

ISB&M - COE

DEPARTMENT OF COMPUTER ENGINEERING

SUBJECT CODE: 310248

LAB MANUAL

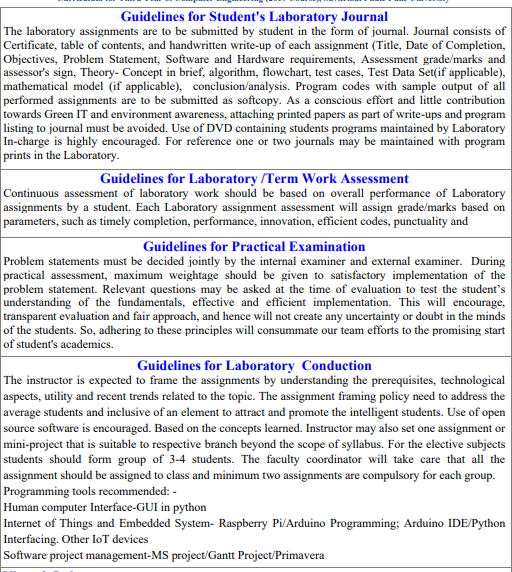
**System Programming & Operating System**

**Semester – Vth, Academic Year: 2022-23**

**List of Laboratory Assignments**

|  |  |
| --- | --- |
| **Sr.**  **No.** | **Group A (Any Two)** |
| 1 | Design suitable data structures and implement pass-I of a two-pass assembler for  pseudo-machine in Java using object oriented feature. Implementation should consist of a few instructions from each category and few assembler directives. |
| 2 | Implement Pass-II of two pass assembler for pseudo-machine in Java using object oriented features. The output of assignment-1 (intermediate file and symbol table)  should be input for this assignment. |
| 3 | Design suitable data structures and implement pass-I of a two-pass macro-  processor using OOP features in Java |
| 4 | Write a Java program for pass-II of a two-pass macro-processor. The output of  assignment-3 (MNT, MDT and file without any macro definitions) should be input for this assignment. |
|  | **Group B (Any Two)** |
| 5 | Write a program to simulate CPU scheduling algorithms: FCFS , SJF  (Preemptive), Priority (Non-Preemptive) and Round Robin (Preemptive) |
| 6 | Write a program to simulate memory replacement strategies- First Fit, Best Fit,  Worst Fit and Nest Fit. |
| 7 | Write a program to simulate page replacement algorithms using  1. FIFO 2. Least Recently Used (LRU) 3.Optimal algorithm |

Instructions



# Course Objectives:

1. To learn system programming tools
2. To learn modern operating system

# Course Outcome:

On completion of this course, learners will be able to

CO1: Implement different system software’s like assembler, macro processor, DLL, etc. CO2: Implement concept of synchronization and concurrency

CO3: Implement scheduling policies and memory management concepts of operating system

|  |
| --- |
| **Assignment No.: 01** |
| **Problem Statement:** Design suitable data structures and implement pass-I of a two-pass  assembler for pseudo machine in Java/C++ using object oriented feature. Implementation should consist of a few instructions from each category and few assembler directives. |
| **Objectives:**   1. To study the design and implementation of 1st pass of two pass assembler. 2. To study the categorized instruction set of assembler. 3. To study the data structure used in assembler implementation. |
| **Theory:**   1. Explain various Data and Instruction formats of assembly language programming. 2. Explain the design of Pass- I of assembler with the help of flowchart and example. 3. Discuss various Data structure used in Pass-I along with its format and significance of each field. |
| **Algorithm/Flowchart:**  C:\3.SPOS_2017_18\SPOS_2015_Pat_2017-18-sem2\DNP_SPOS_Unit_1\2 pass assem2.jpg |
| **Design diagrams (if any):**   1. Class Diagram 2. Use case Diagram 3. ER Diagram |
| **Input:**  Source code of Assembly Language  SAMPLE START 100  USING \*, 15  L 1, FOUR |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| A 1, =F’3’  ST 1, RESULT  SR 1, 2  LTORG  L 2, FIVE  A 2, =F’5’  A 2, =F’7’  FIVE DC F’5’  FOUR DC F’4’  RESULT DS 1F  END | | | |
| **Output:**  100 SAMPLE START 100  100 USING \*, 15  100 L 1, FOUR  104 A 1, =F’3’  108 ST 1, RESULT  112 SR 1, 2  114 LTORG  124 L 2, FIVE  128 A 2, =F’5’  132 A 2, =F’7’  136 FIVE DC F’5’  140 FOUR DC F’4’  144 RESULT DS 1F  152 5  156 7  160 END  **Machine Opcode Table (MOT)**  Mnemonic Hex / Binary Length (Bytes) Format  Code  L 5A 4 RX  A 1B 4 RX  ST 50 4 RX  SR 18 2 RR  **Pseudo Opcode Table (POT)** | | | |
|  | Pseudo op | Address / Name of Procedure to implement pseudo  operation |  |
| START | PSTART |
| USING | PUSING |
| DC | PDC |
| DS | PDS |
| LTORG | PLTORG |
| END | PEND |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Symbol Table (ST)**  Sr. No Symbol name Address Value Length Relocation 1 SAMPLE 100 -- 160 R   1. FIVE 136 5 4 R 2. FOUR 140 4 4 R 3. RESULT 144 4 R   **Literal Table (LT)** | | | | | |
|  | Sr. No | Literal | Address | Length |  |
| 1 | 3 | 120 | 4 |
| 2 | 5 | 152 | 4 |
| 3 | 7 | 156 | 4 |
| **Instructions :**  Not specific | | | | | |
| **Test Cases:**   1. Check syntax of instruction (Correct and wrong) 2. Symbol not found 3. Wrong instruction 4. Duplicate symbol declaration 5. Test the output of program by changing value of START pseudo opcode. 6. Test the output of program by changing position of LTORG pseudo-op. | | | | | |
| **Software Requirement:**   1. Fedora 2. Eclipse 3. JDK | | | | | |
| **Frequently Asked Questions:**   1. What is two pass assembler? 2. What is the significance of symbol table? 3. Explain the assembler directives EQU, ORIGIN. 4. Explain the assembler directives START, END, LTORG. 5. What is the use of POOLTAB and LITTAB? 6. How literals are handled in pass I? 7. What are the tasks done in Pass I? 8. How error handling is done in pass I? 9. Which intermediate data structures are designed and implemented in PassI? 10. What is the format of a machine code generated in PassII? 11. What is forward reference? How it is resolved by assembler? 12. How error handling is done in pass II? 13. What is the difference between IS, DL and AD? | | | | | |

