DATE - / /

EXPERIMENT - 1

AIM - Introduction to Linux and Vi editor.

THEORY

1. Introduction to Linux

Linux is a **free, open-source, multi-user, multitasking operating system** based on **UNIX**. It was created in **1991 by Linus Torvalds**. The core of Linux is the **kernel**, which manages hardware resources and enables communication between hardware and software.

Today, Linux powers a wide range of systems such as **servers, desktops, embedded devices, Android phones, cloud infrastructure, and supercomputers**. It is popular because of its **stability, security, flexibility, and cost-effectiveness**.Pp

Key Characteristics of Linux

1. **Open Source** – Anyone can download, use, modify, and distribute Linux.
2. **Multi-user System** – Multiple users can log in and work simultaneously without interfering with each other.
3. **Multitasking** – Linux can run multiple processes at the same time efficiently.
4. **Security** – Strong permission system prevents unauthorized access.
5. **Portability** – Runs on different hardware platforms, from PCs to embedded systems.
6. **Community Support** – Thousands of developers contribute worldwide.

Structure of Linux System

- **Kernel** → The core of the system, manages hardware.
- **Shell** → Command interpreter between user and kernel.
- **Utilities/Commands** → Basic programs like ls, cp, mv.
- **Applications** → User-installed software (browsers, editors, etc.).
-

Common Linux Commands

- pwd → Shows current directory
- ls → Lists files and directories
- cd → Changes directory

- mkdir → Creates a new directory
- rm → Deletes a file
- cp file1 file2 → Copies file1 to file2
- mv file1 file2 → Moves/renames a file
- cat filename → Displays file contents
- chmod → Changes permissions

---

2. Introduction to Vi Editor

The **Vi editor** (short for *Visual Editor*) is a **screen-based text editor** used in UNIX and Linux systems. It is very popular because it is **lightweight, fast, and available by default on all Linux distributions**. Vi is mostly used for editing **configuration files, scripts, and source code** directly from the terminal.

Although Vi looks simple, it is extremely powerful once mastered.

Modes of Vi Editor

Vi operates in three major modes:

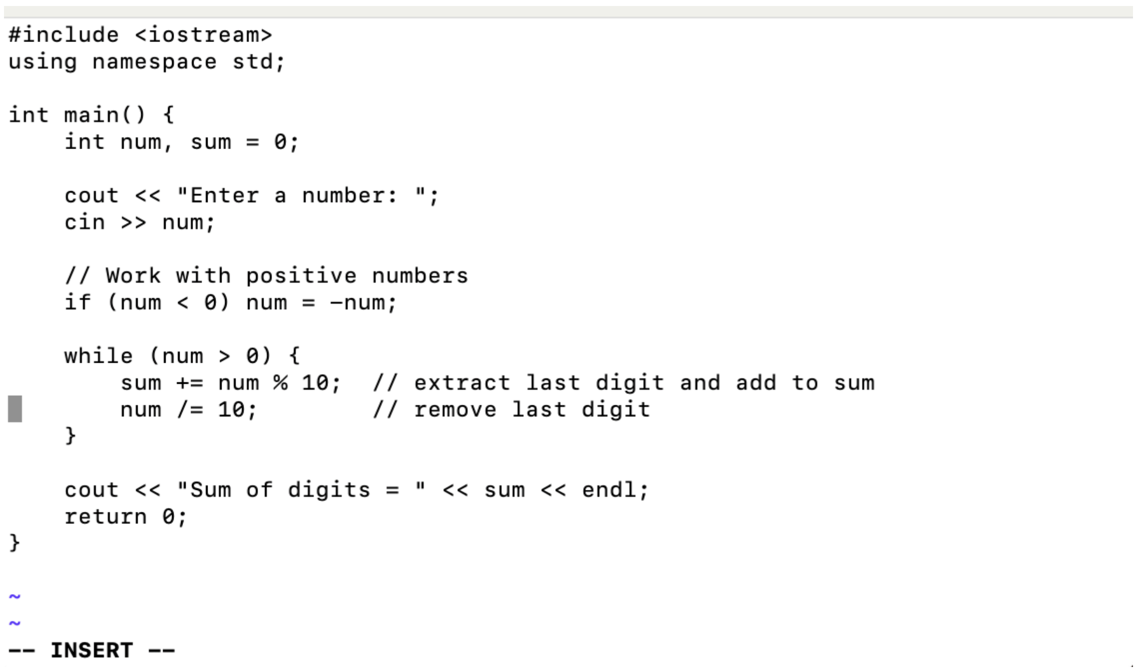
1. **Command Mode (default mode)**
   - Every time you open Vi, it starts in command mode.
   - Keystrokes are treated as commands (not inserted as text).
   - Example commands:
     - dd → delete a line
     - yy → copy a line
     - p → paste
     - u → undo last change
2. **Insert Mode**
   - Used for entering and editing text.
   - To enter insert mode, press:
     - i → insert before cursor
     - a → insert after cursor
     - o → open new line below cursor

    - Press Esc to return to command mode.

3. **Last Line Mode (Ex Mode)**
    - Accessed by pressing : from command mode.
    - Allows advanced commands for saving, quitting, searching, etc.
    - Examples:
        - :w → save file
        - :q → quit
        - :wq → save and quit
        - :q! → quit without saving

Common Vi Editor Commands

Command	**Function**
i	Switch to insert mode
a	Insert after cursor
o	Open new line below
x	Delete a character
dd	Delete a line
yy	Copy a line
p	Paste after cursor
u	Undo last change
:w	Save file
:q	Quit Vi
:wq	Save and quit
:q!	Quit without saving



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

int main() {
    int num, sum = 0;

    cout << "Enter a number: ";
    cin >> num;

    // Work with positive numbers
    if (num < 0) num = -num;

    while (num > 0) {
        sum += num % 10;  // extract last digit and add to sum
        num /= 10;        // remove last digit
    }

    cout << "Sum of digits = " << sum << endl;
    return 0;
}

~
~
-- INSERT --
```

DATE -   /   /

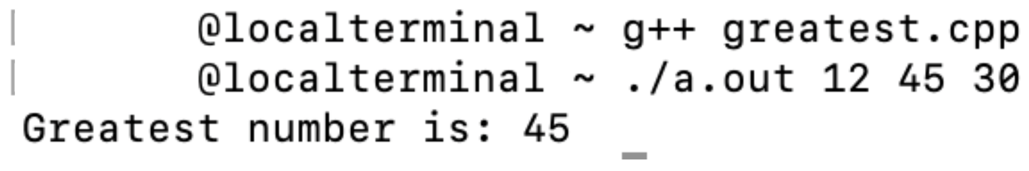
EXPERIMENT - 2

AIM - Write a program to find the greatest of three numbers (numbers passed as command line parameters)

CODE :

```
#include <iostream>
#include <string>

using namespace std;

int main(int argc, char* argv[]) {
    if (argc < 4) {
        cout << "Please provide 3 numbers as command line arguments." << endl;
        return 1;
    }

    int a = stoi(argv[1]);
    int b = stoi(argv[2]);
    int c = stoi(argv[3]);

    int greatest = a;

    if (b > greatest) greatest = b;
    if (c > greatest) greatest = c;

    cout << "Greatest number is: " << greatest << endl;

    return 0;
}
```

OUTPUT :



```
        @localterminal ~ g++ greatest.cpp
        @localterminal ~ ./a.out 12 45 30
 Greatest number is: 45
```

DATE -   /   /

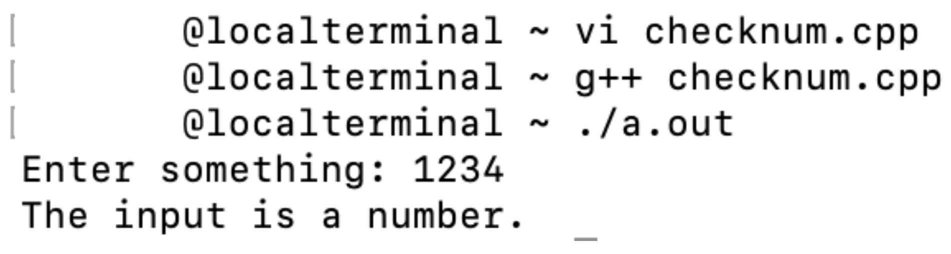
EXPERIMENT - 3

AIM - Write a program to check whether the given input is a number or a string

CODE :

