|  |  |  |  |
| --- | --- | --- | --- |
| **Course Name:** | **Operating Systems and Compilers** | **Semester:** | **VI** |
| **Date of Performance:** | **21 / 03 / 2025** | **Batch No.:** | **B - 2** |
| **Faculty Name:** | **Prof. Nilesh Lakade** | **Roll No.:** | **16014022050** |
| **Faculty Sign & Date:** |  | **Grade/Marks:** | **\_\_\_ / 25** |

**Experiment No.: 5**

**Title:** **Simulate Bankers Algorithm for Deadlock Avoidance.**

|  |
| --- |
| **Aim and Objective of the Experiment:** |
| To implement Banker’s Algorithm for Deadlock Avoidance. |

|  |
| --- |
| **COs to be achieved:** |
| **CO2:** Describe the problems related to process concurrency and the different synchronization mechanisms available to solve them. |

|  |
| --- |
| **Theory:** |
| The Banker's algorithm is a [resource allocation](http://en.wikipedia.org/wiki/Resource_allocation) and [deadlock](http://en.wikipedia.org/wiki/Deadlock) avoidance [algorithm](http://en.wikipedia.org/wiki/Algorithm) developed by [Edsger Dijkstra.](http://en.wikipedia.org/wiki/Edsger_Dijkstra) DATA STRUCTURES (where *n* is the number of processes in the system and *m* is the number of resource types). |
| **Implementation details:** |
| #include <iostream>  #include <vector>  using namespace std;  bool isSafe(vector<int> &available, vector<vector<int>> &max\_need, vector<vector<int>> &allocation, vector<string> &processes) {      int num\_processes = processes.size();      int num\_resources = available.size();      vector<vector<int>> need(num\_processes, vector<int>(num\_resources));      vector<int> work = available;      vector<bool> finish(num\_processes, false);      vector<string> safe\_sequence;      for (int i = 0; i < num\_processes; i++) {          for (int j = 0; j < num\_resources; j++) {              need[i][j] = max\_need[i][j] - allocation[i][j];          }      }      cout << "Initial Available Resources: ";      for (int j = 0; j < num\_resources; j++) {          cout << work[j] << " ";      }      cout << endl;      while (safe\_sequence.size() < num\_processes) {          bool allocated\_in\_this\_round = false;          for (int i = 0; i < num\_processes; i++) {              if (!finish[i]) {                  bool can\_allocate = true;                  for (int j = 0; j < num\_resources; j++) {                      if (need[i][j] > work[j]) {                          can\_allocate = false;                          break;                      }                  }                  if (can\_allocate) {                      cout << "\nProcess " << processes[i] << " can be allocated resources.";                      cout << "\nAvailable before allocation: ";                      for (int j = 0; j < num\_resources; j++) {                          cout << work[j] << " ";                      }                      cout << "\nReleasing resources allocated to " << processes[i] << ": ";                      for (int j = 0; j < num\_resources; j++) {                          cout << allocation[i][j] << " ";                          work[j] += allocation[i][j];                      }                      cout << "\nAvailable after allocation: ";                      for (int j = 0; j < num\_resources; j++) {                          cout << work[j] << " ";                      }                      cout << endl;                      finish[i] = true;                      safe\_sequence.push\_back(processes[i]);                      allocated\_in\_this\_round = true;                  }              }          }          if (!allocated\_in\_this\_round) {              cout << "\nSystem is in unsafe state";              cout << "\nDeadlock detected\n";              return false;          }      }      cout << "\nSystem is in safe state";      cout << "\nSafe sequence:";      for (const auto &p : safe\_sequence) {          cout << "\n" << p;      }      cout << endl;      return true;  }  int main() {      vector<string> processes = {"P1", "P2", "P3", "P4", "P5"};      vector<int> available = {3, 3, 2};      vector<vector<int>> max\_need = {          {7, 5, 3},          {3, 2, 2},          {9, 0, 2},          {4, 2, 2},          {5, 3, 3}      };      vector<vector<int>> allocation = {          {0, 1, 0},          {2, 0, 0},          {3, 0, 2},          {2, 1, 1},          {0, 0, 2}      };      isSafe(available, max\_need, allocation, processes);      return 0;  } |

|  |
| --- |
| **Post Lab Subjective/Objective type Questions:** |
| 1. **The wait-for graph is a deadlock detection algorithm that is applicable when:**    1. All resources have a single instance    2. All resources have multiple instances    3. Both a and b    4. None of the above   **Ans: A**   1. **Resources are allocated to the process on non-sharable basis is \_**    1. Hold and Wait    2. Mutual Exclusion    3. No pre-emption    4. Circular Wait   **Ans: B**   1. **Which of the following approaches require knowledge of the system state?**    1. Deadlock Detection    2. Deadlock Prevention    3. Deadlock Avoidance    4. All of the above   **Ans: D**   1. **Consider a system having ‘m’ resources of the same type. These resources are shared by 3 processes A, B, C which have peak time demands of 3, 4, 6 respectively. The minimum value of ‘m’ that ensures that deadlock will never occur is,**    1. 11    2. 12    3. 13    4. 14   **Ans: 12** |

|  |
| --- |
| **Conclusion:** |
| The implementation of the Banker's Algorithm was successfully completed, and its output was thoroughly compared and validated against theoretical results. |

|  |
| --- |
| **Signature of faculty in-charge with Date:** |