|  |
| --- |
| 14. What are the tasks done in Pass II? |
| **Conclusion:**  Input assembly language program is processed by applying Pass-I algorithm of assembler and intermediate data structures, Symbol Table, Literal Table, MOT, POT, BT, etc. are generated. |

|  |
| --- |
| **Assignment No.: 02** |
| **Problem Statement:**  Implement Pass-II of two pass assembler for pseudo-machine in Java/C++ using object oriented features. The output of assignment-1 (intermediate file and symbol table) should be input for this assignment. |
| **Objectives:**   1. To study the design and implementation of 2nd pass of two pass assembler. 2. To study the data structure used in Pass-2 of assembler implementation. |
| **Theory:**  1. Explain the design of Pass- II of assembler with the help of flowchart and example. |
| **Algorithm/Flowchart:**  C:\3.SPOS_2017_18\SPOS_2015_Pat_2017-18-sem2\DNP_SPOS_Unit_1\2 pass assem3.jpg |
| **Design diagrams (if any):**   1. Class Diagram 2. Use case Diagram 3. ER Diagram |
| **Input:**  Intermediate code of pass-1.  **LC LABEL INSTR. OPERANDS** |

|  |  |  |  |
| --- | --- | --- | --- |
| Mnemonic | Hex / Binary  Code | Length (Bytes) | Format |
| L | 5A | 4 | RX |
| A | 1B | 4 | RX |
| ST | 50 | 4 | RX |
| SR | 18 | 2 | RR |

|  |  |
| --- | --- |
| Pseudo op | Address / Name of Procedure to implement pseudo  operation |
| START | PSTART |
| USING | PUSING |
| DC | PDC |
| DS | PDS |
| LTORG | PLTORG |
| END | PEND |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sr. No | Symbol name | Address | Value | Length | Relocation |
| 1 | SAMPLE | 100 | -- | 160 | R |
| 2 | FIVE | 136 | 5 | 4 | R |
| 3 | FOUR | 140 | 4 | 4 | R |
| 4 | RESULT | 144 |  | 4 | R |

|  |
| --- |
| **----------------------------------------------------------------**  100 SAMPLE START 100  100 USING \*, 15  100 L 1, FOUR  104 A 1, =F’3’  108 ST 1, RESULT  112 SR 1, 2  114 LTORG  124 L 2, FIVE  128 A 2, =F’5’  132 A 2, =F’7’  136 FIVE DC F’5’  140 FOUR DC F’4’  144 RESULT DS 1F  152 5  156 7  160 END  **Machine Opcode Table (MOT)**  **Pseudo Opcode Table (POT)**  **Symbol Table (ST)** |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Literal Table (LT)** Sr. No Literal 1 3  2 5  3 7 | Address 120  152  156 | | Length 4  4  4 |
| **Output:** |  |  | |
| **Base Table (BT)** |  |  | |
| Register no | Availability | Value/ Contents | |
| 1 | N | -- | |
| : | : | : | |
| : | : | : | |
| : | : | : | |
| 15 | Y | 100 | |
| **Object Code** |  |  | |
| **LC** |  | **OPCODE** | **OPERAND** |
| 100 |  | 5A | 1,40(0,15) |
| 104 |  | 1B | 1,20(0,15) |
| 108 |  | 50 | 1,44(0,15) |
| 112 |  | 18 | 1,2 |
| 124 |  | 5A | 2,36(0,15) |
| 128 |  | 1B | 2,52(0,15) |
| 132 |  | 1B | 2,56(0,15) |
| **Instructions :** | | | |
| **1.** | | | |
| **2.** | | | |
| **3.** | | | |
| **Test Cases:**   1. Check syntax of instruction (Correct and wrong) 2. Symbol not found 3. Wrong instruction 4. Duplicate symbol declaration 5. Test the output of program by changing value of START & USING pseudo opcode. | | | |
| **Software Requirement:**   1. Fedora 2. Eclipse 3. JDK | | | |