```
#include <iostream>
#include <cctype>   // for isdigit
#include <string>
using namespace std;

int main() {
    string input;
    cout << "Enter something: ";
    cin >> input;

    bool isNumber = true;

    for (size_t i = 0; i < input.length(); i++) {
        if (!isdigit(input[i])) {
            isNumber = false;
            break;
        }
    }

    if (isNumber) {
        cout << "The input is a number." << endl;
    } else {
        cout << "The input is a string." << endl;
    }

    return 0;
}
```

OUTPUT :



```
[       @localterminal ~ vi checknum.cpp
[       @localterminal ~ g++ checknum.cpp
[       @localterminal ~ ./a.out
 Enter something: 1234
 The input is a number.  _
```

DATE -   /   /

EXPERIMENT - 4

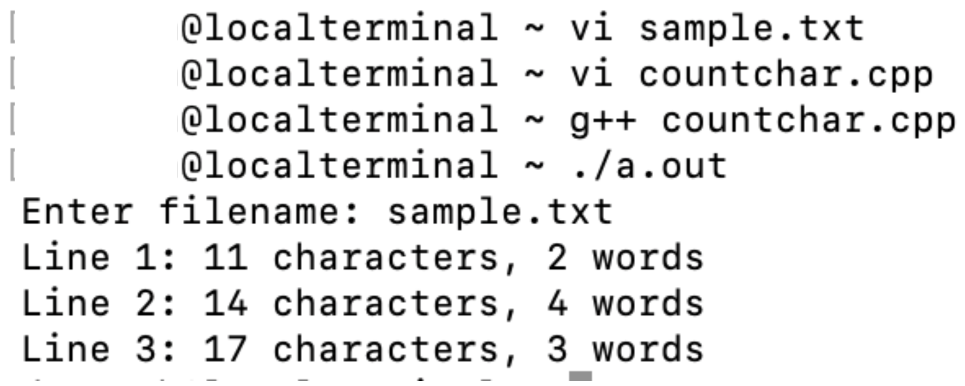
AIM - Write a program to compute no. of characters and words in each line of given file

CODE :

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
using namespace std;

int main() {
    string filename;
    cout << "Enter filename: ";
    cin >> filename;

    ifstream file(filename);
    if (!file) {
        cout << "Error: Could not open file." << endl;
        return 1;
    }

    string line;
    int lineNum = 1;

    while (getline(file, line)) {
        int charCount = line.length();

        // Count words using stringstream
        stringstream ss(line);
        string word;
        int wordCount = 0;
        while (ss >> word) {
            wordCount++;
        }

        cout << "Line " << lineNum << ": "
             << charCount << " characters, "
             << wordCount << " words" << endl;

        lineNum++;
    }

    file.close();
    return 0;
```

}

OUTPUT :



```
[       @localterminal ~ vi sample.txt
[       @localterminal ~ vi countchar.cpp
[       @localterminal ~ g++ countchar.cpp
[       @localterminal ~ ./a.out
 Enter filename: sample.txt
 Line 1: 11 characters, 2 words
 Line 2: 14 characters, 4 words
 Line 3: 17 characters, 3 words
```

DATE -   /   /

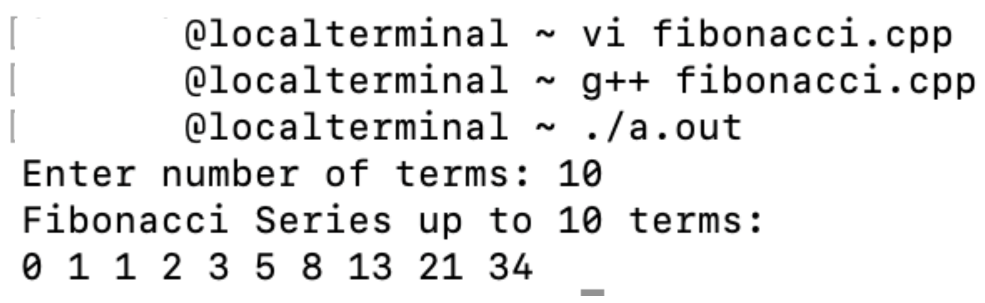
EXPERIMENT - 5

AIM - Write a program to print the Fibonacci series upto n terms

CODE :

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

int main() {
    int n;
    cout << "Enter number of terms: ";
    cin >> n;

    if (n <= 0) {
        cout << "Please enter a positive integer." << endl;
        return 1;
    }

    int a = 0, b = 1;

    cout << "Fibonacci Series up to " << n << " terms:" << endl;

    for (int i = 1; i <= n; i++) {
        cout << a << " ";
        int next = a + b;
        a = b;
        b = next;
    }

    cout << endl;
    return 0;
}
```

OUTPUT :



```
[       @localterminal ~ vi fibonacci.cpp
[       @localterminal ~ g++ fibonacci.cpp
[       @localterminal ~ ./a.out
 Enter number of terms: 10
 Fibonacci Series up to 10 terms:
 0 1 1 2 3 5 8 13 21 34 _
```

DATE -   /   /

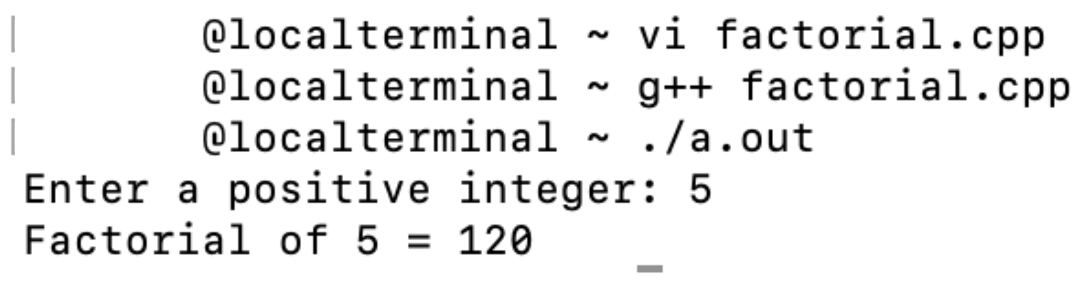
EXPERIMENT - 6

AIM - Write a program to calculate the factorial of a given number

CODE :

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

int main() {
   int n;
   long long factorial = 1;  // use long long for large results

   cout << "Enter a positive integer: ";
   cin >> n;

   if (n < 0) {
      cout << "Factorial is not defined for negative numbers." << endl;
   } else {
      for (int i = 1; i <= n; i++) {
         factorial *= i;
      }
      cout << "Factorial of " << n << " = " << factorial << endl;
   }

   return 0;
}
```

OUTPUT :



```
|       @localterminal ~ vi factorial.cpp
|       @localterminal ~ g++ factorial.cpp
|       @localterminal ~ ./a.out
 Enter a positive integer: 5
 Factorial of 5 = 120
```

DATE -   /   /

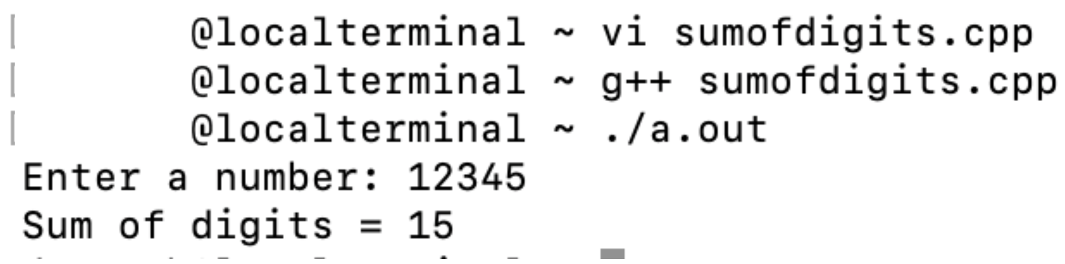
EXPERIMENT - 7

AIM - Write a program to calculate the sum of digits of the given number

CODE :

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

int main() {
   int num, sum = 0;

   cout << "Enter a number: ";
   cin >> num;


   if (num < 0) num = -num;

   while (num > 0) {
      sum += num % 10;
      num /= 10;
   }

   cout << "Sum of digits = " << sum << endl;
   return 0;
}
```

OUTPUT :



