DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

OPERATING SYSTEMS DESIGN

CS207

LAB FILE

SUBMITTED TO:

DR. PRASHANT GIRIDHAR

SUBMITTED BY:

KUNAL BANSAL

23/CS/227

INDEX

EXPERIMENT 1

OBJECTIVE : Write a C/C++ program to simulate the following nonpreemptive CPU scheduling algorithms to find turnaround time and waiting time for a given problem. a) FCFS, b) SJF.

INTRODUCTION:
CPU scheduling is the method by which the OS decides the order in which processes will access the CPU. Non-preemptive scheduling allows a process to run to completion without interruption, making it simpler but potentially inefficient.

- **FCFS (First Come First Serve)**: Processes are scheduled in the order they arrive. This straightforward approach may cause the **Convoy Effect**, where shorter jobs wait for longer jobs, increasing average wait time.
- **SJF (Shortest Job First)**: Prioritizes jobs with the shortest burst time. This minimizes average waiting time, making it an efficient choice in scenarios where job durations are predictable. However, it may lead to **Starvation** if shorter jobs keep arriving.

ALGORITHM :

FCFS: Place processes in a queue in their arrival order. Execute each process fully before moving to the next. Calculate waiting and turnaround times based on process completion.

SJF: Sort processes by burst time, then execute in that order. Calculate waiting time by considering how long each process waits before its turn based on prior executions.

CODE :

```
//FCFS IMPLEMENTATION
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

class Process {
public:
  int pID;
  int arrivalTime;
  int burstTime;
  int completionTime;
  int turnAroundTime;
  int waitingTime;

  Process(int pID, int arrivalTime, int burstTime) {
    this->pID = pID;
```

```
        this->arrivalTime = arrivalTime;
        this->burstTime = burstTime;
        this->completionTime=0;
        this->turnAroundTime = 0;
        this->waitingTime = 0;
    }

    void display() const {
        cout << "Process ID: " << pID << ", Arrival Time: " << arrivalTime
             << ", Burst Time: " << burstTime << ", Completion Time: " << completionTime << ", Waiting Time:
" << waitingTime
             << ", Turnaround Time: " << turnAroundTime << endl;
    }
};

bool compareArrivalTime(const Process &p1, const Process &p2) {
    return p1.arrivalTime < p2.arrivalTime;
}

int main() {
    int n;
    cout << "Enter the Number of Processes for FCFS: ";
    cin >> n;

    vector<Process> processes;

    for (int i = 0; i < n; i++) {
        int pID, arrivalTime, burstTime;
        cout << "Enter pID: ";
        cin >> pID;
        cout << "Enter Arrival Time: ";
        cin >> arrivalTime;
        cout << "Enter Burst Time: ";
        cin >> burstTime;

        processes.push_back(Process(pID, arrivalTime, burstTime));
    }
```

```
sort(processes.begin(), processes.end(), compareArrivalTime);

int currentTime = 0;

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

    if (currentTime < processes[i].arrivalTime) {
        currentTime = processes[i].arrivalTime;
    }

    processes[i].waitingTime = currentTime - processes[i].arrivalTime;
    processes[i].completionTime = currentTime + processes[i].burstTime;

    currentTime += processes[i].burstTime;

    processes[i].turnAroundTime = processes[i].completionTime - processes[i].arrivalTime;



}


cout << "\nProcess Scheduling Results (FCFS):\n";
 for (size_t i = 0; i < processes.size(); i++) {
processes[i].display();
}

double avgWaitingTime = 0;
double avgTurnAroundTime = 0;

for (size_t i = 0; i < processes.size(); i++) {
avgWaitingTime += processes[i].waitingTime;
avgTurnAroundTime += processes[i].turnAroundTime;
}
avgWaitingTime /= n;
avgTurnAroundTime /= n;
```

```
    cout << "\nAverage Waiting Time: " << avgWaitingTime << endl;
    cout << "Average Turnaround Time: " << avgTurnAroundTime << endl;
    return 0;
}

//SJF IMPLEMENTATION
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
using namespace std;

class Process {
public:
    int pID;
    int arrivalTime;
    int burstTime;
    int completionTime;
    int turnAroundTime;
    int waitingTime;
    bool isCompleted;

    Process(int pID, int arrivalTime, int burstTime) {
        this->pID = pID;
        this->arrivalTime = arrivalTime;
        this->burstTime = burstTime;
        this->completionTime = 0;
        this->turnAroundTime = 0;
```

```
        this->waitingTime = 0;
        this->isCompleted = false;
    }

    void display() const {
        cout << "Process ID: " << pID << ", Arrival Time: " << arrivalTime
             << ", Burst Time: " << burstTime << ", Completion Time: " << completionTime
             << ", Waiting Time: " << waitingTime
             << ", Turnaround Time: " << turnAroundTime << endl;
    }
};

int main() {
    int n;
    cout << "Enter the Number of Processes for SJF (Non-Preemptive): ";
    cin >> n;

    vector<Process> processes;

    for (int i = 0; i < n; i++) {
        int pID, arrivalTime, burstTime;
        cout << "Enter pID: ";
        cin >> pID;
        cout << "Enter Arrival Time: ";
        cin >> arrivalTime;
        cout << "Enter Burst Time: ";
        cin >> burstTime;
```

```
    processes.push_back(Process(pID, arrivalTime, burstTime));
}

int currentTime = 0;
int completed = 0;

while (completed < n) {
    int minBurstTime = INT_MAX;
    int selectedProcess = -1;

    // Find the process with the shortest burst time among the arrived ones
    for (int i = 0; i < n; i++) {
        if (!processes[i].isCompleted && processes[i].arrivalTime <= currentTime) {
            if (processes[i].burstTime < minBurstTime) {
                minBurstTime = processes[i].burstTime;
                selectedProcess = i;
            }
            // Break ties by selecting the process with the earlier arrival time
            else if (processes[i].burstTime == minBurstTime) {
                if (processes[i].arrivalTime < processes[selectedProcess].arrivalTime) {
                    selectedProcess = i;
                }
            }
        }
    }

    if (selectedProcess == -1) {
        // No process has arrived, so advance time to the arrival time of the next process
```

```
            currentTime++;
        } else {
            // Process the selected process
            processes[selectedProcess].waitingTime = currentTime -
processes[selectedProcess].arrivalTime;
            processes[selectedProcess].completionTime = currentTime +
processes[selectedProcess].burstTime;
            processes[selectedProcess].turnAroundTime = processes[selectedProcess].completionTime -
processes[selectedProcess].arrivalTime;

            currentTime += processes[selectedProcess].burstTime;
            processes[selectedProcess].isCompleted = true;
            completed++;
        }
    }

    cout << "\nProcess Scheduling Results (SJF Non-Preemptive):\n";
    for (size_t i = 0; i < processes.size(); i++) {
    processes[i].display();
    }

    double avgWaitingTime = 0;
    double avgTurnAroundTime = 0;
    for (size_t i = 0; i < processes.size(); i++) {
    avgWaitingTime += processes[i].waitingTime;
    avgTurnAroundTime += processes[i].turnAroundTime;
    }
    avgWaitingTime /= n;
```

```
    avgTurnAroundTime /= n;


    cout << "\nAverage Waiting Time: " << avgWaitingTime << endl;
    cout << "Average Turnaround Time: " << avgTurnAroundTime << endl;


    return 0;}
```

OUTPUT:
// FCFS

```
kunalbansal@KUNALs-MacBook-Pro OS LAB % cd "/Users/kunalbansal/Desktop/MAC/OS LAB/" && g++ FCFS.cpp -o FCFS && "/Users/kunalbansal/Desktop/MAC/OS LAB/"FCFS
Enter the Number of Processes for FCFS: 4
Enter pID: 1
Enter Arrival Time: 0
Enter Burst Time: 3
Enter pID: 2
Enter Arrival Time: 2
Enter Burst Time: 1
Enter pID: 3
Enter Arrival Time: 5
Enter Burst Time: 2
Enter pID: 4
Enter Arrival Time: 11
Enter Burst Time: 1

Process Scheduling Results (FCFS):
Process ID: 1, Arrival Time: 0, Burst Time: 3, Completion Time: 3, Waiting Time: 0, Turnaround Time: 3
Process ID: 2, Arrival Time: 2, Burst Time: 1, Completion Time: 4, Waiting Time: 1, Turnaround Time: 2
Process ID: 3, Arrival Time: 5, Burst Time: 2, Completion Time: 7, Waiting Time: 0, Turnaround Time: 2
Process ID: 4, Arrival Time: 11, Burst Time: 1, Completion Time: 12, Waiting Time: 0, Turnaround Time: 1

Average Waiting Time: 0.25
Average Turnaround Time: 2
kunalbansal@KUNALs-MacBook-Pro OS LAB %
```

// SJF

```
kunalbansal@KUNALs-MacBook-Pro OS LAB % cd "/Users/kunalbansal/Desktop/MAC/OS LAB/" && g++ SJF.cpp -o SJF && "/Users/kunalbansal/Desktop/MAC/OS LAB/"SJF
Enter the Number of Processes for SJF (Non-Preemptive): 4
Enter pID: 1
Enter Arrival Time: 0
Enter Burst Time: 2
Enter pID: 2
Enter Arrival Time: 0
Enter Burst Time: 1
Enter pID: 3
Enter Arrival Time: 3
Enter Burst Time: 5
Enter pID: 4
Enter Arrival Time: 1
Enter Burst Time: 5

Process Scheduling Results (SJF Non-Preemptive):
Process ID: 1, Arrival Time: 0, Burst Time: 2, Completion Time: 3, Waiting Time: 1, Turnaround Time: 3
Process ID: 2, Arrival Time: 0, Burst Time: 1, Completion Time: 1, Waiting Time: 0, Turnaround Time: 1
Process ID: 3, Arrival Time: 3, Burst Time: 5, Completion Time: 13, Waiting Time: 5, Turnaround Time: 10
Process ID: 4, Arrival Time: 1, Burst Time: 5, Completion Time: 8, Waiting Time: 2, Turnaround Time: 7

Average Waiting Time: 2
Average Turnaround Time: 5.25
kunalbansal@KUNALs-MacBook-Pro OS LAB %
```

LEARNING OUTCOME:
The experiment provided detailed understanding of the preemptive CPU scheduling algorithms: Round Robin, and Priority base Scheduling, and helped in analyzing their impact on system efficiency.

