IT05P Operating Systems Lab

|  |  |
| --- | --- |
| Semester | S.E. Semester IV |
| Subject | Operating Systems Lab |
| Subject Professor In-charge | Prof. Mahesh Khandke |
| Laboratory | L11 |

|  |  |
| --- | --- |
| Student Name | **Sameed Mulla** |
| Roll Number | **23101C0030** |

|  |  |
| --- | --- |
| Experiment No. | **09** |
| Experiment Title | **Simulate a program that simulates virtual memory.** |
| Objective | A video streaming platform, such as Netflix or YouTube, serves millions of users who stream videos on-demand. The platform uses a buffer memory to temporarily store parts of the video being streamed. To ensure smooth playback and minimize data transfer delays, the system maintains a limited buffer that can hold a fixed number of video chunks (pages).  When a user skips forward or backward in a video or streams a different video, the buffer may become full, requiring a page replacement algorithm to decide which video chunk to evict.  Example: Workflow Initial Setup:  Buffer capacity: 3 video chunks.  Sequence of chunk requests for a video: A, B, C, A, D, B, E, C, A, F. |
| Program | import java.util.\*;  public class VirtualMemorySimulator { public static void main(String[] args) {  Scanner scanner = new Scanner(System.*in*); System.*out*.print("Enter buffer capacity: "); int capacity = scanner.nextInt(); scanner.nextLine();  System.*out*.print("Enter sequence of chunk requests (separated by spaces): ");  String inputLine = scanner.nextLine(); String[] requests = inputLine.split("\\s+");  System.*out*.println("\n--- FIFO Simulation ---"); SimulationResult fifoResult = *simulateFIFO*(requests,  capacity);  System.*out*.println("\n--- LRU Simulation ---"); SimulationResult lruResult = *simulateLRU*(requests,  capacity);  System.*out*.println("\n--- Optimal Simulation ---"); SimulationResult optimalResult = *simulateOptimal*(requests,  capacity);  System.*out*.println("\n--- Summary ---"); System.*out*.println("FIFO: " + fifoResult.hits + " HITs, "  + fifoResult.faults + " PAGE FAULTs."); |

|  |  |
| --- | --- |
|  | System.*out*.println("LRU : " + lruResult.hits + " HITs, " + lruResult.faults + " PAGE FAULTs.");  System.*out*.println("Optimal: " + optimalResult.hits + " HITs, " + optimalResult.faults + " PAGE FAULTs.");  *analyzeThrashing*(requests.length, fifoResult, "FIFO");  *analyzeThrashing*(requests.length, lruResult, "LRU");  *analyzeThrashing*(requests.length, optimalResult,  "Optimal");  }  static class SimulationResult { int hits;  int faults;  SimulationResult(int hits, int faults) { this.hits = hits;  this.faults = faults;  }  }  public static SimulationResult simulateFIFO(String[] requests, int capacity) {  Queue<String> buffer = new LinkedList<>(); int hits = 0, faults = 0;  for (String page : requests) { if (buffer.contains(page)) {  System.*out*.println("Request " + page + ": HIT"); hits++;  } else {  faults++;  if (buffer.size() == capacity) { String removed = buffer.poll();  System.*out*.println("Request " + page + ": PAGE FAULT - Evicting " + removed);  } else {  System.*out*.println("Request " + page + ": PAGE  FAULT");  }  buffer.add(page);  }  }  return new SimulationResult(hits, faults);  }  public static SimulationResult simulateLRU(String[] requests, int capacity) {  List<String> buffer = new ArrayList<>(); int hits = 0, faults = 0;  for (String page : requests) { if (buffer.contains(page)) {  System.*out*.println("Request " + page + ": HIT"); hits++;  buffer.remove(page); buffer.add(page);  } else {  faults++; |

|  |  |
| --- | --- |
|  | if (buffer.size() == capacity) { String removed = buffer.remove(0);  System.*out*.println("Request " + page + ": PAGE FAULT - Evicting " + removed);  } else {  System.*out*.println("Request " + page + ": PAGE  FAULT");  }  buffer.add(page);  }  }  return new SimulationResult(hits, faults);  }  public static SimulationResult simulateOptimal(String[] requests, int capacity) {  List<String> buffer = new ArrayList<>(); int hits = 0, faults = 0;  for (int i = 0; i < requests.length; i++) { String page = requests[i];  if (buffer.contains(page)) { System.*out*.println("Request " + page + ": HIT"); hits++;  } else {  faults++;  if (buffer.size() == capacity) { int indexToRemove = -1;  int farthestUsage = -1;  for (int j = 0; j < buffer.size(); j++) { String currentPage = buffer.get(j); int nextUse = Integer.*MAX\_VALUE*;  for (int k = i + 1; k < requests.length;  k++) {  if (requests[k].equals(currentPage)) { nextUse = k;  break;  }  }  if (nextUse > farthestUsage) { farthestUsage = nextUse; indexToRemove = j;  }  }  String removed = buffer.remove(indexToRemove); System.*out*.println("Request " + page + ": PAGE  FAULT - Evicting " + removed);  } else {  System.*out*.println("Request " + page + ": PAGE  FAULT");  }  buffer.add(page);  }  }  return new SimulationResult(hits, faults);  } |

|  |  |
| --- | --- |
|  | private static void analyzeThrashing(int totalRequests, SimulationResult result, String strategy) {  double faultRate = (double) result.faults / totalRequests; System.*out*.printf("%s fault rate: %.2f%%\n", strategy,  faultRate \* 100);  if (faultRate > 0.8) {  System.*out*.println("Warning: High fault rate detected (thrashing) with " + strategy + " strategy.");  }  }  } |
|  |
| Output |  |

|  |  |
| --- | --- |
|  |  |
| Conclusion | This lab demonstrates how different page replacement strategies—FIFO, LRU, and Optimal—affect system performance in a limited buffer environment. The simulations reveal that while simpler methods like FIFO and LRU can be |

|  |  |
| --- | --- |
|  | efficient, they may lead to high fault rates and potential thrashing under certain workloads, whereas the Optimal strategy, despite offering lower fault rates, remains impractical for real-time systems due to its need for future information. |