```
|        @localterminal ~ vi sumofdigits.cpp
|        @localterminal ~ g++ sumofdigits.cpp
|        @localterminal ~ ./a.out
 Enter a number: 12345
 Sum of digits = 15
```

DATE -   /   /

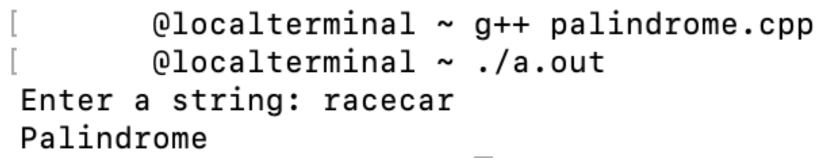
EXPERIMENT - 8

AIM - Write a program to check whether the given string is a palindrome

CODE :

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

int main() {
    string s, rev;
    cout << "Enter a string: ";
    getline(cin, s);

    rev = string(s.rbegin(), s.rend());

    if (s == rev)
        cout << "Palindrome\n";
    else
        cout << "Not a palindrome\n";

    return 0;
}
```

OUTPUT :



```
[       @localterminal ~ g++ palindrome.cpp
[       @localterminal ~ ./a.out
 Enter a string: racecar
 Palindrome
```

DATE - / /

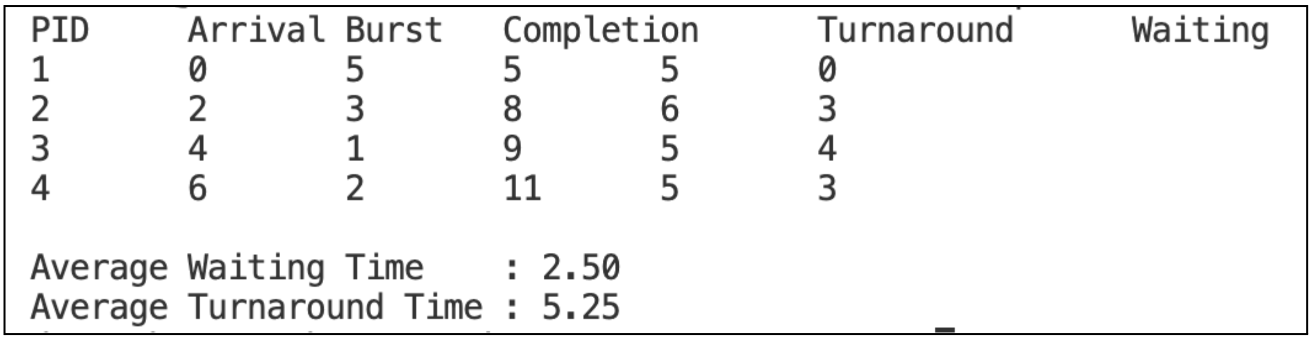
EXPERIMENT - 9

AIM - Write a program to implement CPU scheduling for first come first serve.

CODE :

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

struct Process {
    int pid;
    int arrival;
    int burst;
    int completion;
    int turnaround;
    int waiting;
};

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    vector<Process> procs = {
        {1, 0, 5, 0, 0, 0},
        {2, 2, 3, 0, 0, 0},
        {3, 4, 1, 0, 0, 0},
        {4, 6, 2, 0, 0, 0}
    };

    stable_sort(procs.begin(), procs.end(), [](const Process &a, const Process &b){
        return a.arrival < b.arrival;
    });

    int currentTime = 0;
    long long totalWT = 0, totalTAT = 0;
    int n = procs.size();

    for (int i = 0; i < n; ++i) {
        if (currentTime < procs[i].arrival) currentTime = procs[i].arrival;
        procs[i].completion = currentTime + procs[i].burst;
        procs[i].turnaround = procs[i].completion - procs[i].arrival;
        procs[i].waiting = procs[i].turnaround - procs[i].burst;
        currentTime = procs[i].completion;
        totalWT += procs[i].waiting;
        totalTAT += procs[i].turnaround;
    }

    cout << "PID\tArrival\tBurst\tCompletion\tTurnaround\tWaiting\n";
    for (const auto &p : procs) {
        cout << p.pid << '\t' << p.arrival << '\t' << p.burst << '\t'
             << p.completion << '\t' << p.turnaround << '\t' << p.waiting << '\n';
    }

    cout << fixed << setprecision(2);
    cout << "\nAverage Waiting Time    : " << (double)totalWT / n << "\n";
    cout << "Average Turnaround Time : " << (double)totalTAT / n << "\n";


    return 0;
}
```

OUTPUT :



```
PID     Arrival Burst   Completion      Turnaround      Waiting
1       0       5       5       5       0
2       2       3       8       6       3
3       4       1       9       5       4
4       6       2       11      5       3

Average Waiting Time    : 2.50
Average Turnaround Time : 5.25
```

DATE -   /   /

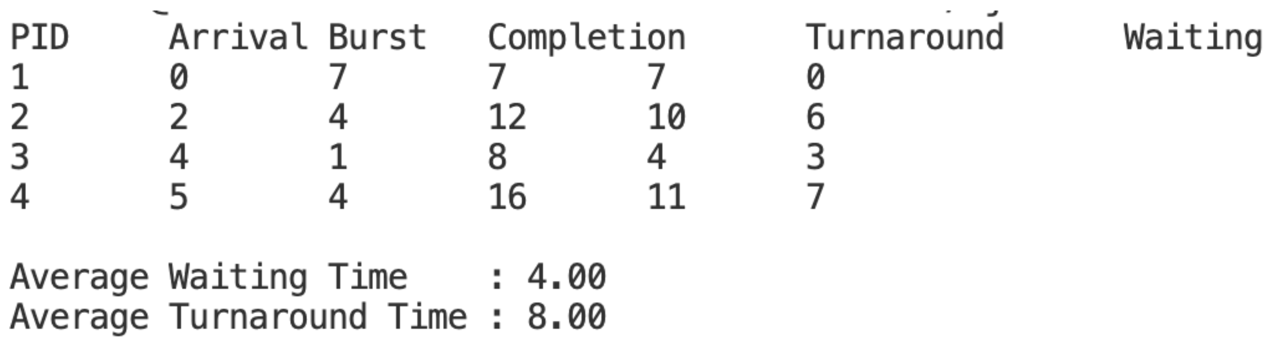
EXPERIMENT - 10

AIM - Write a program to implement CPU scheduling for shortest job first.

CODE :

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

struct Process {
   int pid;
   int arrival;
   int burst;
   int completion;
   int turnaround;
   int waiting;
   bool done;
};

int main() {
   ios::sync_with_stdio(false);
   cin.tie(nullptr);

   vector<Process> procs = {
      {1, 0, 7, 0, 0, 0, false},
      {2, 2, 4, 0, 0, 0, false},
      {3, 4, 1, 0, 0, 0, false},
      {4, 5, 4, 0, 0, 0, false}
   };

   int n = procs.size();
   int currentTime = 0, completed = 0;
   double totalWT = 0, totalTAT = 0;

   while (completed < n) {
      int idx = -1;
      int minBurst = INT_MAX;

      for (int i = 0; i < n; ++i) {
         if (!procs[i].done && procs[i].arrival <= currentTime) {
            if (procs[i].burst < minBurst) {
               minBurst = procs[i].burst;
               idx = i;
            }
            }
        }

        if (idx == -1) {
            currentTime++;
        } else {
            currentTime += procs[idx].burst;
            procs[idx].completion = currentTime;
            procs[idx].turnaround = procs[idx].completion - procs[idx].arrival;
            procs[idx].waiting = procs[idx].turnaround - procs[idx].burst;
            procs[idx].done = true;

            totalWT += procs[idx].waiting;
            totalTAT += procs[idx].turnaround;
            completed++;
        }
    }

    cout << "PID\tArrival\tBurst\tCompletion\tTurnaround\tWaiting\n";
    for (const auto &p : procs) {
        cout << p.pid << '\t' << p.arrival << '\t' << p.burst << '\t'
             << p.completion << '\t' << p.turnaround << '\t' << p.waiting << '\n';
    }

    cout << fixed << setprecision(2);
    cout << "\nAverage Waiting Time    : " << totalWT / n << "\n";
    cout << "Average Turnaround Time : " << totalTAT / n << "\n";

    return 0;
}
```

