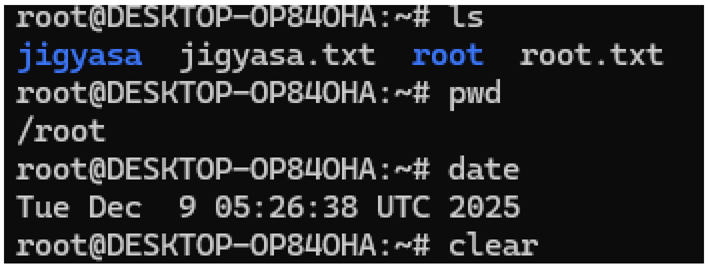
```
root@DESKTOP-OP84OHA:~# lsblk
NAME
    MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
sda    8:0    0  256G  0 disk
sdb    8:16   0  256G  0 disk /
```

24) echo → Print text

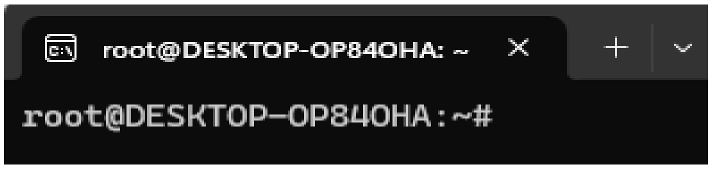


```
root@DESKTOP-OP84OHA:~# echo "hello"
hello
```

25) clear → Clear terminal



```
root@DESKTOP-OP84OHA:~# ls
jigyasa  jigyasa.txt  root  root.txt
root@DESKTOP-OP84OHA:~# pwd
/root
root@DESKTOP-OP84OHA:~# date
Tue Dec  9 05:26:38 UTC 2025
root@DESKTOP-OP84OHA:~# clear
```



```
root@DESKTOP-OP84OHA:~#
```

Lab no: 15 **Date: 2082/05/30**

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

Introduction

Process management is a fundamental concept in operating systems. A process is a running instance of a program that contains the program code and its current activity. Understanding how to create and terminate processes is crucial for system administration, programming, and managing system resources effectively.

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

Terminal: Ubuntu Terminal

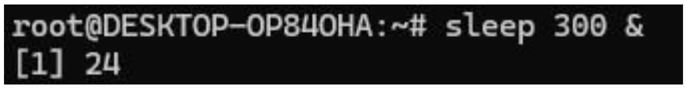
Operating System: Ubuntu Linux

Source code:

1. Process creation sleep 300 &   # Create a sleep

process for 300 seconds

[1] 24   # Process id

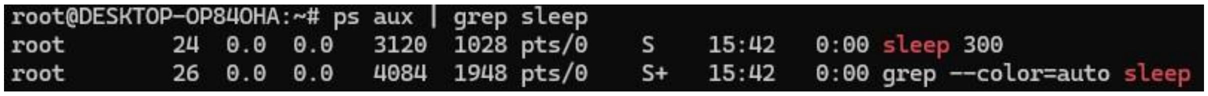


```
root@DESKTOP-OP84OHA:~# sleep 300 &
[1] 24
```

2. **Viewing Processes** ps aux | grep sleep # Find specific process root   24

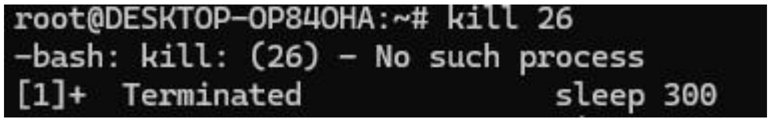
   0.0 0.0   3120 1028 pts/0   S   15:42   0:00 sleep 300 root   26 0.0 0.0

   4084 1948 pts/0   S+   15:42   0:00 grep --color=auto sleep



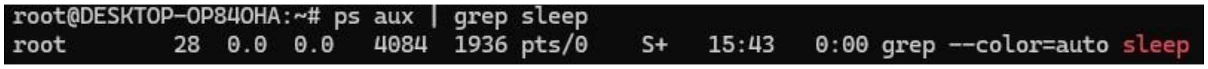
```
root@DESKTOP-OP84OHA:~# ps aux | grep sleep
root        24  0.0  0.0   3120  1028 pts/0    S    15:42   0:00 sleep 300
root        26  0.0  0.0   4084  1948 pts/0    S+   15:42   0:00 grep --color=auto sleep
```

3. **Terminating the process** kill 26   # Stop process by ID number



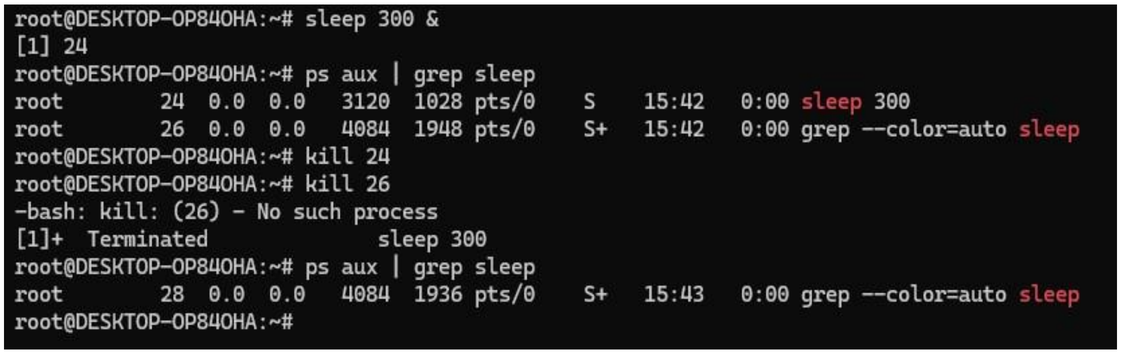
```
root@DESKTOP-OP84OHA:~# kill 26
-bash: kill: (26) - No such process
[1]+  Terminated              sleep 300
```

4. **Checking if the process is terminated or not**



```
root@DESKTOP-OP84OHA:~# ps aux | grep sleep
root        28  0.0  0.0   4084  1936 pts/0    S+   15:43   0:00 grep --color=auto sleep
```

Output:



```
root@DESKTOP-OP84OHA:~# sleep 300 &
[1] 24
root@DESKTOP-OP84OHA:~# ps aux | grep sleep
root        24  0.0  0.0   3120  1028 pts/0    S    15:42   0:00 sleep 300
root        26  0.0  0.0   4084  1948 pts/0    S+   15:42   0:00 grep --color=auto sleep
root@DESKTOP-OP84OHA:~# kill 24
root@DESKTOP-OP84OHA:~# kill 26
-bash: kill: (26) - No such process
[1]+  Terminated              sleep 300
root@DESKTOP-OP84OHA:~# ps aux | grep sleep
root        28  0.0  0.0   4084  1936 pts/0    S+   15:43   0:00 grep --color=auto sleep
root@DESKTOP-OP84OHA:~#
```

Process Information:

- **PID**: Process ID number (unique for each process)
- **Job**: Background job number [1], [2], etc.
- **Status**: Running, Terminated, Stopped **Conclusion:**

Successfully created and terminated processes in Ubuntu:

1. Created sleep process with sleep 300 &
2. Found process using ps aux | grep sleep
3. Terminated process with kill 26
4. Verified termination with ps aux | grep sleep