EXPERIMENT 2

OBJECTIVE : Write a C/C++ program to simulate the following Preemptive CPU scheduling algorithms to find turnaround time and waiting time for a given problem.
a) Round Robin, b) Priority.

INTRODUCTION:
Preemptive scheduling algorithms can interrupt running processes to allow a more suitable process to execute, enhancing system responsiveness.

- **Round Robin**: Each process receives a small time quantum. If the process doesn't finish within its time slice, it is moved to the end of the queue, promoting **fairness** and responsiveness.
- **Priority Scheduling**: Processes are assigned priorities, with higher priority processes preempting lower ones. This can lead to **Starvation** for low-priority processes unless mechanisms like **Aging** are used to increase their priority over time.

ALGORITHM :

- **Round Robin**: Assign a time quantum. Execute each process for this quantum, then move it to the end if unfinished. Repeat until all processes are complete, calculating waiting and turnaround times as each process completes.
- **Priority Scheduling**: Sort processes by priority. Execute the highest priority process first. If a new process arrives with a higher priority, it preempts the running process.

CODE :

```
//Round Robin Implementation
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

class Process {
public:
  int pID;
```

```
    int arrivalTime;
    int burstTime;
    int remainingTime;
    int completionTime;
    int waitingTime;
    int turnAroundTime;

    Process(int pID, int arrivalTime, int burstTime) {
        this->pID = pID;
        this->arrivalTime = arrivalTime;
        this->burstTime = burstTime;
        this->remainingTime = burstTime;
        this->completionTime = 0;
        this->waitingTime = 0;
        this->turnAroundTime = 0;
    }
    void display() const {
        cout << "Process ID: " << pID << ", Arrival Time: " << arrivalTime
             << ", Burst Time: " << burstTime << ", Completion Time: " << completionTime
             << ", Waiting Time: " << waitingTime << ", Turnaround Time: " << turnAroundTime << endl;
    }
};

int main() {
    int n, timeQuantum;
    cout << "Enter the Number of Processes for Round Robin: ";
    cin >> n;

    cout << "Enter Time Quantum: ";
    cin >> timeQuantum;

    vector<Process> processes;


    for (int i = 0; i < n; i++) {
        int pID, arrivalTime, burstTime;
        cout << "Enter pID: ";
```

```
    cin >> pID;
    cout << "Enter Arrival Time: ";
    cin >> arrivalTime;
    cout << "Enter Burst Time: ";
    cin >> burstTime;

    processes.push_back(Process(pID, arrivalTime, burstTime));
}

queue<int> readyQueue;
int currentTime = 0;
int completed = 0;
vector<bool> isInQueue(n, false);
int totalWaitingTime = 0, totalTurnaroundTime = 0;

for (int i = 0; i < n; i++) {
    if (processes[i].arrivalTime <= currentTime && !isInQueue[i]) {
        readyQueue.push(i);
        isInQueue[i] = true;
    }
}

while (completed < n) {
    if (!readyQueue.empty()) {
        int idx = readyQueue.front();
        readyQueue.pop();
        // Process this process for either the time quantum or its remaining time
        int timeToExecute = min(timeQuantum, processes[idx].remainingTime);
        processes[idx].remainingTime -= timeToExecute;
        currentTime += timeToExecute;
        // Check if the process is completed
        if (processes[idx].remainingTime == 0) {
            completed++;
            processes[idx].completionTime = currentTime;
            processes[idx].turnAroundTime = processes[idx].completionTime - processes[idx].arrivalTime;
            processes[idx].waitingTime = processes[idx].turnAroundTime - processes[idx].burstTime;
            totalWaitingTime += processes[idx].waitingTime;
```

```
            totalTurnaroundTime += processes[idx].turnAroundTime;
        }
        // Add new processes that have arrived during the current time to the ready queue
        for (int i = 0; i < n; i++) {
            if (processes[i].arrivalTime <= currentTime && !isInQueue[i] && processes[i].remainingTime >
0) {
                readyQueue.push(i);
                isInQueue[i] = true;
            }
        }
        // If the process is not finished, push it back to the ready queue
        if (processes[idx].remainingTime > 0) {
            readyQueue.push(idx);
        }

    } else {
        // If the ready queue is empty, increment the current time until a process arrives
        currentTime++;
    }
  }

  cout << "\nProcess Scheduling Results (Round Robin):\n";
  for (size_t i = 0; i < processes.size(); i++) {
  processes[i].display();
  }

  double avgWaitingTime = (double)totalWaitingTime / n;
  double avgTurnAroundTime = (double)totalTurnaroundTime / n;

  cout << "\nAverage Waiting Time: " << avgWaitingTime << endl;
  cout << "Average Turnaround Time: " << avgTurnAroundTime << endl;

  return 0;
}

//Priority
#include <iostream>
```

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

class Process {
public:
  int pID;
  int arrivalTime;
  int burstTime;
  int priority;
  int completionTime;
  int waitingTime;
  int turnAroundTime;
  int remainingBurstTime;

  Process(int pID, int arrivalTime, int burstTime, int priority)
    : pID(pID), arrivalTime(arrivalTime), burstTime(burstTime), priority(priority) {
    completionTime = waitingTime = turnAroundTime = 0;
    remainingBurstTime = burstTime;
  }

  void display() const {
    cout << "Process ID: " << pID << ", Arrival Time: " << arrivalTime
        << ", Burst Time: " << burstTime << ", Priority: " << priority
        << ", Completion Time: " << completionTime << ", Waiting Time: " << waitingTime
        << ", Turnaround Time: " << turnAroundTime << endl;
  }
};

// Comparator for priority queue (higher priority, lower arrival time first)
struct compare {
  bool operator()(const Process* a, const Process* b) {
    if (a->priority == b->priority)
      return a->arrivalTime > b->arrivalTime;
    return a->priority > b->priority;
  }
```

```
};

int main() {
    int n;
    cout << "Enter the Number of Processes for Preemptive Priority Scheduling: ";
    cin >> n;

    vector<Process> processes;

    for (int i = 0; i < n; i++) {
        int pID, arrivalTime, burstTime, priority;
        cout << "Enter pID: ";
        cin >> pID;
        cout << "Enter Arrival Time: ";
        cin >> arrivalTime;
        cout << "Enter Burst Time: ";
        cin >> burstTime;
        cout << "Enter Priority (Lower value means higher priority): ";
        cin >> priority;

        processes.push_back(Process(pID, arrivalTime, burstTime, priority));
    }

    priority_queue<Process*, vector<Process*>, compare> pq;
    int currentTime = 0, totalWaitingTime = 0, totalTurnaroundTime = 0;
    int completed = 0;

    // Main loop: increment time until all processes are complete
    while (completed < n) {
        // Add all arriving processes at current time to the queue
        for (auto &proc : processes) {
            if (proc.arrivalTime == currentTime) {
                pq.push(&proc);
            }
        }