OUTPUT :



```
PID     Arrival Burst   Completion      Turnaround      Waiting
1       0       7       7       7       0
2       2       4       12      10      6
3       4       1       8       4       3
4       5       4       16      11      7

Average Waiting Time    : 4.00
Average Turnaround Time : 8.00
```

DATE -   /   /

EXPERIMENT - 11

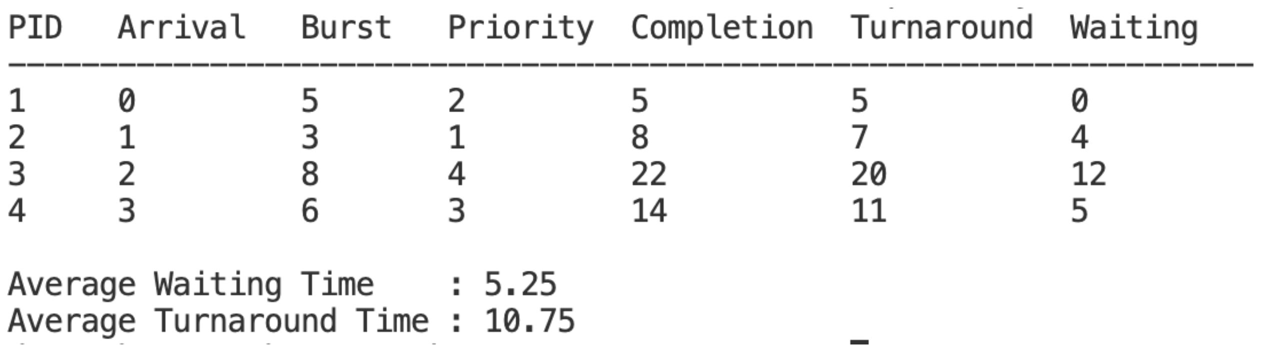
AIM - Write a program to perform priority scheduling.

CODE :

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

struct Process {
   int pid, arrival, burst, priority, completion, turnaround, waiting;
   bool done;
};

int main() {
   vector<Process> procs = {
      {1, 0, 5, 2, 0, 0, 0, false},
      {2, 1, 3, 1, 0, 0, 0, false},
      {3, 2, 8, 4, 0, 0, 0, false},
      {4, 3, 6, 3, 0, 0, 0, false}
   };

   int n = procs.size();
   int currentTime = 0, completed = 0;
   double totalWT = 0, totalTAT = 0;

   while (completed < n) {
      int idx = -1, highestPriority = INT_MAX;

      for (int i = 0; i < n; ++i) {
         if (!procs[i].done && procs[i].arrival <= currentTime) {
            if (procs[i].priority < highestPriority) {
               highestPriority = procs[i].priority;
               idx = i;
            }
         }
      }

      if (idx == -1) {
         currentTime++;
      } else {
         currentTime += procs[idx].burst;
         procs[idx].completion = currentTime;
         procs[idx].turnaround = procs[idx].completion - procs[idx].arrival;
```

```
            procs[idx].waiting = procs[idx].turnaround - procs[idx].burst;
            procs[idx].done = true;

            totalWT += procs[idx].waiting;
            totalTAT += procs[idx].turnaround;
            completed++;
        }
    }

    // Output formatting
    cout << left
         << setw(6)  << "PID"
         << setw(10) << "Arrival"
         << setw(8)  << "Burst"
         << setw(10) << "Priority"
         << setw(12) << "Completion"
         << setw(12) << "Turnaround"
         << setw(10) << "Waiting" << "\n";

    cout << string(68, '-') << "\n";

    for (auto &p : procs) {
        cout << left
             << setw(6)  << p.pid
             << setw(10) << p.arrival
             << setw(8)  << p.burst
             << setw(10) << p.priority
             << setw(12) << p.completion
             << setw(12) << p.turnaround
             << setw(10) << p.waiting << "\n";
    }

    cout << fixed << setprecision(2);
    cout << "\nAverage Waiting Time    : " << totalWT / n << "\n";
    cout << "Average Turnaround Time : " << totalTAT / n << "\n";

    return 0;
}
```

OUTPUT :



```
PID   Arrival   Burst   Priority  Completion  Turnaround  Waiting
--------------------------------------------------------------------
1     0         5       2         5           5           0
2     1         3       1         8           7           4
3     2         8       4         22          20          12
4     3         6       3         14          11          5

Average Waiting Time    : 5.25
Average Turnaround Time : 10.75
```

DATE -   /   /

EXPERIMENT - 12

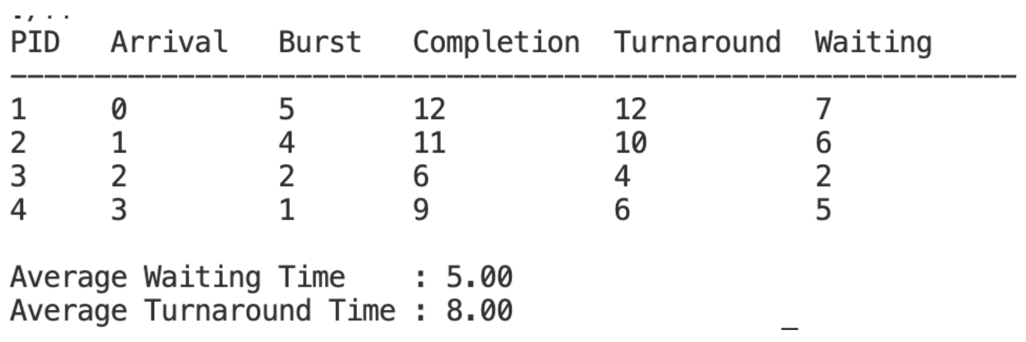
AIM - Write a program to implement CPU scheduling for Round Robin

CODE :

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

struct Process {
    int pid, arrival, burst, remaining, completion, turnaround, waiting;
};

int main() {
    vector<Process> p = {
        {1, 0, 5, 5, 0, 0, 0},
        {2, 1, 4, 4, 0, 0, 0},
        {3, 2, 2, 2, 0, 0, 0},
        {4, 3, 1, 1, 0, 0, 0}
    };

    int n = p.size();
    int tq = 2;   // Time Quantum
    int time = 0, completed = 0;

    queue<int> q;
    vector<bool> inQueue(n, false);

    q.push(0);
    inQueue[0] = true;

    while (!q.empty()) {
        int i = q.front();
        q.pop();

        if (p[i].remaining > tq) {
            p[i].remaining -= tq;
            time += tq;
        } else {
            time += p[i].remaining;
            p[i].remaining = 0;
            p[i].completion = time;
            p[i].turnaround = p[i].completion - p[i].arrival;
            p[i].waiting = p[i].turnaround - p[i].burst;
```

```cpp
            completed++;
        }

        for (int j = 0; j < n; j++) {
            if (!inQueue[j] && p[j].arrival <= time && p[j].remaining > 0) {
                q.push(j);
                inQueue[j] = true;
            }
        }

        if (p[i].remaining > 0) q.push(i);
        if (completed == n) break;
    }

    cout << left
         << setw(6)  << "PID"
         << setw(10) << "Arrival"
         << setw(8)  << "Burst"
         << setw(12) << "Completion"
         << setw(12) << "Turnaround"
         << setw(10) << "Waiting" << "\n";

    cout << string(60, '-') << "\n";

    double totalWT = 0, totalTAT = 0;
    for (auto &x : p) {
        totalWT += x.waiting;
        totalTAT += x.turnaround;

        cout << left
             << setw(6)  << x.pid
             << setw(10) << x.arrival
             << setw(8)  << x.burst
             << setw(12) << x.completion
             << setw(12) << x.turnaround
             << setw(10) << x.waiting << "\n";
    }

    cout << fixed << setprecision(2);
    cout << "\nAverage Waiting Time    : " << totalWT / n << "\n";
    cout << "Average Turnaround Time : " << totalTAT / n << "\n";
    return 0;
}
```

OUTPUT :



```
PID   Arrival   Burst   Completion  Turnaround  Waiting
------------------------------------------------------------
1     0         5       12          12          7
2     1         4       11          10          6
3     2         2       6           4           2
4     3         1       9           6           5

Average Waiting Time    : 5.00
Average Turnaround Time : 8.00
```