|  |
| --- |
| **Frequently Asked Questions:**   1. What is two pass assembler? 2. What is the significance of symbol table? 3. Explain the assembler directives EQU, ORIGIN. 4. Explain the assembler directives START, END, LTORG. 5. What is the use of POOLTAB and LITTAB? 6. How literals are handled in pass I? 7. What are the tasks done in Pass I? 8. How error handling is done in pass I? 9. Which intermediate data structures are designed and implemented in PassI? 10. What is the format of a machine code generated in PassII? 11. What is forward reference? How it is resolved by assembler? 12. How error handling is done in pass II? 13. What is the difference between IS, DL and AD? |
| **Conclusion:**  The intermediate data structures generated in Pass-I of assembler are given as input to the Pass-II of assembler, processed by applying Pass-II algorithm of assembler and machine code is generated |

|  |
| --- |
| **Assignment No.: 03** |
| **Problem Statement:** Design suitable data structures and implement Pass-I of a two pass  macro processor using OOP features in Java/C++. The output of Pass-I (MNT, MDT, ALA & Intermediate code file without any macro definitions) should be input for Pass-II. |
| **Objectives:**   1. To identify and design different data structure used in macro-processor implementation 2. To apply knowledge in implementation of two pass microprocessor. |
| **Theory:**   1. What is macro processor? 2. Differentiate Macro and Function? 3. Explain the design of Pass- I of macro-processor with the help of flowchart? 4. Explain the design of Data structure used in Pass-I? 5. Explain the data structures used in Pass-I? |
| **Algorithm/Flowchart:** |

|  |
| --- |
|  |
| **Design diagrams (if any):**   1. Class diagram 2. Sequence diagram 3. |
| **Input:**  Small assembly language program with macros written in file input.asm.  MACRO  &lab ADDS &arg1,&arg2 &lab L 1, &arg1  A 1, &arg2 MEND  PROG START 0  BALR 15,0 USING \*,15  LAB ADDS DATA1, DATA2 ST 4,1  DATA1 DC F’3’ DATA2 DC F’4’  END |
| **Output:**  Assembly language program without macro definition but with macro call.  **Note:** Follow the following templates during implementation  **Macro Name Table (MNT) :**    **Macro Definition Table (MDT) :**    **Argument List Array (ALA) :** |
| **Test Cases:** |

|  |
| --- |
| 1. Check macro end not found. 2. Duplicate macro name found. 3. Check program output by changing macro name and parameter list. 4. Handle label in macro definition. 5. Handle multiple macro definitions and calls |
| **Software Requirement:**   1. Fedora 2. Eclipse 3. JDK |
| **Frequently Asked Questions:**   1. Define macro? 2. Define purpose of pass-1 of two pass macro processor 3. List out types of macro arguments 4. What is the use of MDT-index field in MNT? 5. What we store in ALA? |
| **Conclusion:** We have successfully completed implementation of Pass-I of macro  processor. |

|  |
| --- |
| **Assignment No.: 04** |
| **Problem Statement:** Design suitable data structures and implement Pass-II of a two pass macro processor using OOP features in Java/C++. The output of Pass-I (MNT, MDT, ALA & Intermediate code file without any macro definitions) should be input for Pass-II. |
| **Objectives:**   1. To identify and design different data structure used in macro-processor implementation 2. To apply knowledge in implementation of pass-2 of two pass microprocessor. |
| **Theory:**  1. Explain design steps of two pass microprocessor, types of statements, data structures required  and flowcharts. |
| **Algorithm/Flowchart:** |
| **Design diagrams (if any):**   1. Class diagram 2. Sequence diagram 3. |
| **Input:** Output of pass-1 (Intermediate File) given as a input to pass-2. |

|  |
| --- |
| PROG START 0  BALR 15,0 USING \*,15  LAB ADDS DATA1, DATA2 ST 4,1  DATA1 DC F’3’ DATA2 DC F’4’  END |
| **Output:**  Assembly language program without macro definition and macro call. PROG START 0  BALR 15,0 USING \*,15  LAB L 1, DATA1 A 1, DATA2  ST 4,1  DATA1 DC F’3’ DATA2 DC F’4’ END |
| **Test Cases:** |

|  |
| --- |
| 1. Check macro definition not found. 2. Check program output by changing parameter list in macro call. |
| **Software Requirement:**   1. Fedora 2. Eclipse 3. JDK |
| **Frequently Asked Questions:**   1. What is macro expansion? 2. Define purpose of pass-2 of two pass macro processor 3. What is positional arguments? 4. What is the use of MDT-index field in MNT? 5. What is the use of MNT table while processing macro call? |
| **Conclusion:** We have successfully completed implementation of Pass-II of macro processor. |