        // If queue is not empty, process the highest priority process
```

```
    if (!pq.empty()) {
        Process* currentProcess = pq.top();
        pq.pop();
        // Execute process for 1 unit time
        currentProcess->remainingBurstTime--;
        currentTime++;
        // If process is completed
        if (currentProcess->remainingBurstTime == 0) {
            completed++;
            currentProcess->completionTime = currentTime;
            currentProcess->turnAroundTime = currentProcess->completionTime -
currentProcess->arrivalTime;
            currentProcess->waitingTime = currentProcess->turnAroundTime -
currentProcess->burstTime;

            totalWaitingTime += currentProcess->waitingTime;
            totalTurnaroundTime += currentProcess->turnAroundTime;
        } else {
            // If not completed, re-add to queue
            pq.push(currentProcess);
        }
    } else {
        currentTime++;  // Increment time if no process is available
    }
  }

  cout << "\nProcess Scheduling Results (Priority Preemptive):\n";
  for (size_t i = 0; i < processes.size(); i++) {
  processes[i].display();
  }

  double avgWaitingTime = (double)totalWaitingTime / n;
  double avgTurnAroundTime = (double)totalTurnaroundTime / n;

  cout << "\nAverage Waiting Time: " << avgWaitingTime << endl;
  cout << "Average Turnaround Time: " << avgTurnAroundTime << endl;

  return 0;
```

}

OUTPUT :

//ROUND ROBIN

```
Enter the Number of Processes for Round Robin: 5
Enter Time Quantum: 2
Enter pID: 1
Enter Arrival Time: 0
Enter Burst Time: 5
Enter pID: 2
Enter Arrival Time: 1
Enter Burst Time: 3
Enter pID: 3
Enter Arrival Time: 2
Enter Burst Time: 1
Enter pID: 4
Enter Arrival Time: 3
Enter Burst Time: 2
Enter pID: 5
Enter Arrival Time: 4
Enter Burst Time: 3

Process Scheduling Results (Round Robin):
Process ID: 1, Arrival Time: 0, Burst Time: 5, Completion Time: 13, Waiting Time: 8, Turnaround Time: 13
Process ID: 2, Arrival Time: 1, Burst Time: 3, Completion Time: 12, Waiting Time: 8, Turnaround Time: 11
Process ID: 3, Arrival Time: 2, Burst Time: 1, Completion Time: 5, Waiting Time: 2, Turnaround Time: 3
Process ID: 4, Arrival Time: 3, Burst Time: 2, Completion Time: 9, Waiting Time: 4, Turnaround Time: 6
Process ID: 5, Arrival Time: 4, Burst Time: 3, Completion Time: 14, Waiting Time: 7, Turnaround Time: 10

Average Waiting Time: 5.8
Average Turnaround Time: 8.6
kunalbansal@KUNALs-MacBook-Pro OS LAB %
```

// PRIORITY (PRE-EMPTIVE)

```
kunalbansal@KUNALs-MacBook-Pro OS LAB % cd "/Users/kunalbansal/Desktop/MAC/OS LAB/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Use
rs/kunalbansal/Desktop/MAC/OS LAB/"tempCodeRunnerFile
tempCodeRunnerFile.cpp:69:14: warning: 'auto' type specifier is a C++11 extension [-Wc++11-extensions]
        for (auto &proc : processes) {
             ^
tempCodeRunnerFile.cpp:69:25: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
        for (auto &proc : processes) {
                        ^
2 warnings generated.
Enter the Number of Processes for Preemptive Priority Scheduling: 5
Enter pID: 1
Enter Arrival Time: 0
Enter Burst Time: 3
Enter Priority (Lower value means higher priority): 3
Enter pID: 2
Enter Arrival Time: 1
Enter Burst Time: 4
Enter Priority (Lower value means higher priority): 2
Enter pID: 3
Enter Arrival Time: 2
Enter Burst Time: 6
Enter Priority (Lower value means higher priority): 4
Enter pID: 4
Enter Arrival Time: 3
Enter Burst Time: 4
Enter Priority (Lower value means higher priority): 6
Enter pID: 5
Enter Arrival Time: 5
Enter Burst Time: 2
Enter Priority (Lower value means higher priority): 10

Process Scheduling Results (Priority Preemptive):
Process ID: 1, Arrival Time: 0, Burst Time: 3, Priority: 3, Completion Time: 7, Waiting Time: 4, Turnaround Time: 7
Process ID: 2, Arrival Time: 1, Burst Time: 4, Priority: 2, Completion Time: 5, Waiting Time: 0, Turnaround Time: 4
Process ID: 3, Arrival Time: 2, Burst Time: 6, Priority: 4, Completion Time: 13, Waiting Time: 5, Turnaround Time: 11
Process ID: 4, Arrival Time: 3, Burst Time: 4, Priority: 6, Completion Time: 17, Waiting Time: 10, Turnaround Time: 14
Process ID: 5, Arrival Time: 5, Burst Time: 2, Priority: 10, Completion Time: 19, Waiting Time: 12, Turnaround Time: 14

Average Waiting Time: 6.2
Average Turnaround Time: 10
kunalbansal@KUNALs-MacBook-Pro OS LAB %
```

LEARNING OUTCOME

: The experiment provided detailed understanding of the preemptive CPU scheduling algorithms: Round Robin, and Priority base Scheduling, and helped in analyzing their impact on system efficiency

EXPERIMENT 3

OBJECTIVE :Write a C/C++ program to simulate the following contiguous memory allocation techniques a) Worst fit, b) Best fit, c) First fit

INTRODUCTION:

Memory allocation strategies optimize how memory is assigned to processes, aiming to use memory efficiently and minimize **fragmentation**.

- **Worst Fit**: Allocates the largest available block, leaving larger chunks of free space. It minimizes **external fragmentation** by preserving larger blocks for future processes but can leave unusable small spaces.
- **Best Fit**: Allocates the smallest block that fits, minimizing wasted space within blocks but possibly increasing fragmentation as it leaves tiny unusable gaps.
- **First Fit**: Allocates the first available block that is large enough. This approach is quick but can lead to significant fragmentation.

ALGORITHM :

Worst Fit: Search for the largest memory block that can accommodate the process. Allocate memory from this block and update free space.

Best Fit: Find the smallest block that can fit the process and allocate memory from it.

First Fit: Scan memory from the beginning, allocate the first fitting block, and stop searching after allocation.

CODE :

```
//Worst-Fit
#include <iostream>
#include <vector>
using namespace std;

class process{
  public:
     int process_id;
     int memory_required;
};
```

```
class block{
  public:
    int block_id;
    int memory;
};

void input_processes(int n, process arr[]){
  for(int i = 0; i < n; i++){
    cout<<"Enter memory_required in process "<<i+1<<": "; cin>>arr[i].memory_required;
    arr[i].process_id = i + 1;
  }
};

void input_memory_blocks(int m, block arr[]){
  for(int i = 0; i < m; i++){
    cout<<"Enter memory in block "<<i+1<<": "; cin>>arr[i].memory;
    arr[i].block_id = i + 1;
  }
};

bool compare(block &b1, block &b2){
  if(b1.memory==b2.memory) return b1.block_id < b2.block_id;
  return b1.memory > b2.memory;
}

int main(){
  int n; cout<<"Enter number of processes: "; cin>>n;
  process arr[n];
  input_processes(n, arr);
  int m; cout<<"Enter number of memory blocks: "; cin>>m;
  block arr2[m];
  input_memory_blocks(m, arr2);
  sort(arr2, arr2 + m, compare);
  int i = 0;
  cout<<"Process_ID\tBlock_ID\tMemory_Used\tMemory_wasted\n";
  for(int j = 0; j < n; j++){
    if(i < m && arr2[i].memory >= arr[j].memory_required){
```

```
            cout<<arr[j].process_id<<"\t\t"<<arr2[i].block_id<<"\t\t"<<
            arr[j].memory_required<<"\t\t"<<arr2[i].memory - arr[j].memory_required<<endl;
            i++;
        }
        else{
            cout<<arr[j].process_id<<"\t\tNA\t\tNA\t\tNA\n";
        }
    }
}
//Best Fit
#include <iostream>
#include <vector>
#include <limits.h>
using namespace std;

class process{
    public:
        int process_id;
        int memory_required;
};

class block{
    public:
        int block_id;
        int memory;
        bool free;
};

void input_processes(int n, process arr[]){
    for(int i = 0; i < n; i++){
        cout<<"Enter memory_required in process "<<i+1<<": "; cin>>arr[i].memory_required;
        arr[i].process_id = i + 1;
    }
};

void input_memory_blocks(int m, block arr[]){
    for(int i = 0; i < m; i++){
```

```
        cout<<"Enter memory in block "<<i+1<<": "; cin>>arr[i].memory;
        arr[i].block_id = i + 1;
        arr[i].free = true;
    }
};

bool compare(block &b1, block &b2){
    if(b1.memory==b2.memory) return b1.block_id < b2.block_id;
    return b1.memory > b2.memory;
}

const int inf = (1<<30);

int main(){
    int n; cout<<"Enter number of processes: "; cin>>n;
    process arr[n];
    input_processes(n, arr);
    int m; cout<<"Enter number of memory blocks: "; cin>>m;
    block arr2[m];
    input_memory_blocks(m, arr2);
    cout<<"Process_ID\tBlock_ID\tMemory_Used\tMemory_wasted\n";
    for(int j = 0; j < n; j++){
        int best_memory = inf, k = -1;
        for(int i = 0; i < m; i++){
            if(arr2[i].free && arr2[i].memory >= arr[j].memory_required){
                if(best_memory > arr2[i].memory){
                    best_memory = arr2[i].memory;
                    k = i;
                }
            }
        }
        if(k==-1){
            cout<<arr[j].process_id<<"\t\tNA\t\tNA\t\tNA\n";
        }
        else{
            cout<<arr[j].process_id<<"\t\t"<<arr2[k].block_id<<"\t\t"<<
            arr[j].memory_required<<"\t\t"<<arr2[k].memory - arr[j].memory_required<<endl;
```