DATE -   /   /

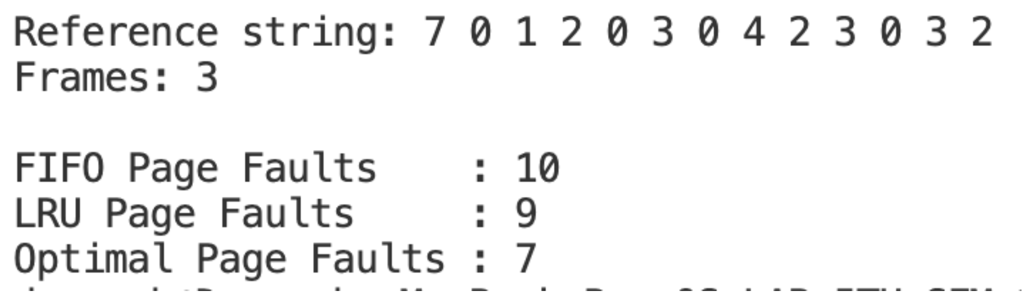
EXPERIMENT - 13

AIM - Write a program for page replacement policy using a) LRU b) FIFO c) Optimal

CODE :

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

int simulateFIFO(const vector<int>& refs, int frames) {
    unordered_set<int> s;
    queue<int> q;
    int faults = 0;
    for (int pg : refs) {
        if (s.find(pg) == s.end()) {
            faults++;
            if ((int)s.size() == frames) {
                int old = q.front(); q.pop();
                s.erase(old);
            }
            s.insert(pg);
            q.push(pg);
        }
    }
    return faults;
}

int simulateLRU(const vector<int>& refs, int frames) {
    unordered_map<int,int> pos;
    list<int> l;
    int faults = 0;
    for (int i = 0; i < (int)refs.size(); ++i) {
        int pg = refs[i];
        if (pos.find(pg) == pos.end()) {
            faults++;
            if ((int)l.size() == frames) {
                int last = l.back();
                l.pop_back();
                pos.erase(last);
            }
            l.push_front(pg);
            pos[pg] = 0;
        } else {
            auto it = find(l.begin(), l.end(), pg);
            l.erase(it);
            l.push_front(pg);
        }
    }
    return faults;
}

int simulateOptimal(const vector<int>& refs, int frames) {
    int n = refs.size();
    unordered_set<int> s;
    int faults = 0;
    for (int i = 0; i < n; ++i) {
        int pg = refs[i];
        if (s.find(pg) != s.end()) continue;
        faults++;
        if ((int)s.size() < frames) {
            s.insert(pg);
        } else {
            int farthest = -1, replace = -1;
            for (int p : s) {
                int j;
                for (j = i + 1; j < n; ++j)
                    if (refs[j] == p) break;

                if (j == n) { replace = p; break; }
                if (j > farthest) { farthest = j; replace = p; }
            }
            s.erase(replace);
            s.insert(pg);
        }
    }
    return faults;
}

int main() {
    vector<int> refs = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
    int frames = 3;

    cout << "Reference string: ";
    for (int x : refs) cout << x << " ";
    cout << "\nFrames: " << frames << "\n\n";

    int fifoFaults = simulateFIFO(refs, frames);
    int lruFaults = simulateLRU(refs, frames);
    int optFaults = simulateOptimal(refs, frames);

    cout << "FIFO Page Faults    : " << fifoFaults << "\n";
    cout << "LRU Page Faults      : " << lruFaults << "\n";
    cout << "Optimal Page Faults : " << optFaults << "\n";

    return 0;
}
```

OUTPUT :



```
Reference string: 7 0 1 2 0 3 0 4 2 3 0 3 2
Frames: 3

FIFO Page Faults    : 10
LRU Page Faults     : 9
Optimal Page Faults : 7
```

DATE -   /   /

EXPERIMENT - 14

AIM - Write a program to implement first fit, best fit and worst fit algorithm for memory management

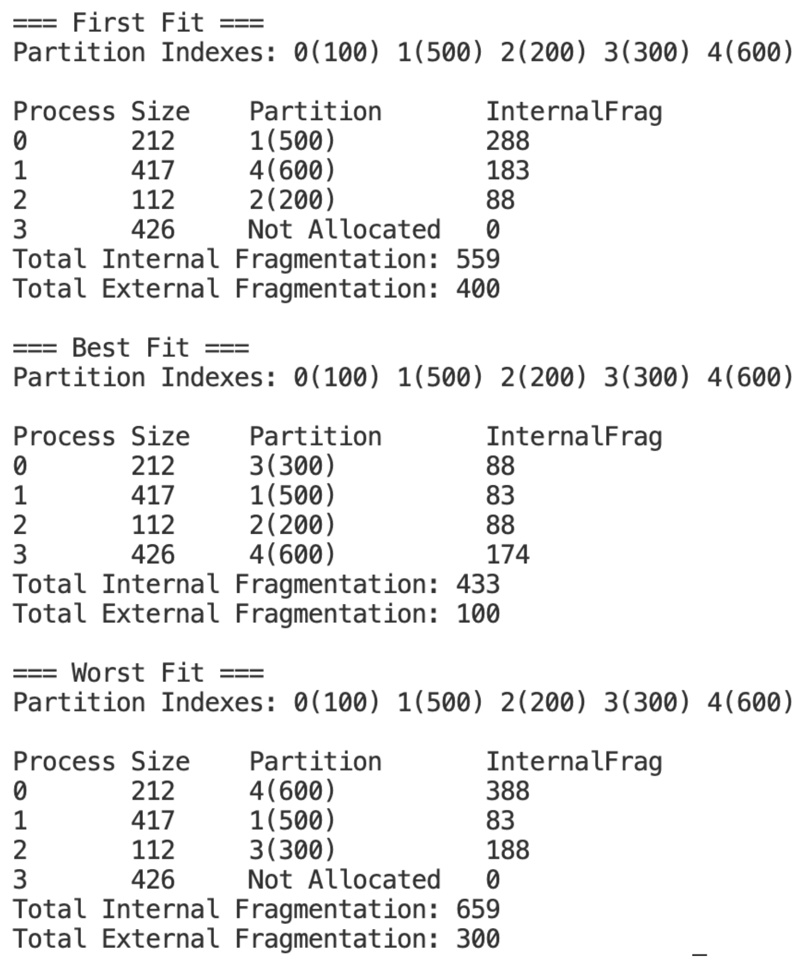
CODE :

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

struct Result {
   vector<int> alloc;
   vector<int> internalFrag;
   int totalInternal;
   int totalExternal;
};

Result firstFit(const vector<int>& parts, const vector<int>& procs) {
   int m = parts.size(), n = procs.size();
   vector<int> alloc(n, -1), internalFrag(n, 0);
   vector<bool> used(m, false);

   for (int i = 0; i < n; ++i) {
      for (int j = 0; j < m; ++j) {
         if (!used[j] && parts[j] >= procs[i]) {
            alloc[i] = j;
            internalFrag[i] = parts[j] - procs[i];
            used[j] = true;
            break;
         }
      }
   }
   int totalInternal = accumulate(internalFrag.begin(), internalFrag.end(), 0);
   int totalExternal = 0;
   for (int j = 0; j < m; ++j)
      if (!used[j]) totalExternal += parts[j];

   return {alloc, internalFrag, totalInternal, totalExternal};
}

Result bestFit(const vector<int>& parts, const vector<int>& procs) {
   int m = parts.size(), n = procs.size();
   vector<int> alloc(n, -1), internalFrag(n, 0);
   vector<bool> used(m, false);
```

```
    for (int i = 0; i < n; ++i) {
        int bestIdx = -1, bestSize = INT_MAX;
        for (int j = 0; j < m; ++j) {
            if (!used[j] && parts[j] >= procs[i] && parts[j] < bestSize) {
                bestSize = parts[j];
                bestIdx = j;
            }
        }
        if (bestIdx != -1) {
            alloc[i] = bestIdx;
            internalFrag[i] = parts[bestIdx] - procs[i];
            used[bestIdx] = true;
        }
    }

    int totalInternal = accumulate(internalFrag.begin(), internalFrag.end(), 0);
    int totalExternal = 0;
    for (int j = 0; j < m; ++j)
        if (!used[j]) totalExternal += parts[j];

    return {alloc, internalFrag, totalInternal, totalExternal};
}

Result worstFit(const vector<int>& parts, const vector<int>& procs) {
    int m = parts.size(), n = procs.size();
    vector<int> alloc(n, -1), internalFrag(n, 0);
    vector<bool> used(m, false);

    for (int i = 0; i < n; ++i) {
        int worstIdx = -1, worstSize = -1;
        for (int j = 0; j < m; ++j) {
            if (!used[j] && parts[j] >= procs[i] && parts[j] > worstSize) {
                worstSize = parts[j];
                worstIdx = j;
            }
        }
        if (worstIdx != -1) {
            alloc[i] = worstIdx;
            internalFrag[i] = parts[worstIdx] - procs[i];
            used[worstIdx] = true;
        }
    }

    int totalInternal = accumulate(internalFrag.begin(), internalFrag.end(), 0);
    int totalExternal = 0;
    for (int j = 0; j < m; ++j)
        if (!used[j]) totalExternal += parts[j];

    return {alloc, internalFrag, totalInternal, totalExternal};
```

```
}

void printResult(const string& name,
                 const vector<int>& parts,
                 const vector<int>& procs,
                 const Result& r) {
    cout << "\n=== " << name << " ===\n";
    cout << "Partition Indexes: ";
    for (size_t i = 0; i < parts.size(); ++i)
        cout << i << "(" << parts[i] << ") ";
    cout << "\n\nProcess\tSize\tPartition\tInternalFrag\n";

    for (size_t i = 0; i < procs.size(); ++i) {
        cout << i << "\t" << procs[i] << "\t";
        if (r.alloc[i] == -1) cout << "Not Allocated\t";
        else cout << r.alloc[i] << "(" << parts[r.alloc[i]] << ")\t\t";
        cout << r.internalFrag[i] << "\n";
    }

    cout << "Total Internal Fragmentation: " << r.totalInternal << "\n";
    cout << "Total External Fragmentation: " << r.totalExternal << "\n";
}

int main() {
    vector<int> partitions = {100, 500, 200, 300, 600};
    vector<int> processes = {212, 417, 112, 426};

    Result r1 = firstFit(partitions, processes);
    Result r2 = bestFit(partitions, processes);
    Result r3 = worstFit(partitions, processes);

    printResult("First Fit", partitions, processes, r1);
    printResult("Best Fit", partitions, processes, r2);
    printResult("Worst Fit", partitions, processes, r3);

    return 0;
}
```