|  |
| --- |
| **Assignment No.: 06** |
| **Problem Statement:**  Write a program to simulate CPU Scheduling Algorithms: FCFS, SJF (Preemptive), Priority (Non- Preemptive) and Round Robin (Preemptive). |
| **Objectives:**   1. To study the process management and various scheduling policies viz. Preemptive and Non preemptive. 2. To study and analyze different scheduling algorithms. |
| **Theory :**   1. Define process. Explain need of process scheduling. 2. Explain different scheduling criteria and policies for scheduling processes. 3. Explain possible process states 4. Explain FCFS, SJF(Preemptive), Priority (Non-Preemptive) and Round Robin (Preemptive) and determine waiting time, turnaround time, throughput using each algorithm. |
| **Algorithm/Flowchart:**  **FCFS**   * 1. Input the processes along with their burst time (bt).   2. Find waiting time (wt) for all processes.   3. As first process that comes need not to wait so waiting time for process 1 will be 0 i.e. wt[0] = 0.   4. Find **waiting time** for all other processes i.e. for process i -> wt[i] = bt[i-1] + wt[i-1] .   5. Find **turnaround time** = waiting\_time + burst\_time for all processes.   6. Find **average waiting time** = total\_waiting\_time / no\_of\_processes.   7. Similarly, find **average turnaround time** = total\_turn\_around\_time / no\_of\_processes.   #include<stdio.h>  void findWaitingTime(int processes[], int n,  int bt[], int wt[])  {  // waiting time for first process is 0  wt[0] = 0;  // calculating waiting time  for (int  i = 1; i < n ; i++ )  wt[i] =  bt[i-1] + wt[i-1] ;  }  // Function to calculate turn around time  void findTurnAroundTime( int processes[], int n,  int bt[], int wt[], int tat[])  {  // calculating turnaround time by adding  // bt[i] + wt[i]  for (int  i = 0; i < n ; i++)  tat[i] = bt[i] + wt[i];  }  //Function to calculate average time  void findavgTime( int processes[], int n, int bt[])  {  int wt[n], tat[n], total\_wt = 0, total\_tat = 0;  //Function to find waiting time of all processes  findWaitingTime(processes, n, bt, wt);  //Function to find turn around time for all processes  findTurnAroundTime(processes, n, bt, wt, tat);  //Display processes along with all details  printf("Processes   Burst time   Waiting time   Turn around time\n");  // Calculate total waiting time and total turn  // around time  for (int  i=0; i<n; i++)  {  total\_wt = total\_wt + wt[i];  total\_tat = total\_tat + tat[i];  printf("   %d ",(i+1));  printf("       %d ", bt[i] );  printf("       %d",wt[i] );  printf("       %d\n",tat[i] );  }  int s=(float)total\_wt / (float)n;  int t=(float)total\_tat / (float)n;  printf("Average waiting time = %d",s);  printf("\n");  printf("Average turn around time = %d ",t);  }  // Driver code  int main()  {  //process id's  int processes[] = { 1, 2, 3};  int n = sizeof processes / sizeof processes[0];  //Burst time of all processes  int  burst\_time[] = {10, 5, 8};  findavgTime(processes, n,  burst\_time);  return 0;  }  **SJF**   * 1. Traverse until all process gets completely executed.      1. Find process with minimum remaining time at every single time lap.      2. Reduce its time by 1.      3. Check if its remaining time becomes 0      4. Increment the counter of process completion.      5. Completion time of current process = current\_time +1;  1. Calculate waiting time for each completed process. wt[i]= Completion time - arrival\_time-burst\_time 2. Increment time lap by one. 3. Find Turn around time.   #include <stdio.h>  int main()  {      int A[100][4]; // Matrix for storing Process Id, Burst                     // Time, Average Waiting Time & Average                     // Turn Around Time.      int i, j, n, total = 0, index, temp;      float avg\_wt, avg\_tat;      printf("Enter number of process: ");      scanf("%d", &n);      printf("Enter Burst Time:\n");      // User Input Burst Time and alloting Process Id.      for (i = 0; i < n; i++) {          printf("P%d: ", i + 1);          scanf("%d", &A[i][1]);          A[i][0] = i + 1;      }      // Sorting process according to their Burst Time.      for (i = 0; i < n; i++) {          index = i;          for (j = i + 1; j < n; j++)              if (A[j][1] < A[index][1])                  index = j;          temp = A[i][1];          A[i][1] = A[index][1];          A[index][1] = temp;            temp = A[i][0];          A[i][0] = A[index][0];          A[index][0] = temp;      }      A[0][2] = 0;      // Calculation of Waiting Times      for (i = 1; i < n; i++) {          A[i][2] = 0;          for (j = 0; j < i; j++)              A[i][2] += A[j][1];          total += A[i][2];      }      avg\_wt = (float)total / n;      total = 0;      printf("P     BT     WT     TAT\n");      // Calculation of Turn Around Time and printing the      // data.      for (i = 0; i < n; i++) {          A[i][3] = A[i][1] + A[i][2];          total += A[i][3];          printf("P%d     %d     %d      %d\n", A[i][0],                 A[i][1], A[i][2], A[i][3]);      }      avg\_tat = (float)total / n;      printf("Average Waiting Time= %f", avg\_wt);      printf("\nAverage Turnaround Time= %f", avg\_tat);  } |

