# Department of Computer Technology

### Vision of the Department

To be a well-known centre for pursuing computer education through innovative pedagogy, value-based education and industry collaboration.

### Mission of the Department

To establish learning ambience for ushering in computer engineering professionals in core and multidisciplinary area by developing Problem- solving skills through emerging technologies**.**

## Session 2025-2026

**Mission:** Means to achieve Vision

**Vision:** Dream of where you want.

**Program Educational Objectives of the program (PEO):** (broad statements that describe the professional and career accomplishments)

|  |  |  |  |
| --- | --- | --- | --- |
| PEO1 | **Preparation** | **P: Preparation** | **Pep-CL abbreviation**  **pronounce as Pep-si-lL easy to recall** |
| PEO2 | **Core Competence** | **E: Environment (Learning Environment)** |
| PEO3 | **Breadth** | **P: Professionalism** |
| PEO4 | **Professionalism** | **C: Core Competence** |
| PEO5 | **Learning**  **Environment** | **L: Breadth (Learning in diverse areas)** |

**Program Outcomes (PO):** (statements that describe what a student should be able to do and know by the end of a program)

## Keywords of POs:

Engineering knowledge, Problem analysis, Design/development of solutions, Conduct Investigations of Complex Problems, Engineering Tool Usage, The Engineer and The World, Ethics, Individual and Collaborative Team work, Communication, Project Management and Finance, Life-Long Learning

**PSO Keywords:** Cutting edge technologies, Research

“I am an engineer, and I know how to apply engineering knowledge to investigate, analyse and design solutions to complex problems using tools for entire world following all ethics in a collaborative way with proper management skills throughout my life.” *to contribute to the development of cutting-edge technologies and Research*.

**Integrity:** I will adhere to the Laboratory Code of Conduct and ethics in its entirety.

## Name and Signature of Student and Date

(Signature and Date in Handwritten)

# Department of Computer Technology

### Vision of the Department

To be a well-known centre for pursuing computer education through innovative pedagogy, value-based education and industry collaboration.

### Mission of the Department

To establish learning ambience for ushering in computer engineering professionals in core and multidisciplinary area by developing Problem- solving skills through emerging technologies**.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Session** | **2025-26 (ODD)** | **Course Name** | **Operating System Lab** |
| **Semester** | **5** | **Course Code** | **23IOT1504** |
| **Roll No** | 71 | **Name of Student** | Vedant Yerne |