```
            arr2[k].free = false;
        }
    }
}
//First Fir
#include <iostream>
#include <vector>
using namespace std;

class process{
    public:
        int process_id;
        int memory_required;
};

class block{
    public:
        int block_id;
        int memory;
        bool free;
};

void input_processes(int n, process arr[]){
    for(int i = 0; i < n; i++){
        cout<<"Enter memory_required in process "<<i+1<<": "; cin>>arr[i].memory_required;
        arr[i].process_id = i + 1;
    }
};

void input_memory_blocks(int m, block arr[]){
    for(int i = 0; i < m; i++){
        cout<<"Enter memory in block "<<i+1<<": "; cin>>arr[i].memory;
        arr[i].block_id = i + 1;
        arr[i].free = true;
    }
};
```

```
bool compare(block &b1, block &b2){
  if(b1.memory==b2.memory) return b1.block_id < b2.block_id;
  return b1.memory > b2.memory;
}

int main(){
  int n; cout<<"Enter number of processes: "; cin>>n;
  process arr[n];
  input_processes(n, arr);
  int m; cout<<"Enter number of memory blocks: "; cin>>m;
  block arr2[m];
  input_memory_blocks(m, arr2);
  cout<<"Process_ID\tBlock_ID\tMemory_Used\tMemory_wasted\n";
  for(int j = 0; j < n; j++){
    bool found = false;
    for(int i = 0; i < m; i++){
      if(arr2[i].free && arr2[i].memory >= arr[j].memory_required){
        found = true;
        cout<<arr[j].process_id<<"\t\t"<<arr2[i].block_id<<"\t\t"<<
        arr[j].memory_required<<"\t\t"<<arr2[i].memory - arr[j].memory_required<<endl;
        arr2[i].free = false;
        break;
      }
    }
    if(found==false){
      cout<<arr[j].process_id<<"\t\tNA\t\tNA\t\tNA\n";
    }
  }
}
```

OUTPUT:

WORST FIT

```
kunalbansal@KUNALs-MacBook-Pro OS LAB % cd "/Users/kunalbansal/Desktop/MAC/OS LAB/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Use
rs/kunalbansal/Desktop/MAC/OS LAB/"tempCodeRunnerFile
Enter number of processes: 3
Enter memory_required in process 1: 10
Enter memory_required in process 2: 15
Enter memory_required in process 3: 2
Enter number of memory blocks: 4
Enter memory in block 1: 100
Enter memory in block 2: 23
Enter memory in block 3: 2
Enter memory in block 4: 50
Process_ID      Block_ID        Memory_Used     Memory_wasted
1               1               10              90
2               4               15              35
3               2               2               21
```

BEST FIT

```
kunalbansal@KUNALs-MacBook-Pro OS LAB % cd "/Users/kunalbansal/Desktop/MAC/OS LAB/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Use
rs/kunalbansal/Desktop/MAC/OS LAB/"tempCodeRunnerFile
Enter number of processes: 3
Enter memory_required in process 1: 10
Enter memory_required in process 2: 12
Enter memory_required in process 3: 14
Enter number of memory blocks: 5
Enter memory in block 1: 100
Enter memory in block 2: 12
Enter memory in block 3: 14
Enter memory in block 4: 2
Enter memory in block 5: 13
Process_ID      Block_ID        Memory_Used     Memory_wasted
1               1               10              90
2               2               12              0
3               3               14              0
kunalbansal@KUNALs-MacBook-Pro OS LAB %
```

FIRST FIT

```
kunalbansal@KUNALs-MacBook-Pro OS LAB % cd "/Users/kunalbansal/Desktop/MAC/OS LAB/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile && "/Use
rs/kunalbansal/Desktop/MAC/OS LAB/"tempCodeRunnerFile
Enter number of processes: 3
Enter memory_required in process 1: 10
Enter memory_required in process 2: 14
Enter memory_required in process 3: 1
Enter number of memory blocks: 4
Enter memory in block 1: 15
Enter memory in block 2: 16
Enter memory in block 3: 70
Enter memory in block 4: 1
Process_ID      Block_ID        Memory_Used     Memory_wasted
1               1               10              5
2               2               14              2
3               4               1               0
kunalbansal@KUNALs-MacBook-Pro OS LAB %
```

LEARNING OUTCOME:
The experiment provided detailed understanding of the contiguous memory allocation techniques: Worst Fit, Best Fit, and First Fit, and helped in analyzing their impact on system efficiency.

EXPERIMENT 4

OBJECTIVE : Write a C/C++ program to simulate the following file allocation strategies. a) Sequential, b) Indexed

INTRODUCTION:

File allocation strategies determine how a file's data blocks are stored on disk. Efficient allocation improves performance and reduces disk fragmentation.

- **Sequential Allocation**: Stores files in contiguous disk blocks, allowing for fast sequential access. However, it can cause **external fragmentation** and limit file expansion.
- **Indexed Allocation**: Uses an index block to keep pointers to each data block of a file, allowing for non-contiguous allocation, making it easier to grow files.

ALGORITHM :

Sequential: Allocate contiguous blocks on disk for the file, record the start block, and read blocks sequentially.

Indexed: Create an index block with pointers to each file block. Each pointer in the index block points to a non-contiguous disk block holding file data.

CODE :

```
#include <iostream>
#include <map>
#include <vector>

using namespace std;

int block_size;
const int disk_size = 20;
int empty_blocks = 0;
int num_files = 0;
map<string, pair<int, int> > directory;
vector<bool> disk_avail(disk_size, false);
vector<pair<string, int> > disk(disk_size);

class file{
  public:
  string file_name;
  int file_size;
  int partitions;

  file() {};
  file(string &fname, int fsize) : file_name(fname), file_size(fsize) {
     partitions = (file_size + block_size - 1)/block_size;
  };
};

void initialize(){
  cout<<"Enter block size: "; cin>>block_size;
  cout<<"Enter number of empty blocks: "; cin>>empty_blocks;
  cout<<"Enter indices of empty blocks: ";
  for(int i = 0; i < empty_blocks; i++){
     int x; cin>>x;
     disk_avail[x] = true;
  }
}
```

```
void sequential_file_allocation(){
  cout<<"Enter number of files: "; cin>>num_files;
  for(int i = 0; i < num_files; i++){
    string fname; int fsize;
    cout<<"Enter file name: "; cin>>fname;
    cout<<"Enter file size: "; cin>>fsize;
    file f(fname, fsize);
    bool avail = false; int count = 0, startblock = -1;
    for(int j = 0; j < disk_size; j++){
      if(disk_avail[j]) count++;
      else count = 0;
      if(count == 1) startblock = j;
      if(count == f.partitions){
        avail = true; break;
      }
    }
    if(avail){
      directory[f.file_name] = make_pair(startblock, f.file_size);
      int count = 1;
      for(int j = startblock; count <= f.partitions; j++, count++){
        disk_avail[j] = false;
        disk[j] = make_pair(f.file_name, count);
      }
      cout<<"File "<<f.file_name<<" has been allocated successfully"<<endl;
    }
    else{
      cout<<"File "<<f.file_name<<" could not be allocated"<<endl;
    }
  }
}

void display_directory(){
  for(pair<string, pair<int, int> > entry: directory){
    string fname = entry.first; int startblock = entry.second.first;
    int fsize = entry.second.second;
    int endblock = startblock + (fsize + block_size - 1)/block_size - 1;
    cout<<"File Name: "<<fname<<endl;
```

```
      cout<<"File Size: "<<fsize<<endl;
      cout<<"Memory Block\tFile\t\tPartition"<<endl;
      for(int j = startblock; j <= endblock; j++){
         cout<<j<<"\t\t"<<disk[j].first<<"\t\t"<<disk[j].second<<endl;
      }
      cout<<endl;
   }
}

int main(){
   initialize();
   sequential_file_allocation();
   display_directory();
}
```

//INDEXED

```
#include<iostream>
#include<vector>
#include<map>

using namespace std;

int block_size;
const int disk_size = 20;
int empty_blocks = 0;
int num_files = 0;
map<string, pair<int, int> > directory;
vector<bool> disk_avail(disk_size, false);
vector<pair<string, int> > disk(disk_size);
vector<vector<int> > index_table;

class file{
   public:
   string file_name;
   int file_size;
   int partitions;
```

```
    file() {};
    file(string &fname, int fsize) : file_name(fname), file_size(fsize) {
        partitions = (file_size + block_size - 1)/block_size;
    };
};

void initialize(){
    cout<<"Enter block size: "; cin>>block_size;
    cout<<"Enter number of empty blocks: "; cin>>empty_blocks;
    cout<<"Enter indices of empty blocks: ";
    for(int i = 0; i < empty_blocks; i++){
        int x; cin>>x;
        disk_avail[x] = true;
    }
}

void indexed_file_allocation(){
    cout<<"Enter number of files: "; cin>>num_files;
    for(int i = 0; i < num_files; i++){
        string fname; int fsize;
        cout<<"Enter file name: "; cin>>fname;
        cout<<"Enter file size: "; cin>>fsize;
        file f(fname, fsize);
        int avail = 0;
        for(int j = 0; j < disk_size; j++){
            if(disk_avail[j]) avail++;
            if(avail >= f.partitions) break;
        }
        if(avail >= f.partitions){
            directory[f.file_name] = make_pair(index_table.size(), f.file_size);
            vector<int> index_table_entry; int count = 0;
            for(int j = 0; count < f.partitions; j++){
                if(disk_avail[j]){
                    disk_avail[j] = false;
                    disk[j] = make_pair(f.file_name, count + 1);
                    index_table_entry.push_back(j);
                    count++;
```