|  |
| --- |
| **Priority**   * 1. First input the processes with their burst time and priority.   2. Sort the processes, burst time and priority according to the priority. 3- Now simply apply FCFS algorithm.   // C++ program for implementation of FCFS  // scheduling  #include<bits/stdc++.h>  using namespace std;    struct Process  {      int pid;  // Process ID      int bt;   // CPU Burst time required      int priority; // Priority of this process  };    // Function to sort the Process acc. to priority  bool comparison(Process a, Process b)  {      return (a.priority > b.priority);  }    // Function to find the waiting time for all  // processes  void findWaitingTime(Process proc[], int n,                       int wt[])  {      // waiting time for first process is 0      wt[0] = 0;        // calculating waiting time      for (int  i = 1; i < n ; i++ )          wt[i] =  proc[i-1].bt + wt[i-1] ;  }    // Function to calculate turn around time  void findTurnAroundTime( Process proc[], int n,                           int wt[], int tat[])  {      // calculating turnaround time by adding      // bt[i] + wt[i]      for (int  i = 0; i < n ; i++)          tat[i] = proc[i].bt + wt[i];  }    //Function to calculate average time  void findavgTime(Process proc[], int n)  {      int wt[n], tat[n], total\_wt = 0, total\_tat = 0;        //Function to find waiting time of all processes      findWaitingTime(proc, n, wt);        //Function to find turn around time for all processes      findTurnAroundTime(proc, n, wt, tat);        //Display processes along with all details      cout << "\nProcesses  "<< " Burst time  "           << " Waiting time  " << " Turn around time\n";        // Calculate total waiting time and total turn      // around time      for (int  i=0; i<n; i++)      {          total\_wt = total\_wt + wt[i];          total\_tat = total\_tat + tat[i];          cout << "   " << proc[i].pid << "\t\t"               << proc[i].bt << "\t    " << wt[i]               << "\t\t  " << tat[i] <<endl;      }        cout << "\nAverage waiting time = "           << (float)total\_wt / (float)n;      cout << "\nAverage turn around time = "           << (float)total\_tat / (float)n;  }    void priorityScheduling(Process proc[], int n)  {      // Sort processes by priority      sort(proc, proc + n, comparison);        cout<< "Order in which processes gets executed \n";      for (int  i = 0 ; i <  n; i++)          cout << proc[i].pid <<" " ;        findavgTime(proc, n);  }    // Driver code  int main()  {      Process proc[] = {{1, 10, 2}, {2, 5, 0}, {3, 8, 1}};      int n = sizeof proc / sizeof proc[0];      priorityScheduling(proc, n);      return 0;  }  **RR**   1. Create an array **rem\_bt[]** to keep track of remaining burst time of processes. This array is initially a copy of bt[] (burst times array) 2. Create another array **wt[]** to store waiting times of processes. Initialize this array as 0. 3- Initialize time : t = 0 3. Keep traversing the all processes while all processes are not done. Do following for i'th process if it is not done yet.    1. If rem\_bt[i] > quantum       1. t = t + quantum       2. bt\_rem[i] -= quantum; 4. Else // Last cycle for this process    1. t = t + bt\_rem[i];    2. wt[i] = t - bt[i]    3. bt\_rem[i] = 0; // This process is over   // C++ program for implementation of RR scheduling  #include<iostream>  using namespace std;    // Function to find the waiting time for all  // processes  void findWaitingTime(int processes[], int n,              int bt[], int wt[], int quantum)  {      // Make a copy of burst times bt[] to store remaining      // burst times.      int rem\_bt[n];      for (int i = 0 ; i < n ; i++)          rem\_bt[i] = bt[i];        int t = 0; // Current time        // Keep traversing processes in round robin manner      // until all of them are not done.      while (1)      {          bool done = true;            // Traverse all processes one by one repeatedly          for (int i = 0 ; i < n; i++)          {              // If burst time of a process is greater than 0              // then only need to process further              if (rem\_bt[i] > 0)              {                  done = false; // There is a pending process                    if (rem\_bt[i] > quantum)                  {                      // Increase the value of t i.e. shows                      // how much time a process has been processed                      t += quantum;                        // Decrease the burst\_time of current process                      // by quantum                      rem\_bt[i] -= quantum;                  }                    // If burst time is smaller than or equal to                  // quantum. Last cycle for this process                  else                  {                      // Increase the value of t i.e. shows                      // how much time a process has been processed                      t = t + rem\_bt[i];                        // Waiting time is current time minus time                      // used by this process                      wt[i] = t - bt[i];                        // As the process gets fully executed                      // make its remaining burst time = 0                      rem\_bt[i] = 0;                  }              }          }            // If all processes are done          if (done == true)          break;      }  }    // Function to calculate turn around time  void findTurnAroundTime(int processes[], int n,                          int bt[], int wt[], int tat[])  {      // calculating turnaround time by adding      // bt[i] + wt[i]      for (int i = 0; i < n ; i++)          tat[i] = bt[i] + wt[i];  }    // Function to calculate average time  void findavgTime(int processes[], int n, int bt[],                                      int quantum)  {      int wt[n], tat[n], total\_wt = 0, total\_tat = 0;        // Function to find waiting time of all processes      findWaitingTime(processes, n, bt, wt, quantum);        // Function to find turn around time for all processes      findTurnAroundTime(processes, n, bt, wt, tat);        // Display processes along with all details      cout << "PN\t "<< " \tBT "          << "  WT " << " \tTAT\n";        // Calculate total waiting time and total turn      // around time      for (int i=0; i<n; i++)      {          total\_wt = total\_wt + wt[i];          total\_tat = total\_tat + tat[i];          cout << " " << i+1 << "\t\t" << bt[i] <<"\t "              << wt[i] <<"\t\t " << tat[i] <<endl;      }        cout << "Average waiting time = "          << (float)total\_wt / (float)n;      cout << "\nAverage turn around time = "          << (float)total\_tat / (float)n;  }    // Driver code  int main()  {      // process id's      int processes[] = { 1, 2, 3};      int n = sizeof processes / sizeof processes[0];        // Burst time of all processes      int burst\_time[] = {10, 5, 8};        // Time quantum      int quantum = 2;      findavgTime(processes, n, burst\_time, quantum);      return 0;  } |