OUTPUT :



```
=== First Fit ===
Partition Indexes: 0(100) 1(500) 2(200) 3(300) 4(600)

Process Size    Partition       InternalFrag
0       212     1(500)          288
1       417     4(600)          183
2       112     2(200)          88
3       426     Not Allocated   0
Total Internal Fragmentation: 559
Total External Fragmentation: 400

=== Best Fit ===
Partition Indexes: 0(100) 1(500) 2(200) 3(300) 4(600)

Process Size    Partition       InternalFrag
0       212     3(300)          88
1       417     1(500)          83
2       112     2(200)          88
3       426     4(600)          174
Total Internal Fragmentation: 433
Total External Fragmentation: 100

=== Worst Fit ===
Partition Indexes: 0(100) 1(500) 2(200) 3(300) 4(600)

Process Size    Partition       InternalFrag
0       212     4(600)          388
1       417     1(500)          83
2       112     3(300)          188
3       426     Not Allocated   0
Total Internal Fragmentation: 659
Total External Fragmentation: 300
```

DATE -   /   /

EXPERIMENT - 15

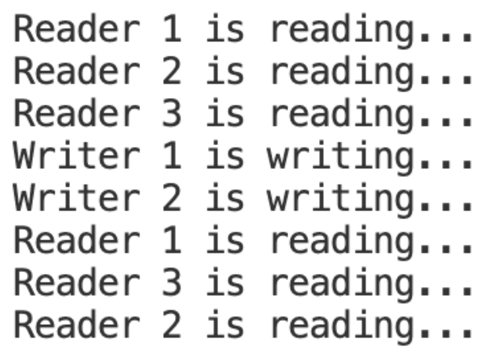
AIM - Write a program to implement reader/writer problem using semaphore.

CODE :

```
#include <iostream>
#include <pthread.h>
#include <dispatch/dispatch.h>
#include <unistd.h>
using namespace std;

dispatch_semaphore_t mutexLock;     // protects readCount
dispatch_semaphore_t writeLock;     // ensures writer exclusive access
pthread_mutex_t printLock;          // keeps console output clean

int readCount = 0;

void safePrint(const string& msg) {
   pthread_mutex_lock(&printLock);
   cout << msg << endl;
   pthread_mutex_unlock(&printLock);
}

void* reader(void* arg) {
   int id = *((int*)arg);

   while (true) {
      dispatch_semaphore_wait(mutexLock, DISPATCH_TIME_FOREVER);
      readCount++;
      if (readCount == 1)
         dispatch_semaphore_wait(writeLock, DISPATCH_TIME_FOREVER);
      dispatch_semaphore_signal(mutexLock);

      safePrint("Reader " + to_string(id) + " is reading...");
      sleep(1);

      dispatch_semaphore_wait(mutexLock, DISPATCH_TIME_FOREVER);
      readCount--;
      if (readCount == 0)
         dispatch_semaphore_signal(writeLock);
      dispatch_semaphore_signal(mutexLock);

      sleep(1);
   }
   return NULL;
}
```

```
void* writer(void* arg) {
    int id = *((int*)arg);

    while (true) {
        dispatch_semaphore_wait(writeLock, DISPATCH_TIME_FOREVER);

        safePrint("Writer " + to_string(id) + " is writing...");
        sleep(2);

        dispatch_semaphore_signal(writeLock);
        sleep(2);
    }
    return NULL;
}

int main() {
    pthread_t r[3], w[2];
    int rID[3] = {1, 2, 3};
    int wID[2] = {1, 2};

    mutexLock = dispatch_semaphore_create(1);
    writeLock = dispatch_semaphore_create(1);
    pthread_mutex_init(&printLock, NULL);

    for (int i = 0; i < 3; i++)
        pthread_create(&r[i], NULL, reader, &rID[i]);

    for (int i = 0; i < 2; i++)
        pthread_create(&w[i], NULL, writer, &wID[i]);

    for (int i = 0; i < 3; i++)
        pthread_join(r[i], NULL);

    for (int i = 0; i < 2; i++)
        pthread_join(w[i], NULL);

    return 0;
}
```

OUTPUT :



```
Reader 1 is reading...
Reader 2 is reading...
Reader 3 is reading...
Writer 1 is writing...
Writer 2 is writing...
Reader 1 is reading...
Reader 3 is reading...
Reader 2 is reading...
```

DATE -   /   /

EXPERIMENT - 16

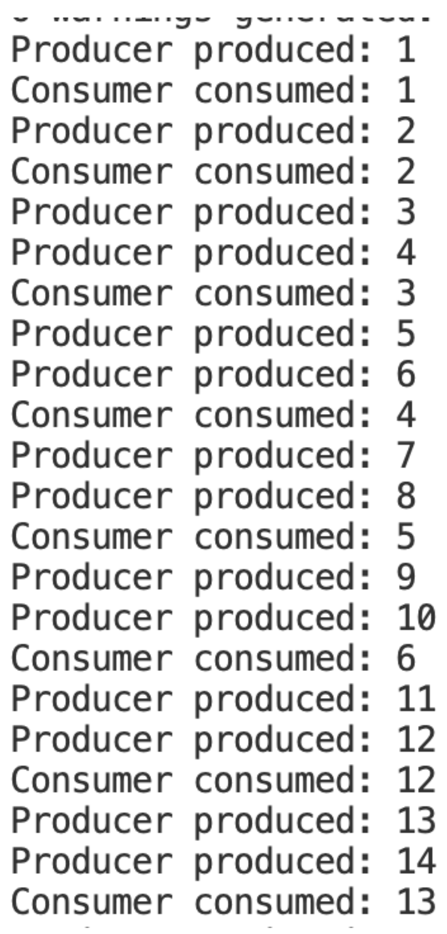
AIM - Write a program to implement producer-consumer problem using semaphore