```
            }
        }
        index_table.push_back(index_table_entry);
        cout<<"File "<<f.file_name<<" has been allocated successfully"<<endl;
    }
    else{
        cout<<"File "<<f.file_name<<" could not be allocated"<<endl;
    }
  }
}

void display_directory(){
  for(pair<string, pair<int, int> > entry: directory){
    string fname = entry.first; int file_index = entry.second.first;
    int fsize = entry.second.second;
    cout<<"File Name: "<<fname<<endl;
    cout<<"File Size: "<<fsize<<endl;
    cout<<"Memory Block\tFile\t\tPartition"<<endl;
    int fpartitions = index_table[file_index].size();
    for(int j = 0; j < fpartitions; j++){
      int block = index_table[file_index][j];
      cout<<block<<"\t\t"<<disk[block].first<<"\t\t"<<disk[block].second<<endl;
    }
    cout<<endl;
  }
}

int main(){
  initialize();
  indexed_file_allocation();
  display_directory();
}
```

OUTPUT:

SEQUENTIAL

```
kunalbansal@KUNALs-MacBook-Pro OS LAB % cd "/Users/kunalbansal/Desktop/MAC/OS LAB/" && g++ sequential.cpp -o sequential && "/Users/kunalbansal/Desktop/MAC/OS LAB/"sequential
sequential.cpp:69:44: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
    for(pair<string, pair<int, int> > entry: directory){
                                           ^
1 warning generated.
Enter block size: 4
Enter number of empty blocks: 10
Enter indices of empty blocks: 0 1 2 3 4 5 6 7 8 9
Enter number of files: 2
Enter file name: kunal.txt
Enter file size: 32
File kunal.txt has been allocated successfully
Enter file name: bansal.mp3
Enter file size: 9
File bansal.mp3 could not be allocated
File Name: kunal.txt
File Size: 32
Memory Block    File            Partition
0               kunal.txt               1
1               kunal.txt               2
2               kunal.txt               3
3               kunal.txt               4
4               kunal.txt               5
5               kunal.txt               6
6               kunal.txt               7
7               kunal.txt               8

kunalbansal@KUNALs-MacBook-Pro OS LAB %
```

INDEXED :

```
rs/kunalbansal/Desktop/MAC/OS LAB/"indexed_allocation
indexed_allocation.cpp:71:44: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
    for(pair<string, pair<int, int> > entry: directory){
                                           ^
1 warning generated.
Enter block size: 4
Enter number of empty blocks: 10
Enter indices of empty blocks: 0 1 2 3 4 5 6 7 8 9
Enter number of files: 2
Enter file name: kunal.txt
Enter file size: 38
File kunal.txt has been allocated successfully
Enter file name: 2
Enter file size: 2
File 2 could not be allocated
File Name: kunal.txt
File Size: 38
Memory Block    File            Partition
0               kunal.txt               1
1               kunal.txt               2
2               kunal.txt               3
3               kunal.txt               4
4               kunal.txt               5
5               kunal.txt               6
6               kunal.txt               7
7               kunal.txt               8
8               kunal.txt               9
9               kunal.txt               10

kunalbansal@KUNALs-MacBook-Pro OS LAB %
```

LEARNING OUTCOME:

The experiment provided detailed understanding of the file allocation strategies: Sequential, and Indexed, and helped in analyzing their impact on system efficiency.

EXPERIMENT 5

OBJECTIVE : Write a C/C++ program to simulate Banker's algorithm for the purpose of Deadlock avoidance.

INTRODUCTION:

- Deadlock occurs when processes hold resources while waiting for others, preventing any from proceeding. Banker's algorithm prevents deadlock by only allocating resources if the system remains in a **safe state**.

ALGORITHM :

When a process requests resources, check if the resources are available. Simulate granting the resources and determine if the system will remain in a safe state (where all processes can complete eventually).

If granting the request results in an unsafe state, deny the allocation. Otherwise, proceed with the allocation.

CODE:

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

void initialize(int &n, int &m, vector<int> &available,
vector<vector<int> > &allocation, vector<vector<int> > &max, vector<vector<int> > &need){
   cout<<"Enter number of processes"; cin>>n;
   cout<<"Enter number of resource types"; cin>>m;
   allocation.assign(n, vector<int>(m, 0));
   max.assign(n, vector<int>(m, 0));
   need.assign(n, vector<int>(m, 0));
   available.assign(m, 0);
   cout<<"Enter number of available resources of each type:"<<endl;
   for(int i = 0; i < m; i++) cin>>available[i];
   for(int i = 0; i < n; i++){
     cout<<"Enter maximum demand of each resource type for process "<<i+1<<":"<<endl;
     for(int j = 0; j < m; j++) cin>>max[i][j];
   }
   for(int i = 0; i < n; i++){
     cout<<"Enter number of allocated resources of each resource type for process "<<i+1<<":"<<endl;
     for(int j = 0; j < m; j++) cin>>allocation[i][j];
   }
   for(int i = 0; i < n; i++){
```

```
      for(int j = 0; j < m; j++){
         need[i][j] = max[i][j] - allocation[i][j];
      }
   }
}

bool is_less_equal(vector<int> &x, vector<int> &y){
   int n = x.size();
   for(int i = 0; i < n; i++){
      if(x[i] > y[i]) return false;
   }
   return true;
}

void add(vector<int> &x, vector<int> &y){
   int n = y.size();
   for(int i = 0; i < n; i++){
      x[i] += y[i];
   }
}

bool bankers_algorithm(int &n, int &m, vector<int> &available,
vector<vector<int> > &allocation, vector<vector<int> > &max, vector<vector<int> > &need){
   vector<int> work(available.begin(), available.end());
   vector<int> finish(n, false);
   vector<int> safe_sequence;
   bool found = false;
   do{
      found = false;
      for(int i = 0; i < n; i++){
         if(finish[i]==false){
            if(is_less_equal(need[i], work)){
               add(work, allocation[i]);
               safe_sequence.push_back(i+1);
               finish[i] = true;
               found = true;
            }
```

```
        }
      }
    } while(found);
    bool all_finished = true;
    for(int i = 0; i < n; i++){
      if(finish[i]==false) all_finished = false;
    }
    if(all_finished){
      cout<<"Possible Safe Sequence for given set of process and resources:"<<endl;
      for(int i = 0; i < n; i++) cout<<safe_sequence[i]<<" ";
      cout<<endl;
      return true;
    }
    else return false;
}
int main(){
    int n, m;
    vector<int> available;
    vector<vector<int> > allocation, max, need;
    initialize(n, m, available, allocation, max, need);
    bool is_safe = bankers_algorithm(n, m, available, allocation, max, need);
    if(is_safe) cout<<"The system is in safe state, and deadlock has been avoided"<<endl;
    else cout<<"The system is in unsafe state, and deadlock can not be avoided"<<endl;
}
```

OUTPUT:

```
kunalbansal@KUNALs-MacBook-Pro OS LAB % cd "/Users/kunalbansal/Desktop/MAC/OS LAB/" && g++ bankers_avoidance.cpp -o bankers_avoidance &&
"/Users/kunalbansal/Desktop/MAC/OS LAB/"bankers_avoidance
Enter number of processes3
Enter number of resource types3
Enter number of available resources of each type:
2 3 4
Enter maximum demand of each resource type for process 1:
1 1 1
Enter maximum demand of each resource type for process 2:
1 2 2
Enter maximum demand of each resource type for process 3:
0 0 1
Enter number of allocated resources of each resource type for process 1:
0 0 0
Enter number of allocated resources of each resource type for process 2:
0 0 1
Enter number of allocated resources of each resource type for process 3:
0 0 0
Possible Safe Sequence for given set of process and resources:
1 2 3
The system is in safe state, and deadlock has been avoided
kunalbansal@KUNALs-MacBook-Pro OS LAB %
```

LEARNING OUTCOME:
The experiment provided detailed understanding of the Banker's algorithm for the purpose of deadlock avoidance and helped in analyzing its impact on system efficiency

EXPERIMENT 6

OBJECTIVE : Write a C/C++ program to simulate Banker's algorithm for the purpose of Deadlock prevention.

INTRODUCTION:

- : Prevents deadlock by ensuring at least one of the necessary conditions (mutual exclusion, hold and wait, no preemption, circular wait) is never true.

ALGORITHM :

- **Algorithm Explanation**: Banker's algorithm can also be used to impose restrictions that prevent deadlock by only granting safe requests or imposing priority constraints.