|  |
| --- |
| **Frequently Asked Questions:**   1. What are the types of CPU scheduler? 2. What is the difference between long and short term scheduling? 3. Logic of program? 4. What is preemptive and non-preemptive scheduling? 5. What are types of scheduling algorithms? 6. Why Priority scheduling may cause low-priority processes to starve? 7. What are the goals of scheduling? 8. Define the difference between preemptive and non-preemptive scheduling. 9. Which scheduling algorithm is best? Why? |
| **Conclusion:**  CPU policies implemented successfully |

|  |
| --- |
| **Assignment No.: 07** |
| **Problem Statement:**  Write a Java/C++ program to simulate memory placement strategies   1. First Fit 2. Best Fit 3. Worst Fit 4. Next Fit |
| **Objectives:**   1. To acquire knowledge memory placement strategies 2. To be able to implement memory placement strategies |
| **Theory:**   1. Why we need memory placement strategies? 2. What is fragmentation? 3. Explain working of memory placement strategies with suitable example. |
| **Algorithm/Flowchart:**   1. **First Fit algorithm/pseudo code**    * Read all required input    * FOR i<-0 to all jobs ‘js’      + FOR j<-0 to all blocks ‘bs’        - IF block[j]>=jobs[i]          * Check jth block is already in use or free   Continue and search next free block   * + - * + Otherwise allocate jth block to ith job   + Display all job with allocated blocks and fragmentation   #include<iostream>  using namespace std;  int main()  {  int bsize[10], psize[10], bno, pno, flags[10], allocation[10], i, j;    for(i = 0; i < 10; i++)  {  flags[i] = 0;  allocation[i] = -1;  }  cout<<"Enter no. of blocks: ";  cin>>bno;  cout<<"\nEnter size of each block: ";  for(i = 0; i < bno; i++)  cin>>bsize[i];    cout<<"\nEnter no. of processes: ";  cin>>pno;  cout<<"\nEnter size of each process: ";  for(i = 0; i < pno; i++)  cin>>psize[i];  for(i = 0; i < pno; i++)         //allocation as per first fit  for(j = 0; j < bno; j++)  if(flags[j] == 0 && bsize[j] >= psize[i])  {  allocation[j] = i;  flags[j] = 1;  break;  }  //display allocation details  cout<<"\nBlock no.\tsize\t\tprocess no.\t\tsize";  for(i = 0; i < bno; i++)  {  cout<<"\n"<< i+1<<"\t\t"<<bsize[i]<<"\t\t";  if(flags[i] == 1)  cout<<allocation[i]+1<<"\t\t\t"<<psize[allocation[i]];  else  cout<<"Not allocated";  }  return 0;  }   1. **Best Fit algorithm/pseudo code**    * Read all required input    * FOR i<-0 to all jobs ‘js’      + SET BestInd -1      + FOR j<-0 to all blocks ‘bs’        - IF block[j]>=jobs[i]          * IF Block is free and BestInd==-1 THEN SET BestIndj          * ELSEIF Block is free and block[BestInd]>block[j] THEN SET BestIndj          * ELSE continue with next block   Continue and search next free block |

