# 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 |  |
| 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 | Execution of file management system call |
| Problem Definition |  |
| Theory  (100 words) | **Definition of Preemptive Scheduling:**  Preemptive scheduling is a CPU scheduling method where the operating system can interrupt or pause a running process. This lets the system assign the CPU to another process that has higher priority or needs to run sooner. As a result, it improves responsiveness and CPU efficiency, particularly in real-time and multitasking systems.  **Example of the preemptive**  Round robin  Short remaining time first  Priority scheduling  **Definition of Non-Preemptive Scheduling:**  Non-preemptive scheduling is a CPU scheduling technique where, once a process starts executing, it continues until it either finishes or voluntarily gives up the CPU, such as during I/O operations. The operating system does not interrupt or switch the process, even if a higher-priority process arrives while it is running.  **Example of the non preemptive**  First come first serve  Shortest job first |
| Procedure and Execution  (100 Words) | Step for Implementation:  **GENERAL STEPS (for both types):**  These steps apply to both preemptive and non-preemptive scheduling:  List all processes with:  Process ID  Arrival Time (AT)  Burst Time (BT)  Priority (if using Priority Scheduling)  Sort processes based on the chosen algorithm:  For SJF: by Shortest Burst Time  For Priority: by Highest Priority (lower number = higher priority) |
| Code:  Preemptive scheduling   1. Round robin   #include <stdio.h>  int main() {  int n, i, time = 0, quantum;  int remaining\_bt[100], wt[100], tat[100];  int bt[100], pid[100];  int done;  // Input number of processes  printf("Enter total number of processes: ");  scanf("%d", &n);  // Input burst time for each process  for (i = 0; i < n; i++) {  pid[i] = i + 1;  printf("Enter burst time for process P%d: ", pid[i]);  scanf("%d", &bt[i]);  remaining\_bt[i] = bt[i]; // copy burst time  }  // Input time quantum  printf("Enter time quantum: ");  scanf("%d", &quantum);  // Round Robin Scheduling logic  do {  done = 1;  for (i = 0; i < n; i++) {  if (remaining\_bt[i] > 0) {  done = 0; // At least one process is still remaining  if (remaining\_bt[i] > quantum) {  time += quantum;  remaining\_bt[i] -= quantum;  } else {  time += remaining\_bt[i];  wt[i] = time - bt[i];  remaining\_bt[i] = 0;  }  }  }  } while (!done);  // Calculate Turnaround Time  for (i = 0; i < n; i++) {  tat[i] = bt[i] + wt[i];  }  // Display result  printf("\nProcess\tBurst Time\tWaiting Time\tTurnaround Time\n");  for (i = 0; i < n; i++) {  printf("P%d\t%d\t\t%d\t\t%d\n", pid[i], bt[i], wt[i], tat[i]);  }  // Calculate averages  float avg\_wt = 0, avg\_tat = 0;  for (i = 0; i < n; i++) {  avg\_wt += wt[i];  avg\_tat += tat[i];  }  avg\_wt /= n;  avg\_tat /= n;  printf("\nAverage Waiting Time: %.2f", avg\_wt);  printf("\nAverage Turnaround Time: %.2f\n", avg\_tat);  return 0;  }   1. Short remaining time first   #include <stdio.h>  #define MAX\_PROCESSES 100  int main() {  int n, i, j, time = 0, smallest, count = 0;  int at[MAX\_PROCESSES], bt[MAX\_PROCESSES], rt[MAX\_PROCESSES];  int wt[MAX\_PROCESSES], tat[MAX\_PROCESSES];  float avg\_wt = 0, avg\_tat = 0;  // Input number of processes  printf("Enter total number of processes: ");  scanf("%d", &n);  // Input arrival time and burst time  for (i = 0; i < n; i++) {  printf("Enter arrival time for process P%d: ", i + 1);  scanf("%d", &at[i]);  printf("Enter burst time for process P%d: ", i + 1);  scanf("%d", &bt[i]);  rt[i] = bt[i]; // remaining time  }  printf("\nProcess execution order:\n");  // Main loop: run until all processes are completed  while (count != n) {  smallest = -1;  // Find process with smallest remaining time at current time  for (i = 0; i < n; i++) {  if (at[i] <= time && rt[i] > 0) {  if (smallest == -1 || rt[i] < rt[smallest]) {  smallest = i;  }  }  }  if (smallest == -1) {  time++; // CPU is idle  continue;  }  // Execute the process for 1 unit  rt[smallest]--;  printf("Time %d: P%d is running\n", time, smallest + 1);  time++;  // If the process is finished  if (rt[smallest] == 0) {  count++;  int finish\_time = time;  wt[smallest] = finish\_time - bt[smallest] - at[smallest];  if (wt[smallest] < 0) wt[smallest] = 0;  tat[smallest] = wt[smallest] + bt[smallest];  }  }  // Output table  printf("\nProcess\tArrival\tBurst\tWaiting\tTurnaround\n");  for (i = 0; i < n; i++) {  printf("P%d\t%d\t%d\t%d\t%d\n", i + 