CODE:

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

void setup_resources(int &num_processes, int &num_resources, int &requesting_process, vector<int>
&available_resources,
                     vector<vector<int> > &allocated_resources, vector<vector<int> > &max_demand,
                     vector<vector<int> > &remaining_need, vector<int> &requested_resources) {
  cout << "Enter number of processes: ";
  cin >> num_processes;
  cout << "Enter number of resource types: ";
  cin >> num_resources;

  allocated_resources.assign(num_processes, vector<int>(num_resources, 0));
  max_demand.assign(num_processes, vector<int>(num_resources, 0));
  remaining_need.assign(num_processes, vector<int>(num_resources, 0));
  available_resources.assign(num_resources, 0);
  requested_resources.assign(num_resources, 0);

  cout << "Enter available resources for each type:\n";
  for (int i = 0; i < num_resources; i++) cin >> available_resources[i];
```

```
    for (int i = 0; i < num_processes; i++) {
        cout << "Enter maximum demand for each resource for process " << i + 1 << ":\n";
        for (int j = 0; j < num_resources; j++) cin >> max_demand[i][j];
    }

    for (int i = 0; i < num_processes; i++) {
        cout << "Enter allocated resources for each type for process " << i + 1 << ":\n";
        for (int j = 0; j < num_resources; j++) cin >> allocated_resources[i][j];
    }

    for (int i = 0; i < num_processes; i++) {
        for (int j = 0; j < num_resources; j++) {
            remaining_need[i][j] = max_demand[i][j] - allocated_resources[i][j];
        }
    }

    cout << "Enter process ID making the request: ";
    cin >> requesting_process;
    requesting_process--;

    cout << "Enter requested instances of each resource for process " << requesting_process + 1 << ":\n";
    for (int j = 0; j < num_resources; j++) cin >> requested_resources[j];
}

bool can_fulfill_request(vector<int> &a, vector<int> &b) {
    for (int i = 0; i < a.size(); i++) {
        if (a[i] > b[i]) return false;
    }
    return true;
}

void increase(vector<int> &target, vector<int> &source) {
    for (int i = 0; i < source.size(); i++) {
        target[i] += source[i];
    }
}
```

```
void decrease(vector<int> &target, vector<int> &source) {
  for (int i = 0; i < source.size(); i++) {
    target[i] -= source[i];
  }
}

bool check_safety(int &num_processes, int &num_resources, vector<int> &available_resources,
            vector<vector<int> > &allocated_resources, vector<vector<int> > &remaining_need) {
  vector<int> work = available_resources;
  vector<bool> finished(num_processes, false);
  bool process_allocated = false;

  do {
    process_allocated = false;
    for (int i = 0; i < num_processes; i++) {
      if (!finished[i] && can_fulfill_request(remaining_need[i], work)) {
        increase(work, allocated_resources[i]);
        finished[i] = true;
        process_allocated = true;
      }
    }
  } while (process_allocated);

  for (bool status : finished) {
    if (!status) return false;
  }
  return true;
}

void handle_request(int &num_processes, int &num_resources, int &requesting_process, vector<int>
&available_resources,
            vector<vector<int> > &allocated_resources, vector<vector<int> > &remaining_need,
vector<int> &requested_resources) {
  if (!can_fulfill_request(requested_resources, remaining_need[requesting_process])) {
    cout << "Request exceeds maximum claim for process " << requesting_process + 1 << ".\n";
    return;
  }
```

```
  if (!can_fulfill_request(requested_resources, available_resources)) {
      cout << "Resources unavailable for process " << requesting_process + 1 << ". Process must wait.\n";
      return;
  }

  decrease(available_resources, requested_resources);
  increase(allocated_resources[requesting_process], requested_resources);
  decrease(remaining_need[requesting_process], requested_resources);

  if (check_safety(num_processes, num_resources, available_resources, allocated_resources, remaining_need)) {
      cout << "Resources allocated to process " << requesting_process + 1 << " successfully. No deadlock detected.\n";
  } else {
      cout << "Allocation denied to avoid deadlock.\n";
      increase(available_resources, requested_resources);
      decrease(allocated_resources[requesting_process], requested_resources);
      increase(remaining_need[requesting_process], requested_resources);
  }
}

int main() {
  int num_processes, num_resources, requesting_process;
  vector<int> available_resources;
  vector<int> requested_resources;
  vector<vector<int> > allocated_resources, max_demand, remaining_need;

setup_resources(num_processes, num_resources, requesting_process, available_resources, allocated_resources, max_demand, remaining_need, requested_resources);
  handle_request(num_processes, num_resources, requesting_process, available_resources, allocated_resources, remaining_need, requested_resources);

  return 0;
}
```

OUTPUT:

```
kunalbansal@KUNALs-MacBook-Pro OS LAB % cd "/Users/kunalbansal/Desktop/MAC/OS LAB/" && g++ bankers_prevention.cpp -o bankers_prevention &
& "/Users/kunalbansal/Desktop/MAC/OS LAB/"bankers_prevention
bankers_prevention.cpp:82:22: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
    for (bool status : finished) {
                     ^
1 warning generated.
Enter number of processes: 3
Enter number of resource types: 3
Enter available resources for each type:
5 5 5
Enter maximum demand for each resource for process 1:
5 0 0
Enter maximum demand for each resource for process 2:
2 3 2
Enter maximum demand for each resource for process 3:
0 2 3
Enter allocated resources for each type for process 1:
0 2 2
Enter allocated resources for each type for process 2:
2 0 0
Enter allocated resources for each type for process 3:
0 0 0
Enter process ID making the request: 1
Enter requested instances of each resource for process 1:
1 2 3
Request exceeds maximum claim for process 1.
kunalbansal@KUNALs-MacBook-Pro OS LAB %
```

LEARNING OUTCOME:
The experiment provided detailed understanding of the Banker's algorithm for the purpose of deadlock prevention and helped in analyzing its impact on system efficiency

EXPERIMENT 7

OBJECTIVE : Write a C/C++ program to simulate page replacement algorithm a) FIFO, b) LRU

INTRODUCTION:

Introduction: Page replacement algorithms manage memory by replacing pages when memory is full, aiming to minimize **page faults**.

- **FIFO (First In First Out)**: The oldest page in memory is replaced, which can lead to **Belady's Anomaly** where increasing frames increases page faults.
- **LRU (Least Recently Used)**: Replaces the page that hasn't been used for the longest time, based on the assumption that recently used pages will be needed again soon.

ALGORITHM :

- **FIFO**: Track pages in a queue. When memory is full, remove the oldest page and insert the new one.
- **LRU**: Maintain a record of page usage timestamps. Replace the page with the oldest timestamp when needed.

CODE:

//FIFO

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

int num_frames, num_pages;
queue<int> frames;
set<int> pages_in_memory;
vector<int> page_reference_string;
```

```
void initialize(){
  cout<<"Enter maximum number of frames: "<<endl; cin>>num_frames;
  cout<<"Enter number of pages to be entered: "<<endl; cin>>num_pages;
  page_reference_string.assign(num_pages, 0);
  cout<<"Enter page sequence: ";
  for(int i = 0; i < num_pages; i++) cin>>page_reference_string[i];
}

void fifo(){
  int page_faults = 0;
  for(int i = 0; i < num_pages; i++){
    if(pages_in_memory.find(page_reference_string[i]) != pages_in_memory.end()){
      cout<<"Page "<<page_reference_string[i]<<" in memory"<<endl;
    }
    else{
      cout<<"Page fault occurred"<<endl;
      if(frames.size() == num_frames){
        int replaced_page = frames.front(); frames.pop();
        pages_in_memory.erase(replaced_page);
        cout<<"Page "<<replaced_page<<" has been swapped out"<<endl;
      }
      pages_in_memory.insert(page_reference_string[i]);
      frames.push(page_reference_string[i]);
      cout<<"Page "<<page_reference_string[i]<<" has been swapped in"<<endl;
      page_faults++;
    }
  }
  cout<<"Number of hits = "<<num_pages - page_faults<<endl;
  cout<<"Number of misses = "<<page_faults<<endl;
}

int main(){
  initialize();
  fifo();
}
//LRU
# include <iostream>
```

```
# include <map>
# include <set>
using namespace std;

int num_frames, num_pages;
set<int> frames;
map<int, int> last_used_time;
vector<int> page_reference_string;

void initialize(){
   cout<<"Enter maximum number of frames: "<<endl; cin>>num_frames;
   cout<<"Enter number of pages to be entered: "<<endl; cin>>num_pages;
   page_reference_string.assign(num_pages, 0);
   cout<<"Enter page sequence: ";
   for(int i = 0; i < num_pages; i++) cin>>page_reference_string[i];
}

void lru(){
   int page_faults = 0;
   for(int i = 0; i < num_pages; i++){
      if(frames.find(page_reference_string[i]) != frames.end()){
         cout<<"Page "<<page_reference_string[i]<<" in memory"<<endl;
         last_used_time[page_reference_string[i]] = i + 1;
      }
      else{
         cout<<"Page fault occurred"<<endl;
         if(frames.size() == num_frames){
            int replaced_page = *frames.begin();
            int least_recent_time = last_used_time[replaced_page];
            for(int f: frames){
               if(last_used_time[f] < least_recent_time){
                  replaced_page = f;
                  least_recent_time = last_used_time[f];
               }
            }
            frames.erase(replaced_page);
            cout<<"Page "<<replaced_page<<" has been swapped out"<<endl;
```

```
        }
        last_used_time[page_reference_string[i]] = i + 1;
        frames.insert(page_reference_string[i]);
        cout<<"Page "<<page_reference_string[i]<<" has been swapped in"<<endl;
        page_faults++;
      }
    }
    cout<<"Number of hits = "<<num_pages - page_faults<<endl;
    cout<<"Number of misses = "<<page_faults<<endl;
}

int main(){
    initialize();
    lru();
}
```