|  |  |
| --- | --- |
| Practical Number | Practical no 6 |
| Course Outcome | 1. Understand Computer System Configuration and Simulate system resources efficiently using Linux Commands (CO1) 2. Analyse operating system functionalities utilizing system calls, thread programming and process scheduling   algorithms (CO2)   1. Apply Synchronization primitives to implement a Deadlock-free solution(CO3) 2. Simulate Disk scheduling, Memory allocation, File allocation, page replacement algorithms (CO4) |
| Aim | To understand and implement different **Page Replacement Algorithms** — **FIFO**, **LRU**, and **Optimal** — which are used by operating systems to manage memory efficiently and reduce page faults. |
| Problem Definition | When a program needs more pages than can fit into the available memory frames, the operating system must decide which page to replace. This process is known as **page replacement**. The aim is to replace pages in such a way that the total number of **page faults** is minimized. We will study and compare three popular algorithms — **FIFO**, **LRU**, and **Optimal** — to see how each one handles this situation. |
| Theory  (100 words) | **FIFO (First In First Out)** In the FIFO method, pages are replaced in the same order they entered memory — the oldest page gets removed first. It is easy to implement but not always efficient, as it might replace a frequently used page just because it arrived earlier. **LRU (Least Recently Used)** The LRU algorithm replaces the page that hasn’t been used for the longest time. It keeps track of past usage to make smarter decisions. This usually results in fewer page faults compared to FIFO. **Optimal Page Replacement** The Optimal algorithm looks into the future and replaces the page that will not be used for the longest time. It gives the minimum possible number of page faults, making it ideal for comparison, though not practical in real systems. |
| Procedure and Execution  (100 Words) | Step for Implementation:   1. **Start the program.** 2. **Input:**    * Number of pages in the reference string (n)    * The reference string (sequence of page numbers)    * Number of frames (f) 3. **Initialize** the page frames (usually to -1 to show they are empty). 4. **For each page** in the reference string:    * Check if the page is already in the frame (Hit).    * If not (Miss / Page Fault):      + Replace a page according to the algorithm’s rule.      + Increase the page fault count.    * Display the current frame contents. 5. **After all pages are processed**, display the total number of page faults. 6. Stop the program. |
| Code:  FIFO  #include <stdio.h>  int main() {  int i, j, n, frames[10], pages[30], f, count = 0, flag, pf = 0;  printf("Enter number of pages: ");  scanf("%d", &n);  printf("Enter the reference string: ");  for(i = 0; i < n; i++)  scanf("%d", &pages[i]);  printf("Enter number of frames: ");  scanf("%d", &f);  for(i = 0; i < f; i++)  frames[i] = -1;  printf("\nPage\tFrames\t\tPage Fault\n");  for(i = 0; i < n; i++) {  flag = 0;  for(j = 0; j < f; j++) {  if(frames[j] == pages[i]) {  flag = 1;  break;  }  }  if(flag == 0) {  frames[count % f] = pages[i];  count++;  pf++;  printf("%d\t", pages[i]);  for(j = 0; j < f; j++)  printf("%d ", frames[j]);  printf("\tPage Fault\n");  } else {  printf("%d\t", pages[i]);  for(j = 0; j < f; j++)  printf("%d ", frames[j]);  printf("\tNo Fault\n");  }  }  printf("\nTotal Page Faults = %d\n", pf);  return 0;  }  Least Recently Used  #include <stdio.h>  int findLRU(int time[], int n) {  int i, min = time[0], pos = 0;  for(i = 1; i < n; i++)  if(time[i] < min) {  min = time[i];  pos = i;  }  return pos;  }  int main() {  int frames[10], pages[30], temp[10], time[10];  int n, f, count = 0, flag1, flag2, pos, pf = 0, i, j;  printf("Enter number of pages: ");  scanf("%d", &n);  printf("Enter reference string: ");  for(i = 0; i < n; i++)  scanf("%d", &pages[i]);  printf("Enter number of frames: ");  scanf("%d", &f);  for(i = 0; i < f; i++)  frames[i] = -1;  printf("\nPage\tFrames\t\tPage Fault\n");  for(i = 0; i < n; i++) {  flag1 = flag2 = 0;  for(j = 0; j < f; j++) {  if(frames[j] == pages[i]) {  count++;  time[j] = count;  flag1 = flag2 = 1;  break;  }  }  if(flag1 == 0) {  for(j = 0; j < f; j++) {  if(frames[j] == -1) {  count++;  pf++;  frames[j] = pages[i];  time[j] = count;  flag2 = 1;  break;  }  }  }  if(flag2 == 0) {  pos = findLRU(time, f);  count++;  pf++;  frames[pos] = pages[i];  time[pos] = count;  }  printf("%d\t", pages[i]);  for(j = 0; j < f; j++)  printf("%d ", frames[j]);  if(flag1 == 0) printf("\tPage Fault\n");  else printf("\tNo Fault\n");  }  printf("\nTotal Page Faults = %d\n", pf);  return 0;  }  Optimal Page Replacement  #include <stdio.h>  int main() {  int i, j, k, n, f, pages[30], frames[10], temp[10], pf = 0, flag, max, pos;  printf("Enter number of pages: ");  scanf("%d", &n);  printf("Enter reference string: ");  for(i = 0; i < n; i++)  scanf("%d", &pages[i]);  printf("Enter number of frames: ");  scanf("%d", &f);  for(i = 0; i < f; i++)  frames[i] = -1;  printf("\nPage\tFrames\t\tPage Fault\n");  for(i = 0; i < n; i++) {  flag = 0;  for(j = 0; j < f; j++) {  if(frames[j] == pages[i]) {  flag = 1;  break;  }  }  if(flag == 0) {  if(i < f) {  frames[i] = pages[i];  } else {  for(j = 0; j < f; j++) {  int found = 0;  for(k = i + 1; k < n; k++) {  if(frames[j] == pages[k]) {  temp[j] = k;  found = 1;  break;  }  }  if(!found)  temp[j] = 9999;  }  max = temp[0];  pos = 0;  for(j = 1; j < f; j++) {  if(temp[j] > max) {  max = temp[j];  pos = j;  }  }  frames[pos] = pages[i];  }  pf++;  }  printf("%d\t", pages[i]);  for(j = 0; j < f; j++)  printf("%d ", frames[j]);  if(flag == 0) printf("\tPage Fault\n");  else printf("\tNo Fault\n");  }  printf("\nTotal Page Faults = %d\n", pf);  return 0;  } |
| Output:  FIFO    LEAST RECENTLY USED    OPTIMAL PAGE REPLACEMENT |

# Department of Computer Technology

### Vision of the Department

To be a well-known centre for pursuing computer education through innovative pedagogy, value-based education and industry collaboration.

### Mission of the Department

To establish learning ambience for ushering in computer engineering professionals in core and multidisciplinary area by developing Problem- solving skills through emerging technologies**.**

|  |  |
| --- | --- |
|  |  |
| Output Analysis | After running the programs for **FIFO**, **LRU**, and **Optimal Page Replacement algorithms**, the results show how each technique behaves when managing memory frames during page requests.  For the given example:  🔹 **Reference String:** 1 2 3 2 4 1 5 🔹 **Number of Frames:** 3 |
| Link of student Github profile where lab assignment has been uploaded |  |
| Conclusion | From the implementation and analysis of the **FIFO**, **LRU**, and **Optimal** page replacement algorithms, we conclude that each algorithm has its own way of managing memory and handling page faults:   * **FIFO (First In First Out)** is the simplest algorithm, easy to implement but not always efficient because it may replace frequently used pages simply based on their arrival time. * **LRU (Least Recently Used)** performs better in most cases as it replaces the page that hasn’t been used for the longest period, closely matching real-world usage patterns. * **Optimal Page Replacement** provides the **lowest number of page faults**, as it replaces the page that will not be used for the longest time in the future. However, it is **theoretical** because future references are not known in practical systems.   Overall, **Optimal gives the best theoretical performance**, but **LRU** is the most **practical and efficient** algorithm used in real operating systems for memory management. Thus, the experiment successfully demonstrates how different page replacement strategies affect the number of page faults and the overall efficiency of memory utilization. |
| Plag Report (Similarity index < 12%) |  |
| Date | 23/10/25 |