CODE :

```
#include <iostream>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

using namespace std;

#define BUFFER_SIZE 5

int buffer[BUFFER_SIZE];
int inIndex = 0, outIndex = 0;

sem_t emptySlots;
sem_t fullSlots;
sem_t mutexLock;

void* producer(void* arg) {
   int item = 1;
   while (true) {
      sem_wait(&emptySlots);
      sem_wait(&mutexLock);

      buffer[inIndex] = item;
      cout << "Producer produced: " << item << endl;
      inIndex = (inIndex + 1) % BUFFER_SIZE;
      item++;

      sem_post(&mutexLock);
      sem_post(&fullSlots);

      sleep(1);
   }
   return NULL;
}

void* consumer(void* arg) {
   while (true) {
      sem_wait(&fullSlots);
      sem_wait(&mutexLock);

      int item = buffer[outIndex];
```

```
        cout << "Consumer consumed: " << item << endl;
        outIndex = (outIndex + 1) % BUFFER_SIZE;

        sem_post(&mutexLock);
        sem_post(&emptySlots);

        sleep(2);
    }
    return NULL;
}

int main() {
    pthread_t prodThread, consThread;

    sem_init(&emptySlots, 0, BUFFER_SIZE);
    sem_init(&fullSlots, 0, 0);
    sem_init(&mutexLock, 0, 1);

    pthread_create(&prodThread, NULL, producer, NULL);
    pthread_create(&consThread, NULL, consumer, NULL);

    pthread_join(prodThread, NULL);
    pthread_join(consThread, NULL);

    sem_destroy(&emptySlots);
    sem_destroy(&fullSlots);
    sem_destroy(&mutexLock);

    return 0;
}
```

OUTPUT :



```
6 warnings generated.
Producer produced: 1
Consumer consumed: 1
Producer produced: 2
Consumer consumed: 2
Producer produced: 3
Producer produced: 4
Consumer consumed: 3
Producer produced: 5
Producer produced: 6
Consumer consumed: 4
Producer produced: 7
Producer produced: 8
Consumer consumed: 5
Producer produced: 9
Producer produced: 10
Consumer consumed: 6
Producer produced: 11
Producer produced: 12
Consumer consumed: 12
Producer produced: 13
Producer produced: 14
Consumer consumed: 13
```

DATE - / /

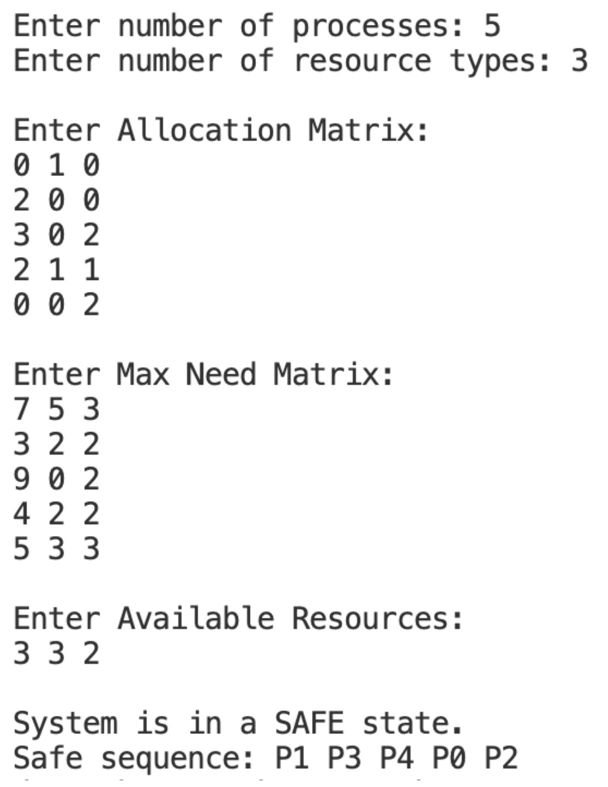
EXPERIMENT - 17

AIM - Write a program to implement Banker's algorithm for deadlock avoidance.

CODE :

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

bool isSafeState(const vector<vector<int>>& alloc,
                 const vector<vector<int>>& maxNeed,
                 vector<int> avail) {
    int n = alloc.size();        // number of processes
    int m = avail.size();        // number of resources

    vector<bool> finish(n, false);
    vector<int> safeSequence;
    vector<vector<int>> need(n, vector<int>(m));

    // Calculate Need matrix: Need = Max - Allocation
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            need[i][j] = maxNeed[i][j] - alloc[i][j];

    // Banker's algorithm
    while (safeSequence.size() < n) {
        bool allocated = false;

        for (int i = 0; i < n; i++) {
            if (!finish[i]) {
                bool canRun = true;
                for (int j = 0; j < m; j++) {
                    if (need[i][j] > avail[j]) {
                        canRun = false;
                        break;
                    }
                }

                if (canRun) {
                    // Process can run, release its resources
                    for (int j = 0; j < m; j++)
                        avail[j] += alloc[i][j];

                    safeSequence.push_back(i);
                    finish[i] = true;
                    allocated = true;
                }
            }
        }

        if (!allocated) {
            cout << "\nSystem is NOT in a safe state (Deadlock possible)\n";
            return false;
        }
    }

    cout << "\nSystem is in a SAFE state.\nSafe sequence: ";
    for (int p : safeSequence) cout << "P" << p << " ";
    cout << "\n";
    return true;
}

int main() {
    int n, m;
    cout << "Enter number of processes: ";
    cin >> n;
    cout << "Enter number of resource types: ";
    cin >> m;

    vector<vector<int>> alloc(n, vector<int>(m));
    vector<vector<int>> maxNeed(n, vector<int>(m));
    vector<int> avail(m);

    cout << "\nEnter Allocation Matrix:\n";
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            cin >> alloc[i][j];

    cout << "\nEnter Max Need Matrix:\n";
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            cin >> maxNeed[i][j];

    cout << "\nEnter Available Resources:\n";
    for (int j = 0; j < m; j++)
        cin >> avail[j];

    isSafeState(alloc, maxNeed, avail);

    return 0;
}
```

OUTPUT :



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

Enter Allocation Matrix:
0 1 0
2 0 0
3 0 2
2 1 1
0 0 2

Enter Max Need Matrix:
7 5 3
3 2 2
9 0 2
4 2 2
5 3 3

Enter Available Resources:
3 3 2

System is in a SAFE state.
Safe sequence: P1 P3 P4 P0 P2
```

DATE -   /   /

EXPERIMENT - 18

AIM - write C program to implement the various File Organization Techniques.

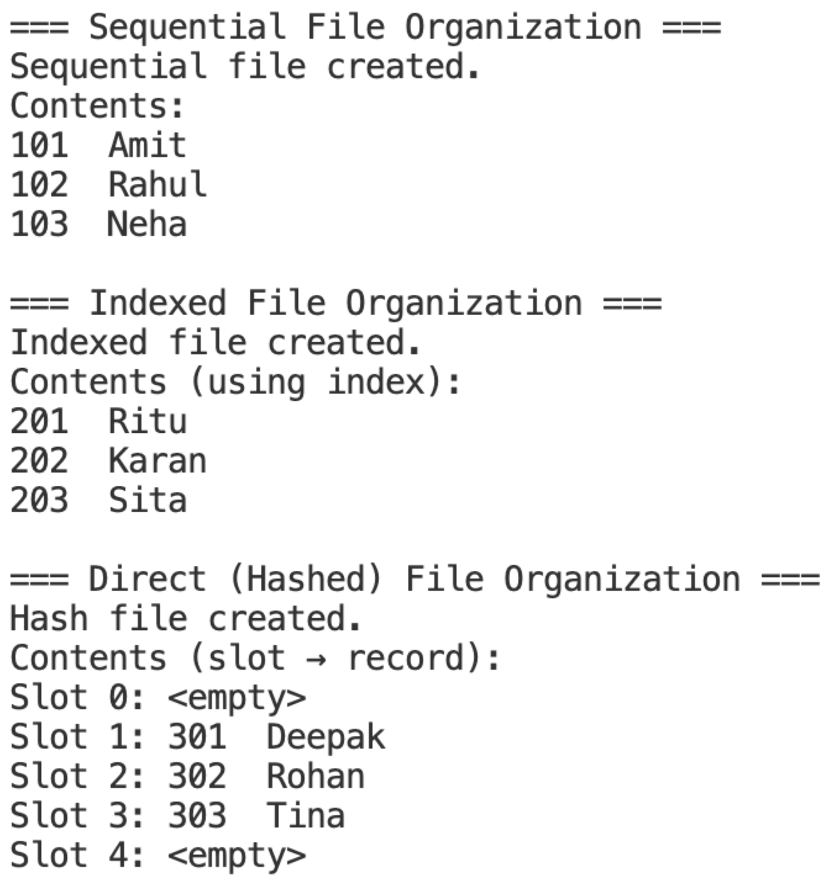
CODE :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct Record {
   int key;
   char name[20];
};

struct IndexEntry {
   int key;
   long offset;
};

int hashFunc(int key) {
   return key % 5;
}

/* ================= SEQUENTIAL FILE ORGANIZATION ================= */
void sequential_demo() {
   printf("\n=== Sequential File Organization ===\n");

   struct Record r1 = {101, "Amit"};
   struct Record r2 = {102, "Rahul"};
   struct Record r3 = {103, "Neha"};