|  |
| --- |
| * IF BestInd!=-1 THEN allocate jth block to ith job   o Display all job with allocated blocks and fragmentation  #include<bits/stdc++.h>  using namespace std;  // Function to allocate memory to blocks as per Best fit  // algorithm  void bestFit(int blockSize[], int m, int processSize[], int n)  {  // Stores block id of the block allocated to a  // process  int allocation[n];  // Initially no block is assigned to any process  memset(allocation, -1, sizeof(allocation));  // pick each process and find suitable blocks  // according to its size ad assign to it  for (int i=0; i<n; i++)  {  // Find the best fit block for current process  int bestIdx = -1;  for (int j=0; j<m; j++)  {  if (blockSize[j] >= processSize[i])  {  if (bestIdx == -1)  bestIdx = j;  else if (blockSize[bestIdx] > blockSize[j])  bestIdx = j;  }  }  // If we could find a block for current process  if (bestIdx != -1)  {  // allocate block j to p[i] process  allocation[i] = bestIdx;  // Reduce available memory in this block.  blockSize[bestIdx] -= processSize[i];  }  }  cout << "\nProcess No.\tProcess Size\tBlock no.\n";  for (int i = 0; i < n; i++)  {  cout << " " << i+1 << "\t\t" << processSize[i] << "\t\t";  if (allocation[i] != -1)  cout << allocation[i] + 1;  else  cout << "Not Allocated";  cout << endl;  }  }  // Driver code  int main()  {  int blockSize[] = {100, 500, 200, 300, 600};  int processSize[] = {212, 417, 112, 426};  int m = sizeof(blockSize)/sizeof(blockSize[0]);  int n = sizeof(processSize)/sizeof(processSize[0]);  bestFit(blockSize, m, processSize, n);  return 0 ;  }   1. **Worst Fit Algorithm/Pseudo code**    * Read all required input    * FOR i<-0 to all jobs ‘js’      + SET WstInd -1      + FOR j<-0 to all blocks ‘bs’        - IF block[j]>=jobs[i]          * IF Block is free and WstInd==-1 THEN SET WstIndj          * ELSEIF Block is free and block[WstInd]<block[j] THEN SET WstIndj          * ELSE continue with next block   Continue and search next free block   * + - * + IF WstInd!=-1 THEN allocate jth block to ith job   + Display all job with allocated blocks and fragmentation   #include<bits/stdc++.h>  using namespace std;    // Function to allocate memory to blocks as per worst fit  // algorithm  void worstFit(int blockSize[], int m, int processSize[],                                                   int n)  {      // Stores block id of the block allocated to a      // process      int allocation[n];        // Initially no block is assigned to any process      memset(allocation, -1, sizeof(allocation));        // pick each process and find suitable blocks      // according to its size ad assign to it      for (int i=0; i<n; i++)      {          // Find the best fit block for current process          int wstIdx = -1;          for (int j=0; j<m; j++)          {              if (blockSize[j] >= processSize[i])              {                  if (wstIdx == -1)                      wstIdx = j;                  else if (blockSize[wstIdx] < blockSize[j])                      wstIdx = j;              }          }            // If we could find a block for current process          if (wstIdx != -1)          {              // allocate block j to p[i] process              allocation[i] = wstIdx;                // Reduce available memory in this block.              blockSize[wstIdx] -= processSize[i];          }      }        cout << "\nProcess No.\tProcess Size\tBlock no.\n";      for (int i = 0; i < n; i++)      {          cout << "   " << i+1 << "\t\t" << processSize[i] << "\t\t";          if (allocation[i] != -1)              cout << allocation[i] + 1;          else              cout << "Not Allocated";          cout << endl;      }  }    // Driver code  int main()  {      int blockSize[] = {100, 500, 200, 300, 600};      int processSize[] = {212, 417, 112, 426};      int m = sizeof(blockSize)/sizeof(blockSize[0]);      int n = sizeof(processSize)/sizeof(processSize[0]);        worstFit(blockSize, m, processSize, n);        return 0 ;  } |

|  |
| --- |
| **Assignment No.: 08** |
| **Problem Statement:**  Write a Java Program (using OOP features) to implement paging simulation using   1. FIFO 2. Least Recently Used (LRU) 3. Optimal algorithm |
| **Objectives:**   1. To study page replacement policies to understand memory management. 2. To understand efficient frame management using replacement policies. |
| **Theory:**  **CONCEPT OF PAGE REPLACEMENT:**   1. Page Fault: Absence of page when referenced in main memory during paging leads to a page fault. 2. Page Replacement: Replacement of already existing page from main memory by the required new page is called as page replacement. And the techniques used for it are called as page replacement algorithms.   **NEED OF PAGE REPLACEMENT:**  Page replacement is used primarily for the virtual memory management because in virtual memory paging system principal issue is replacement i.e. which page is to be removed so as to bring in the new page, thus the use of the page replacement algorithms. Demand paging is the technique used to increase system throughput. To implement demand paging page replacement is primary requirement. If a system has better page replacement technique it improves demand paging which in turn drastically yields system performance gains.  **PAGE REPLACEMENT POLICIES:**   * 1. Determine which page to be removed from main memory.   2. Find a free frame.      1. If a frame is found use it      2. if no free frame found, use page replacement algorithm to select a victim frame.      3. Write the victim page to the disk.   3. Read the desired page into the new free frame, change the page and frame tables.   4. Restart the user process. |

|  |
| --- |
| **PAGE REPLACEMENT ALGORITHMS:**   1. **FIFO**   This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.   1. **OPTIMAL PAGE REPLACEMENT ALGORITHM:** Replace the page that will not be used for longest period of time as compared to the other pages in main memory. An optimal page replacement algorithm has lowest page fault rate of all algorithm. It is called as OPT or MIN.   **ADVANTAGE:**   * 1. This algorithm guarantees the lowest possible page-fault rate for a fixed no. of frames.   **DISADVANTAGE:**  1) The optimal page replacement algorithm is very difficult to implement, as it requires the knowledge of reference strings i.e. strings of memory references.   1. **LEAST RECENTLY USED (LRU):** LRU algorithm uses the time of the page’s last usage. It uses the recent past as an approximation of the near future, then we can replace the page that has not been used for the longest period of the time i.e. the page having larger idle time is replaced.   **ADVANTAGE:**   * 1. The LRU policy is often used for page replacement and is considered to be good.   **DISADVANTAGES:**   1. It is very difficult to implement. 2. Requires substantial hardware assistance. 3. The problematic determination of the order for the frames defined by the time of last usage |
| **Algorithm/Flowchart:**  **1. FIFO :**   1. Start the process 2. Read number of pages n |