CODE:

//FIFO

```
kunalbansal@KUNALs-MacBook-Pro OS LAB % cd "/Users/kunalbansal/Desktop/MAC/OS LAB/" && g++ FIFO.cpp -o FIFO && "/Users/kunalbansal/Deskto
p/MAC/OS LAB/"FIFO
Enter maximum number of frames:
3
Enter number of pages to be entered:
5
Enter page sequence: 2 3 4 1 3
Page fault occurred
Page 2 has been swapped in
Page fault occurred
Page 3 has been swapped in
Page fault occurred
Page 4 has been swapped in
Page fault occurred
Page 2 has been swapped out
Page 1 has been swapped in
Page 3 in memory
Number of hits = 1
Number of misses = 4
kunalbansal@KUNALs-MacBook-Pro OS LAB %
```

//LRU

```
kunalbansal@KUNALs-MacBook-Pro OS LAB % cd "/Users/kunalbansal/Desktop/MAC/OS LAB/" && g++ LRU.cpp -o LRU && "/Users/kunalbansal/Desktop/MAC/OS LAB/"LRU
LRU.cpp:31:26: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
                for(int f: frames){
                         ^
1 warning generated.
Enter maximum number of frames:
3
Enter number of pages to be entered:
5
Enter page sequence: 1 2 3 2 0
Page fault occurred
Page 1 has been swapped in
Page fault occurred
Page 2 has been swapped in
Page fault occurred
Page 3 has been swapped in
Page 2 in memory
Page fault occurred
Page 1 has been swapped out
Page 0 has been swapped in
Number of hits = 1
Number of misses = 4
kunalbansal@KUNALs-MacBook-Pro OS LAB %
```

LEARNING OUTCOME:
The experiment provided detailed understanding of the page replacement algorithms: FIFO, and LRU and helped in analyzing their impact on system efficiency.

EXPERIMENT 8

OBJECTIVE : Write a C/C++ program to simulate page replacement algorithm a) LFU, b) Optimal

INTRODUCTION:

Advanced page replacement algorithms focus on optimizing memory usage based on page access patterns.

- **LFU (Least Frequently Used)**: Replaces the page with the fewest accesses, assuming that frequently accessed pages are more likely to be used again.
- **Optimal**: Replaces the page that won't be used for the longest time in the future, offering optimal results but is impractical without future knowledge.

ALGORITHM :

LFU: Track access frequencies for each page. Replace the page with the lowest count.

Optimal: Identify the page that won't be needed for the longest time in the future. Replace that page to minimize future faults.

CODE:

//LFU

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

```
int num_frames, num_pages;
set<int> frames;
map<int, int> frequency;
vector<int> page_reference_string;

void initialize(){
  cout<<"Enter maximum number of frames: "<<endl; cin>>num_frames;
  cout<<"Enter number of pages to be entered: "<<endl; cin>>num_pages;
  page_reference_string.assign(num_pages, 0);
  cout<<"Enter page sequence: ";
  for(int i = 0; i < num_pages; i++) cin>>page_reference_string[i];
}

void lfu(){
  int page_faults = 0;
  for(int i = 0; i < num_pages; i++){
    if(frames.find(page_reference_string[i]) != frames.end()){
      cout<<"Page "<<page_reference_string[i]<<" in memory"<<endl;
      frequency[page_reference_string[i]]++;
    }
    else{
      cout<<"Page fault occurred"<<endl;
      if(frames.size() == num_frames){
        int replaced_page = *frames.begin();
        int least_frequency = frequency[replaced_page];
        for(int f: frames){
          if(frequency[f] < least_frequency){
            replaced_page = f;
            least_frequency = frequency[f];
          }
        }
        frames.erase(replaced_page);
        cout<<"Page "<<replaced_page<<" has been swapped out"<<endl;
      }
      frequency[page_reference_string[i]] = 1;
      frames.insert(page_reference_string[i]);
```

```
            cout<<"Page "<<page_reference_string[i]<<" has been swapped in"<<endl;
            page_faults++;
        }
    }
    cout<<"Number of hits = "<<num_pages - page_faults<<endl;
    cout<<"Number of misses = "<<page_faults<<endl;
}

int main(){
    initialize();
    lfu();
}
//Optimal
# include <iostream>
# include <set>
# include <map>
using namespace std;

int num_frames, num_pages;
set<int> frames;
vector<int> page_reference_string;

void initialize(){
    cout<<"Enter maximum number of frames: "<<endl; cin>>num_frames;
    cout<<"Enter number of pages to be entered: "<<endl; cin>>num_pages;
    page_reference_string.assign(num_pages, 0);
    cout<<"Enter page sequence: ";
    for(int i = 0; i < num_pages; i++) cin>>page_reference_string[i];
}

int predict(int i){
    map<int, int> farthest;
    for(int j = num_pages - 1; j > i; j--){
        farthest[page_reference_string[j]] = j + 1;
    }
    int farthest_page = *frames.begin();
    int farthest_occurence = farthest[farthest_page];
```

```
    for(int f: frames){
        if(farthest[f] < farthest_occurence){
            farthest_page = f;
            farthest_occurence = farthest[f];
        }
    }
    return farthest_page;
}

void optimal(){
    int page_faults = 0;
    for(int i = 0; i < num_pages; i++){
        if(frames.find(page_reference_string[i]) != frames.end()){
            cout<<"Page "<<page_reference_string[i]<<" in memory"<<endl;
        }
        else{
            cout<<"Page fault occurred"<<endl;
            if(frames.size() == num_frames){
                int replaced_page = predict(i);
                frames.erase(replaced_page);
                cout<<"Page "<<replaced_page<<" has been swapped out"<<endl;
            }
            frames.insert(page_reference_string[i]);
            cout<<"Page "<<page_reference_string[i]<<" has been swapped in"<<endl;
            page_faults++;
        }
    }
    cout<<"Number of hits = "<<num_pages - page_faults<<endl;
    cout<<"Number of misses = "<<page_faults<<endl;
}

int main(){
    initialize();
    optimal();
}
```

OUTPUT:

//LFU

```
kunalbansal@KUNALs-MacBook-Pro OS LAB % cd "/Users/kunalbansal/Desktop/MAC/OS LAB/" && g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile &
& "/Users/kunalbansal/Desktop/MAC/OS LAB/"tempCodeRunnerFile
tempCodeRunnerFile.cpp:31:26: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
                for(int f: frames){
                        ^
1 warning generated.
Enter maximum number of frames:
3
Enter number of pages to be entered:
7
Enter page sequence: 1 1 2 3 1 2 3
Page fault occurred
Page 1 has been swapped in
Page 1 in memory
Page fault occurred
Page 2 has been swapped in
Page fault occurred
Page 3 has been swapped in
Page 1 in memory
Page 2 in memory
Page 3 in memory
Number of hits = 4
Number of misses = 3
kunalbansal@KUNALs-MacBook-Pro OS LAB %
```

//OPTIMAL

```
kunalbansal@KUNALs-MacBook-Pro OS LAB % cd "/Users/kunalbansal/Desktop/MAC/OS LAB/" && g++ Optimal.cpp -o Optimal && "/Users/kunalbansal/
Desktop/MAC/OS LAB/"Optimal
Optimal.cpp:25:14: warning: range-based for loop is a C++11 extension [-Wc++11-extensions]
    for(int f: frames){
             ^
1 warning generated.
Enter maximum number of frames:
3
Enter number of pages to be entered:
10
Enter page sequence: 1 2 3 4 1 2 3 4 1 2
Page fault occurred
Page 1 has been swapped in
Page fault occurred
Page 2 has been swapped in
Page fault occurred
Page 3 has been swapped in
Page fault occurred
Page 1 has been swapped out
Page 4 has been swapped in
Page fault occurred
Page 2 has been swapped out
Page 1 has been swapped in
Page fault occurred
Page 3 has been swapped out
Page 2 has been swapped in
Page fault occurred
Page 4 has been swapped out
Page 3 has been swapped in
Page fault occurred
Page 3 has been swapped out
Page 4 has been swapped in
Page 1 in memory
Page 2 in memory
Number of hits = 2
Number of misses = 8
kunalbansal@KUNALs-MacBook-Pro OS LAB %
```

LEARNING OUTCOME:

The experiment provided detailed understanding of the page replacement algorithms: LFU, and Optimal and helped in analyzing their impact on system efficiency.