   FILE *fp = fopen("sequential.txt", "w");
   fprintf(fp, "%d %s\n", r1.key, r1.name);
   fprintf(fp, "%d %s\n", r2.key, r2.name);
   fprintf(fp, "%d %s\n", r3.key, r3.name);
   fclose(fp);

   printf("Sequential file created.\nContents:\n");

   fp = fopen("sequential.txt", "r");
   int k; char nm[20];
   while (fscanf(fp, "%d %s", &k, nm) != EOF) {
      printf("%d  %s\n", k, nm);
   }
   fclose(fp);
}
```

```
/* ================= INDEXED FILE ORGANIZATION ================= */
void indexed_demo() {
   printf("\n=== Indexed File Organization ===\n");

   struct Record r1 = {201, "Ritu"};
   struct Record r2 = {202, "Karan"};
   struct Record r3 = {203, "Sita"};
   struct IndexEntry idx;

   FILE *df = fopen("indexed_data.bin", "wb");
   FILE *ix = fopen("indexed_index.bin", "wb");

   long pos;

   pos = ftell(df);
   fwrite(&r1, sizeof(r1), 1, df);
   idx.key = r1.key; idx.offset = pos;
   fwrite(&idx, sizeof(idx), 1, ix);

   pos = ftell(df);
   fwrite(&r2, sizeof(r2), 1, df);
   idx.key = r2.key; idx.offset = pos;
   fwrite(&idx, sizeof(idx), 1, ix);

   pos = ftell(df);
   fwrite(&r3, sizeof(r3), 1, df);
   idx.key = r3.key; idx.offset = pos;
   fwrite(&idx, sizeof(idx), 1, ix);

   fclose(df);
   fclose(ix);

   printf("Indexed file created.\nContents (using index):\n");

   ix = fopen("indexed_index.bin", "rb");
   while (fread(&idx, sizeof(idx), 1, ix)) {
      df = fopen("indexed_data.bin", "rb");
      fseek(df, idx.offset, SEEK_SET);
      struct Record r;
      fread(&r, sizeof(r), 1, df);
      printf("%d  %s\n", r.key, r.name);
      fclose(df);
   }
   fclose(ix);
}

/* ================= DIRECT (HASH) FILE ORGANIZATION ================= */
void direct_demo() {
   printf("\n=== Direct (Hashed) File Organization ===\n");
```

```
    FILE *hf = fopen("hash_file.bin", "wb");
    struct Record empty = {-1, "----"};
    for (int i = 0; i < 5; i++)
        fwrite(&empty, sizeof(empty), 1, hf);
    fclose(hf);

    struct Record r1 = {301, "Deepak"};
    struct Record r2 = {302, "Rohan"};
    struct Record r3 = {303, "Tina"};

    hf = fopen("hash_file.bin", "rb+");

    fseek(hf, hashFunc(r1.key) * sizeof(struct Record), SEEK_SET);
    fwrite(&r1, sizeof(r1), 1, hf);

    fseek(hf, hashFunc(r2.key) * sizeof(struct Record), SEEK_SET);
    fwrite(&r2, sizeof(r2), 1, hf);

    fseek(hf, hashFunc(r3.key) * sizeof(struct Record), SEEK_SET);
    fwrite(&r3, sizeof(r3), 1, hf);

    fclose(hf);

    printf("Hash file created.\nContents (slot → record):\n");

    hf = fopen("hash_file.bin", "rb");
    struct Record r;
    for (int i = 0; i < 5; i++) {
        fread(&r, sizeof(r), 1, hf);
        printf("Slot %d: ", i);
        if (r.key == -1) printf("<empty>\n");
        else printf("%d  %s\n", r.key, r.name);
    }
    fclose(hf);
}

/* ================= MAIN ================= */
int main() {
    sequential_demo();
    indexed_demo();
    direct_demo();
    return 0;
}
```

OUTPUT :



```
=== Sequential File Organization ===
Sequential file created.
Contents:
101  Amit
102  Rahul
103  Neha

=== Indexed File Organization ===
Indexed file created.
Contents (using index):
201  Ritu
202  Karan
203  Sita

=== Direct (Hashed) File Organization ===
Hash file created.
Contents (slot → record):
Slot 0: <empty>
Slot 1: 301  Deepak
Slot 2: 302  Rohan
Slot 3: 303  Tina
Slot 4: <empty>
```

DATE -   /   /

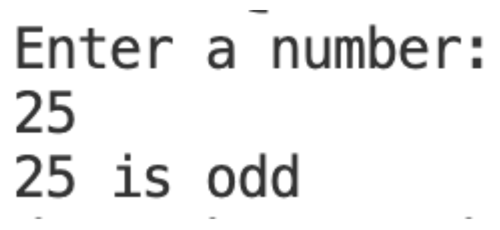
EXPERIMENT - 19

AIM - Write a script to check whether the given no. is even/odd

CODE :

```
echo "Enter a number:"
read num


if (( num % 2 == 0 ))
then
   echo "$num is Even"
else
   echo "$num is Odd"
fi
```

OUTPUT :



```
Enter a number:
25
25 is odd
```

DATE -   /   /

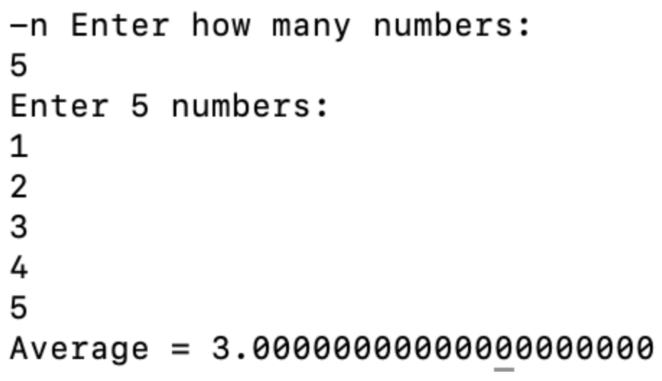
EXPERIMENT - 20

AIM - Write a script to calculate the average of n numbers

CODE :

```
echo -n "Enter how many numbers: "
read n
sum=0
echo "Enter $n numbers:"
for (( i=1; i<=n; i++ ))
do
   read num
   sum=$((sum + num))
done
avg=$(echo "$sum / $n" | bc -l)
echo "Average = $avg"
```

OUTPUT :



```
-n Enter how many numbers:
5
Enter 5 numbers:
1
2
3
4
5
Average = 3.00000000000000000000
```

DATE -   /   /

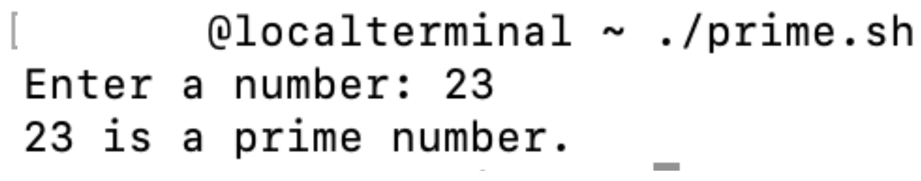
EXPERIMENT - 21

AIM - Write a script to check whether the given number is prime or not

CODE :

```
#!/bin/bash
read -p "Enter a number: " num
if [ $num -le 1 ]; then
   echo "$num is not a prime number."
   exit 0
fi
i=2
while [ $((i * i)) -le $num ]
do
   if [ $((num % i)) -eq 0 ]; then
      echo "$num is not a prime number."
      exit 0
   fi
   i=$((i + 1))
done
echo "$num is a prime number."
```

OUTPUT :



```
[        @localterminal ~ ./prime.sh
 Enter a number: 23
 23 is a prime number.
```