1, at[i], bt[i], wt[i], tat[i]);  avg\_wt += wt[i];  avg\_tat += tat[i];  }  avg\_wt /= n;  avg\_tat /= n;  printf("\nAverage Waiting Time: %.2f", avg\_wt);  printf("\nAverage Turnaround Time: %.2f\n", avg\_tat);  return 0;  }    Non preemptive scheduling   1. First come first serve   #include <stdio.h>  int main() {  int n, i;  int pid[100], at[100], bt[100], wt[100], tat[100];  float avg\_wt = 0, avg\_tat = 0;    // Input number of processes  printf("Enter the number of processes: ");  scanf("%d", &n);  // Input Arrival Time and Burst Time  for (i = 0; i < n; i++) {  pid[i] = i + 1;  printf("Enter arrival time for Process P%d: ", pid[i]);  scanf("%d", &at[i]);  printf("Enter burst time for Process P%d: ", pid[i]);  scanf("%d", &bt[i]);  }  // Sort processes by Arrival Time (FCFS order)  for (i = 0; i < n - 1; i++) {  for (int j = 0; j < n - i - 1; j++) {  if (at[j] > at[j + 1]) {  // Swap everything if arrival time is greater  int temp;  temp = at[j];  at[j] = at[j + 1];  at[j + 1] = temp;  temp = bt[j];  bt[j] = bt[j + 1];  bt[j + 1] = temp;  temp = pid[j];  pid[j] = pid[j + 1];  pid[j + 1] = temp;  }  }  }  // Calculate Waiting Time and Turnaround Time  wt[0] = 0; // First process has no waiting time  for (i = 1; i < n; i++) {  int service\_time = at[i - 1] + bt[i - 1];  wt[i] = service\_time - at[i];  if (wt[i] < 0)  wt[i] = 0;  }  for (i = 0; i < n; i++) {  tat[i] = bt[i] + wt[i];  avg\_wt += wt[i];  avg\_tat += tat[i];  }  // Output  printf("\nProcess\tArrival\tBurst\tWaiting\tTurnaround\n");  for (i = 0; i < n; i++) {  printf("P%d\t%d\t%d\t%d\t%d\n", pid[i], at[i], bt[i], wt[i], tat[i]);  }  avg\_wt /= n;  avg\_tat /= n;  printf("\nAverage Waiting Time: %.2f", avg\_wt);  printf("\nAverage Turnaround Time: %.2f\n", avg\_tat);  return 0;  }   1. Shortest job first   #include <stdio.h>  int main() {  int n, i, j;  int pid[100], bt[100], at[100], wt[100], tat[100], completed[100];  float avg\_wt = 0, avg\_tat = 0;  // Input number of processes  printf("Enter the number of processes: ");  scanf("%d", &n);  // Input arrival time and burst time  for (i = 0; i < n; i++) {  pid[i] = i + 1;  printf("Enter arrival time for Process P%d: ", pid[i]);  scanf("%d", &at[i]);  printf("Enter burst time for Process P%d: ", pid[i]);  scanf("%d", &bt[i]);  completed[i] = 0; // mark all processes as not completed  }  int time = 0, completed\_count = 0;  while (completed\_count < n) {  int idx = -1;  int min\_bt = 9999;  // Find the shortest job available at current time  for (i = 0; i < n; i++) {  if (at[i] <= time && completed[i] == 0) {  if (bt[i] < min\_bt) {  min\_bt = bt[i];  idx = i;  }  // If two processes have same burst time, use arrival time  if (bt[i] == min\_bt && at[i] < at[idx]) {  idx = i;  }  }  }  if (idx != -1) {  wt[idx] = time - at[idx];  if (wt[idx] < 0) wt[idx] = 0;  time += bt[idx];  tat[idx] = wt[idx] + bt[idx];  completed[idx] = 1;  completed\_count++;  } else {  time++; // If no process is ready, time passes  }  }  // Display results  printf("\nProcess\tArrival\tBurst\tWaiting\tTurnaround\n");  for (i = 0; i < n; i++) {  printf("P%d\t%d\t%d\t%d\t%d\n", pid[i], at[i], bt[i], wt[i], tat[i]);  avg\_wt += wt[i];  avg\_tat += tat[i];  }  avg\_wt /= n;  avg\_tat /= n;  printf("\nAverage Waiting Time: %.2f", avg\_wt);  printf("\nAverage Turnaround Time: %.2f\n", avg\_tat);  return 0;  } |
| Output:  Preemtive scheduling   1. Round robin      1. Short remaining time scheduling       Non preemptive scheduling   1. First come first serve      1. Shortest job first |

# 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 | The definitions clearly differentiate between preemptive and non-preemptive CPU scheduling based on whether a running process can be interrupted. Preemptive scheduling improves system responsiveness by allowing higher-priority processes to take over the CPU. Examples include Round Robin and Shortest Remaining Time First. Non-preemptive scheduling focuses on completing a process once it starts, without interruption. This is shown by algorithms like First Come First Serve and Shortest Job First. Preemptive scheduling is better for multitasking and real-time systems, while non-preemptive scheduling is simpler but can cause longer wait times for some processes. Understanding these differences helps in choosing the right scheduling method based on system needs. |
| Link of student Github profile where lab assignment has been uploaded |  |
| Conclusion | Execution of file management system call is perform successfully |
| Plag Report (Similarity index < 12%) |  |
| Date | 18/8/25 |