|  |
| --- |
| 1. Read number of pages no 2. Read page numbers into an array a[i] 3. Initialize avail[i]=0 .to check page hit 4. Replace the page with circular queue, while re-placing check page availability in the frame Place avail[i]=1 if page is placed in the frame Count page faults 5. Print the results. 6. Stop the process.   #include<bits/stdc++.h>  using namespace std;    // print the elements of queue  void print\_queue(queue<int> q)  {      while (!q.empty())      {          cout << q.front() << " ";          q.pop();      }      cout << endl;  }    // Driver code  int main()  {      queue<int> q ;        // Adds elements {0, 1, 2, 3, 4} to queue      for (int i = 0; i < 5; i++)          q.push(i);        // Display contents of the queue.      cout << "Elements of queue-";        print\_queue(q);        // To remove the head of queue.      // In this the oldest element '0' will be removed      int removedele = q.front();      q.pop();      cout << "removed element-" << removedele << endl;        print\_queue(q);        // To view the head of queue      int head = q.front();      cout << "head of queue-" << head << endl;        // Rest all methods of collection interface,      // Like size and contains can be used with this      // implementation.      int size = q.size();      cout << "Size of queue-" << size;        return 0;  }  2. **LEAST RECENTLY USED**   1. Start the process 2. Declare the size 3. Get the number of pages to be inserted 4. Get the value 5. Declare counter and stack 6. Select the least recently used page by counter value 7. Stack them according the selection. 8. Display the values 9. Stop the process   #include<bits/stdc++.h>  using namespace std;    // Function to find page faults using indexes  int pageFaults(int pages[], int n, int capacity)  {      // To represent set of current pages. We use      // an unordered\_set so that we quickly check      // if a page is present in set or not      unordered\_set<int> s;        // To store least recently used indexes      // of pages.      unordered\_map<int, int> indexes;        // Start from initial page      int page\_faults = 0;      for (int i=0; i<n; i++)      {          // Check if the set can hold more pages          if (s.size() < capacity)          {              // Insert it into set if not present              // already which represents page fault              if (s.find(pages[i])==s.end())              {                  s.insert(pages[i]);                    // increment page fault                  page\_faults++;              }                // Store the recently used index of              // each page              indexes[pages[i]] = i;          }            // If the set is full then need to perform lru          // i.e. remove the least recently used page          // and insert the current page          else          {              // Check if current page is not already              // present in the set              if (s.find(pages[i]) == s.end())              {                  // Find the least recently used pages                  // that is present in the set                  int lru = INT\_MAX, val;                  for (auto it=s.begin(); it!=s.end(); it++)                  {                      if (indexes[\*it] < lru)                      {                          lru = indexes[\*it];                          val = \*it;                      }                  }                    // Remove the indexes page                  s.erase(val);                    // insert the current page                  s.insert(pages[i]);                    // Increment page faults                  page\_faults++;              }                // Update the current page index              indexes[pages[i]] = i;          }      }        return page\_faults;  }    // Driver code  int main()  {      int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};      int n = sizeof(pages)/sizeof(pages[0]);      int capacity = 4;      cout << pageFaults(pages, n, capacity);      return 0;  }  3. **OPTIMAL ALGORTHIM:**   1. Start Program 2. Read Number Of Pages And Frames 3.Read Each Page Value 3. Search For Page In The Frames 4. If Not Available Allocate Free Frame 5. If No Frames Is Free Replace The Page With The Page That Is Least Used 7.Print Page Number Of Page Faults   8.Stop process.  #include <bits/stdc++.h>  using namespace std;    // Function to check whether a page exists  // in a frame or not  bool search(int key, vector<int>& fr)  {      for (int i = 0; i < fr.size(); i++)          if (fr[i] == key)              return true;      return false;  }    // Function to find the frame that will not be used  // recently in future after given index in pg[0..pn-1]  int predict(int pg[], vector<int>& fr, int pn, int index)  {      // Store the index of pages which are going      // to be used recently in future      int res = -1, farthest = index;      for (int i = 0; i < fr.size(); i++) {          int j;          for (j = index; j < pn; j++) {              if (fr[i] == pg[j]) {                  if (j > farthest) {                      farthest = j;                      res = i;                  }                  break;              }          }            // If a page is never referenced in future,          // return it.          if (j == pn)              return i;      }        // If all of the frames were not in future,      // return any of them, we return 0. Otherwise      // we return res.      return (res == -1) ? 0 : res;  }    void optimalPage(int pg[], int pn, int fn)  {      // Create an array for given number of      // frames and initialize it as empty.      vector<int> fr;        // Traverse through page reference array      // and check for miss and hit.      int hit = 0;      for (int i = 0; i < pn; i++) {            // Page found in a frame : HIT          if (search(pg[i], fr)) {              hit++;              continue;          }            // Page not found in a frame : MISS            // If there is space available in frames.          if (fr.size() < fn)              fr.push\_back(pg[i]);            // Find the page to be replaced.          else {              int j = predict(pg, fr, pn, i + 1);              fr[j] = pg[i];          }      }      cout << "No. of hits = " << hit << endl;      cout << "No. of misses = " << pn - hit << endl;  }    // Driver Function  int main()  {      int pg[] = { 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 };      int pn = sizeof(pg) / sizeof(pg[0]);      int fn = 4;      optimalPage(pg, pn, fn);      return 0;  } |