EXPERIMENT 9

OBJECTIVE : Write a C/C++ program to simulate producer-consumer problem using semaphores.

INTRODUCTION:

- This classic synchronization problem manages producers (who add items) and consumers (who remove items) working with a shared buffer. Semaphores ensure mutual exclusion and control access.

ALGORITHM :

Use a mutex semaphore for mutual exclusion, and two semaphores to track empty and full slots. Producers wait if the buffer is full, and consumers wait if it's empty, ensuring safe concurrent access.

CODE:

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

class semaphore{
  int S;

  public:
    void init(int x){
```

```
        S = x;
    }

    void wait(){
        S--;
    }

    void signal(){
        S++;
    }

    int val(){
        return S;
    }
};
int n;
queue<int> buffer;

semaphore buffer_lock, empty_buffers, full_buffers;

void set_buffer(){
  cout<<"Enter buffer size: "; cin>>n;
  buffer_lock.init(1);
  empty_buffers.init(n);
  full_buffers.init(0);
}

void producer(){
  if((buffer_lock.val() == 0) || (empty_buffers.val() == 0)){
     cout<<"Buffer is FULL"<<endl;
     return;
  }
  empty_buffers.wait();
  buffer_lock.wait();
  int x; cout<<"Enter item to produce: "; cin>>x;
  buffer.push(x); cout<<"Producer has produced item: "<<x<<endl;
  buffer_lock.signal();
```

```
  full_buffers.signal();
}

void consumer(){
  if((buffer_lock.val() == 0) || (full_buffers.val() == 0)){
     cout<<"Buffer is EMPTY"<<endl;
     return;
  }
  full_buffers.wait();
  buffer_lock.wait();
  int x = buffer.front(); buffer.pop();
  cout<<"Consumer has consumed item: "<<x<<endl;
  buffer_lock.signal();
  empty_buffers.signal();
}

int main(){
  set_buffer();
  cout<<"1. Producer"<<endl;
  cout<<"2. Consumer"<<endl;
  cout<<"3. Exit"<<endl;
  while(true){
     int choice; cout<<"Enter Choice: "; cin>>choice;
     if(choice == 1){
        producer();
     }
     else if(choice == 2){
        consumer();
     }
     else if(choice == 3){
        cout<<"Program Terminated"<<endl;
        return 0;
     }
     else{
        cout<<"Invalid Choice, Try Again"<<endl;
     }
  }
```

}

CODE:

```
kunalbansal@KUNALs-MacBook-Pro OS LAB % cd "/Users/kunalbansal/Desktop/MAC/OS LAB/" && g++ PROD_CONSUM.cpp -o PROD_CONSUM && "/Users/kuna
lbansal/Desktop/MAC/OS LAB/"PROD_CONSUM
Enter buffer size: 5
1. Producer
2. Consumer
3. Exit
Enter Choice: 1
Enter item to produce: 1
Producer has produced item: 1
Enter Choice: 2
Consumer has consumed item: 1
Enter Choice: 1
Enter item to produce: 3
Producer has produced item: 3
Enter Choice: 2
Consumer has consumed item: 3
Enter Choice: 2
Buffer is EMPTY
Enter Choice: 2
Buffer is EMPTY
Enter Choice: 3
Program Terminated
kunalbansal@KUNALs-MacBook-Pro OS LAB %
```

LEARNING OUTCOME:
The experiment provided a detailed understanding of the use of semaphores in solving the producer-consumer problem, and helped in analyzing how process synchronization works.

EXPERIMENT 10

OBJECTIVE : Write a C/C++ program to simulate disk scheduling algorithms a) FCFS, b) SCAN

INTRODUCTION:

Disk scheduling algorithms determine the order of disk I/O requests, minimizing **seek time** and improving system performance.

- **FCFS**: Processes requests in the order they arrive, which is simple but inefficient when requests are scattered.
- **SCAN**: The disk arm moves in one direction (e.g., towards the end of the disk), servicing requests along the way, then reverses. It reduces seek time for clustered requests.

ALGORITHM :

- **FCFS: Queue requests in arrival order. Move the disk head to each request sequentially.**
- **SCAN: Start from one end, service requests in order, then reverse direction, reducing seek time by covering each request in a sweeping motion.**

CODE:

//FCFS

```
#include <iostream>
#include <vector>
#include <cmath>  // For abs function
using namespace std;
```

```
class DiskScheduler {
  vector<int> request_sequence;
  vector<int> seek_sequence;
  int head;
  int seek_time;
  const int disk_size;

public:
  DiskScheduler(int disk_size = 200) : disk_size(disk_size), seek_time(0), head(0) {}

  void initialize() {
    int n;
    cout << "Enter the number of tracks in request sequence: ";
    cin >> n;

    request_sequence.resize(n);
    cout << "Enter request sequence: ";
    for (int &track : request_sequence) cin >> track;

    cout << "Enter initial head position: ";
    cin >> head;
  }

  void fcfs() {
    seek_time = 0;
    seek_sequence.clear();

    for (int track : request_sequence) {
      cout << "Head moving from " << head << " to " << track << endl;
      seek_sequence.push_back(track);

      int distance = abs(track - head);
      seek_time += distance;
      head = track;
    }
```

```
        display_result();
    }

private:
    void display_result() const {
        cout << "Total Seek Time: " << seek_time << endl;
        cout << "Seek Sequence: ";
        for (int track : seek_sequence) cout << track << " ";
        cout << endl;
    }
};

int main() {
    DiskScheduler scheduler;
    scheduler.initialize();
    scheduler.fcfs();
}

//SCAN
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
using namespace std;

class DiskScheduler {
    vector<int> request_sequence;
    vector<int> seek_sequence;
    string direction;
    int head;
    int seek_time;
    const int disk_size;

public:
    DiskScheduler(int disk_size = 200)
        : disk_size(disk_size), head(0), seek_time(0), direction("left") {}
```

```
  void initialize() {
    int n;
    cout << "Enter number of tracks in request sequence: ";
    cin >> n;

    request_sequence.resize(n);
    cout << "Enter request sequence: ";
    for (int &track : request_sequence) cin >> track;

    cout << "Enter initial head position: ";
    cin >> head;

    cout << "Enter initial direction of movement (left/right): ";
    cin >> direction;
  }

  void scan() {
    vector<int> left, right;
    if (direction == "left") left.push_back(0);
    else if (direction == "right") right.push_back(disk_size - 1);

    for (int track : request_sequence) {
      if (track < head) left.push_back(track);
      if (track > head) right.push_back(track);
    }

    sort(left.begin(), left.end());
    sort(right.begin(), right.end());

    process_tracks(left, right);
    display_result();
  }

private:
  void process_tracks(vector<int> &left, vector<int> &right) {
    int runs = 2;
    while (runs--) {
```

```
        if (direction == "left") {

            for (auto it = left.rbegin(); it != left.rend(); ++it) {
                move_head_to(*it);
            }
            direction = "right";
        } else if (direction == "right") {

            for (int track : right) {
                move_head_to(track);
            }
            direction = "left";
        }
    }
  }

  void move_head_to(int track) {
    cout << "Head has moved from " << head << " to " << track << endl;
    seek_sequence.push_back(track);

    int distance = abs(track - head);
    seek_time += distance;
    head = track;
  }

  void display_result() const {
    cout << "Total Seek Time: " << seek_time << endl;
    cout << "Seek Sequence: ";
    for (int track : seek_sequence) cout << track << " ";
    cout << endl;
  }
};
int main() {
  DiskScheduler scheduler;
  scheduler.initialize();
  scheduler.scan();
}
```

OUTPUT:

//FCFS

```
kunalbansal@KUNALs-MacBook-Pro OS LAB % cd "/Users/kunalbansal/Desktop/MAC/OS LAB/" && g++ fcfs_disk.cpp -o fcfs_disk && "/Users/kunalban
sal/Desktop/MAC/OS LAB/"fcfs_disk
Enter number of tracks in request sequence: 5
Enter request sequence: 12 20 29 1 10
Enter initial head position: 50
Head has moved from 50 to 12
Head has moved from 12 to 20
Head has moved from 20 to 29
Head has moved from 29 to 1
Head has moved from 1 to 10
Total Seek Time: 92
Seek Sequence:
12 20 29 1 10
kunalbansal@KUNALs-MacBook-Pro OS LAB %
```

//SCAN

```
3 warnings generated.
Enter number of tracks in request sequence: 5
Enter request sequence: 10 12 2 51 9
Enter initial head position: 50
Enter initial direction of movement (left/right): left
Head has moved from 50 to 12
Head has moved from 12 to 10
Head has moved from 10 to 9
Head has moved from 9 to 2
Head has moved from 2 to 0
Head has moved from 0 to 51
Total Seek Time: 101
Seek Sequence: 12 10 9 2 0 51
kunalbansal@KUNALs-MacBook-Pro OS LAB %
```

LEARNING OUTCOME:

The experiment provided a detailed understanding of disk scheduling algorithms: FCFS and SCAN, and helped in analyzing their impact on system efficiency.